Lecture Notes on Cryptography

Shafi Goldwasser[1] Mihir Bellare[2]

July 10, 1996

[1] MIT Laboratory of Computer Science, 545 Technology Square, Cambridge, MA 02139, USA. E-mail: shafi@theory.lcs.mit.edu ; Web page: http://theory.lcs.mit.edu/ shafi

[2] Department of Computer Science and Engineering, Mail Code 0114, University of California at San Diego, 9500 Gilman Drive, La Jolla, CA 92093, USA. E-mail: mihir@cs.ucsd.edu ; Web page: http://www-cse.ucsd.edu/users/mihir

Table of Contents

1 Introduction to Modern Cryptography **12**

1.1 Encryption: Historical Glance . . . 12
1.2 Modern Encryption: A Computational Complexity Based Theory . . . 13
1.3 A Short List of Candidate One Way Functions . . . 14
1.4 Security Definitions . . . 15
1.5 The Model of Adversary . . . 16
1.6 Road map to Encryption . . . 16

2 One-way and trapdoor functions **17**

2.1 One-Way Functions: Motivation . . . 17
2.2 One-Way Functions: Definitions . . . 18
2.2.1 (Strong) One Way Functions . . . 18
2.2.2 Weak One-Way Functions . . . 19
2.2.3 Non-Uniform One-Way Functions . . . 20
2.2.4 Collections Of One Way Functions . . . 21
2.2.5 Trapdoor Functions and Collections . . . 22
2.3 In Search of Examples . . . 22
2.3.1 The Discrete Logarithm Function . . . 24
2.3.2 The RSA function . . . 26
2.3.3 Connection Between The Factorization Problem And Inverting RSA . . . 29
2.3.4 The Squaring Trapdoor Function Candidate by Rabin . . . 29
2.3.5 A Squaring Permutation as Hard to Invert as Factoring . . . 32
2.4 Hard-core Predicate of a One Way Function . . . 33
2.4.1 Hard Core Predicates for General One-Way Functions . . . 34
2.4.2 Bit Security Of The Discrete Logarithm Function . . . 35
2.4.3 Bit Security of RSA and SQUARING functions . . . 36
2.5 One-Way and Trapdoor Predicates . . . 37
2.5.1 Examples of Sets of Trapdoor Predicates . . . 38

3 Pseudo-random bit generators **39**

3.0.2 Generating Truly Random bit Sequences . . . 39

3.0.3 Generating Pseudo-Random Bit or Number Sequences . . . 40
3.0.4 Provably Secure Pseudo-Random Generators: Brief overview . . . 41
3.1 Definitions . . . 41
3.2 The Existence Of A Pseudo-Random Generator . . . 42
3.3 Next Bit Tests . . . 45
3.4 Examples of Pseudo-Random Generators . . . 46
3.4.1 Blum/Blum/Shub Pseudo-Random Generator . . . 46

4 Pseudo-random function families 48
4.1 Random functions . . . 48
4.2 Function Families . . . 49
4.2.1 Finite families . . . 49
4.2.2 Infinite families . . . 50
4.3 Motivation for PRFs . . . 50
4.4 Definition of Pseudorandom function families . . . 51
4.4.1 Distinguishers . . . 51
4.4.2 Finite PRFs . . . 52
4.4.3 Infinite PRFs . . . 53
4.5 Usage of PRFs: Examples and intuition . . . 53
4.5.1 Some simple examples . . . 53
4.5.2 Substituting random functions by PRFs . . . 54
4.6 Constructions of PRF families . . . 54
4.6.1 PRFs from PRBGs: trees and synthesizers . . . 55
4.6.2 PRFs from one-way functions . . . 55
4.6.3 Block ciphers as finite PRFs . . . 55
4.6.4 Extending the domain size . . . 56
4.7 Some applications of PRFs . . . 56
4.7.1 Cryptographically Strong Hashing . . . 56
4.7.2 Prediction . . . 57
4.7.3 Learning . . . 57
4.7.4 Identify Friend or Foe . . . 57
4.7.5 Private-Key Encryption . . . 57
4.8 History and discussion . . . 57

5 Private key encryption 59
5.1 Private key encryption schemes . . . 59
5.2 Some issues in security . . . 60
5.3 Example schemes and their analysis . . . 60
5.3.1 Primitives for encryption . . . 60
5.3.2 Encrypting a single block message . . . 62
5.3.3 Modes of DES: ECB, CBC . . . 62
5.3.4 XOR encryption . . . 63
5.4 Definition of security . . . 63
5.5 Methods for secure encryption . . . 65
5.5.1 CBC and XOR schemes . . . 65
5.5.2 Encryption with pseudorandom functions . . . 65
5.5.3 Encryption with pseudorandom bit generators . . . 66

5.5.4 Encryption with one-way functions . . . 66

6 Public key encryption **67**
6.1 Definition of Public-Key Encryption . . . 67
6.2 Simple Examples of PKC: The Trapdoor Function Model . . . 69
6.2.1 Problems with the Trapdoor Function Model . . . 69
6.2.2 Problems with Deterministic Encryption in General . . . 69
6.2.3 The RSA Cryptosystem . . . 70
6.2.4 Rabin's Public key Cryptosystem . . . 71
6.2.5 Knapsacks . . . 72
6.3 Defining Security . . . 73
6.3.1 Definition of Security: Polynomial Indistinguishability . . . 73
6.3.2 Another Definition: Semantic Security . . . 74
6.4 Probabilistic Public Key Encryption . . . 74
6.4.1 Encrypting Single Bits: Trapdoor Predicates . . . 75
6.4.2 Encrypting Single Bits: Hard Core Predicates . . . 75
6.4.3 General Probabilistic Encryption . . . 76
6.4.4 Efficient Probabilistic Encryption . . . 78
6.4.5 An implementation of EPE with cost equal to the cost of RSA . . . 78
6.4.6 Practical RSA based encryption: OAEP . . . 80
6.4.7 Enhancements . . . 81
6.5 Exploring Active Adversaries . . . 81

7 Message authentication **83**
7.1 Message authentication schemes . . . 83
7.2 Some issues in security . . . 84
7.3 Example schemes and their analysis . . . 85
7.3.1 Simple examples . . . 85
7.3.2 XOR MACs . . . 86
7.4 Definition of Security . . . 87
7.5 Methods for secure message authentication . . . 88
7.5.1 Pseudorandom functions make good MACs . . . 88
7.5.2 The CBC MAC . . . 88
7.5.3 Security of XOR MACs . . . 89
7.5.4 Hash then tag . . . 92
7.5.5 MACing with hash functions: HMAC . . . 92
7.5.6 Wegman-Carter approach . . . 94
7.5.7 Minimizing assumptions for MACs . . . 95

8 Digital signatures **96**
8.1 The Ingredients of Digital Signatures . . . 96
8.2 Digital Signatures: the Trapdoor Function Model . . . 97
8.3 Defining and Proving Security for Signature Schemes . . . 97
8.3.1 Attacks Against Digital Signatures . . . 98
8.3.2 The RSA Digital Signature Scheme . . . 99
8.3.3 El Gamal's Scheme . . . 99
8.3.4 Rabin's Scheme . . . 100

8.4 Probabilistic Signatures . . . 101
8.4.1 Claw-free Trap-door Permutations . . . 101
8.4.2 Example: Claw-free permutations exists if factoring is hard . . . 102
8.5 The signature Scheme . . . 102
8.5.1 How to sign one bit . . . 102
8.5.2 How to sign a message . . . 103
8.5.3 A secure signature scheme based on claw free permutations . . . 104
8.5.4 A secure signature scheme based on trapdoor permutations . . . 108
8.6 Practical RSA based signatures: PSS . . . 109
8.6.1 Signing with RSA– Current practice . . . 110
8.6.2 Definitions . . . 110
8.6.3 The Full-Domain-Hash Scheme – FDH . . . 112
8.6.4 The Probabilistic Signature Scheme – PSS . . . 113
8.6.5 Signing with Message Recovery – PSS-R . . . 115
8.6.6 How to implement the hash functions h, g . . . 116
8.6.7 Comparison with other schemes . . . 116
8.7 Threshold Signature Schemes . . . 117
8.7.1 Key Generation for a Threshold Scheme . . . 118
8.7.2 The Signature Protocol . . . 118

9 Key distribution **119**
9.1 Diffie Hellman secret key exchange . . . 119
9.1.1 The protocol . . . 119
9.1.2 Security against eavesdropping: The DH problem . . . 119
9.1.3 The DH cryptosystem . . . 120
9.1.4 Bit security of the DH key . . . 120
9.1.5 The lack of authenticity . . . 121
9.2 Session key distribution . . . 121
9.2.1 Trust models and key distribution problems . . . 122
9.2.2 History of session key distribution . . . 123
9.2.3 An informal description of the problem . . . 123
9.2.4 Issues in security . . . 124
9.2.5 Entity authentication versus key distribution . . . 124
9.3 Authenticated key exchanges . . . 125
9.3.1 The symmetric case . . . 125
9.3.2 The asymmetric case . . . 126
9.4 Three party session key distribution . . . 127
9.5 Forward secrecy . . . 127

10 Protocols **129**
10.1 Some two party protocols . . . 129
10.1.1 Oblivious transfer . . . 129
10.1.2 Simultaneous contract signing . . . 130
10.1.3 Bit Commitment . . . 130
10.1.4 Coin flipping in a well . . . 131
10.1.5 Oblivious circuit evaluation . . . 131
10.1.6 Simultaneous Secret Exchange Protocol . . . 131

10.2 Zero-Knowledge Protocols . . . 132
10.2.1 Interactive Proof-Systems(IP) . . . 132
10.2.2 Examples . . . 133
10.2.3 Zero-Knowledge . . . 134
10.2.4 Definitions . . . 134
10.2.5 If there exists one way functions, then NP is in KC[0] . . . 135
10.2.6 Applications to User Identification . . . 136
10.3 Multi Party protocols . . . 136
10.3.1 Secret sharing . . . 136
10.3.2 Verifiable Secret Sharing . . . 137
10.3.3 Anonymous Transactions . . . 137
10.3.4 Multiparty Ping-Pong Protocols . . . 137
10.3.5 Multiparty Protocols When Most Parties are Honest . . . 138
10.4 Electronic Elections . . . 138
10.4.1 The Merritt Election Protocol . . . 139
10.4.2 A fault-tolerant Election Protocol . . . 139
10.4.3 The protocol . . . 140
10.4.4 Uncoercibility . . . 142
10.5 Digital Cash . . . 142
10.5.1 Required properties for Digital Cash . . . 143
10.5.2 A First-Try Protocol . . . 143
10.5.3 Blind signatures . . . 144
10.5.4 RSA blind signatures . . . 144
10.5.5 Fixing the dollar amount . . . 145
10.5.6 On-line digital cash . . . 145
10.5.7 Off-line digital cash . . . 145

A Some complexity theory background **157**
A.1 Complexity Classes and Standard Definitions . . . 157
A.1.1 Complexity Class P . . . 157
A.1.2 Complexity Class NP . . . 157
A.1.3 Complexity Class BPP . . . 158
A.2 Probabilistic Algorithms . . . 158
A.2.1 Notation For Probabilistic Turing Machines . . . 158
A.2.2 Different Types of Probabilistic Algorithms . . . 159
A.2.3 Non-Uniform Polynomial Time . . . 159
A.3 Adversaries . . . 159
A.3.1 Assumptions To Be Made . . . 160
A.4 Some Inequalities From Probability Theory . . . 160

B Some number theory background **161**
B.1 Groups: Basics . . . 161
B.2 Arithmatic of numbers: +, *, GCD . . . 162
B.3 Modular operations and groups . . . 162
B.3.1 Simple operations . . . 162
B.3.2 The main groups: Z_n and Z_n^* . . . 163
B.3.3 Exponentiation . . . 163

B.4 Chinese remainders . . . 164
B.5 Primitive elements and Z_p^* . . . 165
B.5.1 Definitions . . . 165
B.5.2 The group Z_p^* . . . 166
B.5.3 Finding generators . . . 166
B.6 Quadratic residues . . . 167
B.7 Jacobi Symbol . . . 167
B.8 RSA . . . 168
B.9 Primality Testing . . . 169
B.9.1 PRIMES $\in$ NP . . . 169
B.9.2 Pratt's Primality Test . . . 169
B.9.3 Probabilistic Primality Tests . . . 170
B.9.4 Solovay-Strassen Primality Test . . . 170
B.9.5 Miller-Rabin Primality Test . . . 171
B.9.6 Polynomial Time Proofs Of Primality . . . 172
B.9.7 An Algorithm Which Works For Some Primes . . . 172
B.9.8 Goldwasser-Kilian Primality Test . . . 172
B.9.9 Correctness Of The Goldwasser-Kilian Algorithm . . . 173
B.9.10 Expected Running Time Of Goldwasser-Kilian . . . 174
B.9.11 Expected Running Time On Nearly All Primes . . . 174
B.10 Factoring Algorithms . . . 175
B.11 Elliptic Curves . . . 175
B.11.1 Elliptic Curves Over Z_n . . . 177
B.11.2 Factoring Using Elliptic Curves . . . 177
B.11.3 Correctness of Lenstra's Algorithm . . . 178
B.11.4 Running Time Analysis . . . 178

C About PGP **180**
C.1 Authentication . . . 180
C.2 Privacy . . . 180
C.3 Key Size . . . 181
C.4 E-mail compatibility . . . 181
C.5 One-time IDEA keys generation . . . 181
C.6 Public-Key Management . . . 181

D Problems **182**
D.1 Secret Key Encryption . . . 182
D.1.1 DES . . . 182
D.1.2 Error Correction in DES ciphertexts . . . 182
D.1.3 Brute force search in CBC mode . . . 182
D.1.4 E-mail . . . 183
D.2 Number Theory . . . 183
D.2.1 Number Theory Facts . . . 183
D.2.2 Relationship between problems . . . 183
D.2.3 Probabilistic Primality Test . . . 183
D.3 Public Key Encryption . . . 184
D.3.1 Simple RSA question . . . 184

D.3.2 Another simple RSA question . . . 184
D.3.3 Protocol Failure involving RSA . . . 184
D.3.4 RSA for paranoids . . . 184
D.3.5 Hardness of Diffie-Hellman . . . 185
D.3.6 Bit commitment . . . 185
D.3.7 Perfect Forward Secrecy . . . 185
D.3.8 Plaintext-awareness and non-malleability . . . 185
D.3.9 Probabilistic Encryption . . . 186
D.4 Secret Key Authentication . . . 186
D.4.1 CBC MAC with variable length . . . 186
D.4.2 PRFs are good MACs . . . 187
D.4.3 Simultaneous encryption and authentication . . . 187
D.5 Hash Functions . . . 187
D.5.1 Birthday Paradox . . . 187
D.5.2 Hash functions from DES . . . 187
D.5.3 Hash functions from RSA . . . 188
D.6 Pseudo-randomness . . . 188
D.6.1 Extending PRGs . . . 188
D.6.2 From PRG to PRF . . . 188
D.6.3 From PRF to PRG . . . 188
D.7 Digital Signatures . . . 189
D.7.1 Table of Forgery . . . 189
D.7.2 ElGamal . . . 189
D.7.3 Suggested signature scheme . . . 189
D.7.4 Ong-Schnorr-Shamir . . . 189
D.8 Protocols . . . 190
D.8.1 Unconditionally Secure Secret Sharing . . . 190
D.8.2 Secret Sharing with cheaters . . . 190
D.8.3 Zero–Knowledge proof for discrete logarithms . . . 190
D.8.4 Oblivious Transfer . . . 190
D.8.5 Electronic Cash . . . 190
D.8.6 Atomicity of withdrawal protocol . . . 192
D.8.7 Blinding with ElGamal/DSS . . . 192

List of Figures

8.1 PSS: Components of image $y = 0 \,.\, w \,.\, r^* \,.\, g_2(w)$ are darkened. The signature of M is $y^d \bmod N$. 114
8.2 PSS-R: Components of image $y = 0 \,.\, w \,.\, r^* \,.\, M^*$ are darkened. 115

9.1 Protocol AKEP1: Session key distribution in symmetric setting. 125
9.2 Protocol for exchange of symmetric key in asymmetric setting. 126
9.3 Three party session key distribution protocol. 127
9.4 Protocol for exchange of symmetric key with forward secrecy. 128

B.1 Addition on the elliptic curve $y^2 = x^3 - x$. 176

Foreword

This is a set of lecture notes on cryptography compiled for 6.87s, a summer course on cryptography taught at MIT by Shafi Goldwasser and Mihir Bellare in July 1996. Some of the materiel is based on courses we have taught at our respective institutions in the past; some is inspired from survey papers in the field; the rest is newly written.

Cryptography is of course a vast subject. The thread followed by these notes is to develop and explain the notion of provable security and its usage for the design of secure protocols.

The Teaching Assistant for the course was Rosario Gennaro. He contributed Section 8.7, Section 10.4, Section 10.5, and Appendix C to the notes, and also compiled, from various sources, the list of problems in Appendix D.

Some of the material in the pseudorandom number generation chapter (Chapter 3) is from the chapter on Cryptography, by R. Rivest, in the Handbook of Theoretical Computer Science.

All rights reserved.

Shafi Goldwasser and Mihir Bellare — Cambridge, Massachusetts, July 6, 1996.

CHAPTER 1

Introduction to Modern Cryptography

Cryptography is about communication in the presence of an adversary. It encompasses many problems (encryption, authentication, key distribution to name a few). The field of modern cryptography provides a theoretical foundation based on which we may understand what exactly these problems are, how to evaluate protocols that purport to solve them, and how to build protocols in whose security we can have confidence. We introduce the basic issues by discussing the problem of encryption.

1.1 Encryption: Historical Glance

The most ancient and basic problem of cryptography is secure communication over an insecure channel. Party A wants to send to party B a secret message over a communication line which may be tapped by an adversary.

The traditional solution to this problem is called *private key encryption*. In private key encryption A and B hold a meeting before the remote transmission takes place and agree on a pair of encryption and decryption algorithms $\mathcal{E}$ and $\mathcal{D}$, and an additional piece of information S to be kept secret. We shall refer to S as the *common secret key*. The adversary may know the encryption and decryption algorithms $\mathcal{E}$ and $\mathcal{D}$ which are being used, but does not know S.

After the initial meeting when A wants to send B the *cleartext* or *plaintext* message m over the insecure communication line, A *encrypts* m by computing the *ciphertext* $c = \mathcal{E}(S, m)$ and sends c to B. Upon receipt, B *decrypts* c by computing $m = \mathcal{D}(S, c)$. The line-tapper (or adversary), who does not know S, should not be able to compute m from c.

Let us illustrate this general and informal setup with an example familiar to most of us from childhood, the *substitution cipher*. In this method A and B meet and agree on some secret permutation $f: \Sigma \to \Sigma$ (where Σ is the alphabet of the messages to be sent). To encrypt message $m = m_1 \ldots m_n$ where $m_i \in \Sigma$, A computes $\mathcal{E}(f, m) = f(m_1) \ldots f(m_n)$. To decrypt $c = c_1 \ldots c_n$ where $c_i \in \Sigma$, B computes $\mathcal{D}(f, c) = f^{-1}(c_1) \ldots f^{-1}(c_n) = m_1 \ldots m_n = m$. In this example the common secret key is the permutation f. The encryption and decryption algorithms $\mathcal{E}$ and $\mathcal{D}$ are as specified, and are known to the adversary. We note that the substitution cipher is easy to break by an adversary who sees a moderate (as a function of the size of the alphabet Σ) number of ciphertexts.

A rigorous theory of perfect secrecy based on information theory was developed by Shannon [169] in 1943. [1]. In this theory, the adversary is assumed to have unlimited computational resources. Shannon showed that secure (properly defined) encryption system can exist only if the size of the secret information S that A and B agree on prior to remote transmission is as large as the number of secret bits to be ever exchanged remotely using the encryption system.

[1] Shannon's famous work on information theory was an outgrowth of his work on security ([170]).

An example of a private key encryption method which is secure even in presence of a computationally unbounded adversary is the *one time pad*. A and B agree on a secret bit string $pad = b_1 b_2 \ldots b_n$, where $b_i \in_R \{0,1\}$ (i.e pad is chosen in $\{0,1\}^n$ with uniform probability). This is the common secret key. To encrypt a message $m = m_1 m_2 \ldots m_n$ where $m_i \in \{0,1\}$, A computes $\mathcal{E}(pad, m) = m \oplus pad$ (bitwise exclusive or). To decrypt ciphertext $c \in \{0,1\}^n$, B computes $\mathcal{D}(pad, c) = pad \oplus c = pad \oplus (m \oplus pad) = m$. It is easy to verify that $\forall m, c$ the $\Pr_{pad}[\mathcal{E}(pad, m) = c] = \frac{1}{2^n}$. From this, it can be argued that seeing c gives "no information" about what has been sent. (In the sense that the adversary's a posteriori probability of predicting m given c is no better than her a priori probability of predicting m without being given c.)

Now, suppose A wants to send B an additional message m'. If A were to simply send $c = \mathcal{E}(pad, m')$, then the sum of the lengths of messages m and m' will exceed the length of the secret key pad, and thus by Shannon's theory the system cannot be secure. Indeed, the adversary can compute $\mathcal{E}(pad, m) \oplus \mathcal{E}(pad, m') = m \oplus m'$ which gives information about m and m' (e.g. can tell which bits of m and m' are equal and which are different). To fix this, the length of the pad agreed upon a-priori should be the sum total of the length of all messages ever to be exchanged over the insecure communication line.

1.2 Modern Encryption: A Computational Complexity Based Theory

Modern cryptography abandons the assumption that the Adversary has available infinite computing resources, and assumes instead that the adversary's computation is resource bounded in some reasonable way. In particular, in these notes we will assume that the adversary is a probabilistic algorithm who runs in polynomial time. Similarly, the encryption and decryption algorithms designed are probabilistic and run in polynomial time.

The running time of the encryption, decryption, and the adversary algorithms are all measured as a function of a *security parameter* k which is a parameter which is fixed at the time the cryptosystem is setup. Thus, when we say that the adversary algorithm runs in polynomial time, we mean time bounded by some polynomial function in k.

Accordingly, in modern cryptography, we speak of the *infeasibility* of breaking the encryption system and computing information about exchanged messages where as historically one spoke of the *impossibility* of breaking the encryption system and finding information about exchanged messages. We note that the encryption systems which we will describe and claim "secure" with respect to the new adversary are not "secure" with respect to a computationally unbounded adversary in the way that the one-time pad system was secure against an unbounded adversary. But, on the other hand, it is no longer necessarily true that the size of the secret key that A and B meet and agree on before remote transmission must be as long as the total number of secret bits ever to be exchanged securely remotely. In fact, at the time of the initial meeting, A and B do not need to know in advance how many secret bits they intend to send in the future.

We will show how to construct such encryption systems, for which the number of messages to be exchanged securely can be a polynomial in the length of the common secret key. How we construct them brings us to anther fundamental issue, namely that of cryptographic, or complexity, assumptions.

As modern cryptography is based on a gap between efficient algorithms for encryption for the legitimate users versus the computational infeasibility of decryption for the adversary, it requires that one have available *primitives* with certain special kinds of computational hardness properties. Of these, perhaps the most basic is a *one-way function*. Informally, a function is one-way if it is easy to compute but hard to invert. Other primitives include *pseudo-random number generators*, and *pseudorandom function families*, which we will define and discuss later. From such primitives, it is possible to build secure encryption schemes.

Thus, a central issue is where these primitives come from. Although one-way functions are widely believed to exist, and there are several conjectured candidate one-way functions which are widely used, we currently do not know how to mathematically prove that they actually exist. We shall thus design cryptographic schemes assuming we are given a one-way function. We will use the conjectured candidate one-way functions for our working examples, throughout our notes. We will be explicit about what exactly can and cannot be proved and is thus assumed, attempting to keep the latter to a bare minimum.

We shall elaborate on various constructions of private-key encryption algorithms later in the course.

The development of *public key cryptography* in the seventies enables one to drop the requirement that A and B must share a key in order to encrypt. The receiver B can publish authenticated[2] information (called the *public-key*) for anyone including the adversary, the sender A, and any other sender to read at their convenience (e.g in a phone book). We will show encryption algorithms in which whoever can read the public key can send encrypted messages to B without ever having met B in person. The encryption system is no longer intended to be used by a pair of prespecified users, but by many senders wishing to send secret messages to a single recipient. The receiver keeps secret (to himself alone!) information (called the receiver's *private key*) about the public-key, which enables him to decrypt the cyphertexts he receives. We call such an encryption method *public key encryption.*

We will show that secure public key encryption is possible given a *trapdoor function.* Informally, a trapdoor function is a one-way function for which there exists some *trapdoor* information known to the receiver alone, with which the receiver can invert the function. The idea of public-key cryptosystems and trapdoor functions was introduced in the seminal work of Diffie and Hellman in 1976 [61, 62]. Soon after the first implementations of their idea were proposed in [151], [146], [125].

A simple construction of public key encryption from trapdoor functions goes as follows. Recipient B can choose at random a trapdoor function f and its associated trapdoor information t, and set its public key to be a description of f and its private key to be t. If A wants to send message m to B, A computes $\mathcal{E}(f, m) = f(m)$. To decrypt $c = f(m)$, B computes $f^{-1}(c) = f^{-1}(f(m)) = m$. We will show that this construction is not secure enough in general, but construct probabilistic variants of it which are secure.

1.3 A Short List of Candidate One Way Functions

As we said above, the most basic primitive for cryptographic applications is a one-way function which is "easy" to compute but "hard" to invert. (For public key encryption, it must also have a trapdoor.) By "easy", we mean that the function can be computed by a probabilistic polynomial time algorithm, and by "hard" that any probabilistic polynomial time (PPT) algorithm attempting to invert it will succeed with "small" probability (where the probability ranges over the elements in the domain of the function.) Thus, to qualify as a potential candidate for a one-way function, the hardness of inverting the function should not hold only on rare inputs to the function but with high probability over the inputs.

Several candidates which seem to posses the above properties have been proposed.

1. *Factoring.* The function $f : (x, y) \mapsto xy$ is conjectured to be a one way function. The asymptotically proven fastest factoring algorithms to date are variations on Dixon's random squares algorithm [113]. It is a randomized algorithm with running time $L(n)^{\sqrt{2}}$ where $L(n) = e^{\sqrt{\log n \log\log n}}$. The number field sieve by Lenstra, Lenstra, Manasee, and Pollard with modifications by Adlemann and Pomerance is a factoring algorithm proved under a certain set of assumptions to factor integers in expected time

$$e^{((c+o(1))(\log n)^{\frac{1}{3}}(\log\log n)^{\frac{2}{3}})}$$

   [115, 3].

2. *The discrete log problem.* Let p be a prime. The multiplicative group $\mathcal{Z}_p^* = (\{x < p | (x, p) = 1\}, \cdot \bmod p)$ is cyclic, so that $\mathcal{Z}_p^* = \{g^i \bmod p | 1 \leq i \leq p-1\}$ for some generator $g \in \mathcal{Z}_p^*$. The function $f : (p, g, x) \mapsto (g^x \bmod p, p, g)$ where p is a prime and g is a generator for Z_p^* is conjectured to be a one-way function. Computing $f(p, g, x)$ can be done in polynomial time using repeated squaring. However, The fastest known proved solution for its inverse, called the discrete log problem is the index-calculus algorithm, with expected running time $L(p)^{\sqrt{2}}$ (see [113]). An interesting problem is to find an algorithm which will generate a prime p and a generator g for Z_p^*. It is not known how to find generators in polynomial time. However, in [8], E. Bach shows how to generate random factored integers (in a given range $\frac{N}{2} \ldots N$). Coupled with a fast primality tester (as found in [113], for example), this can be used to efficiently generate random tuples $(p-1, q_1, \ldots, q_k)$ with p prime. Then picking $g \in \mathcal{Z}_p^*$ at random, it can be checked if $(g, p-1) = 1, \forall q_i, g^{\frac{p-1}{q_i}} \bmod p \neq 1$, and $g^{p-1} \bmod p = 1$, in which case $order(g) = p-1$

[2] Saying that the information is "authenticated" means that the sender is given a guarantee that the information was published by the legal receiver. How this can be done is discussed in a later chapter.

$(order(g) = |\{g^i \bmod p | 1 \leq i \leq p-1\}|)$. It can be shown that the density of $\mathcal{Z}_p^*$ generators is high so that few guesses are required. The problem of efficiently finding a generator for a specific $\mathcal{Z}_p^*$ is an intriguing open research problem.

3. *Subset sum.* Let $a_i \in \{0,1\}^n, \vec{a} = (a_1, \ldots, a_n), s_i \in \{0,1\}, \vec{s} = (s_1, \ldots, s_n)$, and let $f : (\vec{a}, \vec{s}) \mapsto (\vec{a}, \sum_{i=1}^n s_i a_i)$. An inverse of $(\vec{a}, \sum_{i=1}^n s_i a_i)$ under f is any $(\vec{a}, \vec{s_i'})$ so that $\sum_{i=1}^n s_i a_i = \sum_{i=1}^n s_i' a_i$. This function f is a candidate for a one way function. The associated decision problem (given $(\vec{a}, y)$, does there exists $\vec{s}$ so that $\sum_{i=1}^n s_i a_i = y$?) is NP-complete. Of course, the fact that the subset-sum problem is NP-complete cannot serve as evidence to the one-wayness of f_{ss}. On the other hand, the fact that the subset-sum problem is easy for special cases (such as "hidden structure" and low density) can not serve as evidence for the weakness of this proposal. The conjecture that f is one-way is based on the failure of known algorithm to handle random high density instances. Yet, one has to admit that the evidence in favor of this candidate is much weaker than the evidence in favor of the two previous ones.

4. *DES with fixed message.* Fix a 64 bit message M and define the function $f(K) = \text{DES}_K(M)$ which takes a 56 bit key K to a 64 bit output $f(K)$. This appears to be a one-way function. Indeed, this construction can even be proven to be one-way assuming DES is a family of pseudorandom functions, as shown by Luby and Rackoff [120].

5. *RSA.* This is a candidate one-way trapdoor function. Let $N = pq$ be a product of two primes. It is believed that such an N is hard to factor. The function is $f(x) = x^e \bmod N$ where e is relatively prime to $(p-1)(q-1)$. The trapdoor is the primes p, q, knowledge of which allows one to invert f efficiently. The function f seems to be one-way. To date the best attack is to try to factor N, which seems computationally infeasible.

In Chapter 2 we discuss formal definitions of one-way functions and are more precise about the above constructions.

1.4 Security Definitions

So far we have used the terms "secure" and "break the system" quite loosely. What do we really mean? It is clear that a minimal requirement of security would be that: any adversary who can see the ciphertext and knows which encryption and decryption algorithms are being used, can not recover the entire cleartext. But, many more properties may be desirable. To name a few:

1. It should be hard to recover the messages from the ciphertext when the messages are drawn from arbitrary probability distributions defined on the set of all strings (i.e arbitrary *message spaces*). A few examples of message spaces are: the English language, the set $\{0,1\}$). We must assume that the message space is known to the adversary.

2. It should be hard to compute partial information about messages from the ciphertext.

3. It should be hard to detect simple but useful facts about traffic of messages, such as when the same message is sent twice.

4. The above properties should hold with high probability.

In short, it would be desirable for the encryption scheme to be the mathematical analogy of opaque envelopes containing a piece of paper on which the message is written. The envelopes should be such that all legal senders can fill it, but only the legal recipient can open it.

We must answer a few questions:

- How can "opaque envelopes" be captured in a precise mathematical definition? Much of Chapters 5 and 6 is dedicated to discussing the precise definition of security in presence of a computationally bounded adversary.

- Are "opaque envelopes" achievable mathematically? The answer is positive . We will describe the the proposals of private (and public) encryption schemes which we prove secure under various assumptions.

We note that the simple example of a public-key encryptions system based on trapdoor function, described in the previous section, does not satisfy the above properties. We will show later, however, probabilistic variants of the simple system which do satisfy the new security requirements under the assumption that trapdoor functions exist. More specifically, we will show probabilistic variants of RSA which satisfy the new security requirement under, the assumption that the original RSA function is a trapdoor function, and are similar in efficiency to the original RSA public-key encryption proposal.

1.5 The Model of Adversary

The entire discussion so far has essentially assumed that the adversary can listen to cyphertexts being exchanged over the insecure channel, read the public-file (in the case of public-key cryptography), generate encryptions of any message on his own (for the case of public-key encryption), and perform probabilistic polynomial time computation. This is called a *passive adversary*.

One may imagine a more powerful adversary who can intercept messages being transmitted from sender to receiver and either stop their delivery all together or alter them in some way. Even worse, suppose the adversary can request a polynomial number of cyphertexts to be decrypted for him. We can still ask whether there exists encryption schemes (public or secret) which are secure against such more powerful adversaries.

Indeed, such adversaries have been considered and encryption schemes which are secure against them designed. The definition of security against such adversaries is more elaborate than for passive adversaries.

In Chapters 5 and 6 we consider a passive adversary who knows the probability distribution over the message space. We will also discuss more powerful adversaries and appropriate definitions of security.

1.6 Road map to Encryption

To summarize the introduction, our challenge is to design both secure private-key and public-key encryption systems which provably meet our definition of security and in which the operations of encryption and decryption are as fast as possible for the sender and receiver.

Chapters 5 and 6 embark on an in depth investigation of the topic of encryption, consisting of the following parts. For both private-key and public-key encryption, we will:

- Discuss formally how to define security in presence of a bounded adversary.
- Discuss current proposals of encryption systems and evaluate them respect to the security definition chosen.
- Describe how to design encryption systems which we can prove secure under explicit assumptions such as the existence of one-way functions, trapdoor functions, or pseudo random functions.
- Discuss efficiency aspects of encryption proposals, pointing out to possible ways to improve efficiency by performing some computations off-line, in batch mode, or in a incremental fashion.

We will also overview some advanced topics connected to encryption such chosen-ciphertext security, non-malleability, key-escrow proposals, and the idea of shared decryption among many users of a network.

CHAPTER 2

One-way and trapdoor functions

One Way functions, namely functions that are "easy" to compute and "hard" to invert, are an extremely important cryptographic primitive. Probably the best known and simplest use of one-way functions, is for passwords. Namely, in a time-shared computer system, instead of storing a table of login passwords, one can store, for each password w, the value $f(w)$. Passwords can easily be checked for correctness at login, but even the system administrator can not deduce any user's password by examining the stored table.

In Section 1.3 we had provided a short list of some candidate one-way functions. We now develop a theoretical treatment of the subject of one-way and trapdoor functions, and carefully examine the candidate one-way functions proposed in the literature. We will occasionaly refer to facts about number theory discussed in Chapter B.

We begin by explaining why one-way functions are of fundamental importance to cryptography.

2.1 One-Way Functions: Motivation

In this section, we provide motivation to the definition of one-way functions. We argue that the existence of one-way functions is a necessary condition to the existence of most known cryptographic primitives (including secure encryption and digital signatures). As the current state of knowledge in complexity theory does not allow to prove the existence of one-way function, even using more traditional assumptions as $\mathcal{P} \neq \mathcal{NP}$, we will have to assume the existence of one-way functions. We will later try to provide evidence to the plausibility of this assumption.

As stated in the introduction chapter, modern cryptography is based on a gap between efficient algorithms guaranteed for the legitimate user versus the unfeasibility of retrieving protected information for an adversary. To make the following discussion more clear, let us concentrate on the cryptographic task of secure data communication, namely encryption schemes.

In secure encryption schemes, the legitimate user is able to decipher the messages (using some private information available to him), yet for an adversary (not having this private information) the task of decrypting the ciphertext (i.e., "breaking" the encryption) should be infeasible. Clearly, the breaking task can be performed by a non-deterministic polynomial-time machine. Yet, the security requirement states that breaking should not be feasible, namely could not be performed by a probabilistic polynomial-time machine. Hence, the existence of secure encryption schemes implies that there are tasks performed by non-deterministic polynomial-time machines yet cannot be performed by deterministic (or even randomized) polynomial-time machines. In other words, a necessary condition for the existence of secure encryption schemes is that $\mathcal{NP}$ is not contained in $\mathcal{BPP}$ (and hence that $\mathcal{P} \neq \mathcal{NP}$).

However, the above mentioned necessary condition (e.g., $\mathcal{P} \neq \mathcal{NP}$) is not a sufficient one. $\mathcal{P} \neq \mathcal{NP}$ only implies that the encryption scheme is hard to break in the worst case. It does not rule-out the possibility

that the encryption scheme is easy to break in almost all cases. In fact, one can easily construct "encryption schemes" for which the breaking problem is NP-complete and yet there exist an efficient breaking algorithm that succeeds on 99% of the cases. Hence, worst-case hardness is a poor measure of security. Security requires hardness on most cases or at least average-case hardness. Hence, a necessary condition for the existence of secure encryption schemes is the existence of languages in $\mathcal{NP}$ which are hard on the average. Furthermore, $\mathcal{P} \neq \mathcal{NP}$ is not known to imply the existence of languages in $\mathcal{NP}$ which are hard on the average.

The mere existence of problems (in NP) which are hard on the average does not suffice. In order to be able to use such problems we must be able to generate such hard instances together with auxiliary information which enable to solve these instances fast. Otherwise, the hard instances will be hard also for the legitimate users and they gain no computational advantage over the adversary. Hence, the existence of secure encryption schemes implies the existence of an efficient way (i.e. probabilistic polynomial-time algorithm) of generating instances with corresponding auxiliary input so that

(1) it is easy to solve these instances given the auxiliary input; and

(2) it is hard on the average to solve these instances (when not given the auxiliary input).

We avoid formulating the above "definition". We only remark that the coin tosses used in order to generate the instance provide sufficient information to allow to efficiently solve the instance (as in item (1) above). Hence, without loss of generality one can replace condition (2) by requiring that these coin tosses are hard to retrieve from the instance. The last simplification of the above conditions essentially leads to the definition of a one-way function.

2.2 One-Way Functions: Definitions

In this section, we present several definitions of one-way functions. The first version, hereafter referred to as strong one-way function (or just one-way function), is the most convenient one. We also present weak one-way functions which may be easier to find and yet can be used to construct strong one way functios, and non-uniform one-way functions.

2.2.1 (Strong) One Way Functions

The most basic primitive for cryptographic applications is a one-way function. Informally, this is a function which is "easy" to compute but "hard" to invert. Namely, any probabilistic polynomial time (PPT) algorithm attempting to invert the one-way function on a element in its range, will succeed with no more than "negligible" probability, where the probability is taken over the elements in the domain of the function and the coin tosses of the PPT attempting the inversion.

This informal definition introduces a couple of measures that are prevalent in complexity theoretic cryptography. An easy computation is one which can be carried out by a PPT algorithm; and a function ν: $\mathbb{N} \rightarrow \mathbf{R}$ is negligible if it vanishes faster than the inverse of any polynomial. More formally,

Definition 2.2.1 ν is negligible if for every constant $c \geq 0$ there exists an integer k_c such that $\nu(k) < k^{-c}$ for all $k \geq k_c$.

Another way to think of it is $\nu(k) = k^{-\omega(1)}$.

A few words, concerning the notion of negligible probability, are in place. The above definition and discussion considers the success probability of an algorithm to be *negligible* if as a function of the input length the success probability is bounded by any polynomial fraction. It follows that repeating the algorithm polynomially (in the input length) many times yields a new algorithm that also has a negligible success probability. In other words, events which occur with negligible (in n) probability remain negligible even if the experiment is repeated for polynomially (in k) many times. Hence, defining negligible success as "occurring with probability smaller than any polynomial fraction" is naturally coupled with defining feasible as "computed within polynomial time". A "strong negation" of the notion of a negligible fraction/probability is the notion of a non-negligible fraction/probability. we say that a function ν is *non-negligible* if there exists a polynomial p such that for all sufficiently large k's it holds that $\nu(k) > \frac{1}{p(k)}$. Note that functions may be neither negligible nor non-negligible.

Definition 2.2.2 A function $f\colon \{0,1\}^* \to \{0,1\}^*$ is *one-way* if:
(1) there exists a PPT that on input x output $f(x)$;
(2) For every PPT algorithm A there is a negligible function ν_A such that for sufficiently large k,

$$\Pr\left[f(z) = y \ : \ x \stackrel{R}{\leftarrow} \{0,1\}^k \ ; \ y \leftarrow f(x) \ ; \ z \leftarrow A(1^k, y)\right] \ \leq \ \nu_A(k)$$

Remark 2.2.3 The guarantee is probabilistic. The adversary is not unable to invert the function, but has a low probability of doing so where the probability distribution is taken over the input x to the one-way function where x if of length k, and the possible coin tosses of the adversary. Namely, x is chosen at random and y is set to $f(x)$.

Remark 2.2.4 The advsersary is not asked to find x; that would be pretty near impossible. It is asked to find some inverse of y. Naturally, if the function is 1-1 then the only inverse is x.

Remark 2.2.5 Note that the adversary algorithm takes as input $f(x)$ and the security parameter 1^k (expressed in unary notatin) which corresponds to the binary length of x. This represents the fact the adversary can work in time polynomial in $|x|$, even if $f(x)$ happends to be much shorter. This rules out the possibility that a function is considered one-way merely because the inverting algorithm does not have enough time to print the output. Consider for example the function defined as $f(x) = y$ where y is the $\log k$ least significant bits of x where $|x| = k$. Since the $|f(x)| = \log |x|$ no algorithm can invert f in time polynomial in $|f(x)|$, yet there exists an obvious algorithm which finds an inverse of $f(x)$ in time polynomial in $|x|$. Note that in the special case of length preserving functions f (i.e., $|f(x)| = |x|$ for all x's), the auxiliary input is redundant.

Remark 2.2.6 By this definition it trivially follows that the size of the output of f is bounded by a polynomial in k, since $f(x)$ is a poly-time computable.

Remark 2.2.7 The definition is typical to definitions from computational complexity theory works with asymptotic complexity—what happens as the size of the problem becomes large. Security is only asked to hold for large enough input lengths, namely as k goes to infinity. Per this definition, it may be entirely feasible to invert f on, say, 512 bit inputs. Thus such definitions are less directly relevant to practice, but useful for studying things on a basic level. To apply this definition to practice in cryptography we must typically envisage not a single one-way function but a family of them, parameterized by a *security parameter* k. That is, for each value of the security parameter k there is be a specific function $f : \{0,1\}^k \to \{0,1\}^*$. Or, there may be a family of functions (or cryptosystems) for each value of k. We shall define such familes in subsequent section.

The next two sections discuss variants of the strong one-way function definition. The first time reader is encouraged to directly go to Section 2.2.4.

2.2.2 Weak One-Way Functions

One way functions come in two flavors: strong and weak. The definition we gave above, refers to a strong way function. We could weaken it by replacing the second requirement in the definition of the function by a weaker requirement as follows.

Definition 2.2.8 A function $f\colon \{0,1\}^* \to \{0,1\}^*$ is *weak one-way* if:
(1) there exists a PPT that on input x output $f(x)$;
(2) There is a polynomial functions Q such that for every PPT algorithm A, and for sufficiently large k,

$$\Pr\left[f(z) \neq y \ : \ x \stackrel{R}{\leftarrow} \{0,1\}^k \ ; \ y \leftarrow f(x) \ ; \ z \leftarrow A(1^k, y)\right] \geq \frac{1}{Q(k)}$$

The difference between the two definitions is that whereas we only require some non-negligible fraction of the inputs on which it is hard to invert a weak one-way function, a strong one-way function must be hard to invert on all but a negligible fraction of the inputs. Clearly, the latter is preferable, but what if only weak one-way functions exist ? Our first theorem is that the existence of a weak one way function implies the existence of a strong one way function. Moreover, we show how to construct a strong one-way function from a weak one. This is important in practice as illustarted by the following example.

Example 2.2.9 Consider for example the function $f : \mathbf{Z} \times \mathbf{Z} \mapsto \mathbf{Z}$ where $f(x, y) = x \cdot y$. This function can be easily inverted on at least half of its outputs (namely, on the even integers) and thus is not a strong one way function. Still, we said in the first lecture that f is hard to invert when x and y are primes of roughly the same length which is the case for a polynomial fraction of the k-bit composite integers. This motivated the definition of a weak one way function. Since the probability that an k-bit integer x is prime is approximately $1/k$, we get the probability that both x and y such that $|x| = |y| = k$ are prime is approximately $1/k^2$. Thus, for all k, about $1 - \frac{1}{k^2}$ of the inputs to f of length $2k$ are prime pairs of equal length. It is believed that no adversary can invert f when x and y are primes of the same length with non-negligible success probability, and under this belief, f is a weak one way function (as condition 2 in the above definition is satisfied for $Q(k) = O(k^2)$).

Clearly, it would be more desirable to have strong one way functions. But, what if only weak one way functions exist? Theorem 2.2.10 tells us that the existence of a weak one way function implies the existence of a strong one way function.

Theorem 2.2.10 *Weak one way functions exist if and only if strong one way functions exist.*

Proof Sketch: By definition, a strong one way function is a weak one way function. Now assume that f is a weak one way function such that Q is the polynomial in condition 2 in the definition of a weak one way function. Define the function

$$f_1(x_1 \ldots x_N) = f(x_1) \ldots f(x_N)$$

where $N = 2kQ(k)$ and each x_i is of length k.

We claim that f_1 is a strong one way function. Since f_1 is a concatenation of N copies of the function f, to correctly invert f_1, we need to invert $f(x_i)$ correctly for each i. We know that every adversary has a probability of at least $\frac{1}{Q(k)}$ to fail to invert $f(x)$ (where the probability is taken over $x \in \{0,1\}^k$ and the coin tosses of the adversary), and so intuitively, to invert f_1 we need to invert $O(kQ(k))$ instances of f. The probability that the adversary will fail for at least one of these instances is extremely high.

The formal proof (which is omitted here and will be given in appendix) will take the form of a reduction; that is, we will assume for contradiction that f_1 is not a strong one way function and that there exists some adversary A_1 that violates condition 2 in the definition of a strong one way function. We will then show that A_1 can be used as a subroutine by a new adversary A that will be able to invert the original function f with probability better than $1 - \frac{1}{Q(|x|)}$ (where the probability is taken over the inputs $x \in \{0,1\}^k$ and the coin tosses of A). But this will mean that f is not a weak one way function and we have derived a contradiction. ∎

This proof technique is quite typical of proofs presented in this course. Whenever such a proof is presented it is important to examine the cost of the reduction. For example, the construction we have just outlined is not length preserving, but expands the size of the input to the function quadratically.

2.2.3 Non-Uniform One-Way Functions

In the above two definitions of one-way functions the inverting algorithm is probabilistic polynomial-time. Stronger versions of both definitions require that the functions cannot be inverted even by non-uniform families of polynomial size algorithm We stress that the "easy to compute" condition is still stated in terms of uniform algorithms. For example, following is a non-uniform version of the definition of (strong) one-way functions.

Definition 2.2.11 A function f is called *non-uniformly strong one-way* if the following two conditions hold

(1) *easy to compute:* as before There exists a PPT algorithm to compute for f.

(2) *hard to invert:* For every (even non-uniform) family of polynomial-size algorithms $A = \{M_k\}_{k \in \mathbb{N}}$, there exists a negligble ν_A such that for all sufficiently large k's

$$\Pr\left[f(z) \neq y \ : \ x \stackrel{R}{\leftarrow} \{0,1\}^k \ ; \ y \leftarrow f(x) \ ; \ z \leftarrow M_k(y)\right] \ \leq \ \nu_A(k)$$

Note that it is redundent to give 1^k as an auxiliary input to M_k.

It can be shown that if f is non-uniformly one-way then it is (strongly) one-way (i.e., in the uniform sense). The proof follows by converting any (uniform) probabilistic polynomial-time inverting algorithm into a non-uniform family of polynomial-size algorithm, without decreasing the success probability. Details follow. Let A' be a probabilistic polynomial-time (inverting) algorithm. Let r_k denote a sequence of coin tosses for A' maximizing the success probability of A'. The desired algorithm M_k incorporates the code of algorithm A' and the sequence r_k (which is of length polynomial in k).

It is possible, yet not very plausible, that strongly one-way functions exist and but there are no non-uniformly one-way functions.

2.2.4 Collections Of One Way Functions

Instead of talking about a single function $f : \{0,1\}^* \rightarrow \{0,1\}^*$, it is often convenient to talk about collections of functions, each defined over some finite domain and finite ranges. We remark, however, that the single function format makes it easier to prove properties about one way functions.

Definition 2.2.12 Let I be a set of indices and for $i \in I$ let D_i and R_i be finite. A collection of strong one way functions is a set $F = \{f_i : D_i \rightarrow R_i\}_{i \in I}$ satisfying the following conditions.

(1) There exists a PPT S_1 which on input 1^k outputs an $i \in \{0,1\}^k \cap I$

(2) There exists a PPT S_2 which on input $i \in I$ outputs $x \in D_i$

(3) There exists a PPT A_1 such that for $i \in I$ and $x \in D_i$, $A_1(i,x) = f_i(x)$.

(4) For every PPT A there exists a negligible ν_A such that $\forall$ k large enough

$$\Pr\left[f_i(z) \neq y \ : \ i \stackrel{R}{\leftarrow} I \ ; \ x \stackrel{R}{\leftarrow} D_i \ ; \ y \leftarrow f_i(x) \ ; \ z \leftarrow A(i,y)\right] \ \leq \ \nu_A(k)$$

(here the probability is taken over choices of i and x, and the coin tosses of A).

In general, we can show that the existence of a single one way function is equivalent to the existence of a collection of one way functions. We prove this next.

Theorem 2.2.13 *A collection of one way functions exists if and only if one way functions exist.*

Proof: Suppose that f is a one way function.

Set $F = \{f_i : D_i \rightarrow R_i\}_{i \in I}$ where $I = \{0,1\}^*$ and for $i \in I$, take $D_i = R_i = \{0,1\}^{|i|}$ and $f_i(x) = f(x)$. Furthermore, S_1 uniformly chooses on input 1^k, $i \in \{0,1\}^k$, S_2 uniformly chooses on input i, $x \in D_i = \{0,1\}^{|i|}$ and $A_1(i,x) = f_i(x) = f(x)$. (Note that f is polynomial time computable.) Condition 4 in the definition of a collection of one way functions clearly follows from the similar condition for f to be a one way function.

Now suppose that $F = \{f_i : D_i \rightarrow R_i\}_{i \in I}$ is a collection of one way functions. Define $f_F(1^k, r_1, r_2) = A_1(S_1(1^k, r_1), S_2(S_1(1^k, r_1), r_2))$ where A_1, S_1, and S_2 are the functions associated with F as defined in Definition 2.2.12. In other words, f_F takes as input a string $1^k \circ r_1 \circ r_2$ where r_1 and r_2 will be the coin tosses of S_1 and S_2, respectively, and then

- Runs S_1 on input 1^k using the coin tosses r_1 to get the index $i = S_1(1^k, r_1)$ of a function $f_i \in F$.
- Runs S_2 on the output i of S_1 using the coin tosses r_2 to find an input $x = S_2(i, r_2)$.

- Runs A_1 on i and x to compute $f_F(1^k, r_1, r_2) = A_1(i, x) = f_i(x)$.

Note that randomization has been restricted to the input of f_F and since A_1 is computable in polynomial time, the conditions of a one way function are clearly met. ▌

A possible example is the following, treated thoroughly in Section 2.3.

Example 2.2.14 The hardness of computing discrete logarithms yields the following collection of functions. Define $EXP = \{EXP_{p,g}(i) = g^i \bmod p,\ EXPp, g : Z_p \rightarrow Z_p^*\}_{<p,g> \in I}$ for $I = \{< p, g > p \text{ prime}, g \text{ generator for } Z_p^*\}$.

2.2.5 Trapdoor Functions and Collections

Infromally, a *trapdoor function* f is a one-way function with an extra property. There also exists a secret inverse function (the*trapdoor*) that allows its possessor to efficiently invert f at any point in the domain of his choosing. It should be easy to compute f on any point, but infeasible to invert f on any point without knowledge of the inverse function . Moreover, it should be easy to generate matched pairs of f's and corresponding trapdoor. Once a matched pair is generated, the publication of f should not reveal anything about how to compute its inverse on any point.

Definition 2.2.15 A *trapdoor* function is a one-way function $f : \{0,1\}^* \rightarrow \{0,1\}^*$ such that there exists a polynomial p and a probabilistic polynomial time algorithm I such that for every k there exists an $t_k \in \{0,1\}^*$ such that $|t_k| \leq p(k)$ and for all $x \in \{0,1\}^*$, $I(f(x), t_k) = y$ such that $f(y) = f(x)$.

An example of a function which may be trapdoor if factoring integers is hard was proposed by Rabin[146]. Let $f(x, n) = x^2 \bmod n$ where $n = pq$ a product of two primes and $x \in Z_n^*$. Rabin[146] has shown that inverting f is easy iff factoring composite numbers product of two primes is easy. The most famous candidate trapdoor function is the RSA[151] function $f(x, n, l) = x^l \bmod n$ where $(l, \phi(n)) = 1$.
Again it will be more convenient to speak of families of trapdoor functions parameterized by security parameter k.

Definition 2.2.16 Let I be a set of indices and for $i \in I$ let D_i be finite. A collection of strong one way trapdoor functions is a set $F = \{f_i : D_i \rightarrow D_i\}_{i \in I}$ satisfying the following conditions.

(1) There exists a polynomial p and a PTM S_1 which on input 1^k outputs pairs (i, t_i) where $i \in I \cap \{0,1\}^k$ and $|t_i| < p(k)$ The information t_i is referred to as the trapdoor of i.

(2) There exists a PTM S_2 which on input $i \in I$ outputs $x \in D_i$

(3) There exists a PTM A_1 such that for $i \in I$, $x \in D_i$ $A_1(i, x) = f_i(x)$.

(4) There exists a PTM A_2 such that $A_2(i, t_i, f_i(x)) = x$ for all $x \in D_i$ and for all $i \in I$ (that is, f_i is easy to invert when t_i is known).

(5) For every PPT A there exists a negligble ν_A such that $\forall$ k large enough

$$\Pr\left[f_i(z) \neq y \ : \ i \stackrel{R}{\leftarrow} I \ ; \ x \stackrel{R}{\leftarrow} D_i \ ; \ y \leftarrow f_i(x) \ ; \ z \leftarrow A(i, y)\right] \ \leq \ \nu_A(k)$$

A possible example is the following treated in in detail in the next sections.

Example 2.2.17 [The RSA collections of possible trapdoor functions] Let p, q denote primes, $n = pq$, $Z_n^* = \{1 \leq x \leq n, (x, n) = 1\}$ the multiplicative group whose cardinality is $\varphi(n) = (p-1)(q-1)$, and $e \in Z_{p-1}$ relatively prime to $\varphi(n)$. Our set of indices will be $I = \{< n, e >$ such that $n = pq$ $|p| = |q|\}$ and the trapdoor associated with the particular index $< n, e >$ be d such that $ed = 1 \bmod \phi(n)$. Let $RSA = \{RSA_{<n,e>} : Z_n^* \rightarrow Z_n^*\}_{<n,e> \in I}$ where $RSA_{<n,e>}(x) = x^e \bmod n$

2.3 In Search of Examples

Number theory provides a source of candidates for one way and trapdoor functions. Let us start our search for examples by a digression into number theorey. See also the mini-course on number theory in Appendix B.

Calculating Inverses in Z_p^*

Consider the set $\mathbf{Z}_p^* = \{x : 1 \leq x < p \text{ and } \gcd(x,p) = 1\}$ where p is prime. $\mathbf{Z}_p^*$ is a group under multiplicaton modulo p. Note that to find the inverse of $x \in \mathbf{Z}_p^*$; that is, an element $y \in \mathbf{Z}_p^*$ such that $yx \equiv 1 \bmod p$, we can use the Euclidean algorithm to find integers y and z such that $yx + zp = 1 = \gcd(x,p)$. Then, it follows that $yx \equiv 1 \bmod p$ and so $y \bmod p$ is the desired inverse.

The Euler Totient Function $\varphi(n)$

Euler's Totient Function φ is defined by $\varphi(n) = |\{x : 1 \leq x < p \text{ and } \gcd(x,n) = 1\}$. The following are facts about φ.

(1) For p a prime and $\alpha \geq 1$, $\varphi(p^\alpha) = p^{\alpha-1}(p-1)$.

(2) For integers m, n with $\gcd(m,n) = 1$, $\varphi(mn) = \varphi(m)\varphi(n)$.

Using the rules above, we can find φ for any n because, in general,

$$\begin{aligned}
\varphi(n) &= \varphi(\prod_{i=1}^{k} {p_i}^{\alpha_i}) \\
&= \prod_{i=1}^{k} \varphi({p_i}^{\alpha_i}) \\
&= \prod_{i=1}^{k} {p_i}^{\alpha_i - 1}(p_i - 1)
\end{aligned}$$

Z_p^* Is Cyclic

A group G is cyclic if and only if there is an element $g \in G$ such that for every $a \in G$, there is an integer i such that $g^i = a$. We call g a *generator* of the group G and we denote the index i by $ind_g(a)$.

Theorem 2.3.1 (Gauss) *If p is prime then $\mathbf{Z}_p^*$ is a cyclic group of order $p-1$. That is, there is an element $g \in \mathbf{Z}_p^*$ such that $g^{p-1} \equiv 1 \bmod p$ and $g^i \not\equiv 1 \bmod p$ for $i < p-1$.*

From Theorem 2.3.1 the following fact is immediate.

Fact 2.3.2 Given a prime p, a generator g for $\mathbf{Z}_p^*$, and an element $a \in \mathbf{Z}_p^*$, there is a unique $1 \leq i \leq p-1$ such that $a = g^i$.

The Legendre Symbol

Fact 2.3.3 If p is a prime and g is a generator of $\mathbf{Z}_p^*$, then

$$g^c = g^a g^b \bmod p \Leftrightarrow c = a + b \bmod p - 1$$

From this fact it follows that there is an homomorphism $f : \mathbf{Z}_p^* \rightarrow \mathbf{Z}_{p-1}$ such that $f(ab) = f(a) + f(b)$. As a result we can work with $\mathbf{Z}_{p-1}$ rather than $\mathbf{Z}_p^*$ which sometimes simplifies matters. For example, suppose we wish to determine how many elements in $\mathbf{Z}_p^*$ are perfect squares (these elements will be referred to as *quadratic residues* modulo p). The following lemma tells us that the number of quadratic residues modulo p is $\frac{1}{2}|\mathbf{Z}_p^*|$.

Lemma 2.3.4 *$a \in \mathbf{Z}_p^*$ is a quadratic residue modulo p if and only if $a = g^x \bmod p$ where x satisfies $1 \leq x \leq p-1$ and is even.*

Proof: Let g be a generator in $\mathbf{Z}_p^*$.
($\Leftarrow$) Suppose an element $a = g^{2x}$ for some x. Then $a = s^2$ where $s = g^x$.

($\Rightarrow$) Consider the square of an element $b = g^y$. $b^2 = g^{2y} \equiv g^e \bmod p$ where e is even since $2y$ is reduced modulo $p-1$ which is even. Therefore, only those elements which can be expressed as g^e, for e an even integer, are squares. ∎

Consequently, the number of quadratic residues modulo p is the number of elements in $\mathbf{Z}_p^*$ which are an even power of some given generator g. This number is clearly $\frac{1}{2}|\mathbf{Z}_p^*|$.

The *Legendre Symbol* $\mathbf{J}_p(x)$ specifies whether x is a perfect square in $\mathbf{Z}_p^*$ where p is a prime.

$$\mathbf{J}_p(x) = \begin{cases} 1 & \text{if } x \text{ is a square in } \mathbf{Z}_p^* \\ 0 & \text{if } \gcd(x,p) \neq 1 \\ -1 & \text{if } x \text{ is not a square in } \mathbf{Z}_p^* \end{cases}$$

The Legendre Symbol can be calculated in polynomial time due to the following theorem.

Theorem 2.3.5 *[Euler's Criterion]* $\mathbf{J}_p(x) \equiv x^{\frac{p-1}{2}} \bmod p$.

Using repeated doubling to compute exponentials, one can calculate $x^{\frac{p-1}{2}}$ in $O(|p|^3)$ steps. Though this $\mathbf{J}_p(x)$ can be calculated when p is a prime, it is not known how to determine for general x and n, whether x is a square in $\mathbf{Z}_n^*$.

2.3.1 The Discrete Logarithm Function

Let EXP be the function defined by $\text{EXP}(p,g,x) = (p,g,g^x \bmod p)$. We are particularly interested in the case when p is a prime and g is a generator of Z_p^*. Deine an index set $I = \{(p,g) : p \text{ is prime and } g \text{ is a generator of } Z_p^*\}$. For $(p,g) \in I$, it follows by Fact 2.3.2 that $\text{EXP}(p,g,x)$ has a unique inverse and this allows us to define for $y \in \mathbf{Z}_p^*$ the discrete logarithm function DL by $\text{DL}(p,g,y) = (p,g,x)$ where $x \in \mathbf{Z}_{p-1}$ and $g^x \equiv y \bmod p$. Given p and g, $\text{EXP}(p,g,x)$ can easily be computed in polynomial time. However, it is unknown whether or not its inverse DL can be computed in polynomial time unless $p-1$ has very small factors (see [140]). Pohlig and Hellman [140] present effective techniques for this problem when $p-1$ has only small prime factors.

The best fully proved up-to-date algorithm for computing discrete logs is the Index-calculus algorithm. The expected running time of such algorithm is polynomial in $e^{\sqrt{\log k \log\log k}}$ where k is the size of the modulos p. There is a recent variant of the number field sieve algorithm for discrete logarithm which seems to work in faster running time of $e^{(\log k \log\log k)^{\frac{1}{3}}}$. It interesting to note that working over the finite field $GF(2^k)$ rather than working modulo p seems to make the problem substantially easier (see Coppersmith [53] and Odlyzko [135]). Curiously, computing discrete logarithms and factoring integers seem to have essentially the same difficulty at least as indicated by the current state of the art algorithms.

With all this in mind, we consider EXP a good candidate for a one way function. We make the following explicit assumption in this direction. The assumption basically says that there exists no polynomial time algorithm that can solvethe discrete log problem with prime modulos.

Strong Discrete Logarithm Assumption (DLA):[1] For every polynomial Q and every PPT A, for all sufficiently large k,

$$\Pr[A(p,g,y) = x \text{ such that } y \equiv g^x \bmod p \text{ where } 1 \leq x \leq p-1] < \frac{1}{Q(k)}$$

(where the probability is taken over all primes p such that $|p| \leq k$, the generators g of $\mathbf{Z}_p^*$, $x \in \mathbf{Z}_p^*$ and the coin tosses of A).

An immediate consequence of this assumption we get

[1] We note that a weaker assumption can be made concerning the discrete logarithm problem, and by the standard construction one can still construct a strong one-way function. We will assume for the purpose of the course the first stronger assumption. **Weak Discrete Logarithm Assumption:** There is a polynomial Q such that for every PTM A there exists an integer k_0 such that $\forall k > k_0$ $\Pr[A(p,g,y) = x \text{ such that } y \equiv g^x \bmod p \text{ where } 1 \leq x \leq p-1] < 1 - \frac{1}{Q(k)}$ (where the probability is taken over all primes p such that $|p| \leq k$, the generators g of $\mathbf{Z}_p^*$, $x \in \mathbf{Z}_p^*$ and the coin tosses of A).

Theorem 2.3.6 *Under the strong discrete logarithm assumption there exists a strong one way function; namely, exponentiation modulo a prime p.*

Some useful properties of EXP and DL follow.

Remark 2.3.7 If $\mathrm{DL}(p, g_1, y)$ is easy to calculate for some generator $g_1 \in \mathbf{Z}_p^*$ then it is also easy to calculate $\mathrm{DL}(p, g_2, y)$ for any other generator $g_2 \in \mathbf{Z}_p^*$. (The group $\mathbf{Z}_p^*$ has $\varphi(p-1)$ generators.) To see this suppose that $x_1 = \mathrm{DL}(p, g_1, y)$ and $x_2 = \mathrm{DL}(p, g_2, y)$. If $g_2 \equiv {g_1}^z \bmod p$ where $\gcd(z, p-1)$ then $y \equiv {g_1}^{x_2 z} \bmod p$ and consequently, $x_2 \equiv z^{-1} x_1 \bmod p-1$.

The following result shows that to efficiently calculate $\mathrm{DL}(p, g, y)$ for $(p, g) \in I$ it will suffice to find a polynomial time algorithm which can calculate $\mathrm{DL}(p, g, y)$ on at least a $\frac{1}{Q(|p|)}$ fraction of the possible inputs $y \in \mathbf{Z}_p^*$ for some polynomial Q.

Proposition 2.3.8 *Let ϵ, $\delta \in (0,1)$ and let S be a subset of the prime integers. Suppose there is a probabilistic algorithm A such that for all primes $p \in S$ and for all generators g of $\mathbf{Z}_p^*$*

$$\Pr[A(p, g, y) = x \text{ such that } g^x \equiv y \bmod p] > \epsilon$$

(where the probability is taken over $y \in \mathbf{Z}_p^*$ and the coin tosses of A) and A runs in time polynomial in $|p|$. Then there is a probabilistic algorithm A' running in time polynomial in $\epsilon^{-1}, \delta^{-1}$, and $|p|$ such that for all primes $p \in S$, generators g of $\mathbf{Z}_p^*$, and $y \in \mathbf{Z}_p^*$

$$\Pr[A'(p, g, y) = x \text{ such that } g^x \equiv y \bmod p] > 1 - \delta$$

(where the probability is taken over the coin tosses of A').

Proof: Choose the smallest integer N for which $\frac{1}{e^N} < \delta$.

Consider the algorithm A′ running as follows on inputs $p \in S$, g a generator of $\mathbf{Z}_p^*$ and $y \in \mathbf{Z}_p^*$.

Repeat $\epsilon^{-1} N$ times.
    Randomly choose z such that $1 \le z \le p-1$.
    Let $w = A(p, g, g^z y)$
    If A succeeds then $g^w = g^z y = g^{z+x} \bmod p$ where $x = \mathrm{DL}_{p,g}(y)$
        and therefore $\mathrm{DL}_{p,g}(y) = w - z \bmod p-1$.
    Otherwise, continue to next iteration.
End loop

We can estimate the probability that A′ fails:

$$\begin{aligned}\Pr[A'(p, g, y) \text{ fails}] &= \Pr[\text{A single iteration of the loop of A}' \text{ fails}]^{\epsilon^{-1} N} \\ &< (1-\epsilon)^{\epsilon^{-1} N} \\ &< (e^{-N}) \\ &< \delta\end{aligned}$$

Note that since $N = O(\log(\delta^{-1})) = O(\delta^{-1})$, A′ is a probabilistic algorithm which runs in time polynomial in ϵ^{-1}, δ^{-1}, and $|p|$. ∎

The discrete logarithm problem also yields the following collection of functions.
Let $I = \{(p, g) : p \text{ is prime and } g \text{ is a generator of } \mathbf{Z}_p^*\}$ and define

$$\mathrm{EXP} = \{\mathrm{EXP}_{p,g} : \mathbf{Z}_{p-1} \to \mathbf{Z}_p^* \text{ where } \mathrm{EXP}_{p,g}(x) = g^x \bmod p\}_{(p,g) \in I}.$$

Then, under the strong discrete logarithm assumption, EXP is a collection of strong one way functions. This claim will be shown to be true next.

Theorem 2.3.9 *Under the strong discrete logarithm assumption there exists a collection of strong one way functions.*

Proof: We shall show that under the DLA EXP is indeed a collection of one way functions. For this we must show that it satisfies each of the conditions in the definition of a collection of one way functions.

For condition 1, define S_1 to run as follows on input 1^k.

(1) Run Bach's algorithm (given in [8]) to get a random integer n such that $|n| = k$ along with its factorization.
(2) Test whether $n+1$ is prime. See primality testing in section B.9.
(3) If so, let $p = n+1$. Given the prime factorization of $p-1$ we look for generators g of $\mathbf{Z}_p^*$ as follows.
  (1) Choose $g \in \mathbf{Z}_p^*$ at random.
  (2) If $p-1 = \prod_i q_i^{\alpha_i}$ is the prime factorization of $p-1$ then for each q_i check that $g^{\frac{p-1}{q_i}} \not\equiv 1 \bmod p$.
  If so, then g is a generator of $\mathbf{Z}_p^*$. Output p and g.
  Otherwise, repeat from step 1.

Claim 2.3.10 g is a generator of $\mathbf{Z}_p^*$ if for each prime divisor q of $p-1$, $g^{\frac{p-1}{q}} \not\equiv 1 \bmod p$.

Proof: The element g is a generator of $\mathbf{Z}_p^*$ if $g^{p-1} \equiv 1 \bmod p$ and $g^j \not\equiv 1 \bmod p$ for all j such that $1 \leq j < p-1$; that is, g has order $p-1$ in $\mathbf{Z}_p^*$.

Now, suppose that g satisfies the condition of Claim 2.3.10 and let m be the order of g in $\mathbf{Z}_p^*$. Then $m \mid p-1$. If $m < p-1$ then there exists a prime q such that $m \mid \frac{p-1}{q}$; that is, there is an integer d such that $md = \frac{p-1}{q}$. Therefore $g^{\frac{p-1}{q}} = (g^m)^d \equiv 1 \bmod n$ contradicting the hypothesis. Hence, $m = p-1$ and g is a generator of $\mathbf{Z}_p^*$. ∎

Also, note that the number of generators in $\mathbf{Z}_p^*$ is $\varphi(p-1)$ and in [155] it is shown that

$$\varphi(k) > \frac{k}{6 \log\log k}.$$

Thus we expect to have to choose $O(\log\log p)$ candidates for g before we obtain a generator. Hence, S_1 runs in expected polynomial time.

For condition 2 in the definition of a collection of one way functions, we can define S_2 to simply output $x \in Z_{p-1}$ at random given $i = (p, g)$.

Condition 3 is true since the computation of $g^x \bmod p$ can be performed in polynomial time and condition 4 follows from the strong discrete logarithm assumption. ∎

2.3.2 The RSA function

In 1977 Rivest, Shamir, and Adleman [151] proposed trapdoor function candidate motivated by finding a public-key cryptosystem satisfying the requirements proposed by Diffie and Hellman. The trapdoor function proposed is $RSA(n, e, x) = x^e \bmod n$ where the case of interest is that n is the product of two large primes p and q and $\gcd(e, \phi(n)) = 1$. The corresponding trapdoor information is d such that $d \cdot e \equiv 1 \bmod \phi(n)$.

Viewd as a collection, let RSA $= \{\text{RSA}_{n,e} : \mathbf{Z}_n^* \to \mathbf{Z}_n^* \text{ where } \text{RSA}_{n,e}(x) = x^e \bmod n\}_{(n,e)\in I}$. for $I = \{< n, e > \text{ s.t. } n = pq \ |p| = |q|, (e, \phi(n)) = 1\}$.

RSA is easy to compute. How hard is it to invert? We know that if we can factor n we can invert RSA via the Chinese Remainder Theorem, however we don't know if the converse is true. Thus far, the best way known to invert RSA is to first factor n. There are a variety of algorithms for this task. The

best running time for a fully proved algorithm is Dixon's random squares algorithms which runs in time $O(e^{\sqrt{\log n \log\log n}})$. In practice we may consider others. Let $\ell = |p|$ where p is the smallest prime divisor of n. The Elliptic Curve algorithm takes expected time $O(e^{\sqrt{2\log \ell \log\log \ell}})$. The Quadratic Sieve algorithm runs in expected $O(e^{\sqrt{\log n \log\log n}})$. Notice the difference in the argument of the superpolynomial component of the running time. This means that when we suspect that one prime factor is substantially smaller than the other, we should use the Elliptic Curve method, otherwise one should use the Quadratic sieve. The new number field sieve algorithm seems to achieve a $O(e^{\log n \log\log n^{\frac{1}{3}}})$ running time which is a substantial improvement asymptotically although in practice it still does not seem to run faster than the Quadratic Sieve algorithm for the size of integers which people currently attempt to factor. The recommended size for n these days is 1024 bits.

With all this in mind, we make an explicit assumption under which one can prove that RSA provides a collection of trapdoor functions.

Strong RSA Assumption:[2] Let $H_k = \{n = pq : p \neq q$ are primes and $|p| = |q| = k\}$. Then for every polynomial Q and every PTM A, there exists an integer k_0 such that $\forall k > k_0$

$$\Pr[A(n, e, \mathrm{RSA}_{n,e}(x)) = x] < \frac{1}{Q(k)}$$

(where the probability is taken over all $n \in H_k$, e such that $\gcd(e, \varphi(n)) = 1$, $x \in \mathbf{Z}_n^*$, and the coin tosses of A).

We need to prove some auxilary claims.

Claim 2.3.11 For $(n, e) \in I$, $\mathrm{RSA}_{n,e}$ is a permutation over $\mathbf{Z}_n^*$.

Proof: Since $\gcd(e, \varphi(n)) = 1$ there exists an integer d such that $ed \equiv 1 \bmod \varphi(n)$. Given $x \in \mathbf{Z}_n^*$, consider the element $x^d \in \mathbf{Z}_n^*$. Then $\mathrm{RSA}_{n,e}(x^d) \equiv (x^d)^e \equiv x^{ed} \equiv x \bmod n$. Thus, the function $\mathrm{RSA}_{n,e} : \mathbf{Z}_n^* \longrightarrow \mathbf{Z}_n^*$ is onto and since $|\mathbf{Z}_n^*|$ is finite it follows that $\mathrm{RSA}_{n,e}$ is a permutation over $\mathbf{Z}_n^*$. ∎

Remark 2.3.12 Note that the above is a constructive proof that RSA has an unique inverse. Since $\gcd(e, \varphi(n)) = 1$ if we run the extended Euclidean algorithm we can find $d \in \mathbf{Z}_n^*$ such that

$$RSA_{n,e}^{-1}(x) = (x^e \bmod n)^d \bmod n = x^{ed} \bmod n = x \bmod n$$

. Note that once we found a d such that $ed \equiv 1 \bmod \varphi(n)$ then we can invert $\mathrm{RSA}_{n,e}$ efficiently because then $\mathrm{RSA}_{n,e}(x)^d \equiv x^{ed} \equiv x \bmod \varphi(n)$.

Theorem 2.3.13 *Under the strong RSA assumption, RSA is a collection of strong one way trapdoor permutations.*

Proof: First note that by Claim 2.3.11, $\mathrm{RSA}_{n,e}$ is a permutation of $\mathbf{Z}_n^*$. We must also show that RSA satisfies each of the conditions in Definition 2.2.16. For condition 1, define S_1 to compute, on input 1^k,

a pair $(n, e) \in I \cap \{0,1\}^k$ and corresponding d such that $ed \equiv 1 \bmod \varphi(n)$. The algorithm picks two random primes of equal size by choosing random numbers and testing them for primality and setting n to be their procuct, then $e \in Z_{\phi(n)}$ is chosen at random, and finally d is computed in polynomial time by first computing $\varphi(n) = (p-1)(q-1)$ and then using the extended Euclidean algorithm. For condition 2, define S_2 to randomly generate $x \in \mathbf{Z}_n^*$ on input (n, e). Let $A_1((n, e), x) = \mathrm{RSA}_{n,e}(x)$. Note that exponentiation modulo n is a polynomial time computation and therefore condition 3 holds. Condition 4 follows from the Strong RSA assumption. For condition 5, let $A_2((n, e), d, \mathrm{RSA}_{n,e}(x)) \equiv \mathrm{RSA}_{n,e}(x)^d \equiv x^{ed} \equiv x \bmod n$ and this is a polynomial time computation. ∎

[2] A weaker assumption can be made which under standard constructions is equivalent to the stronger one which is made in this class. **Weak RSA Assumption:** Let $H_k = \{n = pq : p \neq q$ are prime and $|p| = |q| = k\}$. There is a polynomial Q such that for every PTM A, there exists an integer k_0 such that $\forall k > k_0$ $\Pr[A(n, e, \mathrm{RSA}_{n,e}(x)) = x] < 1 - \frac{1}{Q(k)}$ (where the probability is taken over all $n \in H_k$, e such that $\gcd(e, \varphi(n)) = 1$, $x \in \mathbf{Z}_n^*$, and the coin tosses of A).

One of the properties of the RSA function is that if we have a polynomial time algorithm that inverts $RSA_{n,e}$ on at least a polynomial proportion of the possible inputs $x \in Z_n^*$ then a subsequent probabilistic expected polynomial time algorithm can be found which inverts $\mathrm{RSA}_{n,e}$ on almost all inputs $x \in Z_n^*$. This can be taken to mean that for a given n, e if the function is hard to invert then it is almost everywhere hard to invert.

Proposition 2.3.14 *Let ϵ, $\delta \in (0,1)$ and let $S \subseteq I$. Suppose there is a probabilistic algorithm A such that for all $(n,e) \in S$*

$$\Pr[A(n, e, RSA_{n,e}(x)) = x] > \epsilon$$

(where the probability is taken over $x \in \mathbf{Z}_n^*$ and the coin tosses of A) and A runs in time polynomial in $|n|$. Then there is a probabilistic algorithm A' running in time polynomial in $\epsilon^{-1}, \delta^{-1}$, and $|n|$ such that for all $(n,e) \in S$, and $x \in \mathbf{Z}_n^*$

$$\Pr[A'(n, e, RSA_{n,e}(x)) = x] > 1 - \delta$$

(where the probability is taken over the coin tosses of A').

Proof: Choose the smallest integer N for which $\frac{1}{e^N} < \delta$.

Consider the algorithm A′ running as follows on inputs $(n,e) \in S$ and $\mathrm{RSA}_{n,e}(x)$.

Repeat $\epsilon^{-1}N$ times.
    Randomly choose $z \in \mathbf{Z}_n^*$.
    Let $y = A(n, e, \mathrm{RSA}_{n,e}(x) \cdot \mathrm{RSA}_{n,e}(z)) = A(n, e, \mathrm{RSA}_{n,e}(xz))$.
    If A succeeds then $y = xz$ and therefore $x = yz^{-1} \bmod n$. Output x.
    Otherwise, continue to the next iteration.
End loop

We can estimate the probability that A′ fails:

$$\begin{aligned}\Pr[A'(n, e, RSA_{n,e}(x)) \neq x] &= \Pr[\text{A single iteration of the loop of A}' \text{ fails}]^{\epsilon^{-1}N} \\ &< (1-\epsilon)^{\epsilon^{-1}N} \\ &< (e^{-N}) \\ &< \delta\end{aligned}$$

Note that since $N = O(\log(\delta^{-1})) = O(\delta^{-1})$, A′ is a probabilistic algorithm which runs in time polynomial in ϵ^{-1}, δ^{-1}, and $|n|$. ∎

Open Problem 2.3.15 It remains to determine whether a similar result holds if the probability is also taken over the indices $(n,e) \in I$. Specifically, if ϵ, $\delta \in (0,1)$ and A is a PTM such that

$$\Pr[A(n, e, RSA_{n,e}(x)) = x] > \epsilon$$

(where the probability is taken over $(n,e) \in I$, $x \in \mathbf{Z}_n^*$ and the coin tosses of A), does there exist a PTM A' running in time polynomial in ϵ^{-1} and δ^{-1} such that

$$\Pr[A'(n, e, RSA_{n,e}(x)) = x] > 1 - \delta$$

(where the probability is taken over $(n,e) \in I$ and the coin tosses of A′)?

2.3.3 Connection Between The Factorization Problem And Inverting RSA

Fact 2.3.16 If some PPT algorithm A can factor n then there exists a PPT A' that can invert $RSA_{\langle n,e\rangle}$.

The proof is obvious as $\phi(n) = (p-1)(q-1)$. The trapdoor information d can be found by using the extended Euclidean algorithm because $d = e^{-1} \bmod \phi(n)$.

Fact 2.3.17 If there exists a PTM B which on input $\langle n, e\rangle$ finds d such that $ed \equiv 1 \bmod \phi(n)$ then there exists a PTM, B$'$ that can factor n.

Open Problem 2.3.18 It remains to determine whether inverting RSA and factoring are equivalent. Namely, if there is a PTM C which, on input $\langle n, e\rangle$, can invert $RSA_{\langle n,e\rangle}$, does there exist a PTM C' that can factor n? The answer to this question is unknown. Note that Fact 2.3.17 does not imply that the answer is yes, as there may be other methods to invert RSA which do not necessarily find d.

2.3.4 The Squaring Trapdoor Function Candidate by Rabin

Rabin in [146] introduced a candidate trapdoor function which we call the squaring function. The squaring function resemble the RSA function except that Rabin was able to actually *prove* that inverting the squaring function is as hard as factoring integers. Thus, inverting the squaring function is a computation which is at least as hard as inverting the RSA function and possibly harder.

Definition 2.3.19 Let $I = \{n = pq : p \text{ and } q \text{ are distinct odd primes.}\}$. For $n \in$ I, the squaring function $SQUARE_n : \mathbf{Z}_n^* \longrightarrow \mathbf{Z}_n^*$ is defined by $SQUARE_n(x) \equiv x^2 \bmod n$. The trapdoor information of $n = pq \in I$ is $t_n = (p, q)$. We will denote the entire collection of Rabin's functions by RABIN $= \{SQUARE_n : \mathbf{Z}_n^* \longrightarrow \mathbf{Z}_n^*\}_{n\in I}$.

Remark 2.3.20 Observe that while Rabin's function squares its input, the RSA function uses a varying exponent; namely, e where $\gcd(e, \phi(n)) = 1$. The requirement that $\gcd(e, \phi(n)) = 1$ guarentees that the RSA function is a permutation. On the other hand, Rabin's function is 1 to 4 and thus it does not have a uniquely defined inverse. Specifically, let $n = pq \in I$ and let $a \in \mathbf{Z}_p^*$. As discussed in section B.4, if $a \equiv x^2 \bmod p$ then x and $-x$ are the distinct square roots of a modulo p and if $a \equiv y^2 \bmod q$ then y and $-y$ are the distinct square roots of a modulo q. Then, there are four solutions to the congruence $a \equiv z^2 \bmod n$, constructed as follows. Let $c, d \in \mathbf{Z}_n$ be the Chinese Remainder Theorem coefficients as discussed in Appendix B.4. Then

$$c = \begin{cases} 1 \bmod p \\ 0 \bmod q \end{cases}$$

and

$$d = \begin{cases} 0 \bmod p \\ 1 \bmod q \end{cases}$$

and the four solutions are $cx + dy$, $cx - dy$, $-cx + dy$, and $-cx - dy$.

The main result is that RABIN is a collection of strong one way trapdoor functions and the proof relies on an assumption concerning the difficulty of factoring. We state this assumption now.

Factoring Assumption: Let $H_k = \{pq : p \text{ and } q \text{ are prime and } |p| = |q| = k\}$. Then for every polynomial Q and every PTM A, $\exists k_0$ such that $\forall k > k_0$

$$\Pr[A(n) = p : p \mid n \text{ and } p \neq 1, n] < \frac{1}{Q(k)}$$

(where the probability is taken over all $n \in H_k$ and the coin tosses of A).

Our ultimate goal is to prove the following result.

Theorem 2.3.21 *Under the factoring assumption, RABIN is a collection of one way trapdoor functions.*

Before proving this, we consider two auxiliary lemmas. Lemma 2.3.22 constructs a polynomial-time machine A which computes square roots modulo a prime. Lemma 2.3.25 constructs another polynomial-time machine, SQRT, that inverts Rabin's function using the trapdoor information; specifically, it computes a square root modulo composites given the factorization. SQRT makes calls to A.

Lemma 2.3.22 *Let p be an odd prime and let a be a square modulo p. There exists a probabilistic algorithm A running in expected polynomial time such that $A(p,a) = x$ where $x^2 \equiv a \bmod p$.*

Proof: Let p be an odd prime and let a be a quadratic residue in $\mathbf{Z}_p^*$. There are two cases to consider; $p \equiv 1 \bmod 4$ and $p \equiv 3 \bmod 4$.

Case 1 $p \equiv 3 \bmod 4$; that is, $p = 4m+3$ for some integer m.

Since a is a square we have $1 = \mathbf{J}_p(a) \equiv a^{\frac{p-1}{2}} \bmod p \Longrightarrow a^{2m+1} \equiv 1 \bmod p$
$\Longrightarrow a^{2m+2} \equiv a \bmod p$

Therefore, a^{m+1} is a square root of a modulo p.

Case 2 $p \equiv 1 \bmod 4$; that is, $p = 4m+1$ for some integer m.

As in Case 1, we will attempt to find an odd exponent e such that $a^e \equiv 1 \bmod p$.

Again, a is a square and thus $1 = \mathbf{J}_p(a) \equiv a^{\frac{p-1}{2}} \bmod p \Longrightarrow a^{2m} \equiv 1 \bmod p$.

However, at this point we are not done as in Case 1 because the exponent on a in the above congruence is even. But notice that $a^{2m} \equiv 1 \bmod p \Longrightarrow a^m \equiv \pm 1 \bmod p$. If $a^m \equiv 1 \bmod p$ with m odd, then we proceed as in Case 1.

This suggests that we write $2m = 2^l r$ where r is an odd integer and compute $a^{2^{l-i}r} \bmod p$ for $i = 1, \ldots, l$ with the intention of reaching the congruence $a^r \equiv 1 \bmod p$ and then proceeding as in Case 1. However, this is not guarenteed as there may exist an integer l' satisfying $0 \leq l' < l$ such that $a^{2^{l'}r} \equiv -1 \bmod p$. If this congruence is encountered, we can recover as follows. Choose a quadratic nonresidue $b \in \mathbf{Z}_p^*$. Then $-1 = \mathbf{J}_p(b) \equiv b^{\frac{p-1}{2}} \bmod p$ and therefore $a^{2^{l'}r} \cdot b^{2^{l'}r} = a^{2^{l'}r} \cdot b^{\frac{p-1}{2}} \equiv 1 \bmod p$. Thus, by multiplying by $b^{2^{l'}r} \equiv -1 \bmod p$, we obtain a new congruence $\left(a^r b^{2^{l-l'}r}\right)^{2^{l'}} \equiv 1 \bmod p$. We proceed by taking square roots in this congruence. Since $l' < l$, we will, after l steps, arrive at $a^r b^{2s} \equiv 1 \bmod p$ where s is integral. At this point we have $a^{r+1} b^{2s} \equiv a \bmod p \Longrightarrow a^{\frac{r+1}{2}} b^s$ is a square root of $a \bmod p$.

From the above discussion (Cases 1 and 2) we obtain a probabilistic algorithm A for taking square roots. The algorithm A runs as follows on input a, p where $\mathbf{J}_p(a) = 1$.

(1) If $p = 4m+3$ for some integer m then output a^{m+1} as a square root of $a \bmod p$.

(2) If $p = 4m+1$ for some integer m then randomly choose $b \in \mathbf{Z}_p^*$ until a value is found satisfying $\mathbf{J}_p(b) = -1$.

  (1) Initialize $i = 2m$ and $j = 0$.

  (2) Repeat until i is odd.
  $i \leftarrow \frac{i}{2}$ and $j \leftarrow \frac{j}{2}$.
  If $a^i b^j = -1$ then $j \leftarrow j + 2m$.
  Output $a^{\frac{i+1}{2}} b^{\frac{j}{2}}$ as a square root of $a \bmod p$.

This algorithm terminates after $O(l)$ iterations because in step 2 (ii) the exponent on a is divided by 2. Note also, that since exactly half of the elements in $\mathbf{Z}_p^*$ are quadratic nonresidues, it is expected that 2 iterations

will be required to find an appropriate value for b at the beginning of step 2. Thus, A runs in expected polynomial time and this completes the proof of Lemma 2.3.22. ▮

Remark 2.3.23 There is a deterministic algorithm due to René Schoof (see [162]) which computes the square root of a quadratic residue a modulo a prime p in time polynomial in $|p|$ and a (specifically, the algorithm requires $O((a^{\frac{1}{2}+\epsilon}\log p)^9)$ elementary operations for any $\epsilon > 0$). However, it is unknown whether there exists a deterministic algorithm running in time polynomial in $|p|$.

Open Problem 2.3.24 Does there exist a deterministic algorithm that computes square roots modulo a prime p in time polynomial in $|p|$?

The next result requires knowledge of the Chinese Remainder Theorem. The statement of this theorem as well as a constructive proof is given in Appendix B.4. In addition, a more general form of the Chinese Remainder Theorem is presented there.

Lemma 2.3.25 *Let p and q be primes, $n = pq$ and a a square modulo p. There exists a probabilistic algorithm $SQRT$ running in expected polynomial time such that $SQRT(p, q, n, a) = x$ where $x^2 \equiv a \bmod n$.*

Proof: The algorithm SQRT will first make calls to A, the algorithm of Lemma 2.3.22, to obtain square roots of a modulo each of the primes p and q. It then combines these square roots, using the Chinese Remainder Theorem, to obtain the required square root.

The algorithm SQRT runs as follows.

(1) Let $A(p, a) = x_1$ and $A(q, a) = x_2$.
(2) Use the Chinese Remainder Theorem to find (in polynomial time) $y \in \mathbf{Z}_n$ such that $y \equiv x_1 \bmod p$ and $y \equiv x_2 \bmod q$ and output y.

Algorithm SQRT runs correctly because $y^2 \equiv \begin{cases} x_1^2 \equiv a \bmod p \\ x_2^2 \equiv a \bmod q \end{cases} \Longrightarrow y^2 \equiv a \bmod n.$ ▮

On the other hand, if the factors of n are unknown then the computation of square roots modulo n is as hard as factoring n. We prove this result next.

Lemma 2.3.26 *Computing square roots modulo $n \in H_k$ is as hard as factoring n.*

Proof: Suppose that I is an algorithm which on input $n \in H_k$ and a a square modulo n outputs y such that $a \equiv y^2 \bmod n$ and consider the following algorithm B which on input n outputs a nontrivial factor of n.

(1) Randomly choose $x \in \mathbf{Z}_n^*$.
(2) Set $y = I(n, x^2 \bmod n)$.
(3) Check if $x \equiv \pm y \bmod n$. If not then $\gcd(x - y, n)$ is a nontrivial divisor of n. Otherwise, repeat from 1.

Algorithm B runs correctly because $x^2 \equiv y^2 \bmod n \Longrightarrow (x + y)(x - y) \equiv 0 \bmod n$ and so $n|[(x + y)(x - y)]$. But $n \not| (x - y)$ because $x \not\equiv y \bmod n$ and $n \not| (x + y)$ because $x \not\equiv -y \bmod n$. Therefore, $\gcd(x - y, n)$ is a nontrivial divisor of n. Note also that the congruence $a \equiv x^2 \bmod n$ has either 0 or 4 solutions (a proof of this result is presented in Appendix B.4). Therefore, if $I(n, x^2) = y$ then $x \equiv \pm y \bmod n$ with probability $\frac{1}{2}$ and hence the above algorithm is expected to terminate in 2 iterations. ▮

We are now in a position to prove the main result, Theorem 2.3.21.

Proof: For condition 1, define S_1 to find on input 1^k an integer $n = pq$ where p and q are primes of equal length and $|n| = k$. The trapdoor information is the pair of factors (p, q).

For condition 2 in the definition of a collection of one way trapdoor functions, define S_2 to simply output $x \in Z_n^*$ at random given n.

Condition 3 is true since the computation of $x^2 \bmod n$ can be performed in polynomial time and condition 4 follows from the factoring assumption and Lemma 2.3.26.

Condition 5 follows by applying the algorithm $SQRT$ from Lemma 2.3.25. ∎

Lemma 2.3.26 can even be made stronger as we can also prove that if the algorithm I in the proof of Lemma 2.3.26 works only on a small portion of its inputs then we are still able to factor in polynomial time.

Proposition 2.3.27 *Let $\epsilon, \delta \in (0,1)$ and let $S \subseteq H_k$. Suppose there is a probabilistic algorithm I such that for all $n \in S$*

$$\Pr[I(n,a) = x \text{ such that } a \equiv x^2 \bmod n] > \epsilon$$

(where the probability is taken over $n \in S$, $a \in \mathbf{Z}_n^{*\,2}$, and the coin tosses of I). Then there exists a probabilistic algorithm FACTOR running in time polynomial in ϵ^{-1}, δ^{-1}, and $|n|$ such that for all $n \in S$,

$$\Pr[FACTOR(n) = d \text{ such that } d \mid n \text{ and } d \neq 1, n] > 1 - \delta$$

(where the probability is taken over n and over the coins tosses of FACTOR).

Proof: Choose the smallest integer N such that $\frac{1}{e^N} < \delta$.

Consider the algorithm FACTOR running as follows on inputs $n \in S$.

Repeat $2\epsilon^{-1}N$ times.
    Randomly choose $x \in \mathbf{Z}_n^*$.
    Set $y = I(n, x^2 \bmod n)$.
    Check if $x \equiv \pm y \bmod n$. If not then $\gcd(x-y, n)$ is a nontrivial divisor of n.
    Otherwise, continue to the next iteration.
End loop

We can estimate the probability that FACTOR fails. Note that even when $I(n, x^2 \bmod n)$ produces a square root of $x^2 \bmod n$, FACTOR(n) will be successful exactly half of the time.

$$\begin{aligned}\Pr[\text{FACTOR}(n) \text{ fails to factor } n] &= \Pr[\text{A single iteration of the loop of FACTOR fails}]^{\epsilon^{-1}N} \\ &< (1 - \tfrac{1}{2}\epsilon)^{2\epsilon^{-1}N} \\ &< (e^{-N}) \\ &< \delta\end{aligned}$$

Since $N = O(\log(\delta^{-1})) = O(\delta^{-1})$, FACTOR is a probabilistic algorithm which runs in time polynomial in ϵ^{-1}, δ^{-1}, and $|n|$. ∎

2.3.5 A Squaring Permutation as Hard to Invert as Factoring

We remarked earlier that Rabin's function is not a permutation. If $n = pq$ where p and q are primes and $p \equiv q \equiv 3 \bmod 4$ then we can reduce the Rabin's function $SQUARE_n$ to a permutation g_n by restricting its domain to the quadratic residues in $\mathbf{Z}_n^*$, denoted by Q_n. This will yield a collection of one way permutations as we will see in Theorem 2.3.5. This suggestion is due to Blum and Williams.

Definition 2.3.28 Let $J = \{pq : p \neq q \text{ are odd primes}, |p| = |q|, \text{ and } p \equiv q \equiv 3 \bmod 4\}$. For $n \in$ J let the function $g_n : Q_n \longrightarrow Q_n$ be defined by $g_n(x) \equiv x^2 \bmod n$ and let BLUM-WILLIAMS $= \{g_n\}_{n \in J}$.

We will first prove the following result.

Lemma 2.3.29 *Each function $g_n \in$ BLUM-WILLIAMS is a permutation. That is, for every element $y \in Q_n$ there is a unique element $x \in Q_n$ such that $x^2 = y \bmod n$.*

Proof: Let $n = p_1 p_2 \in \mathbf{J}$. Note that by the Chinese Remainder Theorem, $y \in Q_n$ if and only if $y \in Q_n$ and $y \in Q_{p_1}$ and $y \in Q_{p_2}$. Let a_i and $-a_i$ be the square roots of $y \bmod p_i$ for $i = 1, 2$. Then, as is done in the proof of the Chinese Remainder Theorem, we can construct Chinese Remainder Theorem coefficients c_1, c_2 such that $c_1 = \begin{cases} 1 \bmod p_1 \\ 0 \bmod p_2 \end{cases}$ and $c_2 = \begin{cases} 0 \bmod p_1 \\ 1 \bmod p_2 \end{cases}$ and consequently, the four square

roots of $y \bmod n$ are $w_1 = c_1 a_1 + c_2 a_2$,
$w_2 = c_1 a_1 - c_2 a_2$,
$w_3 = -c_1 a_1 - c_2 a_2 = -(c_1 a_1 + c_2 a_2) = -w_1$,
and $w_4 = -c_1 a_1 + c_2 a_2 = -(c_1 a_1 - c_2 a_2) = -w_2$.

Since $p_1 \equiv p_2 \equiv 3 \bmod 4$, there are integers m_1 and m_2 such that $p_1 = 4m_1 + 3$ and $p_2 = 4m_2 + 3$. Thus, $\mathbf{J}_{p_1}(w_3) = \mathbf{J}_{p_1}(-w_1) = \mathbf{J}_{p_1}(-1)\mathbf{J}_{p_1}(w_1) = (-1)^{\frac{p_1-1}{2}}\mathbf{J}_{p_1}(w_1) = -\mathbf{J}_{p_1}(w_1)$ because $\frac{p_1-1}{2}$ is odd and similarly, $\mathbf{J}_{p_1}(w_4) = -\mathbf{J}_{p_1}(w_2)$, $\mathbf{J}_{p_2}(w_3) = -\mathbf{J}_{p_2}(w_1)$, and $\mathbf{J}_{p_2}(w_4) = -\mathbf{J}_{p_2}(w_2)$. Therefore, without loss of generality, we can assume that $\mathbf{J}_{p_1}(w_1) = \mathbf{J}_{p_1}(w_2) = 1$ (and so $\mathbf{J}_{p_1}(w_3) = \mathbf{J}_{p_1}(w_4) = -1$).

Since only w_1 and w_2 are squares modulo p_1 it remains to show that only one of w_1 and w_2 is a square modulo n or equivalently modulo p_2.

First observe that $\mathbf{J}_{p_2}(w_1) \equiv (w_1)^{\frac{p_2-1}{2}} \equiv (c_1 a_1 + c_2 a_2)^{2m_2+1} \equiv (a_2)^{2m_2+1} \bmod p_2$ and that $\mathbf{J}_{p_2}(w_2) \equiv (w_2)^{\frac{p_2-1}{2}} \equiv (c_1 a_1 - c_2 a_2)^{2m_2+1} \equiv (-a_2)^{2m_2+1} \bmod p_2$ (because $c_1 \equiv 0 \bmod p_2$ and $c_2 \equiv 1 \bmod p_2$). Therefore, $\mathbf{J}_{p_2}(w_2) = -\mathbf{J}_{p_2}(w_1)$. Again, without loss of generality, we can assume that $\mathbf{J}_{p_2}(w_1) = 1$ and $\mathbf{J}_{p_2}(w_2) = -1$ and hence, w_1 is the only square root of y that is a square modulo both p_1 and p_2. Therefore, w_1 is the only square root of y in Q_n. ∎

Theorem 2.3.30 [Williams, Blum] *BLUM-Williams is a collection of one-way trapdoor permutations.*

Proof: This follows immediately from Lemma 2.3.29 because each function $g_n \in \mathbf{J}$ is a permutation. The trapdoor information of $n = pq$ is $t_n = (p, q)$. ∎

2.4 Hard-core Predicate of a One Way Function

Recall that $f(x)$ does not necessarily hide everything about x even if f is a one-way function. E.g. if f is the RSA function then it preserves the Jacobi symbol of x, and if f is the discrete logarithm function EXP then it is easy to compute the least significant bit of x from $f(x)$ by a simple Legendre symbol calculation. Yet, it seems likely that there is at least one bit about x which is hard to "guess" from $f(x)$, given that x in its entirety is hard to compute. The question is: can we point to specific bits of x which are hard to compute, and how hard to compute are they. The answer is encouraging. A number of results are known which give a particular bit of x which is hard to guess given $f(x)$ for some particular f's such as RSA and the discrete logarithm function. We will survey these results in subsequent sections.

More generally, we call a predicate about x which is impossible to compute from $f(x)$ better than guessing it at random a *hard-core predicate* for f.

We first look at a general result by Goldreich and Levin [83] which gives for any one-way function f a predicate B such that it is as hard to guess $B(x)$ from $f(x)$ as it is to invert f.

Historical Note: The idea of a hard-core predicate for one-way functions was introduced by Blum, Goldwasser and Micali. It first appears in a paper by Blum and Micali [36] on pseduo random number generation. They showed that a if the EXP function ($f_{p,g}(x) = g^x \pmod p$) is hard to invert then it is hard to even guess better than guessing at random the most significant bit of x. Under the assumption that quadratic residues

are hard to distinguish from quadratic non-residues modulo composite moduli, Goldwasser and Micali in [87] showed that the squaring function has a hard core perdicate as well. Subsequently, Yao [184] showed a general result that given any one way function, there is a predicate $B(x)$ which is as hard to guess from $f(x)$ as to invert f for any function f. Goldreich and Levin's result is a significantly simpler construction than Yao's earlier construction.

2.4.1 Hard Core Predicates for General One-Way Functions

We now introduce the concept of a *hard-core predicate* of a function and show by explicit construction that any strong one way function can be modified to have a hard-core predicate.

Note: Unless otherwise mentioned, the probabilities during this section are calculated uniformly over all coin tosses made by the algorithm in question.

Definition 2.4.1 A hard-core predicate of a function $f : \{0,1\}^* \rightarrow \{0,1\}^*$ is a boolean predicate $B : \{0,1\}^* \rightarrow \{0,1\}$, such that

(1) $\exists PPT\, A$, such that $\forall x\, A(x) = B(x)$

(2) $\forall$ PPT G, $\forall$ constants c, $\exists\, k_0$, s.t. $\forall_{k>k_0}$

$$\Pr[G(f(x)) = B(x)] < \frac{1}{2} + \frac{1}{k^c}.$$

The probability is taken over the random coin tosses of G, and random choices of x of length k.

Intuitively, the definition guarantees that given x, $B(x)$ is efficiently computable, but given only $f(x)$, it is hard to even "guess" $B(x)$; that is, to guess $B(x)$ with a probability significantly better than $\frac{1}{2}$.

Yao, in [184], showed that the existence of any trapdoor length-preserving permutation implies the existence of a trapdoor predicate. Goldreich and Levin greatly simplified Yao's construction and show that any one-way function can be modified to have a trapdoor predicate as follows (we state a simple version of their general result).

Theorem 2.4.2 [83] *Let f be a (strong) length preserving one-way function. Define $f'(x \circ r) = f(x) \circ r$, where $|x| = |r| = k$, and $\circ$ is the concatenation function. Then*

$$B(x \circ r) = \Sigma_{i=1}^{k} x_i r_i (mod\ 2).$$

is a hard-core predicate for f'.

Note: $v \circ w$ denotes concatenation of strings v and w. Computing B from f' is trivial as $f(x)$ and r are easily recoveravle from $f'(x,r)$. Finaly notice that if f is one-way then so is f'.

Proof Sketch: The proof is by contradiction. Assume that the predicate B is not a hard-core for function f'. Then we show how to invert f, contradicting the assumption that it is one-way.

Following [83] we can change the range of b to $\{-1,+1\}$, and limit I to output numbers in the interval $[-1,+1]$. Then the probability in condition 2 above can be replaced by: $E_{x,\omega}(G(\omega, f(x)) \cdot b(x)) < \frac{1}{|x|^c}$, where the expectation E is taken uniformly over x and the coin tosses ω of G. We use the notation $G(\omega, f(x))$ to denote the execution of algorithm G with coins tosses specified by ω.

Using the guessing algorithm G, which given $f'(x,p)$ guesses with a noticeable correlation $B(x,p)$, we try to construct a list of elements on which G is more successful on the average (over the random internal coin-flips and p). This list contains x. And moreover, the list is not too long. The list is constructed by extending all potential prefixes of x one bit at a time (starting with the empty string) and "pruning" the ones that do not look "promising". To estimate how "promising" a prefix z is a function $c(z)$ is used, i.e. a prefix z is "promissing" if $c(z) > t$ for some threshold t. The function c is defined as $c(z) = E_s(E_r(B(z,r)E_\omega(G(\omega, y, rs)))^2$, where G is the guessing algorithm that on random tape ω, $y = f(x)$, and p attempts to guess $B(x,p)$. This function c has a number of important properties. First, it is monotone

under padding, i.e. $c(z\sigma) \leq c(z)$. Second, $\sum_z c(z) \leq 1$, where $|z| = k$. This is proven in Lemma 2 of [83]. The proof (which is much simplified from the earlier versions) uses the orthogonality of the matrix B defined by $B_{z,r} = B(z,r)$. Finally, c can be approximated efficiently. The monotonicity is useful because once we exclude a prefix as "not promising" we are guaranteed that none of its extensions could have become "promising". The second property guarantees that the number of "promising" prefixes is not too large. Namely, if we fix a threshold for $c(z)$, then there are at most polynomially many "promising" prefixes of the given length.

Instead of following the proof of [83] further we consider a simpler case which has a much simpler proof. This proof is actually very different from the one for the general theorem. Suppose, that the correlation of the guessing algorithm is not just noticeable, but *very* noticeable. Namely, $prob[G(f(x),p) = B(x,p)] > \frac{3}{4} + \epsilon$. Note, that we use $\frac{3}{4}$ instead of $\frac{1}{2}$. Let r_i be i-th digit of r. Then to compute x_i

(1) Select r at random.
(2) Define $r' : r'_j = r_j$ for all $j \neq i$, $r'_i = -r_i$
(3) $v := G(f(x),r) \cdot G(f(x),r')$
(4) $x_i := (1-v)/2$

In the step 3, *both* G's are correct with a noticeable correlation. Therefore, if they are both correct, $v = -1$ implies that changing i-th bit of r changes B, therefore i-th bit of x cannot be 0, i.e. $(-1)^{x_i} = B(x,r) \cdot B(x,r')$. For a full proof of the theorem we refer the reader to [83]. ∎

It is trivial to extend the definition of a hard-core predicate for a one way function, to a collection of hard core predicates for a collection of one-way functions.

Definition 2.4.3 A hard-core predicate of a one-way function collection $F = \{f_i : D_i \rightarrow R_i\}_{i \in I}$ is a collection of boolean predicates $B = \{B_i : D_i \rightarrow R_i\}_{i \in I}$ such that

(1) $\exists PPT\, A$, such that $\forall i, x\, A(i,x) = B_i(x)$
(2) $\forall$ PPT G, $\forall$ constants c, $\exists$, k_0, s.t. $\forall_{k>k_0}$

$$\Pr[G(i, f_i(x)) = B_i(x)] < \frac{1}{2} + \frac{1}{k^c}.$$

The probability is taken over the random coin tosses of G, random choices of $i \in I \cap \{0,1\}^k$ and random $x \in D_i$.

2.4.2 Bit Security Of The Discrete Logarithm Function

Let us examine the bit security of the EXP collection of functions directly rather than through the Goldreich Levin general construction.

We will be interested in the most significant bit of the discrete logarithm x of y modulo p.

For $(p,g) \in I$ and $y \in \mathbf{Z}_p^*$, let $B_{p,g}(y) = \begin{cases} 0 & \text{if } y = g^x \bmod p \\ & \quad \text{where } 0 \leq x < \frac{p-1}{2} \\ 1 & \text{if } y = g^x \bmod p \\ & \quad \text{where } \frac{p-1}{2} \leq x < p-1 \end{cases}$.

We want to show that if for p a prime and g a generator of $\mathbf{Z}_p^*$, $\mathrm{EXP}_{p,g}(x) \equiv g^x \bmod p$ is hard to invert, then given $y = \mathrm{EXP}_{p,g}(x)$, $B_{p,g}(y)$ is hard to compute in a very strong sense; that is, in attempting to compute $B_{p,g}(y)$ we can do no better than essentially guessing its value randomly. The proof will be by way of a reduction. It will show that if we can compute $B_{p,g}(y)$ in polynomial time with probability greater than $\frac{1}{2} + \epsilon$ for some non-negligible $\epsilon > 0$ then we can invert $\mathrm{EXP}_{p,g}(x)$ in time polynomial in $|p|$, $|g|$, and ϵ^{-1}. The following is a formal statement of this fact.

Theorem 2.4.4 *Let S be a subset of the prime integers. Suppose there is a polynomial Q and a PTM G such that for all primes $p \in S$ and for all generators g of $\mathbf{Z}_p^*$*

$$\Pr[G(p,g,y) = B_{p,g}(y)] > \frac{1}{2} + \frac{1}{Q(|p|)}$$

(where the probability is taken over $y \in \mathbf{Z}_p^*$ and the coin tosses of G). Then for every polynomial P, there is a PTM I such that for all primes $p \in S$, generators g of $\mathbf{Z}_p^*$, and $y \in \mathbf{Z}_p^*$

$$\Pr[I(p,g,y) = x \text{ such that } y \equiv g^x \bmod p] > 1 - \frac{1}{P(|p|)}$$

(where the probability is taken over the coin tosses of I).

We point to [36] for a proof of the above theorem.

As a corollary we immediately get the following.

Definition 2.4.5 Define $MSB_{p,g}(x) = 0$ if $1 \leq x < \frac{p-1}{2}$ and 1 otherwise for $x \in Z_{p-1}$, and $MSB = \{MSB_{p,g}(x) : Z_{p-1} \to \{0,1\}\}_{(p,g)\in I}$. for $I = \{(p,g) : p \text{ is prime and } g \text{ is a generator of } \mathbf{Z}_p^*\}$.

Corollary 2.4.6 *Under the strong DLA, MSB is a collection of hard-core predicates for EXP.*

It can be shown that actually $O(\log\log p)$ of the most significant bits of $x \in \mathbf{Z}_{p-1}$ are hidden by the function $EXP_{p,g}(x)$. We state this result here without proof.

Theorem 2.4.7 *For a PTM A, let*

$$\alpha = Pr[A(p,g,g^x, x_{\log\log p} x_{\log\log p - 1} \ldots x_0) = 0 \mid x = x_{|p|} \ldots x_0]$$

(where the probability is taken over $x \in \mathbf{Z}_n^*$ and the coin tosses of A) and let

$$\beta = Pr[A(p,g,g^x, r_{\log\log p} r_{\log\log p - 1} \ldots r_0) = 0 \mid r_i \in_R \{0,1\}]$$

(where the probability is taken over $x \in \mathbf{Z}_n^*$, the coin tosses of A, and the bits r_i). Then under the Discrete Logarithm Assumption, we have that for every polynomial Q and every PTM A, $\exists k_0$ such that $\forall k > k_0$, $|\alpha - \beta| < \frac{1}{Q(k)}$.

Corollary 2.4.8 *Under the Discrete Logarithm Assumption we have that for every polynomial Q and every PTM A, $\exists k_0$ such that $\forall k > k_0$ and $\forall k_p < \log\log p$*

$$\Pr[A(p,g,g^x, x_{k_p} \ldots x_0) = x_{k_p+1}] < \frac{1}{2} + \frac{1}{Q(k)}$$

(where the probability is taken over the primes p such that $|p| = k$, the generators g of $\mathbf{Z}_p^*$, $x \in \mathbf{Z}_p^*$, and the coin tosses of A).

For further information on the simultaneous or individual security of the bits associated with the discrete logarithm see [118, 97].

2.4.3 Bit Security of RSA and SQUARING functions

Let $I = \{< n, e > \text{—} \ n = pq \ |p| = |q|, (e, \phi(n)) = 1\}$, and $RSA = \{RSA_{<n,e>} : Z_n^* \to Z_n^*\}_{<n,e>\in I}$ be the collection of functions as defined in 2.2.17.

Alexi, Chor, Goldreich and Schnoor [6] showed that guessing the least significant bit of x from $RSA_{<n,e>}(x)$ better than at random is as hard as inverting RSA.

Theorem 2.4.9 *[6] Let $S \subset I$. Let $c > 0$. If there exists a probabilistic polynomial-time algorithm O such that for $(n,e) \in S$,*

$$prob(O(n,e,x^e \bmod n) = \text{ least significant bit of } x \bmod n) \geq \frac{1}{2} + \frac{1}{k^c}$$

(taken over coin tosses of O and random choices of $x \in Z_n^*$) Then there exists a probabilistic expected polynomial time algorithm A such that for all $n, e \in S$, for all $x \in Z_n^*$, $A(n,e,x^e \bmod n) = x \bmod n$.

Now define $LSB = \{LSB_{<n,e>} : Z_n^* \rightarrow Z_n^*\}_{<n,e>\in I}$ where $LSB_{<n,e>}(x)$ =least significant bit of x.
A direct corollary to the above theorem is.

Corollary 2.4.10 *Under the (strong) RSA assumption, LSB is a collection of hard core predicates for RSA.*

A similary result can be shown for the most signifant bit of x and in fact for the $\log\log n$ least (and most) significant bits of x simultaneously. Moreover, similar results can be shown for the RABIN and BLUM-WILLIAMS collections. We refer to [6], [179] for the detailed results and proofs.

2.5 One-Way and Trapdoor Predicates

A *one-way predicate*, first introduced in [86, 87] is a notion which preceeds hard core predicates for one-way functions and is strongly related to it. It will be very useful for both design of secure encryption and protocol design.

A *one-way predicate* is a boolean function $B : \{0,1\}^* \rightarrow \{0,1\}$ for which

(1) *Sampling is possible*: There exists a PPT algorithm that on input $v \in \{0,1\}$ and 1^k, selects a random x such that $B(x) = v$ and $|x| \leq k$.

(2) *Guessing is hard*: For all $c > 0$, for all k sufficiently large, no PPT algorithm given $x \in \{0,1\}^k$ can compute $B(x)$ with probability greater than $\frac{1}{2} + \frac{1}{k^c}$. (The probability is taken over the random choices made by the adversary and x such that $|x| \leq k$.)

A *trapdoor predicate* is a one-way predicate for which there exists, for every k, trapdoor information t_k whose size is bounded by a polynomial in k and whose knowledge enables the polynomial-time computation of $B(x)$, for all x such that $|x| \leq k$.

Restating as a collection of one-way and trapdoor predicates is easy.

Definition 2.5.1 Let I be a set of indices and for $i \in I$ let D_i be finite. A collection of one-way predicates is a set $B = \{B_i : D_i \rightarrow \{0,1\}\}_{i\in I}$ satisfying the following conditions. Let $D_i^v = \{x \in D_i, B_i(x) = v$.

(1) There exists a polynomial p and a PTM S_1 which on input 1^k finds $i \in I \cap \{0,1\}^k$.

(2) There exists a PTM S_2 which on input $i \in I$ and $v \in \{0,1\}$ finds $x \in D_i$ such that $B_i(x) = v$.

(3) For every PPT A there exists a negligble ν_A such that $\forall$ k large enough

$$\Pr\left[z = v \ : \ i \stackrel{R}{\leftarrow} I \cap \{0,1\}^k \ ; \ v \stackrel{R}{\leftarrow} \{0,1\} \ ; \ x \stackrel{R}{\leftarrow} D_i^v \ ; \ z \stackrel{R}{\leftarrow} A(i,x)\right] \leq \frac{1}{2} + \nu_A(k)$$

Definition 2.5.2 Let I be a set of indices and for $i \in I$ let D_i be finite. A collection of trapdoor predicates is a set $B = \{B_i : D_i \rightarrow \{0,1\}\}_{i\in I}$ satisfying the following conditions. Let $D_i^v = \{x \in D_i, B_i(x) = v$.

(1) There exists a polynomial p and a PTM S_1 which on input 1^k finds pairs (i, t_i) where $i \in I \cap \{0,1\}^k$ and $|t_i| < p(k)$ The information t_i is referred to as the trapdoor of i.

(2) There exists a PTM S_2 which on input $i \in I$ and $v \in \{0,1\}$ finds $x \in D_i$ such that $B_i(x) = v$.

(3) There exists a PTM A_1 such that for $i \in I$ and trapdoor t_i, $x \in D_i$ $A_1(i, t_i, x) = B_i(x)$.

(4) For every PPT A there exists a negligble ν_A such that $\forall$ k large enough

$$\Pr\left[z = v \ : \ i \stackrel{R}{\leftarrow} I \cap \{0,1\}^k \ ; \ v \stackrel{R}{\leftarrow} \{0,1\} \ ; \ x \stackrel{R}{\leftarrow} D_i^v \ ; \ z \leftarrow A(i,x)\right] \leq \frac{1}{2} + \nu_A(k)$$

Note that this definition implies that D_i^0 is roughly the same size as D_i^1.

2.5.1 Examples of Sets of Trapdoor Predicates

A Set of Trapdoor Predicates Based on the Quadratic Residue Assumption

Let Q_n denote the set of all *quadratic residues* (or *squares*) modulo n; that is, $x \in Q_n$ iff there exists a y such that $x \equiv y^2 \bmod n$.

Recall that the Jacobi symbol $(\mathbf{J}_n(x))$ is defined for any $x \in Z_n^*$ and has a value in $\{-1, 1\}$; this value is easily computed by using the law of quadratic reciprocity, even if the factorization of n is unknown. If n is prime then $x \in Q_n \Leftrightarrow (\mathbf{J}_n(x)) = 1$; and if n is composite, $x \in Q_n \Rightarrow (\mathbf{J}_n(x)) = 1$. We let J_n^{+1} denote the set $\{x \mid x \in Z_n^* \wedge (\mathbf{J}_n(x)) = 1\}$, and we let $\tilde{Q}_n$ denote the set of *pseudo-squares* modulo n: those elements of J_n^{+1} which do *not* belong to Q_n. If n is the product of two primes then $|Q_n| = |\tilde{Q}_n|$, and for any pseudo-square y the function $f_y(x) = y \cdot x$ maps Q_n one-to-one onto $\tilde{Q}_n$.

The *quadratic residuousity problem* is: given a composite n and $x \in J_n^{+1}$, to determine whether x is a square or a pseudo-square modulo n. This problem is believed to be computationally difficult, and is the basis for a number of cryptosystems.

The following theorem informally shows for every n, if the quadratic residusosity is hard to compute at all then it is hard to distinguish between squares and non-squares for almost everywhere.

Theorem 2.5.3 *[86, 87]: Let $S \subset \{ns.t.n = pq, p, q,\ primes\}$ If there exists a probabilistic polynomial-time algorithm O such that for $n \in S$,*

$$prob(O(n, x)\ decides\ correctly\ whether\ x \in J_n^{+1}) > \frac{1}{2} + \epsilon\ , \tag{2.1}$$

where this probability is taken over the choice of $x \in J_n^{+1}$ and O's random choices, then there exists a probabilistic algorithm B with running time polynomial in $\epsilon^{-1}, \delta^{-1}$ and $|n|$ such that for all $n \in S$, for all $x \in J_n^{+1}$,

$$prob(B(x, n)\ decides\ correctly\ whether\ x \in Q_n | x \in J_n^{+1}) > 1 - \delta\ , \tag{2.2}$$

where this probability is taken over the random coin tosses of B.

Namely, a probabilistic polynomial-time bounded adversary can not do better (except by a smaller than any polynomial advantage) than guess at random whether $x \in J_n$ is a square mod n, if quadratic residuosity problem is not in polynomial time.

This suggests immediately the following set of predicates: Let

$$QR_{n,z}(x) = \left\{ \begin{array}{l} 0 \text{ if } x \text{ is a square mod } n \\ 1 \text{ if } x \text{ is a non-square mod } n \end{array} \right\}$$

where $QR_{n,z} : J_n^{+1} \rightarrow \{0, 1\}$ and $I_k = \{n\#z \mid n = pq,\ |p| = |q| = \frac{k}{2},\ p$ and q primes, $(\mathbf{J}_n(z)) = +1,\ z$ a non-square mod $n\}$. It is clear that $QR = \{QR_{n,z}\}$ is a set of trapdoor predicates where the trapdoor information associated with every index $< n, z >$ is the factorization $< p, q >$. Lets check this explicitly.

(1) To select pairs (i, t_i) at random, first pick two primes, p and q, of size $\left|\frac{k}{2}\right|$ at random, determining n. Next, search until we find a non-square z in Z_n^* with Jacobi symbol $+1$. The pair we have found is then $(< n, z >, < p, q >)$. We already know how to do all of these operations in expected polynomial time .

(2) Follows from the existence of the following algorithm to select elements out of $D_{n,z}^v$:

- To select $x \in D_{n,z}^0$, let $x = y^2 \bmod n$ where y is an element of Z_n^* chosen at random.
- To select $x \in D_{n,z}^1$, let $x = zy^2 \bmod n$ where y is an element of Z_n^* chosen at random.

(3) To compute $QR_{n,z}(x)$ given $< p, q >$, we compute $(\mathbf{J}_p(x))$ and $(\frac{x}{q})$. If both are $+1$ then $QR_{n,z}(x)$ is 0, otherwise it is 1.

(4) This follows from the Quadaratic Residuosity Assumption and the above theorem.

A Set of Trapdoor Predicates Based on the RSA Assumption

Define $B_{n,e}(x) =$ the least significant bit of $x^d \bmod n$ for $x \in Z_n^*$ where $ed = 1 \bmod \phi(n)$. Then, to select uniformly an $x \in Z_n^*$ such that $B_{n,e}(x) = v$ simply select a $y \in Z_n^*$ whose least significant bit is v and set $x = y^e \bmod n$. Given d it is easy to compute $B_{n,e}(x) =$ least significant bit of $x^d \bmod n$.

The security of this construction follows trivially from the definition of collection of hard core predicates for the RSA collection of functions.

CHAPTER 3

Pseudo-random bit generators

In this chapter, we discuss the notion of pseudo-random generators. Intuitively, a $PSRG$ is a deterministic program used to generate *long* sequence of bits which *looks like random sequence*, given as input a *short random sequence* (the input seed).
The notion of $PSRG$ has applications in various fields:

Cryptography:
: In the case of private key encryption, Shannon showed (see lecture 1) that the length of the clear text should not exceed the length of the secret key, that is, the two parties have to agree on a very long string to be used as the secret key. Using a $PSRG$ G, they need to agree only on a short seed r, and exchange the message $G(r) \bigoplus m$.

Algorithms Design:
: An algorithm that uses a source of random bits, can manage with a shorter string, used as a seed to a $PSRG$.

Complexity Theory:
: Given a probabilistic algorithm, an important question is whether we can make it deterministic. Using the notion of a $PSRG$ we can prove, assuming the existence of one-way function that $BPP \subseteq \cap_\epsilon DTIME(2^{n^\epsilon})$

In this chapter we will define good pseudo random number generators and give constructions of them under the assumption that one way functions exist.

We first ask where do can we actually find **truly** random bit sequences. [1]

3.0.2 Generating Truly Random bit Sequences

Generating a one-time pad (or, for that matter, any cryptographic key) requires the use of a "natural" source of random bits, such as a coin, a radioactive source or a noise diode. Such sources are absolutely essential for providing the initial secret keys for cryptographic systems.

However, many natural sources of random bits may be defective in that they produce *biased* output bits (so that the probability of a one is different than the probability of a zero), or bits which are *correlated* with each other. Fortunately, one can remedy these defects by suitably processing the output sequences produced by the natural sources.

To turn a source which supplies biased but uncorrelated bits into one which supplies unbiased uncorrelated bits, von Neumann proposed grouping the bits into pairs, and then turning 01 pairs into 0's, 10 pairs into

[1] some of the preliminary discussion in the following three subsections is taken from Rivest's survey article on cryptography which appears in the handbook of computer science

1's, and discarding pairs of the form 00 and 11 [182]. The result is an unbiased uncorrelated source, since the 01 and 10 pairs will have an identical probability of occurring. Elias [69] generalizes this idea to achieve an output rate near the source entropy.

Handling a correlated bit source is more difficult. Blum [34] shows how to produce unbiased uncorrelated bits from a biased correlated source which produces output bits according to a known finite Markov chain.

For a source whose correlation is more complicated, Santha and Vazirani [159] propose modeling it as a *slightly random source*, where each output bit is produced by a coin flip, but where an adversary is allowed to choose *which* coin will be flipped, from among all coins whose probability of yielding "Heads" is between δ and $1-\delta$. (Here δ is a small fixed positive quantity.) This is an extremely pessimistic view of the possible correlation; nonetheless U. Vazirani [180] shows that if one has *two, independent*, slightly-random sources X and Y then one can produce "almost independent" ϵ-biased bits by breaking the outputs of X and Y into blocks $\mathbf{x}, \mathbf{y}$ of length $k = \Omega(1/\delta^2 \log(1/\delta) \log(1/\epsilon))$ bits each, and for each pair of blocks $\mathbf{x}, \mathbf{y}$ producing as output the bit $\mathbf{x} \cdot \mathbf{y}$ (the inner product of $\mathbf{x}$ and $\mathbf{y}$ over $GF(2)$). This is a rather practical and elegant solution. Chor and Goldreich [50] generalize these results, showing how to produce independent ϵ-biased bits from even worse sources, where some output bits can even be completely determined.

These results provide effective means for generating truly random sequences of bits—an essential requirement for cryptography—from somewhat defective natural sources of random bits.

3.0.3 Generating Pseudo-Random Bit or Number Sequences

The one-time pad is generally impractical because of the large amount of key that must be stored. In practice, one prefers to store only a short random key, from which a long pad can be produced with a suitable cryptographic operator. Such an operator, which can take a short *random* sequence x and deterministically "expand" it into a *pseudo-random* sequence y, is called a *pseudo-random sequence generator*. Usually x is called the *seed* for the generator. The sequence y is called "pseudo-random" rather than random since not all sequences y are possible outputs; the number of possible y's is at most the number of possible seeds. Nonetheless, the intent is that for all practical purposes y should be indistinguishable from a truly random sequence of the same length.

It is important to note that the use of pseudo-random sequence generator reduces *but does not eliminate* the need for a natural source of random bits; the pseudo-random sequence generator is a "randomness expander", but it must be given a truly random seed to begin with.

To achieve a satisfactory level of cryptographic security when used in a one-time pad scheme, the output of the pseudo-random sequence generator must have the property that an adversary who has seen a portion of the generator's output y must remain unable to efficiently predict other unseen bits of y. For example, note that an adversary who knows the ciphertext C can guess a portion of y by correctly guessing the corresponding portion of the message M, such as a standardized closing "Sincerely yours,". We would not like him thereby to be able to efficiently read other portions of M, which he could do if he could efficiently predict other bits of y. Most importantly, the adversary should not be able to efficiently infer the seed x from the knowledge of some bits of y.

How can one construct secure pseudo-random sequence generators?

Classical Pseudo-random Generators are Unsuitable

Classical techniques for pseudo-random number generation [107, Chapter 3] which are quite useful and effective for Monte Carlo simulations are typically unsuitable for cryptographic applications. For example, *linear* feedback shift registers [93] are well-known to be cryptographically insecure; one can solve for the feedback pattern given a small number of output bits.

Linear congruential random number generators are also insecure. These generators use the recurrence

$$X_{i+1} = aX_i + b \pmod{m} \tag{3.1}$$

to generate an output sequence $\{X_0, X_1, \ldots\}$ from secret parameters a, b, and m, and starting point X_0. It is possible to infer the secret parameters given just a few of the X_i [139]. Even if only a fraction of the bits of each X_i are revealed, but a, b, and m are known, Frieze, Håstad, Kannan, Lagarias, and Shamir show

how to determine the seed X_0 (and thus the entire sequence) using the marvelous *lattice basis reduction* (or "L^3") algorithm of Lenstra, Lenstra, and Lovász [76, 114].

As a final example of the cryptographic unsuitability of classical methods, Kannan, Lenstra, and Lovasz [105] use the L^3 algorithm to show that the binary expansion of any algebraic number y (such as $\sqrt{5} = 10.001111000110111\ldots$) is insecure, since an adversary can identify y exactly from a sufficient number of bits, and then extrapolate y's expansion.

3.0.4 Provably Secure Pseudo-Random Generators: Brief overview

This section provides a brief overview of the history of the modern history of pseudo random bit generators. Subsequent section define these concepts formally and give constructions.

The first pseudo-random sequence generator proposed for which one can prove that it is impossible to predict the next number in the sequence from the previous numbers assuming that it is infeasible to invert the RSA function is due to Shamir [166]. However, this scheme generates a sequence of *numbers* rather than a sequence of *bits*, and the security proof shows that an adversary is unable to predict the next *number*, given the previous numbers output. This is not strong enough to prove that, when used in a one-time pad scheme, each *bit* of the message will be well-protected.

Blum and Micali [36] introduced the first method for designing provably secure pseudo-random *bit* sequence generators, based on the use of one-way predicates. They call a pseudo-random bit generator secure if an adversary cannot guess the next bit in the sequence from the prefix of the sequence, better than guessing at random. Blum and Micali then proposed a particular generator based on the difficulty of computing discrete logarithms. Blum, Blum, and Shub [31] propose another generator, called the *squaring generator*, which is simpler to implement and is provably secure assuming that the quadratic residuosity problem is hard. Alexi, Chor, Goldreich, and Schnorr [6] show that the assumption that the quadratic residuosity problem is hard can be replaced by the weaker assumption that factoring is hard. A related generator is obtained by using the RSA function. Kaliski shows how to extend these methods so that the security of the generator depends on the difficulty of computing elliptic logarithms; his techniques also generalize to other groups [102, 103]. Yao [184] shows that the pseudo-random generators defined above are *perfect* in the sense that no probabilistic polynomial-time algorithm can guess with probability greater by a non-negligible amount than $1/2$ whether an input string of length k was randomly selected from $\{0,1\}^k$ or whether it was produced by one of the above generators. One can rephrase this to say that a generator that passes the next-bit test is perfect in the sense that it will *pass all polynomial-time statistical tests*. The Blum-Micali and Blum-Blum-Shub generators, together with the proof of Yao, represent a major achievement in the development of provably secure cryptosystems. Impagliazzo, Luby, Levin and Håstad show that actually the existence of a one-way function is equivalent to the existence of a pseudo random bit generator which passes all polynomial time statistical tests.

3.1 Definitions

Definition 3.1.1 Let X_n, Y_n be probability distributions on $\{0,1\}^n$ (That is, by $t \in X_n$ we mean that $t \in \{0,1\}^n$ and it is selected according to the distribution X_n). We say that $\{X_n\}$ is **poly-time indistinguishable** from $\{Y_n\}$ if $\forall PTM\ A$, $\forall$ polynomial Q, $\exists n_0$, s.t $\forall n > n_0$,

$$|\Pr_{t\in X_n}(A(t)=1) - \Pr_{t\in Y_n}(A(t)=1)| < \frac{1}{Q(n)}$$

i.e., for sufficiently long strings, no PTM can tell whether the string was sampled according to X_n or according to Y_n.

Intuitively, Pseudo random distribution would be a indistinguishable from the uniform distribution. We denote the uniform probability distribution on $\{0,1\}^n$ by U_n. That is, for every $\alpha \in \{0,1\}^n$, $\Pr_{x\in U_n}[x = \alpha] = \frac{1}{2^n}$.

Definition 3.1.2 We say that $\{X_n\}$ is **pseudo random** if it is poly-time indistinguishable from $\{U_n\}$.

That is, $\forall PTM\ A$, $\forall$ polynomial Q, $\exists n_0$, such that $\forall n > n_0$,

$$|\Pr_{t \in X_n}[A(t) = 1] - \Pr_{t \in U_n}[A(t) = 1]| < \frac{1}{Q(n)}$$

Comments:

The algorithm A used in the above definition is called a polynomial time *statistical test.* (Knuth, vol. 2, suggests various examples of statistical tests). It is important to note that such a definition cannot make sense for a single string, as it can be drawn from either distribution.

If $\exists A$ and Q such that the condition in definition 2 is violated, we say that X_n fails the test A.

Definition 3.1.3 A polynomial time deterministic program $G : \{0,1\}^k \rightarrow \{0,1\}^{\hat{k}}$ is a pseudo-random generator (PSRG) if the following conditions are satisfied.

1. $\hat{k} > k$
2. $\{G_{\hat{k}}\}_{\hat{k}}$ is pseudo-random where $G_{\hat{k}}$ is the distribution on $\{0,1\}^{\hat{k}}$ obtained as follows: to get $t \in G_{\hat{k}}$:

   pick $x \in U_k$

   set $t = G(x)$

That is, $\forall PTM A, \forall$ polynomial $Q, \forall$ sufficiently large k,

$$|\Pr_{t \in G_{\hat{k}}}(A(t) = 1) - \Pr_{t \in U_{\hat{k}}}(A(t) = 1)| < \frac{1}{Q(\hat{k})} \tag{3.2}$$

3.2 The Existence Of A Pseudo-Random Generator

Next we prove the existence of $PSRG$'s, if length-preserving one way permutations exist. It has been shown that if one-way functions exist (without requiring them to be length-preserving permutations) then one-way functions exist, but we will not show this here.

Theorem 3.2.1 *Let $f : \{0,1\}^* \rightarrow \{0,1\}^*$ be a length preserving one-way permutation. Then*

1. *$\exists\, PSRG \quad G : \{0,1\}^k \rightarrow \{0,1\}^{k+1}$ (such G is called an extender).*
2. *$\forall$ polynomial Q, $\exists\, PSRG \quad G^Q : \{0,1\}^k \rightarrow \{0,1\}^{Q(k)}$.*

Proof:
Proof of 1: Let f be as above. Let B be the *hard core bit* of f. (that is, B is a boolean predicate, $B : \{0,1\}^* \rightarrow \{0,1\}$, s.t it is efficient to compute $B(x)$ given x, but given only $f(x)$, it is hard to compute $B(x)$ with probability greater than $\frac{1}{2} + \epsilon$ for non-negligible ϵ.) Recall that we showed last class that every OWF $f(x)$ can be converted into $f_1(x,r)$ for which $B'(x,r) = \sum_1^{|x|} x_i r_i \bmod 2$ is a hard core bit. For notational ease, assume that B is already a hard-core bit for f.

Define

$$G^1(x) = f(x) \circ B(x)$$

($\circ$ denotes the string concatenation operation). We will prove that $G^1(x)$ has the required properties. Clearly, G^1 is computed in poly-time, and for $|x| = k$, $|G^1(x)| = k + 1$. It remains to show that the distribution $\{G^1_{k+1}\}$ is pseudo random.

Intuition : Indeed, knowing $f(x)$ should not help us to predict $B(x)$. As f is a permutation, $f(U_k)$ is uniform on $\{0,1\}^k$, and any separation between U_{k+1} and G_{k+1} must caused by the hard core bit. We would like to show that any such separation would enable us to predict $B(x)$ given only $f(x)$ and obtain a contradiction to B being a hard core bit for f.

We proceed by contradiction: Assume that (G is not good) $\exists$ statistical test A, polynomial Q s.t

$$\Pr_{t\in G_{k+1}}(A(t)=1) - \Pr_{t\in U_{k+1}}(A(t)=1) > \frac{1}{Q(k+1)}$$

(Note that we have dropped the absolute value from the inequality 3.2. This can be done wlog. We will later see what would change if the the other direction of the inequality were true).

Intuitively, we may thus interpret that if A answers 1 on a string t it is more likely that t is drawn from distribution G_{k+1}, and if A answers 0 on string t that it is more likely that t is drawn from distribution U_{k+1}.

We note that the probability that $A(f(x)\circ b)$ returns 1, is the sum of the weighted probability that A returns 1 conditioned on the case that $B(x)=b$ and conditioned on the case $B(x)=1$. By, the assumed separation above, we get that it is more likely that $A(f(x)\circ b)$ will return 1 when $b=B(x)$. This easily translates to an algorithm for predicting the hard core bit of $f(x)$.

Formally, we have

$$\begin{aligned}\Pr_{x\in U_k, b\in U_1}[A(f(x)\circ b)=1] &= \Pr[A(f(x)\circ b)=1 \mid b=B(x)]\cdot\Pr[b=B(x)] \\ &\quad + \Pr[A(f(x)\circ b)=1 \mid b=\overline{B(x)}]\cdot\Pr[b=\overline{B(x)}] \\ &= \tfrac{1}{2}(\alpha+\beta)\end{aligned}$$

where $\alpha=\Pr[A(f(x)\circ b)=1 \mid b=B(x)]$ and $\beta=\Pr[A(f(x)\circ b)=1 \mid b=\overline{B(x)}]$.

From the assumption we therefore get

$$\begin{aligned}\Pr_{x\in U_k}[A(f(x)\circ B(x))=1] - \Pr_{x\in U_k}[A(f(x)\circ b)=1] &= \alpha - \tfrac{1}{2}(\alpha+\beta) \\ &= \tfrac{1}{2}(\alpha-\beta) \\ &> \tfrac{1}{Q(k)}.\end{aligned}$$

We now exhibit a polynomial time algorithm A' that on input $f(x)$ computes $B(x)$ with success probability significantly better than 1/2.
A' takes as input $f(x)$ and outputs either a 0 or 1.

1. choose $b\in\{0,1\}$
2. run $A(f(x)\circ b)$
3. If $A(f(x)\circ b)=1$, output b, otherwise output $\overline{b}$.

Notice that, when dropping the absolute value from the inequality 3.2, if we take the second direction we just need to replace b by $\overline{b}$ in the definition of A'.

Claim 3.2.2 $\Pr[A'(f(x)=B(x))] > \frac{1}{2}+\frac{1}{Q(k)}$.

Proof:

$$\begin{aligned}\Pr[A'(f(x)=b)] &\ \Pr[A(f(x)\circ b)=1 \mid b=B(x)]\Pr[b=B(x)) \\ &\ \Pr[A(f(x)\circ b)=0 | b=\overline{B(x)})\Pr[b=\overline{B(x)}] \\ &\quad +\tfrac{1}{2}\alpha+\tfrac{1}{2}(1-\beta) \\ &= \tfrac{1}{2}+\tfrac{1}{2}(\alpha-\beta) \\ &> \tfrac{1}{2}+\tfrac{1}{Q(k)}.\end{aligned}$$

∎

This contradicts the hardness of computing $B(x)$. It follows that G^1 is indeed a $PSRG$.
Proof of 2*:* Given a $PSRG$ G that expands random strings of length k to pseudo-random strings of length $k+1$, we need to show that, $\forall$ polynomial Q, $\exists\, PSRG\, G^Q : \{0,1\} \rightarrow \{0,1\}^{Q(k)}$. We define G^Q by first using G $Q(k)$ times as follows:

$$\begin{array}{lll} x & \rightarrow G \rightarrow & f(x) \circ B(x) \\ f(x) \circ B(x) & \rightarrow G \rightarrow & f(f(x)) \circ B(f(x)) \\ f^2(x) \circ B(f(x)) & \rightarrow G \rightarrow & f^3(x) \circ B(f^2(x)) \\ & \bullet & \\ & \bullet & \\ & \bullet & \\ f^{Q(k)-2}(x) \circ B(f^{Q(k)-1}(x)) & \rightarrow G \rightarrow & f^{Q(k)}(x) \circ B(f^{Q(k)-1}(x)) \end{array}$$

The output of $G^Q(x)$ is the concatenation of the last bit from each string *i.e.*,

$$G^Q(x) = B(x) \circ B(f(x)) \circ \cdots \circ B(f^{Q(k)-1}(x) = b_1^G(x) \circ b_2^G(x) \circ \cdots \circ b_{Q(|x|)}^G(x)$$

Clearly, G^Q is poly-time and it satisfies the length requirements. We need to prove that the distribution generated by G^Q, $G^Q(U_k)$, is poly-time indistinguishable from $U_{Q(k)}$. We proceed by contradiction, (and show that it implies that G is not $PSRG$)
If G_{Qk} is not poly-time indistinguishable from $U_{Q(k)}$, $\exists$ statistical test A, and $\exists$ polynomial P , s.t.

$$\Pr_{t \in G_{Q(k)}}(A(t) = 1) - \Pr_{t \in U_{Q(k)}}(A(t) = 1) > \frac{1}{P(k)}$$

(As before we omit the absolute value). We now define a sequence $D_1, D_2, ..., D_{Q(k)}$ of distributions on $\{0,1\}^{Q(k)}$, s.t. D_1 is uniform (i.e. strings are random), $D_{Q(k)} = G_{Q(k)}$, and the intermediate D_i's are distributions composed of concatenation of random followed by pseudorandom distributions. Specifically,

$$\begin{array}{lll} t \in D_1 \text{ is obtained by letting} & t = s & \text{where } s \in U_{Q(k)} \\ t \in D_2 \text{ is ontained by letting} & t = s \circ B(x) & \text{where } s \in U_{Q(k)-1}, x \in U_k \\ t \in D_3 \text{ is ontained by letting} & t = s \circ B(x) \circ B(f(x)) & \text{where } s \in U_{Q(k)-2}, x \in U_k \\ & \bullet & \\ & \bullet & \\ & \bullet & \\ t \in D_{Q(k)} \text{ is ontained by letting} & t = B(x)... \circ B(f^{Q(k)-1}(x)) & \text{where } x \in U_k \end{array}$$

Since the sequence 'moves' from $D_1 = U_{Q(k)}$ to $D_{Q(k)} = G_{Q(k)}$, and we have an algorithms A that distinguishes between them, there must be two successive distributions between which A distinguishes.
i.e. $\exists\, i$, $1 \leq i \leq Q(k)$, s.t.

$$\Pr_{t \in D_i}(A(t) = 1) - \Pr_{t \in D_{i+1}}(A(t) = 1) > \frac{1}{P(k)Q(k)}$$

We now present a poly-time algorithm A' that distinguishes between U_{k+1} and G_{k+1}, with success probability significantly better than $\frac{1}{2}$, contradicting the fact that G is a $PSRG$.

A' works as follows on input $\alpha = \alpha_1\alpha_2...\alpha_{k+1} = \alpha' \circ b$

1. Choose $1 \leq i \leq q(k)$ at random.
2. Let
$$t = \gamma_1 \circ ... \circ \gamma_{Q(k)-i-1} \circ b \circ b_1^G(\alpha') \circ b_2^G(\alpha') \circ ... \circ b_i^G(\alpha')$$
where the γ_j are chosen randomly.
(Note that $t \in D_i$ if $\alpha' \circ b \in U_{k+1}$, and that $t \in D_{i+1}$ if $\alpha' \circ b \in G_{k+1}$.)

3. We now run $A(t)$. If we get 1, A' returns 0 If we get 0, A' returns 1

   (i.e if A returns 1, it is interpreted as a vote for D_i and therefore for $b \neq B(\alpha`)$ and $\alpha \in U_{k+1}$. On the other hand, if A returns 0, it is interpreted as a vote for D_{i+1} and therefore for $b = B(\alpha')$ and $\alpha \in G_{k+1}$.)

It is immediate that:

$$\Pr_{\alpha \in U_{k+1}}(A'(\alpha) = 1) - \Pr_{\alpha \in G_{k+1}}(A'(\alpha) = 1) > \frac{1}{P(k)Q^2(k)}$$

The extra $\frac{1}{Q(k)}$ factor comes form the random choice of i. This violates the fact that G was a pseudo random generator as we proved in part 1. This is a contradiction

∎

3.3 Next Bit Tests

If a pseudo-random bit sequence generator has the property that it is difficult to predict the next bit from previous ones with accuracy greater than $\frac{1}{2}$ by a non-negligible amount in time polynomial in the size of the seed, then we say that the generator *passes the "next-bit" test.*

Definition 3.3.1 A *next bit test* is a special kind of statistical test which takes as input a prefix of a sequence and outputs a prediction of the next bit.

Definition 3.3.2 A (discrete) probability distribution on a set S is a function $D : S \rightarrow [0,1] \subset \mathbf{R}$ so that $\sum_{s \in S} D(s) = 1$. For brevity, probability distributions on $\{0,1\}^k$ will be subscripted with a k. The notation $x \in X_n$ means that x is chosen so that $\forall z \in \{0,1\}^n Pr[x = z] = X_n(z)$. In what follows, U_n is the uniform distribution.

Recall the definition of a pseudo-random number generator:

Definition 3.3.3 A *pseudo-random number generator (PSRG)* is a polynomial time deterministic algorithm so that:

1. if $|x| = k$ then $|G(x)| = \hat{k}$
2. $\hat{k} > k$,
3. $G_{\hat{k}}$ is pseudo-random[2], where $G_{\hat{k}}$ is the probability distribution induced by G on $\{0,1\}^{\hat{k}}$.

Definition 3.3.4 We say that a pseudo-random generator *passes the next bit test* A if for every polynomial Q there exists ,an integer k_0 such that for all $\hat{k} > k_0$ and $p < \hat{k}$

$$\Pr_{t \in G_{\hat{k}}}[A(t_1 t_2 \ldots t_p) = t_{p+1}] < \frac{1}{2} + \frac{1}{Q(k)}$$

Theorem 3.3.5 *G passes all next bit tests $\Leftrightarrow$ G passes all statistical tests.*

Proof:

($\Leftarrow$) Trivial.

($\Rightarrow$) Suppose, for contradiction, that G passes all next bit test but fails some statistical test A. We will use A to construct a next bit test A' which G fails. Define an operator $\odot$ on probability distributions

[2] A pseudo-random distribution is one which is polynomial time indistinguishable from $U_{\hat{k}}$

so that $[X_n \odot Y_m](z) = X_n(z_n) \cdot Y_m(z_m)$ where $z = z_n \circ z_m, |z_n| = n, |z_m| = m$ ($\circ$ is concatenation). For $j \leq \hat{k}$ let $G_{j,\hat{k}}$ be the probability distribution induced by $G_{\hat{k}}$ on $\{0,1\}^j$ by taking prefixes. (That is $G_{j,\hat{k}}(x) = \sum_{z \in \{0,1\}^{\hat{k}}, z \text{ extends } x} G_{\hat{k}}(z)$.)

Define a sequence of distributions $H_i = G_{i,\hat{k}} \odot U_{\hat{k}-i}$ on $\{0,1\}^{\hat{k}}$ of "increasing pseudo-randomness." Then $H_0 = U_{\hat{k}}$ and $H_{\hat{k}} = G_{\hat{k}}$. Because G fails A, A can differentiate between $U_{\hat{k}} = H_0$ and $G_{\hat{k}} = H_{\hat{k}}$; that is, $\exists Q \in \mathcal{Q}[x]$ so that $|Pr_{t \in H_0}[A(t) = 1] - Pr_{t \in H_{\hat{k}}}[A(t) = 1]| > \frac{1}{Q(k)}$. We may assume without loss of generality that $A(t) = 1$ more often when t is chosen from $U_{\hat{k}}$ (otherwise we invert the output of A) so that we may drop the absolute value markers on the left hand side. Then $\exists i, 0 \leq i \leq \hat{k} - 1$ so that $Pr_{t \in H_i}[A(t) = 1] - Pr_{t \in H_{i+1}}[A(t) = 1] > \frac{1}{\hat{k}Q(k)}$.

The next bit test A' takes $t_1 t_2 \ldots t_i$ and outputs a guess for t_{i+1}. A' first constructs

$$s_0 = t_1 t_2 \ldots t_i 0 r_{i+2} r_{i+3} \ldots r_{\hat{k}}$$

$$s_1 = t_1 t_2 \ldots t_i 1 \hat{r}_{i+2} \hat{r}_{i+3} \ldots \hat{r}_{\hat{k}}$$

where r_j and $\hat{r}_j$ are random bits for $i + 2 \leq j \leq \hat{k}$. A' then computes $A(s_0)$ and $A(s_1)$.

If $A(s_0) = A(s_1)$, then A' outputs a random bit.

If $0 = A(s_0) = \overline{A(s_1)}$, then A' outputs 0.

If $1 = A(s_0) = \overline{A(s_1)}$, then A' outputs 1.

Claim 3.3.6 By analysis similar to that done in the previous lecture, $Pr[A'(t_1 t_2 \ldots t_i) = t_{i+1}] > \frac{1}{2} + \frac{1}{\hat{k}Q(k)}$.

Thus we reach a contradiction: A' is a next bit test that G fails, which contradicts our assumption that G passes all next bit tests.

∎

3.4 Examples of Pseudo-Random Generators

Each of the one way functions we have discussed induces a pseudo-random generator. Listed below are these generators (including the Blum/Blum/Shub generator which will be discussed afterwards) and their associated costs. See [36, 31, 151].

Name	One way function	Cost of computing one way function	Cost of computing j^{th} bit of generator
RSA	$x^e \bmod n, n = pq$	k^3	jk^3
Rabin	$x^2 \bmod n, n = pq$	k^2	jk^2
Blum/Micali	$\text{EXP}(p, g, x)$	k^3	jk^3
Blum/Blum/Shub	(see below)	k^2	$\max(k^2 \log j, k^3)$

3.4.1 Blum/Blum/Shub Pseudo-Random Generator

The Blum/Blum/Shub pseudo-random generator uses the (proposed) one way function $g_n(x) = x^2 \bmod n$ where $n = pq$ for primes p and q so that $p \equiv q \equiv 3 \bmod 4$. In this case, the squaring endomorphism $x \mapsto x^2$ on $\mathbf{Z}_n^*$ restricts to an isomorphism on $(\mathbf{Z}_n^*)^2$, so g_n is a permutation on $(\mathbf{Z}_n^*)^2$. (Recall that every square has a unique square root which is itself a square.)

Claim 3.4.1 The least significant bit of x is a hard bit for the one way function g_n.

The j^{th} bit of the Blum/Blum/Shub generator may be computed in the following way:

$$B(x^{2^j} \bmod n) = B(x^{\alpha} \bmod m)$$

where $\alpha \equiv 2^j \bmod \phi(n)$. If the factors of n are known, then $\phi(n) = (p-1)(q-1)$ may be computed so that α may be computed prior to the exponentiation. $\alpha = 2^j \bmod \phi(n)$ may be computed in $O(k^2 \log j)$ time and x^{α} may be be computed in k^3 time so that the computation of $B(x^{2^j})$ takes $O(\max(k^3, k^2 \log j))$ time.

An interesting feature of the Blum/Blum/Shub generator is that if the factorization of n is known, the $2^{\sqrt{n}\,th}$ bit can be generated in time polynomial in $|n|$. The following question can be raised: let $G^{BBS}(x,p,q) = B(f^{2^{\sqrt{n}}}(x)) \circ \ldots \circ B(f^{2^{\sqrt{n}}+2k}(x))$ for $n = pq$ and $|x| = k$. Let G^{BBS}_{2k} be the distribution induced by G^{BBS} on $\{0,1\}^{2k}$.

Open Problem 3.4.2 Is this distribution G^{BBS}_{2k} pseudo-random? Namely, can you prove that

$$\forall Q \in \mathbf{Q}[x], \forall PTM\ A, \exists k_0, \forall k > k_0 | Pr_{t \in G^{BBS}_{2k}}[A(t) = 1] - Pr_{t \in U_{2k}}[A(t) = 1]| < \frac{1}{Q(2k)}$$

The previous proof that G is pseudo-random doesn't work here because in this case the factorization of n is part of the seed so no contradiction will be reached concerning the difficulty of factoring.

More generally,

Open Problem 3.4.3 Pseudo-random generators, given seed x, implicitly define an infinite string $g^x_1 g^x_2 \ldots$. Find a pseudo-random generator so that the distribution created by restricting to any polynomially selected subset of bits of g^x is pseudo-random. By polynomially selected we mean examined by a polynomial time machine which can see g^x_i upon request for a polynomial number of i's (the machine must write down the i's, restricting $|i|$ to be polynomial in $|x|$).

CHAPTER 4

Pseudo-random function families

Pseudorandom function families are one of the fundamental primitives of cryptographic protocol design. Most importantly, they provide a basis for private key cryptography.

As compared to one-way functions, pseudorandom function families are a more "high level" primitive. They have very strong properties, and it is this that makes them useful.

Below we will discuss both what are pseudorandom function families and various candidate examples for them.

4.1 Random functions

The starting point is to look at *random functions*. Intuitively, these are the ideal objects which pseudorandom functions will be trying to approximate. Before saying why we look at them, lets see what they are.

Let X, Y be sets. (X might be infinite, Y is finite.) Let $R(X, Y)$ denote the set of all functions mapping X to Y. A random function is a function chosen randomly from this set. That is, all functions are equally likely.

Typically it will be the case that $X = \{0,1\}^l$ and $Y = \{0,1\}^L$ for some integers $l, L > 0$, and this is assumed in what follows. In such a case we often write $R(l, L)$ for $R(\{0,1\}^l, \{0,1\}^L)$. When l, L (or X, Y) are evident from the context we drop them and simply write R.

A function is described by a table giving its output value for each input. Thus, a function mapping X to Y can be thought of as a table having $|X|$ entries, each entry storing an element of Y. Thus, a random function is a large object: $L2^l$ bits long.

Notice that the set R is even bigger: $|R(X, Y)| = |Y|^{|X|} = 2^{L2^l}$.

There are many ways to think about random functions. The most useful is the following. It is a box which given any input $x \in X$ returns a random number, except that if you give it an input you already gave it before, it returns the same thing as last time. It is this "dynamic" view that we suggest the reader have in mind in thinking about random functions.

The dynamic view can be thought of as following program. The program maintains a table of entries of the form (x, y).

Input: x
If an entry (x, y) exists in table return y and exit
Else flip coins to determine a string $y \in Y$ and return it
Also put (x, y) into the table.

The answers on any point is random and independent of the answers on other points.

Another way to think about a random function is as a large, pre-determined random table. The entries are of the form (x, y). For each x someone has flipped coins to determine y and put it into the table.

We are more used to the idea of picking points at random. Here we are picking a *function* at random.

Let's do some simple probabilistic computations to understand random functions. Let $f\colon \{0,1\}^l \to \{0,1\}^L$ be chosen at random. Then:

- Fix, $x \in \{0,1\}^l$ and $y \in \{0,1\}^L$. Then:

$$\Pr[f(x) = y] = 2^{-L} .$$

  Notice it doesn't depend on l. Also it doesn't depend on the values of x, y.
- Fix, $x_1, x_2 \in \{0,1\}^l$ and $y \in \{0,1\}^L$. Then:

$$\Pr[f(x_1) = f(x_2) = y] = \begin{cases} 2^{-2L} & \text{if } x_1 \neq x_2 \\ 2^{-L} & \text{if } x_1 = x_2 \end{cases}$$

  This illustrates *independence*.
- Fix, $x_1, x_2 \in \{0,1\}^l$ and $y \in \{0,1\}^L$. Then:

$$\Pr[f(x_1) \oplus f(x_2) = y] = \begin{cases} 2^{-L} & \text{if } x_1 \neq x_2 \\ 0 & \text{if } x_1 = x_2 \text{ and } y \neq 0^L \\ 1 & \text{if } x_1 = x_2 \text{ and } y = 0^L \end{cases}$$

  Similar things hold for the sum of more than two things.

4.2 Function Families

We begin with finite families and then go on to collections of them.

4.2.1 Finite families

A *finite function family* is a collection F of functions having the same domain and range. A typical example is domain $\{0,1\}^l$ and range $\{0,1\}^L$ for some integers l, L. In such a case, we say that the *input length* is l, while the *output length* is L. Typically, there is an underlying probability distribution on the space F according to which we would pick functions at random. We don't make this explicit in the notation; it will be specified in the context.

Notation: $f \xleftarrow{R} F$ denotes the operation of selecting a function at random according to the underlying distribution on F.

Example 4.2.1 Recall $R(l, L)$ be the set of all functions of $\{0,1\}^l$ to $\{0,1\}^L$. The probability distribution on $R(l, L)$ is the uniform one; ie. all functions are equally likely to be chosen. Thus $f \xleftarrow{R} R(l, L)$ is just the operation of selecting a random function of l bits to L bits.

Definition 4.2.2 A *keyed family* is one where there is some set of keys, namely the set of all strings of some length k, and each key $K \in \{0,1\}^k$ specifies a function $F_K\colon \{0,1\}^l \to \{0,1\}^L$. The underlying distribution is that induced by a random choice of key. Namely, $f \xleftarrow{R} F$ means $K \xleftarrow{R} \{0,1\}^k$ and $f \leftarrow F_K$.

The number of functions in the family is at most 2^k (not exactly this since two different keys might specify the same function).

Example 4.2.3 Block ciphers yield one example of such families, and a particular example in this category is $F = \text{DES}$, where $k = 56$ and $l = L = 64$. Another example is the compression function of MD5, viewed as keyed in its first argument; here $k = L = 128$ and $l = 512$. (All lengths are in bits).

4.2.2 Infinite families

Above, F is a finite object, consisting of functions of l to L. For example, $l = L = 64$. There is no function applying, say, to 128 bit inputs.

Sometimes we want to consider families of finite families. A family of functions is a collection $F^1, F^2, F^3, \ldots$, written $\{F^k\}_{k \geq 1}$. Each F^i is a finite family of functions. Here k is a security parameter. The number l, L now turn into functions of k, and for each k, the finite family F^k has input length $l(k)$ and output length $L(k)$. Finally, the family is keyed if there is a function $\kappa(k)$ which yields the key length of F^k.

We will sometimes discuss things in terms of finite families, sometimes infinite ones. The difference is largely a question of what we construct.

4.3 Motivation for PRFs

Before providing the definition of pseudorandom function families let us talk about the motivation and applications.

In private key cryptography, the parties A and B share a key K which the adversary doesn't know. They want to use this key to achieve various things. In particular, to encrypt and authenticate the data they send to each other.

A key is (or ought to be) a short string. Suppose however that we allow the parties a very long shared string. The form it takes is a random function f of l bits to L bits, for some pre-specified l, L. This is called the shared random function model.

The shared random function model cannot really be realized because, as we saw, random functions are just too big to even store. It is a conceptual model.

To work in this model, we give the parties oracle access to f. They may write down $x \in \{0,1\}^l$ and in one step be returned $f(x)$.

Now it turns out that the shared random function model is a very convenient model in which to think about cryptography, formulate schemes, and analyze them. In particular, we will see many examples where we design schemes in the shared random function model and prove them secure. This is true for a variety of problems, but most importantly for encryption and message authentication. The proof of security here is absolute: we do not make any restrictions on the computational power of the adversary, but are able to simply provide an upper bound on the success probability of the adversary.

But now what? We have schemes which are secure but a priori can't be efficiently realized, since they rely on random functions. That's where pseudorandom function families come in. A PRF family is a family F of functions indexed by small keys (eg. 56 or 128 bits). However, it has the property that if K is shared between A and B, and we use F_K in place of a random function f in some scheme designed in the shared random function model, the resulting scheme is still secure as long as the adversary is computationally bounded.

In other words, what we want is that PRFs can be used in place of random functions in shared key schemes. Pseudorandom functions are designed to make this happen for as wide a range of applications as possible. A pseudorandom function is specified by a short key K, and the parties need only store this key. Then, they use this function in place of the random function in the scheme. And things should work out, in the sense that if the scheme was secure when a random function was used, it should still be secure.

This is a very rough idea. Technically, it is not always true: this is the intuition. Pseudorandom functions don't always work. That is, you can't substitute them for random functions in any usage of the latter and expect things to work out. But if used right, it works out in a large number of cases. How do we identify these cases? We have to provide a formal definition of a pseudorandom function family and see how to test whether, in a particular usage of random functions, it is OK to substitute pseudorandom ones. In this context we stress one important point straightaway. Pseudorandom functions rely on the key K being *secret*. The adversary is not given K and cannot directly compute the function. (Of course it might gain some information about values of F_K on various points via the usage of F_K by the legitimate parties, but that will be OK.) In other words, you can substitute *shared, secret* random functions by PRFs, but not public ones.

There is another (related) motivation for the model. Sometimes, we are given schemes, defined and used in practice. They might make use of a block cipher like DES. How can we assess the quality of these schemes? A reasonable thing to do is assume DES is a family of pseudorandom functions, and analyze the

scheme assuming this. If we can prove the security of the scheme given this assumption we would take it as an indication the scheme is good.

Pseudorandom functions are an intriguing notion and a powerful tool. Try to appreciate the strength and utility of the claim. Whether you want to design a scheme for encryption, authentication, or some other purpose, design it in the shared random function model. Then simply substitute the random function with a pseudorandom one, and things should still be secure!

Before going to the definitions, we note that in some cases we will want permutations, not functions, and also want that the parties, given f, also have f^{-1}. We will discuss such cases as they arise.

4.4 Definition of Pseudorandom function families

As we discussed above, what we want is functions which can be used in place of random functions in cryptographic schemes like MACs. We will have a keyed family F of functions. The parties will share a key K specifying a function $f = F_K$, and then use this function in place of the random function in the scheme. What we want is that this "works out." Namely, the scheme should still be secure.

The notion of pseudorandom function families was designed to make this happen. It considers usages of a random function in which the description of the function is hidden and the function is only accessed in a black box way. Roughly, a pseudo-random function has the property that if you see input-output examples of the function, they look like input-output examples of a random function.

We begin with finite families of pseudorandom functions and go on to infinite ones. The central notion is that of a distinguisher.

4.4.1 Distinguishers

In fact pseudorandomness is a property of a family, not a particular function. Let F be some finite family of functions of input length l and output length L. (Think of F as a keyed family, like a block cipher. In particular, it is not a very large family, compared to $R(l, L)$.) For succinctness, let $R = R(l, L)$. Now, consider two worlds, in each of which you are given a box containing a function g, as follows:

World 0: You are given a box for a function g drawn at random from R, namely $g \stackrel{R}{\leftarrow} R$. (So g is just a random function of l bits to L bits.)

World 1: You are given a box for a function g drawn at random from F, namely $g \stackrel{R}{\leftarrow} F$. (If F is keyed via keys of length K, this means someone chose a key K at random, let $g = F_K$, and put g in the box. But they did NOT give you K!)

Now someone puts you in one of these worlds and asks you which one you are in. To find out, you are allowed to query the function g on points of your choice. Based on its input-output behavior, you must eventually make some decision as to which world you are in. Note that the only way you are allowed to access the box is by giving it an input x and obtaining output $g(x)$; you can't otherwise "look inside" the box.

The quality of pseudo-random family F can be thought of as measured by the difficulty of telling, in the above game, whether a function belongs in World 0 or World 1.

Intuitively, the game just models some way of "using" the function g in an application like a MAC scheme. If it is not possible to distinguish the input-output behavior of a random member of F from a truly random function, the application should behave in roughly the same way whether it uses a function from F or a random function. Later we will see exactly how this works out; for now let us continue to develop the notion. But right away one warning should be issued. Pseudorandom functions can't be substituted for random functions in *all* usages of random functions. To make sure it is OK in a particular application, you have to make sure that it falls within the realm of applications for which the formal definition below can be applied.

Example 4.4.1 If $f(0) = 0$ for all f in a family F, then F is not a pseudorandom function family. This is because, were this property true, we could easily tell whether we were in world 0 or world 1. Query the given function g at 0. If we get back 0 we predict we are in world 1, else in world 0. The probability of being wrong is the probability that $g(0) = 0$ if g is random, and this is at most 2^{-L}.

The act of trying to tell which world you are in is formalized via the notion of a *distinguisher*. This is an algorithm which is provided oracle access to a function g and tries to decide if g is random or pseudorandom. (Ie. whether it is in world 0 or world 1.) A distinguisher can only interact with the function by giving it inputs and examining the outputs for those inputs; it cannot examine the function directly in any way. Intuitively, a family is pseudorandom if the probability that the distinguisher says 0 is roughly the same regardless of which world it is in. To capture this mathematically, we define two probabilities:

$$P_D(F) = \Pr\left[D^g = 1 : g \stackrel{R}{\leftarrow} F\right]$$

$$P_D(R) = \Pr\left[D^g = 1 : g \stackrel{R}{\leftarrow} R\right]$$

We call the difference between them the "advantage" of D.

Definition 4.4.2 The advantage of distinguisher D is $\mathsf{Adv}_D(F) = |P_D(F) - P_D(R)|$.

If the advantage is small, it means that distinguisher D cannot tell the difference between the "worlds." We expect that the advantage grows as the distinguisher invests more effort (more time, more queries) in the process. We can't expect it to be impossible for *any* distinguisher to tell the two worlds apart, but then neither is this what we need. Conforming to a general theme in cryptographic definitions, we only care what happens when the adversary (here the distinguisher) is limited in the amount of resources it can bring to the problem. This is because in reality, adversaries are limited; for example, they can't do 2^{80} steps of computation.

4.4.2 Finite PRFs

Informally, F is a finite pseudo-random function (PRF) family if any "efficient" distinguisher D has only a "small" advantage. To be more precise, we quantify the various parameters. Efficiency of the distinguisher is captured by two quantities. One is its running time, denoted t.[1] Another is the number of queries it makes of g, namely the number of input-output examples it sees. We denote this by q. Finally we let ϵ be (a bound on) the advantage. We expect that a distinguisher that works harder will earn a greater advantage, so that the advantage will grow as t and q grow. Accordingly, the definition of security is in terms of the advantage achievable in a given t, q.

Definition 4.4.3 F is a (t, q, ϵ)-secure finite PRF family if any distinguisher D who makes at most q queries and run in time at most t has advantage $\mathsf{Adv}_D(F) \leq \epsilon$.

The strength of this definition lies in the fact that it does not specify anything about the kinds of strategies that can be used by a distinguisher; it only limits its time and number of queries. A distinguisher can use whatever means desired to distinguish the function as long as it stays within the specified resource bounds. Nonetheless, it should fail, except possibly with probability ϵ.

If we have in hand a particular family of functions F that we believe to be pseudorandom, we will make various assumptions about how good it is. For example, that it is (t, q, ϵ)-secure for $t = 2^{65}$, $q = 2^{45}$ and $\epsilon = 2^{-10}$.

Remark 4.4.4 There is one feature of the above parameterization about which everyone asks. Typically, F is a keyed family. Let k be the length of the key. Obviously, the key length is a fundamental determinant of security: larger key length will typically mean more security. Yet, the key length k appears nowhere in the above: the parameters are t, q, ϵ. Why is this? The truth is that the key length itself does not matter: what matters is just the advantage ϵ the adversary obtains. This ϵ is of course a function of k, but it is ϵ that matters; k matters only to the extent that it influences ϵ. Typically, we get our estimates of ϵ based on k (and other things), but once we have them, it is only ϵ that counts.

[1] There is a technical point here. The running time of D^g might depend on whether g was from F or from R. Which do we measure? The convention is to measure the former, because if the distinguisher is to be run in practice that's what we would get. Furthermore the convention is to count the time to do the computations of f in the running time of D, again because in practice it has to do them. In particular, we let T_F denote the time to compute $f(x)$ given x and $f \in F$, a quantity we will make use of later.

Remark 4.4.5 Notice that the definition of a finite PRF family F does not make any explicit requirement on how fast it is to compute $F_K(x)$ given K and x; it only talks about security. Of course, in practice it is important that the computation of the function is fast and that the key K is short.

4.4.3 Infinite PRFs

Let $\mathcal{F} = \{F^k\}_{k \geq 1}$ be an infinite collection of functions with input length $l(\cdot)$ and output length $L(\cdot)$. We say it is polynomial time computable if there is an algorithm which given k, K and x outputs $F_K^k(x)$ in time poly(k). To define security, we now consider a family $\{D^k\}_{k \geq 1}$ of distinguishers. Let $R^k = R(l(k), L(k))$.

Definition 4.4.6 The advantage of distinguisher family $\mathcal{D} = \{D^k\}_{k \geq 1}$ is the function $\mathsf{Adv}_{\mathcal{D}}(\mathcal{F}, k) = |P_{D^k}(F^k) - P_{D^k}(R^k)|$.

We say that $\mathcal{D}$ is polynomial time if in its experiment it halts after poly(k) steps.

Definition 4.4.7 $\mathcal{F} = \{F^k\}_{k \geq 1}$ is an infinite family of pseudorandom functions if it is polynomial time computable and also $\mathsf{Adv}_{\mathcal{D}}(\mathcal{F}, k)$ is negligible (as a function of k) for every polynomial time distinguisher family $\mathcal{D}$.

Notice that this time the definition insists that the functions themselves can be efficiently computed.

4.5 Usage of PRFs: Examples and intuition

Let us now try to get some feel for how to work with PRF families.

4.5.1 Some simple examples

Let F be a (t, q, ϵ)-secure finite PRF family with input length l and output length L, and assume t, q are quite large. Then:

Example 4.5.1 Fix $x \in \{0,1\}^l$ and $y \in \{0,1\}^L$. We claim that

$$\frac{1}{2^L} - \epsilon \;\leq\; \Pr\left[f(x) = y \;:\; f \stackrel{R}{\leftarrow} F\right] \;\leq\; \frac{1}{2^L} + \epsilon \;. \tag{4.1}$$

Why is this true? The intuition is that to see what is the probability of some event involving f chosen randomly from F, imagine that it was chosen randomly from R, and compute the probability in this case. Then the probability in the case of $f \stackrel{R}{\leftarrow} F$ will differ from this amount by at most ϵ. Otherwise, it would be possible to tell the two worlds apart. In this case, we note that

$$\Pr\left[f(x) = y \;:\; f \stackrel{R}{\leftarrow} R\right] \;=\; \frac{1}{2^L} \;, \tag{4.2}$$

so the probability in Equation (4.1) is within ϵ of this amount.

That this is true requires a proof, and we will now proceed to provide one. But first we note that what is happening here is not going to *always* be true. Namely, not every event true for a random function happens with probability only ϵ different for a pseudo-random function. Only certain kinds of events. Which kinds? The ones, intuitively, which can be "measured" by a distinguisher. In other words, ones for which a proof like the following works.

Let us now proceed to formally prove that Equation (4.1) is correct. We will only prove the upper bound; the lower bound is an exercise. How do we set up the proof? There are not too many choices; like all these proofs in cryptography it is a reductionist argument that proceeds by contradiction. We assume the claim is false. Now we will show that this contradicts the assumption that F is (t, q, ϵ)-secure.

The claim being false means (recall we are focusing on the upper bound only) that

$$\Pr\left[f(x) = y \;:\; f \stackrel{R}{\leftarrow} F\right] \;>\; \frac{1}{2^L} + \epsilon \;.$$

But we know, from Equation (4.2), that if f were random, the probability would be exactly 2^{-L}. This means we have an event whose probability is different depending on which world we are in. But that means we have a means of telling which world we are in: just see whether the event happens, and, if so, bet on world 1 (ie. f is pseudorandom) because the event is more likely here; if not, bet on world 0 because the event is less likely here.

Formally, define a distinguisher D who, given an oracle for a function f, operates as follows. It queries f at x to get back $z = f(x)$. Then it checks if $z = y$. If so, it outputs 1, else it outputs 0. Now we see that

$$\begin{aligned} P_D(F) &> 2^{-L} + \epsilon \\ P_D(R) &= 2^{-L} . \end{aligned}$$

The first is true by assumption, the second is a fact. This means that $\mathsf{Adv}_D(F) = |P_D(F) - P_D(R)| > \epsilon$. Now, we note also that D used only one query and very little time, so that we may safely assume the time used is less than t and the number of queries used is less than q, because t, q were assumed to be large numbers. But, if so, we have a contradiction. Using very little resources, our distinguisher has achieved an advantage that exceeds ϵ. But if F was (t, q, ϵ)-secure, this is not possible. So our original assumption must have been false.

It is left to the reader to check that the lower bound works out the same way.

Example 4.5.2 Assume $L \geq 6$. We claim that

$$\frac{1}{2^6} - \epsilon \;\leq\; \Pr\left[\text{The first 6 bits of } f(0^l) \oplus f(1^l) \text{ are all 0} \;:\; f \xleftarrow{R} F\right] \;\leq\; \frac{1}{2^6} + \epsilon \; . \tag{4.3}$$

Again, the reason this is true is similar. Forget about F, and look at the probability of the event being true when $f \xleftarrow{R} R$. It is a simple exercise that

$$\Pr\left[\text{The first 6 bits of } f(0^l) \oplus f(1^l) \text{ are all 0} \;:\; f \xleftarrow{R} R\right] \;=\; \frac{1}{2^6} \; .$$

Now, to see that Equation (4.3) is true, go through a contradiction argument similar to the one in the previous example. This time, the constructed distinguisher will make two queries, namely 0^l and 1^l.

4.5.2 Substituting random functions by PRFs

The above examples, although simple, indicate roughly what we expect will happen when PRFs are substituted for random functions. The probability of an event changes by an amount proportional to the quality (security) of the PRF family. What we will see is that when we have some cryptographic scheme using a (shared, secret) random function, we will be able to view the success of the adversary as just such an event. When the PRF replaces the random function, the probability of this event does not grow too much, meaning if the adversary had small probability of success in the random case, this probability is still not too much. Put another way, if the probability of success of the adversary changes by too much, we would have a distinguisher.

Of course we must be careful to not blindly assume this is true no matter what the scheme. We must each time carefully go through a constructive argument (proof) like in the above examples. After a few of these however, you will get the feel and be able to tell, by looking at a scheme, whether or not the argument is likely to go through.

4.6 Constructions of PRF families

Where can we find PRF families? There are a variety of ways. We can build pseudorandom function families out of pseudorandom bit generators or one-way functions, a conservative but to date inefficient approach. In practice we might be willing to assume that block ciphers like the DES have the property, and under this approach can get efficient but proven secure schemes for private key cryptography. Let's examine both possibilities.

4.6.1 PRFs from PRBGs: trees and synthesizers

The notion of a pseudorandom bit generator (PRBG) was discussed in Chapter 3. Recall it is a polynomial time computable function G which takes a k bit seed and produces a $p(k) > k$ bit sequence of bits that look random to any efficient test.

The first construction of PRF families was from PRBGs which are length doubling: the output length is twice the input length.

Theorem 4.6.1 [82] *Given a length-doubling pseudorandom bit generator we can construct an infinite family of pseudorandom functions.*

The construction, called the binary tree construction, is like this. The function G induces a tree of functions G_z in the following way:

- Define $G_0(x) \circ G_1(x) = G(x)$ where $k = |G_0(x)| = |G_1(x)|$.
- Define $G_{z\circ 0}(x) \circ G_{z\circ 1}(x) = G_z(x)$ where $k = |G_{z\circ 0}| = |G_{z\circ 1}|$.

Then $f_i(x)$ is defined in terms of the binary tree induced by G as follows: $\forall x f_i(x) = G_x(i)$. We now let $\mathcal{F} = \{F^k\}_{k\geq 1}$ where F^k is $\{f_i : \{0,1\}^k \rightarrow \{0,1\}^k || i| = k\}$. It is shown in [82] that this is secure.

Another construction based on a primitive called synthesizers was given by Naor and Reingold [128]. This yields a PRBG based construction which is more parallelizable than the binary tree based one.

4.6.2 PRFs from one-way functions

We saw before that we can construct PRBGs from one-way functions [98, 96]. It follows from the above that we can build (infinite) PRF families from one-way functions. Furthermore, one can see that given any pseudorandom function family one can construct a one-way function [99]. Thus we have the following.

Theorem 4.6.2 *There exist (infinite) families of pseudorandom functions if and only if there exists a one-way function.*

This is quite a strong statement. One-way functions are a seemingly weak primitive, a priori quite unrelated to PRFs. Yet the one can be transformed into the other. Unfortunately the construction is not efficient enough to be practical.

4.6.3 Block ciphers as finite PRFs

We could get more efficient candidates for PRF families by a more direct route, by turning to block ciphers. Under this approach, we would assume that a block cipher like DES is a (finite) PRF family. Thus $F = \text{DES}$, $l = L = 64$, and the key length is $\kappa = 56$. Note that by the nature of the cipher we have a finite PRF family: DES is defined only for these fixed input lengths and not for others.

How good is DES as a PRF? That is, for what values of t, q, ϵ may we reasonably assume it is (t, q, ϵ)-secure? Certainly we know that q must be less than 2^{43} since otherwise DES can be broken by linear cryptanalysis. But in fact we cannot even allow q to be close to this high. This is because DES is actually a family of *permutations* on 64 bits. Namely, each DES_K map is a permutation of $\{0,1\}^{64}$. Now recall that the quality of a PRF family is measured by how well a distinguisher can distinguish it from a random function. A distinguisher can distinguish a random member of the DES family from a random function with reasonably probability in just 2^{32} queries by a birthday attack. Given an oracle g it would ask for the value of g on some 2^{32} distinct points, and if the returned values were all distinct would bet that g was from the DES family, else that g was random. A standard calculation shows that this distinguisher succeeds quite often. Thus, about the best we could do is assume DES is a finite PRF family which is (t, q, ϵ)-secure with, say, $q \approx 2^{25}$ and $\epsilon \approx 2^{-4}$.

This is not very good. There are various variations one could make to improve it. One is to drop the last 16 bits of the DES output, yielding a function returning only 48 bits. The other is to do various transformations to overcome the birthday attack.

But the truth is DES is feeling its age. Our view is that new block ciphers are needed. The kind of cipher we feel is now adequate is one with a key of at least 80 bits, and $l = L = 128$ or more. Furthermore, we suggest the cipher be designed to be a random function, not a permutation! This "future" cipher is the kind we will assume when we talk of block ciphers in what follows.

4.6.4 Extending the domain size

Suppose we start with a finite PRF family with input length l. Often we need to extend the functions to a larger domain in such a way that they are still PRFs on the larger domain. (One of the important reasons for this is that pseudorandom functions make good message authentication codes. See Theorem 7.5.1.) There are various ways to do this. One is to use the CBC (cipher block chaining) construction.

Here we are given a (t, q, ϵ)-secure finite PRF family with input length l and output length also l. An integer n is fixed and we want to construct a family of functions mapping $\{0,1\}^{ln}$ to $\{0,1\}^l$. The construction is like this. The keys for the new family are the same as for the old one. Let $f = F_K$ be the l bit to l bit function indicated by key K. The new function, given $M_1 \ldots M_n$, does the following:

- Set $Y_0 = 0^l$
- Set $Y_1 = f(M_1 \oplus Y_0)$
- Set $Y_2 = f(M_2 \oplus Y_1)$
- $\vdots$
- Set $Y_n = f(M_n \oplus Y_{n-1})$
- Output Y_n.

Let F^n denote the family of functions in which the function indexed by K maps $\{0,1\}^{nl}$ to $\{0,1\}^l$ and is given by the CBC using $f = F_K$.

Theorem 4.6.3 [15] *Suppose F is a (t, q, ϵ)-secure family of PRFs. Then F^n is itself a family of PRFs, with input length nl and output length l, and is (t', q', ϵ')-secure, where*

$$\begin{aligned} t' &= t - O(nql) \\ q' &= \frac{q}{n} \\ \epsilon' &= \epsilon + \frac{3(q')^2 n^2}{2^l} = \epsilon + \frac{3q^2}{2^l} . \end{aligned}$$

There are also other constructions. For example, a recent one is the cascade construction of [12].

Similarly (or simultaneously) we may want to extend the output length. It turns out this is easier, so we won't discuss it in detail.

4.7 Some applications of PRFs

4.7.1 Cryptographically Strong Hashing

Let P_1, P_2 be polynomials so that $\forall x, P_1(x) > P_2(x)$. Define $F^{P_1,P_2} = \{f : \{0,1\}^{P_1(k)} \to \{0,1\}^{P_1(k)}\}$. Then we wish to hash names into address where $|\text{Name}| = P_1(k)$ and $|\text{Address}| = P_2(k)$. We may use pseudo-random functions to hash these names so that Address $= f_i(\text{Name})$.

Claim 4.7.1 If there exist one way functions, then for all polynomials P, and for all integers k sufficiently large, the previous hashing algorithm admits no more than $O(\frac{1}{2\sqrt{\text{Address}}}) + \frac{1}{P(k)}$ collisions even if, after fixing the scheme, the names are chosen by an adversary with access to previous (Name, Address) pairs.

4.7.2 Prediction

A *prediction test* $T(1^k)$

1. queries an oracle for $f \in F_k$, discovering $(x_1, f(x_1)), \ldots, (x_l, f(x_l))$,
2. outputs an "exam", x, and
3. is given y so that with probability $\frac{1}{2}$, $y = f(x)$ (otherwise, y is chosen randomly in $\{0,1\}^{|f(x)|} - \{f(x)\}$).
4. outputs 1 if it guesses that $y = f(x)$, 0 otherwise.

F is said to *pass the prediction test* T if $\forall Q \in \mathbf{Q}[x], \exists k_0, \forall k > k_0$,

$$Pr[T(1^k) \text{ guesses correctly given } y \text{ in step 3}] < \frac{1}{2} + \frac{1}{Q(k)}$$

The above pseudo-random functions then pass all prediction tests (assuming there exist one way functions).

4.7.3 Learning

Define a *concept space* S and a *concept* $C \subseteq S$. A learner is exposed to a number of pairs $(e_i, \pm_i)$ where $e_i \in S$ and $\pm_i = + \Leftrightarrow e_i \in C$. The learner is then requested to determine if a given $e \in S$ is an element of C.

The above pseudo-random function show that if there exist one way functions, then there exist concepts not learnable in polynomial time. (The concept in this case would be $\{x, f(x)\} \subseteq \{x, y\}$.)

4.7.4 Identify Friend or Foe

Consider the situation of two forces of war planes fighting an air battle. Each plane wishes to identify potential targets as friendly or enemy. This can be done using pseudo-random functions in the following way:

1. All the planes on a certain force know i.
2. To identify a target, a plane sends the target a random number r and expects to receive back $f_i(r)$ if the target is a friend.

Then, even though the enemy planes see many pairs of the form $(x, f(x))$, they cannot compute $f(y)$ for y they have not yet seen.

4.7.5 Private-Key Encryption

Let A and B privately agree on i. Then to encrypt message m, A produces a random string r and sends $(r, f_i(r) \oplus m)$. B can compute $f_i(r)$ and so compute $f_i(r) \oplus m \oplus f_i(r) = m$. Assuming that there exist one way functions, such a system is secure again chosen ciphertext attack, that is, secure even if the adversary can compute $(r, f_i(r))$ for a collection of r's. See Chapter 5 for more on this.

4.8 History and discussion

The basic notion of pseudorandom functions is due to Goldreich, Goldwasser and Micali [82]. In particular these authors introduced the important notion of distinguishers.

What [82] formalized is what we have here called infinite pseudorandom function families. They were interested in the complexity theoretic setting.

The notion of finite pseudorandom function families, and the quantization of security in terms of (t, q, ϵ)-security, was introduced by [15]. While the basic technical idea is the same (namely that of distinguishing

tests), the motivation, and hence the formalization, are different. The idea of [15] is to model block ciphers via PRF families. Accordingly, input and output lengths are fixed. The provided family consists of maps of l bits to L bits (eg. $l = L = 64$ for DES) and there is no directly provided function taking more bits of input, as is true for DES. Furthermore, in the quantitative treatment of security, not only is time quantified, but queries are separately quantified because these resources have different costs; this is a distinction not made in the complexity theoretic works.

When finite PRFs are instantiated with block ciphers, the approach yields *efficient* PRFs. However, we pay a price for the efficiency, namely the finiteness. It is harder to design schemes with finite PRFs than with infinite ones. Nonetheless, to get efficient schemes, we do start with finite PRFs. We will see later how to get provably secure but efficient private key encryption and message authentication schemes from them.

One should note that underlying the "asymptotic" or complexity theoretic results are concrete constructions, and their security can be analyzed exactly by delving into the proofs. However in most cases this has not been done.

Whenever possible, we will try to be clear whether we are talking of finite or infinite PRF families. But we might forget, and just talk of pseudorandom function families.

CHAPTER 5

Private key encryption

There are two main components to securing communication between users across a network: privacy and authenticity. Here we consider achieving privacy in the private key, or symmetric setting, where the communicating parties share a key which the adversary doesn't know.

5.1 Private key encryption schemes

Parties A and B share a key K and communicate across a line on which the adversary E can eavesdrop. That is, any communication sent from one party to the other is seen by the adversary. The adversary wants to know what the parties are saying to each other. It wants to read the data.

We stress that the adversary is assumed not to be active. That is we assume it won't insert messages on the line, or modify the sent messages. Not because this is not possible: in practice of course we must protect against these kinds of attacks too. But because protecting against these attacks is a different problem. That is the goal of message integrity, and we will address it in depth later. However, we do allow that the adversary influence, or even choose, the data to be encrypted. Think of this as a worst case choice of the legitimate parties.

In this context, the message that A wants to send to B is often called a *plaintext*. To protect the confidentiality of her plaintext M, the parties will use an encryption scheme.

An encryption scheme is specified via an encryption function (algorithm) $\mathcal{E}$ and a decryption function (algorithm) $\mathcal{D}$. The first may be probabilistic, the second may too, but usually is not. (Probabilistic means that the algorithm tosses coins, anew each time it is invoked.) Both algorithms take as one of their inputs the secret key K shared between the parties. The algorithms are used as follows. When the sender wants to send a message M privately to the receiver, she computes $C = \mathcal{E}(K, M)$. (We write this also as $\mathcal{E}_K(M)$). This C is called the *ciphertext*. It is this ciphertext she transmits, not the M. The receiver however can recover the plaintext from the ciphertext. Upon receiving C, the receiver computes $\mathcal{D}(K, C)$. (We write this also as $\mathcal{D}_K(C)$). We require that this is M. To summarize:

Definition 5.1.1 An encryption scheme is a pair $(\mathcal{E}, \mathcal{D})$ of probabilistic, polynomial time algorithms together with an indication of the key length κ. The first algorithm is called the encryption algorithm and the second is called the decryption algorithm. We require that if $C = \mathcal{E}_K(M)$ then $M = \mathcal{D}_K(C)$.

This is the simplest case. In addition, either algorithm may depend on state information (such as a counter) or an explicit security parameter. (When no explicit security parameter is present, the length of the shared key functions as the security parameter.)

Notice that above we have only specified the algorithmic functionality of an encryption scheme. We have not said what is a *secure* encryption scheme. For example, $\mathcal{E}_K(M) = M$ is an encryption scheme meeting

the above conditions. It is indeed an encryption scheme; just not a secure one. Below we turn to the issue of what is a secure encryption scheme.

5.2 Some issues in security

The first thing we will do is try to build up some intuition about what a secure encryption scheme should be, towards eventually distilling a notion and formal definition of security.

A first cut at a notion of security is to say that what we want is: given ciphertext C, the adversary E can't recover the plaintext M. But actually, this isn't good enough. The reason is that E might be able to figure out *partial information*. For example, even though she might not be able to recover M, the adversary might, given C, be able to recover the first bit of M, or the sum of all the bits of M. This is not good, because these bits might carry valuable information.

For a concrete example, say I am communicating to my broker a message which is a sequence of "buy" or "sell" decisions for a pre-specified sequence of stocks. That is, we have certain stocks, numbered 1 through l, and bit i of the message is 1 if I want to buy stock i and 0 otherwise. The message is sent encrypted. But if the first bit leaks, the adversary knows whether I want to buy or sell stock 1, which may be something I definitely don't want to reveal. If the sum of the bits leaks, the adversary knows how many stocks I am buying.

Granted, this might not be a problem at all if the data was in a different format. However, making assumptions, or requirements, on how users format data, or how they use it, is a bad and dangerous approach to secure protocol design. It is an important principle of our approach that the encryption scheme should yield security no matter what is the format of the data. That is, we don't want people to have to worry about how they format their data: it should be secure regardless.

In other words, as designers of security protocols, we cannot make assumptions about data content or formats. Our protocols must protect any data, no matter how formatted. We view it as the job of the protocol designer to ensure this is true. And we want schemes that are secure in the strongest possible natural sense.

So what is the best we could hope for? It is useful to make a thought experiment. What would an "ideal" encryption be like? Well, it would be as though some angel took the message M from the sender and delivered it to the receiver, in some magic way. The adversary would see nothing at all. Intuitively, our goal is to "approximate" this as best as possible. We would like encryption to have the properties of ideal encryption. In particular, no partial information would leak. Of course, our setting will be "computational security." The security will only hold with respect to adversaries of limited computing power. If the adversary works harder, she can figure out more, but a "feasible" amount of effort yields no noticeable information.

Now let's try to come up with some schemes and see these issues more concretely.

5.3 Example schemes and their analysis

We look at various ways to encrypt, and analyze the security. We will use this to gather intuition about how to define security.

First we discuss primitives which are used to construction encryption schemes.

5.3.1 Primitives for encryption

Various primitives can be used for encryption. The one we focus on most to begin with is pseudorandom function families. (Later we will get to using pseudorandom bit generators and one-way functions.) This primitive was introduced in Chapter 4. Let us here explain what kinds of families we are talking about, how we think about them and how they will be used.

Let F be a finite family of functions with input length l, output length L and key length κ. This means that every κ-bit key K indexes a function F_K from the family and this function maps from $\{0,1\}^l$ to $\{0,1\}^L$. Associated to the family is a probability distribution on functions; choosing a random function of the family means choosing a random κ bit string K and choosing the function F_K indexed by this string.

Notice that here we have made the input and output lengths parameters of the scheme. This model is motivated by block ciphers, and in particular the DES cipher, which has the above form. (For DES, $l = L = 64$ and $\kappa = 56$.) Chapter 4 discusses the notion of pseudorandom function families and how we might assume block ciphers are such families.

Let K be the shared key between the users in our encryption scheme. What they will do is use this to specify a function F_K of the family, and use this function, somehow, to encrypt. We will denote F_K by f.

Accordingly, for each encryption method we study, we actually get a family of different encryption schemes, one for each choice of the underlying function family F. In practice, we are interested in the case where F is some concrete block cipher. That, however, is not the way to begin thinking about it. Instead, what we will do is consider ideal ciphers. What are these ideal ciphers? By this we mean random functions, or, in some cases, random permutations. In the first case, F is simply the family of all functions of l bits to L bits, and selecting a function at random from the family just yields a random function of l bits to L bits. In the second case, F would be the set of all permutations of l bits to L bits. In the terminology of Chapter 4, this is the *shared random function* model.

Thus, think below that f is chosen at random from the space of all the 2^{L2^l} functions of l bits to L bits. Perhaps an easier way to think of it is as f as assigning a random L-bit value to each l-bit string x. For *distinct* strings x, x', the assigned values are independent. Thus, for example, for any distinct l-bit strings x and x', and any L-bit strings y, y' we have

$$\begin{aligned} \Pr_f [f(x) = y] &= 2^{-L} \\ \Pr_f [f(x) = y \text{ and } f(x') = y'] &= 2^{-2L} \end{aligned}$$

In order to make sense of the parties being given a random function, it is thought of as provided as an oracle. This means the two parties are each given a black box for f. They can write a string x and in one step are returned $f(x)$. It is important to note that the parties have the same box, and that the adversary is not given this box. In some cases when f is a permutation we may want to give the parties oracle access to f^{-1} as well, and do so.

Let us briefly summarize the (important) motivation for this shared random function model. On the one hand, random functions are a convenient tool for scheme design and analysis in private key cryptography. We shall see this clearly in what follows. Thus, we work assuming we have them. The problem we then face is, however, that in practice these objects are totally unrealistic to realize. As a key, a random function is just too large! (The key K would have to be $L2^l$ bits long.) However, in practice we would use a family of pseudorandom functions F. When the scheme is implemented, the parties simply share a k-bit key K and use F_K in place of the random function f. The key point is that *this is secure.* We will see that substituting shared random functions by pseudorandom functions preserves security if done properly. Accordingly, we can work with random functions and preserve all the analytical advantages of the latter model. In particular plugging in DES_K in place of f yields DES based schemes about whose security we can argue under the assumption that DES is a finite pseudorandom function family.

We stress one crucial fact. The random function here is *secret.* It is shared between A and B. (It is the shared key.) The adversary is not directly given access to it, and the adversary's knowledge of it accrues only from what she can gather from its usage by the parties in the process of encryption. This is an important restriction in the eventual ability to replace the random function with a pseudorandom one. In particular, the pseudorandom function will be keyed by the shared secret key K, and the security relies on the secrecy of this key. We cannot replace a *public* random function with a pseudorandom one and hope that a scheme using it retains its security.

The materiel in Chapter 4 is thus crucial to the subject of private key cryptography that we are now aborting. The section of the current chapter you are now reading (and in particular the examples that follow) serves actually as a motivation and introduction to the subject of random and pseudorandom functions, but to understand the results we will later quote about the schemes that follow, it is important to understand Chapter 4, and suggest it be read in conjunction with the materiel you are reading now.

5.3.2 Encrypting a single block message

As indicated above, we have fixed a finite family of functions F and let $f = F_K$ be the function indicated by the shared key K between A and B, and we are thinking of f as random. Let us begin by assuming that the message we need to encrypt is just one l-bit block, and also assume $l = L$.

Example 5.3.1 How about setting $\mathcal{E}_f(M) = f(M)$. It is pretty good at hiding information; the outcome is just a random value. But it is not even decryptable.

This explains one of the motivations for making DES a permutation. It is to enable recovery.

Example 5.3.2 Let's assume we have a *permutation* f and its inverse f^{-1}, the pair of which form the key. Now we set $\mathcal{E}_{f,f^{-1}}(M) = f(M)$ and $\mathcal{D}_{f,f^{-1}}(C) = M$.

This looks pretty good. But I claim there is still one problem. Suppose I have just two messages, 0^l for "buy" and 1^l for "sell." I keep sending data, but always one of these two. What happens? The adversary sees which are the same. That is, it might see that the first two are the same and equal to the third, etc.

In a secure encryption scheme, it should not be possible to co-relate ciphertexts of different messages in such a way that information is leaked.

This has a somewhat dramatic implication. Namely, *encryption must be probabilistic* or *depend on state information*. If not, you can always tell if the same message was sent twice. Each encryption must use fresh coin tosses, or, say, a counter, and a the encryption of a particular message will be different each time. In terms of our setup it means $\mathcal{E}$ is a *probabilistic* or *stateful* algorithm, using coin tosses, which are fresh each time it is invoked.

The reason this is dramatic is that it goes in many ways against the historical or popular notion of encryption. Encryption is thought of as a code, a fixed mapping of plaintexts to ciphertexts. This is no longer true. A single plaintext will have many possible ciphertexts. Yet, it should be possible to decrypt! How can we set this up?

Example 5.3.3 Let's return to having just a plain old (random) function f, not a permutation, and encrypt via the following:

$\mathcal{E}_f(M)$
Pick a random l bit string r
Return $(r, f(r) \oplus M)$

Note you can decrypt! The receiver gets (r, Y), and sets $M = f(r) \oplus Y$.

This is probabilistic in the choice of r. There are many possible encryptions of M.

Why is it secure? Roughly, it is just like a one-time pad, which, as we indicated in Chapter 1, has perfect security. Assume we encrypt a sequence of messages $M^1, M^2, \ldots, M^q$. Then as long as the r values are distinct, things are OK.

One observation: Notice we did not need a permutation! In fact, we decided that we can encrypt better without a permutation than with.

Not just due to the new scheme: note that if we used a permutation in the above it would be worse than using a function: can you see why, ie. spot the slight weakness that would arise? Say we get ten distinct values of r in ten successive encryptions, namely the adversary sees $(r^1, Y^1), \ldots, (r^{10}, Y^{10})$ with $r^1, \ldots, r^{10}$ all distinct. Also suppose for some reason the adversary knows $M^1, \ldots, M^9$, and hence $f(r^1), \ldots, f(r^9)$. So it knows $f(r^{10})$ is not one of these. This rules out nine possibilities for M^{10}. Nine is not much, but this can grow.

We reach the perhaps surprising conclusion that making DES a permutation may have been a mistake. Encryption is better done with functions.

Note you could use a counter in place of the random number above.

5.3.3 Modes of DES: ECB, CBC

Now let's go on to another issue. Above we are assuming messages are $l = L$ bits long, where L is the output length of f. What would we do if the message is longer, say $M = M_1 \ldots M_n$, with each block being l bits long?

DES encryption has to solve this problem. Let's look at what they do, except that we'll use a random function (or permutation) instead of DES itself (it can only be better!). The first mode of DES we consider is ECB: Electronic codebook mode. It is essentially a substitution cipher.

ECB (Electronic Code Book) Mode: We encrypt M as $f(M_1)\ldots f(M_n)$. Here f is a permutation: decryption uses f^{-1}.

Of course this is no good. We already saw that with $n = 1$; it won't get better when we have more blocks! But still it is an important point to make. Namely, one of the modes of DES is inherently insecure as an encryption.

CBC (Cipher Block Chaining) Mode: Since this is an important scheme, let's specify it in full. The scheme is denoted $(\mathcal{E}\text{-}\mathcal{CBC}^F, \mathcal{D}\text{-}\mathcal{CBC}^F)$. The encryption function takes message M and returns $C_0C_1\ldots C_n$ where $C_0 = f(\mathrm{IV})$ and $C_i = f(C_{i-1}\oplus M_i)$. Here IV is a l-bit string called the *initialization vector* and as usual $f = F_K$. There does not seem to be a consensus on how it should be chosen. We will discuss CBC in terms of the various possibilities.

How do we decrypt? $P_i = C_{i-1}\oplus f^{-1}(C_i)$ for $i = n, n-1, \ldots, 1$. Note the CBC scheme requires that F be a family of permutations in which given K one has both $f = F_K$ and f^{-1}.

Suppose $\mathrm{IV} = 0^l$. Is this secure? We saw it should not be so viewed: a single message always has the same ciphertext.

The right thing to do is choose IV to be random. It is chosen afresh each time, different for each message. In fact, this is what is usually recommended for CBC. Henceforth, this is what we mean by the CBC encryption scheme.

We now reach an interesting point. We know of no weakness of CBC encryption with a random IV. So what do we do? Adopt it? Our view is that the lack of discernible attacks is not enough. We should seek a justification.

5.3.4 XOR encryption

Let's try something else. We saw above a nice way to encrypt a single block message M was as $(r, f(r)\oplus M)$ for randomly chosen r. How could we encrypt many blocks?

The simplest thing is to encrypt block by block. If $M = M_1\ldots M_n$ is the message, just do the above for every block. That is, pick $r_1, \ldots, r_n$ at random and output

$$(r_1, f(r_1)\oplus M_1), \ldots, (r_n, f(r_n)\oplus M_n)\ .$$

Seems fine in terms of security: notice, for example, you can't tell if two blocks are the same. But a bit expensive: the length of the ciphertext is twice the length of the plaintext.

How about $(r, f(r)\oplus M_1, f(r)\oplus M_2, f(r)\oplus M_n)$? Not so good because we can recover $M_1\oplus M_2$, etc. This is partial information!

Here is the proposal from [23]:

The Randomized XOR encryption scheme: The scheme is denoted $(\mathcal{E}\text{-}\mathcal{RXOR}, \mathcal{D}\text{-}\mathcal{RXOR})$. To encrypt $M = M_1\ldots M_n$ the encryption function picks r at random from $\{0,1\}^l$ and outputs

$$(r, f(r+1)\oplus M_1, \ldots, f(r+n)\oplus M_n)\ ,$$

where $f = F_K$. The $+$ here is really addition, say modulo 2^l where l is the input length to f and the length of r. That is, it wraps around.

This looks good. It would be something I'd be willing to use. But what exactly does "good" mean? How can we set about finding out? We need *definitions*. Then we can come back and look at the schemes.

5.4 Definition of security

There are two approaches, rooted in the definitions for security of public key encryption schemes of [87].

The first is the indistinguishability approach. The idea is this. A pair of messages M_0, M_1 is fixed. The adversary is given the ciphertext of one of them, and must say which it was. The scheme is secure if she succeeds only about half the time. (She can always succeed half the time by just guessing, so the most secure schemes are ones where she succeeds only about half the time!)

The second is the semantic security approach. Here one directly models the adversary trying to get partial information specified as some function h of the plaintext which she is trying to compute given the ciphertext. The definition again says the scheme is secure if she is unsuccessful.

It is shown in [87] that the two are equivalent.

For the private key case, a new issue arises, namely that the adversary must be supplied a way to obtain legitimate encryptions. Then many definitions can be proposed. First there are the natural analogues of the definitions for the public key case. However there are other possibilities. The definition we state below is from [23]. It is in the "indistinguishability" tradition.

The idea is that an encryption scheme is secure if you can't tell the encryption of a message from the encryption of a random string of the same length. Namely, consider the following experiment. Let M be a message, and I give it to you. Now I give you either $\mathcal{E}_K(M)$ or $\mathcal{E}_K(X)$ for a random X, the choice of which being made at random, and ask you to guess which of these two things I did. If you have a tough time telling, I want to say the scheme is secure.

Notice we have to allow the adversary some powers in determining which. It should be able to see ciphertexts that were previously encrypted by the sender. We formalize this in a worst case way: it should be allowed to get ciphertexts of plaintexts of its choice.

We can set this up via a "two worlds" type idea. In each world the adversary is provided with a particular box, and it has to tell which box it is talking to.

World 0: The adversary is given a box for $\mathcal{E}_K(\cdot)$. It can give the box M and get back $\mathcal{E}_K(M)$. Note the box is probabilistic: at each invocation, it flips coins. In particular, giving the box M twice is likely to result in two different outputs. Also note the coins flipped by the box are not given to the adversary.

World 1: The adversary is given a box which, upon being provide a message M, ignores M except to determine its length, flips coins to get a string X of length $|M|$, and returns $\mathcal{E}_K(X)$.

The problem is to tell which world you are in.

More formally, the adversary is given an oracle which behaves in one of the two ways described above, and, after querying this oracle, must say which type of oracle it was. Let the following represent the probabilities of saying 1 in the two worlds:

$$P_A^0 = \Pr\left[A^{\mathcal{E}_K(\cdot)} = 1 \ : \ K \xleftarrow{R} \{0,1\}^k\right]$$

$$P_A^1 = \Pr\left[A^{\mathcal{E}_K(\$)} = 1 \ : \ K \xleftarrow{R} \{0,1\}^k\right]$$

We may specify the space from which the messages are drawn in these experiments. This message space is usually denoted $\mathcal{M}$. (And, of course, is the same for both experiments!) Typically, however we want it to be $\{0,1\}^*$, namely any string is a valid message.

Def: $\mathsf{Adv}_A = |P_A^1 - P_A^0|$.

The scheme is secure if this advantage is small for all efficient algorithms A. In the first way of stating this, view the advantage as a function of the security parameter. (Recall this is the key length if no security parameter is explicitly specified in the scheme.)

Definition 5.4.1 Encryption scheme $(\mathcal{E}, \mathcal{D})$ is secure if for any PPT adversary A, the advantage Adv_A is negligible.

When we analyze specific schemes and want to know how they perform when implemented with primitives of specific strength, we often want more information. We need to know exactly how the advantage of an adversary would grow as a function of the time it uses and the number of encryption queries it makes. For such settings we have the following "exact" or "concrete" version of the definition.

Definition 5.4.2 An encryption scheme $(\mathcal{E}, \mathcal{D})$ is (t, q, ϵ)-secure if an A making $\leq q$ queries and running for time $\leq t$ has $\mathsf{Adv}_A \leq \epsilon$.

Sometimes, people are confused because in the above experiment, the messages are all in the clear. Isn't encryption about hiding the plaintext? Actually, the above is a much *stronger* notion. Any scheme meeting this notion has the property that you can't compute the plaintext from the ciphertext, but not vice versa: there may be schemes (and we have seen examples) where it is not possible to compute the plaintext from the ciphertext, but the scheme is nonetheless insecure.

To see this, let us see how our intuition translates into the game of the experiment. First, suppose that given a ciphertext $C = \mathcal{E}_K(M)$ you could compute M. Here is how you tell which world you are in. Ask for an encryption of a message X. Get back C, and use your ability to figure out the plaintext to get the plaintext $M = \mathcal{D}_K(C)$. If $M = X$ you were in world 0, and if $M \neq X$ then you were in world 1, most probably.

Similarly, check how the attacks we have discussed translate into this setting. For example, consider the scheme $\mathcal{E}_f(M) = f(M)$. We can distinguish between worlds by twice asking for the encryption of M; in world 0 we would twice get back the same thing, but not in world 1.

You may test your intuition by trying to see if you tell the worlds apart under the XOR scheme. You will see that you fail! The reason is the scheme is secure.

5.5 Methods for secure encryption

Now that we have definitions, we can discuss whether various encryption schemes are provably secure. We first consider the two candidate schemes we had above, then go on to other schemes.

5.5.1 CBC and XOR schemes

Both these schemes are secure assuming the underlying family F is pseudorandom.

Theorem 5.5.1 [23] *Suppose F is a (finite) family of pseudorandom functions. Then $(\mathcal{E}\text{-}\mathcal{CBC}^F, \mathcal{D}\text{-}\mathcal{CBC}^F)$ is a secure private key encryption scheme.*

Theorem 5.5.2 [23] *Suppose F is a (finite) family of pseudorandom functions. Then $(\mathcal{E}\text{-}\mathcal{RXOR}^F, \mathcal{D}\text{-}\mathcal{RXOR}^F)$ is a secure private key encryption scheme.*

These results are proved in work in progress by [23]. They show the CBC (with random IV) and XOR schemes meet the above definition. The values t, q, ϵ for which they are secure depend on the values t', q', ϵ' for which the PRF family is assumed secure.

5.5.2 Encryption with pseudorandom functions

There is a general way to encrypt with pseudorandom functions. Suppose you want to encrypt m bit messages. (Think of m as large.) Suppose we have a pseudorandom function family F in which each key K specifies a function F_K mapping l bits to m bits, for some fixed but quite large value l. Then we can encrypt M via $\mathcal{E}_K(M) = (r, F_K(r) \oplus M)$ for random r. We decrypt (r, C) by computing $M = F_K(r) \oplus C$. This is the method of [82].

Theorem 5.5.3 [82] *Suppose F is a pseudorandom function family with output length m. Then the scheme $(\mathcal{E}, \mathcal{D})$ define above is a secure private key encryption scheme for m-bit messages.*

The difference between this and the CBC and XOR methods is that in the latter, we only needed a PRF mapping l bits to l bits for some fixed l independent of the message length. One way to get such a PRF is to use DES or some other block cipher. Thus the CBC and XOR methods result in efficient encryption. To use the general scheme we have just defined we need to constructing PRFs that map l bits to m bits for large m.

There are several approaches to constructing "large" PRFs, depending on the efficiency one wants and what assumptions one wants to make. We have seen in Chapter 4 that pseudorandom function families can be built given one-way functions. Thus we could go this way, but it is quite inefficient. Alternatively, we could try to build these length extending PRFs out of given fixed length PRFs. See Section 4.6.

5.5.3 Encryption with pseudorandom bit generators

A pseudorandom bit generator is a deterministic function G which takes a k-bit seed and produces a $p(k) > k$ bit sequence of bits that looks pseudorandom. These object were defined and studied in Chapter 3. Recall the property they have is that no efficient algorithm can distinguish between a random $p(k)$ bit string and the string $G(K)$ with random K.

Recall the one-time pad encryption scheme: we just XOR the message bits with the pad bits. The problem is we run out of pad bits very soon. Pseudorandom bit generators provide probably the most natural way to get around this. If G is a pseudorandom bit generator and K is the k-bit shared key, the parties implicitly share the long sequence $G(K)$. Now, XOR message bits with the bits of $G(K)$. Never use an output bit of $G(K)$ more than once. Since we can stretch to any polynomial length, we have enough bits to encrypt.

More precisely, the parties maintain a counter N, initially 0. Let $G_i(K)$ denote the i-th bit of the output of $G(K)$. Let M be the message to encrypt. Let M_i be its i-th bit, and let n be its length. The sender computes $C_i = G_{N+i}(K) \oplus M_i$ for $i = 1, \ldots, n$ and lets $C = C_1 \ldots C_n$ be the ciphertext. This is transmitted to the receiver. Now the parties update the counter via $N \leftarrow N + n$. The total number of bits that can be encrypted is the number $p(k)$ of bits output by the generator. One can show, using the definition of PRBGs, that this works:

Theorem 5.5.4 *If G is a secure pseudorandom bit generator then the above is a secure encryption scheme.*

One seeming disadvantage of using a PRBG is that the parties must maintain a common, synchronized counter, since both need to know where they are in the sequence $G(K)$. (Note that the schemes we have discussed above avoid this. Although some of the schemes above may optionally use a counter instead of a random value, this counter is not a synchronized one: the sender maintains a counter, but the receiver does not, and doesn't care that the sender thinks of counters.) To get around this, we might have the sender send the current counter value N (in the clear) with each message. If authentication is being used, the value N should be authenticated.

The more major disadvantage is that the pseudorandom sequence $G(K)$ may not have random access. To produce the i-th bit one may have to start from the beginning and produce all bits up to the i-th one. (This means the time to encrypt M depends on the number and length of message encrypted in the past, not a desirable feature.) Alternatively the sequence $G(K)$ may be pre-computed and stored, but this uses a lot of storage. Whether this drawback exists or not depends of course on the choice of PRBG G.

So how do we get pseudorandom bit generators? We saw some number theoretic constructions in Chapter 3. These are less efficient than block cipher based methods, but are based on different kinds of assumptions which might be preferable. More importantly, though, these constructions have the drawback that random access is not possible. Alternatively, one could build pseudorandom bit generators out of finite PRFs. This can be done so that random access is possible. However the resulting encryption scheme ends up being not too different from the XOR scheme with a counter so it isn't clear it is worth a separate discussion.

5.5.4 Encryption with one-way functions

We saw in Chapter 3 that pseudorandom bit generators exist if one-way functions exist [98, 96]. It is also known that given any secure private key encryption scheme one can construct a one-way function [99]. Thus we have the following.

Theorem 5.5.5 *There exists a secure private key encryption scheme if and only if there exists a one-way function.*

We will see later that the existence of secure public key encryption schemes requires different kinds of assumptions, namely the existence of primitives with "trapdoors."

CHAPTER 6

Public key encryption

The idea of a public-key cryptosystem (PKC) was proposed by Diffie and Hellman in their pioneering paper [62] in 1976. Their revolutionary idea was to enable secure message exchange between sender and receiver without ever having to meet in advance to agree on a common secret key. They proposed the concept of a trapdoor function and how it can be used to achieve a public-key cryptosystem. Shortly there after Rivest, Shamir and Adelman proposed the first candidate trapdoor function, the RSA. The story of modern cryptography followed.

The setup for a public-key cryptosystem is of a network of users $u_1 \cdots u_n$ rather than an single pair of users. Each user u in the network has a pair of keys $< P_u, S_u >$ associated with him, the *public key* P_u which is published under the users name in a "public directory" accessible for everyone to read, and the private-key S_u which is known only to u. The pairs of keys are generated by running a *key-generation* algorithm. To send a secret message m to u everyone in the network uses the *same* exact method, which involves looking up P_u, computing $E(P_u, m)$ where E is a public encryption algorithm, and sending the resulting ciphertext c to u. Upon receiving ciphertext c, user u can decrypt by looking up his private key S_u and computing $D(S_u, c)$ where D is a public decryption algorithm. Clearly, for this to work we need that $D(S_u, E(P_u, m)) = m$.

A particular PKC is thus defined by a triplet of public algorithms (G, E, D), the key generation, encryption, and decryption algorithms.

6.1 Definition of Public-Key Encryption

We now formally define a public-key encryption scheme. For now the definition will say nothing about we mean by "security" of a scheme (which is the subject of much discussion in subsequent sections).

Definition 6.1.1 A *public-key encryption scheme* is a triple, (G, E, D), of probabilistic polynomial-time algorithms satisfying the following conditions

(1) key generation algorithm : a probabilistic expected polynomial-time algorithm G, which, on input 1^k (the security parameter) produces a pair (e, d) where e is called the public key , and d is the corresponding private key. (Notation: $(e, d) \in G(1^k)$). We will also refer to the pair (e, d) a pair of *encryption/decryption* keys.

(2) An encryption algorithm: a probabilistic polynomial time algorithm E which takes as input a security parameter 1^k, a public-key e from the range of $G(1^k)$ and string $m \in \{0,1\}^k$ called the *message*, and produces as output string $c \in \{0,1\}^*$ called the *ciphertext*. (We use the notation $c \in E(1^k, e, m)$ to denote c being an encryption of message m using key e with security parameter k. When clear, we use shorthand $c \in E_e(m)$, or $c \in E(m)$.)

(3) A decryption algorithm: a probabilistic polynomial time algorithm D that takes as inputs a security parameter 1^k, a private-key d from the range of $G(1^k)$, and a ciphertext c from the range of $E(1^k, e, m)$, and produces as output a string $m' \in \{0,1\}^*$, such that for every pair (e,d) in the range of $G(1^k)$, for every m, for every $c \in D(1^k, e, m)$, the $prob(D(1^k, d, c) \neq m')$ is negligible.

(4) Furthermore, this system is "secure" (see Definition 6.3).

How to use this definition. To use a public-key encryption scheme (G, E, D) with security parameter 1^k, user A runs $G(1^k)$ to obtain a pair (e,d) of encryption/decryption keys. User A then "publishes" e in a public file, and keeps private d. If anyone wants to send A a message, then need to lookup e and compute $E(1^k, e, m)$. Upon receipt of $c \in E(1^k, e, m)$, A computes message $m = D(1^k, d, c)$.

Comments on the Definitions

Comment 0: Note that essentially there is no difference between the definition of a private-key encryption scheme and the definition of a public-key encryption scheme at this point. We could have defined a private key encryption scheme to have one key e for encryption and a different key d for decryption. The difference between the two definitions comes up in the security definition. In a public-key encryption scheme the adversary or "breaking algorithm" is given e (the *public key*) as an additional input; whereas in private-key scheme e is not given to the adversary (thus without loss of generality one may assume that $e = d$).

Comment 1: At this stage, encryption using a key of length k is defined only for messages of length k; generalization is postponed to Convention 6.1.

Comment 2: Note that as algorithm G is polynomial time, the length of its output (e,d) (or e in the private-key encryption case) is bounded by a polynomial in k. On the other hand, since k also serves as the "security parameter", k must be polynomial in $|d|$ (or $|e|$ in the private-key encryption case) in which case "polynomial in k" is equivalent to "polynomial in $|d|$".

Comment 3: Condition (3) in Definition 6.4.3 and 6.1.1 may be relaxed so that inequality may occur with negligible probability. For simplicity, we chose to adopt here the more conservative requirement.

Comment 4: We have allowed the encryption algorithm in both of the above definitions to be probabilistic. Namely, there can be many cyphertexts corresponding to the same message. In the simple (informal) example of a public-key encryption scheme based on a trapdoor function outlined in the introduction, every message has a unique corresponding ciphertext. That is too restrictive as, for example, if E is deterministic, the same inputs would always produce the same outputs, an undesirable characteristic.

Comment 5: We allowed D to be a probabilistic algorithms. This may conceivably allow the consideration of encryption schemes which may offer higher security ([42]). Accordingly, we may relax the requirement that $\forall m, D(E(m)) = m$ to hold only with high probability.

Conventions Regarding Definitions

Messages of length not equal to k (the length of the encryption key) are encrypted by breaking them into blocks of length k and possibly padding the last block. We extend the notation so that

$$E_e(\alpha_1 \cdots \alpha_l \alpha_{l+1}) = E_e(\alpha_1) \cdots E_e(\alpha_l) \cdot E_e(\alpha_{l+1} p)$$

where $|\alpha_1| = \cdots = |\alpha_l| = k$, $|\alpha_{l+1}| \leq k$, and p is some standard padding of length $k - |\alpha_{l+1}|$.

The above convention may be interpreted in two ways. First, it waves the extremely restricting convention by which the encryption scheme can be used only to encrypt messages of length equal to the length of the key. Second, it allows to reduce the security of encrypting many messages using the same key to the security of encrypting a single message.

The next convention regarding encryption schemes introduces a breach of security: namely, the length of the cleartext is always revealed by encryption schemes which follow this convention. However, as we show in a latter section some information about the length of the cleartext must be leaked by any encryption scheme.

The encryption algorithm maps messages of the same length to cryptograms of the same length.

6.2 Simple Examples of PKC: The Trapdoor Function Model

A collection of trapdoor functions, discussed at length in the chapter on one-way functions and trapdoor functions, has been defined as $F = \{f_i : D_i \rightarrow D_i\}_{i \in I}$ where $D_i \subseteq \{0,1\}^{|i|}$, and I is a set of indices. Recall that $\forall i$, f_i was easy to compute, but hard to invert; and $\forall i$, there existed t_i such that given t_i and $f_i(x)$, $f_i(x)$ could be inverted in polynomial time.

Diffie and Hellman suggested using the supposed existence of trapdoor functions to implement Public Key Cryptosystems as follows.

(1) The generator G on security parameter 1^k outputs pairs (f, t_f) where f is a trapdoor function and t_f its associated trapdoor information.

(2) For every message $m \in M$, $E(f, m) = f(m)$.

(3) Given $c \in E(f, m)$ and t_f, $D(t_f, c) = f^{-1}(c) = f^{-1}(f(m)) = m$.

6.2.1 Problems with the Trapdoor Function Model

There are several immediate problems which come up in using the trapdoor function model for public key encryption.

We summarize briefly the main problems which will be elaborated on in the next few sections.

(1) *Special Message Spaces.* The fact that f is a trapdoor function doesn't imply that inverting $f(x)$ when x is *special* is hard. Namely, suppose that the set of messages that we would like to send is drawn from a highly structured message space such as the English language, or more simply $M = \{0,1\}$, it may be easy to invert $f(m)$. In fact, it is always easy to distinguish between $f(0)$ and $f(1)$).

(2) *Partial Information.* The fact that f is a one-way or trapdoor function doesn't necessarily imply that $f(x)$ hide all information about x. Even a bit of leakage many be too much for some applications. For example, for candidate one-way function $f(p, g, x) = g^x \bmod p$ where p is prime and g is a generator, the least significant bit of x is always easily computable from $f(x)$. For RSA function $f(n, l, x) = x^l \bmod n$, the Jacobi symbol $\mathbf{J}_n(x) = \mathbf{J}_n(x^l \bmod n)$. Namely, the Jacobi symbol of x is easy to compute from $f(n, l, x)$ – this was observed by Lipton[117] who used this fast to crack a protocol for Mental poker by Shamir Rivest and Adleman[168]. See below. Moreover, In fact, for any one-way function f, information such as "the parity of $f(m)$" about m is always easy to compute from $f(m)$. See below.

(3) *Relationship between Encrypted Messages* Clearly, we may be sending messages which are related to each other in the course of a communication. Some examples are: sending the same secret message to several recipients, or sending the same message (or slight variants) many times. It is thus desirable and sometimes essential that such dependencies remain secret. In the trapdoor function model, it is trivial to see that sending the same message twice is always detectable. More serious problems were noted by several researchers, most notably by Håstad who shows [95] that if RSA with an exponent l is used, and the same message (or known linear combinations of the same message) is send to l recipients, then the message can be computed by an adversary.

6.2.2 Problems with Deterministic Encryption in General

The above problems are actually shared by any public-key cryptosystem in which the encryption algorithm is deterministic.

It is obvious for problems 1 and 3 above. It is easy to show also for problem 3 as follows. Let E is any deterministic encryption algorithm, we can extract partial information by using something similar to the following predicate:

$$P(x) = \left\{ \begin{array}{l} 1 \text{ if } E(x) \text{ even} \\ 0 \text{ if } E(x) \text{ odd} \end{array} \right\}$$

It is clear that we can easily compute this predicate since all we have to do is take the low bit of $E(x)$. Unless $E(x)$ is always even or always odd for all the x's in the message space, we have obtained partial information about x. If $E(x)$ is always even or odd, the low bit of $E(x)$ contains no information. But, some other bit of $E(x)$ must contain some information otherwise the message space is composed of only one

message in which case we have total information. Then, simply use that bit instead of the lowest bit and we have a partial information obtaining predicate.

6.2.3 The RSA Cryptosystem

In 1977 Shamir, Rivest and Adelman proposed the first implementation of trapdoor function, the RSA function, [151]. We refer the reader to chapter 2, in particular sections 2.2.5 and Section 2.2.17 for a thorough treatment of the RSA trapdoor function.

Here, let us examine the use of the RSA trapdoor function for the purpose of encryption in the straight forward manner proposed by Diffie and Hellman. We will show that it will not satisfy the kind of security which we desire. We will later see that a probabilistic variant will do the job.

Recall the definition of RSA trapdoor function 2.2.17. Let p, q denote primes, $n = pq$, $Z_n^* = \{1 \leq x \leq n, (x, n) = 1\}$ the multiplicative group whose cardinality is $\varphi(n) = (p-1)(q-1)$, and $e \in Z_{p-1}$ relatively prime to $\varphi(n)$. Our set of indices will be $I = \{< n, e > \text{ such that } n = pq \ |p| = |q|\}$ and the trapdoor associated with the particular index $< n, e >$ be $t_{<n,e>} = d$ such that $ed = 1 \bmod \phi(n)$. Let $RSA = \{RSA_{<n,e>} : Z_n^* \to Z_n^*\}_{<n,e> \in I}$ where

$$RSA_{<n,e>}(x) = x^e \bmod n$$

Sparse Message Spaces

We showed that the RSA function has some nice properties that seem especially good for use as a PKC. For example, we showed for a given pair $< n, e >$, it is either hard to invert $RSA_{<n,e>}$ for all but a negligible fraction of x's in Z_n^*, or easy to invert $RSA_{<n,e>}(x)$ $\forall x, x \in \mathbf{Z}^*$. Does this mean that the RSA cryptosystem is difficult to break for almost all messages if factoring integers is hard? The answer is negative.

Suppose that the message space M we are interested in is the English language. Then, let $M_k = \{0,1\}^k$ where $m \in M_k$ is an English sentence. Compared to the entire space, the set of English sentences is quite small. For example, $\frac{|M_k|}{|\mathbf{Z}_n^*|} \leq \frac{1}{2\sqrt{n}}$. Thus it is possible that $f_{n,e}(x)$ is easy to invert for all $x \in M_k$, even if the factorization problem is hard. In other words, English sentences are highly structured; it might well be the case that our function can be easily inverted on all such inputs. Clearly, we would ultimately like our encryption schemes to be secure for all types of message spaces, including English text.

Partial Information about RSA

What partial information about x can be computed from $RSA_{<n,e>}(x)$.

We showed in the chapter on one-way and trapdoor functions, that indeed some bits such as the least significant bit and most significant bit of RSA are very well hidden. This is the good news.

Unfortunately, in some cases very subtle leakage of partial information can defeat the whole purpose of encryption. We present a "cute" example of this shown by Lipton shortly after RSA was invented.

An Example: Mental Poker (SRA '76): Mental Poker is a protocol by which two parties each of whom distrusts the other can deal each other cards from a deck without either being able to cheat. The protocol for A to deal B a card goes like this:

(1) A and B agree on a set $X = \{x_1, \ldots, x_{52}\}, x_i \in Z_n^*$, of random numbers where $n = pq$, p and q prime and known to both A and B. These numbers represent the deck of cards, x_i representing the ith card in the deck.

(2) A picks s such that $(s, \varphi(n)) = 1$, and t such that $st \equiv 1 \pmod{\varphi(n)}$ secretly. B does the same for e and f. (I.e., $ef \equiv 1 \pmod{\varphi(n)}$)

(3) A calculates $x_i^s \bmod n$ for $i = 1 \ldots 52$, shuffles the numbers, and sends them to B.

(4) B calculates $(x_i^s \bmod n)^e \bmod n$ for $i = 1 \ldots 52$, shuffles the numbers, and sends them to A.

(5) A calculates $((x_i^s \bmod n)^e \bmod n)^t \bmod n = x_i^e \bmod n$ for $i = 1 \ldots 52$. A then chooses a card randomly (I.e., picks x_j^e where $j \in [1 \ldots 52]$) and sends it to B.

(6) B then takes $(x_j^e \bmod n)^d \bmod n = x_j \bmod n$. This is the card B has been dealt.

Why it works: Note that so long as no partial information can be obtained from the RSA trapdoor function, neither A nor B can influence in any way the probably of B getting any given card. A is unable to give B bad cards and likewise B can not deal himself good cards. This follows from the fact that encrypting the cards is analogous to placing each of them in boxes locked with padlocks. So long as a card is locked in a box with a padlock of the other player's on it, nothing can be told about it and it is indistinguishable from the other locked boxes.

When B gets the deck in step 3, he has no idea which card is which and thus is unable to influence which card he is dealt. However, A can still tell them apart since its A's padlocks that are on the boxes. To prevent A from being able to influence the cards, B then puts his own locks on the boxes as well and shuffles the deck. Now A also can not tell the cards apart so when he is forced to make his choice, he is forced to just deal a random card. Thus, the two players in spite of distrusting each other can play poker.

How to extract partial information from the RSA function: The protocol fails, however, because it is possible to extract partial information from the RSA function and thus determine to some degree of accuracy which cards are which and hence influence the outcome of the draw. One way to do this is by computing the Jacobi symbol since $(\mathbf{J}_n(x_i)) = (\mathbf{J}_n(x_i^s))$ since s is odd. Thus, since half of the x_i's have a Jacobi symbol of 1 on the average since they are random numbers in Z_n^*, we can extract roughly one bit of information from each of the cards. In order to influence the draw in our favor, we simply determine whether or not the cards with a Jacobi symbol of 1 or the cards with a Jacobi symbol of -1 are better for us and then draw only from that set of cards.

One's immediate reaction to this, of course, is simply to modify the protocol so that in step 1 only numbers with say a Jacobi symbol of 1 are chosen. Then no information will be gained by computing the Jacobi symbol. However, this is no guarantee that some other more clever predicate does not exist which can still extract partial information and indeed such functions must exist by the very nature of trapdoor functions.

Low exponent attacks

Let the exponent be $e = 3$. We saw that any exponent relatively prime to $\varphi(N)$ is OK, and we can easily choose $N = pq$ so that 3 is relatively prime to $(p-1)(q-1) = \varphi(N)$. This is a popular choice for performance reasons. Encryption is now fast. And we saw that RSA is still (assumed) one-way under this choice.

So encryption of m is now $m^3 \bmod N$. Here is an interesting attack illustrating the weaknesses of RSA encryption, due to Coppersmith, Franklin, Patarin and Reiter [54]. Suppose I encrypt m and then $m+1$. I claim you can recover m. We have ciphertexts:

$$\begin{aligned} c_1 &= m^3 \\ c_2 &= (m+1)^3 = m^3 + 3m + 3m^2 + 1 = c_1 + 3m + 3m^2 + 1 \end{aligned}$$

Now lets try to solve for m. Perhaps the first thought that springs to mind is that we have a quadratic equation for m. But taking square roots is hard, so we don't know how to solve it that way. It turns out the following works:

$$\frac{c_2 + 2c_1 - 1}{c_2 - c_1 + 2} = \frac{(m+1)^3 + 2m^3 - 1}{(m+1)^3 - m^3 + 2} = \frac{3m^3 + 3m^2 + 3m}{3m^2 + 3m + 3} = m \ .$$

This can be generalized. First, you can generalize to messages m and $\alpha m + \beta$ for known α, β. Second, it works for exponents greater than 3. The attack then runs in time $O(e^2)$ so it is feasible for small exponents. Finally, it can work for k messages related by a higher degree polynomial.

These are the kinds of attacks we most definitely would like to avoid.

6.2.4 Rabin's Public key Cryptosystem

Recall Rabin's trapdoor function from Chapter 2.

$$f_n(m) \equiv m^2 \mod n$$

where n is the product of two large primes, p and q. Once again, this function can yield another example of a trapdoor/public key cryptosystem except that f_n is not as permutation but a 4-to-1 function. An inverse of $f_n(m)$:

$$f_n^{-1}(m^2) = x \text{ such that } x^2 = m^2 \mod n$$

However, in practice, when we invert Rabin's function, we do not simply want any square root of the encrypted message, but the correct one of the four that was meant to be sent by the sender and would be meaningful to the intended recipient. So, we need to add a constraint to uniquely identify the root x which must be output by the decryption algorithm on $f_n(m^2)$ such as find x such that $x^2 = m^2 \mod n$, and $x \in S$ where S is a property for which it is quite unlikely that there exists two roots $m, x \in S$. What could S be? Well if the message space M is sparse in $\mathbf{Z}_n^*$ (which would usually be the case), then S may be simply M. In such case it is unlikely that there exists $m \neq m' \in M$ such that $m'^2 = m^2 \mod n$. (If M is not sparse, S may be the all x whose last 20 digits are r for some random r. Then to send m in secrecy, $(f_n(m') = f_n(2^{20}m + r), r)$ need be sent.)

Recall, that earlier in the class, we had shown that inverting Rabin's function is as hard as factoring. Namely, we had shown that inverting Rabin's function for ϵ of the $m^2 \in \mathbf{Z}_n^*$'s implies the ability to factor. The proof went as follows:

- Suppose there existed a black box that on inputs x^2 responded with a y such that $x^2 = y^2 \mod n$. Then, to factor n, choose an i at random from $\mathcal{Z}_n^*$ and give as input $i^2 \mod n$ to the black box. If the box responds with a y, such that $y \neq \pm i$, then we can indeed factor n by computing $gcd(i \pm y, n)$. In the case that $y = \pm i$, we have gained no information, so repeat.

If we think of this black box as a decoding algorithm for the public key system based on Rabin's function used to encrypt messages in message space M, can we conclude that *if it is possible to decrypt the public key system $f_n(m)$ for $m \in M$, then it is possible to factor n?*

If the message space M is sparse, then the answer is **no.** Why? for the black box (above) to be of any use we need to feed it with an $f_n(i)$ for which there exists an y such that $y \in M$ and $y \neq i$. The probability that such y exists is about $\frac{|M|}{|\mathbf{Z}_n^*|}$, which may be exponentially small.

If the message space M is not sparse, we run into another problem. Rabin's scheme would not be secure in the presence of an active adversary who is capable of a chosen ciphertext attack. This is easy to see again using the above proof that inverting Rabin's function is as hard as factoring. Temporary access to a decoding algorithm for Rabin's public key encryption for message in M, is the same as having access to the black box of the above proof. The adversary chooses i at random and feeds the decoding algorithm with $f_n(i)$. If the adversary gets back y such that $y^2 = i^2 \mod n$, (again, $i \neq \pm y$), factor n, and obtain the secret key. If M is not sparse this will be the case after trying a polynomial number of i's. From here on, the adversary would be able to decrypt any ciphertext with the aid of the secret key and without the need for a black box.

Therefore, either Rabin's scheme is not equivalent to factoring, which is the case when inverting on a sparse message space, or (when M is not sparse) it is insecure before a chosen ciphertext adversary.

6.2.5 Knapsacks

A number of public-key cryptosystems have been proposed which are based on the *knapsack* (or — more properly — the *subset sum*) problem: given a vector $\mathbf{a} = (a_1, a_2, \ldots, a_n)$ of integers, and a target value C, to determine if there is a length-n vector $\mathbf{x}$ of zeroes and ones such that $\mathbf{a} \cdot \mathbf{x} = C$. This problem is NP-complete [78].

To use the knapsack problem as the basis for a public-key cryptosystem, you create a public key by creating a knapsack vector $\mathbf{a}$, and publish that as your public key. Someone else can send you the encryption of a message M (where M is a length-n bit vector), by sending you the value of the inner product $C = M \cdot \mathbf{a}$. Clearly, to decrypt this ciphertext is an instance of the knapsack problem. To make this problem easy for you, you need to build in hidden structure (that is, a trapdoor) into the knapsack so that the encryption operation becomes one-to-one and so that you can decrypt a received ciphertext easily. It seems, however, that the problem of solving knapsacks containing a trapdoor is *not* NP-complete, so that the difficulty of breaking such a knapsack is no longer related to the $P = NP$ question.

In fact, history has not been kind to knapsack schemes; most of them have been broken by extremely clever analysis and the use of the powerful L^3 algorithm [114] for working in lattices. See [125, 163, 165, 2, 167, 110, 40, 134].

Some knapsack or knapsack-like schemes are still unbroken. The Chor-Rivest scheme [52], and the multiplicative versions of the knapsack [125] are examples. McEliece has a knapsack-like public-key cryptosystem based on error-correcting codes [122]. This scheme has not been broken, and was the first scheme to use randomization in the encryption process.

We are now ready to introduce what is required from a *secure* Public Key Cryptosystem.

6.3 Defining Security

Brain storming about what it means to be secure brings immediately to mind several desirable properties. Let us start with the the minimal requirement and build up.

First and foremost the private key should not be recoverable from seeing the public key. Secondly, with high probability for any message space, messages should not be entirely recovered from seeing their encrypted form and the public file. Thirdly, we may want that in fact no useful information can be computed about messages from their encrypted form. Fourthly, we do not want the adversary to be able to compute any useful facts about traffic of messages, such as recognize that two messages of identical content were sent, nor would we want her probability of successfully deciphering a message to increase if the time of delivery or relationship to previous encrypted messages were made known to her.

In short, it would be desirable for the encryption scheme to be the mathematical analogy of opaque envelopes containing a piece of paper on which the message is written. The envelopes should be such that all legal senders can fill it, but only the legal recipient can open it.

We must answer a few questions:

- How can "opaque envelopes" be captured in a precise mathematical definition?
- Are "opaque envelopes" achievable mathematically?

Several definitions of security attempting to capture the "opaque envelope" analogy have been proposed. All definitions proposed so far have been shown to be equivalent. We describe two of them and show they are equivalent.

6.3.1 Definition of Security: Polynomial Indistinguishability

Informally, we say that an encryption scheme is polynomial time indistinguishable if no adversary can find even two messages, whose encryptions he can distinguish between. If we recall the envelope analogy, this translates to saying says that we cannot tell two envelopes apart.

Definition 6.3.1 We say that a Public Key Cryptosystem (G, E, D) is *polynomial time indistinguishable* if for every PPT M, A, and for every polynomial Q, $\forall$ sufficiently large k

$$Pr(A(1^k, e, m_0, m_1, c) = m \mid (e,d) \stackrel{R}{\leftarrow} G(1^k) \ ; \ \{m_0, m_1\} \stackrel{R}{\leftarrow} M(1^k) \ ; \ m \stackrel{R}{\leftarrow} \{m_0, m_1\} \ ; \ c \stackrel{R}{\leftarrow} E(e,m)) < \frac{1}{2} + \frac{1}{Q(k)} \tag{6.1}$$

In other words, it is impossible in polynomial in k time to find two messages m_0, m_1 such that a polynomial time algorithm can distinguish between $c \in E(e, m_0)$ and $c \in E(e, m_1)$.

Remarks about the definition:

(1) We remark that a stronger form of security would be: the above holding $\forall m_0, m_1$, (not only those which can be found in polynomial time by running $M(1^k)$). Such security can be shown in a non-uniform model, or when the messages are chosen before the keys, and thus can not involve any information about the secret keys themselves.

(2) In the case of private-key encryption scheme, the definition changes ever so slightly. The encryption key e is not given to algorithm A.

(3) Note that any encryption scheme in which the encryption algorithm E is deterministic immediately fails to pass this security requirement. (e.g given f, m_0, m_1 and $c \in \{f(m_1), f(m_0)\}$ it is trivial to decide whether $c = f(m_0)$ or $c = f(m_1)$.

(4) Note that even if the adversary know that the messages being encrypted is one of two, he still cannot tell the distributions of ciphertext of one message apart from the other.

6.3.2 Another Definition: Semantic Security

Consider the following two games. Let $h : M \rightarrow \{0,1\}^*$, where M is a message space in which we can sample in polynomial time, or equivalently, a probabilistic polynomial time algorithm M that takes as input 1^k and generates a message $m \in \{0,1\}^k$, and $h(m)$ is some information about the message (for example, let be such that $h(m) = 1$ if m has the letter 'e' in it, then $V = \{0,1\}$).

- Game 1: I tell the adversary that I am about to choose $m \in M(1^k)$ and, ask her to guess h(m).
- Game 2: I tell the adversary $\alpha \in E(m)$, for some $m \in M(1^k)$ and once again, ask her to guess h(m).

In both of the above cases we may assume that the adversary knows the message space algorithm M and the public key P.

In the first game, the adversary only knows that a message m is about to be chosen. In addition to this fact, the adversary of the Game 2 sees the actual ciphertext itself. For all types of message spaces, semantic security will essentially require that the probability of the adversary winning Game 1 to be about the same as her probability of winning Game 2. Namely, that the adversary should not gain any advantage or information from having seen the ciphertext resulting from our encryption algorithm.

Said differently, this definition will require that for all probability distributions over the message space, no partial information about the message can be computed from the ciphertext. This requirement is reminiscent of Shannon's perfect security definition – with respect to a computationally bounded adversary.

Definition 6.3.2 We say that an encryption scheme (G, E, D) is *semantically secure* if for all PPT algorithms M and A, functions h, polynomials Q there is a PPT B such that for sufficiently large k,

$$\begin{aligned}\Pr(A(1^k, c, e) = h(m) \mid (e,d) \stackrel{R}{\leftarrow} G(1^k) \ ; \ m \stackrel{R}{\leftarrow} M(1^k) \ ; \ c \stackrel{R}{\leftarrow} E(e,m)) \\ \leq \Pr(B(1^k) = h(m) \mid m \stackrel{R}{\leftarrow} M(1^k)) + \frac{1}{Q(k)} \qquad (6.2)\end{aligned}$$

Here, Game 1 is represented by PTM B, and Game 2 by PTM A. Again, this can only hold true when the encryption algorithm is a probabilistic one which selects one of many possible encodings for a message; otherwise, if E were deterministic, and $M = \{0,1\}$, then any adversary would have 100% chance of guessing correctly $h(m)$ for $m \in M$ by simply testing whether $E(0) = c$ or $E(1) = c$.

Theorem 6.3.3 *A Public Key Cryptosystem passes Indistinguishable Security if and only if it passes Semantic Security.*

6.4 Probabilistic Public Key Encryption

We turn now to showing how to actually build a public key encryption scheme which is polynomial time indistinguishable.

In order to do so, we must abandon the trapdoor function PKC model and deterministic algorithms of encryption all together, in favor of probabilistic encryption algorithm. The probabilistic encryption algorithm which we will construct will still assume the existence of trapdoor functions and use them as a primitive building block.

The key to the construction is to first answer a simpler problem: How to securely encrypt single bits. We show two ways to approach this problem. The first is based on trapdoor predicates as discussed in Section 2.5, and the second is based on hard core predicates as discussed in Section 2.4.

6.4.1 Encrypting Single Bits: Trapdoor Predicates

To encrypt single bits, the notion of one-way and trapdoor predicates was introduced by [87]. It later turned out to be also quite useful for protocol design. We refer the reader to section 2.5 for a general treatment of this subject. Here we look at its use for encryption.

The Idea: Briefly, a one-way predicate, is a Boolean function which is hard to compute in a very strong sense. Namely, an adversary cannot compute the predicate value better than by taking a random guess. Yet, it is possible to sample the domain of the predicate for elements for which the predicate evaluates to 0 and to 1. A trapdoor predicate possesses the extra feature that there exists some trapdoor information that enables the computation of the predicate. We can construct examples of collection of trapdoor predicates based on the intractability of factoring, RSA inversion and the difficulty of distinguishing quadratic residues from non-residues.

Now, given a collection of trapdoor predicates exist, we use them to set up a cryptosystem for one bit encryption as follows. Every user A chooses and publishes a random trapdoor predicate, keeping secret the corresponding trapdoor information. To send A a one bit message m, any other user chooses at random an element in the domain of the trapdoor predicate for which the predicate evaluates to m. To decrypt, A uses his trapdoor information to compute the value of predicate on the domain element it receives. Note, that this is a probabilistic encryption with many possible cyphertexts for 0 as well as 1, where essentially an adversary cannot distinguish between an encoding of 0 and an encoding of 1.

Recall, the formal definition of trapdoor predicates 2.5.2.

Let I be a set of indices and for $i \in I$ let D_i be finite. A *collection of trapdoor predicates* is a set $B = \{B_i : D_i \rightarrow \{0,1\}\}_{i \in I}$ satisfying the following conditions. Let $D_i^v = \{x \in D_i, B_i(x) = v$.

1. There exists a polynomial p and a PTM S_1 which on input 1^k finds pairs (i, t_i) where $i \in I \cap \{0,1\}^k$ and $|t_i| < p(k)$ The information t_i is referred to as the trapdoor of i.
2. There exists a PTM S_2 which on input $i \in I$ and $v \in \{0,1\}$ outputs $x \in D_i$ at random such that $B_i(x) = v$.
3. There exists a PTM A_1 such that for $i \in I$ and trapdoor t_i, $x \in D_i$ $A_1(i, t_i, x) = B_i(x)$.
4. For every PPT A there exists a negligible ν_A such that $\forall\ k$ large enough

$$\Pr\left[z \neq v \ : \ i \stackrel{R}{\leftarrow} I \cap \{0,1\}^k \ ; \ v \stackrel{R}{\leftarrow} \{0,1\} \ ; \ x \stackrel{R}{\leftarrow} D_i^v \ ; \ z \leftarrow A(i,x)\right] \leq \nu_A(k)$$

Definition 6.4.1 Assume that B is a collection of trapdoor predicates. We can now define a public key cryptosystem $(G, E, D)_B$ for sending single bit messages as follows:

- Key generation algorithm: $G(1^k)$ chooses (i, t_i) (public key is then i and private key is t_i). This is doable by running algorithm S_1.
- Encryption algorithm: Let $m \in \{0,1\}$ be the message. Encryption algorithm $E(i, e)$ selects $x \in D_i^m$. (The ciphertext is thus x). This is doable by running algorithm S_2.
- Decryption algorithm: $D(c, t_i)$ computes $B_i(c)$. This is doable using A_1 given the trapdoor information.

It is clear from the definition of a set of trapdoor predicates, that all of the above operations can be done in expected polynomial time and that messages can indeed be sent this way. It follows immediately from the definition of trapdoor predicates than indeed this system is polynomially indistinguishable when restricted to one bit message spaces.

6.4.2 Encrypting Single Bits: Hard Core Predicates

Alternatively, you may take the following perhaps simpler approach, starting directly with trapdoor functions and using their hard core predicates. For a detailed discussion of trapdoor functions and hard core predicates for them see section Section 2.5.2. The discussion here assumes such knowledge.

Recall that a collection of trapdoor permutations is a set $F = \{f_i : D_i \longrightarrow D_i\}_{i \in I}$ such that:

1. $S_1(1^k)$ samples (i, t_i) where $i \in I$, $|i| = k$ and $|t_i| < p(k)$ for some polynomial p.
2. $S_2(i)$ samples $x \in D_i$.
3. $\exists$ PTM A_1 such that $A_1(i, x) = f_i(x)$.
4. $\Pr[A(i, f_i(x)) \in f_i^{-1}(f_i(x))] < \frac{1}{Q(k)}$ $\forall$ PTM A, $\forall Q$, $\forall k > k_0$.
5. $\exists$ PTM A_2 such that $A_2(i, t_i, f_i(x)) = x$, $\forall x \in D_i$, $i \in I$.

Further, let $B_i(x)$ be hard core for $f_i(x)$. Recall that the existence of F implies the existence of $F\prime$ that has a hard core predicate. So, for notational simplicity assume that $F = F\prime$. Also recall that for the RSA collection of trapdoor functions, LSB is a collection of hard core predicate the LSB.

Definition 6.4.2 Given a collection F with hard core predicates B, define public key cryptosystem $(G, E, D)_B$ for sending a single bit as follows:

- Key generation algorithm: $G(1^k)$ chooses pair $< i, t_i >$ by running $S_1(1^k)$. (for RSA, $G(1^k)$ chooses $< n, e >, d$ such that n is an RSA modulus, and $ed = 1 \bmod \phi(n)$.)
- Encryption algorithm: $E(i, m)$ chooses at random an $x \in D_i$ such that $B_i(x) = m$, and output as a ciphertext $f_i(x)$. Using the Goldreich Levin construction of a hard core predicate, simply choose x, r such that the inner product of x and r is m and output $f(x) \circ r$. (for RSA, to encrypt bit m, choose at random an $x \in Z_n^*$ such that $LSB_{<n,e>}(x) = m$ and output as a ciphertext $RSA_{<n,e>}(x)$.)
- Decryption algorithm: To decrypt $c = f_i(x)$, given i and t_i, the decryption algorithm $D(t_i, c)$ compute $B_i(f_i^{-1}(c)) = B_i(x) = m$. Using the Goldreich Levin construction this amounts to given $c = f_i(x) \circ r$ to computing the inner product of x and r. (for RSA, to decrypt c, given n, e and d, compute the $LSB((RSA_{<n,e>}(x))^d)$ = least significant bit of x.)

6.4.3 General Probabilistic Encryption

How should we encrypt arbitrary length messages?

The first answer is to simply encrypt each bit individually using one of the above methods. as above. Before considering whether this is wise from an efficiency point of view, we need to argue that it indeed will produce a encryption scheme which is polynomial time indistinguishable. This requires reflection, as even through every bit individually is secure, it can be the case that say that some predicate computed on all the bits is easily computable, such as the exclusive or of the bits. This turns out luckily not to be the case, but requires proof.

We now provide construction and proof.

Definition 6.4.3 We define a probabilistic encryption based on trapdoor collection F with hard core bit B to be $PE = (G, E, D)$ where:

- $G(1^k)$ chooses (i, t_i) by running $S_1(1^k)$ (Public key is i, private key is t_i).
- Let $m = m_1 \ldots m_k$ where $m_j \in \{0, 1\}$ be the message.
  $E(i, m)$ encrypts m as follows:
  Choose $x_j \in_R D_i$ such that $B_i(x_j) = m_j$ for $j = 1, \ldots, k$.
  Output $c = f_i(x_1) \ldots f_i(x_k)$.
- Let $c = y_1 \ldots y_k$ where $y_i \in D_i$ be the cyph ertext.
  $D(t_i, c)$ decrypts c as follows:
  Compute $m_j = B_i(f_i^{-1}(y_j))$ for $j = 1, \ldots, k$.
  Output $m = m_1 \ldots m_k$.

Claim 6.4.4 If F is a collection of trapdoor permutations then the probabilistic encryption $PE = (G, E, D)$ is indistinguishably secure.

Proof: Suppose that (G, E, D) is not indistinguishably secure. Then there is a polynomial Q, a PTM A and a message space algorithm M such that for infinitely many k, $\exists m_0, m_1 \in M(1^k)$ with,

$$\Pr[A(1^k, i, m_0, m_1, c) = j | m_j \in \{m_0, m_1\}, c \in E(i, m_j))] > \frac{1}{2} + \frac{1}{Q(k)}$$

where the probability is taken over the coin tosses of A, $(i, t_i) \in G(1^k)$, the coin tosses of E and $m_j \in \{m_0, m_1\}$. In other words, A says 0 more often when c is an encryption of m_0 and says 1 more often when c is an encryption of m_1.

Define distributions $D_j = E(i, s_j)$ for $j = 0, 1, \ldots, k$ where $k = |m_0| = |m_1|$ and such that $s_0 = m_0, s_k = m_1$ and s_j differs from s_{j+1} in precisely 1 bit.

Let $P_j = \Pr[A(1^k, i, m_0, m_1, c) = 1 | c \in D_j = E(i, s_j)]$.

Then $\frac{1}{2} + \frac{1}{Q(k)} < \Pr[A \text{ chooses } j \text{ correctly}] = (1 - P_0)(\frac{1}{2}) + P_k(\frac{1}{2})$.

Hence, $P_k - P_0 > \frac{2}{Q(k)}$ and since $\sum_{j=0}^{k-1}(P_{j+1} - P_j) = P_k - P_0$, $\exists j$ such that $P_{j+1} - P_j > \frac{2}{Q(k)k}$.

Now, consider the following algorithm B which takes input $i, f_i(y)$ and outputs 0 or 1. Assume that s_j and s_{j+1} differ in the l^{th} bit; that is, $s_{j,l} \neq s_{j+1,l}$ or, equivalently, $s_{j+1} = \bar{s_j}$.

B runs as follows on input $i, f_i(y)$:

(1) Choose $y_1, \ldots, y_k$ such that $B_i(y_r) = s_{j,r}$ for $r = 1, \ldots, k$.

(2) Let $c = f_i(y_1), \ldots, f_i(y), \ldots, f_i(y_k)$ where $f_i(y)$ has replaced $f_i(y_l)$ in the l^{th} block.

(3) If $A(1^k, i, , m_0, m_1, c) = 0$ then output $s_{j,l}$.
If $A(1^k, i, , m_0, m_1, c) = 0$ then output $s_{j+1,l} = \bar{s}_{j,l}$.

Note that $c \in E(i, s_j)$ if $B_i(y) = s_{j,l}$ and $c \in E(i, s_{j+1})$ if $B_i(y) = s_{j+1,l}$.

Thus, in step 3 of algorithm B, outputting $s_{j,l}$ corresponds to A predicting that c is an encoding of s_j; in other words, c is an encoding of the string nearest to m_0.

Claim. $\Pr[B(i, f_i(y)) = B_i(y)] > \frac{1}{2} + \frac{1}{Q(k)k}$

Proof:

$$\begin{aligned}
\Pr[B(i, f_i(y)) = B_i(y)] &= \Pr[A(1^k, i, m_0, m_1, c) = 0 | c \in E(i, s_j)] \Pr[c \in E(i, s_j)] \\
&\quad + \Pr[A(1^k, i, m_0, m_1, c) = 1 | c \in E(i, s_{j+1})] \Pr[c \in E(i, s_{j+1})] \\
&\geq (1 - P_j)(\tfrac{1}{2}) + (P_{j+1})(\tfrac{1}{2}) \\
&= \tfrac{1}{2} + \tfrac{1}{2}(P_{j+1} - P_j) \\
&> \tfrac{1}{2} + \tfrac{1}{Q(k)k} \quad \square
\end{aligned}$$

Thus, B will predict $B_i(y)$ given i, $f_i(y)$ with probability better than $\frac{1}{2} + \frac{1}{Q(k)k}$. This contradicts the assumption that $B_i(y)$ is hard core for $f_i(y)$.

Hence, the probabilistic encryption $PE = (G, E, D)$ is indistinguishably secure.

∎

In fact, the probabilistic encryption $PE = (G, E, D)$ is also semantically secure. This follows from the fact that semantic and indistinguishable security are equivalent.

6.4.4 Efficient Probabilistic Encryption

How efficient are the probabilistic schemes? In the schemes described so far, the ciphertext is longer than the cleartext by a factor proportional to the security parameter. However, it has been shown [31, 35] using later ideas on pseudo-random number generation how to start with trapdoor functions and build a probabilistic encryption scheme that is polynomial-time secure for which the ciphertext is longer than the cleartext by only an additive factor. The most efficient probabilistic encryption scheme is due to Blum and Goldwasser [35] and is comparable with the RSA deterministic encryption scheme in speed and data expansion. Recall, that private-key encryption seemed to be much more efficient. Indeed, in practice the public-key methods are often used to transmit a secret session key between two participants which have never met, and subsequently the secret session key is used in conjunction with a private-key encryption method.

We first describe a probabilistic public key cryptosystem based on any trapdoor function collection which suffers only from a small additive bandwidth expansion.

As in the previous probabilistic encryption PE, we begin with a collection of trapdoor permutations $F = \{f_i : D_i \to D_i\}$ with hard core predicates $B = \{B_i : D_i \to \{0,1\}$. For this section, we consider that $D_i \subseteq \{0,1\}^k$, where $k = |i|$.

Then EPE $= (G, E, D)$ is our PKC based on F with:

Key Generation: $G(1^k) = S_1(1^k) = (i, t_i)$. The public key is i, and the secret key is t_i.

Encryption Algorithm: To encrypt m, $E(i, m)$ runs as follows, where $l = |m|$:

- **(1)** Choose $r \in D_i$ at random.
- **(2)** Compute $f_i(r), f_i^2(r), \ldots, f_i^l(r)$.
- **(3)** Let $p = B_i(r)B_i(f_i(r))B_i(f_i^2(r)) \ldots B_i(f_i^{l-1}(r))$.
- **(4)** Output the ciphertext $c = (p \oplus m, f_i^l(r))$.

Decryption Algorithm: To decrypt a ciphertext $c = (m', a)$, $D(t_i, c)$ runs as follows, where $l = |m'|$:

- **(1)** Compute r such that $f_i^l(r) = a$. We can do this since we can invert f_i using the trapdoor information, t_i, and this r is unique since f_i is a permutation.
- **(2)** Compute the pad as above for encryption: $p = B_i(r)B_i(f_i(r)) \ldots B_i(f_i^{l-1}(r))$.
- **(3)** Output decrypted message $m = m' \oplus p$.

To consider the efficiency of this scheme, we note that the channel bandwidth is $|c| = |m| + k$, where k is the security parameter as defined above. This is a significant improvement over the $|m| \cdot k$ bandwidth achieved by the scheme proposed in the previous lecture, allowing improvement in security with only minimal increase in bandwidth.

If C_{i1} is the cost of computing f_i, and C_{i2} is the cost of computing f_i^{-1} given t_i, then the cost of encryption is $|m| \cdot C_{i1}$, and the cost of decryption is $|m| \cdot C_{i2}$, assuming that the cost of computing B_i is negligible.

Another interesting point is that for all functions currently conjectured to be trapdoor, even with t_i, it is still easier to compute f_i than f_i^{-1}, that is, $C_{i1} < C_{i2}$, though of course, both are polynomial in $k = |i|$. Thus in EPE, if it is possible to compute f_i^{-l} more efficiently than as l compositions of f_i^{-1}, then computing $r = f_i^{-l}(a)$, and then computing $f_i(r), f_i^2(r), \ldots, f_i^{l-1}(r)$ may reduce the overall cost of decryption. The following implementation demonstrates this.

6.4.5 An implementation of EPE with cost equal to the cost of RSA

In this section, we consider a particular implementation of EPE as efficient as RSA. This uses for F a subset of Rabin's trapdoor functions which were introduced in Lecture 5. Recall that we can reduce Rabin's functions to permutations if we only consider the Blum primes, and restrict the domain to the set of quadratic residues. In fact, we will restrict our attention to primes of the form $p \equiv 7 \bmod 8$.[1]

Let $\mathcal{N} = \{n | n = pq; |p| = |q|; p, q \equiv 7 \bmod 8\}$. Then let $F = \{f_n : D_n \longrightarrow D_n\}_{n \in \mathcal{N}}$, where $f_n(x) \equiv x^2 \bmod n$, and $D_n = Q_n = \{y | y \equiv x^2 \bmod n\}$. Because $p, q \equiv 3 \bmod 4$, we have that f_n is a permutation on

[1] More recent results indicate that this additional restriction may not be necessary.

D_n. $B_n(x)$ is the least significant bit (LSB) of x, which is a hard core bit if and only if factoring is difficult, i.e., the Factoring Assumption from Lecture 5 is true. (This fact was stated, but not proven, in Lecture 7.)

Then consider the EPE (G, E, D), with:

Generation: $G(1^k) = (n, (p, q))$ where $pq = n \in \mathcal{N}$, and $|n| = k$. Thus n is the public key, and (p,q) is the secret key.

Encryption: $E(n, m)$, where $l = |m|$ (exactly as in general case above):

(1) Choose $r \in Q_n$ randomly.

(2) Compute $r^2, r^4, r^8, \ldots, r^{2^l} \pmod{n}$.

(3) Let $p = \text{LSB}(r)\text{LSB}(r^2)\text{LSB}(r^4)\ldots\text{LSB}(r^{2^{l-1}})$.

(4) Output $c = (m \oplus p, r^{2^l} \bmod n)$.

The cost of encrypting is $O(k^2 \cdot l)$.

Decryption: $D((p, q), c)$, where $c = (m', a), l = |m'|$ (as in general case above):

(1) Compute r such that $r^{2^l} \equiv a \bmod n$.

(2) Compute $p = \text{LSB}(r)\text{LSB}(r^2)\text{LSB}(r^4)\ldots\text{LSB}(r^{2^{l-1}})$.

(3) Output $m = m' \oplus p$.

Since $p, q \equiv 7 \bmod 8$, we have $p = 8t + 7$ and $q = 8s + 7$ for some integers s, t. Recall from Lecture 3 that if p is prime, the *Legendre symbol* $\mathbf{J}_p(a) = a^{\frac{p-1}{2}} \equiv 1 \bmod p$ if and only if $a \in Q_p$, Since $a \in Q_n$, we also have $a \in Q_p$. Thus we can compute

$$a \equiv a \cdot a^{\frac{p-1}{2}} \equiv a^{1+4t+3} \equiv (a^{2t+2})^2 \pmod{p},$$

yielding, $\sqrt{a} \equiv a^{2t+2} \bmod p$. Furthermore, $a^{2t+2} = (a^{t+1})^2 \in Q_p$, so we can do this repeatedly to find $r_p \equiv \sqrt[2^l]{a} \equiv a^{(2t+2)^l} \bmod p$. (This is why we require $p \equiv 7 \bmod 8$.) Analogously, we can find $r_q \equiv \sqrt[2^l]{a} \equiv a^{(2s+2)^l} \bmod q$, and using the *Chinese Remainder Theorem* (Lecture 5), we can find $r \equiv \sqrt[2^l]{a} \bmod n$. The cost of decrypting in this fashion is $O(k^3 \cdot l)$.

However, we can also compute r directly by computing $u = (2t + 2)^l$ and $v = (2s + 2)^l$ first, and in fact, if the length of the messages is known ahead of time, we can compute u and v off-line. In any event, the cost of decrypting then is simply the cost of computing $a^u \bmod p$ and $a^v \bmod q$, using the Chinese Remainder Theorem, and then computing p given r, just as when encrypting. This comes out to $O(k^3 + k^2 \cdot l) = O(k^3)$ if $l = O(k)$.

EPE Passes Indistinguishable Security

We wish to show that EPE also passes indistinguishable security. To do this we use the notion of *pseudo-random number generators* (PSRG) introduced in the chapter on pseudo random number generation. Note that $\text{PSRG}(\text{r}, \text{i}) = f_i^l(r) \circ B_i(r)B_i(f_i(r))B_i(f_i^2(r))\ldots B_i(f_i^{l-1}(r)) = a \circ p$ where p and a are generated while encrypting messages, ($\circ$ is the concatenation operator.) is a pseudo-random number generator. Indeed, this is the construction we used to prove the existence of PSRGs, given *one-way permutations*.

Certainly if the pad p were completely random, it would be impossible to decrypt the message since $m' = m \oplus p$ maps m' to a random string for any m. Since p is pseudo-random, it appears random to any PTM without further information i.e., the trapdoor t_i. However, the adversary does know $a = f_i^l(r)$, and we have to show that it cannot use this to compute p.

More precisely, we note that if there exists a PTM A that can distinguish between $(m \oplus p) \circ a$ and $(m \oplus R) \circ a$ where R is a completely random string from $\{0, 1\}^l$, then it can distinguish between $p \circ a$ and $R \circ a$. We can use this then, as a statistical test to check whether a given string is a possible output of PSRG, which contradicts the claim that PSRG is pseudo-random, and thus the claim that f_i is one-way. It is left as an exercise to express this formally.

6.4.6 Practical RSA based encryption: OAEP

Consider a sender who holds a k-bit to k-bit trapdoor permutation f and wants to transmit a message x to a receiver who holds the inverse permutation f^{-1}. We concentrate on the case which arises most often in cryptographic practice, where $n = |x|$ is at least a little smaller than k. Think of f as the RSA function.

Encryption schemes used in practice have the following properties: encryption requires just one computation of f; decryption requires just one computation of f^{-1}; the length of the enciphered text should be precisely k; and the length n of the text x that can be encrypted is close to k. Examples of schemes achieving these conditions are [156, 100].

Unfortunately, these are heuristic schemes. A provably secure scheme would be preferable. We have now seen several provably good asymmetric (i.e. public key) encryption schemes. The most efficient is the Blum-Goldwasser scheme [35]. But, unfortunately, it still doesn't match the heuristic schemes in efficiency. Accordingly, practioners are continuing to prefer the heuristic constructions.

This section presents a scheme called the OAEP (Optimal Asymmetric Encryption Padding) which can fill the gap. It was designed by Bellare and Rogaway [19]. It meets the practical constraints but at the same time has a security that can be reasonably justified, in the following sense. The scheme can be proven secure assuming some underlying hash functions are ideal. Formally, the hash functions are modeled as random oracles. In implementation, the hash functions are derived from cryptographic hash functions.

This random oracle model represents a practical compromise under which we can get efficiency with reasonable security assurances. See [20] for a full discussion of this approach.

OAEP is currently used in the SET (Secure Electronic Transactions) protocol of Visa and Mastercard. Various efforts to standardize it for public key encryption and distribution of session keys are also underway.

Simple embedding schemes and OAEP features

The heuristic schemes invariably take the following form: one (probabilistically, invertibly) embeds x into a string r_x and then takes the encryption of x to be $f(r_x)$.[2] Let's call such a process a *simple-embedding scheme.* We will take as our goal to construct provably-good simple-embedding schemes which allow n to be close to k.

OAEP is a simple embedding scheme that is bit-optimal (i.e., the length of the string x that can be encrypted by $f(r_x)$ is almost k). It is proven secure assuming the underlying hash functions are ideal. It achieves semantic security [87] and also a notion called "plaintext-aware encryption" defined in [19]. The latter notion is very strong, and in particular implies "ambitious" goals like chosen-ciphertext security and non-malleability [65] in the ideal-hash model.

The methods described here are simple and completely practical. They provide a good starting point for an asymmetric encryption/key distribution standard.

Now we briefly describe the basic scheme and its properties. We refer the reader to [19] for full descriptions and proofs of security.

The scheme

Recall k is the security parameter, f mapping k-bits to k-bits is the trapdoor permutation. Let k_0 be chosen such that the adversary's running time is significantly smaller than 2^{k_0} steps. We fix the length of the message to encrypt as let $n = k - k_0 - k_1$ (shorter messages can be suitably padded to this length). The scheme makes use of a "generator" $G\colon \{0,1\}^{k_0} \to \{0,1\}^{n+k_1}$ and a "hash function" $H\colon \{0,1\}^{n+k_1} \to \{0,1\}^{k_0}$. To encrypt $x \in \{0,1\}^n$ choose a random k_0-bit r and set

$$\mathcal{E}^{G,H}(x) \quad = \quad f(x0^{k_1} \oplus G(r).r \oplus H(x0^{k_1} \oplus G(r))).$$

The decryption $\mathcal{D}^{G,H}$ is defined as follows. Apply f^{-1} to the ciphertext to get a string of the form $a.b$ with $|a| = k - k_0$ and $|b| = k_0$. Compute $r = H(a) \oplus b$ and $y = G(r) \oplus a$. If the last k_1 bits of y are not all zero then reject; else output the first n bits of y as the plaintext.

[2] It is well-known that a naive embedding like $r_x = x$ is no good: besides the usual deficiencies of any deterministic encryption, f being a trapdoor permutation does not mean that $f(x)$ conceals all the interesting properties of x. Indeed it was exactly such considerations that helped inspire ideas like semantic security [87] and hardcore bits [36, 184].

The use of the redundancy (the 0^{k_1} term and the check for it in decryption) is in order to provide plaintext awareness.

Efficiency

The function f can be set to any candidate trapdoor permutation such as RSA [151] or modular squaring [146, 31]. In such a case the time for computing G and H is negligible compared to the time for computing f, f^{-1}. Thus complexity is discussed only in terms of f, f^{-1} computations. In this light the scheme requires just a single application of f to encrypt, a single application of f^{-1} to decrypt, and the length of the ciphertext is k (as long as $k \geq n + k_0 + k_1$).

The ideal hash function paradigm

As we indicated above, when proving security we take G, H to be random, and when we want a concrete scheme, G, H are instantiated by primitives derived from a cryptographic hash function. In this regard we are following the paradigm of [20] who argue that even though results which assume an ideal hash function do not provide provable security with respect to the standard model of computation, assuming an ideal hash function and doing proofs with respect to it provides much greater assurance benefit than purely *ad. hoc.* protocol design. We refer the reader to that paper for further discussion of the meaningfulness, motivation and history of this ideal hash approach.

Exact security

We want the results to be meaningful for practice. In particular, this means we should be able to say meaningful things about the security of the schemes for specific values of the security parameter (e.g., $k = 512$). This demands not only that we avoid asymptotics and address security "exactly," but also that we strive for security reductions which are as efficient as possible.

Thus the theorem proving the security of the basic scheme quantifies the resources and success probability of a potential adversary: let her run for time t, make q_{gen} queries of G and q_{hash} queries of H, and suppose she could "break" the encryption with advantage ϵ. It then provides an algorithm M and numbers t', ϵ' such that M inverts the underlying trapdoor permutation f in time t' with probability ϵ'. The strength of the result is in the values of t', ϵ' which are specified as functions of $t, q_{\text{gen}}, q_{\text{hash}}, \epsilon$ and the underlying scheme parameters k, k_0, n ($k = k_0 + n$). Now a user with some idea of the (assumed) strength of a particular f (e.g., RSA on 512 bits) can get an idea of the resources necessary to break our encryption scheme. See [19] for more details.

6.4.7 Enhancements

An enhancement to OAEP, made by Johnson and Matyas [101], is to use as redundancy, instead of the 0^{k_1} above, a hash of information associated to the key. This version of OAEP is proposed in the ANSI X9.44 draft standard.

6.5 Exploring Active Adversaries

Everything proven until this point has been concerned only with *passive* adversaries. But what happens if the adversaries are *active*? In particular, we consider the security of this scheme against a *chosen ciphertext attack*. In this model, we assume that our adversary has temporary access to the decoding equipment, and can use it to decrypt some cyphertexts that it chooses. Afterwards, the adversary sees the ciphertext it wants to decrypt without any further access to the decoding equipment. Notice that this is different from simply being able to generate pairs of messages and cyphertexts, as the adversary was always capable of doing that by simply encrypting messages of its choice. In this case, the adversary gets to choose the ciphertext and get the corresponding message from the decoding equipment.

We saw in previous sections that such an adversary could completely break Rabin's scheme. It is not known whether any of the other schemes discussed for PKC are secure in the presence of this adversary. However, attempts to provably defend against such an adversary have been made.

One idea is to put checks into the decoding equipment so that it will not decrypt cyphertexts unless it has evidence that someone knew the message (i.e., that the ciphertext was not just generated without knowledge of what the message being encoded was). We might think that a simple way to do this would be to require two distinct encodings of the same message, as it is unlikely that an adversary could find two separate encodings of the same message without knowing the message itself. Thus a ciphertext would be (α_1, α_2) where α_1, α_2 are chosen randomly from the encryptions of m.

Unfortunately, this doesn't work because if the decoding equipment fails to decrypt the ciphertext, the adversary would still gain some knowledge, i.e., that α_1 and α_2 do not encrypt the same message. For example, in the *probabilistic encryption* scheme proposed last lecture, an adversary may wish to learn the hard-core bit $B_i(y)$ for some unknown y, where it has $f_i(y)$. Given decoding equipment with the protection described above, the adversary could still discover this bit as follows:

(1) Pick $m \in \mathcal{M}(1^l)$, the message space, and let b be the last bit of m.

(2) Pick $\alpha_1 \in E(i, m)$ randomly and independently.

(3) Recall that $\alpha_1 = (f_i(x_1), f_i(x_2), \ldots, f_i(x_l))$, with x_j chosen randomly from D_i for $j = 1, 2, \ldots, l$. Let $\alpha_2 = (f_i(x_1), \ldots, f_i(x_{l-1}), f_i(y))$.

(4) Use the decoding equipment on $c = (\alpha_1, \alpha_2)$. If it answers m, then $B_i(y) = b$. If it doesn't decrypt c, then $B_i(y) = \overline{b}$.

What is done instead uses the notion of *Non-Interactive Zero-Knowledge Proofs* (NIZK) [37, 130]. The idea is that anyone can check a NIZK to see that it is correct, but no knowledge can be extracted from it about what is being proved, except that it is correct. Shamir and Lapidot have shown that if trapdoor functions exist, then NIZKs exist. Then a ciphertext will consist of three parts: two distinct encodings α_1, α_2 of the message, and a NIZK that α_1 and α_2 encrypt the same message. Then the decoding equipment will simply refuse to decrypt any ciphertext with an invalid NIZK, and this refusal to decrypt will not give the adversary any new knowledge, since it already knew that the proof was invalid.

CHAPTER 7

Message authentication

Data authentication is used to provide data integrity and proof of origin. Thus, it provides a means for party A to send party B a message in such a way that if the message is modified in route, B will detect this. In the private key case, the tool enabling data authentication is a message authentication scheme. This chapter examines both the notion of message authentication and its implementations.

7.1 Message authentication schemes

Parties A and B share a key K and communicate across a line on which is present an *active* adversary A. This adversary can modify data in transit or try to spoof its origin. (It can also read data on the line.) This can be quite dangerous. For example say you have bought something, and in order to be paid the seller sends you his or her bank account number. The adversary modifies this to be her bank account number, so she gets paid instead. Similarly, say you are buying goods and send the shop your delivery address electronically. But the adversary modifies it to her delivery address, and she gets the goods, for free.

Notice the difference between privacy and authenticity. We are not concerned here with whether the adversary knows the content of the data. She is welcome to know that. What we want is that any attempt to modify it is detected. We are talking of active adversaries who inject messages on the line from their own machines, or modify messages in route. Of course (typically) in practice we want both privacy and authenticity. If so we use (in the private key case) both a private key encryption scheme (as discussed in Chapter 5) and a message authentication scheme as we discuss here. One has to be a little careful as to how exactly to use them in conjunction, as we will discuss later, but it is not hard.

It is important to understand that authenticity and privacy are entirely different things, and are achieved in different ways. There is a quite widespread fallacy in parts of the literature that authenticity is achieved by encryption. Not only is this false, it is just the wrong viewpoint. Authenticity, as a goal, has nothing to do with encryption.

The tool to achieving authenticity of data in the private key setting is called a message authentication scheme. It will enable the legitimate parties to detect any attempt at tampering with the data. It is specified by two algorithms, the tagging algorithm Tag and the verification algorithm Vf. When the sender A wants to send M in an authenticated way she computes $\sigma = Tag(K, M)$ (we write also $Tag_K(M)$). Then she attaches this tag to the message before transmission; namely, she transmits (M, σ). Notice that the message is sent in the clear: there is no privacy here! Also notice the transmission is longer than the original message by the length of the tag σ. Upon receiving a pair (M', σ') purporting to be from A, the receiver B verifies the authenticity of the tag by applying the verification algorithm. Namely he computes $Vf(K, (M', \sigma'))$, which we write also as $Vf_K(M', \sigma')$. This value is a bit, and he and accepts if and only if this value is one.

Notice that both the tagging and the verification process depend on the shared secret key K.

We want of course that if the adversary does not try to tamper with the data then the verification should succeed. That is, correct tags should be accepted. For security, what we want is that if the adversary tampers with the message M (ie. changes even one bit in it) this will be detected: the verification function will output zero, because it will be infeasible for the adversary to modify the tag so that it is a correct tag of the modified message. But we are getting ahead of ourselves. Let us first summarize the components of the scheme.

Definition 7.1.1 A message authentication scheme is a pair (Tag, Vf) of probabilistic, polynomial time algorithms together with an indication of the key length κ. The first algorithm is calling the tagging algorithm or the MACing algorithm, and the second is called the verification algorithm. We require that if $\sigma = Tag_K(M)$ then $Vf_K(M, \sigma) = 1$.

This is the simplest case. In addition, either algorithm may depend on state information (such as a counter) or an explicit security parameter. (When no explicit security parameter is present, the length of the shared key functions as the security parameter.)

In many cases, the tagging function will be deterministic. In such a case, verification can be accomplished by re-computing the tag and checking it equals the transmitted tag σ. We call this a message authentication code, or MAC.

Definition 7.1.2 A message authentication code is a deterministic, polynomial time computable function MAC such that (MAC, Vf) is a message authentication scheme, where $Vf_K(M, \sigma)$ is defined to compute $MAC_K(M)$ and output one if and only if this value is equal to σ.

In some abuse of terminology, the tag σ is often called the MAC, and furthermore a message authentication scheme may be called a MAC even if the tagging function is not deterministic. You'll get used to it.

7.2 Some issues in security

The goal of an authentication scheme is to detect any attempt by the adversary to modify the transmitted data. What we are afraid of is that the adversary can produce messages that the receiver accepts. That is, A can produce M, σ such that $Vf_f(M, \sigma) = 1$, but M, σ was not sent by the sender. This is called a *forgery*.

Perhaps the first question one might ask is whether one should try to gauge the value of the forgery to the adversary, for example by asking what is the content of the message. For example, say the messages are expected to have certain formats, and the forgery is just a random string. Should this really be viewed as a forgery? The answer is *yes*. We have seen this general issue before. It would be unwise to make assumptions about how the messages are formatted or interpreted. Good protocol design means the security is guaranteed no matter what is the application. Accordingly we view the adversary as successful if she produces (M, σ) such that the sender never transmitted M and $Vf_K(M, \sigma) = 1$.

In some discussions of security in the literature, the adversary's goal is viewed as being to recover the shared secret key K. Certainly if she could do this, it would be a disaster, since she could forge anything. It is important to understand, however, that she might be able to forge without recovering the key. Thus, we are here making the notion of adversarial success more liberal: what counts is forgery, not key recovery. So a secure scheme has a stronger guarantee of security.

In making forgeries we must consider various attacks, of increasing severity. The simplest setting is that the sender has not yet sent any transmissions, and the adversary may simply attempt to concoct a pair (M, σ) which passes the verification test, namely such that $Vf_K(M, \sigma) = 1$. This is called a no-message attack.

However, the adversary also has the ability to see what is transmitted. She might try to make forgeries based on this information. So suppose the sender sends the transmission (M, σ) consisting of some message M and its legitimate (correct) tag σ. The receiver will certainly accept this.

At once, a simple attack comes to mind. The adversary can just copy this transmission. That is, she stores (M, σ), and at some later time re-transmits it. If the receiver accepted it the first time, he will do so again. This is called a replay attack.

Is this a valid forgery? In real life it probably should be so considered. Say the first message was "Transfer \$1000 from my account to the account of E." E suddenly sees a way of enriching herself. She keeps replaying this message and her bank balance increases.

It is imprtant to protect against replay attacks. But for the moment we will not try to do this. We will say that a replay is NOT a valid forgery. To be valid, a forgery must be of a message M which was *not* transmitted by the sender. We will that later we can achieve protection against replay by addition of time stamps or counters. This make the problem cleaner and allows greater *modularity* in protocol design. Try to cut up the problem into logical parts and solve them one by one. Accordingly from now on don't regard replay as a valid attack.

So if the adversary wants to be successful, it must somehow use the valid transmission (M, σ) to concoct a pair (M', σ') such that $M \neq M'$ but $Vf_K(M', \sigma') = 1$. If she can do this, we say she is successful. Thus, we have a very *liberal* notion of adversarial success. So when we say a scheme is secure, it is secure in a very strong sense.

We have allowed the adversary to see an example message. Of course, it may see more than one, and forgery must still be hard. Now another issue arises. In many settings, it is possible for the adversary to influence the choice of legitimate messages being tagged. In the worst case, we can think the adversary herself chooses these messages. This is called a chosen plaintext attack.

At first glance a chosen plaintext attack may seem unrealisticaly strong. There are two arguments against this view. First, we will see examples where such an attack is quite realistic. Second, recall our general principles. We want to design schemes which are as secure as possible. We should not make assumptions about how the adversary behaves, even if in some specific setting such assumptions are reasonable, because we want a general primitive which can be used in all settings.

Let us now look at some schemes.

7.3 Example schemes and their analysis

We follow the expository pattern of Chapter 5 by now discussing some example message authentication schemes. As in that chapter, the first thing to discuss is primitives on which to base the schemes. For this discussion we refer the reader to Section 5.3.1. Following that discussion, we will work with a finite family of functions of input length l and output length L, which we will think of as random, but later instantiate via a pseudorandom function family. In particular, think now of being in the shared random function model, so that the private key of the parties is a random function f of l bits to L bits.

7.3.1 Simple examples

Look at the data M as a sequence of blocks, $M = M_1 \ldots M_n$, where each block has length l, the length of the input to f. Now let

$$Tag_f^1(M_1 \ldots M_n) = f(M_1) \oplus \cdots \oplus f(M_n) \ .$$

The verifier receives (M, σ) and recomputes the tag to check that it is accurate. That is,

$Vf_f^1(M, \sigma')$
If $\sigma' = f(M_1) \oplus \cdots \oplus f(M_n)$ then accept else reject

Can the adversary forge this tag? The value $Tag_f^1(M)$ is random and uniformly distributed. How could it compute it?

Let's look at various levels of attacks. First, assume the adversary has seen no valid message-tag pairs. Say $n = 2$. The adversary sends $(M_1 M_1, 0^L)$. This is a valid forgery. So even in the absence of seeing valid data it can forge. Thus, forgery is possible under a *no message* attack.

Now suppose the adversary sees some valid data. For example $(M_1 M_2, \sigma)$ with $\sigma = f(M_1) \oplus f(M_2)$ goes by. What can the adversary do? It could permute message blocks in the seen message. The tag remains the same. So $(M_2 M_1, \sigma)$ is a forgery. This may be a more effective attack, especially if the adversary can influence the choice of M.

Let's try to fix it. Now divide the message into $l/2$ size blocks, which for notational simplicity we still denote $M_1 \ldots M_n$. Also let $\langle i \rangle$ denote the binary representation of integer i as an $l/2$ bit string. Example: if $l = 64$ then $\langle 2 \rangle = 0^{30}10$.

For each i note that $\langle i \rangle . M_i$ is a l bit string, where the symbol . denotes concatenation. The new scheme is

$$Tag_f^2(M_1 \ldots M_n) = f(\langle 1 \rangle . M_1) \oplus \cdots f(\langle n \rangle . M_n) \ .$$

Again the verifier receives (M, σ) and recomputes the tag to check that it is accurate.

Permuting message blocks no longer works. So is this a good scheme?

The attack this time is more sophisticated. It is a chosen-message attack. If you just see data going by it doesn't help much. But if you see several pieces of data bearing specific relations to each other you can mount an attack. Like this:

$$\begin{array}{rcl} \sigma_1 & = & f(\langle 1 \rangle . M_1) \oplus f(\langle 2 \rangle . M_2) \\ \sigma_2 & = & f(\langle 1 \rangle . M_1') \oplus f(\langle 2 \rangle . M_2) \\ \sigma_3 & = & f(\langle 1 \rangle . M_1) \oplus f(\langle 2 \rangle . M_2') \\ \hline \sigma & = & f(\langle 1 \rangle . M_1') \oplus f(\langle 2 \rangle . M_2') \end{array}$$

Namely, the adversary sees $(M_1 M_2, \sigma_1)$, $(M_1' M_2, \sigma_2)$ and $(M_1 M_2', \sigma_3)$. She adds the three tags to get $\sigma = \sigma_1 \oplus \sigma_2 \oplus \sigma_3$. But notice this is a correct tag for the message $M_1' M_2'$!

This requires that sender indeed send related data. But it can happen in practice.

Can we fix this too? The idea is to use counters, or a random string, which, recall, we were allowed, and lead to two different proposals.

7.3.2 XOR MACs

A family of message authentication schemes called XOR MACs was designed by [14]. The schemes make use of a finite PRF family F, viewed as a parameter of the scheme. Let l be the input length and L the output length of F. Let $f = F_K$ be the function specified by the shared key K. The message M is viewed as divided into $l/2$ bit blocks, $M = M_1 \ldots M_n$, and $\langle i \rangle$ denotes the $(l/2) - 1$ bit binary representation of a block index i. To define the schemes we first define the function XOR which takes a $l - 1$ bit string x and the message to return

$$\mathrm{XOR}_K(x, M) \quad = \quad f(0.x) \oplus f(1.\langle 1 \rangle . M_1) \oplus f(1.\langle 2 \rangle . M_2) \oplus \cdots \oplus f(1.\langle n \rangle . M_n) \ .$$

We are now ready to describe the scheme. There are actually two versions. One uses a counter, and the other, in order to avoid storing state information, uses a nonce.

Randomized XOR MAC

The scheme $(\text{R-XOR-Tag}^F, \text{R-XOR-Vf}^F)$ is the following. Given message M, the tagging function picks at random a $l - 1$ bit string r (called the seed) and outputs $(r, \mathrm{XOR}_K(r, M))$. Notice the seed is transmitted. Verification is done by re-computing the tag. Namely, $\text{R-XOR-Vf}_K(M, (r, \tau)) = 1$ iff $\mathrm{XOR}_K(r, M) = \tau$.

Counter based XOR MAC

The scheme $(\text{C-XOR-Tag}^F, \text{C-XOR-Vf}^F)$ is the following. The sender maintains a $l - 1$ bit counter C which is incremented at every message transmission. The tagging function is defined by $\text{C-XOR-Tag}_K(M) = (C, \mathrm{XOR}_K(C, M))$. Notice the counter is transmitted. After this tag is computed, the sender increments the counter. Verification is done by re-computing the tag. Namely, $\text{C-XOR-Vf}_K(M, (C, \tau)) = 1$ iff $\mathrm{XOR}_K(C, M) = \tau$. Notice the verifier (receiver) does not maintain a counter: to verify he just uses the counter value in the tag.

Obviously we can only go as long as it takes the counter to wrap around. That however is 2^{l-1} messages, which for $l = 64$ is 2^{63} which is a lot. We can always change the key after that. Or use a higher value of l and never worry.

Properties of XOR MACs

XOR MACs have nice properties. One is that they are parallelizable: the different applications of f can be made in parallel. This is useful when implementing the MAC in hardware, where f might be DES. Also these MACs are incremental in the sense of [13], in that if we have computed the tag for M and want to compute a tag of the message M' obtained by modifying just one block of M, we don't need to recompute a tag of M' from scratch: in four computations of f we can update the tag of M to get a tag for M'. However a problem with XOR MACs is that there is a loss of bandwidth.

We claim the above XOR MAC schemes are secure. But you ought to ask why. After all, these schemes are only slightly different from things we saw were insecure. But we have a proof. First, however, we need to state the definition, which we can now do quite easily. Then we will also look at other schemes.

7.4 Definition of Security

Let (Tag, Vf) be an arbitrary message authentication scheme. Let K be the shared key, chosen at random. To define security we consider the following game. It is played between the attacker A and the legitimate parties. Underlying the game is some fixed space $\mathcal{M}$ of valid messages.

First, the attacker is allowed a chosen message attack. This means it engages in a learning session with the sender:

- It picks a message $M^1 \in \mathcal{M}$ and asks the sender to provide the tag $\sigma^1 = Tag_K(M^1)$,m which the sender obligingly does.
- Given M^1 and σ^1 the adversary ruminates and comes up with a message $M^2 \in \mathcal{M}$ whose tag it would like to see. It asks the sender to tag M^2 and $\sigma^2 = Tag_K(M^2)$ is returned
- And so on, for a certain number q of examples, called the number of queries made.

Finally the adversary outputs a pair (M, σ) with $M \in \mathcal{M}$. We say this is a valid forgery if $M \notin \{M^1, \ldots, M^q\}$ and $Vf_K(M, \sigma) = 1$. We say the adversary is successful if it produces a valid forgery.

Note that the adversary is not given the key K! All it sees is legitimate tags. These are functions of the key, and if based on them it can learn the key, it will certainly be able to be successful. Yet, it might not learn K and still have a way of being successful. We saw examples of that above.

A more concise way to think about this is that the adversary is given an oracle for the function $Tag_K(\cdot)$. It may query this oracle some number of times, and is declared successful if following this it outputs a forgery.

We want to say the scheme is secure if the adversary, via this game, cannot produce a valid forgery. More precisely, we need to look at the *forgery probability* of an adversary A. This is the probability that the adversary is successful, namely that it output a valid forgery after the chosen message attack. (The probability here is over the choice of key K, any probabilistic choices that Tag or Vf might make, and the probabilistic choices, if any, that A makes.) Then we can ask that the forgery probability be low as long as A is not allowed too much computing time.

Formulated in terms of polynomial time adversaries we get the following. It is the natural adaptation to message authentication of the definition of security for digital signatures first given in [90]. The forgery probability here is measured as a function of the security parameter.

Definition 7.4.1 Message authentication scheme (Tag, Vf) is secure over message space $\mathcal{M}$ if any probabilistic polynomial time adversary has negligible forgery probability.

In the practical analysis of message authentication schemes we need to be more quantitative. We do this following [15].

Let us look more closely at the resources of the adversary. Certainly, there is her running time t. It is useful to separately quantify the number of queries she makes to the tagging oracle: call this q. In practice, these queries correspond to tagged messages sent by the legitimate sender, and it would make sense that getting these examples is more expensive than just computing on ones own. That is, we would expect q to be smaller than t. An investment of time t and q queries would bring the adversary some success probability. We let ϵ be a bound on this number to get the following quantitative version of the definition.

Definition 7.4.2 [15] Message authentication scheme (Tag, Vf) is (t, q, ϵ)-secure over message space $\mathcal{M}$ if for any adversary who runs in time t and makes $\leq q$ message queries, the forgery probability is at most ϵ.

Conversely, we say an adversary (t, q, ϵ)-breaks a message authentication scheme if it runs in time, makes at most q tagging queries, and succeeds in forgery with probability at least ϵ.

Example 7.4.3 To exercise the model, let see how the attacks we have discussed can be executed in this framework. Lets take the scheme (Tag^2, Vf^2) from above. The adversary will make three queries, namely the messages M_1M_2, $M_1'M_2$ and M_1M_2', getting a tag in response to each. Then it will add the tags to get σ and output (M, σ) as the forgery. Thus she (t, q, ϵ)-breaks the scheme with $q = 3$, $\epsilon = 1$, and $t = O(l)$.

We have allowed the adversary queries to the signing oracle. What about queries to the verification oracle? One can argue that they are not explicitly needed, because any adversary making them can be transformed into one that does not. (Very roughly, the reason is that we expect the answer to a verify query to be 0, meaning the query was not a valid message and tag, unless we are at the end of the game and the query is the valid forgery. Thus, verify queries can be predicted. There is a little more to it than this, but it works out.)

Although one can in principle omit verify queries from consideration, it may be good to consider them when one is doing the quantitative analysis, because the above discussed transformation can affect the security bounds. For this reason verify queries are considered explicitly in [14]. Here we omit them for simplicity.

7.5 Methods for secure message authentication

Here are various ways to get secure message authentication schemes.

7.5.1 Pseudorandom functions make good MACs

Let F be a family of pseudorandom functions with input length m and output length L. For an m-bit message M define $MAC_K(M) = F_K(M)$. This is a secure message authentication code. This was first pointed out by [82]. Below we state an exact security version of the theorem which is from [15].

Theorem 7.5.1 *Let F be a (t, q, ϵ)-secure family of pseudorandom functions with input length m and output length L. For an m-bit message M define $MAC_K(M) = F_K(M)$. Then MAC is a (t', q', ϵ')-secure message authentication code over message space $\{0,1\}^m$, for $t' = t - O(q(m + L))$, $q' = q - 1$, and $\epsilon' = \epsilon + 2^{-L}$.*

Notice that this requires us to have PRFs which take long inputs, namely of length the message length. This is one reason we discussed how to extend l bit to L bit finite PRFs to m bit to L bit ones in Section 4.6.4.

7.5.2 The CBC MAC

The most popular message authentication code in use is the CBC (Cipher Block Chaining) MAC. Let $f\colon \{0,1\}^l \to \{0,1\}^l$ be a function. Let $f^n\colon \{0,1\}^{nl} \to \{0,1\}^l$ be the function which on input $x_1 \ldots x_n$ outputs y_n where $y_i = f(y_{i-1} \oplus x_i)$ and $y_0 = 0^l$. If F is a finite family of functions with input length l and output length l then let F^n denote the family of functions in which the function indexed by key K is F_K^n. The new family has input length nl and output length l and is called the CBC of F.

When F is DES, we have the CBC construction used in practice, called the DES CBC MAC. This construction is both a US and an International Standard, extensively used in the banking and financial sectors. Its security is worth investigation.

We discussed the CBC construction in Section 4.6.4 and noted in Theorem 4.6.3 that if F is a PRF family then so is F^n. From Theorem 7.5.1 we can now conclude that the CBC construction makes a good MAC as long as the underlying functions are pseudorandom.

Theorem 7.5.2 [15] *Suppose F is a (t, q, ϵ)-secure family of PRFs. For an nl bit message M define $MAC_K(M) = F_K^n(M)$. This is a (t', q', ϵ')-secure message authentication code on nl-bit messages, where*

$$\begin{aligned} t' &= t - O(nql) \\ q' &= \frac{q}{n} - 1 \\ \epsilon' &= \epsilon + \frac{3(q')^2 n^2 + 1}{2^l} = \epsilon + \frac{3q^2 + 1}{2^l} . \end{aligned}$$

In particular, if F = DES we have an assessment of the strength of the DES CBC MAC in terms of the strength of DES as a PRF. Unfortunately as we discussed in Section 4.6.3, DES is not too strong as a PRF. We would be better of with stronger block ciphers.

On the other hand, this raises the question of whether the weakness is in our analysis rather than in the CBC MAC. Namely, could it be that the CBC MAC is actually much stronger than what we can show? It turns out not. The above analysis is quite good, and captures the strength of the CBC MAC quite effectively. This is evidenced by an attack, due to Preneel and Van Oorschot [144]. Roughly, they show how to break the CBC MAC with probability $q^2 n \cdot 2^{-l}$ using q queries. Their analysis looks at the case where the function used is truly random.

Proposition 7.5.3 [144] *Let $F = R(l, l)$ be the family of all functions of l bits to l bits and let $MAC_K(M) = F_K^n(M)$ for nl bit messages M. There is an attack on this scheme which using q queries succeeds in forgery with probability $\Omega(nq^2 \cdot 2^{-l})$.*

In contrast Theorem 7.5.2 showed that the forgery probability would not be more than $O(n^2 q^2) \cdot 2^{-l}$. Thus there is a slight gap (a factor of n) and we don't know which way it goes.

7.5.3 Security of XOR MACs

Refer to the specification of the XOR MAC schemes in Section 7.3.2. The analysis of XOR MACs exploits and illustrates the paradigm of protocol design with pseudorandom functions. Namely, we first work in the shared random function model. That is, we assume f is a random function. (Formally, F is the family of all functions of l bits to L bits with uniform distribution.) We analyze the scheme in this setting, proving an absolute bound on the forgery probability of the adversary. (The bound is different for the two schemes.) This analysis is not in a "computational" framework, but in an "information theoretic" one. There is no restriction on the computation time of the adversary. All that matters is the number of queries she makes.

A priori this result is a mathematical curiosity since it is derived assuming an ideal primitive (a random function) that cannot be realized in practice. But now we exploit the notion of pseudorandomness to see that when f is drawn from a pseudorandom function family, the forgery probability of a now computationally bounded adversary is not too much more than that in the ideal scheme.

On the technical level, all the work is in the first part, namely the information theoretic analysis. (It is done by combining some ideas of linear algebra and probability theory.) The reduction to the case of PRFs is quite standard.

We do the analysis quantitatively. The results quoted here enable the scheme user to know precisely what is the security of the MAC against forgery given a particular security of the PRF family. One of the interesting outcomes is to see that the security of the two XOR schemes is quite different. Also, both are "more secure" than the CBC MAC. When we are thinking of using DES as the PRF, this makes quite a difference.

Security of the randomized XOR MAC

The information theoretic lemma below considers the randomized XOR MAC scheme with $F = R(l, L)$, namely the shared function f between the users is a random function of l bits to L bits. Now it considers an adversary who makes q queries, and provides an absolute bound on its forgery probability.

Lemma 7.5.4 [14] *Consider the randomized XOR MAC scheme with $F = R(l, L)$ being the family of all functions of l bits to L bits. Let A be any adversary who makes q queries, each consisting of messages of at most n blocks. Then A's probability of forgery is at most $2^{-L} + 2q^2 \cdot 2^{-l}$.*

This theorem indicates that some forgery probability is inevitable even if the underlying family F is as good as possible. It turns out this is not a weakness of the analysis, but a fact. There is an attack that forges with roughly this probability. More precisely:

Proposition 7.5.5 [14] *Consider the randomized XOR MAC scheme with $F = R(l, L)$ being the family of all functions of l bits to L bits. Then there is a constant $c > 0$ such that for any q satisfying $q^2 \leq 2^l$, there is an adversary A who (t, q, ϵ)-breaks (R-XOR-Tag^F, R-XOR-Vf^F) with*

$$t = c(l + L)q \ ; \ \epsilon = \max\left\{ \left(1 - \frac{1}{e}\right) \cdot \frac{q^2 - 3q}{2 \cdot 2^l} \ , \ \frac{1}{2^L} \right\} .$$

Now we turn to the case of real interest. What happens when we use some (finite) PRF family F in the randomized XOR scheme? The following theorem provides the security in this case. It is this theorem that we actually use in practice when we implement the scheme with a particular block cipher.

Theorem 7.5.6 [14] *Suppose F is a (t, q, ϵ)-secure pseudorandom function family with input length l and output length L. Then the randomized XOR MAC scheme (R-XOR-Tag^F, R-XOR-Vf^F) is (t', q', ϵ')-secure for*

$$\begin{aligned} t' &= t - \Theta(n) \cdot (T_F + l + L) \\ q' &= \frac{q}{n+1} - 1 \\ \epsilon' &= 2^{-L} + \frac{2(q')^2}{2^l} + \epsilon \ . \end{aligned}$$

Here T_F is the time to compute a function $f \in F$ on an l-bit input, and n is an upper bound on the number of blocks in any chosen-message query made by the adversary or in the message in the forgery it outputs.

Proof: In this kind of proof, it is best to begin by ignoring the actual numerical relationships. What we will do is develop the proof in the natural way, and see the relations fall out. We will of course proceed by contradiction. Assume that the randomized XOR MAC scheme is not secure when using a function f drawn (randomly) from F. We will show then how to construct a distinguisher with high advantage, contradicting the pseudorandomness of F.

By assumption, there is an adversary A who (t', q', ϵ')-breaks the randomized XOR MAC scheme when it uses a function from F. What does this mean? We need to recall the model for MACs. This adversary is allowed to interact with the sender. (Formally, it is provided the tagging function as an oracle.) The adversary runs for at most t' steps, makes q' queries to the tagging oracle (sender), and finally outputs some forgery (M, σ), which, with probability ϵ', is correct. Meaning that M is different from the messages queried of the sender, and $Vf_f(M, \sigma) = 1$. The probability here is over the choice of f from F, over the coins used by the sender to make the tags (a feature of the XOR scheme) and any coins the adversary might toss.

Don't try to think, at this point, what are the values of t', q', ϵ'. Just think of ϵ' as "large," in particular, appreciably larger than $\delta = 2^{-L} + 2(q')^2/2^l$, the maximum success probability when f was random. This means that the adversary is successful much more often when f is from F than when f is from R. But that sounds fishy; it means that effectively, it can tell where f came from. As before, we want to capitalize on this in the following way. We are now in a completely different situation, namely someone has given us a function f, drawn either from F or from R, we know not which, and we want to decide which. Suppose we could somehow make this the underlying function for the MAC scheme, and let the adversary attack the MAC, and see whether she is successful or not. Then we claim we can tell where f came from. How? If A is successful, bet that f came from F, because that's where she is successful more often.

The difficulty is how we can set things up so that we turn one setting into another. Namely, we want to build a distinguisher D which given oracle access to a function f, outputs 0 or 1 according to whether the

adversary A is successful in forging the MAC which uses f. But the adversary A talks to a tagging function (sender). Where does that come from? Easy, the distinguisher "simulates" it. After all it has access to f, so it can do that. (It is important to note this is true due to the way in which the XOR MAC uses f, as we will see. It may not always be true.)

A is just some piece of code. D can "run" A. Upon doing so, A will come up with some message M^1 as a query to its tagging oracle. Namely, it is asking what is the tag of M^1. D provides the answer exactly as the sender would. It picks a $l-1$ bit string r^1 at random, computes $\tau^1 = \mathrm{XOR}_f(r^1, M^1)$, and returns (r^1, τ^1) to A. How does it compute $\mathrm{XOR}_f(r^1, M^1)$? It uses its oracle for f to compute all the f values involved. Namely, it first forms the string $0.r^1$, and goes to the oracle to get the value $f(0.r^1)$, and saves it. Then it forms $1.\langle 1\rangle.M_1^1$, where M_1^1 is the first $l/2$ bits of M^1, and gets the value $f(1.\langle 1\rangle.M_1^1)$ using its oracle, and xors it to the saved value. And so on. After having made a total of at most $n+1$ queries to f, it has the value $\mathrm{XOR}_f(r^1, M^1)$.

Given (r^1, τ^1), A will clunk away for a while and come up with another query M^2. Now, just as before, D computes a response (r^2, τ^2), and returns it, again using at most $n+1$ oracle queries. This goes on until A has asked for tags of q' messages. Finally, now, A outputs a pair $(M, (r, \tau))$ and claims it is a forgery. What D does is check up on this. It computes $\mathrm{XOR}_f(r, M)$ and sees if this is τ. If so, it is likely f was pseudorandom, so it outputs 1. Else it outputs 0.

Now we see that

$$\begin{aligned} P_D(F) &> \epsilon' \\ P_D(R) &\leq \delta = 2^{-L} + \frac{2(q')^2}{2^l}\ . \end{aligned}$$

The first is by our assumption on the success of A, which the distinguisher exactly captures. The second is the fact of Lemma 7.5.4. This tells us that $\mathsf{Adv}_D(F) > \epsilon' - \delta$. Now, we want a contradiction. We will get this as long as $\epsilon' - \delta \geq \epsilon$ and we can ensure that D runs in time at most t and makes at most q queries. So this tells us we better set $\epsilon' = \epsilon + \delta$, which is exactly what the theorem did. Next, lets tackle the query count of D. It makes at most $n+1$ queries to f to answer each tagging query of A, plus it makes another $n+1$ at the end to check that the forgery is correct, a total of $(q'+1)(n+1)$ queries to f. We want this to be q, so just solve the equation $(q'+1)(n+1) = q$ for q' and see that you get what is claimed in the theorem. Finally, we turn to the running time of D.

D runs A, which runs for time t'. Plus it invokes f a total of q times. But here we note that in the experiment of running A in its real setting, where it talks to the sender, it is also that case that the same number of f computations were made. Our convention is that they were counted as part of the time for A. So we have effectively already accounted for the computations of f in the query process. Plus there is the $n+1$ computations of f in the verification, and, over and above that, D only has to do some book-keeping. This yields the claimed relation. ∎

Security of the counter based XOR MAC

We proceed analogously to the above. The interesting thing is that in the information theoretic case we straight away see a drastic improvement in security. Namely:

Lemma 7.5.7 [14] *Consider the counter based XOR MAC scheme with $F = R(l, L)$ being the family of all functions of l bits to L bits. Let A be any adversary who makes $q < 2^{l-1}$ queries, each consisting of messages of at most n blocks. Then A's probability of forgery is at most 2^{-L}.*

Notice that the bound on the forging probability does not depend on q (as long as $q < 2^{l-1}$ as assumed) or even on the input length l of the function! It depends only on L.

As above, it turns out this bound is tight. We won't bother to state it as a proposition because it is quite obvious. Now here is the security in the computational setting.

Theorem 7.5.8 [14] *Suppose F is a (t, q, ϵ)-secure pseudorandom function family with input length l and output length L. Then the counter based XOR MAC scheme (C-XOR-Tag^F, C-XOR-Vf^F) is (t', q', ϵ')-secure for*

$$\begin{aligned} t' &= t - \Theta(n) \cdot (T_F + l + L) \\ q' &= \frac{q}{n+1} - 1 \\ \epsilon' &= 2^{-L} + \epsilon \ . \end{aligned}$$

Here T_F is the time to compute a function $f \in F$ on an l-bit input, and n is an upper bound on the number of blocks in any chosen-message query made by the adversary or in the message in the forgery it outputs.

Thus the counter based scheme is considerably more secure than the randomized scheme. Note also both are more secure than the CBC MAC.

7.5.4 Hash then tag

There is a one general paradigm that is worth mention. Suppose we have a message authentication scheme (Tag, Vf) which applies to messages of some fixed length l. (For example, $l = 160$). We want a message authentication scheme which applies to messages in $\{0,1\}^*$. We can use a collision-resistant hash function returning l bits. Recall such a function $H\colon \{0,1\}^* \to \{0,1\}^l$ has the property that it is computationally infeasible to find messages M, M' such that $M \neq M'$ and $H(M) = H(M')$. (For example, one might use SHA-1, the secure hash algorithm.) Now, define the message authentication scheme (Tag^*, Vf^*) by $Tag^*_K(M) = Tag_K(H(M))$ and $Vf^*_K(M, \sigma) = Vf_K(H(M), \sigma)$. In other words, to tag, first hash the message to an l-bit string, then tag with the original scheme. To verify, check that $H(M)$ is correctly tagged under the original scheme. One can show that if (Tag, Vf) is secure and H is collision resistant then (Tag^*, Vf^*) is also secure (see for example [18]).

In practice this is often a good approach. The reason is that in software, hash functions like SHA-1 are faster than block ciphers like DES. We could implement (Tag, Vf) using one of the above block cipher based schemes, and then use the above transformation with H being SHA.

7.5.5 MACing with hash functions: HMAC

Recently there has been a surge of interest in MACing using *only* cryptographic hash functions like MD5 or SHA. It is easy to see why. The popular hash functions like MD5 and SHA-1 are faster than block ciphers in software implementation; these software implementations are readily and freely available; and the functions are not subject to the export restriction rules of the USA and other countries.

The more difficult question is how best to do it. These hash functions were not originally designed to be used for message authentication. (One of many difficulties is that hash functions are not keyed primitives, ie. do not accommodate naturally the notion of secret key.) So special care must be taken in using them to this end.

A variety of constructions have been proposed and analyzed. (See Tsudik [177] for an early description of such constructions and Touch [176] for a list of Internet protocols that use this approach. Preneel and van Oorschot [144, 145] survey existing constructions and point out to some of their properties and weaknesses; in particular, they present a detailed description of the effect of birthday attacks on iterated constructions. They also present a heuristic construction, the MDx-MAC, based on these findings. Kaliski and Robshaw [104] discuss and compare various constructions. Performance issues are discussed in [176, 14].) Recently, one construction seems to be gaining acceptance. This is the HMAC construction of [11]. In particular HMAC was recently chosen as the mandatory to implement authentication transform for Internet security protocols and for this purpose is described in an Internet draft [109] and an upcoming RFC.

The HMAC construction

Let H be the hash function. For simplicity of description we may assume H to be MD5 or SHA-1; however the construction and analysis can be applied to other functions as well (see below). H takes inputs of any

length and produces l-bit output ($l = 128$ for MD5 and $l = 160$ for SHA-1). Let Text denote the data to which the MAC function is to be applied and let K be the message authentication secret key shared by the two parties. (It should not be larger than 64 bytes, the size of a hashing block, and, if shorter, zeros are appended to bring its length to exactly 64 bytes.) We further define two fixed and different 64 byte strings ipad and opad as follows (the "i" and "o" are mnemonics for inner and outer):

ipad = the byte 0x36 repeated 64 times
opad = the byte 0x5C repeated 64 times.

The function HMAC takes the key K and Text, and produces $\mathrm{HMAC}_K(\mathrm{Text}) =$

$$H(K \oplus \mathsf{opad}, H(K \oplus \mathsf{ipad}, \mathrm{Text})) \ .$$

Namely,

(1) Append zeros to the end of K to create a 64 byte string
(2) XOR (bitwise exclusive-OR) the 64 byte string computed in step (1) with ipad
(3) Append the data stream Text to the 64 byte string resulting from step (2)
(4) Apply H to the stream generated in step (3)
(5) XOR (bitwise exclusive-OR) the 64 byte string computed in step (1) with opad
(6) Append the H result from step (4) to the 64 byte string resulting from step (5)
(7) Apply H to the stream generated in step (6) and output the result

The recommended length of the key is at least l bits. A longer key does not add significantly to the security of the function, although it may be advisable if the randomness of the key is considered weak.

HMAC optionally allows truncation of the final output say to 80 bits.

As a result we get a simple and efficient construction. The overall cost for authenticating a stream Text is close to that of hashing that stream, especially as Text gets large. Furthermore, the hashing of the padded keys can be precomputed for even improved efficiency.

Note HMAC uses the hash function H as a black box. No modifications to the code for H are required to implement HMAC. This makes it easy to use library code for H, and also makes it easy to replace a particular hash function, such as MD5, with another, such as SHA-1, should the need to do this arise.

Security of HMAC

The advantage of HMAC is that its security can be justified given some reasonable assumptions about the strength of the underlying hash function.

The assumptions on the security of the hash function should not be too strong, since after all not enough confidence has been gathered in current candidates like MD5 or SHA. (In particular, we now know that MD5 is not collision-resistant [64]. We will discuss the MD5 case later.) In fact, the weaker the assumed security properties of the hash function, the stronger the resultant MAC construction.

We make assumptions that reflect the more standard existing usages of the hash function. The properties we require are mainly a certain kind of *weak* collision-freeness and some limited "unpredictability." What is shown is that if the hash function function has these properties the MAC is secure; the *only* way the MAC could fail is if the hash function fails.

The analysis of [11] applies to hash functions of the iterated type, a class that includes MD5 and SHA, and consists of hash functions built by iterating applications of a compression function CF according to the procedure of Merkle [124] and Damgård [57]. (In this construction a l-bit initial variable IV is fixed, and the output of H on text x is computed by breaking x into 512 bit blocks and hashing in stages using CF, in a simple way that the reader can find described in many places, e.g. [104].) Roughly what [11] say is that an attacker who can forge the HMAC function can, with the same effort (time and collected information), break the underlying hash function in one of the following ways:

(1) The attacker finds collisions in the hash function even when the IV is random and secret, or
(2) The attacker is able to compute an output of the compression function even with an IV that is random, secret and unknown to the attacker. (That is, the attacker is successful in forging with respect to the application of the compression function secretly keyed and viewed as a MAC on fixed length messages.)

The feasibility of any of these attacks would contradict some of our basic assumptions about the cryptographic strength of these hash functions. Success in the first of the above attacks means success in finding collisions, the prevention of which is the main design goal of cryptographic hash functions, and thus can usually be assumed hard to do. But in fact, even more is true: success in the first attack above is even *harder* than finding collisions in the hash function, because collisions when the IV is secret (as is the case here) is far more difficult than finding collisions in the plain (fixed IV) hash function. This is because the former requires interaction with the legitimate user of the function (in order to generate pairs of input/outputs from the function), and disallows the parallelism of traditional birthday attacks. Thus, even if the hash function is not collision-free in the traditional sense, our schemes could be secure.

Some "randomness" of hash functions is assumed in their usage for key generation and as pseudo-random generators. (For example the designers of SHA suggested that SHA be used for this purpose [74].) Randomness of the function is also used as a design methodology towards achieving collision-resistance. The success of the second attack above would imply that these randomness properties of the hash functions are very poor.

It is important to realize that these results are guided by the desire to have simple to state assumptions and a simple analysis. In reality, the construction is even stronger than the analyses indicates, in the sense that even were the hash functions found not to meet the stated assumptions, the schemes might be secure. For example, even the weak collision resistance property is an overkill, because in actuality, in our constructions, the attacker must find collisions in the keyed function without seeing any outputs of this function, which is significantly harder.

The later remark is relevant to the recently discovered collision attacks on MD5 [64]. While these attacks could be adapted to attack the weak collision-resistance property of MD5, they do not seem to lead to a breaking of HMAC even when used with MD5.

Resistance to known attacks

As shown in [144, 12], birthday attacks, that are the basis to finding collisions in cryptographic hash functions, can be applied to attack also keyed MAC schemes based on iterated functions (including also CBC-MAC, and other schemes). These attacks apply to most (or all) of the proposed hash-based constructions of MACs. In particular, they constitute the best known forgery attacks against the HMAC construction. Consideration of these attacks is important since they strongly improve on naive exhaustive search attacks. However, their practical relevance against these functions is negligible given the typical hash lengths like 128 or 160. Indeed, these attacks require the collection of the MAC value (for a given key) on about $2^{l/2}$ messages (where l is the length of the hash output). For values of $l \geq 128$ the attack becomes totally infeasible. In contrast to the birthday attack on key-less hash functions, the new attacks require interaction with the key owner to produce the MAC values on a huge number of messages, and then allow for no parallelization. For example, when using MD5 such an attack would require the authentication of 2^{64} blocks (or 2^{73} bits) of data using the same key. On a 1 Gbit/sec communication link, one would need 250,000 years to process all the data required by such an attack. This is in sharp contrast to birthday attacks on key-less hash functions which allow for far more efficient and close-to-realistic attacks [178].

7.5.6 Wegman-Carter approach

Another approach to the construction of message authentication schemes is to first hash the message under certain special kinds of hash functions (not cryptographic ones like MD5, SHA-1!) and then *encrypt* the output in a certain way. The hash functions are "universal" hash functions. The hash function is secret; it will be specified by the shared key between the users.

This approach begins with Wegman and Carter [183]. It has been much investigated, especially recently. See Rogaway [153] for a complete history as well as fast hashing schemes.

This approach is very promising. It has the potential to yield faster MACs than any of the above discussed approaches.

7.5.7 Minimizing assumptions for MACs

As with the other primitives of private key cryptography, the existence of secure message authentication schemes is equivalent to the existence of one-way functions. That one-way functions yield message authentication schemes follows from Theorem 4.6.2 and Theorem 7.5.1. The other direction is [99]. In summary:

Theorem 7.5.9 *There exists a secure message authentication scheme for message space* $\{0,1\}^*$ *if and only if there exists a one-way function.*

CHAPTER 8

Digital signatures

The notion of a *digital signature* may prove to be one of the most fundamental and useful inventions of modern cryptography. A signature scheme provides a way for each user to *sign* messages so that the signatures can later be *verified* by anyone else. More specifically, each user can create a matched pair of private and public keys so that only he can create a signature for a message (using his private key), but anyone can verify the signature for the message (using the signer's public key). The verifier can convince himself that the message contents have not been altered since the message was signed. Also, the signer can not later repudiate having signed the message, since no one but the signer possesses his private key.

By analogy with the paper world, where one might sign a letter and seal it in an envelope, one can sign an electronic message using one's private key, and then *seal* the result by encrypting it with the recipient's public key. The recipient can perform the inverse operations of opening the letter and verifying the signature using his private key and the sender's public key, respectively. These applications of public-key technology to electronic mail are quite widespread today already.

If the directory of public keys is accessed over the network, one needs to protect the users from being sent fraudulent messages purporting to be public keys from the directory. An elegant solution is the use of a *certificate* – a copy of a user's public key digitally signed by the public key directory manager or other trusted party. If user A keeps locally a copy of the public key of the directory manager, he can validate all the signed communications from the public-key directory and avoid being tricked into using fraudulent keys. Moreover, each user can transmit the certificate for his public key with any message he signs, thus removing the need for a central directory and allowing one to verify signed messages with no information other than the directory manager's public key. Some of the protocol issues involved in such a network organization, are discussed in the section on key distribution in these lecture notes.

8.1 The Ingredients of Digital Signatures

A *digital signature scheme* within the public key framework, is defined as a triple of algorithms (G, σ, V) such that

- Key generation algorithm G is a probabilistic, polynomial-time algorithm which on input a security parameter 1^k, produces pairs (P, S) where P is called a public key and S a secret key. (We use the notation $(P, S) \in G(1^k)$ indicates that the pair (P, S) is produced by the algorithm G.)
- Signing algorithm σ is a probabilistic polynomial time algorithm which is given a security parameter 1^k, a secret key S in range $G(1^k)$, and a message $m \in \{0,1\}^k$ and produces as output string s which we call the *signature of* m. (We use notation $s \in \sigma(1^k, S, m)$ if the signing algorithm is probabilistic and $s = \sigma(1^k, S, m)$ otherwise. As a shorthand when the context is clear, the secret key may be omitted and we will write $s \in \sigma(S, m)$ to mean meaning that s is the signature of message m.)

- Verification algorithm V is a probabilistic polynomial time algorithm which given a public key P, a digital signature s, and a message m, returns 1 (i.e "true") or 0 (i.e "false") to indicate whether or not the signature is valid. We require that $V(P, s, m) = 1$ if $s \in \sigma(m)$ and 0 otherwise. (We may omit the public key and abbreviate $V(P, s, m)$ as $V(s, m)$ to indicate verifying signature s of message m when the context is clear.)

- The final characteristic of a digital signature system is its security against a probabilistic polynomial-time forger. We delay this definition to later.

Note that if V is probabilistic, we can relax the requirement on V to accept valid signatures and reject invalid signatures with high probability for all messages m, all sufficiently large security parameters k, and all pairs of keys $(P, S) \in G(1^k)$. The probability is taken over the coins of V and S. Note also that the message to be signed may be plain text or encrypted, because the message space of the digital signature system can be any subset of $\{0,1\}^*$.

8.2 Digital Signatures: the Trapdoor Function Model

Diffie and Hellman [62] propose that with a public key cryptosystem (G, E, D) based on the trapdoor function model, user A can sign any message M by appending as a digital signature $D(M) = f^{-1}(M)$ to M where f is A's trapdoor public function for which A alone knows the corresponding trapdoor information. Anyone can check the validity of this signature using A's public key from the public directory, since $E(D(M)) = f^{-1}(f(M))$. Note also that this signature becomes invalid if the message is changed, so that A is protected against modifications after he has signed the message, and the person examining the signature can be sure that the message he has received that was originally signed by A.

Thus, in their original proposal Diffie and Hellman linked the two tasks of encryption and digital signatures. We, however, **separate** these two tasks. It turns out that just as some cryptographic schemes are suited for encryption but not signatures, many proposals have been made for *signature-only* schemes which achieve higher security.

The RSA public-key cryptosystem which falls in the Diffie and Hellman paradigm allows one to implement digital signatures in a straightforward manner. The private exponent d now becomes the *signing exponent*, and the signature of a message M i which falls in the Diffie and Hellman paradigm s now the quantity $M^d \bmod n$. Anyone can verify that this signature is valid using the corresponding public *verification exponent* e by checking the identity $M = (M^d)^e \bmod n$. If this equation holds, then the signature M^d must have been created from M by the possessor of the corresponding signing exponent d. (Actually, it is possible that the reverse happened and that the "message" M was computed from the "signature" M^d using the verification equation and the public exponent e. However, such a message is likely to be unintelligible. In practice, this problem is easily avoided by always signing $f(M)$ instead of M, where f is a standard public one-way function.)

Cast in our notation for digital signature schemes, the Diffie-Hellman proposal is the following triple of algorithms (G, σ, V):

- Key Generation: $G(1^k)$ picks pairs (f_i, t_i) from F where $i \in I \cap \{0,1\}^k$.

- Signing Algorithm: $\sigma(1^k, f_i, t_i, m)$ outputs $f_i^{-1}(m)$.

- Verification Algorithm: $V(f_i, s, m)$ outputs 1 if $f_i(s) = m$ and 0 otherwise.

We will consider the security of this proposal and others. We first define security for digital signatures.

8.3 Defining and Proving Security for Signature Schemes

A theoretical treatment of digital signatures security was started by Goldwasser, Micali and Yao in [92] and continued in [90, 16, 129, 154, 68].

8.3.1 Attacks Against Digital Signatures

We distinguish three basic kinds of attacks, listed below in the order of increasing severity.

- *Key-Only Attack:* In this attack the adversary knows only the public key of the signer and therefore only has the capability of checking the validity of signatures of messages given to him.
- *Known Signature Attack:* The adversary knows the public key of the signer and has seen message/signature pairs chosen and produced by the legal signer. In reality, this the minimum an adversary can do.
- *Chosen Message Attack:* The adversary is allowed to ask the signer to sign a number of messages of the adversary's choice. The choice of these messages may depend on previously obtained signatures. For example, one may think of a notary public who signs documents on demand.

For a finer subdivision of the adversary's possible attacks see [90].

What does it mean to successfully forge a signature?

We distinguish several levels of success for an adversary, listed below in the order of increasing success for the adversary.

- *Existential Forgery:* The adversary succeeds in forging the signature of one message, not necessarily of his choice.
- *Selective Forgery:* The adversary succeeds in forging the signature of some message of his choice.
- *Universal Forgery:* The adversary, although unable to find the secret key of the The forger, is able to forge the signature of any message.
- *Total Break :* The adversary can compute the signer's secret key.

Clearly, different levels of security may be required for different applications. Sometimes, it may suffice to show that an adversary who is capable of a known signature attack can not succeed in selective forgery, while for other applications (for example when the signer is a notary-public or a tax-return preparer) it may be required that an adversary capable of a chosen signature attack can not succeed even at existential forgery with non-negligible probability.

The security that we will aim at, in these notes are that with high probability a polynomial time adversary would not be able to even existentially forge in the presence of a chosen message attack.

We say that *a digital signature is secure* if an enemy who can use the real signer as "an oracle" can not in time polynomial in the size of the public key forge a signature for any message whose signature was not obtained from the real signer. Formally, let B be a black box which maps messages m to valid signatures, i.e , $V(P, B(m), m) = 1$ for all messages m. Let the forging algorithm F on input the public key P have access to B, denoted as $F^B(P)$. The forging algorithm runs in two stages: it first launches a chosen message attack, and then outputs a "new forgery" which is defined to be any message-signature pair such that the message was not signed before and that signature is valid. We require that for all forging algorithms F, for all polynomials Q, for all sufficiently large k, $Prob(V(P, s, m) = 1 : (P, S) \stackrel{R}{\leftarrow} G(1^k)\ ;\ (m, s) \stackrel{R}{\leftarrow} F^B(P)) \leq \frac{1}{Q(k)}$. The probability is taken over the choice of the keys $(P, S) \in G(1^k)$, the coin tosses of the forgery algorithm F, and the coins of B.

Diffie and Hellman's original proposal does not meet this strict definition of security; it is possible to existentially forge with just the public information: Choose an s at random. Apply the public key to s to produce $m = f(s)$. Now s is a valid signature of m.

Many digital signature systems have been proposed. For a fairly exhaustive list we refer to the paper [90] handed out.

We examine the security of three systems here.

8.3.2 The RSA Digital Signature Scheme

The first example is based on the RSA cryptosystem.

The public key is a pair of numbers (n, e) where n is the product of two large primes and e is relatively prime to $\phi(n)$, and the secret key is d such that $ed = 1 \bmod \phi(n)$. Signing is to compute $\sigma(m) = m^d mod n$. Verifying is to raise the signature to the power e and compare it to the original message.

Claim 8.3.1 RSA is universallly forgable under a chosen-message attack. (alternatively, existentially forgable under known message attack)

Proof: If we are able to produce signatures for two messages, the signature of the the product of the two messages is the product of the signatures. Let $m1$ and $m2$ be the two messages. Generate signatures for these messages with the black box: $\sigma(m1) = m1^d \bmod n$, $\sigma(m2) = m2^d \bmod n$. Now we can produce the signature for the product of these two messages: $\sigma(m1m2) = (m1m2)^d = m1^d m2^d = \sigma(m1)\sigma(m2) \bmod n$

To produce a signature for a message m, begin by choosing a random number $r \in 2n^*$. Now define $m1$ and $m2$ as follows: $m1 = mr \bmod n$, and $m2 = r^{-1} \bmod n$ Using the strategy above, we can find a signature for the product of these messages, which is the original message m, as follows: $m1m2 = (mr)r^{-1} = m$. ∎

8.3.3 El Gamal's Scheme

This digital signature system security relies on the difficulty of solving a a problem called the Diffie-Hellman-key-exchange (DHKE)problem, which is related to the discrete log problem. The DHKE problem is on input a prime p, a generator g, and $g^y, g^x \in Z_p^*$, compute output $g^{xy} \bmod p$. The best way currently known to solve the DHKE is to first solve the discrete log problem. Whether computing a discrete log is as hard as the Diffie-Hellman problem is currently an open question.

The following digital signature scheme is probabilistic. A close variant of it called DSS has been endorsed as a national standard.

Idea of the scheme:

- Public key: A triple (y, p, g), where $y = g^x \bmod p$, p is prime and g is a generator for Z_p^*.
- Secret key: x such that $y = g^x \bmod p$.
- Signing: The signature of message m is a pair (r, s) such that $0 \neq r, s \neq p-1$ and $g^m = y^r r^s \bmod p$.
- Verifying: Check that $g^m = y^r r^s \bmod p$ actually holds.

In order to generate a pair (r, s) which constitutes a signature, the signer begins by choosing a random number k such that $0 \neq k \neq p-1$ and $GCD(k, p-1) = 1$. Let $r = g^k (modp)$. Now we want to compute an s such that $g^m = y^r r^s = g^{xr+ks} modp$. In terms of the exponents, this relationship is $m = xr + ks(modp-1)$. Hence $s = (m - xr)^{k^{-1}} modp - 1$. The signature of m is the pair (r, s).

Clearly, If an attacker could solve the discrete logarithm problem, he could break the scheme completely by computing the secret key x from the information in the public file. Moreover, if an attacker finds k for one message, he can solve the discrete logarithm problem, so the pseudo random number generator employed to generate k's has to be of superior quality.

Claim 8.3.2 This scheme is existentially forgable in the presence of a known message attack.

Exercise.

Note on a key exchange protocol based on discrete log: It is interesting to note that it is possible for two people to exchange a secret key without prior secret meeting using the DL problem which is not known to yield a trapdoor function. This can be done by Persons A and B agree on a prime p and a generator g. Person A chooses a secret number x and sends $g^x (modp)$ to B. Person B chooses a secret number y and sends $g^y (modp)$ to A. Now each user can readily compute $g^{xy} (modp)$; let this be the shared secret key. It is not known if computing xy is as difficult as DLP.

8.3.4 Rabin's Scheme

Rabin [146] proposed a method where the signature for a message M was essentially the square root of M, modulo n, the product of two large primes. Since the ability to take square roots is provably equivalent to the ability to factor n, an adversary should not be able to forge any signatures unless he can factor n. For our purpose let's consider the variant of it when $n = pq$ and $p = q = 3 \bmod 4$, so that the signature is uniquely determined.

This argument assumes that the adversary only has access to the public key containing the modulus n of the signer. An enemy may break this scheme with an *active* attack by asking the real signer to sign $M = x^2 \bmod n$, where x has been chosen randomly. If the signer agrees and produces a square root y of M, there is half a chance that $\gcd(n, x - y)$ will yield a nontrivial factor of n — the signer has thus betrayed his own secrets! Although Rabin proposed some practical techniques for circumventing this problem, they have the effect of eliminating the constructive reduction of factoring to forgery.

Let us look at this in some detail.

This digital signature scheme is based on the difficulty of computing square roots modulo a composite number.

- Public key: $n = pq$
- Secret key: primes p, q
- Signing: $s = \sqrt{m} \bmod n$ (assume WLOG that all m are squares)
- Verification: Check that $s^2 = m \bmod n$.

Claim 8.3.3 This system is existentially forgable with key-only attack.

Proof: Choose a signature and square it to produce a corresponding message. ▌

Claim 8.3.4 The system is totally breakable in the face of a chosen message attack.

Proof: We know that if we can find two distinct square roots of a message, we can factor the modulus. Choose a value s and let $m = s^2$. Now s is a valid signature of m. Submit m to the black box. There is a one in two chance that it will produce the same signature s. If so, repeat this process. If not, we have both square roots of m and can recover the factors of n. ▌

Security when "Breaking" is Equivalent to Factoring

Given the insecurity of Rabin's scheme in the face of a chosen message attack, one might hypothesize that there exists no secure digital signature system based on factoring. That is, a scheme wherein:

- "Breaking" the scheme is equivalent to factoring.
- The signature scheme is secure against a chosen message attack.

False proof: We assume (1) and show that (2) is impossible. Since the first statement is that "breaking" the scheme is equivalent to factoring, we know that the following reduction must be possible on input of a composite number n.

- Generate a public key P.
- Produce a message m.
- Produce a valid signature $s \in \sigma(P, m)$ using the "breaker" algorithm. (Repeat these three steps up to a polynomial number of times.)
- Factor n.

Conclude that the system must be insecure in the face of a chosen message attack, since we can substitute the CMA for the "breaker" algorithm in step 3. QED

What is wrong with this argument? First, there is only a vague definition of the public information P; it need not contain the number n. Second, the CMA will always produce signatures with respect to fixed public information, whereas in the above reduction it may be necessary to use different public information in every call to the "breaker".

8.4 Probabilistic Signatures

Probabilistic techniques have also been applied to the creation of digital signatures. This approach was pioneered by Goldwasser, Micali, and Yao [92], who presented signature schemes based on the difficulty of factoring and on the difficulty of inverting the RSA function for which it is provably hard for the adversary to existentially forge using a known signature attack.

Goldwasser, Micali, and Rivest [90] have strengthened this result by proposing a signature scheme which is not existentially forgable under a chosen message attack. Their scheme is based on the difficulty of factoring, and more generally on the existence of claw-free trapdoor permutations (that is, pairs f_0, f_1 of trapdoor permutations defined on a common domain for which it is hard to find x, y such that $f_0(x) = f_1(y)$).

The scheme, as originally described,although attractive in theory, is quite inefficient. However, it can be modified to allow more compact signatures, to make no use of memory between signatures other than for the public and secret keys, and even to remove the need of making random choices for every new signature. In particular, Goldreich [81] has made suggestions that make the factoring-based version of this scheme more practical while preserving its security properties.

Bellare and Micali in [16] have shown a digital signature scheme whose security can be based on the existence of any trapdoor permutation (a weaker requirement than claw-freeness). Then Naor and Yung [129] have shown how, starting with any *one-way* permutation, to design a digital signature scheme which is secure against existential forgery by a chosen signature attack. Finally, Rompel [154] has shown how to sign given any one-way function. These works build on an early idea due to Lamport on how to sign a single bit in [112]. The idea is as follows. If f is a one-way function, and Alice has published the two numbers $f(x_0) = y_0$ and $f(x_1) = y_1$, then she can sign the message 0 by releasing x_0 and she can similarly sign the message 1 by releasing the message x_1. Merkle [123] introduced some extensions of this basic idea, involving building a tree of authenticated values whose root is stored in the public key of the signer.

We now proceed to describe in detail some of these theoretical developments.

8.4.1 Claw-free Trap-door Permutations

We introduce the notion of *claw-free trap-door permutations* and show how to construct a signature scheme assuming the existence of a claw-free pair of permutations.

Definition 8.4.1 [f-claw] Let f_0, f_1 be permutation over a common domain D. We say that (x, y, z) is *f-claw* if $f_0(x) = f_1(y) = z$.

Definition 8.4.2 [A family of claw-free permutations] A family $F = \{f_{0,i}, f_{1,i} : D_i \rightarrow D_i\}_{i \in I}$ is called *a family of claw-free trap-door permutations* if:

1. There exists an algorithm G such that $G(1^k)$ outputs two pairs (f_0, t_0), (f_1, t_1) where where t_i is the trapdoor information for f_i.
2. There exists PPT an algorithm that given f_i and $x \in D_i$ computes $f_i(x)$.
3. $\forall$ (inverting algorithm) I, there exists some negligible function ν_I such that for all sufficiently large k, $Prob(f_0(x) = f_1(y) = z : ((f_0, t_0), (f_1, t_1)) \stackrel{R}{\leftarrow} G(1^k) \ ; \ (x, y, z) \stackrel{R}{\leftarrow} I(f_0, f_1)) < \nu_I(k)$

The following observation shows that the existence of a pair of trap-door permutations does not immediately imply the existence of a claw-free permutation. For example, define a family of ("RSA") permutations by

$$f_{0,n}(x) \equiv x^3 \bmod n \quad f_{1,n}(x) \equiv x^5 \bmod n$$

($\gcd(x,n) = 1$, and $\gcd(15, \Phi(n)) = 1$). Since the two functions commute, it is easy to create a claw by choosing w at random and defining $x = f_{1,n}(w), y = f_{0,n}(w)$, and

$$z = f_{0,n}(x) = f_{1,n}(y) = w^{15} \bmod n$$

In general, the following question is

Open Problem 8.4.3 Does the existence of a family of trap-door permutations imply the existence of a family of claw-free trap-door permutations ?

The converse of the above is clearly true: Given a claw-free permutations generator, it is easy to generate a trap-door permutation. If $\{f_0, f_1\}$ is a pair of claw-free permutations over a common domain, that is, it is computationally infeasible to find a triple x, y, z such that $f_0(x) = f_1(y) = z$, then (f_0, f_0^{-1}) is trap-door. (Otherwise, give the inverting algorithm I, $z = f_1(y)$; z is also distributed uniformly over D, so with non-negligible probability, I can produce $x = f_0^{-1}(z)$. Therefore (x, y, z) is a claw, contradiction.)

8.4.2 Example: Claw-free permutations exists if factoring is hard

Let $n = pq$, where p and q are primes ($p, q \in H_k$) and $p \equiv 3 \bmod 8$, $q \equiv 7 \bmod 8$. Observe that about $1/16$ of odd prime pairs fit this requirement. Let QR_n denote the set of quadratic residues mod$n$.

We first note that:

1. $(\mathbf{J}_n(-1)) = +1$, but $-1 \notin QR_n$

2. $(\mathbf{J}_n(2)) = -1$, (and $2 \notin QR_n$) .

3. $x \in QR_n$ has exactly one square root $y \in QR_n$ (x is a Blum integer), but has four square root $y, -y, w, -w$ in general. Roots $w, -w$ have Jacobi symbol -1, y and $-y$ have Jacobi symbol $+1$.

We now define a family of pairs of functions, and prove, assuming the intractability of factoring, that it is a family of claw-free trap-door permutations over QR_n.
Define, for $x \in QR_n$:

$$f_{0,n}(x) = x^2 \bmod n \quad f_{1,n}(x) = 4x^2 \bmod n$$

It follows from the above notes that the functions $f_{0,n}$, $f_{1,n}$ are permutations of QR_n.
Claim: $\{f_{0,n}, f_{1,n}\}$ is claw-free.
Proof: Suppose that the pair is not claw-free. Assume $x, y \in QR_n$ satisfy

$$x^2 \equiv 4y^2 \bmod n$$

This implies that $(x - 2y)(x + 2y) \equiv 0 \bmod n$. However, checking the Jacobi symbol of both sides we have:

$$(\mathbf{J}_n(x)) = +1 \quad (\mathbf{J}_n(2y)) = (\frac{y}{n})(\frac{2}{n}) = -1 \quad (\mathbf{J}_n(-2y)) = (\frac{-1}{n}) = -1$$

That is, x is a quadratic residue, but $\pm 2y$ are not. Since $x \not\equiv \pm 2y \bmod n$ $\gcd(x \pm 2y, n)$ will produce a nontrivial factor on n. ∎

8.5 The signature Scheme

8.5.1 How to sign one bit

We first describe the basic building block of the signature scheme: signing one bit.

Let D be the common domain of the claw-free pair $\{f_0, f_1\}$, and assume x is selected randomly in D.

Public	Secret
$x \in D, f_0, f_1$	f_0^{-1}, f_1^{-1}

To sign the bit $b \in \{0,1\}$ let $s = \sigma(b) = f_b^{-1}(x)$.

To verify the signature s, check that $f_b(s) = x$.

Claim 8.5.1 The above scheme is existentially secured against Chosen Message Attack.

Proof: Suppose, by way of contradiction, that the scheme is not secure. That is, $\exists$ a forging algorithm $F^{CMA}(P)$ that can forge the signature (given the public information); F asks for the signature of b and ($\forall$ polynomial Q and infinitely many k's) can sign $\overline{b}$ correctly with probability $> 1/Q(k)$. To derive the contradiction, we design an algorithm that, given F^{CMA}, can make claws:

input: f_0, f_1.

output: x, y, z, such that $f_0(x) = f_1(y) = z$ (with probability $> 1/Q(k)$).

(1) Select randomly $x \in D$; flip a coin and put in the public file: $z = f_{coin}(x) \in D, f_0, f_1$. (Note that f_0, f_1 are permutations, so z is uniform in D).

(2) Run algorithm $F^{CMA}(P)$:

1. If F asks for signature of $b = \overline{coin}$, go back to *(1)*.
2. If F asks for signature of $b = coin$, answer with $x = f_b^{-1}(f_{coin}(x))$.

(3) By the assumption, F can produce now a signature for $\overline{b}$, $y = f_{\overline{b}}^{-1}(f_{coin}(x))$, i.e. $z = f_b(x) = f_{\overline{b}}(y)$. That is, we have a claw:

∎

8.5.2 How to sign a message

As before, D is the common domain of the claw-free pair $\{f_0, f_1\}$, and x is selected randomly in D.

Public	Secret
$x \in D, f_0, f_1$	f_0^{-1}, f_1^{-1}

For $x \in D$, we sign the *first* message m^1 by:

$$s^1 = \sigma(m^1) = f_{m^1}^{-1}(x)$$

and verify by:

$$V(s^1, m^1) = \begin{cases} 1 & \text{if } f_{m^1}(s^1) = x \\ 0 & \text{otherwise} \end{cases}$$

where, for $m^1 = m_1^1 m_2^1 \ldots m_k^1$:

$$f_{m^1}^{-1}(x) = f_{m_k^1}^{-1}(\ldots(f_{m_2^1}^{-1}(f_{m_1^1}^{-1}(x))))$$

$$f_{m^1}(x) = f_{m_1^1}(\ldots(f_{m_{k-1}^1}(f_{m_k^1}(x))))$$

Clearly f_m is a permutation on D, and is easy to compute. To sign the next message m^2, we apply the new permutation on the previous signature:

$$s^2 = \sigma(m^2) = (f_{m^2}^{-1}(s^1), m^1)$$

and verify by:

$$V(s^2, m^2) = \begin{cases} 1 & \text{if } f_{m^2}(s^2) = s^1 \text{ and } f_{m^1}(s^1) = x \\ 0 & \text{otherwise} \end{cases}$$

Notes:

1. With this scheme, the length of the signature grows linearly with the number of messages signed so far.
2. It is clearly easy to forge signatures for prefix of a message we have already seen. We therefore assume here that we pre-process the messages to be presented in a prefix-free encoding scheme. (i.e no messages is a prefix of another message).

Claim: The scheme is not existentially secure with respect to a Known Message Attack.

Proof: Assume $\exists F(H,P)$ that ($\forall$ polynomial Q and sufficiently large k), given the public information P and the history $H = ((m_1, \sigma(m_1)), \ldots, (m_l, \sigma(m_l)))$, for messages $m_1, m_2, ..., m_l$ selected by running $M(1^k)$, can find a message $\hat{m} \neq m_i$, $(1 \leq i \leq l)$, can produce a signature $\sigma(\hat{m})$ such that

$$Prob\{V(\sigma(\hat{m}), \hat{m}) = 1\} > \frac{1}{Q(k)}$$

where the probability is taken over all public files and coin tosses of F.

We now design an algorithm A that uses F to come up with a claw:

input: f_0, f_1.

output: a, b, c, such that $f_0(a) = f_1(b) = c$ (with probability $> 1/Q(k)$).

(1) Choose $m_1, m_2, \ldots, m_i \in \mathcal{M}(1^k), x \in_R D$. Let $z = f_{m_l}(\ldots(f_{m_1}(x)))$. Let $P = \{f_0, f_1, x\}$ be the public file. (Notice that z is also selected uniformly in D).

(2) Generate the history $H = (m_1, f_{m_1}(z)), \ldots (m_l, (f_{m_l}(\ldots(f_{m_1}(z)))))$, Denote $m = m_1 \circ m_2 \circ \ldots \circ m_l$, the string of all messages generated.

(3) Run the forging algorithm $F(H,P)$ to produce $(\hat{m}, \sigma(\hat{m}))$.

(4) With non negligible probability, $\sigma(\hat{m})$ is a valid signature; that is, "walking back" with $f_{\hat{m}}$ from $\sigma(\hat{m})$, according to the history it supplies, will get to x, and therefore must meet the path going back from $\sigma(m_i)$ Let l be the location at which the two paths meet, that is, m agrees with $\hat{m}$ on the first $l-1$ bits, and denote $w = f^{-1}_{m_{l-1}}(\ldots(f^{-1}_{m_0}(z)))$. Assume, w.l.o.g that the $l-th$ bit of m is 0, the $l-th$ bit of $\hat{m}$ is 1, and let u, v be the corresponding $f_0^{-1}(w), f_1^{-1}(w)$. Output (u, v, w).

Clearly (u, v, w) is a claw. Thus, applying the public f_0, f_1 on the output of the forging algorithm F results in a claw, with non-negligible probability; contradiction. ∎

However, this scheme does not seem to be secure against a Chosen Message Attack. At least we do not know how to prove that it is. In the next section we modify it to achieve this.

8.5.3 A secure signature scheme based on claw free permutations

Let D_f be the common domain of the claw-free permutations pair Consider the following scheme, for signing messages $m_i \in \{0,1\}^k$ where $i \in \{1, \cdots, B(k)\}$ and $B(k)$ is a polynomial in k:

Choose two pairs of claw-free permutations, (f_0, f_1) and (g_0, g_1) for which we know $f_0^{-1}, f_1^{-1}, g_0^{-1}, g_1^{-1}$. Choose $X \in D_f$. Let the public key contain $D_f, X, f_0, f_1, g_0, g_1$ and let the secret key contain $f_0^{-1}, f_1^{-1}, g_0^{-1}, g_1^{-1}$.

PK	SK
D_f, X, f_0, f_1	f_0^{-1}, f_1^{-1}
g_0, g_1	g_0^{-1}, g_1^{-1}

Let $\circ$ be the concatenation function and set the history $H_1 = \emptyset$. To sign m_i, for $i \in \{1, \cdots, B(k)\}$:

1. Choose $R_i \in D_g$ uniformly.
2. Set $z_1^i = f^{-1}_{H_i \circ R_i}(X)$.
3. Set $z_2^i = g^{-1}_{m_i}(R_i)$.
4. Set signature $\sigma(m_i) = (z_1^i, z_2^i, H_i)$.
5. Set $H_{i+1} = H_i \circ R_i$.

To verify a message-signature pair (m, s) where $s = (z_1, z_2, H)$,

1. Let $R = g_m(z_2)$.
2. Check that $f_{H \circ R}(z_1) = X$.

If so, then the signature is valid and the verification function $V(m, s) = 1$. Otherwise, $V(m, s) = 0$. This scheme takes advantage of the fact that a new random element z_1^i can be used in place of X for each message so that the forger is unable to gain information by requesting signatures for a polynomial number of messages.

It is clear that the signing and verification procedures can be performed in polynomial time as required. The following theorem also shows that it is secure:

Theorem 8.5.2 *The claw-free permutation signature scheme is existentially secure against CMA if claw-free permutations exist.*

Proof: (by contradiction) Suppose not. Then there is a forger $F^{CMA}(f_0, f_1, g_0, g_1, X)$ which consists of the following two stages:

Stage 1: F obtains signatures $\sigma(m_i)$ for up to $B(k)$ messages m_i of its choice.

Stage 2: F outputs $(\hat{m}, \hat{s})$ where $\hat{s} = (\hat{z}_1, \hat{z}_2, \hat{H})$ such that $\hat{m}$ is different than all m_i's requested in stage 1 and $V(\hat{m}, \hat{s}) = 1$.

We show that if such an F did exist, then there would be a PTM A which would:

Input: Uniformly chosen (h_0, h_1, D_h) claw-free such that h_0^{-1} and h_1^{-1} are not known.

Output: Either a h-claw with probability greater than $\frac{1}{Q(k)}$ where $Q(k)$ is a polynomial in k.

This is a contradiction by the definition of h_0 and h_1.

PTM A is based on the fact that when F is successful it does one of the following in stage 2:

Type 1 forgery: Find a g-claw

Type 2 forgery: Find a f-claw

Type 3 forgery: Find $f_0^{-1}(\omega)$ or $f_1^{-1}(\omega)$ for $\omega = z_1^{B(k)}$ the last point in the history provided by the signer

PTM A consists of two PTM's A_1 and A_2 which are run one after the other. A_1 attempts to find an h-claw based on the assumption that F produces a g-claw. A_2 attempts to find a h-claw based on the assumption that F produces a f-claw. Both A_1 and A_2 will use h_0 and h_1 in their public keys. In order to sign a message using h_0 and h_1, these PTM's will compute $v = h_i(R)$ for some $R \in D_h$ and use R as $h_b^{-1}(v)$. Thus, neither A_1 nor A_2 will need to invert h_b when answering F's requests. Note that since h_b is a permutation, v will be random if R is.

Description of A_1:

1. Choose (f_0, f_1, D_f) claw-free such that we know f_0^{-1} and f_1^{-1}. Let the public key contain $D_f, X, f_0, f_1, g_0 = h_0$, and $g_1 = h_1$. Let the secret key contain f_0^{-1} and f_1^{-1}.

PK	SK
D_f, X, f_0, f_1 $g_0 = h_0, g_1 = h_1$	f_0^{-1}, f_1^{-1}

2. Set history $H_1 = \emptyset$ and run $F(f_0, f_1, g_0, g_1, X)$. When F asks for the signature of a message m_i,
   (a) Choose $z_2^i \in D_g$ at random.
   (b) Set $R_i = g_{m_i}(z_2^i)$.
   (c) Set $z_1^i = f^{-1}_{H_i \circ R_i}(X)$.
   (d) Output (z_1^i, z_2^i, H_i).
   (e) Set $H_{i+1} = H_i \circ R_i$.

   F then outputs $(\hat{m}, \hat{s})$ where $\hat{s} = (\hat{z}_1, \hat{z}_2, \hat{H})$.
3. Test to see that $V(\hat{m}, \hat{s}) = 1$. If not then A_1 fails.
4. Let $\hat{R} = g_{\hat{m}}(\hat{z}_2)$. If $\hat{R} \neq R_i$ for any i, then A_1 fails since F did not produce a type 1 forgery
5. Otherwise, let j be such that $\hat{R} = R_j$. We now have $h_{\hat{m}}(\hat{z}_2) = h_{m_j}(z_2^j) = R_j$. From this we easily obtain a h-claw.

Description of A_2:

1. Choose (g_0, g_1, D_g) claw-free such that we know g_0^{-1} and g_1^{-1}. Let $f_0 = h_0$ and $f_1 = h_1$. Choose $R_1, R_2, \cdots, R_{B(k)} \in D_g$, $c \in \{0,1\}$ and $z \in D_f$ uniformly and independently. Set $X = f_{R_1 \circ R_2 \circ \cdots \circ R_{B(k)} \circ c}(z)$. Let the public key contain D_f, X, f_0, f_1, g_0 and g_1. Let the secret key contain g_0^{-1} and g_1^{-1}.

PK	SK
D_f, X, g_0, g_1 $f_0 = h_0, f_1 = h_1$	g_0^{-1}, g_1^{-1}

2. Set history $H_1 = \emptyset$ and run $F(f_0, f_1, g_0, g_1, X)$. When F asks for signature of message m_i,
   (a) Set $z_1^i = f_{R_{i+1} \circ \cdots \circ R_{B(k)}}(X)$.
   (b) Set $z_2^i = g_{m_i}^{-1}(R_i)$.
   (c) Output (z_1^i, z_2^i, H_i).
   (d) Set $H_{i+1} = H_i \circ R_i$.

   F then outputs $(\hat{m}, \hat{s})$ where $\hat{s} = (\hat{z}_1, \hat{z}_2, \hat{H})$.
3. Let $\hat{R} = g_{\hat{m}}(\hat{z}^2)$.
4. There are three possibilities to consider:

   F made type 1 forgery: This means $\hat{H} \circ \hat{R} = H_i$ for some i. In this case A_2 fails.

F made type 2 forgery: There is some first bit in $\hat{H} \circ \hat{R}$ which differs from A_2's final history H_N. As a result, $\hat{H} \circ \hat{R} = H \circ b \circ \hat{S}$ and $H_N = H \circ \bar{b} \circ S$ for some $b \in \{0,1\}$ and strings $H, \hat{S}, S$. From this we obtain $f_b(f_{\hat{S}}(\hat{z}_1)) = f_{\bar{b}}(f_S(z_1^N))$ which provides A_2 with a h-claw.

F made type 3 forgery: $\hat{H} \circ \hat{R} = H_N \circ b \circ S$ for some bit b and string S. Since the bit d chosen by A_2 to follow H_N if another request were made is random, b will be different from d with probability 1/2. In this case, A_2 will have $h_0^{-1}(h_{H_N}^{-1}(X))$ and $h_1^{-1}(h_{H_N}^{-1}(X))$ providing A_2 with a h-claw.

Suppose that with probability p_1 $F(f_0, f_1, g_0, g_1, X)$ provides a type 1 forgery, with probability p_2 $F(f_0, f_1, g_0, g_1, X)$ provides a type 2 forgery, and with probability p_3 $F(f_0, f_1, g_0, g_1, X)$ provides a type 3 forgery. Since $f_0, f_1, g_0, g_1, h_0, h_1$ are chosen uniformly over claw-free permutations, A_1 will succeed with probability p_1 and A_2 will succeed with probability $p_2 + \frac{p_3}{2}$. Thus, A_1 or A_2 will succeed with probability at least $\max(p_1, p_2 + \frac{p_3}{2}) \geq \frac{1}{3Q(k)}$. ∎

Notes:

1. Unlike the previous scheme, the signature here need not contain all the previous messages signed by the scheme; only the elements $R^i \in D_g$ are attached to the signature.

2. The length of the signature need not be linear with the number of messages signed. It is possible instead of linking the R^i together in a linear fashion, to build a tree structure, where R^1 authenticates R^2 and R^3, and R^2 authenticates R^4 and R^5 and so forth till we construct a full binary tree of depth logarithmic in $B(k)$ where $B(k)$ is a bound on the total number of signatures ever to be signed. Then, relabel the R^j's in the leafs of this tree as $r^1, ..., r^{B(k)}$.

   In the computation of the signature of the i-th message, we let $z_2^i = g_{m^i}^{-1}(r^i)$, and let $z_1^i = f_{r^i}^{-1}(R)$ where R is the father of r^i in the tree of authenticated R's. The signature of the ith message needs to contain then all R's on the path from the leaf r^i to the root, which is only logarithmic in the number of messages ever to be signed.

3. The cost of computing a $f_m^{-1}(x)$ is $|m|$(cost of computing f^{-1}). Next we show that for the implementation of claw-free functions based on factoring, the m factor can be saved.

Example: Efficient way to compute $f_m^{-1}(z)$

As we saw in Example 8.4.2, if factoring is hard, a particular family of trap-door permutations is claw free. Let $n = pq$, where p and q are primes and $p \equiv 3 \bmod 8$, $q \equiv 7 \bmod 8$. for $x \in QR_n$:

$$f_{0,n}(x) = x^2 \bmod n \quad f_{1,n}(x) = 4x^2 \bmod n$$

is this family of claw-free trap-door permutations.

Notation: We write $\sqrt{x} = y$ when that $y^2 = x$ and $y \in QR_n$.

To compute $f_m^{-1}(z)$ we first compute (all computations below are mod$n$):

$f_{00}^{-1}(z) = \sqrt{\sqrt{z}}$

$f_{01}^{-1}(z) = \sqrt{\frac{\sqrt{z}}{4}} = \frac{1}{\sqrt{4}}\sqrt{\sqrt{z}}$

$f_{10}^{-1}(z) = \sqrt{\sqrt{\frac{z}{4}}} = \frac{1}{\sqrt{\sqrt{4}}}\sqrt{\sqrt{z}}$

$f_{11}^{-1}(z) = \sqrt{\frac{1}{4}\sqrt{\frac{z}{4}}}$

Let $i(m)$ be the integer corresponding to the string m reversed. It is easy to see that in the general case we get: $f_m^{-1}(z) = (\frac{z}{4^{i(m)}})^{\frac{1}{2^{|m|}}}$

Now, all we need is to compute the $2^{|m|}$th root mod$n$ once, and this can be done efficiently, by raising to a power mod$\Phi(n)$.

8.5.4 A secure signature scheme based on trapdoor permutations

This section contains the trapdoor permutation signature scheme. We begin by showing the method for signing a single bit b:

1. Choose a trapdoor permutation f for which we know the inverse. Choose $X_0, X_1 \in D_f$ uniformly and independently. Let the public key contain $f, f(X_0)$, and $f(X_1)$. Let the secret key contain X_0 and X_1.

PK	SK
$f, f(X_0), f(X_1)$	X_0, X_1

2. The signature of b, $\sigma(b) = X_b$.

To verify (b, s) simply test $f(s) = f(X_b)$.

The scheme for signing multiple messages, uses the scheme above as a building block. The problem with signing multiple messages is that f cannot be reused. Thus, the trapdoor permutation signature scheme generates and signs a new trapdoor permutation for each message that is signed. The new trapdoor permutation can then be used to sign the next message.

Description of the trapdoor permutation signature scheme:

1. Choose a trapdoor permutation f_1 for which we know the inverse. Choose $\alpha_0^j, \alpha_1^j \in \{0,1\}^k$ for $j \in \{1, \cdots, k\}$ and $\beta_0^j, \beta_1^j \in \{0,1\}^k$ for $j \in \{1, \cdots, K(k)\}$ where $K(k)$ is a polynomial in k uniformly and independently. Let the public key contain f_1 and all α's and β's. Let the secret key contain f_1^{-1}. Let history $H_1 = \emptyset$.

PK	SK
$f_1, \alpha_b^i, \beta_b^j$ for $b \in \{0,1\}, i \in 1, \cdots, k, j \in 1, \cdots, K(k)$	f_1^{-1}

To sign message $m^i = m_1 m_2 \cdots m_k$:

2. Set $AUTH_{m^i}^{\alpha, f_i} = (f_i^{-1}(\alpha_{m_1}^1), f_i^{-1}(\alpha_{m_2}^2), \cdots, f_1^{-1}(\alpha_{m_k}^k))$. $AUTH_{m^i}^{\alpha, f_i}$ is the signature of m^i using f_i and the α's.

3. Choose a new trapdoor function f_{i+1} such that we know f_{i+1}^{-1}.

4. Set $AUTH_{f_{i+1}}^{\beta, f_i} = (f_i^{-1}(\beta_{f_{i+1,1}}^1), (f_i^{-1}(\beta_{f_{i+1,2}}^2)), \cdots, (f_i^{-1}(\beta_{f_{i+1,k}}^K(k))))$
where $f_{i+1} = f_{i+1,1} \circ f_{i+1,2} \circ \cdots \circ f_{i+1,K(k)}$ is the binary representation of f_{i+1}.

5. The signature of m^i is $\sigma(m^i) = (AUTH_{m^i}^{\alpha, f_i}, AUTH_{f_{i+1}}^{\beta, f_i}, H_i)$. $AUTH_{f_{i+1}}^{\beta, f_i}$ is the signature of f_{i+1} using f_i and β's.

6. Set $H_{i+1} = H_i \circ (AUTH_{m^i}^{\alpha, f_i}, AUTH_{f_{i+1}}^{\beta, f_i})$.

Note: We assume that to describe f_{i+1}, $K(k)$ bits are sufficient.

Theorem 8.5.3 *The trapdoor permutation signature scheme is existentially secure against CMA if trapdoor permutations exist.*

Proof: (by contradiction) Suppose not. Then there is a forger F which can request messages of its choice and then forge a message not yet requested with probability at least $\frac{1}{Q(k)}$ where $Q(k)$ is a polynomial in k. We show that if such an F did exist, we could find a PTM A' which would:

Input: Trapdoor function h for which inverse is unknown and $\omega \in \{0,1\}^k$.

Output: $h^{-1}(\omega)$ with probability at least $\frac{1}{Q'(k)}$ where $Q'(k)$ is a polynomial in k. Probability is taken over h's, ω's, coins of A'.

The construction of A' is as follows:

1. A' will attempt to use h as one of its trapdoor permutations in answering a signature request by F. Since A' does not know h^{-1}, it generates an appropriate set of α's and β's as follows: Randomly and uniformly choose $\gamma_b^j, \delta_b^j \in \{0,1\}^k$ for all $b \in \{0,1\}$ and $j \in \{1,\cdots,k\}$. Let $\alpha_b^j = h(\gamma_b^j)$ and $\beta_b^j = h(\delta_b^j)$ for the same range of b and j. Choose $n \in \{1,\cdots,B(k)\}$ uniformly. For the first phase, A' will act very much like a trapdoor permutation signature scheme with one exception. When it is time for A' to choose its one way permutation f_n, it will choose h. If A' were to leave the α's and β's unchanged at this point, it would be able to sign F's request for m^n though it does not know h^{-1}. However A' does change one of the α's or β's, as follows:
2. Randomly choose one of the α's or β's and set it equal to the input ω. Let the public key contain f_1 (this is h if $n = 1$), the α's and β's:
3. Run F using the current scheme. Note that with probability at least $\frac{1}{B(k)}$, F will make at least n message requests. Note also that when F does request a signature for message m^n, A' will be able to sign m^n with probability $1/2$. This is because with probability $1/2$ A' will not have to calculate (using h) the inverse of the α (or β) which was set to ω.
4. With probability $\frac{1}{Q(k)}$, F will successfully output a good forgery $(\hat{m}, \hat{s})$. In order for $\hat{s}$ to be a good forgery it must not only be verifiable, but it must diverge from the history of requests made to A'. With probability at least $\frac{1}{B(k)}$ the forger will choose to diverge from the history precisely at request n. Thus, F will use h as its trapdoor permutation.
5. If this is the case, the probability is $\frac{1}{2(k+K(k))}$ that the forger will invert the α (or β) which was set to ω.
6. If so, A' outputs $h^{-1}(\omega)$.

The probability that A' succeeds is therefore at least $\frac{1}{Q'(k)} = \frac{1}{4(k+K(k))B^2(k)}$ and since $4(k+K(k))B^2(k)$ is a polynomial in k we have a contradiction. ∎

8.6 Practical RSA based signatures: PSS

In practice, the most widely employed paradigm for signing with RSA is "hash then decrypt:" First "hash" the message into a domain point of RSA and then decrypt (ie. exponentiate with the RSA decryption exponent). The attraction of this paradigm is clear: signing takes just one RSA decryption, and verification just one RSA encryption. Furthermore it is simple to implement. Thus, in particular, this is the basis of several existing standards.

In this section we analyze this paradigm. We will see that, unfortunately, the security of the standardized schemes cannot be justified under standard assumptions about RSA, even assuming the underlying hash functions are ideal. Schemes with better justified security would be recommended.

We have already seen that such schemes do exist. Unfortunately, none of them match the schemes of the hash then decrypt paradigm in efficiency and simplicity. (See Section 8.6.7 for comparisons). So what can we do?

We present here some schemes that match "hash then decrypt" ones in efficiency but are provably secure assuming we have access to ideal hash functions. (As discussed in Section 6.4.6, this means that formally, the hash functions are modeled as random oracles, and in implementation, the hash functions are derived from cryptographic hash functions. This represents a practical compromise under which we can get efficiency with reasonable security assurances. See [20] for a full discussion of this approach.)

We present and analyze two schemes. The first is the FDH scheme of [20]. The second is the PSS of [22]. Furthermore we present a scheme called PSS-R which has the feature of *message recovery*. This is a useful way to effectively shorten signature sizes.

Let us now expand on all of the above. We begin by looking at current practice. Then we consider the full domain hash scheme of [18] and discuss its security. Finally we come to PSS and PSS-R, and their exact security.

We present these schemes for RSA. The same can be done for the Rabin scheme.

The materiel of this section is taken largely from [22].

8.6.1 Signing with RSA– Current practice

THE RSA SYSTEM. Recall that in the RSA public key system [151] a party has public key (N, e) and secret key (N, d), where N is a k-bit modulus, the product of two $(k/2)$-bit primes, and $e, d \in \mathsf{Z}^*_{\phi(N)}$ satisfy $ed \equiv 1 \bmod \varphi(N)$. (Think of $k = 1024$, a recommended modulus size these days.) Recall that the RSA function $f\colon \mathsf{Z}^*_N \to \mathsf{Z}^*_N$ is defined by $f(x) = x^e \bmod N$ and its inverse $f^{-1}\colon \mathsf{Z}^*_N \to \mathsf{Z}^*_N$ is defined by $f^{-1}(y) = y^d \bmod N$ $(x, y \in \mathsf{Z}^*_N)$. The generally-made assumption is that f is trapdoor one-way— roughly, if you don't know d (or the prime factors of N) then it is hard to compute $x = f^{-1}(y)$ for a y drawn randomly from Z^*_N.

HASH-THEN-DECRYPT SCHEMES. A widely employed paradigm to sign a document M is to first compute some "hash" $y = \mathit{Hash}(M)$ and then set the signature to $x = f^{-1}(y) = y^d \bmod N$. (To verify that x is a signature of M, compute $f(x) = x^e \bmod N$ and check this equals $\mathit{Hash}(M)$.) In particular, this is the basis for several existing standards. A necessary requirement on *Hash* in such a scheme is that it be collision-intractable and produce a k-bit output in Z^*_N. Accordingly, *Hash* is most often implemented via a cryptographic hash function like H = MD5 (which yields a 128 bit output and was assumed to be collision-intractable) and some padding. A concrete example of such a scheme is [156, 157], where the hash is

$$\mathit{Hash}_{\mathrm{PKCS}}(M) \quad = \quad \mathtt{0x\,00\,01\,FF\,FF} \cdots \mathtt{FF\,FF\,00} \,.\, H(M)\ .$$

Here . denotes concatenation, and enough `0xFF`-bytes are used so as to make the length of $\mathit{Hash}_{\mathrm{PKCS}}(M)$ equal to k bits.

SECURITY. We draw attention to the fact that the security of a hash-then-decrypt signature depends very much on how exactly one implements *Hash*. In particular, it is important to recognize that the security of a signature scheme like $\mathit{Sign}_{\mathrm{PKCS}}(M) = f^{-1}(\mathit{Hash}_{\mathrm{PKCS}}(M))$ can't be justified given (only) that RSA is trapdoor one-way, even under the assumption that hash function H is ideal. (The reason is that the set of points $\{\ \mathit{Hash}_{\mathrm{PKCS}}(M)\ :\ M \in \{0,1\}^*\ \}$ has size at most 2^{128} and hence is a very sparse, and a very structured, subset of Z^*_N.) We consider this to be a disadvantage. We stress that we don't know of any *attack* on this scheme. But we prefer, for such important primitives, to have some proof of security rather than just an absence of known attacks.

The same situation holds for other standards, including ISO 9796 [1]. (There the function *Hash* involves no cryptographic hashing, and the message M is easily recovered from $\mathit{Hash}(M)$. This doesn't effect the points we've just made.)

The above discussion highlights that *collision-intractability is not enough.* The function $\mathit{Hash}_{\mathrm{PKCS}}$ is guaranteed to be collision-intractable if we use a collision-intractable H. But this won't suffice to get a proof of security.

8.6.2 Definitions

We provide definitions for an exact security treatment of RSA, basic signature schemes, and signing with recovery.

An exact treatment of RSA

THE RSA FAMILY. As discussed in Chapter 2, RSA is a family of *trapdoor permutations.* It is specified by the RSA generator, $\mathcal{RSA}$, which, on input 1^k, picks a pair of random distinct $(k/2)$-bit primes and multiplies them to produce a modulus N. It also picks, at random, an encryption exponent $e \in \mathsf{Z}^*_{\varphi(N)}$ and computes the corresponding decryption exponent d so that $ed \equiv 1 \bmod \varphi(N)$. The generator returns N, e, d, these specifying $f\colon \mathsf{Z}^*_N \to \mathsf{Z}^*_N$ and $f^{-1}\colon \mathsf{Z}^*_N \to \mathsf{Z}^*_N$, which are defined by $f(x) = x^e \bmod N$ and $f^{-1}(y) = y^d \bmod N$. Recall that both functions are permutations, and, as the notation indicates, inverses of each other.

The trapdoor permutation generator $\mathcal{RSA}$–3 is identical to $\mathcal{RSA}$ except that the encryption exponent e is fixed to be 3. More generally, $\mathcal{RSA}$–e provides an encryption exponent of the specified constant. Other variants of $\mathcal{RSA}$ use a somewhat different distribution on the modulus N. Our results, though stated for $\mathcal{RSA}$, also hold for these other variants.

EXACT SECURITY OF THE RSA FAMILY. The security of trapdoor permutation families was discussed in Chapter 2. Here we provide an exact security version of the definition.

An *inverting algorithm* for $\mathcal{RSA}$, I, gets input N, e, y and tries to find $f^{-1}(y)$. Its success probability is the probability it outputs $f^{-1}(y)$ when N, e, d are obtained by running $\mathcal{RSA}(1^k)$ and y is set to $f(x)$ for an x chosen at random from Z^*_N. The asymptotic definition of security asks that the success probability of any PPT (probabilistic, polynomial time) algorithm be a negligible function of k. We want to go further. We are interested in exactly how much time an inverting algorithm uses and what success probability it achieves in this time. Formally an inverting algorithms is said to be a t-inverter, where $t\colon \mathsf{N} \to \mathsf{N}$, if its running time plus the size of its description is bounded by $t(k)$, in some fixed standard model of computation. We say that I (t, ϵ)-breaks $\mathcal{RSA}$, where $\epsilon\colon \mathsf{N} \to [0, 1]$, if I is a t-inverter and for each k the success probability of I is at least $\epsilon(k)$. Finally, we say that $\mathcal{RSA}$ is (t, ϵ)-secure if there is no inverter which (t, ϵ)-breaks $\mathcal{RSA}$.

EXAMPLE. The asymptotically best factoring algorithm known (NFS) takes time which seems to be about $e^{1.9(\ln N)^{1/3}(\ln \ln N)^{2/3}}$ to factor a modulus N. So one might be willing to assume that the trapdoor permutation family $\mathcal{RSA}$ is (t, ϵ)-secure for any (t, ϵ) satisfying $t(k)/\epsilon(k) \leq Ce^{k^{1/4}}$, for some particular constant C.

Exact security of signature schemes

SIGNATURE SCHEMES. Recall a *digital signature scheme* $\Pi = (\mathit{KeyGen}, \mathit{Sign}, \mathit{Vf})$ is specified by a key generation algorithm, KeyGen, a signing algorithm, Sign, and a verifying algorithm, Vf. The first two are probabilistic, and all three should run in expected polynomial time. Given 1^k, the key generation algorithm outputs a pair of matching public and secret keys, (pk, sk). The signing algorithm takes the message M to be signed and the secret key sk, and it returns a signature $x = \mathit{Sign}_{sk}(M)$. The verifying algorithm takes a message M, a candidate signature x', and the public key pk, and it returns a bit $\mathit{Vf}_{pk}(M, x')$, with 1 signifying "accept" and 0 signifying "reject." We demand that if x was produced via $x \leftarrow \mathit{Sign}_{sk}(M)$ then $\mathit{Vf}_{pk}(M, x) = 1$.

One or more strong hash functions will usually be available to the algorithms Sign and Vf, their domain and range depending on the scheme. We model them as ideal, meaning that if hash function h is invoked on some input, the output is a uniformly distributed point of the range. (But if invoked twice on the same input, the same thing is returned both times.) Formally, h is a random oracle. It is called a hash oracle and it is accessed via oracle queries: an algorithm can write a string z and get back $h(z)$ in time $|z|$.

SECURITY OF SIGNATURE SCHEMES. Definitions for the security of signatures in the asymptotic setting as provided by Goldwasser, Micali and Rivest [90] have already been discussed. They were enhanced to take into account the presence of an ideal hash function in [20]. Here we provide an exact version of these definitions.

A forger takes as input a public key pk, where $(pk, sk) \stackrel{R}{\leftarrow} \mathit{KeyGen}(1^k)$, and tries to forge signatures with respect to pk. The forger is allowed a chosen message attack in which it can request, and obtain, signatures of messages of its choice. This is modeled by allowing the forger oracle access to the signing algorithm. The forger is deemed *successful* if it outputs a *valid forgery*—namely, a message/signature pair (M, x) such that $\mathit{Vf}_{pk}(M, x) = 1$ but M was not a message of which a signature was requested earlier of the signer. The forger is said to be a $(t, q_{\mathrm{sig}}, q_{\mathrm{hash}})$-forger if its running time plus description size is bounded by $t(k)$; it makes

at most $q_{\rm sig}(k)$ queries of its signing oracle; and it makes a total of at most $q_{\rm hash}(k)$ queries of its various hash oracles. As a convention, the time $t(k)$ includes the time to answer the signing queries. Such a forger F is said to $(t, q_{\rm sig}, q_{\rm hash}, \epsilon)$-break the signature scheme if, for every k, the probability that F outputs a valid forgery is at least $\epsilon(k)$. Finally we say that the signature scheme $(KeyGen, Sign, Vf)$ is $(t, q_{\rm sig}, q_{\rm hash}, \epsilon)$-secure if there is no forger who $(t, q_{\rm sig}, q_{\rm hash}, \epsilon)$-breaks the scheme.

For simplicity we will assume that a forger does any necessary book-keeping so that it never repeats a hash query. (It might repeat a signing query. If the scheme is probabilistic, this might help it.)

Quantifying the quality of reductions

The theorems we present here will have the form: If $\mathcal{RSA}$ is (t', ϵ')-secure, then some signature scheme $\Pi = (KeyGen, Sign, Vf)$ is $(t, q_{\rm sig}, q_{\rm hash}, \epsilon)$-secure. The proof will take a forger F who $(t, q_{\rm sig}, q_{\rm hash}, \epsilon)$-breaks Π and produce from F an inverter I who (t', ϵ')-breaks $\mathcal{RSA}$. The quality of the reduction is in how the primed variables depend on the unprimed ones. We will typically view $q_{\rm sig}, q_{\rm hash}$ as given, these being query bounds we are willing to allow. (For example, $q_{\rm sig} = 2^{30}$ and $q_{\rm hash} = 2^{60}$ are reasonable possibilities.) Obviously we want t' to be as large as possible and we want ϵ' to be as small as possible. We are usually satisfied when $t' = t - \text{poly}(q_{\rm hash}, q_{\rm sig}, k)$ and $\epsilon' \approx \epsilon$.

8.6.3 The Full-Domain-Hash Scheme – FDH

THE SCHEME. Signature scheme $\mathsf{FDH} = (GenFDH, SignFDH, VerifyFDH)$ is defined as follows [20]. The key generation algorithm, on input 1^k, runs $\mathcal{RSA}(1^k)$ to obtain (N, e, d). It outputs (pk, sk), where $pk = (N, e)$ and $sk = (N, d)$. The signing and verifying algorithms have oracle access to a hash function $H_{\rm FDH}$: $\{0,1\}^* \to \mathsf{Z}_N^*$. (In the security analysis it is assumed to be ideal. In practice it can be implemented on top of a cryptographic hash function such as SHA-1.) Signature generation and verification are as follows:

$SignFDH_{N,d}(M)$
  $y \leftarrow H_{\rm FDH}(M)$
  return $y^d \bmod N$

$VerifyFDH_{N,e}(M, x)$
  $y \leftarrow x^e \bmod N$; $y' \leftarrow H_{\rm FDH}(M)$
  if $y = y'$ **then return** 1 **else return** 0

SECURITY. The following theorem summarizes the exact security of the FDH scheme as provided by the reduction of [20]. The proof is straightforward, but it is instructive all the same, so we include it. The disadvantage of the result, from our point of view, is that ϵ' could be much smaller than ϵ.

Theorem 8.6.1 [20, 22] *Suppose $\mathcal{RSA}$ is a (t', ϵ')-secure. Then, for any $q_{\rm sig}, q_{\rm hash}$, signature scheme* FDH *is $(t, q_{\rm sig}, q_{\rm hash}, \epsilon)$-secure, where*

$$\begin{aligned} t(k) &= t'(k) - [q_{\rm hash}(k) + q_{\rm sig}(k) + 1] \cdot \Theta(k^3) \quad \textit{and} \\ \epsilon(k) &= [q_{\rm sig}(k) + q_{\rm hash}(k)] \cdot \epsilon'(k) \ . \end{aligned}$$

Proof: Let F be a forger which $(t, q_{\rm sig}, q_{\rm hash}, \epsilon)$-breaks FDH. We present an inverter I which (t', ϵ')-breaks $\mathcal{RSA}$.

Inverting algorithm I is given as input (N, e, y) where N, e, d were obtained by running the generator $\mathcal{RSA}(1^k)$, and y was chosen at random from Z_N^*. It is trying to find $x = f^{-1}(y)$, where f is the RSA function described by N, e. It forms the public key N, e of the Full-Domain-Hash signature scheme, and starts running F on input of this key. Forger F will make two kinds of oracle queries: hash oracle queries and signing queries. Inverter I must answer these queries itself. For simplicity we assume that if F makes sign query M then it has already made hash oracle query M. (We will argue later that this is wlog.) Let $q = q_{\rm sig} + q_{\rm hash}$. Inverter I picks at random an integer j from $\{1, \ldots, q\}$. Now we describe how I answers oracle queries. Here i is a counter, initially 0.

Suppose F makes hash oracle query M. Inverter I increments i and sets $M_i = M$. If $i = j$ then it sets $y_i = y$ and returns y_i. Else it picks r_i at random in Z_N^*, sets $y_i = f(r_i)$, and returns y_i.

Alternatively, suppose F makes signing query M. By assumption, there was already a hash query of M, so $M = M_i$ for some i. Let I return the corresponding r_i as the signature.

Eventually, F halts, outputting some (attempted forgery) (M, x). Let inverting algorithm I output x. Without loss of generality (see below) we may assume that $M = M_i$ for some i. In that case, if (M, x) is a valid forgery, then, with probability at least $1/q$, we have $i = j$ and $x = f^{-1}(y_i) = f^{-1}(y)$ was the correct inverse for f.

The running time of I is that of F plus the time to choose the y_i-values. The main thing here is one RSA computation for each y_i, which is cubic time (or better). This explains the formula for t.

It remains to justify the assumptions. Recall that I is running F. So if the latter makes a sign query without having made the corresponding hash query, I at once goes ahead and makes the hash query itself. Similarly for the output forgery. All this means that the effective number of hash queries is at most $q_{\text{hash}} + q_{\text{sig}} + 1$, which is the number we used in the time bound above. ∎

The plus here is that we have a proof of security. Now, let us consider the quantitative aspects. We note that ϵ could thus be considerably larger than ϵ'. This means that even if RSA is quite strong, the guarantee on the signature scheme could be quite weak. To see this, say we would like to allow the forger to see at least $q_{\text{sig}}(k) = 2^{30}$ example signatures and compute hashes on, say, $q_{\text{hash}} = 2^{60}$ strings. Then even if the RSA inversion probability was originally as low as 2^{-61}, all we can say is that the forging probability is now at most $1/2$, which is not good enough. To compensate, we will have to be able to assume that $\epsilon'(k)$ is very, very low, like 2^{-100}. This means that we must have a fairly large value of k, ie. a larger modulus. But this affects the efficiency of the scheme, because the time to do the underlying modular exponentiation grows (and rather quickly) as the modulus size increases. We prefer to avoid this.

We reiterate the crucial point: if the reduction proving security is "loose," like the one above, the efficiency of the scheme is impacted, because we must move to a larger security parameter. Thus, it would be nice to have "tighter" reductions, meaning ones in which ϵ is almost the same as ϵ', with the relations amongst the other parameters staying about the same as they are now.

One might suggest that it is possible to prove a better security bound for FDH than that outlined above. Perhaps, but we don't know how. Instead, we will present a different scheme for which a better security bound can be proven. The main idea is to make the hashing probabilistic.

Before going on, let us clarify our assessments of scheme quality. We are *not* saying the FDH scheme is bad. Indeed, since it is provable, it is ahead of schemes discussed in Section 8.6.1, and a viable alternative to them. What we are saying is that it is possible to do *even better* than FDH. That is, it is possible to get a scheme which is not only proven secure, but has strong exact security. This successor to FDH is the scheme we discuss next.

8.6.4 The Probabilistic Signature Scheme – PSS

Here we present another scheme—a probabilistic generalization of FDH. It preserves the efficiency and provable security of FDH but achieves the latter with a much better security bound. This is the probabilistic signature scheme (PSS) of [22].

Description of the PSS

Signature scheme $\mathsf{PSS}[k_0, k_1] = (\mathit{GenPSS}, \mathit{SignPSS}, \mathit{VerifyPSS})$ is parameterized by k_0 and k_1, which are numbers between 1 and k satisfying $k_0 + k_1 \leq k - 1$. To be concrete, the reader may like to imagine $k = 1024$, $k_0 = k_1 = 128$.

The key generation algorithm GenPSS is identical to GenFDH: on input 1^k, run $\mathcal{RSA}(1^k)$ to obtain (N, e, d), and output (pk, sk), where $pk = (N, e)$ and $sk = (N, d)$.

The signing and verifying algorithms make use of two hash functions. The first, h, called the compressor, maps as $h\colon \{0,1\}^* \to \{0,1\}^{k_1}$ and the second, g, called the generator, maps as $g\colon \{0,1\}^{k_1} \to \{0,1\}^{k-k_1-1}$.

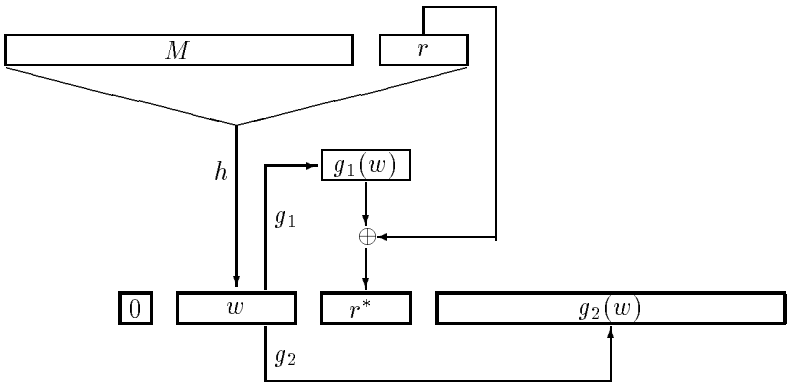

Figure 8.1: PSS: Components of image $y = 0 \,.\, w \,.\, r^* \,.\, g_2(w)$ are darkened. The signature of M is $y^d \bmod N$.

(The analysis assumes these to be ideal. In practice they can be implemented in simple ways out of cryptographic hash functions like MD5, as discussed in Appendix 8.6.6.) Let g_1 be the function which on input $w \in \{0,1\}^{k_1}$ returns the first k_0 bits of $g(w)$, and let g_2 be the function which on input $w \in \{0,1\}^{k_1}$ returns the remaining $k - k_0 - k_1 - 1$ bits of $g(w)$. We now describe how to sign and verify. Refer to Figure 8.1 for a picture.

SignPSS (M)
  $r \xleftarrow{R} \{0,1\}^{k_0}$; $w \leftarrow h(M \,.\, r)$; $r^* \leftarrow g_1(w) \oplus r$
  $y \leftarrow 0 \,.\, w \,.\, r^* \,.\, g_2(w)$
  return $y^d \bmod N$

VerifyPSS (M, x)
  $y \leftarrow x^e \bmod N$
  Break up y as $b \,.\, w \,.\, r^* \,.\, \gamma$. (That is, let b be the first bit of y, w
    the next k_1 bits, r^* the next k_0 bits, and γ the remaining bits.)
  $r \leftarrow r^* \oplus g_1(w)$
  if ($h(M \,.\, r) = w$ and $g_2(w) = \gamma$ and $b = 0$) **then return** 1
  else return 0

The step $r \xleftarrow{R} \{0,1\}^{k_0}$ indicates that the signer picks at random a seed r of k_0 bits. He then concatenates this seed to the message M, effectively "randomizing" the message, and hashes this down, via the "compressing" function, to a k_1 bit string w. Then the generator g is applied to w to yield a k_0 bit string $r^* = g_1(w)$ and a $k - k_0 - k_1 - 1$ bit string $g_2(w)$. The first is used to "mask" the k_0-bit seed r, resulting in the masked seed r^*. Now $w \,.\, r^*$ is pre-pended with a 0 bit and appended with $g_2(w)$ to create the image point y which is decrypted under the RSA function to define the signature. (The 0-bit is to guarantee that y is in Z_N^*.)

Notice that a new seed is chosen for each message. In particular, a given message has many possible signatures, depending on the value of r chosen by the signer.

Given (M, x), the verifier first computes $y = x^e \bmod N$ and recovers r^*, w, r. These are used to check that y was correctly constructed, and the verifier only accepts if all the checks succeed.

Note the efficiency of the scheme is as claimed. Signing takes one application of h, one application of g, and one RSA decryption, while verification takes one application of h, one application of g, and one RSA encryption.

Security of the PSS

The following theorem proves the security of the PSS based on the security of RSA, but with a relation between the two securities that is much tighter than the one we saw for the FDH scheme. The key difference is that $\epsilon(k)$ is within an additive, rather than multiplicative, factor of $\epsilon'(k)$, and this additive factor decreases exponentially with k_0, k_1. The relation between t and t' is about the same as in Theorem 8.6.1.

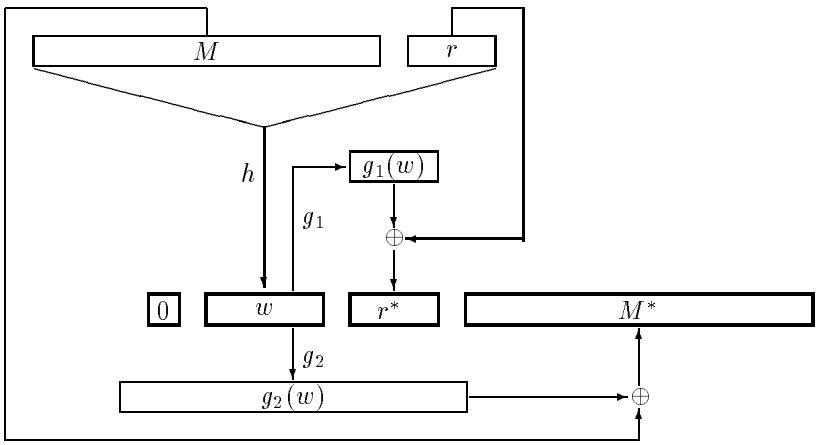

Figure 8.2: PSS-R: Components of image $y = 0\,.\,w\,.\,r^*\,.\,M^*$ are darkened.

Theorem 8.6.2 [22] *Suppose that $\mathcal{RSA}$ is (t', ϵ')-secure. Then for any $q_{\rm sig}, q_{\rm hash}$ the signature scheme* PSS$[k_0, k_1]$ *is $(t, q_{\rm sig}, q_{\rm hash}, \epsilon)$-secure, where*

$$\begin{aligned} t(k) &= t'(k) - [q_{\rm sig}(k) + q_{\rm hash}(k) + 1] \cdot k_0 \cdot \Theta(k^3), \quad \textit{and} \\ \epsilon(k) &= \epsilon'(k) + [3(q_{\rm sig}(k) + q_{\rm hash}(k))^2] \cdot (2^{-k_0} + 2^{-k_1})\ . \end{aligned}$$

The proof is in [22]. We stress how that proof differs from that of Theorem 8.6.1. There, we had to "guess" the value of $i \in \{1, \ldots, q_{\rm sig} + q_{\rm hash}\}$ for which F would forge a message, and we were only successful if we guessed right. Here we are successful (except with very small probability) no matter what is the value of i for which the forgery occurs.

8.6.5 Signing with Message Recovery – PSS-R

MESSAGE RECOVERY. In a standard signature scheme the signer transmits the message M in the clear, attaching to it the signature x. In a scheme which provides message recovery, only an "enhanced signature" τ is transmitted. The goal is to save on the bandwidth for a signed message: we want the length of this enhanced signature to be smaller than $|M| + |x|$. (In particular, when M is short, we would like the length of τ to be k, the signature length.) The verifier recovers the message M from the enhanced signature and checks authenticity at the same time.

We accomplish this by "folding" part of the message into the signature in such a way that it is "recoverable" by the verifier. When the length n of M is small, we can in fact fold the entire message into the signature, so that only a k bit quantity is transmitted. In the scheme below, if the security parameter is $k = 1024$, we can fold up to 767 message bits into the signature.

DEFINITION. Formally, the key generation and signing algorithms are as before, but Vf is replaced by $Recover$, which takes pk and x and returns $Recover_{pk}(x) \in \{0,1\}^* \cup \{\mathsf{REJECT}\}$. The distinguished point REJECT is used to indicate that the recipient rejected the signature; a return value of $M \in \{0,1\}^*$ indicates that the verifier accepts the message M as authentic. The formulation of security is the same except for what it means for the forger to be *successful*: it should provide an x such that $Recover_{pk}(x) = M \in \{0,1\}^*$, where M was not a previous signing query. We demand that if x is produced via $x \leftarrow Sign_{sk}(M)$ then $Recover_{pk}(x) = M$.

A simple variant of PSS achieves message recovery. We now describe that scheme and its security.

THE SCHEME. The scheme PSS-R$[k_0, k_1] = (GenPSSR, SignPSSR, RecPSSR)$ is parameterized by k_0 and k_1, as before. The key generation algorithm is $GenPSS$, the same as before. As with PSS, the signing and verifying algorithms depend on hash functions $h\colon \{0,1\}^* \to \{0,1\}^{k_1}$ and $g\colon \{0,1\}^{k_1} \to \{0,1\}^{k-k_1-1}$, and we use the same g_1 and g_2 notation. For simplicity of explication, we assume that the messages to be signed

have length $n = k - k_0 - k_1 - 1$. (Suggested choices of parameters are $k = 1024$, $k_0 = k_1 = 128$ and $n = 767$.) In this case, we produce "enhanced signatures" of only k bits from which the verifier can recover the n-bit message and simultaneously check authenticity. Signature generation and verification proceed as follows. Refer to Figure 8.2 for a picture.

SignPSSR (M)
  $r \xleftarrow{R} \{0,1\}^{k_0}$; $w \leftarrow h(M \,.\, r)$; $r^* \leftarrow g_1(w) \oplus r$
  $M^* \leftarrow g_2(w) \oplus M$
  $y \leftarrow 0 \,.\, w \,.\, r^* \,.\, M^*$
  return $y^d \bmod N$

RecPSSR (x)
  $y \leftarrow x^e \bmod N$
  Break up y as $b \,.\, w \,.\, r^* \,.\, M^*$. (That is, let b be the first bit of y, w the next k_1 bits, r^* the next k_0 bits, and M^* the remaining bits.)
  $r \leftarrow r^* \oplus g_1(w)$
  $M \leftarrow M^* \oplus g_2(w)$
  if ($h(M \,.\, r) = w$ and $b = 0$) **then return** M **else return** REJECT

The difference in *SignPSSR* with respect to *SignPSS* is that the last part of y is not $g_2(w)$. Instead, $g_2(w)$ is used to "mask" the message, and the masked message M^* is the last part of the image point y.

The above is easily adapted to handle messages of arbitrary length. A fully-specified scheme would use about $\min\{k,\ n + k_0 + k_1 + 16\}$ bits.

SECURITY. The security of PSS-R is the same as for PSS.

Theorem 8.6.3 [22] *Suppose that $\mathcal{RSA}$ is (t', ϵ')-secure. Then for any $q_{\rm sig}, q_{\rm hash}$ the signing-with-recovery scheme* PSS-R$[k_0, k_1]$ *is $(t, q_{\rm sig}, q_{\rm hash}, \epsilon)$-secure, where*

$$\begin{aligned} t(k) &= t'(k) - [q_{\rm sig}(k) + q_{\rm hash}(k) + 1] \cdot k_0 \cdot \Theta(k^3), \quad \text{and} \\ \epsilon(k) &= \epsilon'(k) + [3(q_{\rm sig}(k) + q_{\rm hash}(k))^2] \cdot (2^{-k_0} + 2^{-k_1}) \ . \end{aligned}$$

The proof of this theorem is very similar to that of Theorem 8.6.2 and hence is omitted.

8.6.6 How to implement the hash functions h, g

In the PSS we need a concrete hash function h with output length some given number k_1. Typically we will construct h from some cryptographic hash function H such as H = MD5 or H = SHA-1. Ways to do this have been discussed before in [20, 19]. For completeness we quickly summarize some of these possibilities. The simplest is to define $h(x)$ as the appropriate-length prefix of

$$H(\texttt{const}.\langle 0\rangle.x) \,.\, H(\texttt{const}.\langle 1\rangle.x) \,.\, H(\texttt{const}.\langle 2\rangle.x) \,.\, \cdots \ .$$

The constant `const` should be unique to h; to make another hash function, g, simply select a different constant.

8.6.7 Comparison with other schemes

We have already discussed the PKCS standards [156, 157] and the ISO standard [1] and seen that their security cannot be justified based on the assumption that RSA is trapdoor one-way. Other standards, such as [9], are similar to [156], and the same statement applies.

The schemes we discuss in the remainder of this section do not use the hash-then-decrypt paradigm.

Signature schemes whose security can be provably based on the RSA assumption include [90, 17, 129, 154, 68]. The major plus of these works is that they do not use an ideal hash function (random oracle) model— the provable security is in the standard sense. On the other hand, the security reductions are quite

loose for each of those schemes. On the efficiency front, the efficiency of the schemes of [90, 17, 129, 154] is too poor to seriously consider them for practice. The Dwork-Naor scheme [68], on the other hand, is computationally quite efficient, taking two to six RSA computations, although there is some storage overhead and the signatures are longer than a single RSA modulus. This scheme is the best current choice if one is willing to allow some extra computation and storage, and one wants well-justified security *without* assuming an ideal hash function.

Back among signature schemes which assume an ideal hash, a great many have been proposed, based on the hardness of factoring or other assumptions. Most of these schemes are derived from identification schemes, as was first done by [73]. Some of these methods are provable (in the ideal hash model), some not. In some of the proven schemes exact security is analyzed; usually it is not. In no case that we know of is the security tight. The efficiency varies. The computational requirements are often lower than a hash-then-decrypt RSA signature, although key sizes are typically larger.

Finally we note related new work. Pointcheval and Stern [141] consider the provable security of signatures in the random oracle model and show that a modified version of the El Gamal scheme [77], as well as the Schnorr [161] scheme, can be proven secure. (And the scheme of [73] can be proven secure against attacks in which there are no signature queries.) But they don't consider exact security. An interesting question is to consider, and possibly improve, the exact security of their reductions (making, if necessary, modifications to the schemes).

8.7 Threshold Signature Schemes

Using a threshold signature scheme, digital signatures can be produced by a group of players rather than by one party. In contrast to the regular signature schemes where the signer is a single entity which holds the secret key, in threshold signature schemes the secret key is shared by a group of n players. In order to produce a valid signature on a given message m, individual players produce their *partial signatures* on that message, and then combine them into a full signature on m. A distributed signature scheme achieves threshold $t < n$, if no coalition of t (or less) players can produce a new valid signature, even after the system has produced many signatures on different messages. A signature resulting from a threshold signature scheme is the same as if it was produced by a single signer possessing the full secret signature key. In particular, the validity of this signature can be verified by anyone who has the corresponding unique public verification key. In other words, the fact that the signature was produced in a distributed fashion is transparent to the recipient of the signature.

Threshold signatures are motivated both by the need that arises in some organizations to have a group of employees agree on a given message (or a document) before signing it, as well as by the need to protect signature keys from the attack of internal and external adversaries. The latter becomes increasingly important with the actual deployment of public key systems in practice. The signing power of some entities, (e.g., a government agency, a bank, a certification authority) inevitably invites attackers to try and "steal" this power. The goal of a threshold signature scheme is twofold: To increase the availability of the signing agency, and at the same time to increase the protection against forgery by making it harder for the adversary to learn the secret signature key. Notice that in particular, the threshold approach rules out the naive solution based on traditional secret sharing (see Chapter 10), where the secret key is shared in a group but reconstructed by a *single* player each time that a signature is to be produced. Such protocol would contradict the requirement that no t (or less) players can ever produce a new valid signature. In threshold schemes, multiple signatures are produced without an exposure or an explicit reconstruction of the secret key.

Threshold signatures are part of a general approach known as *threshold cryptography*. This approach has received considerable attention in the literature; we refer the reader to [59] for a survey of the work in this area. Particular examples of solutions to threshold signatures schemes can be found in [60, 160] for RSA and in [94] for ElGamal-type of signatures.

A threshold signature scheme is called robust if not only t or less players cannot produce a valid signature, but also cannot *prevent* the remaining players from computing a signature on their own. A robust scheme basically foils possible denial of service attacks on the part of corrupted servers. The solutions mentined above are *not* robust. In this chapter we will concentrate on robust schemes. We will not go into technical details. The goal of this section is to present the reader with the relevant notions and point to the sources

in the literature.

In the following we will refer to the signing servers with the letters $P_1, \ldots, P_n$.

8.7.1 Key Generation for a Threshold Scheme

The task of generating a key for a threshold signature scheme is more complicated than when we are in the presence of a single signer. Indeed we must generate a public key PK whose matching secret key SK is shared in some form among the servers $P_1, \ldots, P_n$.

A way of doing this is to have some trusted *dealer* who generates a key pair (PK, SK) for the given signature scheme, makes PK public and shares SK among the P_i's using a secret sharing protocol (see Chapter 10.) However notice that such a key generation mechanisms contradicts the requirement that no single entity should be able to sign, as now the dealer knows the secret key SK and he is able to sign on his own. This is why people have been trying to avoid the use of such a dealer during the key generation phase.

For the case of discrete-log based signature schemes, this quest has been successful. Robust threshold signature schemes for the El Gamal, Schnorr and DSS signature schemes (see [77, 161, 74]) can be found in [45, 136, 80], all using underlying results of Feldman and Pedersen [72, 137, 138].

Yet, in some cases the dealer solution is the best we can do. For example, if the underlying signature scheme is RSA, then we do not know how to generate a key in a shared form without the use of a dealer.

8.7.2 The Signature Protocol

Once the key is generated and in some way shared among the servers $P_1, \ldots, P_n$ we need a signature protocol.

The idea is that on input a message M, the servers will engage in some form of communication that will allow them to compute a signature σ for M, without revealing the secret key. Such protocol should not leak any information beyond such signature σ. Also in order to obtain the *robustness* property, such protocols should correctly compute the signature even if up to t servers P_i's are corrupted and behave in *any* way during the protocol. If possible the computation required by a server P_i to sign in this distributed manner should be comparable to the effort required if P_i were signing on his own. Interaction should be reduced to a minimum

For El Gamal-like schemes robust threshold signature schemes can be found in [45, 136]. The specific case of DSS turned out to be very difficult to handle. The best solution is in [80].

RSA turned out to be even less amenable to the construction of robust schemes. A somewhat inefficient solution (requires much more computation and a lot of interaction between servers) can be found in [75]. A very efficient and non-interactive solution was independently proposed in [79].

CHAPTER 9

Key distribution

We have looked extensively at encryption and data authentication and seen lots of ways to design schemes for these tasks. We must now address one of the assumptions underlying these schemes. This is the assumption that the parties have available certain kinds of keys.

This chapter examines various methods for key distribution and key management. A good deal of our effort will be expended in understanding the most important practical problem in the area, namely session key distribution.

Let us begin with the classic secret key exchange protocol of Diffie and Hellman.

9.1 Diffie Hellman secret key exchange

Suppose Alice and Bob have no keys (shared or public), and want to come up with a joint key which they would use for private key cryptography. The Diffie-Hellman (DH) secret key exchange (SKE) protocol [62] enables them to do just this.

9.1.1 The protocol

We fix a prime p and a generator $g \in Z_p^*$. These are public, and known not only to all parties but also to the adversary E.

- A picks $x \in Z_{p-1}$ at random and lets $X = g^x \bmod p$. She sends X to B
- B picks $y \in Z_{p-1}$ at random and lets $Y = g^y \bmod p$. He sends Y to A.

Now notice that

$$X^y = (g^x)^y = g^{xy} = (g^y)^x = Y^x \ ,$$

the operations being in the group Z_p^*. Let's call this common quantity K. The crucial fact is that *both* parties can compute it! Namely A computes Y^x, which is K, and B computes X^y, which is also K, and now they have a shared key.

9.1.2 Security against eavesdropping: The DH problem

Is this secure? Consider an adversary that is sitting on the wire and sees the flows that go by. She wants to compute K. What she sees is X and Y. But she knows neither x nor y. How could she get K? The natural attack is to find either x or y (either will do!) from which she can easily compute K. However, notice that computing x given X is just the discrete logarithm problem in Z_p^*, which is widely believed to be intractable (for suitable choices of the prime p). Similarly for computing y from Y. Accordingly, we would be justified in having some confidence that this attack would fail.

A number of issues now arise. The first is that computing discrete logarithms is not the only possible attack to try to recover K from X, Y. Perhaps there are others. To examine this issue, let us formulate the computational problem the adversary is trying to solve. It is the following:

The DH Problem: Given g^x and g^y for x, y chosen at random from Z_{p-1}, compute g^{xy}.

Thus the question is how hard is this problem? We saw that if the discrete logarithm problem in Z_p^* is easy then so is the DH problem; ie. if we can compute discrete logs we can solve the DH problem. Is the converse true? That is, if we can solve the DH problem, can we compute discrete logarithms? This remains an open question. To date it seems possible that there is some clever approach to solving the DH problem without computing discrete logarithms. However, no such approach has been found. The best known algorithm for the DH problem is to compute the discrete logarithm of either X or Y. This has lead cryptographers to believe that the DH problem, although not known to be equivalent to the discrete logarithm one, is still a computationally hard problem, and that as a result the DH secret key exchange is secure in the sense that a computationally bounded adversary can't compute the key K shared by the parties.

The DH Assumption: The DH problem is computationally intractable.

These days the size of the prime p is recommended to be at least 512 bits and preferably 1024. As we have already seen, in order to make sure the discrete logarithm problem modulo p is intractable, $p-1$ should have at least one large factor. In practice we often take $p = 2q + 1$ for some large prime q.

The relationship between the DH problem and the discrete logarithm problem is the subject of much investigation. See for example Maurer [121].

9.1.3 The DH cryptosystem

The DH secret key exchange gives rise to a very convenient public key cryptosystem. A party A will choose as its secret key a random point $x \in Z_{p-1}$, and let $X = g^x$ be its public key. Now if party B wants to privately send A a message M, it would proceed as follows.

First, the parties agree on a private key cryptosystem $(\mathcal{E}, \mathcal{D})$ (cf. Chapter 5). For concreteness assume it is a DES based cryptosystem, so that it needs a 56 bit key. Now B picks y at random from Z_{p-1} and computes the DH key $K = X^y = g^{xy}$. From this, he extracts a 56 bit key a for the private key cryptosystem according to some fixed convention, for example by letting a be the first 56 bits of K. He now encrypts the plaintext M under a using the private key cryptosystem to get the ciphertext $C = \mathcal{E}_a(M)$, and transmits the pair (Y, C) where $Y = g^y$.

A receives (Y, C). Using her secret key x she can compute the DH key $K = Y^x = g^{xy}$, and thus recover a. Now she can decrypt the ciphertext C according to the private key cryptosystem, via $M = \mathcal{D}_a(C)$, and thus recover the plaintext M.

Intuitively, the security would lie in the fact that the adversary is unable to compute K and hence a. This, however, is not quite right, and brings us to the issue of the bit security of the DH key.

9.1.4 Bit security of the DH key

Above the first 56 bits of the key $K = g^{xy}$ is used as the key to a private key cryptosystem. What we know (are willing to assume) is that given g^x, g^y the adversary cannot recover K. This is not enough to make the usage of K as the key to the private key cryptosystem secure. What if the adversary were able to recover the first 56 bits of K, but not all of K? Then certainly the above cryptosystem would be insecure. Yet, having the first 56 bits of K may not enable one to find K, so that we have not contradicted the DH assumption.

This is an issue we have seen before in many contexts, for example with one-way functions and with encryption. It is the problem of partial information. If f is one-way it means given $f(x)$ I can't find x; it doesn't mean I can't find some bits of x. Similarly, here, that we can't compute K doesn't mean we can't compute some bits of K.

Indeed, it turns out that computing the last bit (ie. LSB) of $K = g^{xy}$ given g^x, g^y is easy. To date there do not seem to be other detectable losses of partial information. Nonetheless it would be unwise to just use

some subset of bits of the DH key as the key to a private key cryptosystem. Assuming that these bits are secure is a much stronger assumption than the DH assumption.

So what could we do? In practice, we might hash the DH key K to get a symmetric key a. For example, applying a cryptographic hash function like SHA-1 to K yields 160 bits that might have better "randomness" properties than the DH key. Now use the first 56 bits of this if you need a DES key.

However, while the above may be a good heuristic in practice, it can't be validated without very strong assumptions on the randomness properties of the hash function. One possibility that can be validated is to extract hard bits from the DH key via an analogue of Theorem 2.4.2. Namely, let r be a random string of length $|p|$ and let b be the dot product of K and r. Then predicting b given g^x, g^y is infeasible if computing $K = g^{xy}$ given g^x, g^y is infeasible. The drawback of this approach is that one gets very few bits. To get 56 bits one would need to exchange several DH keys and get a few bits from each.

We saw in Chapter 2 that for certain one way functions we can present hardcore predicates, the prediction of which can be reduced to the problem of inverting the function itself. A theorem like that for the DH key would be nice, and would indicate how to extract bits to use for a symmetric key. Recently results of this kind have been proved by Boneh and Venkatesan [38].

9.1.5 The lack of authenticity

At first glance, the DH secret key exchange might appear to solve in one stroke the entire problem of getting keys to do cryptography. If A wants to share a key with B, they can just do a DH key exchange to get one, and then use private key cryptography.

Don't do it. The problem is *authenticity*. The security of the DH key is against a *passive* adversary, or *eavesdropper*. It is assumed that the adversary will recover the transmitted data but not try to inject data on the line. In practice, of course, this is an untenable assumption. It is quite easy to inject messages on networks, and hackers can mount active attacks.

What damage can this do? Here is what the adversary does. She calls up B and simply plays the role of A. That is, she claims to be A, who is someone with whom B would like to share a key, and then executes the DH protocol like A would. Namely she picks x at random and sends $X = g^x$ to B. B returns $Y = g^y$ and now B and the adversary share the key $K = g^{xy}$. But B thinks the key is shared with A. He might encrypt confidential data using K, and then the adversary would recover this data.

Thus in the realistic model of an active adversary, the DH key exchange is of no direct use. The real problem is to exchange a key in an authenticated manner. It is this that we now turn to.

However, we remark that while the DH key exchange is not a solution, by itself, to the key distribution problem in the presence of an active adversary, it is a useful tool. We will see how to use it in conjunction with other tools we will develop to add to session key distribution protocols nice features like "forward secrecy."

9.2 Session key distribution

Assume now we are in the presence of an active adversary. The adversary can inject messages on the line and alter messages sent by legitimate parties, in addition to eavesdropping on their communications. We want to get shared keys.

A little thought will make it clear that if the legitimate parties have no information the adversary does not know, it will not be possible for them to exchange a key the adversary does not know. This is because the adversary can just impersonate one party to another, like in the attack on DH above. Thus, in order to get off the ground, the legitimate parties need an "information advantage." This is some information, pre-distributed over a trusted channel, which the adversary does not know, and which enables them to securely exchange keys in the future.

We now discuss various ways to realize this information advantage, and the session key distribution problems to which they give rise. Then we explain the problem in more depth. We largely follow [18, 21].

9.2.1 Trust models and key distribution problems

What forms might the information advantage take? There are various different *trust models* and corresponding key distribution problems.

The three party model

This model seems to have been first mentioned by Needham and Schroeder [131]. It has since been popularized largely by the *Kerberos* system [175].

In this model there is a trusted party called the *authentication server*, and denoted S. Each party A in the system has a key K_A which it shares with the server. This is a private key between these two parties, not known to any other party. When two parties A, B, sharing, respectively, keys K_A and K_B with S, want to engage in a communication session, a three party protocol will be executed, involving A, B and S. The result will be to issue a common key K to A and B. They can then use this key to encrypt or authenticate the data they send each other.

The distributed key is supposed to be a secure session key. When the parties have completed their communication session, they will discard the key K. If later they should desire another communication session, the three party protocol will be re-executed to get a new, fresh session key.

What kinds of security properties should this distributed session key have? We will look at this question in depth later. It is an important issue, since, as we will see, session key distribution protocols must resist a variety of novel attacks.

The two party asymmetric model

When public key cryptography can be used, the authentication server's active role can be eliminated. In this trust model, the assumption is that A has the public key pk_B of B, and B has the public key pk_A of A. These keys are assumed authentic. That is, A is assured the key he holds is really the public key of B and not someone else, and analogously for B.[1]

Now, suppose A and B want to engage in a secure communication session. The problem we want to consider is how they can get a shared, private and authentic session key based on the public keys, via a two party protocol.

Questions pertaining to what exactly is the problem, and why, may arise here. We already know that we can authenticate and encrypt data with public keys. That is, the parties already have the means to secure communication. So why do they need a shared session key?

There are several reasons. One is that private key cryptography, at least under current technology, is considerable more efficient than public key cryptography. The second, however, probably more important, is that it is convenient to have *session* keys. They allow one to associate a key uniquely to a session. This is an advantage for the following reasons.

Keys actually used to encrypt or authenticate data get greater exposure. They are used by applications in ways that may not be known, or controllable, beforehand. In particular, an application might mis-use a key, or expose it. This might (or might not) compromise the current session, but we would not like it to compromise the long lived secret key and thus other uses and sessions. Similarly, the long lived secret key of a user A (namely the secret key sk_A corresponding to her public key pk_A) may be stored in protected hardware and accessed only via a special interface, while the session key lies on a more exposed machine.

The two party symmetric model

Probably the simplest model is of two parties who already share a long lived key. Each time they wish to engage in a communication session they will run a protocol to derive a session key.

Again, the motivation is the convenience and security advantages of session keys. We stress the main one. A host of applications might be run by the users, all wanting keys for one reason or another. We don't

[1] How is this situation arrived at? That isn't a problem we really want to discuss yet: it falls under the issue of key management and will be discussed later. But, briefly, what we will have is trusted servers which provide public, certified directories of users and their public keys. The server maintains for each user identity a public key, and provides this upon request to any other user, with a signature of the server that serves as a certificate of authenticity. Barring this directory service, however, the server plays no active role.

want to make assumptions about how they use the key. Some might use it in ways that are secure for their own purposes but compromise the key globally. In order for this not to affect the global security, we assign each run of each application a separate session key.

9.2.2 History of session key distribution

Although session key distribution is an old problem, it is only recently that a cryptographically sound treatment of it, in the "provable security" or "reductionist" tradition that these lecture notes are describing, has emerged [18, 21]. Via this approach we now have models in which to discuss and prove correct protocols, and several protocols proven secure under standard cryptographic assumptions.

The history prior to this was troubled. Session key distribution is an area in which a large number of papers are published, proposing protocols to solve the problem. However, many of them are later broken, or suffer from discernible design flaws.

The problem is deceptively simple. It is easy to propose protocols in which subtle security problems later emerge.

In the three party case, Needham and Schroeder [131] describe a number of candidate protocols. They had prophetically ended their paper with a warning on this approach, saying that "protocols such as those developed here are prone to extremely subtle errors that are unlikely to be detected in normal operations. The need for techniques to verify the correctness of such protocols is great ...". Evidence of the authors' claim came unexpectedly when a bug was pointed out in their own "Protocol 1" (Denning and Sacco, [58]).[2] Many related protocols were eventually to suffer the same fate.

As a result of a long history of such attacks there is finally a general consensus that session key distribution is not a goal adequately addressed by giving a protocol for which the authors can find no attacks.

A large body of work, beginning with Burrows, Abadi and Needham [41], aims to improve on this situation via the use of special-purpose logics. The aim is to demonstrate a lack of "reasoning problems" in a protocol being analyzed. The technique has helped to find errors in various protocols, but a proof that a protocol is "logically correct" does not imply that it is is right (once its abstract cryptographic operations are instantiated). Indeed it is easy to come up with concrete protocols which are logically correct but blatantly insecure.

Examining the work on the session key distribution problem, one finds that the bulk of it is divorced from basic cryptographic principles. For example one find over and over again a confusion between data encryption and data authentication. The most prevalent problem is a lack of specification of what exactly is the problem that one is trying to solve. There is no *model* of adversarial capabilities, or definition of security.

Influential works in this area were Bird et. al. [29] and Diffie et. al. [63]. In particular the former pointed to new classes of attacks, called "interleaving attacks," which they used to break existing protocols, and they suggested a protocol (2PP) defeated by none of the interleaving attacks they considered. Building on this, Bellare and Rogaway provide a model and a definition of security for two party symmetric session key distribution [18] and for three party session key distribution [21], just like we have for primitives like encryption and signatures. Based on this they derive protocols whose security can be proven based on standard cryptographic assumptions. It turns out the protocols are efficient too.

Now other well justified protocols are also emerging. For example, the SKEME protocol of Krawczyk [108] is an elegant and multi-purpose two party session key distribution protocol directed at fulfilling the key distribution needs of Internet security protocols. Even more recently, a proven-secure protocol for session key distribution in smart cards was developed by Shoup and Rubin [171].

9.2.3 An informal description of the problem

We normally think of a party in a protocol as being devoted to that protocol alone; it is not doing doing other things alongside. The main element of novelty in session key distribution is that parties may simultaneously maintain multiple sessions. A party has multiple *instances*. It is these instances that are the logical endpoints of a session, not the party itself.

[2] Insofar as there were no formal statements of what this protocol was supposed to do, it is not entirely fair to call it buggy; but the authors themselves regarded the protocol as having a problem worthy of fixing [132].

We let $\{P_1, \ldots, P_N\}$ denote the parties in the distributed system. As discussed above, a given pair of players, P_i and P_j may simultaneously maintain multiple sessions (each with its own session key). Thus it is not really P_i and P_j which form the logical endpoints of a secure session; instead, it is an *instance* $\Pi^s_{i,j}$ of P_i and an *instance* $\Pi^t_{j,i}$ of P_j. We emphasize instances as a central aspect of the session key distribution problem, and one of the things that makes session key distribution different from many other problems.

It is the goal of a *session-key distribution protocol* to provide $\Pi^s_{i,j}$ and $\Pi^t_{j,i}$ with a session key $\sigma^{s,t}_{i,j}$ to protect their session. Instances $\Pi^s_{i,j}$ and $\Pi^t_{j,i}$ must come up with this key without knowledge of s, t, or whatever other instances may currently exist in the distributed system.

An active adversary attacks the network. She controls all the communication among the players: she can deliver messages out of order and to unintended recipients, concoct messages entirely of her own choosing, and start up entirely new instances of players. Furthermore, she can mount various attacks on session keys which we will discuss.

9.2.4 Issues in security

Ultimately, what we want to say is that the adversary cannot compromise a session key exchanged between a pair of instances of the legitimate parties. We must worry about two (related) issues: authentication, and key secrecy. The first means, roughly, that when an instance of i accepts B then it must have been "talking to" an instance of j. The second, roughly, means that if $\Pi^s_{i,j}$ and $\Pi^t_{j,i}$ share a session key then this key must be secure.

It is an important requirement on session keys that the key of one session be independent of another. This is because we cannot make assumptions about how a session key will be used in an application. It might end up exposing it, and we want this not to compromise other session keys. We model this in a worst case way by allowing the adversary to expose session keys at will. Then we will say that a key shared between partners who are unexposed must remain secure even if keys of other sessions are exposed.

One of the most important issues is what is meant by security of the key. The way it has traditionally been viewed is that the key is secure if the adversary cannot compute it. We have by now, however, seen time and again, in a variety of settings, that this is not the right notion of secrecy. We must also prevent partial information from leaking. (Examples of why this is important for session keys are easy to find, analogous to the many examples we have seen previously illustrating this issue.) Accordingly, the definitions ask that a session key be unpredictable in the sense of a probabilistically encrypted message.

We note that insufficient protection of the session key is a flaw that is present in all session key distribution protocols of which we are aware barring those of [18, 21]. In fact, this insecurity is often built in by a desire to have a property that is called "key confirmation." In order to "confirm" that it has received the session key, one party might encrypt a fixed message with it, and its ability to do so correctly is read by the other party as evidence that it has the right session key. But this reveals partial information about the key. It might seem un-important, but one can find examples of usages of the session key which are rendered insecure by this kind of key confirmation. In fact "key confirmation," if needed at all, can be achieved in other ways.

9.2.5 Entity authentication versus key distribution

The goal of the key distributions we are considering is for the parties to simultaneously authenticate one another and come into possession of a secure, shared session key. There are several ways one might interpret the notion of authentication.

The literature has considered two ways. The first is authentication in a very strong sense, considered in [18] for the two party case. This has been relaxed to a weaker notion in [21], considered for the three party case. The weaker notion for the two party case is still under research and development.

Which to prefer depends on the setting. The approach we will follow here is to follow the existing literature. Namely we will consider the stronger notion for the two party setting, and the weaker one for the three party setting. It may perhaps be more correct to use the weaker notion throughout, and in a future version of these notes we would hope to do so; the situation at present is simply that the formalizations of the weaker notion for the two party case have not yet appeared.

9.3 Authenticated key exchanges

We begin by looking at the two party case, both symmetric and asymmetric. We look at providing authentic exchange of a session key, meaning the parties want to authenticate one another and simultaneously come into possession of a shared secret session key. The formal model, and the definition of what constitutes a secure authenticated session key distribution protocol, are provided in [18]. Here we will only describe some protocols.

First however let us note some conventions. We assume the parties want to come into possession of a l-bit, random shared secret key, eg. $l = 56$. (More generally we could distribute a key from some arbitrary samplable distribution, but for simplicity let's stick to what is after all the most common case.) The session key will be denoted by α.

Whenever a party A sends a flow to another party B, it is understood that her identity A accompanies the flow, so that B knows who the flow purports to come from. (This has nothing to do with cryptography or security: it is just a service of the communication medium. Note this identity is not secured: the adversary can change it. If the parties want the claim secured, it is their responsibility to use cryptography to this end, and will see how they do this.)

9.3.1 The symmetric case

Let K be the (long-lived) key shared between the parties. We fix a private key encryption scheme $(\mathcal{E}, \mathcal{D})$ and a private key message authentication scheme (Tag, Vf). The key K is divided into two parts, K^e and K^m, the first to be used for encryption and the second for message authentication. The protocol, called Authenticated Key Exchange Protocol 1, is depicted in Figure 9.1, and a more complete description of the flows follows.

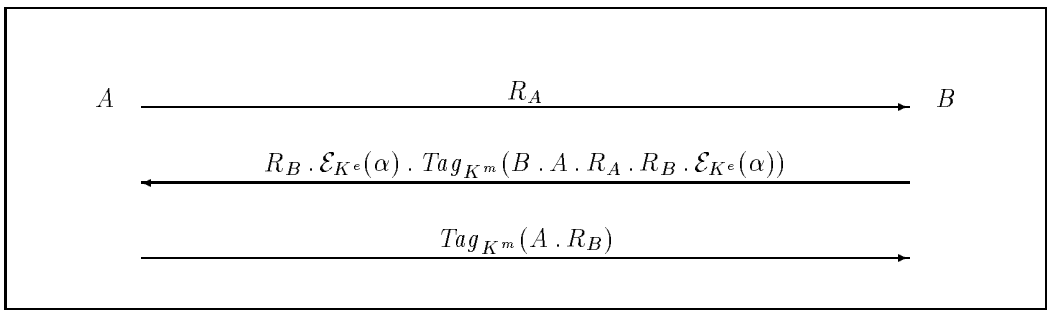

Figure 9.1: Protocol AKEP1: Session key distribution in symmetric setting.

Here is a more complete description of the flows:

(1) A picks at random a string R_A and sends it to B

(2) B picks at random a string R_B. She also picks at random an l-bit session key α. She encrypts it under K^e to produce the ciphertext $C = \mathcal{E}_{K^e}(\alpha)$. She now computes the tag $\mu = Tag_{K^m}(B \,.\, A \,.\, R_A \,.\, R_B \,.\, C)$. She sends R_B, C, μ to A.

(3) A verifies that $Vf_{K^m}(B \,.\, A \,.\, R_A \,.\, R_B \,.\, C, \mu) = 1$. If this is the case she computes the tag $Tag_{K^m}(A \,.\, R_B)$ and sends it to B. She also decrypts C via $\alpha = \mathcal{D}_{K^e}(C)$ to recover the session key.

(4) B verifies the last tag and accepts (outputting session key α) if the last tag was valid.

Remark 9.3.1 Notice that both encryption and message authentication are used. As we mentioned above, one of the commonly found fallacies in session key distribution protocols is to try to use encryption to provide authentication. One should really use a message authentication code.

Remark 9.3.2 It is important that the encryption scheme $(\mathcal{E}, \mathcal{D})$ used above be secure in the sense we have discussed in Chapter 5. Recall in particular this means it is probabilistic. A single plaintext has many possible ciphertexts, depending on the probabilistic choices made by the encryption algorithms. These probabilistic choices are made by S when the latter encrypts the session key, independently for the two encryptions it performs. This is a crucial element in the security of the session key.

These remarks apply also to the protocols that follow, appropriately modified, of course, to reflect a change in setting. We will not repeat the remarks.

9.3.2 The asymmetric case

We will be using public key cryptography. Specifically, we will be using both public key encryption and digital signatures.

Fix a public key encryption scheme, and let $\mathcal{E}, \mathcal{D}$ denote, respectively, the encryption and the decryption algorithms for this scheme. The former takes a public encryption key pk^e and message to return a ciphertext, and the latter takes the secret decryption key sk^e and ciphertext to return the plaintext. This scheme should be secure in the sense we have discussed in Chapter 6.

Fix a digital signature scheme, and let $Sign$, Vf denote, respectively, the signature and verification algorithms for this scheme. The former takes a secret signing key sk^d and message to return a signature, and the latter takes the public verification key pk^d, message, and candidate signature to return an indication of whether or not the signature is valid. This scheme should be secure in the sense we have discussed in Chapter 8.

Every user I in the system has a public key pk_I which is in fact a pair of public keys, $pk_I = (pk_I^e, pk_I^d)$, one for the encryption scheme and the other for the signature scheme. These keys are known to all other users and the adversary. However, the user keeps privately the corresponding secret keys. Namely he holds $sk_I = (sk_I^e, sk_I^d)$ and nobody else knows these keys.

Recall the model is that A has B's public key pk_B and B has A's public key pk_A. The protocol for the parties to get a joint, shared secret key α is depicted in Figure 9.2, and a more complete explanation follows.

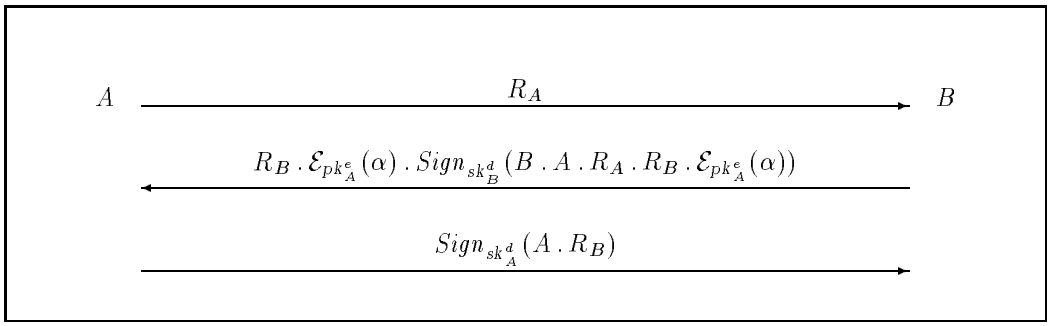

Figure 9.2: Protocol for exchange of symmetric key in asymmetric setting.

Here is a more complete description of the flows:

(1) A picks at random a string R_A and sends it to B

(2) B picks at random a string R_B. She also picks at random an l-bit session key α. She encrypts it under A's public key pk_A^e to produce the ciphertext $C = \mathcal{E}_{pk_A^e}(\alpha)$. She now computes the signature $\mu = Sign_{sk_B^d}(A \,.\, R_A \,.\, R_B \,.\, C)$, under her secret signing key sk_B^d. She sends R_B, C, μ to A.

(3) A verifies that $Vf_{pk_B^d}(A \,.\, R_A \,.\, R_B \,.\, C, \mu) = 1$. If this is the case she computes the signature $Sign_{sk_A^d}(R_B)$ and sends it to B. She also decrypts C via $\alpha = \mathcal{D}_{sk_A^e}(C)$ to recover the session key.

(4) B verifies the last signature and accepts (outputting session key α) if the last signature was valid.

9.4 Three party session key distribution

Fix a private key encryption scheme $(\mathcal{E}, \mathcal{D})$ which is secure in the sense discussed in Chapter 5. Also fix a message authentication scheme (Tag, Vf) which is secure in the sense discussed in Chapter 7. The key K_I shared between the server S and party I is a pair (K_I^e, K_I^m) of keys, one a key for the encryption scheme and the other a key for the message authentication scheme. We now consider parties A, B, whose keys K_A and K_B, respectively have this form. A terse representation of the protocol of [21] is is given in Figure 9.3, and a more complete explanation follows.

Flow 1.	$A \rightarrow B$:	R_A
Flow 2.	$B \rightarrow S$:	$R_A \,.\, R_B$
Flow 3A.	$S \rightarrow A$:	$\mathcal{E}_{K_A^e}(\alpha) \;.\; Tag_{K_A^m}(A \,.\, B \,.\, R_A \,.\, \mathcal{E}_{K_A^e}(\alpha))$
Flow 3B.	$S \rightarrow B$:	$\mathcal{E}_{K_B^e}(\alpha) \;.\; Tag_{K_B^m}(A \,.\, B \,.\, R_B \,.\, \mathcal{E}_{K_B^e}(\alpha))$

Figure 9.3: Three party session key distribution protocol.

Here now is a more complete description of the flows and accompanying computations:

(1) In Step 1, party A chooses a random challenge R_A and sends it to B.

(2) In Step 2, party B chooses a random challenge R_B and sends $R_A \,.\, R_B$ to S.

(3) In Step 3, S picks a random l-bit session key α which he will distribute. Then S encrypts this session key under each of the parties' shared keys. Namely he computes the ciphertexts $\alpha_A = \mathcal{E}_{K_A^e}(\alpha)$ and $\alpha_B = \mathcal{E}_{K_B^e}(\alpha)$. Then S computes $\mu_A = Tag_{K_A^m}(A \,.\, B \,.\, R_A \,.\, \alpha_A)$ and $\mu_B = Tag_{K_B^m}(A \,.\, B \,.\, R_B \,.\, \alpha_B)$. In flow 3A (resp. 3B) S sends A (resp. B) the message $\alpha_A \,.\, \mu_A$ (resp. $\alpha_B \,.\, \mu_B$).

(4) In Step 4A (resp. 4B) Party A (resp. B) receives a message $\alpha'_A \,.\, \mu'_A$ (resp. $\alpha'_B \,.\, \mu'_B$) and accepts, with session key $\mathcal{D}_{K_A^e}(\alpha'_A)$ (resp. $\mathcal{D}_{K_B^e}(\alpha'_B)$), if and only if $Vf_{K_A^m}(A \,.\, B \,.\, R_A \,.\, \alpha'_A, \mu'_A) = 1$ (resp. $Vf_{K_B^m}(A \,.\, B \,.\, R_B \,.\, \alpha'_B, \mu'_A) = 1$).

Remark 9.4.1 This protocol has four flows. Typically, the three party key distribution protocols you will see in the literature have five. In fact, four suffices.

9.5 Forward secrecy

Forward secrecy is an extra security property that a session key can have and which seems very desirable.

Consider, for concreteness, the protocol of Figure 9.2 for exchange of a symmetric key in the asymmetric setting. Suppose A and B have run this protocol and exchanged a session key α, and used it to encrypt data. Suppose the adversary recorded the transcript of this exchange. This means she has in her possession $C = \mathcal{E}_{pk_A^e}(\alpha)$, the encrypted session key, and also any ciphertexts encrypted under α that the parties may have transmitted, call them $C_1, C_2, \ldots$. Since the session key distribution protocol is secure, the information she has doesn't give her anything; certainly she does not learn the session key α.

Now that session is over. But now suppose, for some reason, the *long lived* key of A is exposed. Meaning the adversary, somehow, gets hold of $sk_A = (sk_A^e, sk_A^d)$.

Certainly, the adversary can compromise all future sessions of A. Yet in practice we would expect that A would soon realize her secret information is lost and revoke her public key $pk_A = (pk_A^e, pk_A^d)$ to mitigate the damage. However, there is another issue. The adversary now has sk^e and can decrypt the ciphertext C to get α. Using this, she can decrypt $C_1, C_2, \ldots$ and thereby read the confidential data that the parties sent in the past session.

This does not contradict the security of the basic session key distribution protocol which assumed that the adversary does not gain access to the long-lived keys. But we might ask for a new and stronger property. Namely that even if the adversary got the long-lived keys, at least *past* sessions would not be compromised. This is called *forward secrecy.*

Forward secrecy can be accomplished via the Diffie-Hellman key exchange with which we began this chapter. Let us give a protocol. We do so in the asymmetric, two party setting; analogous protocols can be given in the other settings. The protocol we give is an extension of the STS protocol of [63]. It is depicted in Figure 9.4 and a more complete explanation follows.

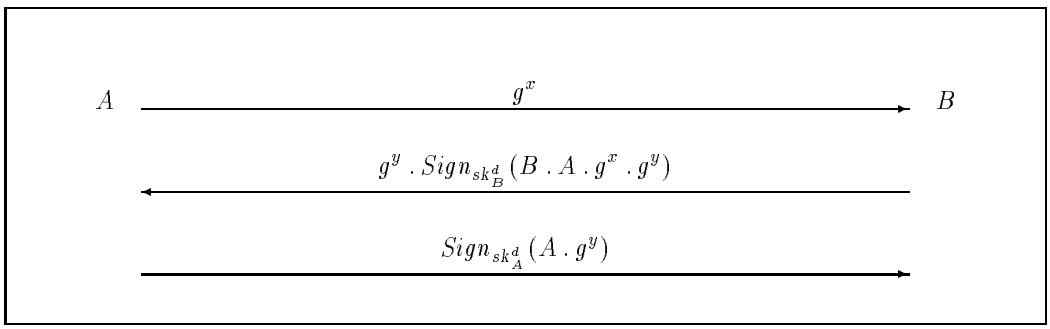

Figure 9.4: Protocol for exchange of symmetric key with forward secrecy.

Here is a more complete description of the flows:

(1) A picks at random a string x, computes $X = g^x$, and sends it to B

(2) B picks at random a string y and lets $Y = g^y$. She now computes the signature $\mu = Sign_{sk_B^d}(A \,.\, X \,.\, Y)$, under her secret signing key sk_B^d. She sends Y, μ to A.

(3) A verifies that $Vf_{pk_B^d}(A \,.\, X \,.\, Y, \mu) = 1$. If this is the case she computes the signature $Sign_{sk_A^d}(Y)$ and sends it to B. She also decrypts outputs the DH key $g^{xy} = Y^x$ as the session key.

(4) B verifies the last signature and accepts (outputting session key $g^{xy} = X^y$) if the last signature was valid.

The use of the DH secret key exchange protocol here is intriguing. Is that the *only* way to get forward secrecy? It turns out it is. Bellare and Rogaway have noted that secret key exchange is not only sufficient but also necessary for the forward secrecy property.

As we noted in Section **??**, the DH key is not by itself a good key because we cannot guarantee bit security. Accordingly, the session key in the above should actually be set to, say, $H(g^{xy})$ rather than g^{xy} itself, for a "good" hash function H.

CHAPTER 10

Protocols

Classical cryptography is concerned with the problem of security communication between users by providing privacy and authenticity. The need for an underlying infrastructure for key management leads naturally into the topic of key distribution. For many years this is all there was to cryptography.

One of the major contributions of modern cryptography has been the development of advanced *protocols*. These protocols enable users to electronically solve many real world problems, play games, and accomplish all kinds of intriguing and very general distributed tasks. Amongst these are zero-knowledge proofs, secure distributed computing, and voting protocols. The goal of this chapter is to give a brief introduction to this area.

10.1 Some two party protocols

We make reference to some number theoretic facts in Section B.6.

10.1.1 Oblivious transfer

This protocol was invented by M. Rabin [148].

An *oblivious transfer* is an unusual protocol wherein Alice transfers a secret bit m to Alice in such a way that the bit is transferred to Bob with probability 1/2;Bob knows when he gets the bit, but Alice doesn't know whether it was transferred or not.

This strange-sounding protocol has a number of useful applications (see, for example [148, 28]). In fact, Kilian has shown [106] that the ability to perform oblivious transfers is a sufficiently strong primitive to enable *any* two-party protocol to be performed.

The following implementation for oblivious transfer has been proposed in the literature (related ideas due to Rabin and Blum.)

(1) Alice picks two primes p, q at random and multiplies them to produce the modulus $N = pq$. She encrypts the message m under this modulus in some standard way, having the property that if you know p, q then you can decrypt, else you can't. She sends N and the ciphertext C to Bob.

(2) Bob picks $a \in Z_N^*$ at random and sends $w = a^2 \bmod N$ to Alice.

(3) Alice computes the four square roots $x, -x, y, -y$ of w, picks one at random and sends it back to Bob

(4) If Bob got back the root which is not $\pm a$ he can factor N and recover m. Else he can't.

And Alice doesn't know which happened since a was random.

It is fairly clear that there is no way for A to cheat in this protocol, since A does not know which square root of z B knows, as x was chosen at random. On first sight it looks like B cannot get anything either, since he only obtains a square root of a random square. However, a formal proof of this fact is not known. It is

not clear whether B can cheat or not. For example, if B chooses a particular value of z instead of choosing x at random and setting $z = x^2 \pmod n$, then this may lead to an advantage in factoring n. It is conceivable, for example, that knowing a square root of (n-1)/2 mod n (or some other special value) could allow B to factor n. Thus condition ii) is satisfied, but we can't prove whether or not the first condition is satisfied.

If we had a method by which B could **prove** to A that he indeed followed the protocol and choose x at random without revealing what x is, the protocol could be modified to provably work. We will see in a later section on zero-knowledge proofs on how such proofs can be done.

There is another form of OT called 1 out of 2 OT. Here Alice has two secrets, m_0 and m_1. Bob has a selection bit c. At the end of the protocol, Bob gets b_c and Alice still does not know c. See [71].

10.1.2 Simultaneous contract signing

Alice and Bob want to sign the contract, but only if the other person does as well. That is, neither wants to be left in the position of being the only one who signs. Thus, if Alice signs first, she is worried Bob will then not sign, and vice versa. (Maybe easier to think of having two contracts, the first promising something to Alice, the second to Bob. It is a trade. Obviously, each wants the other one to sign.) This problem was proposed in [71].

One approach is that Alice signs the first letter of her name and sends the contract to Bob. He does likewise, and sends it back. And so on. Assume their names have the same length. Then this makes some progress towards a solution. Of course the problem is the person who must go last can stop. But you can make this a negligible difference. For example, not a letter at a time, but a few millimeters of the letter at a time. No party is ever much ahead of the other. If at some point they both stop, they both are at about the same point.

Electronically, we are exchanging strings, which are digital signatures of the contract. Alice has signed it to produce σ_A and Bob has signed it to produce σ_B. Now they exchange these strings a bit at a time, each time sending one more bit.

There is a problem with this. What if one person does not send the signature, but just some garbage string? The other will not know until the end. Even, Goldreich and Lempel [71] show how oblivious transfer can be used to fix this.

Alice creates L_A which is the signature of the contract together with the phrase "this is my signature of the left half of the contract." Similarly she creates R_A which is the signature of the contract together with the phrase "this is my signature of the right half of the contract." Similarly, Bob creates L_B and R_B.

Also Alice picks two DES keys, K_A^L and K_A^R, and encrypts L, R respectively to produce C_A^L and C_A^R. Similarly for Bob, replacing As by Bs.

The contract is considered signed if you have both halves of the other person's signature.

All the ciphertexts are sent to the other party.

Alice 1 out of two OTs (K_A^L, K_A^R) to Bob with the latter choosing a random selection bit, and vice versa. Say Bob gets K_A^L and Alice gets K_B^R.

Alice and Bob send each the first bits of both DES keys. Keep repeating until all bits of all keys are sent. In this phase, if a party catches a mistake in the bits corresponding to the key it already has, it aborts, else it continues.

10.1.3 Bit Commitment

Bob wants Alice to commit to some value, say a bid, so that she can't change this at a later time as a function of other things. On the other hand, Alice does not want Bob to know, at this time, what is the value she is committing to, but will open it up later, at the right time.

Alice makes up an "electronic safe." She has a key to it. She puts the value in the safe and sends the safe to Bob. The latter can't open it to extract the contents. This is a committal. Later, Alice will decommit by sending the key. Now Bob can open it. What must be true is that Alice can't produce a safe having two keys, such that either open it, and when you look inside, you see different values.

One way to implement this is via collision-free hashing. To commit to x Alice sends $yH(x)$. From this Bob can't figure out x, since H is one-way. To decommit, Alice sends x and Bob checks that $H(x) = y$. But

Alice can't find $x' \neq x$ such that $H(x') = y$, so can't cheat.

This however has poor bit security. You can fix it with hardcore bits.

Another way is to use quadratic residues. First, we fix a particular number $y \in Z_N^*$ which is known to be a non-residue. Commit to a 0 by sending a random square mod N, namely x^2, and to a 1 by sending a random non-square mod N, in the form yx^2. The QRA says Bob can't tell which is which. To decommit, reveal x in either case.

Notice the QR commitment scheme is secure even against a sender who has unbounded computing power. But not the receiver.

Can you do the opposite? Yes, use discrete logarithms. Let p be a known prime, $g \in Z_p^*$ a known generator of Z_p^*, and $s \in Z_p^*$ a known element of unknown discrete logarithm, namely $\log_g(s)$ is not known. Commit to 0 by picking x at random and sending $y = g^x$; to a 1 by sending sg^x. Notice that to the receiver, each is a random element of the range. But what if the sender could create a y that could be opened both ways? It would have the discrete logarithm of s.

Commitment schemes are useful for lots of things. In particular, ZK proofs, but also coin flipping.

10.1.4 Coin flipping in a well

Blum [32] has proposed the problem of *coin flipping over the telephone*. Alice and Bob want a fair, common, coin. They want to take a random choice, but neither should be able to dictate it. Heads Alice wins, and tails Bob wins.

What if Bob says, "I'll flip the coin and send you the value." No good. Bob will just flip to win. They must both influence the value.

Here is a thought. Alice picks a random bit a and sends it to Bob, and Bob picks a random bit b and send it to Alice, and the value of the coin is $a \oplus b$. The problem is who goes first. If Alice goes first, Bob will choose b to make the coin whatever he wants. Not fair.

So what Alice does is first commit to her coin. She sends $y = \text{Committ}(a)$ to Bob. Now Bob can't make b a function of a. He sends back b, in the clear. Alice may want to make a a function of b, but it is too late since a is committed to. She decommits, and the coin is $a \oplus b$.

10.1.5 Oblivious circuit evaluation

Alice and Bob want to know which of them is older. But neither wants to reveal their age. (Which means they also don't want the reveal the age difference, since from this and their own age, each gets the other's age too!) They just want a single bit to pop out, pointing to the older one.

Sometimes called the Millionaires problem, with the values being the earning of each millionaire.

In general, the problem is that Alice has an input x_A and Bob has an input x_B and they want to compute $f(x_A, x_B)$ where f is some known function, for example $f(x_A, x_B) = 1$ if $x_A \geq x_B$ and 0 otherwise. They want to compute it obliviously, so that at the end of the game they both have the value $v = f(x_A, x_B)$ but neither knows anything else.

There are protocols for this task, and they are quite complex. We refer the reader to [10, 39]

10.1.6 Simultaneous Secret Exchange Protocol

This has been studied in [33, 181, 119, 186].

The protocol given here is an example of a protocol that seems to work at first glance, but is in actuality open to cheating for similar reasons that the above oblivious transfer protocol was open to cheating. The common input consists of $1^k, \alpha \in E_{n_A}(s_A), \beta \in E_{n_B}(s_B), n_A$, and n_B, where n_A and n_B are each products of two equal size primes congruent to 3 mod 4; E_{n_A} (E_{n_B}) are the same encryption as in the oblivious transfer protocol above with respect to n_A (n_B respectively). A's private input has in it the prime factorization $n_A = p_A q_A$ of n_A and B's contains the same for n_B. What we want is for A and B to be able to figure out s_B and s_A at the "same time". We assume equal computing power and knowledge of algorithms. The suggested protocol of Blum [33] follows .

Step 1: A picks $a_1, a_2, ..., a_K$ at random in $Z^*_{n_B}$ and then computes $b_i = a_i^2 (mod\ n_B)$ for $1 \leq i \leq k$. B picks $w_1, w_2, ..., w_k$ at random in Z_{n_B} and then computes $x_i = w_{i^*} \pmod{n_A}$ for $1 \leq i \leq k$.

Step 2: A sends all the b_i's to B and B sends all the x_i's to A.

Step 3: For each x_i A computes y_i and z_i such that $y_i^2 = z_i^2 = x_i \pmod{n_A}$ but $y_i \neq \pm z_i \bmod n_B$. (Note: either y_i or z_i equals $\pm w_i$.) For each b_i, B computes c_i and d_i with similar restrictions. (Note: either c_i or d_i equal $\pm a_i$.

Step 4: While $1 \leq j \leq k$ A sends B the jth significant bit of y_i and z_i for $1 \leq i \leq k$. B sends A the jth significant bit of c_i and d_i for $1 \leq i \leq k$.

Step 5: After completing the above loop, A (and B) figure out the factorization of n_B (and n_A) with the information obtained in Step 4. (A computes $gcd(c_i - d_i, n_B)$ for each i and B computes gcd $(y_i - z_i, n_A)$ for each i. Using this information, they figure out s_B and s_A by decrypting α and β.

Why are k numbers chosen rather that just one? This is to prevent the following type of cheating on the A and B's behalf. Suppose only one x was sent to A. A could figure out y and z and then send the jth significant bits of y and a junk string to B in Step 4, hoping that $y = \pm w$ and A will not notice that junk is being sent. If $y = \pm w$ then B has no way of knowing that A is cheating until the last step, at which time A has all the information he needs to find s_B, but B has not gained any new information to find s_A. So A can cheat with a 50% chance of success. If, on the other hand, k different x's are sent to A, A has an exponentially vanishing chance of successfully cheating in this fashion. Namely $Prob(y_i = \pm w_i \forall i) \leq (\frac{1}{2})^k$.

Unfortunately, Shamir, and Håstad pointed out a way to successfully cheat at this protocol. If, instead of choosing the w_i's at random, A chooses w_1 at random, sets $x_1 = w_1^2 \pmod{n_B}$, and then sets $x_i = x_1/2^{i-1} \pmod{n_B}$, then after one iteration of Step 4, A has all of the information that he needs to factor n_B by the reduction of [91]. So, a seemingly good protocol fails, since B has no way to check whether A chose the x_is at random independently from each as specified in the protocol or not. Note: that this problem is similar to the problem which arose in the oblivious transfer protocol and can be corrected if A and B could check that each other was following the protocol.

10.2 Zero-Knowledge Protocols

The previous sections listed a number of cryptographic protocol applications and some problems they suffer from. In this section we review the theory that has been developed to prove that these protocols are secure, and to design protocols that are "provably secure by construction". The key idea is to reduce the general problem of two-party protocols to a simpler problem: How can A prove to B that x is in a language L so that no more knowledge than $x \in L$ is revealed. If this could be done for any $L \in NP$ A could prove to B that he followed the protocol steps. We proceed to define the loose terms "interactive proof" (or "proof by a protocol") and "zero knowledge".

10.2.1 Interactive Proof-Systems(IP)

Before defining notion of interactive proof-systems, we define the notion of interactive Turing machine.

Definition 10.2.1 An *interactive Turing machine (ITM)* is a Turing machine with a read-only input tape, a read-only random tape, a read/write worktape, a read-only communication tape, a write-only communication tape, and a write-only output tape. The random tape contains an infinite sequence of bits which can be thought of as the outcome of unbiased coin tosses, this tape can be scanned only from left to right. We say that an interactive machine *flips a coin* to mean that it reads the next bit from its random tape. The contents of the write-only communication tape can be thought of as *messages sent* by the machine; while the contents of the read-only communication tape can be thought of as *messages received* by the machine.

Definition 10.2.2 An *interactive protocol* is an ordered pair of ITMs (A, B) which share the same input tape; B's write-only communication tape is A's read-only communication tape and vice versa. The machines take turns in being active with B being active first. During its active stage, the machine first performs some internal computation based on the contents of its tapes, and second writes a string on its write-only communication tape. The i^{th} *message* of $A(B)$ is the string $A(B)$ writes on its write-only communication tape in i^{th} stage. At this point, the machine is deactivated and the other machine becomes active, unless the protocol has terminated. Either machine can terminate the protocol, by not sending any message in its active stage. Machine B *accepts* (or rejects) the input by entering an accept (or reject) state and terminating the protocol. The first member of the pair, A, is a computationally unbounded Turing machine. The *computation time* of machine B is defined as the sum of B's computation time during its active stages, and it is bounded by a polynomial in the length of the input string.

Definition 10.2.3 Let $L \in \{0,1\}^*$ We say that
L has an *interactive proof-system* if $\exists$ ITM V *s.t.*

1. $\exists\, ITM\ P\ s.t\ (P,V)$ is an interactive protocol and $\forall x \in L\ s.t\ |x|$ is sufficiently large the $prob(V\ \ accepts) > \frac{2}{3}$ (when probabilities are taken over coin tosses of V and P).
2. $\forall\, ITM\ P\ \ s.t\ (P,V)$ is an interactive protocol $\forall x \notin L\ s.t.\ |x|$ is sufficiently large $Prob(V\ \ accepts) > \frac{1}{3}$ (when probabilities are taken over coin tosses of V and P's).

Note that it does not suffice to require that the verifier cannot be fooled by the predetermined prover (such a mild condition would have presupposed that the "prover" is a trusted oracle). **NP** is a special case of interactive proofs, where the interaction is trivial and the verifier tosses no coins.

We say that (P,V) (for which condition 1 holds) is an *interactive proof-system* for L.
Define $IP = \{L \mid L$ has interactive proof$\}$.

10.2.2 Examples

Notation

Throughout the lecture notes, whenever an interactive protocol is demonstrated, we let $B \longrightarrow A$: denote an active stage of machine B, in the end of which B sends A a message. Similarly, $A \longrightarrow B$: denotes an active stage of machine A.

Example 1: (From Number Theory)

Let $Z_n^* = \{x < n, ; (x,n) = 1\}$
$QR = \{(x,n) \mid x < n, (x,n)$ and $\exists y\ \ s.t\ y^2 \equiv x \bmod n\}$
$QNR = \{(x,n) \mid x < n, (x,n)$ and $\nexists y\ \ s.t\ y^2 \equiv x \bmod n\}$
We demonstrate an interactive proof-system for QNR.
On input (x,n) to interactive protocol (A,B):
$B \longrightarrow A$: B sends to A the list $w_1 \cdots w_k$ where $k = \mid n \mid$ and

$$w_i = \begin{cases} z_i^2 \bmod n & \text{if } b_i = 1 \\ x \cdot z_i^2 \bmod n & \text{if } b_i = 0 \end{cases}$$

where B selected $z_i \in Z_n^*, b_i \in \{0,1\}$ at random.
$A \longrightarrow B$: A sends to B the list $c_1 \cdots c_k$ *s.t.*

$$c_i = \begin{cases} 1 & \text{if } w_i \text{ is a quadratic residue mod } n \\ 0 & \text{otherwise} \end{cases}$$

B accepts iff $\forall_{1 \le i \le k},\ c_i = b_i$
B interprets $b_i = c_i$ as evidence that $(x,n) \in QRN$;while $b_i \neq c_i$ leads him to reject.

We claim that (A,B) is an interactive proof-system for QNR. If $(x,n) \in QNR$, then w_i is a quadratic residue mod$n$ iff $b_i = 1$. Thus, the all powerful A can easily compute whether w_i is a quadratic residue

mod$n$ or not, compute c_i correctly and make B accept with probability 1. If $(x,n) \notin QNR$ and $(x,n) \in QR$ then w_i is a random quadratic residue mod$n$ regardless of whether $b_i = 0$ or 1. Thus, the probability that A (no matter how powerful he is) can send c_i *s.t* $c_i = b_i$, is bounded by $\frac{1}{2}$ for each i and probability that B accepts is at most $(\frac{1}{2})^k$.

Example 2: (From Graph Theory)

To illustrate the definition of an interactive proof, we present an interactive proof for *Graph Non-Isomorphism*. The input is a pair of graphs G_1 and G_2, and one is required to prove that there exists no 1-1 edge-invariant mapping of the vertices of the first graph to the vertices of the second graph. (A mapping π from the vertices of G_1 to the vertices G_2 is *edge-invariant* if the nodes v and u are adjacent in G_1 iff the nodes $\pi(v)$ and $\pi(u)$ are adjacent in G_2.) It is interesting to note that no short NP-proofs are known for this problem; namely Graph Non-isomorphism is *not known* to be in **NP**.

The interactive proof (A,B) on input (G_1, G_2) proceeds as follows:

$B \longrightarrow A$: B chooses at random one of the two input graphs, G_{α_i}

where $\alpha_i \in \{1,2\}$. B creates a random isomorphic copy of G_{α_i} and sends it to A. (This is repeated k times, for $1 \leq i \leq k$, with independent random choices.)

$A \longrightarrow B$: A sends B $\beta_i \in \{1,2\}$ for all $1 \leq i \leq k$.

B accepts iff $\beta_i = \alpha_i$ for all $1 \leq i \leq k$.

B interprets $\beta_i = \alpha_i$ as evidence that the graphs are not isomorphic; while $\beta_i \neq \alpha_i$ leads him to reject.

If the two graphs are not isomorphic, the prover has no difficulty to always answer correctly (i.e., a β equal to α), and the verifier will accept. If the two graphs are isomorphic, it is impossible to distinguish a random isomorphic copy of the first from a random isomorphic copy of the second, and the probability that the prover answers correctly to one "query" is at most $\frac{1}{2}$. The probability that the prover answers correctly all k queries is $\leq (\frac{1}{2})^k$.

10.2.3 Zero-Knowledge

Now that we have extended the notion of what is an efficient proof-system, we address the question of how much "knowledge" need to be transferred in order to convince a polynomial-time bounded verifier, of the truth of a proposition. What do we mean by "knowledge"? For example, consider SAT, the NP-complete language of satisfiable sentences of propositional calculus. The most obvious proof-system is one in which on logical formula F the prover gives the verifier a satisfying assignment I, which the verifier can check in polynomial time. If finding this assignment I by himself would take the verifier more than polynomial time (which is the case if $P \neq NP$), we say that the verifier gains additional knowledge to the mere fact that $F \in SAT$.

Goldwasser, Micali and Rackoff [88] make this notion precise. They call an interactive proof-system for language L zero-knowledge if $\forall x \in L$ whatever the verifier can compute after participating in the interaction with the prover, could have been computed in polynomial time on the input x alone by a probabilistic polynomial time Turing machine.

We give the technical definition of zero-knowledge proof-systems and its variants in section 10.2.4, and briefly mention a few interesting results shown in this area.

10.2.4 Definitions

Let (A,B) be an interactive protocol. Let *view* be a random variable denoting the verifier view during the protocol on input x. Namely, for fixed sequence of coin tosses for A and B, *view* is the sequences of messages exchanged between verifier and prover, in addition to the string of coin tosses that the verifier used. The string h denotes any private input that the verifier may have with the only restriction that its length is bounded by a polynomial in the length of the common input. (*view* is distributed over both A's and B's coin tosses).

We say that (A,B) is *perfect zero-knowledge* for L if there exists a probabilistic, polynomial time Turing machine M *s.t* $\forall x \in L$, for all $a > 0$, for all strings h such that $|h| < |x|^a$, the random variable $M(x,h)$ and *view* are identically distributed. ($M(x,h)$ is distributed over the coin tosses of M on inputs x and h).

We say that (A,B) is *statistically zero-knowledge* for L if there exists a probabilistic polynomial time Turing machine M *s.t* $\forall x \in L$, for all $a > 0$, for all strings h such that $|h| < |x|^a$,

$$\sum_{\alpha} |prob(M(x,h)=\alpha) - prob(view=\alpha)| < \frac{1}{|x|^c}$$

for all constants $c > 0$ and sufficiently large $|x|$.

Intuitively the way to think of statistically zero-knowledge protocols, is that an infinite power "examiner" who is given only polynomially large samples of $\{M(x,h)|M$'s coin tosses $\}$ and $\{view\ |A$'s and B's coin tosses$\}$ can't tell the two sets apart.

Finally, we say that a protocol (A,B) is computationally zero-knowledge if a probabilistic polynomial time bounded "examiner" given a polynomial number of samples from the above sets can not tell them apart. Formally,

We say that (A,B) is *computationally zero-knowledge* for L if $\exists$ probabilistic, polynomial time Turing machine M *s.t* $\forall$ polynomial size circuit families $C = \{C_{|x|}\}, \forall$ constants $a, d > 0$, for all sufficiently large $|x|$ s.t $x \in L$, and for all strings h such that $|h| < |x|^a$,

$$prob(C_{|x|}(\alpha) = 1|\alpha \text{ random in } M(x,h)) - prob(C_{|x|}(\alpha) = 1|\alpha \text{ random in } view(x))| < \frac{1}{|x|^d}$$

We say that L has *(computational/statistical/perfect) zero-knowledge proof-system* if

1. $\exists$ interactive proof-system (A,B) for L.
2. $\forall$ ITM's B', interactive protocol (A,B') is (computational/statistical/perfect) zero-knowledge for L.

Clearly,the last definition is the most general of the three. We thus let $KC[0] = \{L|L$ has computational zero-knowledge pr

10.2.5 If there exists one way functions, then NP is in KC[0]

By far, the most important result obtained about zero-knowledge is by Goldreich, Micali and Wigderson [84]. They show the following result.

Theorem[84]: if there exist (non-unifrom) polynomial-time indistinguishable encryption scheme then every NP language has a computational zero-knowledge interactive proof-system.

The non uniformity condition is necessary for technical reasons (i.e the encryption scheme should be secure against non-uniform adversary. see section 3.7). The latest assumption under which such encryption scheme exists is the existence of one-way functions (with respect to non-uniform adversary) by results of Imagliazzo-Levin-Luby and Naor.

The proof outline is to show a zero-knowledge proof system for an NP-complete language, graph three colorability. We outline the protocol here. Suppose the prover wish to convince the verifier that a certain input graph is three-colorable, without revealing to the verifier the coloring that the prover knows. The prover can do so in a sequence of $|E|^2$ stages, each of which goes as follows.

- The prover switchs the three colors at random (e.g., switching all red nodes to blue, all blue nodes to yellow, and all yellow nodes to red).
- The prover encrypts the color of each node, using a different probabilistic encryption scheme for each node, and show the verifier all these encryptions, together with the correspondence indicating which ciphertext goes with which vertex.
- The verifier selects an edge of the graph at random.
- The prover reveals the decryptions of the colors of the two nodes that are incident to this edge by revealing the corresponding decryption keys.
- The verifier confirms that the decryptions are proper, and that the two endpoints of the edge are colored with two different but legal colors.

(any private probabilistic encryption scheme which is polynomial time indistinguishable will work here) If the graph is indeed three-colorable (and the prover know the coloring), then the verifier will never detect any edge being incorrectly labeled. However, if the graph is not three-colorable, then there is a chance of at least $|E|^{-1}$ on each stage that the prover will be caught trying to fool the verifier. The chance that the prover could fool the verifier for $|E|^2$ stages without being caught is exponentially small.

Note that the history of our communications—in the case that the graph is three-colorable—consists of the concatenation of the messages sent during each stage. It is possible to prove (on the assumption that secure encryption is possible) that the probability distribution defined over these histories by our set of possible interactions is indistinguishable in polynomial time from a distribution that the verifier can create on these histories by itself, without the provers participation. This fact means that the verifier gains zero (additional) knowledge from the protocol, other than the fact that the graph is three-colorable.

The proof that graph three-colorability has such a zero-knowledge interactive proof system can be used to prove that every language in NP has such a zero-knowledge proof system.

10.2.6 Applications to User Identification

Zero knowledge proofs provide a revolutionary new way to realize passwords [89, 73]. The idea is for every user to store a statement of a theorem in his publicly readable directory, the proof of which only he knows. Upon login, the user engages in a zero-knowledge proof of the correctness of the theorem. If the proof is convincing, access permission is granted. This guarantees that even an adversary who overhears the zero-knowledge proof can not learn enough to gain unauthorized access. This is a novel property which can not be achieved with traditional password mechanisms. Fiat and Shamir [73] have developed variations on some of the previously proposed zero-knowledge protocols [89] which are quite efficient and particularly useful for user identification and passwords.

10.3 Multi Party protocols

In a typical multi-party protocol problem, a number of parties wish to coordinate their activities to achieve some goal, even though some (sufficiently small) subset of them may have been corrupted by an adversary. The protocol should guarantee that the "good" parties are able to achieve the goal even though the corrupted parties send misleading information or otherwise maliciously misbehave in an attempt to prevent the good parties from succeeding.

10.3.1 Secret sharing

Secret Sharing protocols were invented independently by Blakley and Shamir [30, 164]. In the multi-party setting, secret sharing is a fundamental protocol and tool.

The basic idea is protection of privacy of information by distribution. Say you have a key to an important system. You are afraid you might loose it, so you want to give it to someone else. But no single person can be trusted with the key. Not just because that person may become untrustworthy, but because the place they keep the key may be compromised. So the key is shared amongst a bunch of people.

Let's call the key the secret s. A way to share it amongst five people is split it up as $s = s_1 \oplus \cdots \oplus s_5$ and give s_i to person i. No one person can figure out s. Even more, no four people can to it: it takes all five. If they all get together they can recover s. (Once that is done, they may discard it, ie it may be a one time key! Because now everyone knows it.)

We call s_i a share. Who creates the shares? The original holder of s. Sometimes it is one of the n players, sometimes not. We call this person the dealer.

Notice that s_i must be given *privately* to the i-th player. If other players see it, then, of course, this doesn't work.

We may want something more flexible. Say we have n people. We want that any $t+1$ of them can recover the secret but no t of them can find out anything about it, for some parameter t. For example, say $n = 5$ and $t = 2$. Any three of your friends can open your system, but no two of them can. This is better since above if one of them looses their share, the system can't be opened.

Shamir's idea is to use polynomials [164]. Let F be a finite field, like Z_p^*. A degree t polynomial is of the form $f(x) = a_0 + a_1 x + \cdots + a_t x^t$ for coefficients $a_0, \ldots, a_t \in F$. It has $t+1$ terms, not t! One more term than the degree. Polynomials have the following nice properties:

- Interpolation: Given $t+1$ points on the polynomial, namely $(x_1, y_1), \ldots, (x_{t+1}, y_{t+1})$ where $x_1, \ldots, x_{t=1}$ are distinct and $y_i = f(x_i)$, it is possible to find $a_0, \ldots, a_t$. The algorithm to do this is called interpolation. You can find it in many books.
- Secrecy: Given any t points on the polynomial, namely $(x_1, y_1), \ldots, (x_t, y_t)$ where $x_1, \ldots, x_t$ are distinct and $y_i = f(x_i)$, one can't figure out anything about a_0. More precisely, for any value v, the number of polynomials satisfying these t constraints does not depend on v. (In fact there is exactly one of them.)

These makes them a tool for secret sharing. Associate to each player i a point $x_i \in F$, these points being all distinct. (So $|F| \geq n$). To share secret s, the dealer picks $a_1, \ldots, a_t$ at random, sets $a_0 = s$ and forms the polynomial $f(x) = a_0 + a_1 x + \cdots + a_t x^t$. Now he computes $s_i = f(x_i)$ and sends this *privately* to player i. Now if $t+1$ players get together they can figure out f and hence s; any set of at most t players can't figure out anything about s.

10.3.2 Verifiable Secret Sharing

Shamir's scheme suffers from two problems. If the dealer of the secret is dishonest, he can give pieces which when put together do not uniquely define a secret. Secondly, if some of the players are dishonest, at the reconstruction stage they may provide other players with different pieces than they received and again cause an incorrect secret to be reconstructed.

Chor, Goldwasser, Micali, and Awerbuch [51] have observed the above problems and showed how to achieve secret sharing based on the intractability of factoring which does not suffer from the above problems. They call the new protocol *verifiable secret sharing* since now every party can verify that the piece of the secret he received is indeed a proper piece. Their protocol tolerated up to $O(\log n)$ colluders. Benaloh [25], and others [84, 72] showed how to achieve verifiable secret sharing if any one-way function exists which tolerates a minority of colluders. In [24] it has been recently shown how to achieve verifiable secret sharing against a third of colluders using error correcting codes, without making cryptographic assumptions. This was improved to a minority of colluders in [149].

10.3.3 Anonymous Transactions

Chaum has advocated the use of *anonymous transactions* as a way of protecting individuals from the maintenance by "Big Brother" of a database listing all their transactions, and proposes using *digital pseudonyms* to do so. Using pseudonyms, individuals can enter into electronic transactions with assurance that the transactions can not be later traced to the individual. However, since the individual is anonymous, the other party may wish assurance that the individual is authorized to enter into the transaction, or is able to pay. [46, 49].

10.3.4 Multiparty Ping-Pong Protocols

One way of demonstrating that a cryptographic protocol is secure is to show that the primitive operations that each party performs can not be composed to reveal any secret information.

Consider a simple example due to Dolev and Yao [67] involving the use of public keys. Alice sends a message M to Bob, encrypting it with his public key, so that the ciphertext C is $E_B(M)$ where E_B is Bob's public encryption key. Then Bob "echos" the message back to Alice, encrypting it with Alice's public key, so that the ciphertext returned is $C' = E_A(M)$. This completes the description of the protocol.

Is this secure? Since the message M is encrypted on both trips, it is clearly infeasible for a *passive* eavesdropper to learn M. However, an *active* eavesdropper X can defeat this protocol. Here's how: the eavesdropper X overhears the previous conversation, and records the ciphertext $C = E_B(M)$. Later, X starts up a conversation with Bob using this protocol, and sends Bob the encrypted message $E_B(M)$ that he has recorded. Now Bob dutifully returns to X the ciphertext $E_X(M)$, which gives X the message M he desires!

The moral is that an adversary may be able to "cut and paste" various pieces of the protocol together to break the system, where each "piece" is an elementary transaction performed by a legitimate party during the protocol, or a step that the adversary can perform himself.

It is sometimes possible to *prove* that a protocol is invulnerable to this style of attack. Dolev and Yao [67] pioneered this style of proof; additional work was performed by Dolev, Even, and Karp [66], Yao [185], and Even and Goldreich [70]. In other cases a modification of the protocol can eliminate or alleviate the danger; see [152] as an example of this approach against the danger of an adversary "inserting himself into the middle" of a public-key exchange protocol.

10.3.5 Multiparty Protocols When Most Parties are Honest

Goldreich, Micali, and Wigderson [84] have shown how to "compile" a protocol designed for honest parties into one which will still work correctly even if some number less than half of the players try to "cheat". While the protocol for the honest parties may involve the disclosure of secrets, at the end of the compiled protocol none of the parties know any more than what they knew originally, plus whatever information is disclosed as the "official output" of the protocol. Their compiler correctness and privacy is based on the existence of trapdoor functions.

Ben-Or, Goldwasser and Wigderson [24] and Chaum, Crépeau, and Damgård [48] go one step further. They assume secret communication between pairs of users as a primitive. Making no intractability assumption, they show a "compiler" which, given a description (e.g., a polynomial time algorithm or circuit) of any polynomial time function f, produces a protocol which always computes the function correctly and guarantees that no additional information to the function value is leaked to dishonest players . The "compiler" withstands up to 1/3 of the parties acting dishonestly in a manner directed by a worst-case unbounded-computation-time adversary.

These "master theorems" promise to be very powerful tool in the future design of secure protocols.

10.4 Electronic Elections

Electronic Elections can be considered the typical example of secure multiparty computations. The general instance of such a problem is that there are m people, each of them with their own private input x_i and we want to compute the result of a n-ary function f over such values, without revealing them.

In the case of electronic elections the parties are the voters, their input a binary value, the function being computed is just a simple sum and the result is the tally.

In general, these are the properties that we would like our Election Protocols to have:

1. Only authorized voters can vote.
2. No one can vote more than once.
3. Secrecy of votes is maintained.
4. No one can duplicate anyone else's vote.
5. The tally is computed correctly.
6. Anybody should be able to check 5.
7. The protocol should be fault-tolerant, meaning it should be able to work even in the presence of a number of "bad" parties.
8. It should be impossible to coerce a voter into revealing how she voted (e.g. vote-buying)

Usually in in election protocols it is not desirable to involve all the voters V_i in the computation process. So we assume that there are n government centers $C_1, \ldots, C_n$ whose task is to collect votes and compute the tally.

10.4.1 The Merritt Election Protocol

Consider the following scheme by Michael Merritt [126].

Each center C_i publishes a public key E_i and keeps secret the corresponding secret key. In order to cast her vote v_j, each voter V_j chooses a random number s_j and computes,

$$E_1(E_2(\ldots E_n(v_j, s_j))) = y_{n+1,j} \tag{10.1}$$

(The need for the second index $n+1$ will become clear in a minute, for now it is just irrelevant.)

Now we have the values y's posted. In order from center C_n to center C_1, each center C_i does the following. For each $y_{i+1,j}$, C_i chooses a random value $r_{i,j}$ and broadcasts $y_{i,j'} = E_i(y_{i+1,j}, j)$. The new index j' is computed by taking a random permutation π_i of the integers $[1..n]$. That is $j' = \pi_i(j)$. C_i keeps the permutation secret.

At the end we have

$$y_{1,j} = E_1(E_2(\ldots E_n(y_{n+1,j}, r_{n,j}) \ldots r_{2,j})r_{1,j})$$

At this point, the verification cycle begins. It consists of two rounds of decryption in the order $C_1 \longrightarrow C_2 \ldots \longrightarrow C_n$.

The decrypted values are posted and the tally computed by taking the sums of the votes v_j's.

(1) and (2) are clearly satisfied. (3) is satisfied, as even if the votes are revealed, what is kept hidden is the connection between the vote and the voter who casted it. Indeed in order to reconstruct such link we need to know all the permutations π_i. (4) is not satisfied as voter V_1 can easily copy voter V_2, by for example casting the same encrypted string. (5) and (6) are satisfied using the random strings: during the first decryption rounds each center checks that his random strings appear in the decrypted values, making sure that all his ciphertexts are being counted. Also at the end of the second decryption round each voter looks for her string s_j to make sure her vote is being counted (choosing a large enough space for the random string should eliminate the risk of duplicates.) Notice that in order to verify the correctness of the election we need the cooperation of all the voters (a negative feature especially in large protocols.)

(7) requires a longer discussion. If we are concerned about the secrecy of the votes being lost because of parties going "bad", then the protocol is ideal. Indeed even if $n-1$ of the centers cooperate, they will not be able to learn who casted what vote. Indeed they need to know *all* the permutations π_i. However even if one of the government agencies fails, by for example crashing, the entire system falls apart. The whole election needs to be repeated.

(8) is not satisfied. Indeed the voter can be forced to reveal both v_j and s_j and she tries to lie about the vote she will be discovered since the declared values will not match the ciphertext $y_{n+1,j}$.

10.4.2 A fault-tolerant Election Protocol

In this section we describe a protocol which has the following features

- satisfies (4), meaning it will be impossible to copy other people vote (the protocol before did not)
- Does not require the cooperation of each voter to publicly verify the tally (better solution to (6) than the above)
- introduces fault-tolerance: we fix a threshold t and we assume that if there are less than t "bad" centers the protocol will correctly compute the tally and the secrecy of each vote will be preserved (better solution to (7) than the above.)

This protocol is still susceptible to coercion (requirement (8)). We will discuss this point at the end.

The ideas behind this approach are due to Josh Benaloh [27]. The protocol described in the following section is the most efficient one in the literature due to Cramer, Franklin, Schoemakers and Yung [56].

Homomorphic Commitments

Let B be a commitment scheme (a one-way function basically.)

We say that a commitment scheme B is $(+,\times)$-homomorphic if

$$B(X+Y) = B(X) \times B(Y)$$

One possible example of such commitment is the following (invented by Pedersen [138]):

Discrete-Log based Homomorphic Commitment: Let p be a prime of the form $p = kq+1$ and let g, h be two elements in the subgroup of order q. We assume nobody knows the discrete log in base g of h. To commit to a number m in $[1..q]$:

$$B_a(m) \;=\; g^a \; h^m \tag{10.2}$$

for a randomly chosen a modulo q. To open the commitment a and m must be revealed.

Notice that this is a $(+,\times)$-homomorphic commitment as:

$$B_{a_1}(m_1)B_{a_2}(m_2) = g^{a_1}h^{m_1}g^{a_2}h^{m_2} = g^{a_1+a_2}h^{m_1+m_2} = B_{a_1+a_2}(m_1+m_2)$$

For now on let E be an $(+,\times)$-homomorphic commitment scheme.

10.4.3 The protocol

For ease of presentation we will show the protocol in two version. First we assume that there is only one center. Then we show how to generalize the ideas to the case of many centers.

Vote Casting – 1 center

Assume for now that there is only one center C and let E be his encryption function.

Assuming the votes are either -1 or 1, each voter V_j encrypts his vote v_j by computing and posting $B_{a_j}(v_j)$ for a randomly chosen a_j. V_j also sends the values a_j and v_j to C encrypted.

The voter now must prove that the vote is correct (i.e. it's the encryption of a -1 or of a 1.) He does this by performing a zero-knowledge proof of validity.

For the discrete-log based homomorphic commitment scheme described above, here is a very efficient protocol. Let us drop the index j for simplicity.

For $v = 1$:

1. The voter V chooses at random a, r_1, d_1, w_2 modulo q. He posts $B_a(v) = g^a h$ and also posts $\alpha_1 = g^{r_1}(B_a(v)h)^{-d_1}$, $\alpha_2 = g^{w_2}$.
2. The center C sends a random challenge c modulo q
3. The voter V responds as follows: V computes $d_2 = c - d_1$ and $r_2 = w_2 + ad_2$ and posts d_1, d_2, r_1, r_2
4. The center C checks that
   - $d_1 + d_2 = c$
   - $g^{r_1} = \alpha_1(B_a(v)h)^{d_1}$
   - $g^{r_2} = \alpha_2(B_a(v)/h)^{d_2}$

For $v = -1$:

1. The voter V chooses at random a, r_2, d_2, w_1 modulo q. He posts $B_a(v) = g^a/h$ and also posts $\alpha_1 = g^{w_1}$, $\alpha_2 = g^{r_2}(B_a(v)/h)^{-d_2}$
2. The center C sends a random challenge c modulo q
3. The voter V responds as follows: V computes $d_1 = c - d_2$ and $r_1 = w_1 + ad_1$ and posts d_1, d_2, r_1, r_2
4. The center C checks that
   - $d_1 + d_2 = c$
   - $g^{r_1} = \alpha_1(B_a(v)h)^{d_1}$
   - $g^{r_2} = \alpha_2(B_a(v)/h)^{d_2}$

For now on we will refer to the above protocol as $\mathsf{Proof}(B_a(v))$.

Tally Computation – 1 center

At the end of the previous phase we were left with $B_{a_j}(v_j)$ for each voter V_j. The center reveals the tally $T = \sum_j v_j$ and also the value $A = \sum_j a_j$. Everybody can check that the tally is correct by performing the following operation:

$$B_A(T) = \prod_j B_{a_j}(v_j)$$

which should be true for the correct tally, because of the homomorphic property of B.

The 1-center version of the protocol however has the drawback that this center learns everybody's vote.

Vote Casting – n centers

Assume n centers $C_1, \ldots, C_n$ and let E_i be the encryption function of C_i.

In this case voter V_j encrypts the vote v_j in the following manner. First he commits to the vote by posting

$$B_j = B_{a_j}(v_j)$$

for a randomly chosen a_j modulo q. He also proves that this is a correct vote by performing $\mathsf{Proof}(B_{a_j}(v_j))$.

Then he shares the values a_j and v_j among the centers using Shamir's (t, n) threshold secret sharing. That is, he chooses random polynomials $H_j(X)$ and $A_j(X)$ of degree t such that $H_j(0) = v_j$ and $A_j(0) = a_j$. Let

$$R_j(X) = v_j + r_{1,j}X + \ldots + r_{t,j}X^t$$
$$S_j(X) = a_j + s_{1,j}X + \ldots + s_{t,j}X^t$$

The coefficients are all modulo q.

Now the voter sends the value $u_{i,j} = R_j(i)$ and $w_{i,j} = S_j(i)$ to the center C_i (encrypted with E_i.)

Finally he commits to the coefficients of the polynomial H_j by posting

$$B_{\ell,j} = B_{s_{\ell,j}}(r_{\ell,j})$$

The centers perform the following check

$$g^{w_{i,j}} h^{u_{i,j}} = B_j \prod_{\ell=1}^{t} (B_{\ell,j})^{i^\ell} \tag{10.3}$$

to make sure that the shares he received encrypted are correct.

Tally counting – n centers

Each center C_i posts the partial sums:

$$T_i = \sum_j u_{i,j}$$

this is the sum of the shares of the votes received by each player.

$$A_i = \sum_j w_{i,j}$$

this is the sum of the shares of the random string a_j used to commit to the vote by each player.

Anybody can check that the center is revealing the right stuff by using the homomorphic property of the commitment scheme B. Indeed it must hold that

$$g^{A_i} h^{T_i} = \prod_{j=1}^{m} \left(B_j \prod_{\ell=1}^{t} (B_{\ell,j})^{j^\ell} \right) \tag{10.4}$$

Notices that the correct T_i's are shares of the tally T in a (t, n) Shamir's secret sharing scheme. So it is enough to take $t + 1$ of them to interpolate the tally.

Notice: Equations (10.3) and (10.4) are valid only under the assumption that nobody knows the discrete log in base g of h. Indeed who knows some value can open the commitment B in both ways and so reveal incorrect values that satisfies such equations.

Analysis: Let's go through the properties one by one. (1) and (2) are clearly satisfied. (3) is satisfied assuming that at most t centers can cooperate to learn the vote. If $t+1$ centers cooperate, then the privacy of the votes is lost. (4) is true for the following reason: assume that V_1 is trying to copy the action of V_2. When it comes to the point of proving the correctness of the vote (i.e. perform $\mathsf{Proof}(B)$), V_1 will probably receive a different challenge c than V_2. He will not be able to answer it and he will be eliminated from the election. (5) is true under the discrete-log assumption (see note above.) (6) is true as anybody can check on the the ZK proofs and Equations (10.3) and (10.4). (7) is true as we need only $t+1$ good centers to reconstruct the tally.

It is easy to see that because we need $t+1$ good centers and at most t centers can be bad, the maximum number of corrupted centers being tolerated by the protocol is $\frac{n}{2}-1$.

(8) is *not* satisfied. This is because somebody could be coerced into revealing both a and v when posting the commitment $B_a(v)$.

10.4.4 Uncoercibility

The problem of coercion of voters is probably the most complicated one. What exactly does it mean? In how many ways can a coercer, try to force a voter to cast a given vote.

Let's try to simplify the problem. We will consider two possible kinds of coercer. One who contacts the voter *before* the election starts and one who contacts the voter *after* the election is concluded.

The "before" coercer has a greater power. He can tell the voter what vote to cast and also what randomness to use during the protocol. This basically would amount to **fix** the behavior of the voter during the protocol. If the voter does not obey, it will be easy for the coercer to detect such occurrence. There have been some solutions proposed to this problem that use some form of physical assumption. For example one could allow the voter to exchange a limited number of bits over a secure channel with the voting centers [26, 158]. This would hopefully prevent the coercer from noticing that the voter is not following his instructions. Or one could force the voter to use some tamper-proof device that encrypts messages for him, choosing the randomness. This would prevent the coercer from forcing the user to use some fixed coin tosses as the user has no control on what coins the tamper-proof device is going to generate.

The "after" coercer has a smaller power. He can only go to the voter and ask to see the vote v and the randomness ρ used by the voter during the protocol. Maybe there could be a way for the voter to construct different v' and ρ' that "match" his execution of the protocol. This is *not* possible in the protocol above (unless the voter solves the discrete log problem.) Recently however a protocol for this purpose has been proposed by Canetti and Gennaro [43]. They use a new tool called *deniable encryption* (invented by Canetti, Dwork, Naor and Ostrovsky [42]), which is a new form of public key probabilistic encryption E with the following property.

Let m be the message and r the coin tosses of the sender. The sender computes the ciphertext $c = E_r(m)$. After if somebody approaches him and asks for the value of m, the sender will be able to produce m' and r' such that $E_{r'}(m') = c$.

10.5 Digital Cash

The primary means of making monetary transactions on the Internet today is by sending credit card information or establishing an account with a vendor ahead of time.

The major opposition to credit card based Internet shopping is that it is not anonymous. Indeed it is susceptible to monitoring, since the identity of the customer is established every time he/she makes a purchase. In real life we have the alternative to use cash whenever we want to buy something without establishing our identity. The term *digital cash* describes cryptographic techniques and protocols that aim to recreate the concept of cash-based shopping over the Internet.

First we will describe a general approach to digital cash based on public-key cryptography. This approach was originally suggested by David Chaum [46]. Schemes based on such approach achieve the anonymity

property.

10.5.1 Required properties for Digital Cash

The properties that one would like to have from Digital Cash schemes, are at least the following:

- forgery is hard
- duplication should be either prevented or detected
- preserve customers' anonymity
- minimize on-line operations on large database

10.5.2 A First-Try Protocol

A Digital Cash scheme consists usually of three protocols. The **withdrawal protocol** which allows a User to obtain a digital coin from the Bank . A **payment protocol** during which the User buys goods from a Vendor in exchange of the digital coin. And finally a **deposit protocol** where the Vendor gives back the coin to the Bank to be credited on his/her account.

In the protocol below we assume that the Bank has a secret key SK_B to sign messages and that the corresponding public key PK_B is known to everybody else. With the notation $\{M\}_{SK}$ we denote the message M together with its signature under key SK.

Let's look at this possible digital cash protocol.

Withdrawal Protocol:

1. User tells Bank she would like to withdraw \$100.
2. Bank returns a \$100 bill which looks like this:

$$\{\text{I am a \$100 bill, \#4527})\}_{SK_B}$$

   and withdraws \$100 from User account

3. User checks the signature and if it is valid accepts the bill

Payment Protocol:

1. The User pays the Vendor with the bill.
2. The Vendor checks the signature and if it's valid accepts the bill.

Deposit Protocol:

1. The Vendor gives the bill to the Bank
2. The Bank checks the signature and if it's valid, credits the Vendor 's account.

Given some suitable assumption on the security of the signature scheme, it is clear that it is impossible to forge digital coins. However it is very easy to duplicate and double-spend the same digital coin several times. It is also cleat that anonymity is not preserved as the Bank can link the name of the User with the serial number appearing on the bill and know where the User spent the coin.

10.5.3 Blind signatures

Let's try to solve the anonymity problem first. This approach involves —em blind signatures. The user presents the bank with a bill inside a container. The bank signs the bill without seeing the contents of the bill. This way, the bank cannot determine the source of a bill when a merchant presents it for deposit.

A useful analogy: The user covers a check with a piece of carbon paper and then seals both of them inside an envelope. The user gives the envelope to the bank. The bank then signs the outside of the envelope with a ball-point pen and returns the envelope to the user (without opening it - acutally the bank is *unable* to open the envelope in the digital version). The user then removes the signed check from the envelope and can spend it. The bank has never seen what it signed, so it cannot associate it with the user when it is returned to be deposited, but it can verify the signature on the check and thus guarantee the validity of the check.

There is, *of course*, **a problem with this:** The bank can be fooled into signing phony bills. For example, a user could tell the bank he's making a \$1 withdrawal and then present a \$100 bill to be signed. The bank will, unknowingly, sign the \$100 bill and allow the user to cheat the bank out of \$99. We will deal with this problem later, for now let us show how to construct blind signatures.

10.5.4 RSA blind signatures

Recall the RSA signature scheme: if M is the message to be signed, then its signature is $s = M^{e^{-1}} \bmod n$ where n and e are publicly known values. The secret information that the bank possesses is the inverse of $e \bmod \phi(n)$, which we will denote by d. The signature can be verified by calculating $s^e \bmod n$ and verifying that it is equal to $M \bmod n$.

In the case of blind signatures, the User wants the Bank to provide him with s, without revealing M to the bank. Here is a possible anonymous withdrawal protocol. Let M be a \$100 bill.

Withdrawal Protocol:

1. User chooses some random number, $r \bmod n$.
2. User calculates $M' = M \cdot r^e \bmod n$.
3. User gives the Bank M.
4. The Bank returns a signature for M', say $s' = (M')^d \bmod n$. Note that
$$s' = (M')^d = M^d \cdot (r^e)^d = M^d \cdot r$$
5. The Bank debits the User account for \$100.
6. Since the User knows r, he can divide s' by r to obtain
$$s = M^d$$

The payment and deposit protocol remain the same as above. This solves the problem of preserving the User anonymity, as when the coin comes back to the Bank there is no link between it and the User it was issued to.

We still have two problems.

1. The bank can still be fooled into signing something that it shouldn't (like a \$100 bill that it thinks is a \$1 bill)
2. Coins can still be duplicated and double-spent

10.5.5 Fixing the dollar amount

One possible solution to the above problem is to have only one denomination (per public key, for example.) That is the Bank would have several public keys $PK1_B, \ldots$ and the signature using PKi_B would be valid only on bills of i dollars.

Another possibility is to use a "cut and choose" procedure:

1. User makes up 100 \$20 bills
2. blinds them all
3. gives them to the Bank
4. The Bank picks one to sign (at random), and requires that the User unblind all of the rest (by revealing the r's). Before the signature is returned, the Bank ensures that all of the bills that were unblinded were correct.

This way the User has only $\frac{1}{100}$ probability of cheating. Of course, one could set up the protocol to create an smaller cheating chance (by requiring that the user provided more blinded messages, for example).

So, now we have a protocol that satisfies the anonymity requirement and can provide sufficiently small possibilities for cheating. We still have to deal with the double-spending problem.

10.5.6 On-line digital cash

In the on-line version of digital cash schemes, one requires the Bank to record all of the bills it receives in a database. During the payment protocol the Vendor would transmit the bill to the Bank and ask if the bill was already received. If this is the first time the bill is being used then the Vendor accepts it, otherwise he will reject it.

Although this is a simple solution it incurs in a high communication overhead as now the payment protocol looks a lot like a credit card transaction, when the Vendor awaits for authorization to finish the trade. Also the size of the database to be managed by the Bank could be problematic.

Notice that we are *preventing* double-spending this way. We are going to show now a way to *detect* double-spending which does not require on-line verification.

10.5.7 Off-line digital cash

The idea behind off-line digital cash is the following. During the payment protocol the User is forced to write a "random identity string", or RIS, on the bill.

The RIS must have the following properties:

- must be different for every payment of the coin.
- only the User can create a valid RIS.
- two different RIS on the same coin should allow the Bank to retrieve the User name.

If the Bank receives two identical bills with different RIS values, then the User has cheated and the bank can identify him. If the Bank receives two identical bills with the same RIS values, then the Vendor has cheated. The above idea appeared first in [47].

Here is a possible solution. Let H to be a one-way hash function.

Withdrawal Protocol:

1. The User prepares 100 bills of \$20 which look like this:

$$M_i = \text{ (I'm \$20 bill, \#4527i,} y_{i,1}, y'_{i,1}, y_{i,2}, y'_{i,2}, \ldots, y_{i,K}, y'_{i,K})$$

where $y_{i,j} = H(x_{i,j})$, $y'_{i,j} = H(x'_{i,j})$, where $x_{i,j}$ and $x'_{i,j}$ are randomly chosen under the condition that

$$x_{i,j} \oplus x'_{i,j} = \text{ User name } \forall i, j$$

2. The User blinds all the M_i to random messages M_i' (using the blinding protocol outlined above) and sends them to the Bank .
3. The Bank asks the User to unblind 99 of the 100 blinded bills.
4. When the User unblinds them, he also reveals the appropriate $x_{i,j}$ and $x'_{i,j}$.
5. The Bank checks not only that the bills are indeed \$20 bills, but also that $y_{i,j} = H(x_{i,j})$, $y'_{i,j} = H(x'_{i,j})$ and $x_{i,j} \oplus x'_{i,j} =$ User name, for the unblinded bills.
6. The Bank returns a signature on the only blind message (say M'_{17})
7. The User retrieves the signature s_{17} on M_{17}.

From now on let us drop the index $i = 17$ for simplicity. The payment protocol is modified to force the User to produce a RIS on the coin. The RIS is going to be one of x_j or x'_j for each $j = 1, \ldots, K$. Which one is going to be depends on a random challenge from the Vendor .

Payment Protocol:

1. The User gives M, s to the Vendor .
2. The Vendor checks the Bank signature on the bill and if it is valid, answers with a random bit string of length K, $b_1 \ldots b_K$.
3. If $b_j = 0$ User reveals x_j, otherwise he reveals x'_j
4. The Vendor checks that $y_j = H(x_j)$ or $y'_j = H(x'_j)$, whichever is the case. If the above equalities hold, he accepts the bill.

Notice that the above properties or RIS are satisfied. Indeed the probability that in a different payment the same RIS is produced is 2^{-K} since the Vendor chooses the "challenge" at random. Only the User can produce a valid RIS since the function H is one-way. Finally two different RIS on the same coin leak the name of the User , as if two RIS are different there must be an index j for which we have both x_j and x'_j.

Deposit Protocol:

1. The Vendor brings the coin M, s, RIS back to the Bank .
2. The Bank verifies the signature and check if the coin M, s has already been returned to the Bank
3. If the coin is already in the database, the Bank compares the RIS's of the two coins. If the RIS are different then the User double-spent the coin, otherwise it is the Vendor who is trying to deposit the coin twice.

Bibliography

[1] ISO/IEC 9796. Information technology security techniques – digital signature scheme giving message recovery, 1991. International Organization for Standards.

[2] L. M. Adleman. On breaking generalized knapsack public key cryptosystems. In *Proc. 15th ACM Symp. on Theory of Computing*, pages 402–412, Boston, 1983. ACM.

[3] L. M. Adleman. Factoring numbers using singular integers. Technical Report TR 90-20, U.S.C. Computer Science Department, September 1990.

[4] L. M. Adleman and M. A. Huang. Recognizing primes in random polynomial time. In *Proc. 19th ACM Symp. on Theory of Computing*, pages 462–469, New York City, 1987. ACM.

[5] L. M. Adleman, C. Pomerance, and R. S. Rumely. On distinguishing prime numbers from composite numbers. *Ann. Math.*, 117:173–206, 1983.

[6] W. B. Alexi, B. Chor, O. Goldreich, and C. P. Schnorr. RSA/Rabin functions: certain parts are as hard as the whole. *SIAM J. Computing*, 17(2):194–209, April 1988.

[7] D. Angluin. Lecture notes on the complexity of some problems in number theory. Technical Report TR-243, Yale University Computer Science Department, August 1982.

[8] Eric Bach. How to generate factored random numbers. *SIAM J. Computing*, 17(2):179–193, April 1988.

[9] D. Balenson. *RFC 1423: Privacy Enhancement for Internet Electronic Mail: Part III – Algorithms, Modes, and Identifiers.* Internet Activities Board, February 1993.

[10] D. Beaver. Efficient multiparty protocols using circuit randomization. In J. Feigenbaum, editor, *Proc. CRYPTO 91*, pages 420–432. Springer, 1992. Lecture Notes in Computer Science No. 576.

[11] M. Bellare, R. Canetti, and H. Krawczyk. Keying hash functions for message authentication. In *Proceedings of Crypto 96*, Lecture Notes in Computer Science. Springer-Verlag, 1996.

[12] M. Bellare, R. Canetti, and H. Krawczyk. Pseudorandom functions revisited: The cascade construction. In *Proc. 37th IEEE Symp. on Foundations of Comp. Science*. IEEE, 1996.

[13] M. Bellare, O. Goldreich, and S. Goldwasser. Incremental cryptography: the case of hashing and signing. In Yvo G. Desmedt, editor, *Proc. CRYPTO 94*, pages 216–233. Springer, 1994. Lecture Notes in Computer Science No. 839.

[14] M. Bellare, R. Guérin, and P. Rogaway. XOR MACs: New methods for message authentication using block ciphers. Technical Report RC 19970, IBM Research Report, March 1995.

[15] M. Bellare, J. Kilian, and P. Rogaway. The security of cipher block chaining. In Yvo G. Desmedt, editor, *Proc. CRYPTO 94*, pages 341–358. Springer, 1994. Lecture Notes in Computer Science No. 839.

[16] M. Bellare and S. Micali. How to sign given any trapdoor function. In *Proc. 20th ACM Symp. on Theory of Computing*, pages 32–42, Chicago, 1988. ACM.

[17] M. Bellare and S. Micali. How to sign given any trapdoor permutation. *Journal of the ACM*, 39(1):214–233, January 1992.

[18] M. Bellare and P. Rogaway. Entity authentication and key distribution. In Douglas R. Stinson, editor, *Proc. CRYPTO 93*, pages 232–249. Springer, 1994. Lecture Notes in Computer Science No. 773.

[19] M. Bellare and P. Rogaway. Optimal asymmetric encryption. In *Proceedings of EUROCRYPT'94*, volume 950 of *Lecture Notes in Computer Science*. Springer-Verlag, 1994.

[20] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *First ACM Conference on Computer and Communications Security*, pages 62–73, Fairfax, 1994. ACM.

[21] M. Bellare and P. Rogaway. Provably secure session key distribution– the three party case protection. In *Proc. 27th ACM Symp. on Theory of Computing*, pages 57–66, Las Vegas, 1995. ACM.

[22] M. Bellare and P. Rogaway. The exact security of digital signatures: How to sign with RSA and Rabin. In *Proceedings of EUROCRYPT'96*, volume 1070 of *Lecture Notes in Computer Science*. Springer-Verlag, 1996.

[23] M. Bellare and P. Rogaway. Symmetric encryption. 1996. Work in progress.

[24] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for fault-tolerant distributed computing. In *Proc. 20th ACM Symp. on Theory of Computing*, pages 1–10, Chicago, 1988. ACM.

[25] J. Benaloh. Secret sharing homomorphisms: Keeping shares of a secret sharing. In A. M. Odlyzko, editor, *Proc. CRYPTO 86*. Springer, 1987. Lecture Notes in Computer Science No. 263.

[26] J. Benaloh and D. Tuinstra. Receipt-free secret ballot elections. In *26th ACM Symposium on Theory of Computing*, pages 544–553, 1994.

[27] Josh Benaloh. Verifiable secret ballot elections. Technical Report TR–561, Yale Department of Computer Science, September 1987.

[28] R. Berger, R. Peralta, and T. Tedrick. A provably secure oblivious transfer protocol. In T. Beth, N. Cot, and I. Ingemarsson, editors, *Proc. EUROCRYPT 84*, pages 379–386. Springer-Verlag, 1985. Lecture Notes in Computer Science No. 209.

[29] R. Bird, I. Gopal, A. Herzberg, P. Janson, S. Kutten, R. Molva, and M. Yung. Systematic design of two-party authentication protocols. In J. Feigenbaum, editor, *Proc. CRYPTO 91*, pages 44–61. Springer, 1992. Lecture Notes in Computer Science No. 576.

[30] G. R. Blakley. Safeguarding cryptographic keys. In *Proc. AFIPS 1979 National Computer Conference*, pages 313–317. AFIPS, 1979.

[31] L. Blum, M. Blum, and M. Shub. A simple unpredictable pseudo-random number generator. *SIAM J. Computing*, 15(2):364–383, May 1986.

[32] M. Blum. Coin flipping by telephone. In *Proc. IEEE Spring COMPCOM*, pages 133–137. IEEE, 1982.

[33] M. Blum. How to exchange (secret) keys. *Trans. Computer Systems*, 1:175–193, May 1983. (Previously published in ACM STOC '83 proceedings, pages 440–447.).

[34] M. Blum. Independent unbiased coin flips from a correlated biased source: A finite state Markov chain. In *Proc. 25th IEEE Symp. on Foundations of Comp. Science*, pages 425–433, Singer Island, 1984. IEEE.

[35] M. Blum and S. Goldwasser. An efficient probabilistic public-key encryption scheme which hides all partial information. In G. R. Blakley and D. C. Chaum, editors, *Proc. CRYPTO 84*, pages 289–302. Springer, 1985. Lecture Notes in Computer Science No. 196.

[36] M. Blum and S. Micali. How to generate cryptographically strong sequences of pseudo-random bits. *SIAM J. Computing*, 13(4):850–863, November 1984.

[37] M. Blum, A. De Santis, S. Micali, and G. Persiano. Noninteractive zero-knowledge. *SIAM J. Computing*, 20(6):1084–1118, December 1991.

[38] D. Boneh and R. Venkatesan. Hardness of computing the most significant bits of secret keys in Diffie-Hellman and related schemes. In *Proceedings of CRYPTO'96*, Lecture Notes in Computer Science. Springer-Verlag, 1996.

[39] Gilles Brassard and Claude Crépeau. Zero-knowledge simulation of boolean circuits. In A.M. Odlyzko, editor, *Proc. CRYPTO 86*, pages 223–233. Springer-Verlag, 1987. Lecture Notes in Computer Science No. 263.

[40] E. F. Brickell. Solving low density knapsacks. In D. Chaum, editor, *Proc. CRYPTO 83*, pages 25–37, New York, 1984. Plenum Press.

[41] M. Burrows, M. Abadi, and R. Needham. A logic for authentication. *ACM Transactions on computer systems*, 8(1).

[42] R. Canetti, C. Dwork, M. Naor, and R. Ostrovsky. Deniable encryption. Manuscript, 1996.

[43] R. Canetti and R. Gennaro. Incoercible multiparty computation. In *Proc. 37th IEEE Symp. on Foundations of Comp. Science*, 1996. To appear.

[44] E.R. Canfield, P. Erdös, and C. Pomerance. On a problem of Oppenheim concerning 'Factorisatio Numerorum'. *J. Number Theory*, 17:1–28, 1983.

[45] M. Cerecedo, T. Matsumoto, and H. Imai. Efficient and secure multiparty generation of digital signatures based on discrete logarithm. *IEICE Trans. on Fund. Electr. Comm. and Comp. Sci.*, E76–A(4):532–545, 1993.

[46] D. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24:84–88, February 1981.

[47] D. Chaum, A. Fiat, and M. Naor. Untraceable electronic cash. In S. Goldwasser, editor, *Proc. CRYPTO 88*, pages 319–327. Springer-Verlag, 1988. Lecture Notes in Computer Science No. 403.

[48] David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols. In Carl Pomerance, editor, *Proc. CRYPTO 87*, pages 462–462. Springer-Verlag, 1988. Lecture Notes in Computer Science No. 293.

[49] David L. Chaum. Verification by anonymous monitors. In Allen Gersho, editor, *Advances in Cryptology: A Report on CRYPTO 81*, pages 138–139. U.C. Santa Barbara Dept. of Elec. and Computer Eng., 1982. Tech Report 82-04.

[50] B. Chor and O. Goldreich. Unbiased bits from sources of weak randomness and probabilistic communication complexity. *SIAM J. Computing*, 17(2):230–261, April 1988.

[51] B. Chor, S. Goldwasser, S. Micali, and B. Awerbuch. Verifiable secret sharing and achieving simultaneity in the presence of faults. In *Proc. 26th IEEE Symp. on Foundations of Comp. Science*, pages 383–395, Portland, 1985. IEEE.

[52] B. Chor and R. L. Rivest. A knapsack type public-key cryptosystem based on arithmetic in finite fields. *IEEE Trans. Inform. Theory*, 34(5):901–909, September 1988.

[53] D. Coppersmith. Evaluating logarithms in $GF(2^n)$. In *Proc. 16th ACM Symp. on Theory of Computing*, pages 201–207, Washington, D.C., 1984. ACM.

[54] D. Coppersmith, M. Franklin, J. Patarin, and M. Reiter. Low-exponent RSA with related messages. Technical Report IBM RC 20318, IBM T.J. Watson Research Lab, December 27, 1995. (To appear in Eurocrypt '96).

[55] T. Cormen, C. Leiserson, and R. Rivest. *Introduction to Algorithms*. MIT Press/McGraw-Hill, 1990.

[56] R. Cramer, M. Franklin, B. Schoenmakers, and M. Yung. Multi-authority secret-ballot elections with linear work. In *EUROCRYPT'96*, volume 1070 of *Lecture Notes in Computer Science*, pages 72–83. Springer-Verlag, 1996.

[57] I. Damgård. A design principle for hash functions. In G. Brassard, editor, *Proc. CRYPTO 89*, pages 416–427. Springer-Verlag, 1990. Lecture Notes in Computer Science No. 435.

[58] D. Denning and G. Sacco. Timestamps in key distribution protocols. *Communications of the ACM*, 24(8):533–536, 1981.

[59] Y. Desmedt. Threshold cryptography. *European Transactions on Telecommunications*, 5(4):449–457, July 1994.

[60] Y. Desmedt and Y. Frankel. Shared generation of authenticators and signatures. In J. Feigenbaum, editor, *Proc. CRYPTO 91*, pages 457–469. Springer, 1992. Lecture Notes in Computer Science No. 576.

[61] W. Diffie and M. E. Hellman. Multiuser cryptographic techniques. In *Proc. AFIPS 1976 National Computer Conference*, pages 109–112, Montvale, N.J., 1976. AFIPS.

[62] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Trans. Inform. Theory*, IT-22:644–654, November 1976.

[63] W. Diffie, P. Van Oorschot, and M. Wiener. Authentication and authenticated key exchanges. *Designs, Codes, and Cryptography*, 2(2):107–125, June 1992.

[64] H. Dobbertin. Md5 is not collision-free. Manuscript, 1996.

[65] D. Dolev, C. Dwork, and M. Naor. Non-malleable cryptography. In *Proc. 23rd ACM Symp. on Theory of Computing*, pages 542–552. ACM, 1991.

[66] D. Dolev, S. Even, and R. M. Karp. On the security of ping-pong protocols. In R. L. Rivest, A. Sherman, and D. Chaum, editors, *Proc. CRYPTO 82*, pages 177–186, New York, 1983. Plenum Press.

[67] D. Dolev and A. C. Yao. On the security of public key protocols. In *Proc. 22nd IEEE Symp. on Foundations of Comp. Science*, pages 350–357, Nashville, 1981. IEEE.

[68] C. Dwork and M. Naor. An efficient existentially unforgeable signature scheme and its applications. In Yvo G. Desmedt, editor, *Proc. CRYPTO 94*, pages 234–246. Springer, 1994. Lecture Notes in Computer Science No. 839.

[69] P. Elias. The efficient construction of an unbiased random sequence. *Ann. Math. Statist.*, 43(3):865–870, 1972.

[70] S. Even and O. Goldreich. On the security of multi-party ping-pong protocols. In *Proc. 24th IEEE Symp. on Foundations of Comp. Science*, pages 34–39, Tucson, 1983. IEEE.

[71] S. Even, O. Goldreich, and A. Lempel. A randomized protocol for signing contracts. *Communications of the ACM*, 28:637–647, 1985.

[72] P. Feldman. A practical scheme for non-interactive verifiable secret sharing. In *Proc. 28th IEEE Symp. on Foundations of Comp. Science*, pages 427–438, Los Angeles, 1987. IEEE.

[73] A. Fiat and A. Shamir. How to prove yourself: practical solutions to identification and signature problems. In A. M. Odlyzko, editor, *Proc. CRYPTO 86*, pages 186–194. Springer, 1987. Lecture Notes in Computer Science No. 263.

[74] National Institute for Standards and Technology. A proposed federal information processing standard for digital signature standard (DSS). Technical Report FIPS PUB XX, National Institute for Standards and Technology, August 1991. DRAFT.

[75] Y. Frankel, P. Gemmell, and M. Yung. Witness-based cryptographic program checking and robust function sharing. In *28th ACM Symposium on Theory of Computing*, 1996.

[76] A. M. Frieze, J. Hastad, R. Kannan, J. C. Lagarias, and A. Shamir. Reconstructing truncated integer variables satisfying linear congruences. *SIAM J. Computing*, 17(2):262–280, April 1988.

[77] T. El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Trans. Inform. Theory*, 31:469–472, 1985.

[78] M. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, 1979.

[79] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Robust and efficient sharing of rsa functions. To appear in CRYPTO 1996.

[80] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Robust threshold dss signatures. In *EUROCRYPT'96*, volume 1070 of *Lecture Notes in Computer Science*, pages 354–371. Springer-Verlag, 1996.

[81] O. Goldreich. Two remarks concerning the Goldwasser-Micali-Rivest signature scheme. Technical Report MIT/LCS/TM-315, MIT Laboratory for Computer Science, September 1986.

[82] O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. *Journal of the ACM*, 33(4):792–807, October 1984.

[83] O. Goldreich and L. Levin. A hard-core predicate for all one-way functions. In *21st ACM Symposium on Theory of Computing*, 1989.

[84] O. Goldreich, S. Micali, and A. Wigderson. Proofs that yield nothing but their validity and a methodology of cryptographic protocol design. In *Proc. 27th IEEE Symp. on Foundations of Comp. Science*, pages 174–187, Toronto, 1986. IEEE.

[85] S. Goldwasser and J. Kilian. Almost all primes can be quickly certified. In *Proc. 18th ACM Symp. on Theory of Computing*, pages 316–329, Berkeley, 1986. ACM.

[86] S. Goldwasser and S. Micali. Probabilistic encryption and how to play mental poker keeping secret all partial information. In *Proc. 14th ACM Symp. on Theory of Computing*, pages 365–377, San Francisco, 1982. ACM.

[87] S. Goldwasser and S. Micali. Probabilistic encryption. *JCSS*, 28(2):270–299, April 1984.

[88] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof-systems. In *Proc. 17th ACM Symp. on Theory of Computing*, pages 291–304, Providence, 1985. ACM.

[89] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof-systems. *SIAM. J. Computing*, 18(1):186–208, February 1989.

[90] S. Goldwasser, S. Micali, and R. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Computing*, 17(2):281–308, April 1988.

[91] S. Goldwasser, S. Micali, and P. Tong. Why and how to establish a private code on a public network. In *Proc. 23rd IEEE Symp. on Foundations of Comp. Science*, pages 134–144, Chicago, 1982. IEEE.

[92] S. Goldwasser, S. Micali, and A. Yao. Strong signature schemes. In *Proc. 15th ACM Symp. on Theory of Computing*, pages 431–439, Boston, 1983. ACM.

[93] S. W. Golomb. *Shift Register Sequences*. Aegean Park Press, Laguna Hills, 1982. Revised edition.

[94] L. Harn. Group–oriented (t, n) threshold digital signature scheme and digital multisignature. *IEE Proc. Comput. Digit. Tech.*, 141(5):307–313, 1994.

[95] J. Hastad. Solving simultaneous modular equations of low degree. *SIAM J. Computing*, 17(2):336–341, April 1988.

[96] J. Håstad. Pseudo-random generators under uniform assumptions. In *22nd ACM Symposium on Theory of Computing*, 1990.

[97] J. Hastad, A.W. Schrift, and A. Shamir. The discrete logarithm modulo a composite hides $o(n)$ bits. *Journal of Computer and Systems Sciences*, 47:376–404, 1993.

[98] R. Impagliazzo, Leonid A. Levin, and M. Luby. Pseudo-random generation from one-way functions. In *Proc. 21st ACM Symp. on Theory of Computing*, pages 12–24, Seattle, 1989. ACM.

[99] R. Impagliazzo and M. Luby. One-way functions are essential for complexity based cryptography. In *Proc. 30th IEEE Symp. on Foundations of Comp. Science*, 1989.

[100] D. Johnson, A. Lee, W. Martin, S. Matyas, and J. Wilkins. Hybrid key distribution scheme giving key record recovery. *IBM Technical Disclosure Bulletin*, 37(2A):5–16, February 1994. See also U.S. Patent 5,142,578.

[101] D. Johnson and M. Matyas. Asymmetric encryption: Evolution and enhancements. *RSA Labs Cryptobytes*, 2(1), Spring 1996.

[102] B. Kaliski. A pseudo-random bit generator based on elliptic logarithms. In A.M. Odlyzko, editor, *Proc. CRYPTO 86*, pages 84–103. Springer-Verlag, 1987. Lecture Notes in Computer Science No. 263.

[103] B. Kaliski. *Elliptic Curves and Cryptography: A Pseudorandom Bit Generator and Other Tools*. PhD thesis, MIT EECS Dept., January 1988. Published as MIT LCS Technical Report MIT/LCS/TR-411 (Jan. 1988).

[104] B. Kaliski and M. Robshaw. Message authentication with MD5. *CryptoBytes*, 1(1):5–8, Spring 1995.

[105] R. Kannan, A. Lenstra, and L. Lovász. Polynomial factorization and non-randomness of bits of algebraic and some transcendental numbers. In *Proc. 16th ACM Symp. on Theory of Computing*, pages 191–200, Washington, D.C., 1984. ACM.

[106] J. Kilian. Founding cryptography on oblivious transfer. In *Proc. 20th ACM Symp. on Theory of Computing*, pages 20–31, Chicago, 1988. ACM.

[107] D. Knuth. *Seminumerical Algorithms*, volume 2 of *The Art of Computer Programming*. Addison-Wesley, 1969. Second edition, 1981.

[108] H. Krawczyk. Skeme: A versatile secure key exchange mechanism for internet. In *Proceedings of the Symposium on Network and Distributed System Security*, 1996.

[109] H. Krawczyk, M. Bellare, and R. Canetti, March 1996. Internet draft draft-ietf-ipsec-hmac-md5-txt.00.

[110] J. C. Lagarias and A. M. Odlyzko. Solving low-density subset sum problems. In *Proc. 24th IEEE Symp. on Foundations of Comp. Science*, pages 1–10, Tucson, 1983. IEEE.

[111] X. Lai and J. Massey. A proposal for a new block encryption standard. In I.B. Damgård, editor, *Proc. EUROCRYPT 90*, pages 389–404. Springer-Verlag, 1990. Lecture Notes in Computer Science No. 473.

[112] L. Lamport. Constructing digital signatures from a one-way function. Technical Report CSL-98, SRI International, October 1979.

[113] A. K. Lenstra and H. W. Lenstra, Jr. Algorithms in number theory. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science (Volume A: Algorithms and Complexity)*, chapter 12, pages 673–715. Elsevier and MIT Press, 1990.

[114] A. K. Lenstra, H. W. Lenstra, Jr., and L. Lovász. Factoring polynomials with rational coefficients. *Mathematische Ann.*, 261:513–534, 1982.

[115] A. K. Lenstra, H. W. Lenstra, Jr., M. S. Manasse, and J. M. Pollard. The number field sieve. In *Proc. 22nd ACM Symp. on Theory of Computing*, pages 564–572, Baltimore, Maryland, 1990. ACM.

[116] H. W. Lenstra, Jr. Factoring integers with elliptic curves. *Annals of Mathematics*, 126:649–673, 1987.

[117] R. Lipton. How to cheat at mental poker. In *Proc. AMS Short Course on Cryptography*, 1981.

[118] D. L. Long and A. Wigderson. The discrete logarithm problem hides $O(\log n)$ bits. *SIAM J. Computing*, 17(2):363–372, April 1988.

[119] M. Luby, S. Micali, and C. Rackoff. How to simultaneously exchange a secret bit by flipping a symmetrically biased coin. In *Proc. 24th IEEE Symp. on Foundations of Comp. Science*, pages 11–22, Tucson, 1983. IEEE.

[120] M. Luby and C. Rackoff. A study of password security. In Carl Pomerance, editor, *Proc. CRYPTO 87*, pages 392–397. Springer-Verlag, 1988. Lecture Notes in Computer Science No. 293.

[121] U. Maurer. Towards the equivalence of breaking the Diffie-Hellman protocol and computing discrete algorithms. In Yvo G. Desmedt, editor, *Proc. CRYPTO 94*, pages 271–281. Springer, 1994. Lecture Notes in Computer Science No. 839.

[122] R. J. McEliece. *A Public-Key System Based on Algebraic Coding Theory*, pages 114–116. Jet Propulsion Lab, 1978. DSN Progress Report 44.

[123] R. Merkle. Secrecy, authentication, and public key systems. Technical report, Stanford University, Jun 1979.

[124] R. Merkle. A certified digital signature. In G. Brassard, editor, *Proc. CRYPTO 89*, pages 218–238. Springer-Verlag, 1990. Lecture Notes in Computer Science No. 435.

[125] R. Merkle and M. Hellman. Hiding information and signatures in trapdoor knapsacks. *IEEE Trans. Inform. Theory*, IT-24:525–530, September 1978.

[126] M. Merritt. *Cryptographic Protocols*. PhD thesis, Georgia Institute of Technology, February 1983.

[127] G. Miller. Riemann's hypothesis and tests for primality. *JCSS*, 13(3):300–317, 1976.

[128] M. Naor and O. Reingold. Synthesizers and their application to the parallel construction of pseudo-random functions. In *Proc. 36th IEEE Symp. on Foundations of Comp. Science*. IEEE, 1995.

[129] M. Naor and M. Yung. Universal one-way hash functions and their cryptographic applications. In *Proc. 21st ACM Symp. on Theory of Computing*, pages 33–43, Seattle, 1989. ACM.

[130] M. Naor and M. Yung. Public-key cryptosystems provably secure against chosen ciphertext attack. In *Proc. of the Twenty-Second Annual ACM Symposium on Theory of Computing*, pages 427–437, Baltimore, Maryland, 1990. ACM.

[131] R. M. Needham and M. D. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, December 1978.

[132] R. M. Needham and M. D. Schroeder. Authentication revisited. *Operating Systems Review*, 21(1):7, January 1987.

[133] I. Niven and H. S. Zuckerman. *An Introduction to the Theory of Numbers*. Wiley, 1972.

[134] A. M. Odlyzko. Cryptanalytic attacks on the multiplicative knapsack scheme and on Shamir's fast signature scheme. *IEEE Trans. Inform. Theory*, IT-30:594–601, July 1984.

[135] A. M. Odlyzko. Discrete logarithms in finite fields and their cryptographic significance. In T. Beth, N. Cot, and I. Ingemarsson, editors, *Proc. EUROCRYPT 84*, pages 224–314, Paris, 1985. Springer. Lecture Notes in Computer Science No. 209.

[136] C. Park and K. Kurosawa. New elgamal type threshold signature scheme. *IEICE Trans. on Fund. Electr. Comm. and Comp. Sci.*, E79–A(1):86–93, 1996.

[137] T. Pedersen. Distributed provers with applications to undeniable signatures. In *EuroCrypt'91*, 1991.

[138] T.P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In J. Feigenbaum, editor, *Proc. CRYPTO 91*, pages 129–140. Springer, 1992. Lecture Notes in Computer Science No. 576.

[139] J. Plumstead. Inferring a sequence generated by a linear congruence. In *Proc. 23rd IEEE Symp. on Foundations of Comp. Science*, pages 153–159, Chicago, 1982. IEEE.

[140] S. C. Pohlig and M. E. Hellman. An improved algorithm for computing logarithms over $GF(p)$ and its cryptographic significance. *IEEE Trans. Inform. Theory*, IT-24:106–110, January 1978.

[141] D. Pointcheval and J. Stern. Security proofs for signatures. In *Proceedings of EUROCRYPT'96*, volume 1070 of *Lecture Notes in Computer Science*, pages 387–398. Springer-Verlag, 1996.

[142] J. M. Pollard. Theorems on factorization and primality testing. *Proc. Cambridge Philosophical Society*, 76:521–528, 1974.

[143] V. Pratt. Every prime has a succint certificate. *SIAM J. Comput.*, 4:214–220, 1975.

[144] B. Preneel and P. Van Oorschot. MDx-MAC and building fast MACs from hash functions. In Don Coppersmith, editor, *Proc. CRYPTO 94*, pages 1–14. Springer, 1995. Lecture Notes in Computer Science No. 963.

[145] B. Preneel and P.C. van Oorschot. On the security of two MAC algorithms. In *Proceedings of EUROCRYPT'96*, volume 1070 of *Lecture Notes in Computer Science*, pages 19–32. Springer-Verlag, 1996.

[146] M. Rabin. Digitalized signatures as intractable as factorization. Technical Report MIT/LCS/TR-212, MIT Laboratory for Computer Science, January 1979.

[147] M. Rabin. Probabilistic algorithms for testing primality. *J. Number Theory*, 12:128–138, 1980.

[148] M. Rabin. How to exchange secrets by oblivious transfer. Technical Report TR-81, Harvard Aiken Computation Laboratory, 1981.

[149] T. Rabin and M. Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority. In *21st ACM Symposium on Theory of Computing*, pages 73–85, 1989.

[150] R. Rivest. The MD5 message-digest algorithm. Internet Request for Comments, April 1992. RFC 1321.

[151] R. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[152] R. L. Rivest and A. Shamir. How to expose an eavesdropper. *Communications of the ACM*, 27:393–395, April 1984.

[153] P. Rogaway. Bucket hashing and its application to fast message authentication. In D. Coppersmith, editor, *Proc. CRYPTO 95*. Springer-Verlag, 1995. Lecture Notes in Computer Science No. 963.

[154] J. Rompel. One-way functions are necessary and sufficient for secure signatures. In *Proc. 22nd ACM Symp. on Theory of Computing*, pages 387–394, Baltimore, Maryland, 1990. ACM.

[155] J. Rosser and L. Schoenfield. Approximate formulas for some functions of prime numbers. *Illinois J. Math.*, 6:64–94, 1962.

[156] RSA Data Security, Inc. *PKCS #1: RSA Encryption Standard*, June 1991. Version 1.4.

[157] RSA Data Security, Inc. *PKCS #7: Cryptographic Message Syntax Standard*, June 1991. Version 1.4.

[158] K. Sako and J. Kilian. Receipt-free mix-type voting schemes. a practical implementation of a voting booth. In *EUROCRYPT'95*, volume 921 of *Lecture Notes in Computer Science*, pages 393–403. Springer-Verlag, 1995.

[159] M. Santha and U. V. Vazirani. Generating quasi-random sequences from slightly-random sources. In *Proc. 25th IEEE Symp. on Foundations of Comp. Science*, pages 434–440, Singer Island, 1984. IEEE.

[160] Alredo De Santis, Yvo Desmedt, Yair Frankel, and Moti Yung. How to share a function securely. In *Proc. 26th ACM Symp. on Theory of Computing*, pages 522–533, Santa Fe, 1994. IEEE.

[161] C. P. Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4:161–174, 1991.

[162] R. J. Schoof. Elliptic curves over finite fields and the computation of square roots mod p. *Math. Comp.*, 44:483–494, 1985.

[163] R. Schroeppel and A. Shamir. A $TS^2 = O(2^n)$ time/space tradeoff for certain NP-complete problems. In *Proc. 20th IEEE Symp. on Foundations of Comp. Science*, pages 328–336, San Juan, Puerto Rico, 1979. IEEE.

[164] A. Shamir. How to share a secret. *Communications of the ACM*, 22:612–613, November 1979.

[165] A. Shamir. On the cryptocomplexity of knapsack schemes. In *Proc. 11th ACM Symp. on Theory of Computing*, pages 118–129, Atlanta, 1979. ACM.

[166] A. Shamir. On the generation of cryptographically strong pseudo-random sequences. In *Proc. ICALP*, pages 544–550. Springer, 1981.

[167] A. Shamir. A polynomial-time algorithm for breaking the basic Merkle-Hellman cryptosystem. In *Proc. 23rd IEEE Symp. on Foundations of Comp. Science*, pages 145–152, Chicago, 1982. IEEE.

[168] A. Shamir, R. L. Rivest, and L. M. Adleman. Mental poker. In D. Klarner, editor, *The Mathematical Gardner*, pages 37–43. Wadsworth, Belmont, California, 1981.

[169] C. E. Shannon. A mathematical theory of communication. *Bell Sys. Tech. J.*, 27:623–656, 1948.

[170] C. E. Shannon. Communication theory of secrecy systems. *Bell Sys. Tech. J.*, 28:657–715, 1949.

[171] V. Shoup and A. Rubin. Session key distribution for smart cards. In U. Maurer, editor, *Proc. CRYPTO 96*. Springer-Verlag, 1996. Lecture Notes in Computer Science No. 1070.

[172] R.D. Silverman. The multiple polynomial quadratic sieve. *Mathematics of Computation*, 48:329–339, 1987.

[173] R. Solovay and V. Strassen. A fast Monte-Carlo test for primality. *SIAM J. Computing*, 6:84–85, 1977.

[174] William Stallings. *Network and Internetwork Security Principles and Practice.* Prentice Hall, 1995.

[175] J.G. Steiner, B.C. Neuman, and J.I. Schiller. Kerberos: an authentication service for open network systems. In *Usenix Conference Proceedings*, pages 191–202, Dallas, Texas, February 1988.

[176] J. Touch. Performance analysis of MD5. In *Proceedings SIGCOMM*, volume 25, pages 77–86, October 1995. Also at `ftp://ftp.isi.edu/pub/hpcc-papers/touch/sigcomm95.ps.Z`.

[177] G. Tsudik. Message authentication with one-way hash functions. *ACM SIGCOMM, Computer Communication Review*, 22(5):29–38, October 1992.

[178] P. van Oorschot and M. Wiener. Parallel collision search with applications to hash functions and discrete logarithms. In *Proceedings of the 2nd ACM Conf. Computer and Communications Security*, November 1994.

[179] U. Vazirani and V. Vazirani. RSA bits are $732 + \epsilon$ secure. In D. Chaum, editor, *Proc. CRYPTO 83*, pages 369–375, New York, 1984. Plenum Press.

[180] U. V. Vazirani. Towards a strong communication complexity theory, or generating quasi-random sequences from two communicating slightly-random sources. In *Proc. 17th ACM Symp. on Theory of Computing*, pages 366–378, Providence, 1985. ACM.

[181] U. V. Vazirani and V. V. Vazirani. Trapdoor pseudo-random number generators, with applications to protocol design. In *Proc. 24th IEEE Symp. on Foundations of Comp. Science*, pages 23–30, Tucson, 1983. IEEE.

[182] J. von Neumann. Various techniques for use in connection with random digits. In *von Neumann's Collected Works*, volume 5, pages 768–770. Pergamon, 1963.

[183] M. Wegman and L. Carter. New hash functions and their use in authentication and set equality. *J. of Computer and System Sciences*, 22:265–279, 1981.

[184] A. C. Yao. Theory and application of trapdoor functions. In *Proc. 23rd IEEE Symp. on Foundations of Comp. Science*, pages 80–91, Chicago, 1982. IEEE.

[185] A.C. Yao. Protocols for secure computations. In *Proc. 23rd IEEE Symp. on Foundations of Comp. Science*, pages 160–164, Chicago, 1982. IEEE.

[186] A.C. Yao. How to generate and exchange secrets. In *Proc. 27th IEEE Symp. on Foundations of Comp. Science*, pages 162–167, Toronto, 1986. IEEE.

CHAPTER A

Some complexity theory background

As of today, we do not even know how to prove a linear lower bound on the time required to solve an NP-complete problem. Thus, in our development of a theory of cryptography in the presence of a computationally bounded adversary we must resort to making assumptions about the existence of hard problems. In fact, an important current research topic in cryptography (on which much progress has been made in recent years) is to find the minimal assumptions required to prove the existence of "secure" cryptosystems.

Our assumptions should enable us to quickly generate instances of problems which are hard to solve for anyone other than the person who generated the instance. For example, it should be easy for the sender of a message to generate a ciphertext which is hard to decrypt for any adversary (naturally, in this example, it should also be easy for the intended recipient of the message to decrypt the ciphertext). To formally describe our assumptions (the existence of one way functions and trapdoor function) we first need to recall some complexity theory definitions.

A.1 Complexity Classes and Standard Definitions

A.1.1 Complexity Class P

A language L is in P if and only if there exists a Turing machine $M(x)$ and a polynomial function $Q(y)$ such that on input string x

1. $x \in L$ iff M accepts x (denoted by $M(x)$).
2. M terminates after at most $Q(|x|)$ steps.

The class of languages P is classically considered to be those languages which are 'easily computable'. We will use this term to refer to these languages and the term 'efficient algorithm' to refer to a polynomial time Turing machine.

A.1.2 Complexity Class NP

A language L is in NP if and only if there exists a Turing machine $M(x, y)$ and polynomials p and l such that on input string x

1. $x \in L \Rightarrow \exists y$ with $|y| \leq l(|x|)$ such that $M(x, y)$ accepts and M terminates after at most $p(|x|)$ steps.
2. $x \notin L \Rightarrow \forall y$ with $|y| \leq l(|x|)$, $M(x, y)$ rejects.

Note that this is equivalent to the (perhaps more familiar) definition of $L \in$ NP if there exists a non-deterministic polynomial time Turing machine M which accepts x if and only if $x \in L$. The string y above corresponds to the *guess* of the non-determinstic Turing machine.

A.1.3 Complexity Class BPP

A language L is in BPP if and only if there exists a Turing machine $M(x, y)$ and polynomials p and l such that on input string x

1. $x \in L \Rightarrow \Pr_{|y|<l(|x|)}[M(x, y)\text{accepts}] \geq \frac{2}{3}$.
2. $x \notin L \Rightarrow \Pr_{|y|<l(|x|)}[M(x, y)\text{accepts}] \leq \frac{1}{3}$.
3. $M(x, y)$ always terminates after at most $p(|x|)$ steps.

As an exercise, you may try to show that if the constants $\frac{2}{3}$ and $\frac{1}{3}$ are replaced by $\frac{1}{2} + \frac{1}{p(|x|)}$ and $\frac{1}{2} - \frac{1}{p(|x|)}$ where p is any fixed polynomial then the class BPP remains the same.
Hint: Simply run the machine $M(x, y)$ on "many" y's and accept if and only if the majority of the runs accept. The magnitude of "many" depends on the polynomial p.

We know that P $\subseteq$ NP and P $\subseteq$ BPP. We do not know if these containments are strict although it is often conjectured to be the case. An example of a language known to be in BPP but not known to be in P is the language of all prime integers (that is, primality testing). It is not known whether BPP is a subset of NP.

A.2 Probabilistic Algorithms

The class BPP could be alternatively defined using probabilistic Turing machines (probabilistic algorithms). A *probabilistic polynomial time Turing machine* M is a Turing machine which can flip coins as an additional primitive step, and on input string x runs for at most a polynomial in $|x|$ steps. We could have defined BPP by saying that a language L is in BPP if there exists a probabilistic polynomial time Turing machine $M(x)$ such that when $x \in L$, the probability (over the coin tosses of the machine) that $M(x)$ accepts is greater than $\frac{2}{3}$ and when $x \notin L$ the probability (over the coin tosses of the machine) that $M(x)$ rejects is greater than $\frac{2}{3}$. The string y in the previous definition corresponds to the sequence of coin flips made by the machine M on input x.

From now on we will consider probabilistic polynomial time Turing machines as "efficient algorithms" (extending the term previously used for deterministic polynomial time Turing machines). We also call the class of languages in BPP "easily computable". Note the difference between a non-deterministic Turing machine and a probabilistic Turing machine. A non-deterministic machine is not something we could implement in practice (as there may be only one good guess y which will make us accept). A probabilistic machine is something we could implement in practice by flipping coins to yield the string y (assuming of course that there is a source of coin flips in nature). Some notation is useful when talking about probabilistic Turing machines.

A.2.1 Notation For Probabilistic Turing Machines

Let M denote a probabilistic Turing machine (PTM). $M(x)$ will denote a probability space of the outcome of M during its run on x. The statement $z \in M(x)$ indicates that z was output by M when running on input x. $\Pr[M(x) = z]$ is the probability of z being the output of M on input x (where the probability is taken over the possible internal coin tosses made by M during its execution). $M(x, y)$ will denote the outcome of M on input x when internal coin tosses are y.

A.2.2 Different Types of Probabilistic Algorithms

Monte Carlo algorithms and Las Vegas algorithms are two different types of probabilistic algorithms. The difference between these two types is that a Monte Carlo algorithm always terminates within a polynomial number of steps but its output is only correct with high probability whereas a Las Vegas algorithm terminates within an expected polynomial number of steps and its output is always correct. Formally, we define these algorithms as follows.

Definition A.2.1 A Monte Carlo algorithm is a probabilistic algorithm M for which there exists a polynomial P such that for all x, M terminates within $P(|x|)$ steps on input x. Further,

$$\Pr[\mathrm{M}(x) \text{ is correct }] > \frac{2}{3}$$

(where the probability is taken over the coin tosses of M).

A Las Vegas algorithm is a probabilistic algorithm M for which there exists a polynomial p such that for all x, $\mathrm{E}(\text{running time}) = \sum_{t=1}^{\infty} t \cdot \Pr[\mathrm{M}(x) \text{ takes exactly } t \text{ steps}] < p(|x|)$. Further, the output of $\mathrm{M}(x)$ is always correct.

All Las Vegas algorithms can be converted to Monte Carlo algorithms but it is unknown whether all Monte Carlo algorithms can be converted to Las Vegas algorithms. Some examples of Monte Carlo algorithms are primality tests such as Solovay Strassen (see [173]) or Miller-Rabin (see [147]) and testing the equivalence of multivariate polynomials and some examples of Las Vegas algorithms are computing square roots modulo a prime p, computing square roots modulo a composite n (if the factors of n are known) and primality tests based on elliptic curves (see [4] or [85]).

A.2.3 Non-Uniform Polynomial Time

An important concept is that of polynomial time algorithms which can behave differently for inputs of different size, and may even be polynomial in the size of the input (rather than constant as in the traditional definition of a Turing machine).

Definition A.2.2 A *non-uniform algorithm* A is an infinite sequence of algorithms $\{M_i\}$ (one for each input size i) such that on input x, $M_{|x|}(x)$ is run. We say that $A(x)$ accepts if and only if $M_{|x|}(x)$ accepts. We say that A is a *polynomial time* non-uniform algorithm if there exist polynomials P and Q such that $M_{|x|}(x)$ terminates within $P(|x|)$ steps and the size of the description of M_i (according to some standard encoding of all algorithms) is bounded by $Q(|i|)$.

Definition A.2.3 We say that a language L is in P/poly if $\exists$ a polynomial time non-uniform algorithm $A = \{M_i\}$ such that $x \in L$ iff $M_{|x|}(x)$ accepts.

There are several relationships known about $\mathrm{P/poly}$. It is clear that $P \subset \mathrm{P/poly}$ and it has been shown by Adleman that $\mathrm{BPP} \subset \mathrm{P/poly}$.

We will use the term 'efficient non-uniform algorithm' to refer to a non-uniform polynomial time algorithm and the term 'efficiently non-uniform computable' to refer to languages in the class $\mathrm{P/poly}$.

A.3 Adversaries

We will model the computational power of the adversary in two ways. The first is the (uniform) adversary (we will usually drop the "uniform" when referring to it). A *uniform adversary* is any polynomial time probabilistic algorithm. A *non-uniform adversary* is any non-uniform polynomial time algorithm. Thus, the adversary can use different algorithms for different sized inputs. Clearly, the non-uniform adversary is stronger than the uniform one. Thus to prove that "something" is "secure" even in presence of a non-uniform adversary is a better result than only proving it is secure in presence of a uniform adversary.

A.3.1 Assumptions To Be Made

The weakest assumption that must be made for cryptography in the presence of a uniform adversary is that P $\neq$ NP. Namely, $\exists L \in$ NP such that $L \notin$ P. Unfortunately, this is not enough as we assumed that our adversaries can use probabilistic polynomial time algorithms. So we further assume that BPP $\neq$ NP. Is that sufficient? Well, we actually need that it would be hard for an adversary to crack our systems most of the time. It is not sufficient that our adversary can not crack the system once in a while. Assuming that BPP $\neq$ NP only means that there exists a language in $L \in$ NP such that every uniform adversary makes (with high probability) the wrong decision about infinitely many inputs x when deciding whether $x \in L$. These wrong decisions, although infinite in number, may occur very infrequently (such as once for each input size).

We thus need yet a stronger assumption which will guarantee the following. There exists a language $L \in$ NP such that for every sufficiently large input size n, every uniform adversary makes (with high probability) the wrong decision on many inputs x of length n when deciding whether x is in L. Moreover, we want it to be possible, for every input size n, to generate input x of length n such that with high probability every uniform adversary will make the wrong decision on x.

The assumption that will guarantee the above is the existence of (uniform) one way functions. The assumption that would guarantee the above in the presence of non-uniform adversary is the existence non-uniform one way functions. For definitions, properties, and possible examples of one-way functions see Chapter 2.

A.4 Some Inequalities From Probability Theory

Proposition A.4.1 *[Markov's Inequality] If Z is a random variable that takes only non-negative values, then for any value $a > 0$,* $\Pr[Z \geq a] \leq \frac{E[Z]}{a}$.

Proposition A.4.2 *[Weak Law of Large Numbers] Let $z_1, \ldots, z_n$ be independent 0-1 random variables (Bernoulli random variables) with mean μ. Then* $\Pr[|\frac{\sum_{i=1}^{n} z_i}{n} - \mu| < \epsilon] > 1 - \delta$ *provided that* $n > \frac{1}{4\epsilon^2\delta}$.

CHAPTER B

Some number theory background

Many important constructions of cryptographic primitives are based on problems from number theory which seem to be computationally intractable. The most well-known of these problems is that of factoring composite integers. In order to work with these problems, we need to develop some basic materiel on number theory and number theoretic algorithms. Accordingly, we provide here a mini-course on this subject. The materiel here will be used later when we discuss candidate example one-way and trapdoor functions.

There are many sources for information on number theory in the literatures. For example try Angluin's notes [7] and Chapter 33 of Cormen, Leiserson and Rivest [55].

B.1 Groups: Basics

A *group* is a set G together with some operation, which we denote $*$. It takes pairs of elements to another element, namely $a * b$ is the result of $*$ applied to a, b. A group has the following properties:

(1) If $a, b \in G$ so is $a * b$

(2) The operation is associative: $(a * b) * c = a * (b * c)$

(3) There is an *identity element* I such that $I * a = a * I = a$ for all $a \in G$

(4) Every $a \in G$ has an inverse, denoted a^{-1}, such that $a * a^{-1} = a^{-1} * a = I$.

We will encounter this kind of structure a lot. First recall Z, N and $\mathbf{R}$. Now, for example:

- Integers under addition: $I = 0$; $a^{-1} = -a$.
- Real numbers under multiplication: $I = 1$; $a^{-1} = 1/a$.
- What about N under addition? Not a group!
- What about Z under multiplication? Not a group!

Notation: a^m is a multiplied by itself m times. Etc. Namely, notation is what you expect. What is a^{-m}? It is $(a^{-1})^m$. Note that it "works" like it should.

These groups are all infinite. We are usually interested in finite ones. In such a case:

Def: We call $|G|$ the order of G.

Fact B.1.1 Let $m = |G|$. Then $a^m = I$ for any $a \in G$.

We will use this later.

$a \equiv b \pmod{n}$ means that if we divide a by n then the remainder is b. (In C, this is $a\%n = b$).

An important set is the set of integers modulo an integer n. This is $Z_n = \{0, \ldots, n-1\}$. We will see it is a group under addition modulo n. Another related important set is $Z_n^* = \{m : 1 \leq m \leq n \text{ and } \gcd(m,n) = 1\}$, the set of integers less than n which are relatively prime to n. We will see this is a group under multiplication modulo n. We let $\phi(n) = |Z_n^*|$. This is the Euler totient function.

A subset $S \subseteq G$ is called a sub-group if it is a group in its own right, under the operation making G a group. In particular if $x, y \in S$ so is xy and x^{-1}, and $1 \in S$.

Fact B.1.2 Suppose S is a subgroup of G. Then $|S|$ divides $|G|$.

B.2 Arithmatic of numbers: +, *, GCD

Complexity of algorithms operating of a number a is measured in terms of the size (length) of a, which is $|a| \approx \lg(a)$. How long do basic operations take? In terms of the number of bits k in the number:

- Addition is linear time. Ie. two k-bit numbers can be added in $O(k)$ time.
- Multiplication of a and b takes $O(|a| \cdot |b|)$ bit operations. Namely it is an $O(k^2)$ algorithm.
- Division of a by b (integer division: we get back the quotient and remainder) takes time $O((1 + |q|)|b|)$ where q is the quotient obtained. Thus this too is a quadratic time algorithm.

Euclid's algorithm can be used to compute GCDs in polynomial time. The way it works is to repeatedly use the identity $\gcd(a,b) = \gcd(b, a \bmod b)$. For examples, see page 10 of [7].

What is the running time? Each division stage takes quadratic time and we have k stages, which would say it is a $O(k^3)$ algorithm. But see Problem 33-2, page 850, of [55]. This yields the following:

Theorem B.2.1 *Euclid's algorithm can be implemented to use only $O(|a| \cdot |b|)$ bit operations to compute $\gcd(a,b)$. That is, for k-bit numbers we get a $O(k^2)$ algorithm.*

Fact B.2.2 $\gcd(a,b) = 1$ if and only if there exist integers u, v such that $1 = au + bv$.

The Extended Euclid Algorithm is given a, b and it returns not only $d = \gcd(a,b)$ but integers u, v such that $d = au + bv$. It is very similar to Euclid's algorithm. We can keep track of some extra information at each step. See page 11 of [7].

B.3 Modular operations and groups

B.3.1 Simple operations

Now we go to modular operations, which are the main ones we are interested in

- Addition is now the following: Given a, b, n with $a, b \in Z_n$ compute $a + b \bmod n$. This is still linear time. Ie. two k-bit numbers can be added in $O(k)$ time. Why? You can't go much over N. If you do, just subtract n. That too is linear time.
- Taking $a \bmod n$ means divide a by n and take remainder. Thus, it takes quadratic time.
- Multiplication of a and b modulo n: First multiply them, which takes $O(|a| \cdot |b|)$ bit operations. Then divide by n and take the remainder. We saw latter too was quadratic. So the whole thing is quadratic.

Z_n is a group under addition modulo N. This means you can add two elements and get back an element of the set, and also subtraction is possible. Under addition, things work like you expect.

We now move to Z_n^*. We are interested in the multiplication operation here. We want to see that it is a group, in the sense that you can multiply and divide. We already saw how to multiply.

Theorem B.3.1 *There is a $O(k^2)$ algorithm which given a, n with $a \in Z_n^*$ outputs $b \in Z_n^*$ satisfying $ab \equiv 1 \pmod{n}$, where $k = |n|$.*

See page 12 of [7]. The algorithms uses the extended Euclid. We know that $1 = \gcd(a, n)$. Hence it can find integers u, v such that $1 = au + nv$. Take this modulo n and we get $au \equiv 1 \pmod{n}$. So can set $b = u \bmod n$. Why is this an element of Z_n^*? Claim that $\gcd(u, n) = 1$. Why? By Fact B.2.2, which says that $1 = au + nv$ means $\gcd(u, n) = 1$.

The b found in the theorem can be shown to be unique. Hence:

Notation: The b found in the theorem is denoted a^{-1}.

B.3.2 The main groups: Z_n and Z_n^*

Theorem B.3.2 *For any positive integer n, Z_n^* forms a group under multiplication modulo n.*

This means that $a, b \in Z_n^*$ implies $ab \bmod n$ is in Z_n^*, something one can verify without too much difficulty. It also means we can multiply and divide. We have an identity (namely 1) and a cancellation law.

Notation: We typically stop writing mod$n$ everywere real quick. It should just be understood.

The way to think about Z_n^* is like the real numbers. You can manipulate things like you are used to. The following is a corollary of Fact B.1.1.

Theorem B.3.3 *For any $a \in Z_n^*$ it is the case that $a^{\phi(n)} = 1$.*

Corollary B.3.4 (Fermat's little theorem) *If p is prime then $a^{p-1} \equiv 1 \pmod{p}$ for any $a \in \{1, \ldots, p-1\}$.*

Why? Because $\phi(p) = p - 1$.

B.3.3 Exponentiation

This is the most basic operation for public key cryptography. The operation is just that given a, n, m where $a \in Z_n$ and m is an integer, computes $a^m \bmod n$.

Example B.3.5 Compute $2^{21} \bmod 22$. Naive way: use 21 multiplications. What's the problem with this? It is an *exponential* time algorithm. Because we want time poly(k) where $k = |n|$. So we do it by repeated squaring:

$$\begin{aligned} 2^1 &\equiv 2 \\ 2^2 &\equiv 4 \\ 2^4 &\equiv 16 \\ 2^8 &\equiv 14 \\ 2^{16} &\equiv 20 \end{aligned}$$

Now $2^{21} = 2^{16+4+1} = 2^{16} * 2^4 * 2^1 = 20 * 16 * 2 \equiv 10 \bmod 21$.

This is the repeated squaring algorithm for exponentiation. It takes cubic time.

Theorem B.3.6 *There is an algorithm which given a, n, m with $a, m \in Z_n$ outputs $a^m \bmod n$ in time $O(k^3)$ where $k = |n|$. More precisely, the algorithm uses at most $2k$ modular multiplications of k-bit numbers.*

Algorithm looks at binary expansion of m. In example above we have $21 = 10101$. What we did is collect all the powers of two corresponding to the ones and multiply them.

4	3	2	1	0
a^{16}	a^8	a^4	a^2	a^1
1	0	1	0	1

Exponentiate(a, n, m)

Let $b_{k-1}...b_1b_0$ be the binary representation of m
Let $x_0 = a$
Let $y = 1$
For $i = 0, \ldots, k-1$ do
    If $b_i = 1$ let $y = y * x_i \bmod n$
    Let $x_{i+1} = x_i^2 \bmod n$
Output y

B.4 Chinese remainders

Let $m = m_1 m_2$. Suppose $y \in Z_m$. Consider the numbers

$$\begin{array}{lcll} a_1 & = & y \bmod m_1 & \in Z_{m_1} \\ a_2 & = & y \bmod m_2 & \in Z_{m_2} \end{array}$$

The chinese remainder theorem considers the question of recombining a_1, a_2 back to get y. It says there is a *unique* way to do this under some conditions, and under these conditions says how.

Example B.4.1 $m = 6 = 3 * 2$

$$\begin{array}{l} 0 \rightarrow (0,0) \\ 1 \rightarrow (1,1) \\ 2 \rightarrow (2,0) \\ 3 \rightarrow (0,1) \\ 4 \rightarrow (1,0) \\ 5 \rightarrow (2,1) \end{array}$$

Example B.4.2 $m = 4 = 2 * 2$

$$\begin{array}{l} 0 \rightarrow (0,0) \\ 1 \rightarrow (1,1) \\ 2 \rightarrow (0,0) \\ 3 \rightarrow (1,1) \end{array}$$

The difference here is that in the first example, the association is unique, in the second it is not. It turns out uniqueness happens when the m_1, m_2 are relatively prime. Here is a simple form of the theorem.

Theorem B.4.3 *[Chinese Remainder Theorem] Let $m_1, m_2, \ldots, m_k$ be pairwise relatively prime integers. That is, $gcd(m_i, m_j) = 1$ for $1 \leq i < j \leq k$. Let $a_i \in \mathbf{Z}_{m_i}$ for $1 \leq i \leq k$ and set $m = m_1 m_2 \cdots m_k$. Then there exists a unique $y \in \mathbf{Z}_m$ such that $y \equiv a_i \bmod m_i$ for $i = 1 \ldots k$. Furthermore there is an $O(k^2)$ time algorithm to compute y given a_1, a_2, m_1, m_2, where $k = \max(|m_1|, |m_2|)$.*

Proof: For each i, let $n_i = \left(\frac{m}{m_i}\right) \in \mathbf{Z}$. By hypothesis, $\gcd(m_i, n_i) = 1$ and hence $\exists b_i \in \mathbf{Z}_{m_i}$ such that $n_i b_i \equiv 1 \bmod m_i$. Let $c_i = b_i n_i$. Then $c_i = \begin{cases} 1 \bmod m_i \\ 0 \bmod m_j \text{ for } j \neq i \end{cases}$.

Set $y = \sum_{i=1}^{k} c_i a_i \bmod m$. Then $y \equiv a_i \bmod m_i$ for each i.

Further, if $y' \equiv a_i \bmod m_i$ for each i then $y' \equiv y \bmod m_i$ for each i and since the m_i's are pairwise relatively prime it follows that $y \equiv y' \bmod m$, proving uniqueness. ∎

Remark B.4.4 The integers c_i appearing in the above proof will be referred to as the Chinese Remainder Theorem coefficients. Note that the proof yields a polynomial time algorithm for finding y because the elements $b_i \in \mathbf{Z}_{m_i}$ can be determined by using the Euclidean algorithm and the only other operations involved are division, multiplication, and addition.

A more general form of the Chinese Remainder Theorem is the following result.

Theorem B.4.5 *Let $a_i \in \mathbf{Z}_{m_i}$ for $1 \leq i \leq k$. A necessary and sufficient condition that the system of congruences $x \equiv a_i \bmod m_i$ for $1 \leq i \leq k$ be solvable is that $gcd(m_i, m_j)|(a_i - a_j)$ for $1 \leq i < j \leq k$. If a solution exists then it is unique modulo $lcm(m_1, m_2, \ldots, m_k)$.*

Solution Of The Quadratic Congruence $a \equiv x^2 \bmod n$ When $a \in \mathbf{Z}_n$.

First observe that for p an odd prime and $a \in \mathbf{Z}_p^{*2}$ so that $a \equiv x^2 \bmod p$ for some $x \in \mathbf{Z}_p^*$ there are exactly two solutions to $a \equiv x^2 \bmod p$ because x and $-x$ are two distinct solutions modulo p and if $y^2 \equiv a \equiv x^2 \bmod p$ then $p|[(x-y)(x+y)] \Longrightarrow p|(x-y)$ or $p|(x+y)$ so that $y \equiv \pm x \bmod p$. (Note that $x \not\equiv -x \bmod p$ for otherwise, $2x \equiv 0 \bmod p \Rightarrow p|x$ as p is odd.) Thus, for $a \in \mathbf{Z}_p^*$, $a \equiv x^2 \bmod p$ has either 0 or 2 solutions.

Next consider the congruence $a \equiv x^2 \bmod p_1 p_2$ where p_1 and p_2 are distinct odd primes. This has a solution if and only if both $a \equiv x^2 \bmod p_1$ and $a \equiv x^2 \bmod p_2$ have solutions. Note that for each pair (x_1, x_2) such that $a \equiv x_1^2 \bmod p_1$ and $a \equiv x_2^2 \bmod p_2$ we can combine x_1 and x_2 by using the Chinese Remainder Theorem to produce a solution y to $a \equiv x^2 \bmod p_1 p_2$ such that $y \equiv x_1 \bmod p_1$ and $y \equiv x_2 \bmod p_2$. Hence, the congruence $a \equiv x^2 \bmod p_1 p_2$ has either 0 or 4 solutions.

More generally, if $p_1, p_2, \ldots, p_k$ are distinct odd primes then the congruence $a \equiv x^2 \bmod p_1^{\alpha_1} p_2^{\alpha_2} \ldots p_k^{\alpha_k}$ has either 0 or 2^k solutions. Again, these solutions can be found by applying the Chinese Remainder Theorem to solutions of $a \equiv x^2 \bmod p_i^{\alpha_i}$. Furthermore, for a prime p a solution to the congruence $a \equiv x^2 \bmod p^k$ can be found by first finding a solution x_0 to $a \equiv x^2 \bmod p$ by using algorithm A of Lemma 2.3.22 and viewing it as an approximation of the desired square root. Then the approximation is improved by the iteration $x_j \equiv \frac{1}{2}(x_{j-1} + \frac{a}{x_{j-1}}) \bmod p^{2^j}$ for $j \geq 1$.

Claim B.4.6 For each integer $j \geq 0$, $a \equiv x_j^2 \bmod p^{2^j}$. **Proof:** The claim is certainly true for $j = 0$. Suppose that for $j > 0$, $a \equiv x_j^2 \bmod p^{2^j}$.

Then $x_j - a x_j^{-1} \equiv 0 \bmod p^{2^j} \Rightarrow (x_j - a x_j^{-1})^2 \equiv 0 \bmod p^{2^{j+1}}$.

Expanding and adding $4a$ to both sides gives $x_j^2 + 2a + a^2 x_j^{-2} \equiv 4a \bmod p^{2^{j+1}}$ and therefore, $\left(\frac{1}{2}(x_j + \frac{a}{x_j})\right)^2 \equiv a \bmod p^{2^{j+1}}$ or $x_{j+1}^2 \equiv a \bmod p^{2^{j+1}}$.

Hence, the claim follows by induction. ∎

From the claim, it follows that after $\lceil \log k \rceil$ iterations we will obtain a solution to $a \equiv x^2 \bmod p^k$.

B.5 Primitive elements and Z_p^*

B.5.1 Definitions

Let G be a group. Let $a \in G$. Look at the powers of a, namely $a^0, a^1, a^2, \ldots,$. We let

$$\langle a \rangle \quad = \quad \{ a^i \ : \ i \geq 0 \} .$$

Let $m = |G|$ be the order of G. We know that a^0 is the identity, call it 1, and $a^m = 1$ also. So the sequence repeats after m steps, ie. $a^{m+1} = a$, etc. But it could repeat before too. Let's look at an example.

Example B.5.1 $Z_9^* = \{1, 2, 4, 5, 7, 8\}$. Size $\phi(9) = 6$. Then:

$$\langle 1 \rangle \quad = \quad \{1\}$$

$$\langle 2 \rangle \quad = \quad \{1, 2, 4, 8, 7, 5\}$$

$$\langle 4 \rangle \quad = \quad \{1, 4, 7\}$$

$$\langle 5 \rangle \quad = \quad \{1, 5, 7, 8, 4, 2\}$$

What we see is that sometimes we get everything, sometimes we don't. It might wrap around early.

Fact B.5.2 $\langle a \rangle$ is a subgroup of G, called the subgroup generated by a.

Let $t = |\langle a \rangle|$. Then we know that t divides m. And we know that in fact $\langle a \rangle = \{a^0, a^1, \ldots, a^{t-1}\}$. That is, these are the distinct elements. All others are repeats.

Definition B.5.3 The order of an element a is the least positive integer t such that $a^t = 1$. That is, $\mathrm{ord}(a) = |\langle a \rangle|$.

Computation in the indices can be done modulo t. That is, $a^i = a^{i \bmod t}$. This is because $a^t = a^0 = 1$.

What's the inverse of a^i? Think what it "should" be: a^{-i}. Does this make sense? Well, think of it as $(a^{-1})^i$. This is correct. On the other hand, what is it as member of the subgroup? It must be a^j for some j. Well $j = t - i$. In particular, inverses are pretty easy to compute if you are given the index.

Similarly, like for real numbers, multiplication in the base corresponds to addition in the exponent. Namely $a^{i+j} = a^i \cdot a^j$. Etc.

Definition B.5.4 An element $g \in G$ is said to be a primitive element, or *generator*, of G if the powers of g generate G. That is, $\langle g \rangle = G$ is the *whole* group G. A group G is called *cyclic* if it has a primitive element.

Note this means that for any $y \in G$ there is a *unique* $i \in \{0, \ldots, m-1\}$ such that $g^i = y$, where $m = |G|$.

Notation: This unique i is denoted $\log_g(y)$ and called the *discrete logarithm of x to base g*.

Consider the following problem. Given g, y, figure out $\log_g(y)$. How could we do it? One way is to go through all $i = 0, \ldots, m-1$ and for each i compute g^i and check whether $g^i = y$. But this process takes exponential time.

It turns out that computing discrete logarithms is hard for many groups. Namely, there is no known polynomial time algorithm. In particular it is true for Z_p^* where p is a prime.

B.5.2 The group Z_p^*

Fact B.5.5 [7, Section 9] The group Z_p^* is cyclic.

Remark B.5.6 What is the order of Z_p^*? It is $\phi(p)$, the number of positive integers below p which are relatively prime to p. Since p is prime this is $p-1$. Note the order is *not* prime! In particular, it is even (for $p \geq 3$).

A one-way function: Let p be prime and let $g \in Z_p^*$ be a generator. Then the function $f_{p,g} \colon Z_p \to Z_p^*$ defined by

$$x \mapsto g^x$$

is conjectured to be one-way as long as some technical conditions hold on p. That is, there is no efficient algorithm to invert it, for large enough values of the parameters. See Chapter 2.

Homomorphic properties: A useful property of the function $f_{p,g}$ is that $g^{a+b} = g^a \cdot g^b$.

Now, how can we use this function? Well, first we have to set it up. This requires two things. First that we can find primes; second that we can find generators.

B.5.3 Finding generators

We begin with the second. We have to look inside Z_p^* and find a generator. How? Even if we have a candidate, how do we test it? The condition is that $\langle g \rangle = G$ which would take $|G|$ steps to check.

In fact, finding a generator given p is in general a hard problem. In fact even checking that g is a generator given p, g is a hard problem. But what we can exploit is that $p = 2q + 1$ with q prime. Note that the order of the group Z_p^* is $p - 1 = 2q$.

Fact B.5.7 Say $p = 2q + 1$ is prime where q is prime. Then $g \in Z_p^*$ is a generator of Z_p^* iff $g^q \neq 1$ and $g^2 \neq 1$.

In other words, testing whether g is a generator is easy given q. Now, given $p = 2q + 1$, how do we find a generator?

Fact B.5.8 If g is a generator and i is not divisible by q or 2 then g^i is a generator.

Proof: $g^{iq} = g^{q+(i-1)q} = g^q \cdot (g^{2q})^{(i-1)/2} = g^q \cdot 1 = g^q$ which is not 1 because g is not a generator. Similarly let $i = r + jq$ and we have $g^{2i} = g^{2r} \cdot g^{2jq} = g^{2r}$. But $2r < 2q$ since $r < q$ so $g^{2r} \neq 1$. ∎

So how many generators are there in Z_p^*? All things of form g^i with i not divisible by 2 or q and $i = 1, \ldots, 2q$. Namely all i in Z_{2q}^*. So there are $\phi(2q) = q - 1$ of them.

So how do we find a generator? Pick $g \in Z_p^*$ at random, and check that $g^q \neq 1$ and $g^2 \neq 1$. If it fails, try again, up to some number of times. What's the probability of failure? In one try it is $(q + 1)/2q$ so in l tries it is

$$\left(\frac{q+1}{2q}\right)^l$$

which is rougly 2^{-l} because q is very large.

B.6 Quadratic residues

An element $x \in Z_N^*$ is a square, or quadratic residue, if it has a square root, namely there is a $y \in Z_N^*$ such that $y^2 \equiv x \bmod N$. If not, it is a non-square or non-quadratic-residue. Note a number may have lots of square roots.

It is easy to compute square roots modulo a prime. It is also easy modulo a composite whose prime factorization you know, via Chinese remainders. (In both cases, you can compute all the roots.) But it is hard modulo a composite of unknown factorization. In fact computing square roots is equivalent to factoring.

Also, it is hard to decide quadratic residuosity modulo a composite.

Fact B.6.1 If N is the product of two primes, every square $w \in Z_N^*$ has exactly four square roots, $x, -x$ and $y, -y$ for some $x, y \in Z_N^*$. If you have two square roots x, y such that $x \neq \pm y$, then you can easily factor N.

The first fact is basic number theory. The second is seen like this. Say $x > y$ are the roots. Consider $x^2 - y^2 = (x - y)(x + y) \equiv 0 \bmod N$. Let $a = x - y$ and $b = x + y \bmod N$. So N divides ab. So p divides ab. Since p is prime, this means either p divides a or p divides b. Since $1 \leq a, b < N$ this means either $\gcd(a, N) = p$ or $\gcd(b, N) = p$. We can compute the gcds and check whether we get a divisor of N.

B.7 Jacobi Symbol

We previously defined the Legendre symbol to indicate the quadratic character of $a \in \mathbf{Z}_p^*$ where p is a prime. Specifically, for a prime p and $a \in \mathbf{Z}_p$

$$\mathbf{J}_p(a) = \begin{cases} 1 & \text{if } a \text{ is a square in } \mathbf{Z}_p^* \\ 0 & \text{if } a = 0 \\ -1 & \text{if } a \text{ is not a square in } \mathbf{Z}_p^* \end{cases}$$

For composite integers, this definition is extended, as follows, giving the *Jacobi Symbol*. Let $n = \prod_{i=1}^k {p_i}^{\alpha_i}$ be the prime factorization of n. For $a \in \mathbf{Z}_n$ define

$$\mathbf{J}_n(a) = \prod_{i=1}^k \mathbf{J}_{p_i}(a)^{\alpha_i}.$$

However, the Jacobi Symbol does not generalize the Legendre Symbol in the respect of indicating the quadratic character of $a \in \mathbf{Z}_n^*$ when n is composite. For example, $\mathbf{J}_9(2) = \mathbf{J}_3(2)\mathbf{J}_3(2) = 1$, although the equation $x^2 \equiv 2 \bmod 9$ has no solution.

The Jacobi Symbol also satisfies identities similar to those satisfied by the Legendre Symbol. We list these here. For proofs of these refer to [133].

1. If $a \equiv b \bmod n$ then $\mathbf{J}_n(a) = \mathbf{J}_n(b)$.
2. $\mathbf{J}_n(1) = 1$.
3. $\mathbf{J}_n(-1) = (-1)^{\frac{n-1}{2}}$.
4. $\mathbf{J}_n(ab) = \mathbf{J}_n(a)\mathbf{J}_n(b)$.
5. $\mathbf{J}_n(2) = (-1)^{\frac{n^2-1}{8}}$.
6. If m and n are relatively prime odd integers then $\mathbf{J}_n(m) = (-1)^{\frac{n-1}{2}\frac{m-1}{2}}\mathbf{J}_m(n)$.

Using these identities, the Jacobi Symbol $\mathbf{J}_n(a)$ where $a \in \mathbf{Z}_n$ can be calculated in polynomial time even without knowing the factorization of n. Recall that to calculate the Legendre Symbol in polynomial time we can call upon Euler's Theorem; namely, for $a \in \mathbf{Z}_p^*$, where p is prime, we have $\mathbf{J}_p(a) \equiv a^{\frac{p-1}{2}} \bmod p$. However, for a composite integer n it is not necessarily true that $\mathbf{J}_n(a) \equiv a^{\frac{n-1}{2}} \bmod n$ for $a \in \mathbf{Z}_n^*$. In fact, this statement is true for at most half of the elements in $\mathbf{Z}_n^*$. From this result, we can derive a Monte Carlo primality test as we shall see later.

B.8 RSA

Here we have a composite modulus $N = pq$ product of two distinct primes p and q of roughly equal length. Let $k = |N|$; this is about 1024, say. It is generally believed that such a number is hard to factor.

Recall that $\phi(N) = |Z_N^*|$ is the Euler Phi function. Note that $\phi(N) = (p-1)(q-1)$. (To be relatively prime to N, a number must be divisible neither by p nor by q. Eliminating multiples of either yields this. Note we use here that $p \neq q$.)

Now let e be such that $\gcd(e, \phi(N)) = 1$. That is, $e \in Z_{\phi(N)}^*$. The RSA function is defined by

$$\begin{array}{rccc} f\colon & Z_N^* & \to & Z_N^* \\ & x & \mapsto & x^e \bmod N \ . \end{array}$$

We know that $Z_{\phi(N)}^*$ is a group. So e has an inverse $d \in Z_{\phi(N)}^*$. Since d is an inverse of e it satisfies

$$ed \equiv 1 \pmod{\phi(N)}$$

Now let $x \in Z_N^*$ be arbitrary and look at the following computation:

$$(x^e)^d \bmod N = x^{ed \bmod \phi(N)} \bmod N = x^1 \bmod N = x \ .$$

In other words, the function $y \mapsto y^d$ is an inverse of f. That is, $f^{-1}(y) = y^d \bmod N$.

Can we find d? Easy: computing inverses can be done in quadratic time, as we already say, using the extended GCD algorithm! But note a crucial thing. We are working modulo $\phi(N)$. So finding d this way requires that we know $\phi(N)$. But the latter involves knowing p, q.

It seems to be the case that given only N and e it is hard to find d. Certainly we agreed it is hard to find p, q; but even more, it seems hard to find d. This yields the conjecture that RSA defines a trapdoor one-way permutation. Namely given N, e defining f, $x \mapsto f(x)$ is easy; $y \mapsto f^{-1}(y)$ is hard; but f^{-1} is easy given p, q (or d). Note that this trapdoorness is a property the discrete logarithm problem did not have.

Computation of f is called encryption, and computation of f^{-1} is called decryption.

Both encryption and decryption are exponentiations, which a priori are cubic in k time operations. However, one often chooses e to be small, so encryption is faster. In hardware, RSA is about 1000 times slower than DES; in software it is about 100 times slower, this with small encryption exponent.

Formally, RSA defines a *family of trapdoor permutations.* The family is indexed by a security parameter k which is the size of the modulus. The RSA generator is an algorithm G which on input 1^k picks two distinct, random $(k/2)$-bit primes p, q, multiplies them to produce $N = pq$, and also computes e, d. It outputs N, e as the description of f and N, d as the description of f^{-1}. See Chapter 2.

RSA provides the ability to do public key cryptography.

B.9 Primality Testing

For many cryptographic purposes, We need to find primes. There is no known polynomial time algorithm to test primality of a given integer n. What we use are probabilistic, polynomial time (PPT) algorithms.

We will first show that the problem of deciding whether an integer is prime is in NP. Then we will discuss the Solovay-Strassen and Miller-Rabin probabilistic primality tests which efficiently find proofs of compositeness. Finally, we will give a primality test due to Goldwasser and Kilian which uses elliptic curves and which efficiently finds a proof of primality.

B.9.1 PRIMES $\in$ NP

We will first present two algorithms for testing primality, both of which are inefficient because they require factoring as a subroutine. However, either algorithm can be used to show that the problem of deciding whether an integer is prime is in NP. In fact, the second algorithm that is presented further demonstrates that deciding primality is in UP$\cap$coUP. Here UP denotes the class of languages L accepted by a polynomial time nondeterministic Turing machine having a unique accepting path for each $x \in L$.

Definition B.9.1 Let PRIMES $= \{p : p \text{ is a prime integer}\}$.

B.9.2 Pratt's Primality Test

Pratt's primality testing algorithm is based on the following result.

Proposition B.9.2 *For an integer $n > 1$, the following statements are equivalent.*

1. $|\mathbf{Z}_n^*| = n - 1$.
2. *The integer n is prime.*
3. *There is an element $g \in \mathbf{Z}_n^*$ such that $g^{n-1} \equiv 1 \bmod n$ and for every prime divisor q of $n-1$, $g^{\frac{n-1}{q}} \not\equiv 1 \bmod n$.*

Pratt's algorithm runs as follows on input a prime p and outputs a proof (or certificate) that p is indeed prime.

1. Find an element $g \in \mathbf{Z}_p^*$ whose order is $p-1$.
2. Determine the prime factorization $\prod_{i=1}^{k} q_i^{\alpha_i}$ of $p-1$.
3. Prove that p is prime by proving that g is a generator of $\mathbf{Z}_p^*$. Specifically, check that $g^{p-1} \equiv 1 \bmod p$ and for each prime q_i check that $g^{\frac{p-1}{q_i}} \not\equiv 1 \bmod p$.
4. Recursively show that q_i is prime for $1 \leq i \leq k$.

Note that if p is a prime, then $\mathbf{Z}_p^*$ has $\varphi(p-1) = \Omega(\frac{p}{\log\log p})$ generators (see [155]). Thus, in order to find a generator g by simply choosing elements of $\mathbf{Z}_p^*$ at random, we expect to have to choose $O(\log\log p)$ candidates for g. If we find a generator g of $\mathbf{Z}_p^*$ and if we can factor $p-1$ and recursively prove that the prime factors of $p-1$ are indeed primes then we have obtained a proof of the primality of p. Unfortunately, it is not known how to efficiently factor $p-1$ for general p. Pratt's primality testing algorithm does demonstrate, however, that PRIMES $\in$ NP because both the generator g in step 1 and the required factorization in step 2 can be guessed. Moreover, the fact that the factorization is correct can be verified in polynomial time and the primality of each q_i can be verified recursively by the algorithm. Note also, as Pratt showed in [143] by a simple inductive argument, that the total number of primes involved is $O(\log p)$. Thus, verifying a Pratt certificate requires $O(\log^2 p)$ modular multiplications with moduli at most p.

B.9.3 Probabilistic Primality Tests

B.9.4 Solovay-Strassen Primality Test

We can derive a Monte Carlo primality test. This algorithm, which we state next, is due to Solovay and Strassen (see [173]).

The Solovay-Strassen primality test runs as follows on input an odd integer n and an integer k, indicating the desired reliability.

1. Test if $n = b^e$ for integers $b, e > 1$; if so, output composite and terminate.
2. Randomly choose $a_1, a_2, \ldots, a_k \in \{1, 2, \ldots, n-1\}$.
3. If $\gcd(a_i, n) \neq 1$ for any $1 \leq i \leq k$ then output composite and terminate.
4. Calculate $\alpha_i = a_i^{\frac{n-1}{2}} \bmod n$ and $\beta_i = \mathbf{J}_n(a_i)$.
5. If for any $1 \leq i \leq k$, $\alpha_i \neq \beta_i \bmod n$ then output composite. If for all $1 \leq i \leq k$, $\alpha_i = \beta_i \bmod n$ then output probably prime.

Since the calculations involved in the Solovay-Strassen primality test are all polynomial time computable (verify that this statement is indeed true for step 1), it is clear that the algorithm runs in time polynomial in $\log n$ and k. The following result guarantees that if n is composite then in step 5 of the algorithm, $\Pr[\alpha_i = \beta_i \bmod n] \leq \frac{1}{2}$ and thus, $\Pr[\alpha_i = \beta_i \bmod n \text{ for } 1 \leq i \leq k] \leq \left(\frac{1}{2}\right)^k$.

Proposition B.9.3 *Let n be an odd composite integer which is not a perfect square and let $G = \{a \in \mathbf{Z}_n^* \text{ such that } \mathbf{J}_n(a) \equiv a^{\frac{n-1}{2}} \bmod n\}$. Then $|G| \leq \frac{1}{2}|\mathbf{Z}_n^*|$.*

Proof: Since G is a subgroup of $\mathbf{Z}_n^*$ it suffices to show that $G \neq \mathbf{Z}_n^*$.

Since n is composite and not a perfect square, it has a nontrivial factorization $n = rp^\alpha$ where p is prime, α is odd, and $\gcd(r, p) = 1$.

Suppose that $a^{\frac{n-1}{2}} \equiv \mathbf{J}_n(a) \bmod n$ for all $a \in \mathbf{Z}_n^*$. Then

$$a^{\frac{n-1}{2}} \equiv \pm 1 \bmod n \text{ for all } a \in \mathbf{Z}_n^*. \tag{B.1}$$

We first show that in fact $a^{\frac{n-1}{2}} \equiv 1 \bmod n$ for all such a. If not, then there is an $a \in \mathbf{Z}_n^*$ with $a^{\frac{n-1}{2}} \equiv -1 \bmod n$. By the Chinese Remainder Theorem there is a unique element $b \in \mathbf{Z}_n$ such that $b \equiv 1 \bmod r$ and $b \equiv a \bmod p^\alpha$. Then $b \in \mathbf{Z}_n^*$ and $b^{\frac{n-1}{2}} \equiv 1 \bmod r$ and $b^{\frac{n-1}{2}} \equiv -1 \bmod p^\alpha$ contradicting equation $(B.1)$. Therefore, $\mathbf{J}_n(a) = 1$ for all $a \in \mathbf{Z}_n^*$.

However, by the Chinese Remainder Theorem, there is a unique element $a \in \mathbf{Z}_{rp}$ such that $a \equiv 1 \bmod r$ and $a \equiv z \bmod p$ where z is one of the $\frac{p-1}{2}$ quadratic nonresidues modulo p. Then $a \in \mathbf{Z}_n^*$ and thus, $\mathbf{J}_n(a) = \mathbf{J}_r(1)\mathbf{J}_p(z)^\alpha = -1$ because α is odd. This is a contradiction. ∎

Note that if we reach step 5 of the Solovay-Strassen algorithm then n is not a perfect square and each $a_i \in \mathbf{Z}_n^*$ because the algorithm checks for perfect powers in step 1 and computes $\gcd(a_i, n)$ in step 3. Thus, the hypotheses of Proposition B.9.3 are satisfied and for each $1 \leq i \leq k$, $\Pr[\alpha_i = \beta_i \bmod n] \leq \frac{1}{2}$.

Remark The assertion in Proposition B.9.3 is in fact true even if n is a perfect square. The proof of the more general statement is very similar to the proof of Proposition B.9.3. For details refer to [7].

Finally, it follows from Proposition B.9.3 that the Solovay-Strassen algorithm runs correctly with high probability. Specifically,

$$\Pr[\text{Solovay-Strassen outputs probably prime} \mid n \text{ is composite}] \leq (\frac{1}{2})^k$$

and

$$\Pr[\text{Solovay-Strassen outputs probably prime} \mid n \text{ is prime}] = 1.$$

B.9.5 Miller-Rabin Primality Test

Fermat's Little Theorem states that for a prime p and $a \in \mathbf{Z}_p^*$, $a^{p-1} \equiv 1 \bmod p$. This suggests that perhaps a possible way to test if n is prime might be to check, for instance, if $2^{n-1} \equiv 1 \bmod n$. Unfortunately, there are composite integers n (called base-2 pseudoprimes) for which $2^{n-1} \equiv 1 \bmod n$. For example, $2^{340} \equiv 1 \bmod 341$ and yet $341 = 11 \cdot 31$. In fact, replacing 2 in the above exponentiation by a random $a \in \mathbf{Z}_n^*$ would not help for some values of n because there are composite integers n (called Carmichael numbers) for which $a^{n-1} \equiv 1 \bmod n$ for all $a \in \mathbf{Z}_n^*$. 561, 1105, and 1729 are the first three Carmichael numbers.

The Miller-Rabin primality test overcomes the problems of the simple suggestions just mentioned by choosing several random $a \in \mathbf{Z}_n^*$ for which $a^{n-1} \bmod n$ will be calculated by repeated squaring. While computing each modular exponentiation, it checks whether some power of a is a nontrivial square root of 1 modulo n (that is, a root of 1 not congruent to ± 1 modulo n). If so, the algorithm has determined n to be composite. The quality of this test relies on Proposition B.9.5 which Rabin proved in [147]. For a simpler proof which only yields $|\{b : \mathrm{W}_n(b) \text{ holds }\}| \geq \frac{1}{2}(n-1)$ (but is nevertheless sufficient for the purposes of the Miller-Rabin algorithm) see Chapter 33, pages 842-843 of [55].

Definition B.9.4 Let n be an odd positive integer. Denote the following condition on an integer b by $\mathrm{W}_n(b)$:

1. $1 \leq b < n$ and
2. (i) $b^{n-1} \not\equiv 1 \bmod n$ or
   (ii) there is an integer i such that $2^i \mid (n-1)$ and $b^{(n-1)/2^i} \not\equiv \pm 1 \bmod n$ but $\left(b^{(n-1)/2^i}\right)^2 \equiv 1 \bmod n$.

An integer b for which $\mathrm{W}_n(b)$ holds will be called a witness to the compositeness of n.

Remark Rabin originally defined the condition $\mathrm{W}_n(b)$ to hold if $1 \leq b < n$ and either $b^{n-1} \not\equiv 1 \bmod n$ or for some integer i such that $2^i \mid (n-1)$, $1 < \gcd(b^{(n-1)/2^i} - 1, n) < n$. In [147] Rabin proves that the two definitions for $\mathrm{W}_n(b)$ are equivalent. This condition was in fact first considered by Miller (see [127]), who used it to give a nonprobabilistic test for primality assuming the correctness of the extended Riemann hypothesis. Rabin's results, however, do not require any unproven assumptions.

Proposition B.9.5 *If n is an odd composite integer then $|\{b : W_n(b)$ holds$\}| \geq \frac{3}{4}(n-1)$.*

The Miller-Rabin algorithm runs as follows on input an odd integer n and an integer k, indicating the desired reliability.

1. Randomly choose $b_1, b_2, \ldots, b_k \in \{1, 2, \ldots, n-1\}$.
2. Let $n - 1 = 2^l m$ where m is odd.
3. For $1 \leq i \leq k$ compute $b_i^m \bmod n$ by repeated squaring.
4. Compute $b_i^{2^j m} \bmod n$ for $j = 1, 2, \ldots, l$. If for some j, $b_i^{2^{j-1} m} \not\equiv \pm 1 \bmod n$ but $b_i^{2^j m} \equiv 1 \bmod n$ then $\mathrm{W}_n(b_i)$ holds.
5. If $b_i^{n-1} \not\equiv 1 \bmod n$ then $\mathrm{W}_n(b_i)$ holds.
6. If for any $1 \leq i \leq k$, $\mathrm{W}_n(b_i)$ holds then output composite. If for all $1 \leq i \leq k$, $\mathrm{W}_n(b_i)$ does not hold then output probably prime.

Proposition B.9.5 shows that the Miller-Rabin algorithm runs correctly with high probability. Specifically,

$$\Pr[\text{Miller-Rabin outputs probably prime} \mid n \text{ is composite}] \leq (\frac{1}{4})^k$$

and

$$\Pr[\text{Miller-Rabin outputs probably prime} \mid n \text{ is prime}] = 1.$$

Furthermore, Miller-Rabin runs in time polynomial in $\log n$ and k as all the computations involved can be performed in polynomial time.

B.9.6 Polynomial Time Proofs Of Primality

Each of the two algorithms discussed in the previous section suffers from the deficiency that whenever the algorithm indicates that the input n is prime, then it is prime with high probability, but no certainty is provided. (In other words, the algorithms are Monte Carlo algorithms for testing primality.) However, when either algorithm outputs that the input n is composite then it has determined that n is indeed composite. Thus, the Solovay-Strassen and Miller-Rabin algorithms can be viewed as compositeness provers. In this section we will discuss a primality test which yields in expected polynomial time a short (verifiable in deterministic polynomial time) proof that a prime input is indeed prime. Therefore, for a general integral input we can run such a primality prover in parallel with a compositeness prover and one of the two will eventually terminate either yielding a proof that the input is prime or a proof that the input is composite. This will provide us with a Las Vegas algorithm for determining whether an integer is prime or composite.

B.9.7 An Algorithm Which Works For Some Primes

Suppose that we could find a prime divisor q of $p-1$ such that $q > \sqrt{p}$. Then the following algorithm can be used to prove the primality of p.

1. Determine a prime divisor q of $p-1$ for which $q > \sqrt{p}$.
2. Randomly choose $a \in \mathbf{Z}_p^* - \{1\}$.
3. If $1 < \gcd(a-1, p) < p$ then output that p is composite.
4. Check that $a^q \equiv 1 \bmod p$.
5. Recursively prove that q is prime.

The correctness of this algorithm follows from the next result.

Claim B.9.6 If $q > \sqrt{p}$ is a prime and for some $a \in \mathbf{Z}_p^*$ $\gcd(a-1, p) = 1$ and $a^q \equiv 1 \bmod p$ then p is a prime.

Proof: Suppose p is not prime. Then there is a prime $d \leq \sqrt{p}$ such that $d \mid p$ and therefore, by the hypothesis, $a \not\equiv 1 \bmod d$ and $a^q \equiv 1 \bmod d$. Thus, in $\mathbf{Z}_d^*$, $\mathrm{ord}(a) \mid q$. But q is prime and a does not have order 1. Hence, $q = \mathrm{ord}(a) \leq |\mathbf{Z}_d^*| = d - 1 < \sqrt{p}$ and this contradicts the assumption that $q > \sqrt{p}$. ∎

Note that if p is prime then in step 4, the condition $a^q \equiv 1 \bmod p$ will be verified with probability at least $\frac{q-1}{p-2} > \frac{1}{\sqrt{p}}$ (since $q > \sqrt{p}$). However, in order for the algorithm to succeed, there must exist a prime divisor q of $p-1$ such that $q > \sqrt{p}$, and this must occur at every level of the recursion. Namely, there must be a sequence of primes $q = q_0, q_1, \ldots, q_k$, where q_k is small enough to identify as a known prime, such that $q_i \mid (q_{i-1} - 1)$ and $q_i > \sqrt{q_{i-1}}$ for $i = 1, \ldots, k$ and this is very unlikely.

This obstacle can be overcome by working with elliptic curves modulo primes p instead of $\mathbf{Z}_p^*$. This will allow us to randomly generate for any prime modulus, elliptic groups of varying orders. In the following sections, we will exploit this additional freedom in a manner similar to Lenstra's elliptic curve factoring algorithm.

B.9.8 Goldwasser-Kilian Primality Test

The Goldwasser-Kilian primality test is based on properties of elliptic curves. The idea of the algorithm is similar to the primality test presented in Section B.9.7, except that we work with elliptic curves $E_{a,b}(\mathbf{Z}_p)$ instead of $\mathbf{Z}_p^*$. By varying a and b we will be able to find an elliptic curve which exhibits certain desired properties.

The Goldwasser-Kilian algorithm runs as follows on input a prime integer $p \neq 2, 3$ of length l and outputs a proof that p is prime.

1. Randomly choose $a, b \in \mathbf{Z}_p$, rejecting choices for which $\gcd(4a^3 + 27b^2, p) \neq 1$.
2. Compute $|E_{a,b}(\mathbf{Z}_p)|$ using the polynomial time algorithm due to Schoof (see [162]).
3. Use a probabilistic pseudo-primality test (such as Solovay-Strassen or Miller-Rabin) to determine if $|E_{a,b}(\mathbf{Z}_p)|$ is of the form cq where $1 < c \leq O(\log^2 p)$ and q is a probable prime. If $|E_{a,b}(\mathbf{Z}_p)|$ is not of this form then repeat from step 1.
4. Select a point $M = (x, y)$ on $E_{a,b}(\mathbf{Z}_p)$ by choosing $x \in \mathbf{Z}_p$ at random and taking y to be the square root of $x^3 + ax + b$, if one exists. If $x^3 + ax + b$ is a quadratic nonresidue modulo p then repeat the selection process.
5. Compute $q \cdot M$.
   (i) If $q \cdot M = O$ output (a, b, q, M). Then, if $q > 2^{l(\frac{1}{\log\log l})}$, [5]), recursively prove that q is prime (specifically, repeat from step 1 with p replaced by q). Otherwise, use the deterministic test due to Adleman, Pomerance, and Rumely (see [5]) to show that q is prime and terminate.
   (ii) If $q \cdot M \neq O$ then repeat from step 4.

Remark The test mentioned in step 5i is currently the best deterministic algorithm for deciding whether an input is prime or composite. It terminates within $(\log n)^{O(\log\log\log n)}$ steps on input n.

B.9.9 Correctness Of The Goldwasser-Kilian Algorithm

Note first that as we saw in Section B.9.3, the probability of making a mistake at step 3 can be made exponentially small. The correctness of the Goldwasser-Kilian algorithm follows from Theorem B.9.7. This result is analogous to Claim B.9.6.

Theorem B.9.7 *Let $n > 1$ be an integer with $gcd(n, 6) = 1$. Let $E_{a,b}(\mathbf{Z}_n)$ be an elliptic curve modulo n and let $M \neq O$ be a point on $E_{a,b}(\mathbf{Z}_n)$. If there is a prime integer q such that $q > (n^{1/4} + 1)^2$ and $q \cdot M = O$ then n is prime.*

Proof: Suppose that n were composite. Then there is a prime divisor p of n such that $p < \sqrt{n}$.

Let $\mathrm{ord}_E(M)$ denote the order of the point M on the elliptic curve E. If $q \cdot M = O$ then $q \cdot M_p = O_p$. Thus, $\mathrm{ord}_{E_p}(M_p) \mid q$.

$$\begin{aligned} \text{However, } \mathrm{ord}_{E_p}(M_p) \leq |E_{a,b}(\mathbf{Z}_p)| &\leq p + 1 + 2\sqrt{p} \qquad \text{(by Hasse's Inequality)} \\ &< n^{1/2} + 1 + 2n^{1/4} \\ &< q \end{aligned}$$

and since q is prime, we have that $\mathrm{ord}_{E_p}(M_p) = 1$. Therefore, $M_p = O_p$ which implies that $M = O$, a contradiction. ∎

Theorem B.9.8 *By using the sequence of quadruples output by the Goldwasser-Kilian algorithm, we can verify in time $O(\log^4 p)$ that p is indeed prime.*

Proof: Let $p_0 = p$. The sequence of quadruples output by the algorithm will be of the form $(a_1, b_1, p_1, M_1), (a_2, b_2, p_2, M_2), \ldots,$ where $\gcd(4a_i^3 + 27b_i^2, p_{i-1}) \neq 1$, $M_i \neq O$ is a point on $E_{a_i,b_i}(\mathbf{Z}_{p_{i-1}})$, $p_i > p_{i-1}^{1/2} + 1 + 2p_{i-1}^{1/4}$, and $p_i \cdot M_i = O$ for $1 \leq i \leq k$. These facts can all be verified in $O(\log^3 p)$ time for each value of i. By Theorem B.9.7 it follows that p_i prime $\Rightarrow p_{i-1}$ prime for $1 \leq i \leq k$. Further, note that in step 3 of the algorithm, $c \geq 2$ and hence, $p_i \leq \frac{p_{i-1}+2\sqrt{p_{i-1}}}{2}$. Therefore, the size of k will be $O(\log p)$ giving a total of $O(\log^4 p)$ steps. Finally, p_k can be verified to be prime in $O(\log p)$ time due to its small size. ∎

B.9.10 Expected Running Time Of Goldwasser-Kilian

The algorithm due to Schoof computes $|E_{a,b}(\mathbf{Z}_p)|$ in $O(\log^9 p)$ time. Then to check that $|E_{a,b}(\mathbf{Z}_p)| = cq$ where $1 < c \leq O(\log^2 p)$ and q is prime requires a total of $O(\log^6 p)$ steps if we use Solovay-Strassen or Miller-Rabin with enough iterations to make the probability of making a mistake exponentially small (the algorithm may have to be run for each possible value of c and each run of the algorithm requires $O(\log^4 p)$ steps).

Next, selecting the point $M = (x, y)$ requires choosing an expected number of at most $\frac{2p}{|E_{a,b}(\mathbf{Z}_p)|-1} \approx 2$ values for x before finding one for which $x^3 + ax + b$ is a quadratic residue modulo p. Note that the computation of square roots modulo a prime p (to find y) can be done in $O(\log^4 p)$ expected time. Since $|E_{a,b}(\mathbf{Z}_p)| = cq$ where q is prime, $E_{a,b}(\mathbf{Z}_p)$ is isomorphic to $\mathbf{Z}_{c_1 q} \times \mathbf{Z}_{c_2}$ where $c = c_1 c_2$ and $c_2 | c_1$. Therefore, $E_{a,b}(\mathbf{Z}_p)$ has at least $q - 1$ points of order q and hence with probability at least $\frac{q-1}{cq} \approx \frac{1}{c}$, the point M selected in step 4 will have order q. Thus, the expected number of points that must be examined before finding a point M of order q will be $c = O(\log^2 p)$. Further, the computation of $q \cdot M$ requires $O(\log p)$ additions, using repeated doubling and so can be done in $O(\log^3 p)$ time. Therefore, dealing with steps 4 and 5 requires $O(\log^5 p)$ expected time.

As remarked previously, the recursion depth is $O(\log p)$. Therefore, the only remaining consideration is to determine how often an elliptic curve $E_{a,b}(\mathbf{Z}_p)$ has to be selected before $|E_{a,b}(\mathbf{Z}_p)| = cq$ where $c = O(\log^2 p)$ and q is prime. By the result of Lenstra concerning the distribution of $|E_{a,b}(\mathbf{Z}_p)|$ in $(p + 1 - \sqrt{p}, p + 1 + \sqrt{p})$ (see [113]) this is $O(\frac{\sqrt{p} \log p}{|S|-2})$ where S is the set of integers in $(p + 1 - \sqrt{p}, p + 1 + \sqrt{p})$ of the desired form cq. Note that $|S| \geq \pi(\frac{p+1+\sqrt{p}}{2}) - \pi(\frac{p+1-\sqrt{p}}{2})$ because S contains those integers in $(p + 1 - \sqrt{p}, p + 1 + \sqrt{p})$ which are twice a prime. Therefore, if one assumes that the asymptotic distribution of primes holds in small intervals, then the expected number of elliptic curves that must be considered is $O(\log^2 p)$. However, there is only evidence to assume the following conjecture concerning the number of primes in small intervals.

Conjecture B.9.9 *There is a positive constant s such that for all $x \in \mathbf{R}_{\geq 2}$, the number of primes between x and $x + \sqrt{2x}$ is $\Omega\left(\frac{\sqrt{x}}{\log^s x}\right)$.*

Under this assumption, the Goldwasser-Kilian algorithm proves the primality of p in $O((\log p)^{11+s})$ expected time.

B.9.11 Expected Running Time On Nearly All Primes

Although the analysis presented in Section B.9.10 relies on the unproven result stated in Conjecture B.9.9, a theorem due to Heath-Brown concerning the density of primes in small intervals can be used to show that the fraction of primes of length l for which the Goldwasser-Kilian algorithm runs in expected time polynomial in l is at least $1 - O(2^{-l^{\frac{1}{\log \log l}}})$. The Heath-Brown result is the following.

Theorem B.9.10 *Let $\#_p[a, b]$ denote the number of primes x satisfying $a \leq x \leq b$.*
Let $i(a, b) = \begin{cases} 1 \text{ if } \#_p[a, b] \leq \frac{b-a}{2\lceil \log a \rceil} \\ 0 \text{ otherwise} \end{cases}$. Then there exists a positive constant α such that
$\sum_{x \leq a \leq 2x} i(a, a + \sqrt{a}) \leq x^{\frac{5}{6}} \log^{\alpha} x.$

Using this theorem, Goldwasser and Kilian were able to prove in [85] that their algorithm terminates in expected time $O(l^{12})$ on at least a $1 - O(2^{l^{\frac{1}{\log \log l}}})$ fraction of those primes of length l. In [4] Adleman and Huang showed, by a more careful analysis of the Goldwasser-Kilian algorithm, that in fact the fraction of primes of length l for which Goldwasser-Kilian may not terminate in expected polynomial time is strictly exponentially vanishing. Further, they proposed a new algorithm for proving primality based on hyperelliptic curves which they showed will terminate in exponential polynomial time on all prime inputs. Thus, the goal of obtaining a Las Vegas algorithm has been finally achieved.

B.10 Factoring Algorithms

In this lecture we discuss some general properties of elliptic curves and present Lenstra's elliptic curve factoring algorithm which uses elliptic curves over $\mathbf{Z}_n$ to factor integers.

Pollard's $p-1$ Method

We begin by introducing a predecessor of the elliptic curve factoring algorithm which uses ideas analogous to those used in the elliptic curve factoring algorithm. This algorithm, known as Pollard's $p-1$ method, appears in [142]. Let n be the composite number that we wish to split. Pollard's algorithm uses the idea that if we can find integers e and a such that $a^e \equiv 1 \bmod p$ and $a^e \not\equiv 1 \bmod q$ for some prime factors p and q of n then, since $p \mid (a^e - 1)$ and $q \not\mid (a^e - 1)$, $\gcd(a^e - 1, n)$ will be a nontrivial factor of n divisible by p but not by q.

The algorithm proceeds as follows on input n.

1. Choose an integer e that is a multiple of all integers less than some bound B. For example, e might be the least common multiple of all integers $\leq B$. To simplify this, we might even let $e = \prod_{i=1}^{\pi(B)} p_i^{\alpha_i}$ where $p_1, p_2, \ldots, p_{\pi(B)}$ are the primes $\leq B$ and α_i is chosen minimally so that $p_i^{\alpha_i} \geq \sqrt{n} > \min_{p \mid n}\{p-1\}$.

2. Choose a random integer a between 2 and $n-2$.

3. Compute $a^e \bmod n$ by repeated squaring.

4. Compute $d = \gcd(a^e - 1, n)$ by the Euclidean algorithm. If $1 < d < n$ output the nontrivial factor d. Otherwise, repeat from step 2 with a new choice for a.

To explain when this algorithm works, assume that the integer e is divisible by every integer $\leq B$ and that p is a prime divisor of n such that $p-1$ is the product of prime powers $\leq B$. Then $e = m(p-1)$ for some integer m and hence $a^e = \left(a^{(p-1)}\right)^m \equiv 1^m = 1 \bmod p$. Therefore, $p \mid \gcd(a^e - 1, n)$ and the only way that we could fail to obtain a nontrivial factor of n in step 4 is if $a^e \equiv 1 \bmod n$. In other words, we could only fail here if for every prime factor q of n the order of $a \bmod q$ divides e and this is unlikely.

Unfortunately, it is not true that for general n there is a prime divisor p of n for which $p-1$ is divisible by no prime power larger than B for a bound B of small size. If $p-1$ has a large prime power divisor for each prime divisor p of n, then Pollard's $p-1$ method will work only for a large choice of the bound B and so will be inefficient because the algorithm runs in essentially $O(B)$ time. For example, if n is the product of two different primes p and q where $|p| \approx |q|$ are primes and $p-1$ and $q-1$ are $O(\sqrt{n})$-smooth then the method will likely require a bound B of size $O(\sqrt{n})$.

Reiterating, the problem is that given input $n = \prod {p_i}^{\alpha_i}$ where the p_i's are the distinct prime factors of n, we are restricted by the possibility that none of the integers $p_i - 1$ are sufficiently smooth. However, we can ameliorate this restriction by working with the group of points defined over elliptic curves. For each prime p we will obtain a large collection of groups whose orders essentially vary "uniformly" over the interval $(p+1-\sqrt{p}, p+1+\sqrt{p})$. By varying the groups involved we can hope to always find one whose order is smooth. We will then show how to take advantage of such a collection of groups to obtain a factorization of n.

B.11 Elliptic Curves

Definition B.11.1 An elliptic curve over a field F is the set of points (x, y) with x, $y \in F$ satisfying the Weierstrass equation $y^2 = x^3 + ax + b$ where $a, b \in F$ and $4a^3 + 27b^2 \neq 0$ together with a special point O called the point at infinity. We shall denote this set of points by $E_{a,b}(F)$.

Remark The condition $4a^3 + 27b^2 \neq 0$ ensures that the curve is nonsingular. That is, when the field F is $\mathbf{R}$, the tangent at every point on the curve is uniquely defined.

Let P, Q be two points on an elliptic curve $E_{a,b}(F)$. We can define the negative of P and the sum $P + Q$ on the elliptic curve $E_{a,b}(F)$ according to the following rules.

1. If P is the point at infinity O then we define $-P$ to be O.
   Otherwise, if $P = (x, y)$ then $-P = (x, -y)$.

2. $O + P = P + O = P$.

3. Let $P, Q \neq O$ and suppose that $P = (x_1, y_1)$, $Q = (x_2, y_2)$.

   (i) If $P = -Q$ (that is, $x_1 = x_2$ and $y_1 = -y_2$) then we define $P + Q = O$

   (ii) Otherwise, let

$$\alpha = \frac{y_1 - y_2}{x_1 - x_2} \text{ if } P \neq Q \tag{B.2}$$

   or

$$\alpha = \frac{3x_1^2 + a}{y_1 + y_2} \text{ if } P = Q \tag{B.3}$$

   That is, if the field $F = \mathbf{R}$, α is the slope of the line defined by P and Q, if $P \neq Q$, or the slope of the tangent at P, if $P = Q$.
   Then $P + Q = R$ where $R = (x_3, y_3)$ with $x_3 = \alpha^2 - (x_1 + x_2)$ and $y_3 = \alpha(x_1 - x_3) - y_1$.

It can be shown that this definition of addition on the elliptic curve is associative and always defined, and thus it imposes an additive abelian group structure on the set $E_{a,b}(F)$ with O serving as the additive identity of the group. We will be interested in $F = \mathbf{Z}_p$ where $p \neq 2, 3$ is a prime. In this case, addition in $E_{a,b}(\mathbf{Z}_p)$ can be computed in time polynomial in $|p|$ as equations $(B.2)$ and $(B.3)$ involve only additions, subtractions, and divisions modulo p. Note that to compute $z^{-1} \bmod p$ where $z \in \mathbf{Z}_p^*$ we can use the extended Euclidean algorithm to compute an integer t such that $tz \equiv 1 \bmod p$ and then set $z^{-1} = t \bmod p$.

To illustrate negation and addition, consider the elliptic curve $y^2 = x^3 - x$ over $\mathbf{R}$ as shown in Figure B.1.

Figure B.1: Addition on the elliptic curve $y^2 = x^3 - x$.

The graph is symmetric about the x-axis so that the point P is on the curve if and only if $-P$ is on the curve. Also, if the line l through two points $P, Q \neq O$ on the curve $E(\mathbf{R})$ is not vertical then there is exactly one more point where this line intersects the curve. To see this, let $P = (x_1, y_1)$, $Q = (x_2, y_2)$ and let $y = \alpha x + \beta$ be the equation of the line through P and Q where $\alpha = \frac{y_1 - y_2}{x_1 - x_2}$ if $P \neq Q$ or $\alpha = \frac{3x_1^2 + a}{y_1 + y_2}$ if $P = Q$ and $\beta = y_1 - \alpha x_1$. Note that in the case where $P = Q$, we take l to be the tangent at P in accordance with the rules of addition on the curve $E(\mathbf{R})$. A point $(x, \alpha x + \beta)$ on the line l lies on the elliptic curve if and only if $(\alpha x + \beta)^2 = x^3 + ax + b$. Thus, there is one intersection point for each root of the cubic equation $x^3 - (\alpha x + \beta)^2 + ax + b = 0$. The numbers x_1 and x_2 are roots of this equation because $(x_1, \alpha x_1 + \beta)$ and $(x_2, \alpha x_2 + \beta)$ are, respectively, the points P and Q on the curve. Hence, the equation must have a third root x_3 where $x_1 + x_2 + x_3 = \alpha^2$. This leads to the expression for x_3 mentioned in the rules of addition for the curve $E(\mathbf{R})$. Thus, geometrically, the operation of addition on $E(\mathbf{R})$ corresponds to drawing the line through P and Q, letting the third intercept of the line with the curve be $-R = (x, y)$ and taking $R = (x, -y)$ to be the sum $P + Q$.

B.11.1 Elliptic Curves Over Z_n

Lenstra's elliptic curve factoring algorithm works with elliptic curves $E_{a,b}(\mathbf{Z}_n)$ defined over the ring $\mathbf{Z}_n$ where n is an odd, composite integer. The nonsingularity condition $4a^3+27b^2 \neq 0$ is replaced by $\gcd(4a^3+27b^2, n) = 1$. The negation and addition rules are given as was done for elliptic curves over fields. However, the addition of two points involves a division (refer to equations $(B.2)$ and $(B.3)$ in the rules for arithmetic on elliptic curves given at the beginning of this section) which is not always defined over the ring $\mathbf{Z}_n$. For addition to be defined the denominators in these equations must be prime to n. Consequently, $E_{a,b}(\mathbf{Z}_n)$ is not necessarily a group. Nevertheless, we may define a method for computing multiples $e \cdot P$ of a point $P \in E_{a,b}(\mathbf{Z}_n)$ as follows.

1. Let $a_0 + a_1 2 + \cdots + a_{m-1}2^{m-1}$ be the binary expansion of e. Let $j = 0$, $S = 0$.
2. If $a_j = 1$ then $S \leftarrow S + 2^j P$. If this sum is not defined (namely, the division in equation $(B.2)$ or equation $(B.3)$ has failed) then output undefined and terminate.
3. $j \leftarrow j + 1$. If $j = m$ then output S as the defined value for $e \cdot P$ and terminate.
4. Calculate $2^j P := 2^{j-1}P + 2^{j-1}P$. If this sum is not defined then output that $e \cdot P$ cannot be calculated and terminate. Otherwise, repeat from step 2.

The elliptic curve algorithm will use[1] this method which will be referred to as repeated doubling. Note that if the repeated doubling method is unable to calculate a given multiple $e \cdot P$ and outputs undefined then we have encountered points $Q_1 = (x_1, y_1)$ and $Q_2 = (x_2, y_2)$ on $E_{a,b}(\mathbf{Z}_n)$ such that $Q_1 + Q_2$ is not defined modulo n (the division in equation $(B.2)$ or equation $(B.3)$ has failed) and hence, either $\gcd(x_1 - x_2, n)$ or $\gcd(y_1 + y_2, n)$ is a nontrivial factor of n.

Next, we state some facts concerning the relationship between elliptic curves defined over $\mathbf{Z}_n$ and elliptic curves defined over $\mathbf{Z}_p$ when p is a prime divisor of n. Let $a, b \in \mathbf{Z}_n$ be such that $\gcd(4a^3 + 27b^2, n) = 1$. Let p be a prime divisor of n and let $a_p = a \bmod p$, $b_p = b \bmod p$.

Fact B.11.2 $E_{a_p,b_p}(\mathbf{Z}_p)$ is an additive abelian group.

Further, given $P = (x, y) \in E_{a,b}(\mathbf{Z}_n)$, define $P_p = (x \bmod p, y \bmod p)$. P_p is a point on the elliptic curve $E_{a_p,b_p}(\mathbf{Z}_p)$.

Fact B.11.3 Let P and Q be two points on $E_{a,b}(\mathbf{Z}_n)$ and let p be a prime divisor of n. If $P + Q$ is defined modulo n then $P_p + Q_p$ is defined on $E_{a,b}(\mathbf{Z}_p)$ and $P_p + Q_p = (P + Q)_p$. Moreover, if $P \neq -Q$ then the sum $P + Q$ is undefined modulo n if and only if there is some prime divisor q of n such that the points P_q and Q_q add up to the point at infinity O on $E_{a_q,b_q}(\mathbf{Z}_q)$ (equivalently, $P_q = -Q_q$ on $E_{a_q,b_q}(\mathbf{Z}_q)$).

B.11.2 Factoring Using Elliptic Curves

The main idea in Lenstra's elliptic curve factoring algorithm is to find points P and Q in $E_{a,b}(\mathbf{Z}_n)$ such that $P + Q$ is <u>not defined</u> in $E_{a,b}(\mathbf{Z}_n)$. We will assume throughout that n has no factors of 2 or 3 because these can be divided out before the algorithm commences.

The algorithm runs as follows on input n.

1. Generate an elliptic curve $E_{a,b}(\mathbf{Z}_n)$ and a point $P = (x, y)$ on $E_{a,b}(\mathbf{Z}_n)$ by randomly selecting x, y and a in $\mathbf{Z}_n$ and setting $b = y^2 - x^3 - ax \bmod n$.
2. Compute $\gcd(4a^3 + 27b^2, n)$. If $1 < \gcd(4a^3 + 27b^2, n) < n$ then we have found a divisor of n and we stop. If $\gcd(4a^3 + 27b^2, n) = 1$ then $4a^3 + 27b^2 \not\equiv 0 \bmod p$ for every prime divisor p of n and hence $E_{a,b}$ is an elliptic curve over $\mathbf{Z}_p$ for each prime divisor p of n and we may proceed. But if $\gcd(4a^3 + 27b^2, n) = n$ then we must generate another elliptic curve $E_{a,b}$.

[1] This statement is incorrect and will soon be corrected in these notes. For the method used in the algorithm, see section 2.3 of [116]

3. Set $e = \prod_{i=1}^{\pi(B)} p_i^{\alpha_i}$ where $p_1, p_2, \ldots, p_{\pi(B)}$ are the primes $\leq B$ and α_i is chosen maximally so that $p_i^{\alpha_i} \leq C$. B and C are bounds that will be determined later so as to optimize the running time and ensure that the algorithm will most likely succeed.

4. Compute $e \cdot P$ in $E_{a,b}(\mathbf{Z}_n)$ by repeated doubling. Every time before adding two intermediate points $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ check if $\gcd(x_1 - x_2, n)$ or $\gcd(y_1 + y_2, n)$ is a nontrivial factor of n. If so, output the factor and stop. Otherwise, repeat from step 1.

An elliptic curve $E_{a,b}(\mathbf{Z}_n)$ will lead to a nontrivial factorization of n if for some prime factors p and q of n, $e \cdot P_p = O$ on $E_{a_p,b_p}(\mathbf{Z}_p)$ but P_q does not have order dividing e on $E_{a_q,b_q}(\mathbf{Z}_q)$. Notice the analogy here between Lenstra's elliptic curve algorithm and Pollard's $p-1$ algorithm. In Pollard's algorithm we seek prime divisors p and q of n such that e is a multiple of the order of $a \in \mathbf{Z}_p^*$ but not a multiple of the order of $a \in \mathbf{Z}_q^*$. Similarly, in Lenstra's algorithm we seek prime divisors p and q of n such that e is a multiple of the order of $P_p \in E_{a_p,b_p}(\mathbf{Z}_p)$ but not a multiple of the order of $P_q \in E_{a_q,b_q}(\mathbf{Z}_q)$. However, there is a key difference in versatility between the two algorithms. In Pollard's algorithm, the groups $\mathbf{Z}_p^*$ where p ranges over the prime divisors of n are fixed so that if none of these groups have order dividing e then the method fails. In Lenstra's elliptic curve algorithm the groups $E_{a_p,b_p}(\mathbf{Z}_p)$ can be varied by varying a and b. Hence, if for every prime divisor p of n, $|E_{a_p,b_p}(\mathbf{Z}_p)| \not| e$, then we may still proceed by simply working over another elliptic curve; that is, choosing new values for a and b.

B.11.3 Correctness of Lenstra's Algorithm

Suppose that there are prime divisors p and q of n such that e is a multiple of $|E_{a_p,b_p}(\mathbf{Z}_p)|$ but in $E_{a_q,b_q}(\mathbf{Z}_q)$, P_q does not have order dividing e. Then $e \cdot P_p = O$ in $E_{a_p,b_p}(\mathbf{Z}_p)$ but $e \cdot P_q \neq O$ in $E_{a_q,b_q}(\mathbf{Z}_q)$ and thus there exists an intermediate addition of two points $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ in the calculation of $e \cdot P$ such that

$$x_1 \equiv x_2 \bmod p \text{ but } x_1 \not\equiv x_2 \bmod q \text{ if } P_1 \neq P_2$$

or

$$y_1 \equiv -y_2 \bmod p \text{ but } y_1 \not\equiv -y_2 \bmod q \text{ if } P_1 = P_2.$$

Hence, either $\gcd(x_1 - x_2, n)$ or $\gcd(y_1 + y_2, n)$ is a nontrivial factor of n. The points P_1, and P_2 will be encountered when $(P_1)_p + (P_2)_p = O$ in $E_{a_p,b_p}(\mathbf{Z}_p)$ but $(P_1)_q + (P_2)_q \neq O$ in $E_{a_q,b_q}(\mathbf{Z}_q)$.

B.11.4 Running Time Analysis

The time needed to perform a single addition on an elliptic curve can be taken to be $M(n) = O(\log^2 n)$ if one uses the Euclidean algorithm. Consequently, since the computation of $e \cdot P$ uses repeated doubling, the time required to process a given elliptic curve is $O((\log e)(M(n))$. Recall that $e = \prod_{i=1}^{\pi(B)} p_i^{\alpha_i}$ where $p_i^{\alpha_i} \leq C$. Then $e \leq C^{\pi(B)}$ and therefore, $\log e \leq \pi(B) \log C$. Now, let p be the smallest prime divisor of n and consider the choice $B = L^{\beta}(p) = \exp[\beta(\log p)^{1/2}(\log\log p)^{1/2}]$ where β will be optimized. Then

$$\log B = \beta(\log p)^{1/2}(\log\log p)^{1/2} = e^{[\log\beta + \frac{1}{2}(\log\log p) + \frac{1}{2}(\log\log\log p)]}$$

and thus

$$\pi(B) \approx \frac{B}{\log B} \approx O\left(e^{[\beta(\log p)^{1/2}(\log\log p)^{1/2} - \frac{1}{2}(\log\log p)]}\right) \approx O(L^{\beta}(p)).$$

Hence, the time required for each iteration of the algorithm is $O(L^{\beta}(p)M(n)(\log C))$. In choosing C, note that we would like e to be a multiple of $|E_{a_p,b_p}(\mathbf{Z}_p)|$ for some prime divisor p of n and thus it is sufficient to take $C = |E_{a_p,b_p}(\mathbf{Z}_p)|$ where p is some prime divisor of n, provided that $|E_{a_p,b_p}(\mathbf{Z}_p)|$ is B-smooth. The value of p is unknown, but if p is the smallest prime divisor of n then $p < \sqrt{n}$. We also know by Hasse's Inequality (see, for example, page 131 of [172]) that $p + 1 - 2\sqrt{p} < |E_{a_p,b_p}(\mathbf{Z}_p)| < p + 1 + 2\sqrt{p}$ and thus $|E_{a_p,b_p}(\mathbf{Z}_p)| < \sqrt{n} + 1 + 2\sqrt[4]{n}$. Hence, it is safe to take $C = \sqrt{n} + 1 + 2\sqrt[4]{n}$.

The only remaining consideration is to determine the expected number of elliptic curves that must be examined before we obtain a factorization of n and this part of the analysis relies on the following result due to Lenstra which appears as Proposition 2.7 in [116].

Proposition B.11.4 *Let $S = \{s \in \mathbf{Z} : |s - (p+1)| < \sqrt{p}$ and s is $L(p)^{\beta}$-smooth$\}$. Let n be a composite integer that has at least two distincet prime divisors exceeding 3. Then*

$$\Pr[\textit{Lenstra's algorithm factors } n] \geq \Omega\left(\frac{|S|-2}{2\sqrt{p}}\right)\left(\frac{1}{\log p}\right)$$

(where the probability is taken over x, y, and a in $\mathbf{Z}_n$).

In other words, the proposition asserts that the probability that a random triple (x, y, a) leads to a factorization of n is essentially the probability that a random integer in the interval $(p+1-\sqrt{p}, p+1+\sqrt{p})$ is $L(p)^{\beta}$-smooth; the latter probability being $\frac{|S|}{2\lfloor\sqrt{p}\rfloor+1}$.
Recall that we dealt earlier with the smoothness of integers less than some bound and saw that a theorem due to Canfield, Erdös and Pomerance (see [44]) implies that

$$\Pr[m \leq x \text{ is } L(x)^{\alpha}\text{-smooth}] = L^{-\frac{1}{2\alpha}}(x).$$

However, as we have just seen, we require here the unproven conjecture that the same result is valid if m is a random integer in the small interval $(p+1-\sqrt{p}, p+1+\sqrt{p})$; specifically, that

$$\Pr[m \in (p+1-\sqrt{p}, p+1+\sqrt{p}) \text{ is } L(p)^{\beta}\text{-smooth}] = L^{-\frac{1}{2\beta}}(p).$$

Consequently, the lower bound on the probability of success in Proposition B.11.4 can be made explicit. Hence, we have

$$\Pr[\text{Lenstra's algorithm factors } n] \geq \Omega\left(L^{-\frac{1}{2\beta}}(p)\right)\frac{1}{\log p}.$$

Therefore, we expect to have to try $L^{\frac{1}{2\beta}}(p)(\log p)$ elliptic curves before encountering one of $L^{\beta}(p)$-smooth order. Thus, the total running time required is expected to be $O(L^{\frac{1}{2\beta}}(p)(\log p)L^{\beta}(p)M(n)(\log C)) = O(L^{\beta+\frac{1}{2\beta}}(p)(\log^4 n))$. This achieves a minimum of $O(L^{\sqrt{2}}(p)(\log^4 n))$ when $\beta = \frac{1}{\sqrt{2}}$.

Remark In step 3 of Lenstra's algorithm a minor practical problem arises with the choice of $B = L^{\beta}(p)$ because the smallest prime divisor p of n is not known before the algorithm begins. This problem can be resolved by taking $B = L^{\beta}(v)$ and performing the algorithm for a gradually increasing sequence of values for v while factorization continues to be unsuccessful and declaring failure if v eventually exceeds $\sqrt{n}$ because the smallest prime divisor of n is less than $\sqrt{n}$.

CHAPTER C

About PGP

PGP is a free software package that performs cryptographic tasks in association with email systems. In this short appendix we will review some of its features. For a complete description of its functioning readers are referred to Chapter 9 in [174].

C.1 Authentication

PGP performs authentication of messages using a hash-and-sign paradigm. That is given a message M, the process is as following:

- The message is timestamped, i.e. date and time are appended to it;
- it is then hashed using MD5 (see [150]);
- the resulting 128-bit digest is signed with the sender private key using RSA [151];
- The signature is prepended to the message.

C.2 Privacy

PGP uses a hybrid system to ensure privacy. That is each message is encrypted using a fast symmetric encryption scheme under a one-time key. Such key is encrypted with the receiver public-key and sent together with the encrypted message.

In detail, assume A wants to send an encrypted message to B.

- A compresses the message using the ZIP compression package; let M be the resulting compressed message.
- A generates a 128-bit random key k;
- The message M is encrypted under k using the symmetric encryption scheme IDEA (see [111] or Chapter 7 of [174]); let C be the corresponding ciphertext;
- k is encrypted under B's public key using RSA; let c be the corresponding ciphertext.
- The pair (c, C) is sent to B.

If both authentication and privacy are required, the message is first signed, then compressed and then encrypted.

C.3 Key Size

PGP allows for three key sizes for RSA

- *Casual* 384 bits
- *Commercial* 512 bits
- *Military* 1024 bits

C.4 E-mail compatibility

Since e-mail systems allow only the transmission of ASCII characters, PGP needs to recovert eventual encrypted parts of the message (a signature or the whole ciphertext) back to ASCII.

In order to do that PGP applies the radix-64 conversion to bring back a binary stream into the ASCII character set. This conversion expands the message by 33%. However because of the original ZIP compression, the resulting ciphertext is still one-third smaller than the original message.

In case the resulting ciphertext is still longer than the limit on some e-mail systems, PGP breaks into pieces and send the messages separately.

C.5 One-time IDEA keys generation

Notice that PGP does not have session keys, indeed each message is encrypted under a key k generated ad hoc for that message.

The generation of such key is done using a pseudo-random number generator that uses IDEA as a building block. The seed is derived from the keystrokes of the user. That is, form the actual keys being typed and the time intervals between them.

C.6 Public-Key Management

Suppose you think that PK is the public key of user B, while instead it is C who knows the corresponding secret key SK.

This can create two major problems:

1. C can read encrypted messages that A thinks she is sending to B
2. C can have A accept messages as coming from B.

The problem of establishing trust in the connection between a public-key and its owner is at the heart of public-key systems, no just of PGP.

There are various ways of solving this problem:

- *Physical exchange* B could give the key to A in person, stored in a floppy disk.
- *Verification* A could call B on the phone and verify the key with him
- *Certification Authorithy* There could be a trusted center $AUTH$ that signs public keys for the users, establishing the connection between the key and the ID of the user (such a signature is usually referred to as a *certificate*.)

Only the last one seems reasonable and it appears to be the way people are actually implementing public key systems in real life.

PGP does not use any of the above systems, but it rather uses a *decentralized trust* system. Users reciprocally certify each other's keys and one trusts a key to the extent that he/she trusts the user who certify it for it. Details can be found in [174]

CHAPTER D

Problems

This chapter contains some problems for you to look at.

D.1 Secret Key Encryption

D.1.1 DES

Let $\bar{m}$ be the bitwise complement of the string m. Let $\mathrm{DES}_K(m)$ denote the encryption of m under DES using key K. It is not hard to see that if

$$c = \mathrm{DES}_K(m)$$

then

$$\bar{c} = \mathrm{DES}_{\bar{K}}(\bar{m})$$

We know that a brute–force attack on DES requires searching a space of 2^{56} keys. This means that we have to perform that many DES encryptions in order to find the key, in the worst case.

1. Under *known plaintext attack* (i.e., you are given a single pair (m, c) where $c = \mathrm{DES}_K(m)$) do the equations above change the number of DES encryption you perform in a brute–force attack to recover K?

2. What is the answer to the above question in the case of *chosen plaintext attack* (i.e., when you are allowed to choose many m's for which you get the pair (m, c) with $c = \mathrm{DES}_K(m)$)?

D.1.2 Error Correction in DES ciphertexts

Suppose that n plaintext blocks $x_1, \ldots, x_n$ are encrypted using DES producing ciphertexts $y_1, \ldots, y_n$. Suppose that one ciphertext block, say y_i, is transmitted incorrectly (i.e. some 1's are changed into 0's and viceversa.) How many plaintext blocks will be decrypted incorrectly if the ECB mode was used for encryption? What if CBC is used?

D.1.3 Brute force search in CBC mode

A brute-force key search for a known-plaintext attack for DES in the ECB mode is straightforward: given the 64-bit plaintext and the 64 bit ciphertext, try all of the possible 2^{56} keys until one is found that generates the known ciphertext from the known plaintext. The situation is more complex for the CBC mode, which includes the use of a 64-bit IV. This seems to introduce an additional 64 bits of uncerntainty.

1. Suggest strategies for known-plaintext attack on the CBC mode that are of the same order of magnitude of effort as the ECB attack.

2. Now consider a ciphertext only attack. For ECB mode the strategy is to try to decrypt the given ciphertext with all possible 2^{56} keys and test each result to see if it appears to be a syntactically correct plaintext. Will this strategy work for the CBC mode? If so, explain. If not, describe an attack strategy for the CBC mode and estimate its level of effort.

D.1.4 E-mail

Electronic mail systems differ in the way in which multiple recipients are handled. In some systems the originating mail handler makes all the necessary copies, and these are sent out independently. An alterantive approach is to determine the route for each destination first. Then a single message is sent out on a common portion of the route and copies are made when the routes diverge (this system is known as *mail-bagging.*)

1. Leaving aside security considerations, discuss the relative advantages and disadvantages of the two methods.

2. Discuss the security requirements and implications of the two methods

D.2 Number Theory

D.2.1 Number Theory Facts

Prove the following facts:

1. If k is the number of distinct prime factors of n then the equation $x^2 = 1 \bmod n$ has 2^k distinct solutions in Z_n^*. *Hint: use Chinese Remainder Theorem*

2. If p is prime and $x \in Z_p^*$ then $(\frac{x}{p}) = x^{\frac{p-1}{2}}$

3. g is a generator of Z_p^* for a prime p, iff $g^{p-1} = 1 \bmod p$ and $g^q \neq 1 \bmod p$ for all q prime divisors of $p-1$

D.2.2 Relationship between problems

Let n be the product of two primes $n = pq$. Describe reducibilities between the following problems (e.g. if we can factor we can invert RSA.) Don't prove anything formally, just state the result.

- computing $\phi(n)$
- factoring n
- computing $QR_n(a)$ for some $a \in Z_n^*$
- computing square roots modulo n
- computing k-th roots modulo n, where $gcd(k, \phi(n)) = 1$

D.2.3 Probabilistic Primality Test

Let $SQRT(p, a)$ denote an expected polynomial time algorithm that on input p, a outputs x such that $x^2 = a \bmod p$ if a is a quadratic residue modulo p. Consider the following probabilistic primality test, which takes as an input an odd integer $p > 1$ and outputs "composite" or "prime".

1. Test if there exist $b, c > 1$ such that $p = b^c$. If so output "composite"

2. Choose $i \in Z_p^*$ at random and set $y = i^2$

3. Compute $x = SQRT(p, y)$

4. If $x = i \bmod p$ or $x = -i \bmod p$ output "prime", otherwise output "composite"

(A) Does the above primality test always terminate in expected polynomial time? Prove your answer.
(B) What is the probability that the above algorithm makes an error if p is prime?
(C) What is the probability that the above algorithm makes an error if p is composite?

D.3 Public Key Encryption

D.3.1 Simple RSA question

Suppose that we have a set of block encoded with the RSA algorithm and we don't have the private key. Assume $n = pq, e$ is the public key. Suppose also someone tells us they know one of the plaintext blocks has a common factor with n. Does this help us in any way?

D.3.2 Another simple RSA question

In the RSA public-key encyption scheme each user has a public key n, e and a private key d. Suppose Bob leaks his private key. Rather than generating a new modulus, he decides to generate a new pair e', d'. Is this a good idea?

D.3.3 Protocol Failure involving RSA

Remember that an RSA public–key is a pair (n, e) where n is the product of two primes.

$$RSA_{(n,e)}(m) = m^e \bmod n$$

Assume that three users in a network Alice, Bob and Carl use RSA public–keys $(n_A, 3)$, $(n_B, 3)$ and $(n_C, 3)$ respectively. Suppose David wants to send the *same* message m to the three of them. So David computes

$$y_A = m^3 \bmod n_A \ , \ y_B = m^3 \bmod n_B \ , \ y_C = m^3 \bmod n_C$$

and sends the ciphertext to the relative user.

Show how an eavesdropper Eve can now compute the message m even without knowing any of the secret keys of Alice, Bob and Carl.

D.3.4 RSA for paranoids

The best factoring algorithm known to date (the *number field sieve*) runs in

$$e^{O(\log^{1/3} n \log\log^{2/3} n)}$$

That is, the running time does not depend on the size of the smallest factor, but rather in the size of the whole composite number.

The above observation seem to suggest that in order to preserve the security of RSA, it may not be necessary to increase the size of both prime factors, but only of one of them.

Shamir suggested the follwong version of RSA that he called *unbalanced RSA* (also known as RSA for paranoids). Choose the RSA modulus n to be 5,000 bits long, the product of a 500-bits prime p and a 4,500-bit prime q. Since usually RSA is usually used just to exchange DES keys we can assume that the messages being encrypted are smaller than p.

(A) How would you choose the public exponent e? Is 3 a good choice?

Once the public exponent e is chosen, one computes $d = e^{-1} \bmod \phi(n)$ and keep it secret. The problem with such a big modulus n, is that decrypting a ciphertext $c = m^e \bmod n$ may take a long time (since one has to compute $c^d \bmod n$.) But since we know that $m < p$ we can just use the Chinese Remainder Theorem

and compute $m_1 = c^d \bmod p = m$. Shamir claimed that this variant of RSA achieves better security against the advances of factoring, without losing in efficiency.

(B) Show how with a single chosen message attack (i.e. obtaining the decryption of a message of your choice) you can completely break the unbalanced RSA scheme, by factoring n.

D.3.5 Hardness of Diffie-Hellman

Recall the Diffie-Hellman key exchange protocol. p is a prime and g a generator of Z_p^*. Alice's secret key is a random $a < p$ and her public key is $g^a \bmod p$. Similarly Bob's secret key is a random $b < p$ and his public key is $g^b \bmod p$. Their common key is g^{ab}.

In this problem we will prove that if the Diffie-Hellman key exchange protocol is secure for a small fraction of the values (a, b), then it is secure for almost all values (a, b).

Assume that there is a *ppt* algorithm $\mathcal{A}$ that

$$Prob[\mathcal{A}(g^a, g^b) = g^{ab}] > \frac{1}{2} + \epsilon$$

(where the probability is taken over the choices of (a, b) and the internal coin tosses of $\mathcal{A}$)

Your task is to prove that for any $\delta < 1$ there exists a *ppt* algorithm $\mathcal{B}$ such that for all (a, b)

$$Prob[\mathcal{B}(g^a, g^b) = g^{ab}] > 1 - \delta$$

(where the probability is now taken only over the coin tosses of $\mathcal{B}$)

D.3.6 Bit commitment

Consider the following "real life" situation. Alice and Bob are playing "Guess the bit I am thinking". Alice thinks a bit $b = 0, 1$ and Bob tries to guess it. Bob declares his guess and Alice tells him if the guess is right or not.

However Bob is losing all the time so he suspects that Alice is cheating. She hears Bob's guess and she declares she was thinking the opposite bit. So Bob requires Alice to write down the bit in a piece of paper, seal it in an envelope and place the envelope on the table. At this point Alice is committed to the bit. However Bob has no information about what the bit is.

Our goal is to achieve this bit commitment without envelopes. Consider the following method. Alice and Bob together choose a prime p and a generator g of Z_p^*. When Alice wants to commit to a bit b she choose a random $x \in Z_p^*$ such that $lsb(x) = b$ and she publishes $y = g^x \bmod p$. Is this a good bit commitment? Do you have a better suggestion?

D.3.7 Perfect Forward Secrecy

Suppose two parties, Alice and Bob, want to communicate privately. They both hold public keys in the traditional Diffie-Hellman model.

An eavesdropper Eve stores all the encrypted messages between them and one day she manages to break into Alice and Bob's computer and find their secret keys, correspondent to their public keys.

Show how using only public–key cryptography we can achieve *perfect forward secrecy*, i.e., Eve will not be able to gain any knowledge about the messages Alice and Bob exchanged before the disclosure of the secret keys.

D.3.8 Plaintext-awareness and non-malleability

We say that an encryption scheme is *plaintext–aware* if it is impossible to produce a valid ciphertext without knowing the corresponding plaintext.

Usually plaintext-aware encryption schemes are implemented by adding some redundancy to the plaintext. Decryption of a ciphertext results either in a valid message or in a flag indicating non–validity (if the redundancy is not of the correct form.) Correct decryption convinces the receiver that the sender *knows* the plaintext that was encrypted.

The concept of plaintext–awareness is related to the concept of malleability. We say that an encryption scheme E is *non–malleable* if it given a ciphertext $c = E(m)$ it is impossible to produce a valid ciphertext c' of a related message m'.

Compare the two definitions and tell us if one implies the other.

D.3.9 Probabilistic Encryption

Assume that you have a message m that you want to encrypt in a probabilistic way. For each of the following methods, tell us if you think it is a good or a bad method.

1. Fix p a large prime and let g be a generator. For each bit b_i in m, choose at random $x_i \in Z_{p-1}$ such that $lsb(x_i) = b_i$ ($lsb(x)$ = least significant bit of x.) The ciphertext is the concatenation of the $y_i = g^{x_i} \bmod p$. What about if you use x such that $msb(x_i) = b_i$?

2. Choose an RSA public key n, e such that $|n| > 2|m|$. Pad m with random bits to get it to the same length of n. Let m' be the padded plaintext. Encrypt $c = m^e \bmod n$.

3. Choose an RSA public key n, e. Assume that $|m|$ is smaller than $\log\log n$ (you can always break the message in blocks of that size.) Pad m with random bits to get it to the same length of n. Let m' be the padded plaintext. Encrypt $c = m^e \bmod n$.

4. Choose two large primes $p, q = 3 \bmod 4$. Let $n = pq$. For each bit b_i in m, choose at random $x_i \in Z_n^*$ and set $y_i = x_i^2 \bmod n$ if $b_i = 0$ or $y_i = -x_i^2 \bmod n$ if $b_i = 1$. The ciphertext is the concatenation of the y_i's.

D.4 Secret Key Authentication

D.4.1 CBC MAC with variable length

Alice and Bob share a secret key k. Bob wants an assurance that the message he receives are really coming from Alice. In order to obtain this Alice sends the message m and a MAC along with it $\mathrm{MAC}_k(m)$.

We say that a MAC is secure against *chosen message attack* if an attacker Eve, after requesting the MAC $\mathrm{MAC}_k(m_i)$ of n messages of her choice $m_1, \ldots, m_n$ will not be able to produce a *new* message $m \neq m_i$ and a valid $\mathrm{MAC}_k(m)$.

CBC-MAC is a way of implementing MACs using symmetric encryption scheme. Let

$$f_k : \{0,1\}^\ell \to \{0,1\}^\ell$$

be an encryption function with key k (for example with $\ell = 64$, f could be DES.) Let m be a message of length $b\ell$ bits.

$$m = m_1 \circ \ldots \circ m_b$$

where $\circ$ denotes concatenation and m_i is a block of length ℓ bits. The MAC of m is computed using the cipher f in CBC mode:

$$\mathrm{MAC}_k(m) = f_k(f_k(\ldots f_k(f_k(m_1) \oplus m_2) \oplus \ldots m_{b-1}) \oplus m_b)$$

1. Show that the CBC–MAC does not support variable length input. That is show that if messages have variable length it is possible to device a chosen message attack. *Hint: The idea is to ask for the MACs of two messages of length ℓ and construct from them the MAC of a message of length 2ℓ.*

2. Suppose the length of the messages is always smaller than 2^ℓ. An attempt to get around the above problem would be to append the length of the message at the end as an extra block, before computing the MAC. Show that it is still possible to mount an effective chosen message attack.

3. Think of a possible fix.

D.4.2 PRFs are good MACs

Suppose $\mathcal{F}$ is a (finite) family of pseudo-random functions with input length ℓ, output length L and key length k. In this problem you are to show that $\mathcal{F}$ is a secure MAC on ℓ-bit messages, and specify the security of the MAC as a function of the security ofthe pseudo-random function family.

More precisely, the parties share a random k-bit key K specifying a function F_K from the family. The MAC on a message M is $\sigma = F_K(M)$. The recipient of a tagged message (M, σ) accepts iff $F_K(M) = \sigma$. Suppose $\mathcal{F}$ is a (t, q, ϵ)-secure PRF. Provide values t', q', ϵ' functions of $t, q, \epsilon, \ell, L, k$, such that the above MAC scheme is (t', q', ϵ')-secure.

D.4.3 Simultaneous encryption and authentication

Let $(\mathcal{E}, \mathcal{D})$ be a symmetric encryption scheme (cf. Chapter 5and MAC a message authentication code (cf. Chapter 7). Suppose Alice and Bob share two keys K_1 and K_2 for privacy and authentication respectively. They want to exchange messages M in a private and authenticated way. Consider sending each of the following as a means to this end:

1. $M, MAC_{K_2}(\mathcal{E}_{K_1}(M))$
2. $\mathcal{E}_{K_1}(M, MAC_{K_2}(M))$
3. $\mathcal{E}_{K_1}(M), MAC_{K_2}(M)$
4. $\mathcal{E}_{K_1}(M), \mathcal{E}_{K_1}(MAC_{K_2}(M))$
5. $\mathcal{E}_{K_1}(M), MAC_{K_2}(\mathcal{E}_{K_1}(M))$
6. $\mathcal{E}_{K_1}(M, A)$ where A encodes the identity of Alice. Bob decrypts the ciphertext and checks that the second half of the plaintext is A

For each say if it secure or not and briefly justify your answer.

D.5 Hash Functions

D.5.1 Birthday Paradox

Let H be a hash function that outputs m-bit values. Assume that H behaves as a *random oracle*, i.e. for each string s, $H(s)$ is uniformly and independently distributed between 0 and $2^m - 1$.

Consider the following brute–force search for a collision: try all possible $s_1, s_2, \ldots$ until a collision is found. (That is, keep hashing until some string yields the same hash value as a previously hashed string.)

Prove that the expected number of hashing performed is approximately $2^{\frac{m}{2}}$.

D.5.2 Hash functions from DES

In this problem we will consider two proposals to construct hash functions from symmetric block encryption schemes as DES.

Let E denote a symmetric block encryption scheme. Let $E_k(M)$ denote the encryption of the 1–block message M under key k. Let $M = M_0 \circ M_1 \circ M_2 \circ \ldots \circ M_n$ denote a message of $n + 1$ blocks.

The first proposed hash function h_1 works as follows: let $H_0 = M_0$ and then define

$$H_i = E_{M_i}(H_{i-1}) \oplus H_{i-1} \quad \text{for} \quad i = 1, \ldots, n.$$

The value of the hash function is defined as

$$h_1(M) = H_n$$

The second proposed hash function h_2 is similar. Again $H_0 = M_0$ and then

$$H_i = E_{H_{i-1}}(M_i) \oplus M_i \quad \text{for} \quad i = 1, \ldots, n.$$

The value of the hash function is defined as

$$h_2(M) = H_n$$

For both proposals, show how to find collisions if the encryption scheme E is chosen to be DES.

D.5.3 Hash functions from RSA

Consider the followng hash function H. Fix an RSA key n, e and denote with $RSA_{n,e}(m) = m^e \bmod n$. Let the message to be hashed be $m = m_1 \ldots m_k$. Denote with $h_1 = m_1$ and for $i > 1$,

$$h_i = RSA_{n,e}(h_{i-1}) \oplus m_i$$

Then $H(m) = h_n$. Show how to find a collision.

D.6 Pseudo-randomness

D.6.1 Extending PRGs

Suppose you are given a PRG G which stretches a k bit seed into a $2k$ bit pseudorandom sequence. We would like to construct a PRG G' which stretches a k bit seed into a $3k$ bit pseudorandom sequence.

Let $G_1(s)$ denote the first k bits of the string $G(s)$ and let $G_2(s)$ the last k bits (that is $G(s) = G_1(s).G_2(s)$ where $a.b$ denotes the concatenation of strings a and b.)

Consider the two constructions

1. $G'(s) = G_1(s).G(G_1(s))$
2. $G''(s) = G_1(s).G(G_2(s))$

For each construction say whether it works or not and justify your answer. That is, if the answer is no provide a simple statistical test that distinguishes the output of, say, G' from a random $3k$ string. If the answer is yes prove it.

D.6.2 From PRG to PRF

Let us recall the construction of PRFs from PRGs we saw in class. Let G be a length-doubling PRG, from seed of length k to sequences of length $2k$.

Let $G_0(x)$ denote the first k bits of $G(x)$ and $G_1(x)$ the last k bits. In other words $G_0(x) \circ G_1(x) = G(x)$ and $|G_0(x)| = |G_1(x)|$.

For any bit string z recursively define $G_{0 \circ z}(x) \circ G_{1 \circ z}(x) = G(G_z(x))$ with $|G_{0 \circ z}(x)| = |G_{1 \circ z}|$.

The PRF family we constructed in class was defined as $\mathcal{F} = \{f_i\}$. $f_i(x) = G_x(i)$. Suppose instead that we defined $f_i(x) = G_i(x)$. Would that be a PRF family?

D.6.3 From PRF to PRG

Suppose you have a keyed PRF family $\mathcal{F}$, with key length k, input length ℓ and output length L. (For example $\mathcal{F} = DES$, $k = 56$, $\ell = L = 64$.) Assume $\mathcal{F}$ is (t, q, ϵ)-secure for some t, q, ϵ with $q \leq 2^\ell$. Given a number n (with $k < n \leq q$) provide a simple construction of a pseudorandom generator G which stretches k bits to n bits, and analyze its security. Namely, computet', ϵ', functions of $t, q, \epsilon, k, \ell, L, n$ such that G is (t', ϵ')-secure.

D.7 Digital Signatures

D.7.1 Table of Forgery

For both RSA and ElGamal say if the scheme is

1. universally forgeable
2. selectively forgeable
3. existentially forgeable

and if it is under which kind of attack.

D.7.2 ElGamal

Suppose Bob is using the ElGamal signature scheme. Bob signs two messages m_1 and m_2 with signatures (r, s_1) and (r, s_2) (the same value of r occurs in both signatures.) Suppose also that $gcd(s_1 - s_2, p-1) = 1$.

1. Show how k can be computed efficiently given this information
2. Show how the signature scheme can subsequently be broken

D.7.3 Suggested signature scheme

Consider the following discrete log based signature scheme. Let p be a large prime and g a generator. The private key is $x < p$. The public key is $y = g^x \bmod p$.

To sign a message M, calculate the hash $h = H(M)$. If $gcd(h, p-1)$ is different than 1 then append h to M and hash again. Repeat this until $gcd(h, p-1) = 1$. Then solve for Z in

$$Zh = X \bmod (p-1)$$

The signature of the message is $s = g^Z \bmod p$. To verify the signature, a user checks that $s^h = Y \bmod p$.

1. Show that valid signatures are always accepted
2. Is the scheme secure?

D.7.4 Ong-Schnorr-Shamir

Ong, Schnorr and Shamir suggested the following signature scheme.

Let n be a large integer (it is not necessary to know the factorization of n.) Then choose $k \in Z_n^*$. Let

$$h = -k^{-2} \bmod n = -(k^{-1})^2 \bmod n$$

The public key is (n, h), the secret key is k.

To sign a message M, generate a random number r, such that r and n are relatively prime. Then calculate

$$S_1 = \frac{M/r + r}{2} \bmod n$$

$$S_2 = \frac{k}{2}(M/r - r)$$

The pair (S_1, S_2) is the signature.

To verify the signature, check that

$$M = S_1^2 + hS_2^2 \bmod$$

1. Prove that reconstructing the private key, from the public key is equivalent to factor n.
2. Is that enough to say that the scheme is secure?

D.8 Protocols

D.8.1 Unconditionally Secure Secret Sharing

Consider a generic Secret Sharing scheme. A dealer D wants to share a secret s between n trustees so that no t of them have *any* information about s, but $t+1$ can reconstruct the secret. Let s_i be the share of trustee T_i. Let v denote the number of possible values that s might have, and let w denote the number of different possible share values that a given trustee might receive, as s is varied. (Let's assume that w is the same for each trustee.)

Argue that $w \geq v$ for any Secret Sharing Scheme. (It then follows that the number of bits needed to represent a share can not be smaller than the number of bits needed to represent the secret itself.)

Hint: Use the fact that t players have NO *information about the secret—no matter what t values they have received, any value of s is possible.*

D.8.2 Secret Sharing with cheaters

Dishonest trustees can prevent the reconstruction of the secret by contributing *bad* shares $\hat{s_i} \neq s_i$. Using the cryptographic tools you have seen so far in the class show how to prevent this denial of service attack.

D.8.3 Zero–Knowledge proof for discrete logarithms

Let p be a prime and g a generator modulo p. Given $y = g^x$ Alice claims she knows the discrete logarithm x of y. She wants to convince Bob of this fact but she does not want to reveal x to him. How can she do that? (Give a zero-knowledge protocol for this problem.)

D.8.4 Oblivious Transfer

An oblivious transfer protocol is a communication protocol between Alice and Bob. Alice runs it on input a value s. At the end of the protocol either Bob learns s or he has no information about it. Alice has no idea which event occurred.

An 1-2 oblivious transfer protocol is a communication protocol between Alice and Bob. Alice runs it on inout two values s_0 and s_1. Bob runs it on input a bit b. At the end of the protocol, Bob learns s_b but has no information about s_{1-b}. Alice has no information about b.

Show that given an oblivious transfer protocol as a black box, one can design a 1-2 oblivious transfer protocol.

D.8.5 Electronic Cash

Real-life cash has two main properties:

- It is *anonymous*: meaning when you use cash to buy something your identity is not revealed, compare with credit cards where your identity and spending habits are disclosed
- It is *transferable*: that is the vendor who receives cash from you can in turn use it to buy something else. He would not have this possibility if you had payed with a non-transferable check.

The electronic cash proposals we saw in class are all "non–transferable". that is the user gets a coin from the bank, spends it, and the vendor must return the coin to the bank in order to get credit. As such they really behave as anonymous non-transferable checks. In this problem we are going to modify such proposals in order to achieve transferability.

The proposal we saw in class can be abstracted as follows: we have three agents: the Bank , the User and the Vendor .

The Bank has a pair of keys (S, P). A signature with S is a *coin* worth a fixed amount (say \$1.). It is possible to make blind signatures, meaning the User gets a signature $S(m)$ on a message m, but the Bank gets no information about m.

Withdrawal protocol

1. The User chooses a message m
2. The Bank blindly signs m and withdraws \$1 from User 's account.
3. The User recovers $S(m)$. The coin is the pair $(m, S(m))$.

Payment Protocol

1. The User gives the coin $(m, S(m))$ to the Vendor .
2. The Vendor verifies the Bank signature and sends a random challenge c to the User .
3. The User replies with an answer r
4. the Vendor verifies that the answer is correct.

The challenge–response protocol is needed in order to detect double–spending. Indeed the system is constructed in such a way that if the User answers two different challenges on the same coin (meaning he's trying to spend the coin twice) his identity will be revealed to the Bank when the two coins return to the bank. This is why the whole history of the payment protocol must be presented to the Bank when the Vendor deposits the coin.

Deposit protocol

1. The Vendor sends $m, S(m), c, r$ to the Bank
2. The Bank verifies it and add \$1 to the Vendor 's account.
3. The Bank searches its database to see if the coin was deposited already and if it was reconstruct the identity of the double–spender User .

In order to make the whole scheme transferrable we give the bank a different pair of keys $(\mathcal{S}, \mathcal{P})$. It is still possible to make blind signatures with $\mathcal{S}$. However messages signed with $\mathcal{S}$ have no value. We will call them *pseudo-coins*. When people open an account with the Bank , they get a lot of these anonymous pseudo–coins by running the withdrawal protocol with $\mathcal{S}$ as the signature key.

Suppose now the Vendor received a payed coin $m, S(m), c, r$ and instead of depositing it wants to use it to buy something from OtherVendor . What she could do is the following:

Transfer protocol

1. The Vendor sends $m, S(m), c, r$ and a pseudo–coin $m', \mathcal{S}(m')$ to OtherVendor
2. OtherVendor verifies all signatures and the pair (c, r). Then sends a random challenge c' for the pseudo–coin.
3. Vendor replies with r'
4. OtherVendor checks the answer.

Notice however that Vendor can still double–spend the coin $m, S(m), c, r$ if she uses two different pseudo–coins to transfer it to two differnt people. Indeed since she will never answer two different challenges on the same pseudo–coin, her identity will never be revealed. The problem is that there is no link between the real coin and the pseudo-coin used during the transfer protocol. If we could force Vendor to use only one pseudo–coin for each real coin she wants to transfer then the problem would be solved.

Show how to achieve the above goal. You will need to modify both the payment and the transfer protocol.

Hint: If Vendor *wants to transfer the true coin she is receiving during the payment protocol, she must be forced* then *to create a link between the true coin and the pseudo–coin she will use for the transfer later. Notice that* Vendor *chooses c at random, maybe c can be chosen in some different way?*

D.8.6 Atomicity of withdrawal protocol

Recall the protocol that allows a User to withdraw a coin of \$1 from the Bank . Let $(n, 3)$ be the RSA public key of the Bank .

1. The User prepares 100 messages $m_1, \ldots, m_{100}$ which are all \$1 coins. The User *blinds* them, that is she chooses at random $r_1, \ldots, r_{100}$ and computes $w_i = r_i^3 m_i$. The User sends $w_1, \ldots, w_{100}$ to the Bank .
2. The Bank chooses at random 99 of the blindings and asks the User to open them. That the Bank chooses $i_1, \ldots, i_{99}$ and sends it to the User .
3. The User opens the required blindings by revealing $r_{i_1}, \ldots, r_{i_{99}}$.
4. The Bank checks that the blindings are constructed correctly and then finally signs the unopened blinding. W.l.o.g. assume this to be the first one. So the Bank signs w_1 by sending to the User $w_1^{\frac{1}{3}} = r_1 m_1^{\frac{1}{3}}$
5. The User divides this signature by r_1 and gets a signature on m_1 which is a valid coin.

Notice that the User has a probability of 1/100 to succesfully cheat.

Suppose now that the protocol is not atomic. That is the communication line may go down at the end of each step between the Bank and the User . What protocol should be followed for each step if the line goes down at the end of that step in order to prevent abuse or fraud by either party?

D.8.7 Blinding with ElGamal/DSS

In class we saw a way to blind messages for signatures using RSA. In this problem we ask you to construct blind signatures for a variation of the ElGamal signature scheme.

The ElGamal-like signature we will consider is as follows. Let p be a large prime, q a large prime dividing $p-1$, g an element of order q in Z_p^*, x the secret key of the Bank and $y = g^x$ the corresponding public key. Let H be a collision-free hash function.

When the Bank wants to sign a message m she computes

$$a = g^k \bmod p$$

for a random k and

$$c = H(m, a)$$

and finally

$$b = kc + xa \bmod q$$

The signature of the message m is $sig(m) = (a, b)$. Given the triple (m, a, b) the verification is performed by computing $c = H(m, a)$ and checking that

$$g^b = a^c y^a$$

So the withdrawal protocol could be as following:

1. The User tells the bank she wants a \$1 coin.
2. The Bank replies with 100 values $a_i = g^{k_i}$ for random k_i.
3. The User sends back $c_i = H(m_i, a_i)$ where m_i are all \$1 coins.
4. The Bank asks the user to open 99 of those.
5. The User reveals 99 of the m_i's.
6. The Bank replies with $b_i = k_i c_i + x a_i \bmod (p-1)$ for the unopened index i

However this is not anonymous since the Bank can recognize the User when the coin comes back. In order to make the protocol really anonymous, the User has to change the value of "challenge" c_i computed at step 3. This modification will allow him to compute a different signature on m_i on her own which will not be recognizable to the Bank when the coin comes back. During the protocol the Bank will check as usual that this modification has been performed correctly by asking the User to open 99 random blindings.