

Introduction to Modern Cryptography

Mihir Bellare¹ **Phillip Rogaway**²

November 24, 2001

¹ Department of Computer Science and Engineering, University of California at San Diego, La Jolla, CA 92093, USA. mihir@cs.ucsd.edu, <http://www-cse.ucsd.edu/users/mihir>

² Department of Computer Science, University of California at Davis, Davis, CA 95616, USA. rogaway@cs.ucdavis.edu, <http://www.cs.ucdavis.edu/~rogaway>

Preface

This is a set of class notes that we have been developing jointly for some years. We use them for the graduate cryptography courses that we teach at our respective institutions. Each time one of us teaches the class, he takes the token and updates the notes a bit. You might think that, within a three or four years, one would have a rather complete and refined set of notes in this way. But somehow it still hasn't worked out that way! You'll find that there are still lots of gaps, as well as plenty of "unharmonized" portions of the notes as they evolved in random ways. Well, eventually it will all get untangled and be a picture of elegance.

The viewpoint taken throughout these notes is to emphasize the *theory of cryptography as it can be applied to practice*. This is an approach that the two of us have pushed in our research, and it seems to be a pedagogically desirable approach as well.

We would like to thank the following students of past versions of our courses who have pointed out errors and made suggestions for changes: Andre Barroso, Keith Bell, Alexandra Boldyreva, Michael Burton, Sashka Davis, Alex Gantman, Bradley Huffaker, Chanathip Namprempre, Adriana Palacio, Fritz Schneider. We welcome further corrections, comments and suggestions.

Mihir Bellare
Phillip Bellare

San Diego, California USA
Davis, California USA

Contents

1	INTRODUCTION	7
1.1	Some sample cryptographic problems	7
1.2	What cryptography is about	17
1.3	Approaches to the study of cryptography	21
1.4	What background do I need?	31
1.5	Historical notes	31
1.6	Exercises and problems	31
2	BLOCK CIPHERS	33
2.1	What is a block cipher?	33
2.2	Data Encryption Standard	34
2.3	Advanced Encryption Standard	37
2.4	Cryptanalysis of 5-round AES	41
2.5	Some modes of operation	42
2.6	Key recovery attacks on block ciphers	43
2.7	Limitations of key-recovery based security	45
2.8	Exercises and Problems	46
3	PSEUDORANDOM FUNCTIONS	49
3.1	Function families	49
3.2	Random functions and permutations	50
3.3	Pseudorandom functions	52
3.4	Pseudorandom permutations	57
3.5	Usage of PRFs and PRPs	59
3.6	Example Attacks	62
3.7	Security against key recovery	65
3.8	The birthday attack	71
3.9	PRFs versus PRPs	73
3.10	One-way functions	74
3.11	Pseudorandom generators	80
3.12	Historical notes	80
3.13	Exercises and problems	80

4	SYMMETRIC ENCRYPTION	83
4.1	A framework for both encryption and message authentication	83
4.2	Some encryption schemes	85
4.3	Issues in security	88
4.4	Indistinguishability under chosen-plaintext attack	90
4.5	Examples of chosen-plaintext attacks	95
4.6	Security against plaintext recovery	98
4.7	Security of CTR encryption	102
4.8	Security of CBC encryption	116
4.9	Other characterizations of IND-CPA security	116
4.10	Indistinguishability under chosen-ciphertext attack	117
4.11	Example chosen-ciphertext attacks	119
4.12	Historical Notes	123
4.13	Exercises and Problems	123
5	HASH FUNCTIONS	125
5.1	Notions of security for hash-function families	125
5.2	The hash function SHA-1	125
5.3	The Merkle-Damgård result	125
5.4	Collision-resistant hash functions are one-way	125
5.5	UOWHFs	125
5.6	Universal hash functions	125
5.7	Exercises and Problems	125
6	MESSAGE AUTHENTICATION	127
6.1	The Setting	127
6.2	Encryption does not provide authenticity	130
6.3	Syntax of message-authentication schemes	131
6.4	Example message-authentication schemes	134
6.5	Towards a Definition of Security	135
6.6	Definition of security	138
6.7	Example schemes	140
6.8	The PRF-as-a-MAC Paradigm	144
6.9	Making a PRF from a PRF and a Universal Hash Function	145
6.10	An XOR Scheme	145
6.11	The EMAC Construction	145
6.12	The HMAC Construction	149
6.13	The UMAC Construction	149
6.14	Problems	155
6.15	References and Related Work	156
7	AUTHENTICATED ENCRYPTION	157

8	NUMBER-THEORETIC BACKGROUND	159
8.1	The basic groups	159
8.2	Algorithms	161
8.3	Cyclic groups and generators	167
8.4	Squares and non-squares	172
8.5	Groups of prime order	177
8.6	Historical Notes	179
8.7	Exercises and Problems	179
9	ASYMMETRIC ENCRYPTION	181
10	DIGITAL SIGNATURES	183
11	KEY DISTRIBUTION	185
12	THE ASYMPTOTIC APPROACH	187
13	INTERACTIVE PROOFS AND ZERO KNOWLEDGE	189
14	MORE PROTOCOLS	191
I	Appendices	193
A	THE BIRTHDAY PROBLEM	195
B	PROBABILITY THEORY	199

Chapter 1

INTRODUCTION

Modern cryptography is a remarkable field. It deals with very human concerns—issues of privacy, authenticity, and trust—but it does so in a way that is concrete and scientific. Making a science out of something as fuzzy as privacy or authenticity might seem an impossible thing to do. But believe it! This course is your invitation to this fascinating young field.

The word “cryptography” comes from the Latin *crypt*, meaning secret, and *graphia*, meaning writing. So “cryptography” is literally “secret writing”: the study of how to obscure what you write so as to render it unintelligible to those who should not read it. Nowadays cryptography entails a lot more than finding good ways for keeping your writings secret. That problem remains one of cryptography’s central problems, but many more problems have been added to the brew.

Despite the scope of cryptography having broadened, much of the flavor of this field is unchanged since the very early days: it’s still a game of clever designs, sneaky attacks, and mathematical slight of hand. The thing that has changed is that the art of cryptography has now been supplemented with a legitimate science. In this course we shall focus on that science.

Be forewarned: cryptography is a slippery subject. Surprisingly often that which seems meaningful turns out to be meaningless, that which seems true turns out to be false, and that what seems impossible turns out to be doable. So have fun—but retain a healthy skepticism, and always watch your step.

1.1 Some sample cryptographic problems

Let us begin by looking at a few of the problems that cryptographers have considered. We’ll describe these problems quite informally, but we’ll be returning to them later in our studies, when they’ll get a much more thorough treatment.

1.1.1 Message Privacy

Imitating the ideal channel Let's introduce the first two members of our cast of characters: our sender, S , and our receiver, R . The sender and receiver want to communicate with each other, say over a network.

(Sometimes people call these characters Alice, A , and Bob, B . Alice and Bob figure in many works on cryptography. But the authors can never remember what is the role for Alice and what is the role for Bob, and we're going to want the letter A for someone else, anyway.)

What is the *ideal* channel over which the sender and receiver could conceivably communicate? Imagine they are provided with a dedicated, untappable, impenetrable lead pipe into which the sender can whisper a message and the receiver will hear it. Nobody else can look inside the pipe or change what's there. This lead pipe provides the perfect medium, available only to the sender and receiver, as though they were alone in the world. See Figure 1.1.

Figure 1.1: Several cryptographic goals aim to imitate some aspect of an ideal channel connecting a sender S to a receiver R .

Unfortunately, in real life, there are no ideal channels connecting the pairs of parties that might like to communicate. Usually all we have is a public network like the Internet.

Several cryptographic goals concern themselves with imitating, in some respect, an ideal channel between the sender and receiver. In these problems the parties are communicating over an insecure channel but they want to imagine that they have a perfect lead pipe between them. Cryptography is used to create the *illusion* that their channel is secure. The parties should be assured of the kinds of properties that they would expect of a secure channel.

Protocols and adversaries What mechanisms are available to help us to imitate the lead-pipe world? All we are allowed to do is to supply the sender and receiver with a *protocol*. A protocol is just like a program, except that it is a distributed program. It tells the sender and receiver what to do.

How do we start building protocols? We first try to isolate the *threats* and the *goals*. Once we have a good idea about these, we can try to find protocol solutions.

At this point we should introduce the third member of our cast. This is our

adversary, denoted A . An adversary is the source of all possible threats. Cryptographic protocols attempt to surmount the influence of the adversary.

In cryptography we must focus on the adversary. What can she do, and what can't she do? It is important straight away to give the adversary full membership and respect in the cast of characters. She is in many ways the central character.

If you think about it, there are actually several different things that the adversary might be trying to do in attacking our protocol-approximation of a lead pipe. The first thing we are concerned with is that the adversary might want to understand the content of the messages sent from the sender to the receiver. This is an attack on the parties' privacy.

Encryption In order to protect the privacy of transmissions we use a tool called **encryption**. The sender *encrypts* his message M and sends it. The message M is called **plaintext**. What the sender creates by encrypting M is called a **ciphertext**, C . The receiver, on receipt of ciphertext C , *decrypts* it. If all goes well, the receiver should now have recovered the same plaintext, M , that the sender sent out.

You might hope for encryption to emulate all the properties of a lead pipe, but this is an impossibly large task. You could try to emulate all the properties that have to do with privacy, but even this is too much to hope for. First, we don't normally expect for encryption to hide the *existence* of a message. This is a potentially important piece of information to the adversary, but it is too often infeasible to try to hide this. Nor is encryption normally intended to hide the *length* of a message. The length of the plaintext is another potentially interesting piece of information to the adversary, but we won't usually try to hide this, either. If we know a maximal length for message we could use padding to hide the length of messages, but this would typically entail a large loss of efficiency. Nor does encryption normally aim to hide who is sending messages to whom, or which messages are associated to which senders.

Keys It is not hard to convince yourself that in order to communicate securely, there must be something that the receiver knows, or can do, that the adversary does not know, or can not do. There has to be some "asymmetry" between the situation the the receiver finds himself in, and the adversary finds herself in. In practice, the simplest and also most common setting is that the sender and receiver share a **key** that the adversary does not know. This is called the *symmetric* trust model. Encrypting in the symmetric trust model is called **symmetric encryption** or **shared-key encryption**.

The shared key is usually a uniformly distributed random string having some number of bits, k . Recall that a *string* is just a sequence of bits. (For language-theoretic background, see Figure 1.2.) The sender and receiver must somehow use the key K to overcome the presence of the adversary.

Notice how randomness enters the picture. The key is random. Randomness is a central and unavoidable element of cryptography. Everything is about probabilities.

We will sometimes use words from the theory of “formal languages.” Here is the vocabulary you should know.

An **alphabet** is a finite nonempty set. We usually use the Greek letter Σ to denote an alphabet. The elements in an alphabet are called **characters**. So, for example, $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ is an alphabet having ten characters, and $\Sigma = \{0, 1\}$ is an alphabet, called the **binary alphabet**, which has two characters. We’ll assume the binary alphabet. A **string** is finite sequence of characters. The number of characters in a string is called its **length**, and the length of a string X is denoted $|X|$. So $X = 1011$ is a string of length four, $Y = \text{cryptology}$ is a string of length 12. The string of length zero is called the **emptystring** and is denoted ε . If X and Y are strings then the concatenation of X and Y , denoted $X\|Y$, is the characters of X followed by the characters of Y . So, for example, $1011\|0 = 10110$. The i -th character of a string X , where $1 \leq i \leq |X|$, is denoted $X[i]$, so that $X = X[1]\|X[2]\|\dots\|X[|X|]$. If a is a character and $i \geq 0$ is a number then a^i is the string consisting of the character a repeated i times. It is understood that $a^0 = \varepsilon$ for any character a . So, for example, $0^3 = 000$ and 1^n is how you’d write the number n in unary notation. We can encode almost anything into a string. Usually the details of how one does this are irrelevant, and so we use the notation $\langle \text{something} \rangle$ for any fixed, natural way to encode *something* as a string. For example, if n is a number and X is a string then $Y = \langle n, X \rangle$ is some string which encodes n and X . It is easy to go from n and X to $Y = \langle n, X \rangle$, and it is also easy to go from $Y = \langle n, X \rangle$ back to n and X . A **language** is a set of strings, all of the strings being drawn from the same alphabet, Σ . If Σ is an alphabet then Σ^* denotes the set of all strings whose characters are drawn from Σ . For example, $\{0, 1\}^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, \dots\}$.

Figure 1.2: Elementary notation from formal-language theory.

Get used to it!

The key is kept securely on the parties’ machines. It is an assumption that we must make that the adversary cannot penetrate these machines and recover the key.

A picture for symmetric encryption can be found in Figure 1.3. The sender sends the receiver a plaintext M by first computing $C \leftarrow \mathcal{E}_K(M)$. The *encryption function* \mathcal{E} may be randomized, or it might keep some state around. The receiver recovers the plaintext M from the ciphertext C by *decrypting* the ciphertext, setting $M \leftarrow \mathcal{D}_K(C)$.

We warn you that a picture like Figure 1.3 is actually a little misleading about what the adversary can and can not do. In particular, it might seem to suggest that the adversary is just a passive eavesdropper, quietly listening to the communications between the sender and receiver. In fact, this might not be the case at all. We will consider adversaries that are much more powerful than that.

Figure 1.3: Symmetric encryption. The sender and the receiver share a secret key, K . The adversary lacks this key. The message M is the plaintext; the message C is the ciphertext.

Encryption with a one-time-pad Now let's give an example of a protocol that encrypts. Here is how the sender and receiver encrypt in any spy novel. Let $K = K[1] \cdots K[k]$ denote the shared key, which is a random sequence of k bits. Think of k as some big number, like a million. Let $M = M[1] \cdots M[m]$ denote the plaintext message that the sender wants to send, also divided up into bits. Assume that $m \leq k$ (that is, the key is at least as long as the plaintext).

What the sender does is to compute $C'[i] = K[i] \oplus M[i]$ for each $i = 1, \dots, m$. The symbol \oplus denotes the exclusive-or (XOR) operation: $0 \oplus 0 = 1 \oplus 1 = 0$, while $0 \oplus 1 = 1 \oplus 0 = 1$. The string C' is the main part of the ciphertext which the sender sends out. The receiver receives $C' = C'[1] \cdots C'[m]$ and can recover M via $M[i] = C'[i] \oplus K[i]$ for all $1 \leq i \leq m$. This is possible for the receiver because he too knows the key K .

When the sender wants to encrypt another message she has to use new key bits. That is, she keeps track of where she is in the key, via a counter, and goes on from there. Key bits are never re-used. That's why this is called a *one-time pad*: each key bit is used at most once. You cannot encrypt more data than you have key bits. An indication of where the sender is in the key should be included in the ciphertext. See Figure 1.4.

What we have just done is specify a protocol. This is the sequence of instructions for the parties to execute. In the case of symmetric encryption, a protocol needs to specify three things: how to encrypt, how to decrypt, and how to generate the shared key. Formally, an encryption protocol Π is a three-tuple of algorithms, $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$. Notice that the adversary is not involved in the protocol. We never try to tell the adversary what to do. That is up to her.

We'll be discussing the security of one-time-pad encryption later in this chapter.

Public-key encryption The shared key K between the sender and the receiver is not the only way to create the information asymmetry that we need between the receiver and the adversary. In *asymmetric encryption*, also called *public-*

<pre> Algorithm \mathcal{K} $K \xleftarrow{R} \{0, 1\}^k$ return K </pre>	<pre> Algorithm $\mathcal{E}_K(M)$ static $ctr \leftarrow 0$ $m \leftarrow M$ if $ctr + m > k$ then return error for $i \leftarrow 1$ to m do $C'[i] \leftarrow K[ctr + i] \oplus M[i]$ $ctr \leftarrow ctr + m$ return $\langle ctr, C' \rangle$ </pre>	<pre> Algorithm $\mathcal{D}_K(C)$ $\langle ctr, C' \rangle \leftarrow C$ $m \leftarrow C'$ if $ctr + m > K$ then return error for $i \leftarrow 1$ to m do $M[i] = K[ctr + i] \oplus C'[i]$ return M </pre>
--	--	---

Figure 1.4: Encryption with a one-time pad. The first algorithm generates the key K , the second encrypts plaintext M , and the last decrypts ciphertext C .

key encryption, the receiver R possesses a *pair* of keys—a *public key*, pk_R , and a *secret key*, sk_R . The receiver’s public key is made publicly known and bound to his identity. For example, the receiver’s public key might be published in a phone book. When the sender wants to send a secret message M to the receiver, she looks up the receiver’s public key in the phone book and computes $y \leftarrow \mathcal{E}_{pk_R}(M)$. When the receiver receives a ciphertext C he computes $M \leftarrow \mathcal{D}_{sk_R}(C)$.

The **trust model** specifies who, initially, has what keys. We have just described two different trust models for achieving the same basic aim: the symmetric (or shared-key) trust model and the asymmetric (or public-key) trust model.

The idea of public-key cryptography, and the fact that we can actually realize this goal, is a remarkable idea. Think about it! You’ve never met the receiver before. But you can send him a secret message by looking up some information in a phone book and then using this information to help you garble up the message you want to send. The intended receiver will be able to understand the content of your message, but nobody else will. The idea of public-key cryptography is due to Whitfield Diffie and Martin Hellman. Diffie was Hellman’s graduate student at Stanford. The idea was published in 1976.

For a picture of encryption in the public-key setting, see Figure 1.5.

1.1.2 Message Authenticity

Now we’re going to try to imitate another aspect of the lead pipe between the sender and the receiver: it is the fact that only the sender can speak into her end of the pipe, and what will emerge at the receiver’s end is exactly what the sender said. That is, in the message-authentication problem the receiver gets some message which is claimed to have originated with a particular sender. The channel on which this message flows is insecure. Thus the receiver R wants to distinguish the case in which the message really did originate with the claimed sender S from the case in

Figure 1.5: Asymmetric encryption. The receiver R has a public key, pk_R , which the sender knows belongs to R . The receiver also has a corresponding secret key, sk_R .

which the message originated with some imposter, A .

Once again, information asymmetry is needed. The symmetric and asymmetric trust models described above are equally applicable here.

The usual tool for solving the message-authentication problem in the symmetric setting is a *message authentication code* (MAC). Here the sender and receiver share a secret key, K , and when the sender wants to send a message M to the receiver she attaches to it a few bits, σ , which is called the *tag* for the message. The tag is computed using M and K . The MAC computation might be probabilistic or use state, just as with encryption. Or it may well be deterministic. Bob, on receipt of M and σ , uses the key K to check if the tag is OK. If so, he accepts M as authentic; otherwise, he regards M as a forgery. An appropriate reaction might range from ignoring the bogus message to tearing down the connection to alerting a responsible party about the possible mischief. See Figure 1.6. The picture is again slightly misleading in terms of what the adversary A might or might not be able to do, but it gets the point across.

The tool for solving the message-authentication problem in the asymmetric setting is a *digital signature*. Here the sender has a public key pk_S and a corresponding secret key sk_S . Everyone (even the adversary) knows the key pk_S and that it belongs to party S . When the sender wants to send a message M she attaches to it some extra bits, σ , which is called a *signature* for the message and is computed as a function of M and sk_S . The receiver, on receipt of M and σ , checks if it is OK using the public key of the sender, pk_S . If it is fine, the receiver regards M as authentic; otherwise, he regards M as an attempted forgery. A picture is given in Figure 1.7.

One difference between a MAC and a digital signature concerns what is called **non-repudiation**. With a MAC anyone who can verify a tagged message can also

Figure 1.6: A message authentication code. The tag σ accompanies the message M . The receiver R uses it to decide if the message is really did originate with the sender S with whom he shares the key K .

Figure 1.7: A digital signature scheme. The signature σ accompanies the message M . The receiver R uses it to decide if the message is really did originate with the sender S with has public key pk_S .

produce one, and so a tagged message would seem to be of little use in a court of law. But with a digitally-signed message the *only* party who should be able to produce a a message that verifies under public key pk_S is the party S herself. Thus if the signature scheme is good party S can not just maintain that the receiver, or the one presenting the evidence, concocted it. If signature σ authenticates M with respect to public key pk_S , then it is only S that should have been able to devise σ . The sender can not refute that. Probably the sender S will have to claim that the

key sk_S was stolen from her. Perhaps this, if true, might still be construed as the sender’s fault.

To summarize, there are two common aims concerned with sending a message so as to an ideal channel: achieving message privacy and achieving message authenticity. There are two main trust models in which we are interested in achieving these goals: the symmetric trust model and the asymmetric trust model. The tools used to achieve these four goals are named as follows:

	symmetric trust model	asymmetric trust model
message privacy	symmetric (private-key) encryption	asymmetric (public-key) encryption
message authenticity	message authentication code (MAC)	digital signature scheme

1.1.3 Pseudorandom Number Generation

Computers are quite deterministic. But for lots of applications, “random numbers” are useful. These applications involve simulation, efficient algorithms, and cryptography itself. We’ve already seen an example protocol that required random bits.

How can a completely deterministic machine generate “random” numbers? Well, it can’t. But a machine can do the next best thing: it can *stretch* a little bit of randomness into a lot of “pseudorandomness.”

Suppose we wire to our computer a Geiger counter that generates a “random” bit every second. We run our computer for a little while and now it has 200 “random” bits. We won’t worry about the “philosophical” question as to whether these bits are random in any *real* sense. We’ll simply assume that these bits are completely unpredictable to anything “beyond” the computer which has gathered this data—mathematically, we’ll treat these bits as random.

A *pseudorandom generator* (PRG) stretches a short (eg., 200-bit) “truly random” string into a much longer (a million bits, say) string which “looks” random. This is another core problem of cryptography.

1.1.4 Authenticated Key Exchange

Suppose that Alice would like to remotely logon to her computer, which we’ll call Alice’s “host” machine. Alice has a secret password, a , while the host has some function of this password, $f(a)$. Alice communicates with the host over the Internet. What should Alice do in order to identify herself to the host and establish a secure connection with it?

One possibility would be to flow a to the host. This would identify Alice to the host, at least initially, but anyone who was listening in on the conversation would now know Alice’s password, and thus be able to logon as though they were she. Clearly that is not a good solution.

What we would really like is to arrange that Alice and her host can have a conversation—a *secure session*—such that, throughout the session, (a) The host is convinced that it is speaking to Alice; (b) Alice herself is convinced that she is speaking to the host; (c) Nobody else has any idea about the content of the information that flows within this session.

The usual way to establish a secure session is with an *authenticated key exchange*. In our example, Alice and the host will engage in a conversation at the end of which these two parties—and only these two parties—will share a secret *session key* σ . With this session key distributed, Alice and the host can use it to encrypt and to authenticate the traffic that flows between them, thus setting up a secure session.

1.1.5 Telephone Coin Flipping

Alice and Bob want to decide which of them has to show up to their AI class. They take turns going to this detested class, you see, but it so happens that there will be an odd number of lectures.

Alice calls Bob on the telephone and offers a simple solution. “Bob,” she says, “I’ve got a penny in my pocket. I’m going to toss it in the air right now. You call *heads* or *tails*. If you get it right, I’ll go to AI class and take notes for us. If you get it wrong, you’ll have to go.”

Bob is not as bright as Alice, but something troubles him about this arrangement.

The *telephone-coin-flip* problem is to come up with a protocol so that, to the maximal extent possible, neither Alice nor Bob can cheat the other and, at the same time, each of them learn the outcome of a fair coin toss.

Here is a solution—sort of. Alice puts a random bit α inside an envelope and sends it to Bob. Bob announces a random bit β . Now Alice opens the envelope for Bob to see. The shared bit is defined as $\alpha \oplus \beta$. See Figure 1.8

Figure 1.8: Envelope solution to the telephone-coin-flipping problem.

To do this over the telephone we need some sort of “electronic envelope” (in cryptography, this called a *commitment scheme*). Alice can put a value in the

envelope and Bob can't see what the envelope contains. Later, Alice can open the envelope so that Bob can see what the envelope contains. Alice can't change her mind about an envelope's contents—it can only be opened up in one way.

Here is a simple technique to implement an electronic envelope. To put a “0” inside an envelope Alice chooses two random 500-bit primes p and q subject to the constraints that $p < q$ and $p \equiv 1 \pmod{4}$ and $q \equiv 3 \pmod{4}$. The product of p and q , say $N = pq$, is the commitment to zero; that is what Alice would send to commit to 0. To put a “1” inside an envelope Alice chooses two random 500-bit primes p and q subject to the constraints that $p < q$ and $p \equiv 3 \pmod{4}$ and $q \equiv 1 \pmod{4}$. The product of these, $N = pq$, is the commitment to 1. Poor Bob, seeing N , would like to figure out if the smaller of its two prime factors is congruent to 1 or to 3 modulo 4. We have no idea how to make that determination short of factoring N —and we don't know how to factor 1000 digit numbers which are the product of random 500-digit primes. Our best algorithms would, take way too long to run. When Alice wants to decommit (open the envelope) N she announces p and q . Bob verifies that they are prime (this is easy to do) and multiply to N , and then he looks to see if the smaller factor is congruent to 1 or to 3 modulo 4.

1.2 What cryptography is about

Protocols and adversaries Let us now move away from the particular examples we have given and ask what, in general, is cryptography about? Briefly, cryptography is about constructing and analyzing *protocols* which overcome the influence of *adversaries*. One way to know that you've left the world of cryptography is that there is no protocol or no adversary anywhere in sight. In the last section we gave examples of several different protocol problems, and a couple of different protocols.

Suppose that you are trying to solve some cryptographic problem. The problem will usually involve some number of *parties*. Us cryptographers often like to anthropomorphize our parties, giving them names like “Alice” and “Bob” and referring to them as though they are actual people. We do this because it's convenient and fun. But you shouldn't think that it means that the parties are *really* human beings. They might be—but they could be lots of other things, too. Like a cell phone, a computer, a processes running on a computer, an institution, or maybe a little gadget sitting on the top of your television set.

We usually think of the parties as the “good guys,” and we want to help them accomplish their goal. We do this by making a protocol for the parties to use.

A protocol tells each party how to behave. A protocol is essentially a program, but it's a distributed program. Here are some features of protocols for you to understand.

- A protocol instructs the parties what to do. It doesn't tell the adversary what to do. That is up to her.
- A protocol can be *probabilistic*. This means that it can make random choices.

To formalize this we usually assume that the model of computation that allows a party to specify a number $n \geq 2$ and then obtain a random value $i \xleftarrow{R} \{0, 1, \dots, n-1\}$. This notation means that i is a random value from the indicated set, all values being equally likely.

- A party might run a protocol for a while, and then pause, waiting to hear from another party. Then the party resumes, running the protocol for another phase. This can go on and on. Whether or not this happens depends on the protocol problem that we are trying to solve.
- A protocol can be *stateful*. This means that when a party finishes what he is doing he can retain some information for the next time that he is active. When that party runs again he will remember the state that he was last in. So, for example, you could have a party that knows “this is the first time I’ve been run,” “this is the second time I’ve been run,” and so on.
- Sometimes we will allow for parties to access an *oracle*. An oracle is a “magic box” to which the parties can ask questions. The protocol will specify, in any given case, how this box is supposed to answer the party’s questions. This is just an “intermediate step” in designing protocols. In a “finished” protocol there aren’t any oracles.

When we formalize protocols, they are usually tuples of algorithms. But the actual formalization will vary from problem to problem. For example, a protocol for symmetric encryption isn’t the same “type” of thing as a protocol for a telephone coin flip.

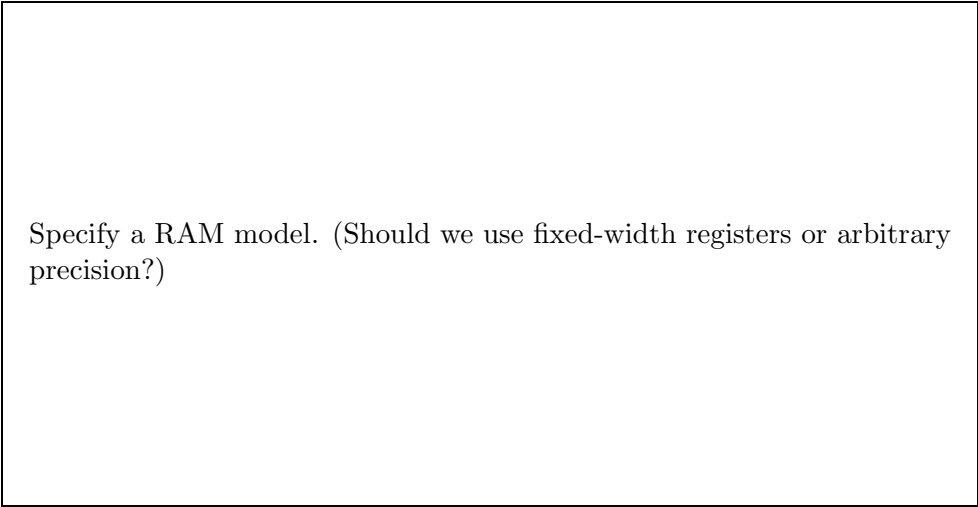
Another word for a protocol is a *scheme*. We’ll use the two words interchangeably. So an encryption scheme is a protocol for encryption, and a message-authentication scheme is a protocol for message authentication. For us, a function, computed by a deterministic, sequential algorithm, is also a protocol. It’s a particularly simple kind of protocol.

How can we devise and analyze protocols? The first step is to try to understand the *threats* and the *goals* for our particular problem. Once we have a good idea about these, we can try to find a protocol solution.

The *adversary* is the agent that embodies the “source” of the threat. Adversaries aim to defeat our protocol’s goals. Protocols, in turn, are designed to surmount the behavior of adversaries. It is a game—a question of who is more clever, protocol designer or adversary.

The adversary is usually what we focus on. In rigorous formalizations of cryptographic problems, the parties may actually vanish, being “absorbed” into the formalization. But the adversary will never vanish. She will be at center stage.

Cryptography is largely about thinking about the adversary. What can she do, and what can’t she do? What is she trying to accomplish? We have to answer these questions before we can get very far.



Specify a RAM model. (Should we use fixed-width registers or arbitrary precision?)

Figure 1.9: The RAM model, with an oracle. An adversary is a program written in this model of computation. Details of the model are not important, but one has to fix *some* model of computation.

Just as we warned that one shouldn't literally regard our parties as people, so too for the adversary. The adversary *might* represent an actual person, but it might just as well be an automated attack program, a competitor's company, a criminal organization, a government institution, one or more of the protocol's legitimate parties, a group of friendly hackers, or merely some unlucky circumstances conspiring together, not controlled by any intelligence at all.

By imagining a powerful adversary we take a pessimistic view about what might go wrong. We aim to succeed even if someone is out to get us. Maybe nobody is out to get us. In that case, we should at least be achieving high *reliability*. After all, if a powerful adversary can't succeed in disrupting our endeavors, then neither will noisy lines, transmission errors due to software bugs, unlucky message delivery times, careless programmers sending improperly formatted messages, and so forth.

When we formalize adversaries they will be "random access machines (RAMs) with access to an oracle." See Figure 1.9 for a description of this model of computation.

Cryptography and computer security Good protocols are an essential tool for making secure computing systems. Badly designed protocols are easily exploited to break into computer systems, to eavesdrop on phone calls, to steal services, and so forth. Good protocol design is also hard. It is easy to under-estimate the task and quickly come up with *ad hoc* protocols that later turn out to be wrong. In industry, the necessary time and expertise for proper protocol design is typically under-estimated, often at future cost. It takes knowledge, effort and ingenuity to

do the job right.

Security has many facets. For a system to be secure, many factors must combine. For example, it should not be possible for hackers to exploit bugs, break into your system, and use your account. They shouldn't be able to buy off your system administrator. They shouldn't be able to steal your back-up tapes. These things lie in the realm of system security.

The cryptographic protocol is just one piece of the puzzle. If it is poorly designed, the attacker will exploit that. For example, suppose the protocol transmits your password in the clear (that is, in a way that anyone watching can understand what it is). That's a protocol problem, not a system problem. And it will certainly be exploited.

The security of the system is only as strong as its weakest link. This is a big part of the difficulty of building a secure system. To get security we need to address all the problems: how do we secure our machines against intruders, how do we administer our machines to maintain security, how do we design good protocols, and so on. All of these problems are important, but we will not address all of these problems here. This course is about the design of secure protocols. We usually have to assume that the rest of the system is competent at doing its job.

The rules of the game Cryptography has rules. The first rule is that we may only try to overcome the adversary by means of protocols. We aren't allowed to overcome the adversary by intimidating her, arresting her, or putting poison in her coffee. These methods might be effective, but they are not cryptography.

(Actually, most cryptographers have quite friendly feelings towards our adversaries, and we'd never want to cause one harm. Without an adversary, at least a hypothetical one, we'd have nothing left to do. We'd have to seek employment as mathematicians, where jobs are scarce and salaries are low. No, better to have plenty of adversaries, and to stay on good terms with them.)

Another rule that most cryptographers insist on is to make the protocols *public*. That which must be secret should be embodied in **keys**. The keys specify data, not algorithms. Why do we insist that our protocols be public? There are several reasons. A resourceful adversary will likely find out what the protocol is anyway, since it usually has to be embodied in many programs or machines; trying to hide the protocol description is likely to be costly or infeasible. More than that, the attempt to hide the protocol makes one wonder if you've achieved security or just obfuscation. Peer review and academic work can not progress in the absence of known mechanisms, so keeping cryptographic methods secret is often seen as anti-intellectual and a sign that one's work will not hold up to serious scrutiny.

Government organizations which deal in cryptography often do not make their mechanisms public. For them, learning the cryptographic mechanism is one more hoop that the adversary must jump through. Why give anything away? Some organizations may have other reasons for not wanting mechanisms to be public, like a fear of disseminating cryptographic know-how, or a fear that the organization's

abilities (or inabilities) will become better known.

1.3 Approaches to the study of cryptography

Phases in cryptography’s development The history of cryptography can roughly be divided into three stages. In the first, early stage, algorithms had to be implementable with paper and ink. Julius Caesar used cryptograms. His and other early schemes often took the form of substitution ciphers. If $\mathcal{A} = \{A, B, \dots, Z\}$ is the alphabet (Caesar of course used the Roman one!), the simplest substitution cipher is simply a permutation $f: \mathcal{A} \rightarrow \mathcal{A}$, associating with each “plaintext” letter x its “ciphertext” letter $f(x)$. (Permutation means it is one-to-one and onto, that is, bijective.) The mapping f is known to receiver and sender, but, at least a priori, not to an adversary. To send a message M , view it as a sequence of letters, $M = M[1] \dots M[m]$. The sender computes $C[i] = f(M[i])$ for $i = 1, \dots, m$ and transmits $C = C[1] \dots C[m]$. The receiver, knowing f , also knows f^{-1} , and can decode. The adversary, not knowing the association f , but seeing only C , may be baffled at first. But once enough words have been transmitted, the code is soon broken, because we can make guesses based on repetitions of letters and knowledge of frequencies of letters in words in the English language. The system can be strengthened in various ways, but none too effective.

The second age of cryptography was that of cryptographic engines. This is associated to the period of the World War II, and the most famous crypto engine was the German Enigma machine. How its codes were broken is a fascinating story.

The last stage is modern cryptography. Its central feature is the reliance on mathematics and electronic computers. Mathematical tools are used to design protocols and computers are used implement them. It is during this most recent stage that cryptography becomes much more a science.

We can characterize much of the work that has been going on in cryptography in a couple of different dimensions. The first distinction is between *cryptanalysis-driven* design and *proof-driven* design. The second distinction is between *information-theoretic* cryptography and *complexity-theoretic* cryptography. We would like to take up these two dimensions.

Cryptanalysis-driven design Traditionally, cryptographic mechanisms have been designed by focusing on concrete attacks and how to defeat them. The approach has worked something like this.

- (1) A cryptographic goal is recognized.
- (2) A solution is offered.
- (3) One searches for an attack on the proposed solution.
- (4) When one is found, if it is deemed damaging or indicative of a potential weakness, you go back to Step 2 and try to come up with a better solution. The process then continues.

The third step is called *cryptanalysis*. In the classical approach to design, cryptanalysis was an essential component of constructing any new design.

Sometimes one finds protocol problem in the form of subtle mathematical relationships which allow one to subvert the protocol's aims. Sometimes, instead, one "jumps out of the system," showing that some essential cryptographic issue was overlooked in the design, application, or implementation of the cryptography.

Some people like to reserve the word *cryptography* to refer to the making of cryptographic mechanisms, *cryptanalysis* to refer to the attacking of cryptographic mechanisms, and *cryptology* to refer to union. Under this usage, we've been saying "cryptography" in many contexts where "cryptology" would be more accurate. Most cryptographers don't observe this distinction between the words "cryptography" and "cryptology," so neither will we.

There are some difficulties with the approach of cryptanalysis-drive design. The obvious problem is that one never knows if things are right, nor when one is finished! The process should iterate until one feels "confident" that the solution is adequate. But one has to accept that design errors might come to light at any time. If one is making a commercial product one must eventually say that enough is enough, ship the product, and hope for the best. With luck, no damaging attacks will subsequently emerge. But sometimes they do, and when this happens the company that owns the product may find it difficult or impossible to effectively fix the fielded solution. They might try to keep secret that there is an good attack, but it is not easy to keep secret such a thing. See Figure 1.10.

Figure 1.10: The classical-cryptography approach.

Doing cryptanalysis well takes great cleverness, and it is not clear that insightful cryptanalysis is a skill that can be effectively taught. Sure, one can study the most famous attacks—but will they really allow you to produce a new, equally insightful one? Great cleverness and great mathematical prowess seem to be the requisite skills, not any specific piece of knowledge. Maybe you have heard of Don Coppersmith or Adi Shamir. These are two of the masters of this field.

Sadly, it is hard to base a science on an area where significant assurance is engendered by knowing that Don thought seriously about the mechanism for some time, and couldn't find an attack. We need to pursue things differently.

Shannon security for symmetric encryption The “systematic” approach to cryptography, where proofs and definitions play a visible role, begins in the work of Claude Shannon. Shannon was not only the father of information theory, but he might also be said to be the father of the modern-era of cryptography.

Let's return to the problem of symmetric encryption and our particular protocol for doing this, which was to use a one-time-pad. Security, we have said, means defeating an adversary, so we have to specify what is the adversary wants to do.

One might think that the adversary's goal is something like this: given a ciphertext C , and not knowing key K , try to figure out the plaintext M . Here is one attempt to make precise that the adversary can't do this if we encrypt using a one-time pad: “It is impossible for the adversary, given C , to write down M .” Is this statement true? No. The adversary might well guess M , by outputting a random sequence of n bits, where $n = |C|$. She would be right with probability 2^{-n} . Not bad if, say $n = 1$. Does that make the scheme bad? Of course not. But it tells us that security is a probabilistic thing.

Another issue is a priori knowledge. Even before M is transmitted the adversary might know something about it. For example, the adversary might have reason to believe that M is either 0^n or 1^n . Why? Maybe because the adversary knows what the sender and receiver are talking about. If we're trying to make a general definition, we can't assume that the adversary *doesn't* know what the parties are talking about. If the adversary knows that the message is either 0^n or 1^n then the adversary can get the message right with probability $1/2$. How is this factored in?

All this tells us that we need a proper definition of security, some formal way of saying what it means for the scheme to be secure. We present the idea of Shannon.

Let $\mathcal{M}: \{0, 1\}^n \rightarrow [0, 1]$ be a probability distribution on the set of n -bit messages. That is, assume Alice chooses M with probability $\mathcal{M}(M)$. This distribution is known to everyone, including the adversary. Thus, before C is transmitted, all the adversary knows is that any particular message M has probability $\mathcal{M}(M)$ of being transmitted.

We want to capture the constraint that the adversary's information about the message does not increase after seeing the ciphertext. We have fixed some encryption scheme $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ in mind. For any string C let $P_C(M)$ denote the a posteriori

probability of M given ciphertext C , namely

$$P_{\mathcal{M}}(C, M) = \Pr[\text{Message was } M \mid \text{Ciphertext was } C] .$$

Here the probability is over the choice of key K and the choice of M from \mathcal{M} . Note it is a conditional probability, namely the probability that M was the message given that a particular ciphertext C has been seen.

Definition 1.1 Encryption scheme $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ is *Shannon secure* if for every distribution \mathcal{M} it is the case that for every ciphertext C which occurs with nonzero probability, and message M , we have $P_{\mathcal{M}}(C, M) = \mathcal{M}(M)$. ■

The way to interpret it is that after having seen C , let the adversary take her best guess as to what M was. The probability that she is right is not more than the probability that she would have been right had the sender simply chosen a message, transmitted nothing at all, and asked the adversary to guess this message.

As long as you don't end up with *more* information about the message after seeing C than you had before, then the encryption is secure.

We claim one-time-pad encryption has the above property, and propose to prove it. You might want to brush up on your probability before you tackle this: Bayes' rule, conditioning, and so on. We will use such tools many times again.

Proposition 1.2 One-time-pad encryption is Shannon secure.

Proof: We have to show that Definition 1.1 is met. Bayes' rule tells us that

$$\Pr[M \mid C] = \Pr[C \mid M] \cdot \frac{\Pr[M]}{\Pr[C]} . \quad (1.1)$$

Let's consider the terms on the right hand side one by one. If M is fixed and known, what's the probability that we see C ? Since $C = K \oplus M$, it only happens if $K = C \oplus M$. The probability that K is this particular string is exactly 2^{-n} . Thus

$$\Pr[C \mid M] = 2^{-n} . \quad (1.2)$$

By definition $\Pr[M] = \mathcal{M}(M)$ is the a priori probability of M . Now for the last term:

$$\begin{aligned} \Pr[C] &= \sum_m \Pr[m] \cdot \Pr[C \mid m] \\ &= \sum_m \mathcal{M}(m) \cdot 2^{-n} \\ &= 2^{-n} \cdot \sum_m \mathcal{M}(m) \\ &= 2^{-n} \cdot 1 . \end{aligned}$$

The sum here was over all possible messages m . We used the fact that $\Pr[C \mid m] = 2^{-n}$ as in Equation (1.2), and that the sum over all m of the probability of m is

of course 1 since \mathcal{M} is a probability distribution. Finally, plugging all this into Equation (1.1) we get

$$\Pr[M | C] = 2^{-n} \cdot \frac{\mathcal{M}(M)}{2^{-n}} = \mathcal{M}(M)$$

as desired. ■ ■

A limitation on Shannon-secure encryption Recall that the key in the one-time-pad scheme had to be at least as long as the number of bits we want to encrypt. It turns out that this is necessary to achieve Shannon security. That is, if an encryption scheme is to meet Definition 1.1, the number of key bits must be at least a total number of plaintext bits we’re going to encrypt.

This fact has some fundamental implications. If we want to do practical cryptography, we must be able to use a single short key to encrypt lots of bits. This means that we will not be able to achieve Shannon security. We must seek a different paradigm and a different notion of security.

Complexity theory Modern cryptography introduces a new dimension: the amount of computing power available to an adversary. It seeks to have security as long as adversaries don’t have “too much” computing time. Schemes are breakable “in principle,” but not in practice. Attacks are infeasible, not impossible.

This is a radical shift from many points of view. It takes cryptography from the realm of information theory into the realm of computer science, and complexity theory in particular, since that is where we study how hard problems are to solve as a function of the computational resources invested. And it changes what we can efficiently achieve.

We will want to be making statements like this:

Assuming the adversary uses no more than t computing cycles, her probability of breaking the scheme is at most $t/2^{200}$.

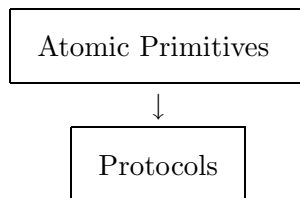
Notice again the statement is probabilistic. Almost of our statements will be.

Notice another important thing. Nobody said anything about *how* the adversary operates. What algorithm, or technique, does she use? We do not know anything about that. The statement holds nonetheless. So it is a very strong statement.

It should be clear that, in practice, a statement like the one above would be good enough. As the adversary works harder, her chance of breaking the scheme increases, and if the adversary had 2^{200} computing cycles at her disposal, we’d have no security left at all. But nobody has that much computing power.

Now we must ask ourselves how we can hope to get protocols with such properties. The legitimate parties must be able to efficiently execute the protocol instructions: their effort should be reasonable. But somehow, the task for the adversary must be harder.

Atomic primitives We want to make a distinction between the protocols that that we use and those that we are designing. At the lowest level are what we call *atomic primitives*. Higher level protocols are built on top of these.



What’s the distinction? Perhaps the easiest way to think of it is that the protocols we build address a cryptographic problem of interest. They say how to encrypt, how to authenticate, how to distribute a key. We build our protocols out of atomic primitives. Atomic primitives are protocols in their own right, but they are simpler protocols. Atomic primitives have some sort of “hardness” or “security” properties, but by themselves they don’t solve any problem of interest. They must be properly used to achieve some useful end.

In the early days nobody bothered to make such a distinction between protocols and the primitives that used them. And if you think of the one-time pad encryption method, there is really just one object, the protocol itself.

Atomic primitives are drawn from two sources: engineered constructs and mathematical problems. In the first class fall standard *block ciphers* such as the well-known DES algorithm. In the second class falls the RSA function. We’ll be looking at both types of primitives later.

The computational nature of modern cryptography means that one must find, and base cryptography on, computationally hard problems. Suitable ones are not so commonplace. Perhaps the first thought one might have for a source of computationally hard problems is **NP**-complete problems. Indeed, early cryptosystems tried to use these, particularly the Knapsack problem. However, these efforts have mostly failed. One reason is that **NP**-complete problems, although apparently hard to solve in the worst-case, may be easy on the average.

An example of a more suitable primitive is a *one-way function*. This is a function $f: D \rightarrow R$ mapping some domain D to some range R with two properties:

- (1) f is easy to compute: there is an efficient algorithm that given $x \in D$ outputs $y = f(x) \in R$.
- (2) f is hard to invert: an adversary I given a random $y \in R$ has a hard time figuring out a point x such that $f(x) = y$, as long as her computing time is restricted.

The above is not a formal definition. The latter, which we will see later, will talk about probabilities. The input x will be chosen at random, and we will then talk of the probability an adversary can invert the function at $y = f(x)$, as a function of the time for which she is allowed to compute.

Can we find objects with this strange asymmetry? It is sometimes said that one-way functions are obvious from real life: it is easier to break a glass than to

put it together again. But we want concrete mathematical functions that we can implement in systems.

One source of examples is number theory, and this illustrates the important interplay between number theory and cryptography. A lot of cryptography has been done using number theory. And there is a very simple one-way function based on number theory—something you already know quite well. Multiplication! The function f takes as input two numbers, a and b , and multiplies them together to get $N = ab$. There is no known algorithm that given a random $N = ab$, always and quickly recovers a pair of numbers (not 1 and N , of course!) that are factors of N . This “backwards direction” is the **factoring** problem, and it has remained unsolved for hundreds of years.

Here is another example. Let p be a prime. The set $Z_p^* = \{1, \dots, p-1\}$ turns out to be a group under multiplication modulo p . We fix an element $g \in Z_p^*$ which generates the group (that is, $\{g^0, g^1, g^2, \dots, g^{p-2}\}$ is all of Z_p^*) and consider the function $f: \{0, \dots, p-2\} \rightarrow Z_p^*$ defined by $f(x) = g^x \bmod p$. This is called *discrete exponentiation*, and its inverse is called **discrete logarithm**: $\log_g(y)$ is the value x such that $y = g^x$. It turns out there is no known fast algorithm that computes discrete logarithms, either. This means that for large enough p (say 1000 bits) the task is infeasible, given current computing power, even in thousands of years. So this is another one-way function.

It should be emphasized though that these functions have not been *proven* to be hard functions to invert. Like **P** versus **NP**, whether or not there is a good one-way function out there is an open question. We have some candidate examples, and we work with them. Thus, cryptography is built on assumptions. If the assumptions are wrong, a lot of protocols might fail. In the meantime we live with them.

The provable-security approach While there are several different ways in which proofs can be effective tools in cryptography, we will generally follow the proof-using tradition which has come to be known as “provable security.” Provable security emerged in 1982, with the work of Shafi Goldwasser and Silvio Micali. At that time, Goldwasser and Micali were graduate students at UC Berkeley. They, and their advisor Manuel Blum, wanted to put public-key encryption on a scientifically firm basis. And they did that, effectively creating a new viewpoint on what cryptography is really about.

We have explained above that we like to start from atomic primitives and transform them into protocols. Now good atomic primitives are rare, as are the people who are good at making and attacking them. Certainly, an important effort in cryptography is to design new atomic primitives, and to analyze the old ones. This, however, is not the part of cryptography that this course will focus on. One reason is that the weak link in real-world cryptography seems to be between atomic primitives and protocols. It is in this transformation that the bulk of security flaws arise. And there is a science that can do something about it, namely, provable security.

We will view a cryptographer as an engine for turning atomic primitives into

protocols. That is, we focus on protocol design under the assumption that good atomic primitives exist. Some examples of the kinds of questions we are interested in are these. What is the best way to encrypt a large text file using DES, assuming DES is secure? What is the best way to design a signature scheme using multiplication, assuming that multiplication is one-way? How “secure” are known methods for these tasks? What do such questions even mean, and can we find a good framework in which to ask and answer them?

A poorly designed protocol can be insecure *even though the underlying atomic primitive is good*. The fault is not of the underlying atomic primitive, but that primitive was somehow misused.

Indeed, lots of protocols have been broken, yet the good atomic primitives, like DES and multiplication and RSA, have never been convincingly broken. We would like to build on the strength of such primitives in such a way that protocols can “inherit” this strength, not lose it. The provable-security paradigm lets us do that.

The provable-security paradigm is as follows. Take some goal, like achieving privacy via symmetric encryption. The first step is to make a formal adversarial *model* and *define* what it *means* for an encryption scheme to be secure. The definition explains exactly when—on which runs—the adversary is successful.

With a definition in hand, a particular protocol, based on some particular atomic primitive, can be put forward. It is then analyzed from the point of view of meeting the definition. The plan is now show security via a *reduction*. A reduction shows that the *only* way to defeat the protocol is to break the underlying atomic primitive. Thus we will also need a formal definition of what the atomic primitive is supposed to do.

A reduction is a proof that if the atomic primitive does the job it is supposed to do, then the protocol we have made does the job that it is supposed to do. Believing this, there is no longer necessary to directly cryptanalyze the protocol: if you were to find a weakness in it, you would have unearthed one in the underlying atomic primitive. So if one is going to do cryptanalysis, one might as well focus on the atomic primitive. And if we believe the latter is secure, then we *know*, without further cryptanalysis of the protocol, that the protocol is secure, too.

A picture for the provable-security paradigm might look like Figure 1.11.

In order to do a reduction one must have a formal notion of what is meant by the security of the underlying atomic primitive: what attacks, exactly, does it withstand? For example, we might assume that RSA is a one-way function.

Here is another way of looking at what reductions do. When I give you a reduction from the onewayness of RSA to the security of my protocol, I am giving you a *transformation* with the following property. Suppose you claim to be able to break my protocol P . Let A be the adversary that you have that does this. My transformation takes A and turns it into another adversary, A' , that breaks RSA. Conclusion: as long as we believe you can't break RSA, there could be no such adversary A . In other words, my protocol is secure.

Figure 1.11: The provable-security paradigm.

We think that computational problem Ξ can't be solved in polynomial time.	We think that cryptographic protocol Π can't be effectively attacked.
We believe this because if Ξ could be solved in polynomial time, then so could SAT (say).	We believe this because if Π could be effectively attacked, then so could RSA (say).
To show this we <i>reduce</i> SAT to Ξ : we show that <i>if</i> somebody could solve Ξ in polynomial time, then they could solve SAT in polynomial time, too.	To show this we <i>reduce</i> RSA to Π : we show that <i>if</i> somebody could break Π by effective means, then they could break RSA by effective means, too.

Figure 1.12: The analogy between reductionist-cryptography and NP-Completeness.

Those familiar with the theory of **NP**-completeness will recognize that the basic idea of reductions is the same. When we provide a reduction from SAT to some computational problem Ξ we are saying our Ξ is hard unless SAT is easy; when we provide a reduction from RSA to our protocol Π , we are saying that Π is secure unless RSA is easy. The analogy is further spelled out in Figure 1.12, for the benefit of those of you familiar with the notion of NP-Completeness.

Experience has taught us that the particulars of reductions in cryptography are a little harder to comprehend than they were in elementary complexity theory. Part of the difficulty lies in the fact that every problem domain will have its own unique notion of what is an “effective attack.” It’s rather like having a different “version” of the notion of NP-Completeness as you move from one problem to another. We will also be concerned with the *quality* of reductions. One could have concerned oneself with this in complexity theory, but it’s not usually done. For doing practical work in cryptography, however, paying attention to the quality of reductions is important. Given these difficulties, we will proceed rather slowly through the ideas. Don’t worry; you will get it (even if you never heard of NP-Completeness).

The concept of using reductions in cryptography is a beautiful and powerful idea. Some of us by now are so used to it that we can forget how innovative it was! And for those not used to it, it can be hard to understand (or, perhaps, believe) at first hearing—perhaps because it delivers so much. Protocols designed this way truly have superior security guarantees.

In some ways the term “provable security” is misleading. As the above indicates, what is probably the central step is providing a model and definition, which does not involve proving anything. And then, one does not “prove a scheme secure:” one provides a reduction of the security of the scheme to the security of some underlying atomic primitive. For that reason, we sometimes use the term “reductionist security” instead of “provable security” to refer to this genre of work.

Theory for practice As you have by now inferred, this course emphasizes general principles, not specific systems. We will not be talking about the latest holes in *sendmail* or *Netscape*, how to configure *PGP*, or the latest attack against the ISO 9796 signature standard. This kind of stuff is interesting and useful, but it is also pretty transitory. Our focus is to understand the fundamentals, so that we know how to deal with new problems as they arise.

We want to make this clear because cryptography and security are now quite hyped topic. There are many buzzwords floating around. Maybe someone will ask you if, having taken a course, you know one of them, and you will not have heard of it. Don’t be alarmed. Often these buzzwords don’t mean much.

This is a theory course. Make no mistake about that! Not in the sense that we don’t care about practice, but in the sense that we approach practice by trying to understand the fundamentals and how to apply them. Thus the main goal is to understand the theory of protocol design, and how to apply it. We firmly believe it is via an understanding of the theory that good design comes. If you know the theory you can apply it anywhere; if you only know the latest technology your knowledge will soon be obsolete. We will see how the theory and the practice can contribute to each other, refining our understanding of both.

In assignments you will be asked to prove theorems. There may be a bit of mathematics for you to pick up. But more than that, there is “mathematical thinking.”

Don’t be alarmed if what you find in these pages contradicts “conventional wisdom.” Conventional wisdom is often wrong! And often the standard texts give an impression that the field is the domain of experts, where to know whether something works or not, you must consult an expert or the recent papers to see if an attack has appeared. The difference in our approach is that you will be given reasoning tools, and you can then think for yourself.

Cryptography is fun. Devising definitions, designing protocols, and proving them correct is a highly creative endeavor. We hope you come to enjoy thinking about this stuff, and that you come to appreciate the elegance in this domain.

1.4 What background do I need?

Now that you have had some introduction to the material and themes of the class, you need to decide whether you should take it. Here are some things to consider in taking this decision.

A student taking this course is expected to be comfortable with the following kinds of things, which are covered in various other courses.

The first is probability theory. Probability is everywhere in cryptography. You should be comfortable with ideas like sample spaces, events, experiments, conditional probability, random variables and their expectations. We won't use anything deep from probability theory, but we will draw heavily on the language and basic concepts of this field.

You should know about alphabets, strings and formal languages, in the style of an undergraduate course in the theory of computation.

You should know about algorithms and how to measure their complexity. In particular, you should have taken and understood at least an undergraduate algorithms class.

Most of all you should have general mathematical maturity, meaning, especially, you need to be able to understand what is (and what is not) a proper definition.

1.5 Historical notes

1.6 Exercises and problems

Exercise 1.1 Suppose that you want to encrypt a single message $M \in \{0, 1, 2\}$ using a random shared key $K \in \{0, 1, 2\}$. Suppose you do this by representing K and M using two bits (00, 01, or 10), and then XORing the two representations. Does this seem like a good protocol to you? Explain.

Exercise 1.2 Suppose that you want to encrypt a single message $M \in \{0, 1, 2\}$ using a random shared key $K \in \{0, 1, 2\}$. Explain a good way to do this.

Exercise 1.3 Besides the symmetric and the asymmetric trust models, think of a couple more ways to “create asymmetry” between the receiver and the adversary. Show how you would encrypt a bit in each of your model.

Exercise 1.4 In the telephone coin-flipping protocol, what should happen if Alice refuses to send her second message? Is this potentially damaging?

Exercise 1.5 Give a clear argument why what we said about keeping the algorithm public but the key secret is fundamentally meaningless.

Problem 1.1 *A limitation on fixed-time fair-coin-flipping TMs.* Consider the model of computation in which we augment a Turing machine so that it can obtain the

output of a random coin flip: by going into a distinguished state Q_S , the next state will be Q_H with probability $1/2$, and the next state will be Q_T with probability $1/2$. Show that, in this model of computation, there is no constant-time algorithm to perfectly deal out five cards to each of two players.

(A deck of cards consists of 52 cards, and a perfect deal means that all hands should be equally likely. Saying that the algorithm is constant-time means that there is some number T such that the algorithm is guaranteed to stop within T steps.)

Problem 1.2 *Symmetric encryption with a deck of cards.* Alice shuffles a deck of cards and deals it all out to herself and Bob (each of them gets half of the 52 cards). Alice now wishes to send a secret message M to Bob by saying something aloud. Eavesdropper Eve is listening in: she hears everything Alice says (but Eve can't see the cards).

Part A. Suppose Alice's message M is a string of 48-bits. Describe how Alice can communicate M to Bob in such a way that Eve will have *no* information about what is M .

Part B. Now suppose Alice's message M is 49 bits. Prove that there exists no protocol which allows Alice to communicate M to Bob in such a way that Eve will have no information about M .

(What does it mean that Eve learns nothing about M ? That for all strings κ , the probability that Alice says κ is independent of M : for all messages M_0, M_1 we have that

$$\Pr[\text{Alice says } \kappa \mid M = M_0] = \Pr[\text{Alice says } \kappa \mid M = M_1] .$$

The probability is over the the random shuffle of the cards.)

Problem 1.3 *Composition of EPT Algorithms.* John designs an EPT (expected polynomial time) algorithm to solve some computational problem Π —but he assumes that he has in hand a black-box (ie., a unit-time subroutine) which solves some other computational problem, Π' . Ted soon discovers an EPT algorithm to solve Π' . True or false: putting these two pieces together, John and Ted now have an EPT algorithm for Π . Give a proof or counterexample.

(When we speak of the worst-case running time of machine M we are looking at the function $T(n)$ which gives, for each n , the maximal time which M might spend on an input of size n : $T(n) = \max_{x, |x|=n} [\#\text{Steps}_M(x)]$. When we speak of the expected running time of M we are instead looking at the function $T(n)$ which gives, for each n , the maximal value among inputs of length n of the expected value of the running time of M on this input—that is, $T(n) = \max_{x, |x|=n} \mathbf{E}[\#\text{Steps}_M(x)]$, where the expectation is over the random choices made by M .)

Chapter 2

BLOCK CIPHERS

Block ciphers are the central tool in the design of protocols for shared-key cryptography. They are the main available “technology” we have at our disposal. This chapter will take a look at these objects and describe the state of the art in their construction.

It is important to stress that block ciphers are just tools—raw ingredients for cooking up something more useful. Block ciphers don’t, by themselves, do something that an end-user would care about. As with any powerful tool, one has to learn to use this one. Even a wonderful block cipher won’t give you security if you use don’t use it right. But used well, these are powerful tools indeed. Accordingly, an important theme in several upcoming chapters will be on how to use block ciphers well. We won’t be emphasizing how to design or analyze block ciphers, as this remains very much an art. The main purpose of this chapter is just to get you acquainted with what typical block ciphers look like. We’ll look at two examples, DES and AES. DES is the “old standby.” It is currently (year 2001) the most widely-used block cipher in existence, and it is of sufficient historical significance that every trained cryptographer needs to have seen its description. AES is a modern block cipher, and it is expected to supplant DES in the years to come.

2.1 What is a block cipher?

A block cipher is a function $E: \{0, 1\}^k \times \{0, 1\}^n \rightarrow \{0, 1\}^n$ that takes two inputs, a k -bit key K and an n -bit “plaintext” M , to return an n -bit “ciphertext” $C = E(K, M)$. The *key-length* k and the *block-length* n are parameters associated to the block cipher. They vary from block cipher to block cipher, as of course does the design of the algorithm itself. For each key $K \in \{0, 1\}^k$ we let $E_K: \{0, 1\}^n \rightarrow \{0, 1\}^n$ be the function defined by $E_K(M) = E(K, M)$. For any block cipher, and any key K , it is required that the function E_K be a *permutation* on $\{0, 1\}^n$. This means that it is a bijection (ie., a one-to-one and onto function) of $\{0, 1\}^n$ to $\{0, 1\}^n$.

Accordingly E_K has an inverse, and we can denote it E_K^{-1} . This function also maps $\{0, 1\}^n$ to $\{0, 1\}^n$, and of course we have $E_K^{-1}(E_K(M)) = M$ and $E_K(E_K^{-1}(C)) = C$ for all $M, C \in \{0, 1\}^n$. We let $E^{-1}: \{0, 1\}^k \times \{0, 1\}^n \rightarrow \{0, 1\}^n$ be defined by $E^{-1}(K, C) = E_K^{-1}(C)$; this is the inverse block cipher to E .

One imagines that the block cipher E is a public and fully specified algorithm. Both the cipher E and its inverse E^{-1} should be easily computable, meaning given K, M we can readily compute $E(K, M)$, and given K, C we can readily compute $E^{-1}(K, C)$.

In typical usage, a random key K is chosen and kept secret between a pair of users. The function E_K is then used by the two parties to process data in some way before they send it to each other. Typically, the adversary will be able to see input-output examples for E_K , meaning pairs of the form (M, C) where $C = E_K(M)$. But, ordinarily, the adversary will not be shown the key K . Security relies on the secrecy of the key. So, as a first cut, you might think of the adversary's goal as recovering the key K given some input-output examples of E_K . The block cipher should be designed to make this task computationally difficult. Later we will refine this (fundamentally incorrect) view.

We emphasize that we've said absolutely nothing about what properties a block cipher should have. A function like $E_K(M) = M$ is a block cipher (the "identity block cipher"), but we shall not regard it as a "good" one. Only in the next chapter do we start to take up what "goodness" means for a block cipher.

How do real block ciphers work? Lets take a look at some of them to get a sense of this.

2.2 Data Encryption Standard

The Data Encryption Standard (DES) is the quintessential block cipher. Even though it is now quite old, and on the way out, no discussion of block ciphers can really omit mention of this construction. DES is a remarkably well-engineered algorithm which has had a powerful influence on cryptography. It is in very widespread use, and probably will be for some years to come. Every time you use an ATM machine, you are using DES.

2.2.1 A brief history

In 1972 the NBS (National Bureau of Standards, now NIST, the National Institute of Standards and Technology) initiated a program for data protection and wanted as part of it an encryption algorithm that could be standardized. They put out a request for such an algorithm. In 1974, IBM responded with a design based on their "Lucifer" algorithm. This design would eventually evolve into the DES.

DES has a key-length of $k = 56$ bits and a block-length of $n = 64$ bits. It consists of 16 rounds of what is called a "Feistel network." We will describe more details shortly.

After NBS, several other bodies adopted DES as a standard, including ANSI (the American National Standards Institute) and the American Bankers Association.

The standard was to be reviewed every five years to see whether or not it should be re-adopted. Although there were claims that it would not be re-certified, the algorithm was re-certified again and again. Only recently did the work for finding a replacement begin in earnest, in the form of the AES (Advanced Encryption Standard) effort.

DES proved remarkably secure. There has, since the beginning, been one primary concern, and that was the threat of key-search. But for a fairly long time, the key size of 56 bits was good enough against all but very well-funded organizations. Interesting attacks emerged only in the nineties, and even then they don't break DES in a sense more significant than the threat of exhaustive key search. But with today's technology, 56 bits is just too small a key size for many security applications. The problem is that, with a modest expenditure of funds, one can build a machine that, given a plaintext-ciphertext pair, say, will search the entire space of 2^{56} keys, locating the correct one (or ones) in a modest amount of time. Indeed the Electronic Frontier Foundation (EFF) has, as a demonstration of DES's vulnerability, constructed just such a machine.

2.2.2 Construction

Revise, drawing a single picture, for the Feistel construction, instead of referring to the FIPS.

The construction is described in FIPS 46 [7]. The following discussion is a quick guide that you can follow if you have the FIPS document at your side.

Begin at page 87 where you see a big picture. The input is 64 bits and in addition there is a 56 bit key K . (They say 64, but actually every eighth bit is ignored. It is often mandated to be the xor of the previous seven bits.) Notice the algorithm is public. You operate with a hidden key, but nothing about the algorithm is hidden.

The first thing the input is hit with is something called the initial permutation, or IP. This just shuffles bit positions. That is, each bit is moved to some other position. How? In a fixed and specified way: see page 88. Similarly, right at the end, notice they apply the inverse of the same permutation. From now on, ignore these. They do not affect security (as far as anyone knows).

The essence of DES is in the round structure. There are 16 rounds. Each round i has an associated subkey K_i which is 48 bits long. The subkeys K_1, \dots, K_{16} are derived from the main key K , in a manner explained on page 95 of the FIPS document.

In each round, the input is viewed as a pair (L_i, R_i) of 32 bit blocks, and these are transformed into the new pair (L_{i+1}, R_{i+1}) , via a certain function f that depends on a subkey K_i associated to round i . The structure of this transformation is important: it is called the Feistel transformation.

The Feistel transformation, in general, is like this. For some function g known to the party computing the transformation, it takes input (L, R) and returns (L', R')

where $L' = R$ and $R' = g(R) \oplus L$. A central property of this transformation is that it is a permutation, and moreover if you can compute g then you can also easily invert the transformation. Indeed, given (L', R') we can recover (L, R) via $R = L'$ and $L = g(R) \oplus R'$. For DES, the role of g in round i is played by $f(K_i, \cdot)$, the round function specified by the subkey K_i . Since $\text{DES}_K(\cdot)$ is a sequence of Feistel transforms, each of which is a permutation, the whole algorithm is a permutation, and knowledge of the key K permits computation of $\text{DES}_K^{-1}(\cdot)$.

Up to now the structure has been quite generic, and indeed many block-ciphers use this high level design: a sequence of Feistel rounds. For a closer look we need to see how the function $f(K_i, \cdot)$ works. This function maps 32 bits to 32 bits. See the picture on page 90 of the FIPS document. Here K_i is a 48-bit subkey, derived from the 56-bit key (just by selecting particular bits) in a way depending on the round number. The 32-bit R_i is first expanded into 48 bits. How? In a precise, fixed way, indicated by the table on the same page, saying E-bit selection table. It has 48 entries. Read it as which inputs bits are output. Namely, output bits 32, 1, 2, 3, 4, 5, then 4, 5 again, and so on. It is NOT random looking! In fact barring that 1 and 32 have been swapped (see top left and bottom right) it looks almost sequential. Why did they do this? Who knows. That's the answer to most things about DES.

Now K_i is XORed with the output of the E-box and this 48 bit input enters the famous S-boxes. There are eight S-boxes. Each takes 8 bits to 6 bits. Thus we get out 32 bits. Finally, there is a P-box, a permutation applied to these 32 bits to get another 32 bits. You can see it on page 91.

What are the S-boxes? Each is a fixed, tabulated function, which the algorithm stores as tables in the code or hardware. You can see them on page 93. How to read them? Take the 6 bit input $b_1, b_2, b_3, b_4, b_5, b_6$. Interpret the first and last bits as a row number (row 0, 1, 2, or 3). Interpret the rest as a column number (column 0, 1, ..., 15). Now look up what you get in the table and write down those four bits.

Well now you know how DES works. Of course, the main questions about the design are: why, why and why? What motivated these design choices? We don't know too much about this, although we can guess a little. And one of the designers of DES, Don Coppersmith, has written a short paper which gives information on why (thought what Don wrote was information which had effectively been reverse-engineered out of the algorithm in the previous years).

2.2.3 Speed

One of the design goals of DES was that it would have fast implementations relative to the technology of its time. How fast can you compute DES? In roughly current technology (well, nothing is current by the time one writes it down!) one can get well over 1 Gbit/sec on high-end VLSI. Specifically at least 1.6 Gbits/sec, maybe more. That's pretty fast. Perhaps a more interesting figure is that one can implement each DES S-boxes with at most 50?? two-input gates, where the circuit has depth of only 3??. Thus one can compute DES by a combinatorial circuit of about $8 \cdot 16 \cdot 50 = 640$

gates and depth of $3 \cdot 16 = 48$ gates.

In software, on a fairly modern processor, DES takes something like 80(?) cycles per byte. This is disappointingly slow—not surprisingly, since DES was optimized for hardware and was designed before the days in which software implementations were considered feasible or desirable.

2.3 Advanced Encryption Standard

In 1998 the National Institute of Standards and Technology (NIST/USA) announced a “competition” for a new block cipher. The new block cipher would, in time, replace DES. The relatively short key length of DES was the main problem that motivated the effort: with the advances in computing power, a key space of 2^{56} keys was just too small. With the development of a new algorithm one could also take the opportunity to address the modest software speed of DES, making something substantially faster, and to increase the block size from 64 to 128 bits (the choice of 64 bits for the block size can lead to security difficulties, as we shall later see. Unlike the design of DES, the new algorithm would be designed in the open and by the public.

Fifteen algorithms were submitted to NIST. They came from around the world. A second round narrowed the choice to five of these algorithms. In the summer of 2001 NIST announced their choice: an algorithm called Rijndael. The algorithm should be embodied in a NIST FIPS (Federal Information Processing Standard) any day now; right now, there is a draft FIPS. Rijndael was designed by Joan Daemen and Vincent Rijmen (from which the algorithm gets its name), both from Belgium. It is descendent of an algorithm called Square.

In this section we shall describe AES.

A word about notation. Purists would prefer to reserve the term “AES” to refer to the standard, using the word “Rijndael” or the phrase “the AES algorithm” to refer to the algorithm itself. (The same naming pundits would have us use the acronym DEA, Data Encryption Algorithm, to refer to the algorithm of the DES, the Data Encryption Standard.) We choose to follow common convention and refer to both the standard and the algorithm as AES. Such an abuse of terminology never seems to lead to any misunderstandings.

The AES has a block length of $n = 128$ bits, and a key length k that is variable: it may be 128, 192 or 256 bits. So the standard actually specifies three different block ciphers: AES128, AES192, AES256. These three block ciphers are all very similar, so we will stick to describing just one of them, AES128. For simplicity, in the remainder of this section, AES means the algorithm AES128. We’ll write $C = \text{AES}_K(M)$ where $|K| = 128$ and $|M| = |C| = 128$.

We’re going to describe AES in terms of four additional mappings: *expand*, *S*, *shift-rows*, and *mix-cols*. The function *expand* takes a 128-bit string and produces a vector of eleven keys, (K_0, \dots, K_{10}) . The remaining three functions bijectively map 128-bits to 128-bits. Actually, we’ll be more general for *S*, letting it be a map

```

function AESK(M)
begin
    (K0, ..., K10) ← expand(K)      s ← M      s ← s ⊕ K0
[1]   s ← S(s)      s ← shift-rows(s)  s ← mix-cols(s)  s ← s ⊕ K1
[2]   s ← S(s)      s ← shift-rows(s)  s ← mix-cols(s)  s ← s ⊕ K2
[3]   s ← S(s)      s ← shift-rows(s)  s ← mix-cols(s)  s ← s ⊕ K3
[4]   s ← S(s)      s ← shift-rows(s)  s ← mix-cols(s)  s ← s ⊕ K4
[5]   s ← S(s)      s ← shift-rows(s)  s ← mix-cols(s)  s ← s ⊕ K5
[6]   s ← S(s)      s ← shift-rows(s)  s ← mix-cols(s)  s ← s ⊕ K6
[7]   s ← S(s)      s ← shift-rows(s)  s ← mix-cols(s)  s ← s ⊕ K7
[8]   s ← S(s)      s ← shift-rows(s)  s ← mix-cols(s)  s ← s ⊕ K8
[9]   s ← S(s)      s ← shift-rows(s)  s ← mix-cols(s)  s ← s ⊕ K9
[10]  s ← S(s)      s ← shift-rows(s)  s ← mix-cols(s)  s ← s ⊕ K10
    return s
end

```

Figure 2.1: The function AES128. See the accompanying text and figures for definitions of the maps *expand*, *S*, *shift-rows*, *mix-cols*.

on $(\{0, 1\}^8)^+$. Let's postpone describing all of these maps and start off with the high-level structure of AES, which is given in Figure 2.3.

Refer to Figure 2.3. The value s is called the *state*. One initializes the state to M and the final state is the ciphertext C one gets by enciphering M . What happens in each of lines 1–10 is called a *round*. So AES (remember this means AES128 in this section) consists of ten rounds. The rounds are identical except that each uses a different subkey K_i and, also, round 10 omits the call to *mix-cols*.

To understand what goes on in S and *mix-cols* we will need to review a bit of algebra. Let us make a pause to do that. We describe a way to do arithmetic on bytes. Identify each byte $a = a_7a_6a_5a_4a_3a_2a_1a_0$ with the formal polynomial $a_7x^7 + a_6x^6 + a_5x^5 + a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0$. We can add two bytes by taking their bitwise xor (which is the same as the mod-2 sum the corresponding polynomials). We can multiply two bytes to get a degree 14 (or less) polynomial, and then take the remainder of this polynomial by the fixed irreducible polynomial

$$m(x) = x^8 + x^4 + x^3 + x + 1 .$$

This remainder polynomial is a polynomial of degree at most seven which, as before, can be regarded as a byte. In this way can add and multiply any two bytes. The resulting algebraic structure has all the properties necessary to be called a *finite field*. In particular, this is one representation of the finite field known as $\text{GF}(2^8)$ —the Galois field on $2^8 = 256$ points. As a finite field, you can find the inverse of any nonzero field point (the zero-element is the zero byte) and you can distribute addition over multiplication, for example.

There are some useful tricks when you want to multiply two bytes. Since $m(\mathbf{x})$ is another name for zero, $\mathbf{x}^8 = \mathbf{x}^4 + \mathbf{x}^3 + \mathbf{x} + 1 = \{1\mathbf{b}\}$. (Here the curly brackets simply indicate a hexadecimal number.) So it is easy to multiply a byte a by the byte $\mathbf{x} = \{02\}$: namely, shift the 8-bit byte a one position to the left, letting the first bit “fall off” (but remember it!) and shifting a zero into the last bit position. We write this operation $a \ll 1$. If that first bit of a was a 0, we are done. If the first bit was a 1, we need to add in (that is, xor in) $\mathbf{x}^8 = \{1\mathbf{b}\}$. In summary, for a a byte, $a \cdot \mathbf{x} = a \cdot \{02\}$ is $a \ll 1$ if the first bit of a is 0, and it is $(a \ll 1) \oplus \{1\mathbf{b}\}$ if the first bit of a is 1.

Knowing how to multiply by $\mathbf{x} = \{02\}$ let’s you conveniently multiply by other quantities. For example, to compute $\{a1\} \cdot \{03\}$ compute $\{a1\} \cdot (\{02\} \oplus \{01\}) = \{a1\} \cdot \{02\} \oplus \{a1\} \cdot \{01\} = \{42\} \oplus \{1\mathbf{b}\} \oplus a1 = \{f8\}$. Try some more examples on your own.

As we said, each nonzero byte a has a multiplicative inverse, $inv(a) = a^{-1}$. The mapping we will denote $S : \{0, 1\}^8 \rightarrow \{0, 1\}^8$ is obtained from the map $inv : a \mapsto a^{-1}$. First, patch this map to make it total on $\{0, 1\}^8$ by setting $inv(\{00\}) = \{00\}$. Then, to compute $S(a)$, first replace a by $inv(a)$, number the bits of a by $a = a_7a_6a_5a_4a_3a_2a_1a_0$, and return the value a' , where $a' = a'_7a'_6a'_5a'_4a'_3a'_2a'_1a'_0$ where

$$\begin{pmatrix} a'_7 \\ a'_6 \\ a'_5 \\ a'_4 \\ a'_3 \\ a'_2 \\ a'_1 \\ a'_0 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} a_7 \\ a_6 \\ a_5 \\ a_4 \\ a_3 \\ a_2 \\ a_1 \\ a_0 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

All arithmetic is in $GF(2)$, meaning that addition of bits is their xor and multiplication of bits is the conjunction (and).

All together, the map S is give by Figure 2.2, which lists the values of $S(0), S(1), \dots, S(255)$. In fact, one could forget how this table is produced, and just take it for granted. But the fact is that it is made in the simple way we have said.

Now that we have the function S , let us extend it (without bothering to change the name) to a function with domain $\{\{0, 1\}^8\}^+$. Namely, given an m -byte string $A = A[1] \dots A[m]$, set $S(A)$ to be $S(A[1]) \dots S(A[m])$. In other words, just apply S byte-wise.

Now we’re ready to understand the first map, $S(s)$. One takes the 16-byte state s and applies the 8-bit lookup table to each of its bytes to get the modified state s .

Moving on, the *shift-rows* operation works like this. Imagine plastering the 16 bytes of $s = s_0s_1 \dots s_{15}$ going top-to-bottom, then left-to-right, to make a 4×4

63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Figure 2.2: The AES S-box, which is a function $S : \{0, 1\}^8 \rightarrow \{0, 1\}^8$ specified by the following list. All values in hexadecimal. The meaning is: $S(00) = 63$, $S(01) = 7c$, ..., $S(ff) = 16$.

table:

s_0	s_4	s_8	s_{12}
s_1	s_5	s_9	s_{13}
s_2	s_6	s_{10}	s_{14}
s_3	s_7	s_{11}	s_{15}

For the *shift-rows* step, left circularly shift the second row by one position; the third row by two positions; and the the fourth row by three positions. The first row is not shifted at all. Somewhat less colorfully, the mapping is simply

$$\text{shift-rows}(s_0s_1s_2 \cdots s_{15}) = s_0s_5s_{10}s_{15}s_4s_9s_{14}s_3s_8s_{13}s_2s_7s_{12}s_1s_6s_{11}$$

Using the same convention as before, the *mix-cols* step takes each of the four columns in the 4×4 table and applies the (same) transformation to it. Thus we define $\text{mix-cols}(s)$ on 4-byte words, and then extend this to a 16-byte quantity wordwise. The value of $\text{mix-cols}(a_0a_1a_2a_3) = a'_0a'_1a'_2a'_3$ is defined by:

$$\begin{pmatrix} a'_0 \\ a'_1 \\ a'_2 \\ a'_3 \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 02 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \cdot \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix}$$

An equivalent way to explain this step is to say that we are multiplying $a(\mathbf{x}) = a_3\mathbf{x}^3 + a_2\mathbf{x}^2 + a_1\mathbf{x}^1 + a_0$ by the fixed polynomial $c(\mathbf{x}) = \{03\}\mathbf{x}^3 + \{01\}\mathbf{x}^2 + \{01\}\mathbf{x} + \{02\}$ and taking the result modulo $\mathbf{x}^4 + 1$.


```

function expand( $K$ )
begin
   $K_0 \leftarrow K$ 
  for  $i \leftarrow 1$  to 10 do
     $K_i[0] \leftarrow K_{i-1}[0] \oplus S(K_{i-1}[3] \ll 8) \oplus C_i$ 
     $K_i[1] \leftarrow K_{i-1}[1] \oplus K_i[0]$ 
     $K_i[2] \leftarrow K_{i-1}[2] \oplus K_i[1]$ 
     $K_i[3] \leftarrow K_{i-1}[3] \oplus K_i[2]$ 
  od
  return ( $K_0, \dots, K_{10}$ )
end

```

Figure 2.3: The AES128 key-expansion algorithm maps a 128-bit key K into eleven 128-bit subkeys, K_0, \dots, K_{10} . Constants (C_1, \dots, C_{10}) are $(\{02000000\}, \{04000000\}, \{08000000\}, \{10000000\}, \{20000000\}, \{40000000\}, \{80000000\}, \{1B000000\}, \{36000000\}, \{6C000000\})$. All other notation is described in the accompanying text.

At this point we have described everything but the key-expansion map, *expand*. That map is given in Figure 2.3.

We have now completed the definition of AES. One key property is that AES *is* a block cipher: the map is invertible. This follows because every round is invertible. That a round is invertible follows from each of its steps being invertible, which is a consequence of S being a permutation and the matrix used in *mix-cols* having an inverse (see Exercise ??).

After seeing a definition like that of AES, one is left having essentially no idea why it should be so. The truth is that there are no satisfying answer to this question. The answer one hears normally amounts to: “we have been unable to find effective attacks, and we have tried attacks along the following lines . . .” If people with enough smarts and experience utter this statement, then it suggests that the block cipher is good. Beyond this, it’s hard to say much. Yet, by now, our community has become reasonably experienced designing these things. It wouldn’t even be that hard a game were it not for the fact we tend to be egressive in optimizing the block-cipher’s speed. (Some may come to the opposite opinion, that it’s a very hard game, seeing just how many reasonable-looking block ciphers have been broken.) In the following section we give some vague sense of the sort of cleverness that people muster against block ciphers.

2.4 Cryptanalysis of 5-round AES

Where 5 is whatever constant leads to a pedagogically interesting section. Alternatively, we could introduce a “toy” cipher to illustrate clever cryptanalysis—there is a

very clean suggestion by Shamir and ??—but I'd rather not introduce any additional block cipher.

Anyone interested in cryptanalysis or AES is welcome to figure out what are the state-of-the-art attacks on reduced-round Rijndael and then, as your final project, write a proposal for a pedagogically good attack on a reduced-round Rijndael.

2.5 Some modes of operation

Fix a block cipher E , and assume two parties share a key K for this block cipher. This gives them the ability to compute the functions $E_K(\cdot)$ and $E_K^{-1}(\cdot)$. These functions can be applied to an input of n -bits. An application of E_K is called *enciphering* and an application of E_K^{-1} is called *deciphering*.

Typically the block size n is 64 or 128. Yet, in practice, we may want to process much larger inputs, say text files to encrypt. To do this one uses a block cipher in some *mode of operation*. There are several well-known modes of operation. We will illustrate by describing three of them, all intended for message privacy. We look at ECB (Electronic Codebook), CBC (Cipher Block Chaining) and CTR (Counter). In each case there is an encryption process which takes an nm -bit string M , usually called the plaintext, and returns a string C , usually called the ciphertext. (If the length of M is not a positive multiple of n then some appropriate padding can be done to make it so. We're not going to worry about that here; we'll simply assume that each plaintext M has a length which is some positive multiple of n .) An associated decryption process recovers M from C .

If M is a string whose length is a multiple of n then we view it as divided into a sequence of n -bit blocks, and let $M[i]$ denote the i -th block, for $i = 1, \dots, |M|/n$. That is, $M = M[1] \dots M[m]$ where $m = |M|/n$.

2.5.1 Electronic codebook mode

Each plaintext block is individually enciphered into an associated ciphertext block.

Algorithm $\mathcal{E}_K(M[1] \dots M[m])$ For $i = 1, \dots, m$ do $C[i] \leftarrow E_K(M[i])$ Return $C[1] \dots C[m]$	Algorithm $\mathcal{D}_K(C[1] \dots C[m])$ For $i = 1, \dots, m$ do $M[i] \leftarrow E_K^{-1}(C[i])$ Return $M[1] \dots M[m]$
---	--

2.5.2 Cipher-block chaining mode

CBC mode processes the data based on some *initialization vector* IV which is an l -bit string, as follows.

Algorithm $\mathcal{E}_K(\text{IV}, M[1] \cdots M[m])$ $C[0] \leftarrow \text{IV}$ For $i = 1, \dots, n$ do $C[i] \leftarrow E_K(C[i-1] \oplus M[i])$ Return $C[0]C[1] \cdots C[m]$	Algorithm $\mathcal{D}_K(C[0]C[1] \cdots C[n])$ For $i = 1, \dots, n$ do $M[i] \leftarrow E_K^{-1}(C[i]) \oplus C[i-1]$ Return $M[1] \dots M[n]$
---	---

Unlike ECB encryption, this operation is not length preserving: the output is n -bits longer than the input. The initialization vector is used for encryption, but it is then made part of the ciphertext, so that the receiver need not be assumed to know it a priori.

Different specific modes result from different ways of choosing the initialization vector. Unless otherwise stated, it is assumed that before applying the above encryption operation, the encryptor chooses the initialization vector at random, anew for each message M to be encrypted. Other choices however can also be considered, such as letting IV be a counter that is incremented by one each time the algorithm is applied. The security attributed of these different choices are discussed later.

CBC is the most popular mode, used pervasively in practice.

2.5.3 Counter mode

CTR mode also uses an auxiliary value, an “initial value” IV which is an integer in the range $0, 1, \dots, 2^n - 1$. In the following, addition is done modulo 2^n , and $[j]_n$ denotes the binary representation of integer j as an n -bit string.

Algorithm $\mathcal{E}_K(\text{IV}, M[1] \cdots M[m])$ For $i = 1, \dots, m$ do $C[i] \leftarrow E_K([\text{IV} + i]_n) \oplus M[i]$ Return $[\text{IV}]_n C[1] \cdots C[m]$	Algorithm $\mathcal{D}_K([\text{IV}]_n C[1] \cdots C[m])$ For $i = 1, \dots, m$ do $M[i] \leftarrow E_K([\text{IV} + i]_n) \oplus C[i]$ Return $M[1] \dots M[m]$
---	---

Notice that in this case, decryption did not require computation of E_K^{-1} , and in fact did not even require that E_K be a permutation. Also notice the efficiency advantage over CBC: the encryption is parallelizable.

Again, there are several choices regarding the initial vector. It could be a counter maintained by the sender and incremented by $m = |M|/n$ after message M has been encrypted. Or, it could be chosen anew at random each time the algorithm is invoked. And there are still other possibilities.

2.6 Key recovery attacks on block ciphers

Old fragments, needs to be rewritten and harmonized with the rest of the chapter.

Historically, cryptanalysis of block ciphers $E: \{0, 1\}^k \times \{0, 1\}^n \rightarrow \{0, 1\}^n$ has always focused on key-recovery. The cryptanalyst may think of the problem to be solved as something like this. A k -bit key K is chosen at random. Let $q \geq 0$ be some integer parameter.

GIVEN: The adversary has a sequence of q input-output examples of E_K , say

$$(M_1, C_1), \dots, (M_q, C_q)$$

where $C_i = E_K(M_i)$ for $i = 1, \dots, q$ and M_1, \dots, M_q are all distinct n -bit strings.

FIND: The adversary must find the key K .

Some typical kinds of “attack” that are considered within this framework:

KNOWN-MESSAGE ATTACK: M_1, \dots, M_q are any distinct points; the adversary has no control over them, and must work with whatever it gets.

CHOSEN-MESSAGE ATTACK: M_1, \dots, M_q are chosen by the adversary, perhaps even adaptively. That is, imagine it has access to an “oracle” for the function E_K . It can feed the oracle M_1 and get back $C_1 = E_K(M_1)$. It can then decide on a value M_2 , feed the oracle this, and get back C_2 , and so on.

Clearly a chosen-message attack gives the adversary much more power, but is also less realistic in practice.

The most obvious attack is exhaustive key search.

EXHAUSTIVE KEY SEARCH: Go through all possible keys $K' \in \{0, 1\}^k$ until you find one that explains the input/output pairs. Probably it is K . (Really?!) How do you know when you hit K ? If $E_{K'}(M_1) = C_1$, you bet that $K' = K$. Of course, you could be wrong. But the “chance” of being wrong is small, and gets much smaller if you do more such tests. (Really?) For DES, two tests is quite enough. That is, the attack in this case only needs $q = 2$, a very small number of input-output examples.

Let us now describe the attack in more detail. For $i = 1, \dots, 2^k$ let K_i denote the i -th k -bit string (in lexicographic order). The following algorithm implements the attack.

```

For  $i = 1, \dots, 2^k$  do
  If  $E(K_i, M_1) = C_1$ 
    then if  $E(K_i, M_2) = C_2$  then return  $K_i$ 

```

How long does this take? In the worst case, 2^k computations of the block cipher. For the case of DES, even if you use the above mentioned 1.6 Gbits/sec chip to do these computations, the search takes about 6,000 years. So key search appears to be infeasible.

Yet, this conclusion is actually too hasty. We will return to key search and see why later.

DIFFERENTIAL AND LINEAR CRYPTANALYSIS: For DES, the discovery of theoretically superior attacks (assuming one has massive amount of plaintext/ciphertext pairs) waited until 1990. Differential cryptanalysis is capable of finding a DES key using about 2^{47} input-output examples (that is, it requires $q = 2^{47}$). However, differential cryptanalysis required a chosen-message attack.

Linear cryptanalysis improved differential in two ways. The number of input-output examples required is reduced to 2^{43} , but also only a known-message attack is required.

These were major breakthroughs in cryptanalysis. Yet, their practical impact is small. Why? Ordinarily it would be impossible to obtain 2^{43} input-output examples. Furthermore, simply storing all these examples requires about 140 terabytes.

Linear and differential cryptanalysis were however more devastating when applied to other ciphers, some of which succumbed completely to the attack.

So what's the best possible attack against DES? The answer is exhaustive key search. What we ignored above is *parallelism*.

KEY SEARCH MACHINES: A few years back it was argued that one can design a \$1 million machine that does the exhaustive key search for DES in about 3.5 hours. More recently, a DES key search machine was actually built, at a cost of \$250,000. It finds the key in 56 hours, or about 2.5 days. The builders say it will be cheaper to build more machines now that this one is built.

Thus DES is feeling its age. Yet, it would be a mistake to take away from this discussion the impression that DES is weak. Rather, what the above says is that it is an impressively strong algorithm. After all these years, the best practical attack known is still exhaustive key search. That says a lot for its design and its designers.

Later we will see that that we would like security properties from a block cipher that go beyond resistance to key-recovery attacks. It turns out that from that point of view, a limitation of DES is its block size. Birthday attacks “break” DES with about $q = 2^{32}$ input output examples. (The meaning of “break” here is very different from above.) Here 2^{32} is the square root of 2^{64} , meaning to resist these attacks we must have bigger block size. The next generation of ciphers—things like AES—took this into account.

2.7 Limitations of key-recovery based security

As discussed above, classically, the security of a block ciphers has been looked at with regard to key recovery. That is, analysis of a block cipher E has focused primarily on the following question: given some number q of input-output examples $(M_1, C_1), \dots, (M_q, C_q)$, where K is a random, unknown key and $C_i = E_K(M_i)$, how hard is it for an attacker to find K ? A block cipher is viewed as “secure” if the best key-recovery attack is computationally infeasible, meaning requires a value of q or a running time t that is too large to make the attack practical. In the sequel, we refer to this as *security against key-recovery*

However, as a notion of security, security against key-recovery is quite limited. A good notion should be sufficiently strong to be useful. This means that if a block cipher is secure, then it should be possible to use the block cipher to make worthwhile constructions and be able to have some guarantee of the security of these constructions. But even a cursory glance at common block cipher usages shows that

good security in the sense of key recovery is not sufficient for security of the usages of block ciphers.

Take for example the CTR mode of operation discussed in Section 2.5. Suppose that the block cipher had the following weakness: Given $C, F_K(C+1), F_K(C+2)$, it is possible to compute $F_K(C+3)$. Then clearly the encryption scheme is not secure, because if an adversary happens to know the first two message blocks, it can figure out the third message block from the ciphertext. (It is perfectly reasonable to assume the adversary already knows the first two message blocks. These might, for example, be public header information, or the name of some known recipient.) This means that if CTR mode encryption is to be secure, the block cipher must have the property that given $C, F_K(C+1), F_K(C+2)$, it is computationally infeasible to compute $F_K(C+3)$. Let us call this property SP1, for “security property one”.

Of course, anyone who knows the key K can easily compute $F_K(C+3)$ given $C, F_K(C+1), F_K(C+2)$. And it is hard to think how one can do it without knowing the key. But there is no guarantee that someone *cannot* do this without knowing the key. That is, confidence in the security of F against key recovery does *not* imply that SP1 is true.

This phenomenon continues. As we see more usages of ciphers, we build up a longer and longer list of security properties SP1, SP2, SP3, ... that are necessary for the security of some block cipher based application.

Furthermore, even if SP1 is true, CTR mode encryption may still be weak. SP1 is not *sufficient* to guarantee the security of CTR mode encryption. Similarly with other security properties that one might naively come up with.

This long list of necessary but not sufficient properties is no way to treat security. What we need is *one single* “MASTER” property of a block cipher which, if met, *guarantees* security of *lots of* natural usages of the cipher.

A good example to convince oneself that security against key recovery is not enough is to consider the block cipher $E: \{0, 1\}^k \times \{0, 1\}^n \rightarrow \{0, 1\}^n$ defined for all keys $K \in \{0, 1\}^k$ and plaintexts $x \in \{0, 1\}^n$ by $F(K, x) = x$. That is, each instance F_K of the block cipher is the identity function. Is this a “good” block cipher? Surely not. Yet, it is exceedingly secure against key-recovery. Indeed, given any number of input-output examples of F_K , an adversary cannot even test whether a given key is the one in use.

This might seem like an artificial example. Many people, on seeing this, respond by saying: “But, clearly, DES and AES are *not* designed like this.” True. But that is missing the point. The point is that security against key-recovery *alone* does not make a “good” block cipher. We must seek a better notion of security. Chapter 3 on pseudorandom functions does this.

2.8 Exercises and Problems

Exercise 2.1 Show that for all $K \in \{0, 1\}^{56}$ and all $x \in \{0, 1\}^{64}$

$$\text{DES}_K(x) = \overline{\text{DES}_{\overline{K}}(\overline{x})}.$$

This is called the key-complementation property of DES.

Exercise 2.2 Explain how to use the key-complementation property of DES to speed up exhaustive key search by about a factor of two. Explain any assumptions that you make.

Exercise 2.3 Find a key K such that $\text{DES}_K(\cdot) = \text{DES}_K^{-1}(\cdot)$. Such a key is sometimes called a “weak” key.

Exercise 2.4 As with AES, suppose we are working in the finite field with 2^8 elements, representing field points using the irreducible polynomial $m(x) = x^8 + x^4 + x^3 + x + 1$. Compute the byte that is the result of multiplying bytes:

$$\{e1\} \cdot \{05\}$$

Exercise 2.5 For AES, we have given two different descriptions of *mix-cols*: one using matrix multiplication (in $\text{GF}(2^8)$) and one based on multiplying by a fixed polynomial $c(x)$ modulo a second fixed polynomial, $d(x) = x^4 + 1$. Show that these two methods are equivalent.

Exercise 2.6 Verify that the matrix used for *mix-cols* has as its inverse the matrix

$$\begin{pmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{pmatrix}$$

Explain why all entries in this matrix begin with a zero-byte.

Exercise 2.7 How many different permutations are there from 128 bits to 128 bits? How many different functions are there from 128 bits to 128 bits?

Exercise 2.8 Upper and lower bound, as best you can, the probability that a random function from 128 bits to 128 bits is actually a permutation.

Problem 2.1 Without consulting any of the numerous public-domain implementations available, implement AES, on your own, from the spec or from the description provided by this chapter. Then test your implementation according to the test vectors provided in the AES documentation.

Problem 2.2 Justify and then refute (both) the following proposition: enciphering under AES can be implemented faster than deciphering.

Chapter 3

PSEUDORANDOM FUNCTIONS

Pseudorandom functions (PRFs) and their cousins, pseudorandom permutations (PRPs), figure as central tools in the design of protocols, especially those for shared-key cryptography. At one level, PRFs and PRPs can be used to model block ciphers, and they thereby enable the security analysis of protocols based on block ciphers. But PRFs and PRPs are also a wonderful conceptual starting point in contexts where block ciphers don't quite fit the bill because of their fixed block-length. So in this chapter we will introduce PRFs and PRPs and investigate their basic properties.

3.1 Function families

A *function family* is a map $F: \text{Keys}(F) \times \text{Dom}(F) \rightarrow \text{Range}(F)$. Here $\text{Keys}(F)$ is the set of keys of F ; $\text{Dom}(F)$ is the domain of F ; and $\text{Range}(F)$ is the range of F . The two-input function F takes a key K and input X to return a point Y we denote by $F(K, X)$. The domain and range of F are nonempty sets of strings. For any key $K \in \text{Keys}(F)$ we define the map $F_K: \text{Dom}(F) \rightarrow \text{Range}(F)$ by $F_K(X) = F(K, X)$. We call the function F_K an *instance* of function family F . Thus, F specifies a collection of maps, one for each key. That's why we call F a function *family* (or a *family of functions* or just a *family*).

Usually $\text{Keys}(F) = \{0, 1\}^k$ for some integer k , the *key length*. Often $\text{Dom}(F) = \{0, 1\}^\ell$ and $\text{Range}(F) = \{0, 1\}^L$ for some integers $\ell, L \geq 1$.

There is some probability distribution on the set of keys $\text{Keys}(F)$. When $\text{Keys}(F)$ is a finite set, this distribution will be the uniform distribution. That is, when $\text{Keys}(F) = \{0, 1\}^k$ we shall draw a random k -bit string as a key. We denote by $K \stackrel{R}{\leftarrow} \text{Keys}(F)$ the operation of selecting a random string from $\text{Keys}(F)$ and naming it K . We denote by $f \stackrel{R}{\leftarrow} F$ the operation: $K \stackrel{R}{\leftarrow} \text{Keys}(F)$; $f \leftarrow F_K$. In other words, let f be the function F_K where K is a randomly chosen key. We are interested in the input-output behavior of this randomly chosen instance of the family.

A *permutation* on strings is a map whose domain and range are the same set, and the map is a length-preserving bijection on this set. That is, a map $\pi: D \rightarrow D$ is a permutation if $|\pi(x)| = |x|$ for all $x \in D$ and also π is one-to-one and onto. We say that F is a family of permutations if $\text{Dom}(F) = \text{Range}(F)$ and each F_K is a permutation on this common set.

Example 3.1 A block cipher is a family of permutations. For example, DES is a family of permutations with $\text{Keys}(\text{DES}) = \{0, 1\}^{56}$ and $\text{Dom}(\text{DES}) = \{0, 1\}^{64}$ and $\text{Range}(\text{DES}) = \{0, 1\}^{64}$. Here $k = 56$ and $\ell = L = 64$. Similarly AES is a family of permutations with $\text{Keys}(\text{AES}) = \{0, 1\}^{128}$ (when “AES” refers to “AES128”) and $\text{Dom}(\text{AES}) = \{0, 1\}^{128}$ and $\text{Range}(\text{AES}) = \{0, 1\}^{128}$. Here $k = 128$ and $\ell = L = 128$.

■

3.2 Random functions and permutations

Let $D, R \subseteq \{0, 1\}^*$ be finite nonempty sets and let $\ell, L \geq 1$ be integers. There are two function families that we fix. One is $\text{Rand}(D, R)$, the family of all functions of D to R . The other is $\text{Perm}(D)$, the family of all permutations on D . For compactness of notation we let $\text{Rand}(\ell, L)$, $\text{Rand}(\ell)$, and $\text{Perm}(\ell)$ denote $\text{Rand}(D, R)$, $\text{Rand}(D, D)$, and $\text{Perm}(D)$, where $D = \{0, 1\}^\ell$ and $R = \{0, 1\}^L$.

What are these families? The family $\text{Rand}(D, R)$ has domain D and range R , while the family $\text{Perm}(D)$ has domain and range D . The set of instances of $\text{Rand}(D, R)$ is the set of all functions mapping D to R , while the set of instances of $\text{Perm}(D)$ is the set of all permutations on D . The key describing any particular instance function is simply a description of this instance function in some canonical notation. For example, order the domain D lexicographically as X_1, X_2, \dots , and then let the key for a function f be the list of values $(f(X_1), f(X_2), \dots)$. The key-space of $\text{Rand}(D, R)$ is simply the set of all these keys, under the uniform distribution.

Let us illustrate in more detail for the cases in which we are most interested. The key of a function in $\text{Rand}(\ell, L)$ is simply a list of all the output values of the function as its input ranges over $\{0, 1\}^\ell$. Thus

$$\text{Keys}(\text{Rand}(\ell, L)) = \{ (Y_1, \dots, Y_{2^\ell}) : Y_1, \dots, Y_{2^\ell} \in \{0, 1\}^L \}$$

is the set of all sequences of length 2^ℓ in which each entry of a sequence is an L -bit string. For any $x \in \{0, 1\}^\ell$ we interpret X as an integer in the range $\{1, \dots, 2^\ell\}$ and set

$$\text{Rand}(\ell, L)((Y_1, \dots, Y_{2^\ell}), X) = Y_X .$$

Notice that the key space is very large; it has size 2^{L2^ℓ} . There is a key for every function of ℓ -bits to L -bits, and this is the number of such functions. The key space is equipped with the uniform distribution, so that $f \stackrel{R}{\leftarrow} \text{Rand}(\ell, L)$ is the operation of picking a random function of ℓ -bits to L -bits.

On the other hand, for $\text{Perm}(\ell)$, the key space is

$$\text{Keys}(\text{Perm}(\ell)) = \{(Y_1, \dots, Y_{2^\ell}) : Y_1, \dots, Y_{2^\ell} \in \{0, 1\}^\ell \text{ and } Y_1, \dots, Y_{2^\ell} \text{ are all distinct}\}.$$

For any $X \in \{0, 1\}^\ell$ we interpret X as an integer in the range $\{1, \dots, 2^\ell\}$ and set

$$\text{Perm}(\ell)((Y_1, \dots, Y_{2^\ell}), X) = Y_X.$$

The key space is again equipped with the uniform distribution, so that $f \stackrel{R}{\leftarrow} \text{Perm}(\ell)$ is the operation of picking a random permutation on $\{0, 1\}^\ell$. In other words, all the possible permutations on $\{0, 1\}^\ell$ are equally likely.

Example 3.2 We exemplify $\text{Rand}(3,2)$, meaning $\ell = 3$ and $L = 2$. The domain is $\{0, 1\}^3$ and the range is $\{0, 1\}^2$. An example instance f of the family is illustrated below via its input-output table:

x	000	001	010	011	100	101	110	111
$f(x)$	10	11	01	11	10	00	00	10

The key corresponding to this particular function is

$$(10, 11, 01, 11, 10, 00, 00, 10).$$

The key-space of $\text{Rand}(3,2)$ is the set of all such sequences, meaning the set of all 8-tuples each component of which is a two bit string. There are

$$2^{2 \cdot 2^3} = 2^{16} = 65,536$$

such tuples, so this is the size of the key-space. ■

We will hardly ever actually think about these families in terms of this formalism. Indeed, it is worth pausing here to see how to think about them more intuitively, because they are important objects.

We will consider settings in which you have black-box access to a function g . This means that there is a box to which you can give any value X of your choice (provided X is in the domain of g), and box gives you back $g(X)$. But you can't "look inside" the box; your only interface to it is the one we have specified. A random function $g: \{0, 1\}^\ell \rightarrow \{0, 1\}^L$ being placed in this box corresponds to the following. Each time you give the box an input, you get back a random L -bit string, with the sole constraint that if you twice give the box the same input X , it will be consistent, returning both times the same output $g(X)$. In other words, a random function of ℓ -bits to L -bits can be thought of as a box which given any input $X \in \{0, 1\}^\ell$ 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 the function in the form of a table T where $T[X]$ holds the value of the function

at X . Initially, the table is empty. The program processes an input $X \in \{0, 1\}^\ell$ as follows:

```

If  $T[X]$  is not yet defined then
  Flip coins to determine a string  $Y \in \{0, 1\}^L$  and set  $T[X] \leftarrow Y$ 
Return  $T[X]$ 

```

The answer 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.

One must remember that the term “random function” is misleading. It might lead one to think that certain functions are “random” and others are not. (For example, maybe the constant function which always returns 0^L is not random, but a function with many different range values is random.) This is not right. The randomness of the function refers to the way it was chosen, not to an attribute of the selected function itself. When you choose a function at random, the constant function is just as likely to appear as any other function. It makes no sense to talk of the randomness of an individual function; the term “random function” just means a function chosen at random.

Example 3.3 Let’s do some simple probabilistic computations to understand random functions. Fix $X \in \{0, 1\}^\ell$ and $Y \in \{0, 1\}^L$. Then:

$$\Pr \left[f \stackrel{R}{\leftarrow} \text{Rand}(\ell, L) : f(X) = Y \right] = 2^{-L} .$$

Notice that the probability doesn’t depend on ℓ . Nor does it depend on the values of X, Y .

Now fix $X_1, X_2 \in \{0, 1\}^\ell$ and $Y \in \{0, 1\}^L$. Then:

$$\Pr \left[f \stackrel{R}{\leftarrow} \text{Rand}(\ell, L) : f(X_1) = f(X_2) = Y \right] = \begin{cases} 2^{-2L} & \text{if } X_1 \neq X_2 \\ 2^{-L} & \text{if } X_1 = X_2 \end{cases}$$

This illustrates *independence*. Finally fix $X_1, X_2 \in \{0, 1\}^\ell$ and $Y \in \{0, 1\}^L$. Then:

$$\Pr \left[f \stackrel{R}{\leftarrow} \text{Rand}(\ell, L) : f(X_1) \oplus f(X_2) = Y \right] = \begin{cases} 2^{-L} & \text{if } X_1 \neq X_2 \\ 0 & \text{if } X_1 = X_2 \text{ and } Y \neq 0^L \\ 1 & \text{if } X_1 = X_2 \text{ and } Y = 0^L \end{cases}$$

■

3.3 Pseudorandom functions

A pseudorandom function is a family of functions with the property that the input-output behavior of a random instance of the family is “computationally indistin-

guishable” from that of a random function. Someone who has only black-box access to a function, meaning can only feed it inputs and get outputs, has a hard time telling whether the function in question is a random instance of the family in question or a random function. The purpose of this section is to arrive at a suitable definition of this notion. Later we will look at motivation and applications.

We fix a family of functions $F: \text{Keys}(F) \times D \rightarrow R$. (You may want to think $\text{Keys}(F) = \{0, 1\}^k$, $D = \{0, 1\}^\ell$ and $R = \{0, 1\}^L$ for some integers $k, \ell, L \geq 1$.) Imagine that you are in a room which contains a terminal connected to a computer outside your room. You can type something into your terminal and send it out, and an answer will come back. The allowed questions you can type must be strings from the domain D , and the answers you get back will be strings from the range R . The computer outside your room implements a function $g: D \rightarrow R$, so that whenever you type a value X you get back $g(X)$. However, your only access to g is via this interface, so the only thing you can see is the input-output behavior of g . We consider two different ways in which g will be chosen, giving rise to two different “worlds.”

World 0: The function g is drawn at random from $\text{Rand}(D, R)$, namely via $g \stackrel{R}{\leftarrow} \text{Rand}(D, R)$. (So g is just a random function of D to R .)

World 1: The function g is drawn at random from F , namely $g \stackrel{R}{\leftarrow} F$. (This means that a key is chosen via $K \stackrel{R}{\leftarrow} \text{Keys}(F)$ and then g is set to F_K .)

You are not told which of the two worlds was chosen. The choice of world, and of the corresponding function g , is made before you enter the room, meaning before you start typing questions. Once made, however, these choices are fixed until your “session” is over. Your job is to discover which world you are in. To do this, the only resource available to you is your link enabling you to provide values X and get back $g(X)$. After trying some number of values of your choice, you must make a decision regarding which world you are in. The quality of pseudorandom family F can be thought of as measured by the difficulty of telling, in the above game, whether you are in World 0 or in World 1.

Intuitively, the game just models some way of “using” the function g in an application like an encryption scheme. If it is not possible to distinguish the input-output behavior of a random instance 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 we warn that 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.

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. We write A^g to mean that distinguisher A is being given oracle access to function g . Intuitively, a family is pseudorandom if the probability that the distinguisher says 1 is roughly the same regardless of which world it is in. We capture this mathematically below. Further explanations follow the definition.

Definition 3.4 Let $F: \text{Keys}(F) \times D \rightarrow R$ be a family of functions, and let A be an algorithm that takes an oracle for a function $g: D \rightarrow R$, and returns a bit. We consider two experiments:

$$\begin{array}{l|l} \text{Experiment } \mathbf{Exmt}_F^{\text{prf-1}}(A) & \text{Experiment } \mathbf{Exmt}_F^{\text{prf-0}}(A) \\ K \stackrel{R}{\leftarrow} \text{Keys}(F) & g \stackrel{R}{\leftarrow} \text{Rand}(D,R) \\ b \leftarrow A^{F_K} & b \leftarrow A^g \\ \text{Return } b & \text{Return } b \end{array}$$

The *prf-advantage* of A is defined as

$$\mathbf{Adv}_F^{\text{prf}}(A) = \Pr[\mathbf{Exmt}_F^{\text{prf-1}}(A) = 1] - \Pr[\mathbf{Exmt}_F^{\text{prf-0}}(A) = 1] .$$

For any t, q, μ we define the *prf-advantage* of F

$$\mathbf{Adv}_F^{\text{prf}}(t, q, \mu) = \max_A \{ \mathbf{Adv}_F^{\text{prf}}(A) \}$$

where the maximum is over all A having time-complexity t and making at most q oracle queries, the sum of the lengths of these queries being at most μ bits. ■

The algorithm A models the person we were imagining in our room, trying to determine which world he or she was in by typing queries to the function g via a computer. In the formalization, the person is an algorithm, meaning a piece of code. We formalize the ability to query g as giving A an oracle which takes input any string $X \in D$ and returns $g(X)$. Algorithm A can decide which queries to make, perhaps based on answers received to previous queries. Eventually, it outputs a bit b which is its decision as to which world it is in. Outputting the bit “1” means that A “thinks” it is in world 1; outputting the bit “0” means that A thinks it is in world 0.

It should be noted that the family F is public. The adversary A , and anyone else, knows the description of the family and is capable, given values K, X , of computing $F(K, X)$.

The worlds are captured by what we call “experiments.” The first experiment picks a random instance F_K of family F and then runs adversary A with oracle $g = F_K$. Adversary A interacts with its oracle, querying it and getting back answers, and eventually outputs a “guess” bit. The experiment returns the same bit. The second experiment picks a random function $g: D \rightarrow R$ and runs A with this as oracle, again returning A ’s guess bit. Each experiment has a certain probability of returning 1. The probability is taken over the random choices made in the experiment. Thus,

for the first experiment, the probability is over the choice of K and any random choices that A might make, for A is allowed to be a randomized algorithm. In the second experiment, the probability is over the random choice of g and any random choices that A makes. These two probabilities should be evaluated separately; the two experiments are completely different.

To see how well A does at determining which world it is in, we look at the difference in the probabilities that the two experiments return 1. If A is doing a good job at telling which world it is in, it would return 1 more often in the first experiment than in the second. So the difference is a measure of how well A is doing. We call this measure the prf-advantage of A . Think of it as the probability that A “breaks” the scheme F , with “break” interpreted in a specific, technical way based on the definition.

Different distinguishers will have different advantages. There are two reasons why one distinguisher may achieve a greater advantage than another. One is that it is more “clever” in the questions it asks and the way it processes the replies to determine its output. The other is simply that it asks more questions, or spends more time processing the replies. Indeed, we expect that as you see more and more input-output examples of g , or spend more computing time, your ability to tell which world you are in should go up. The “security” of family F must thus be measured as a function of the resources allowed to the attacker. We want to know, for any given resource limitations, what is the prf-advantage achieved by the most “clever” distinguisher amongst all those who are restricted to the given resource limits. We associate to the family F a prf-advantage function which on input any values of the resource parameters returns the maximum prf-advantage that an adversary restricted to those resources could obtain. Think of it as the maximum possible achievable probability of “breaking” the scheme F if an attacker is restricted to the given resources.

The choice of resources to consider can vary. In this case we have chosen to measure the time-complexity t of A , the number of queries q it makes, and the total length μ of these queries. We associate to the family F an *advantage function* which on input a particular choice of these resource parameters returns the maximum possible advantage that could be obtained by a distinguisher restricted in resource usage by the given parameters. Put another way, it is the advantage of the “cleverest” or “best” distinguisher restricted to the given resources. The advantage function of F captures the security of F as a PRF.

Let us now explain the resources, and some important conventions underlying their measurement, in more detail. The first resource is the time-complexity of A . To make sense of this we first need to fix a model of computation. We fix some RAM model. Think of it as the model used in your algorithms courses, often implicitly, so that you could measure the running time. However, we adopt the convention that the *time-complexity* of A refers not just to the running time of A , but to the maximum of the running times of the two experiments in the definition, plus the size of the code of A . In measuring the running time of the first experiment, we

must count the time to choose the key K at random, and the time to compute the value $F_K(x)$ for any query x made by A to its oracle. In measuring the running time of the second experiment, we count the time to choose the random function g in a dynamic way, meaning we count the cost of maintaining a table of values of the form $(X, g(X))$. Entries are added to the table as g makes queries. A new entry is made by picking the output value at random.

The number of queries made by A captures the number of input-output examples it sees. In general, not all strings in the domain must have the same length, and hence we also measure the sum of the lengths of all queries made.

There is one feature of the above parameterization about which everyone asks. Suppose that F has key-length k . Obviously, the key length is a fundamental determinant of security: larger key length will typically mean more security. Yet, the key length k does not appear explicitly in the advantage function $\mathbf{Adv}_F^{\text{prf}}(t, q, \mu)$. Why is this? The advantage function is in fact a function of k , but without knowing more about F it is difficult to know what kind of function. The truth is that the key length itself does not matter: what matters is just the advantage a distinguisher can obtain. In a well-designed block cipher, $\mathbf{Adv}_F^{\text{prf}}(t, q, \mu)$ should be about $t/2^k$. But that is really an ideal; in practice we should not assume ciphers are this good.

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 resources. A distinguisher can use whatever means desired to distinguish the function as long as it stays within the specified resource bounds.

What do we mean by a “secure” PRF? Definition 3.4 does not have any explicit condition or statement regarding when F should be considered “secure.” It only associates to F a prf-advantage function. Intuitively, F is “secure” if the value of the advantage function is “low” for “practical” values of the input parameters. This is, of course, not formal. It is possible to formalize the notion of a secure PRF using a complexity theoretic framework; one would say that the advantage of any adversary whose resources are polynomially-bounded is negligible. This requires an extension of the model to consider a security parameter in terms of which asymptotic estimates can be made. We will discuss this in more depth later, but for now we stick to a framework where the notion of what exactly is “secure” is not something binary. One reason is that this better reflects real life. In real life, security is not some absolute or boolean attribute; security is a function of the resources invested by an attacker. All modern cryptographic systems are breakable in principle; it is just a question of how long it takes.

This is our first example of a cryptographic definition, and it is worth spending time to study and understand it. We will encounter many more as we go along. Towards this end let us summarize the main features of the definitional framework as we will see them arise later. First, there are *experiments*, involving an adversary. Then, there is some *advantage* function associated to an adversary which returns the probability that the adversary in question “breaks” the scheme. Finally, there is an advantage function associated to the cryptographic protocol itself, taking as input

resource parameters and returning the maximum possible probability of “breaking” the scheme if the attacker is restricted to those resource parameters. These three components will be present in all definitions. What varies is the experiments; this is here that we pin down how we measure security.

3.4 Pseudorandom permutations

Recall that a block cipher F is a family of permutations: each instance F_K of the family is a permutation. With the intent of modeling block ciphers we introduce the notion of a pseudorandom permutation. We proceed exactly as above, but replace $\text{Rand}(D,R)$ with $\text{Perm}(D)$.

In this setting, there are two kinds of attacks that one can consider. One, as before, is that the adversary gets an oracle for the function g being tested. However when g is a permutation one can also consider the case where the adversary gets, in addition, an oracle for g^{-1} . We consider these settings in turn. The first is the setting of chosen-plaintext attacks while the second is the setting of chosen-ciphertext attacks.

3.4.1 PRP under CPA

We fix a family of functions $F: \text{Keys}(F) \times D \rightarrow D$. (You may want to think $\text{Keys}(F) = \{0,1\}^k$ and $D = \{0,1\}^\ell$, since this is the most common case. We do not mandate that F be a family of permutations although again this is the most common case.) As before, we consider an adversary A that is placed in a room where it has oracle access to a function g chosen in one of two ways.

World 0: The function g is drawn at random from $\text{Perm}(D)$, namely via $g \xleftarrow{R} \text{Perm}(D)$. (So g is just a random permutation on D .)

World 1: The function g is drawn at random from F , namely $g \xleftarrow{R} F$. (This means that a key is chosen via $K \xleftarrow{R} \text{Keys}(F)$ and then g is set to F_K .)

Notice that World 1 is the same in the PRF setting, but World 0 has changed. As before the task facing the adversary A is to determine in which world it was placed based on the input-output behavior of g .

Definition 3.5 Let $F: \text{Keys}(F) \times D \rightarrow D$ be a family of functions, and let A be an algorithm that takes an oracle for a function $g: D \rightarrow D$, and returns a bit. We consider two experiments:

Experiment $\text{Exmt}_F^{\text{prp-cpa-1}}(A)$ $K \xleftarrow{R} \text{Keys}(F)$ $b \leftarrow A^{F_K}$ Return b	Experiment $\text{Exmt}_F^{\text{prp-cpa-0}}(A)$ $g \xleftarrow{R} \text{Perm}(D)$ $b \leftarrow A^g$ Return b
---	---

The *prp-cpa-advantage* of A is defined as

$$\mathbf{Adv}_F^{\text{prp-cpa}}(A) = \Pr \left[\mathbf{Exmt}_F^{\text{prp-cpa-1}}(A) = 1 \right] - \Pr \left[\mathbf{Exmt}_F^{\text{prp-cpa-0}}(A) = 1 \right].$$

For any t, q, μ we define the *prp-cpa-advantage* of F via

$$\mathbf{Adv}_F^{\text{prp-cpa}}(t, q, \mu) = \max_A \{ \mathbf{Adv}_F^{\text{prp-cpa}}(A) \}$$

where the maximum is over all A having time-complexity t and making at most q oracle queries, the sum of the lengths of these queries being at most μ bits. ■

The intuition is similar to that for Definition 3.4. The difference is that here the “ideal” object that F is being compared with is no longer the family of random functions, but rather the family of random permutations.

Experiment $\mathbf{Exmt}_F^{\text{prp-cpa-1}}(A)$ is actually identical to $\mathbf{Exmt}_F^{\text{prf-1}}(A)$. The probability is over the random choice of key K and also over the coin tosses of A if the latter happens to be randomized. The experiment returns the same bit that A returns. In Experiment $\mathbf{Exmt}_F^{\text{prp-cpa-0}}(A)$, a permutation $g: \{0, 1\}^\ell \rightarrow \{0, 1\}^\ell$ is chosen at random, and the result bit of A 's computation with oracle g is returned. The probability is over the choice of g and the coins of A if any. As before, the measure of how well A did at telling the two worlds apart, which we call the *prp-cpa-advantage* of A , is the difference between the probabilities that the experiments return 1.

Conventions regarding resource measures also remain the same as before. Informally, a family F is a secure PRP under CPA if $\mathbf{Adv}_F^{\text{prp-cpa}}(t, q, \mu)$ is “small” for “practical” values of the resource parameters.

3.4.2 PRP under CCA

We fix a family of permutations $F: \text{Keys}(F) \times D \rightarrow D$. (You may want to think $\text{Keys}(F) = \{0, 1\}^k$ and $D = \{0, 1\}^\ell$, since this is the most common case. This time, we do mandate that F be a family of permutations.) As before, we consider an adversary A that is placed in a room, but now it has oracle access to two functions, g and its inverse g^{-1} . The manner in which g is chosen is the same as in the CPA case, and once g is chosen, g^{-1} is automatically defined, so we do not have to say how it is chosen.

World 0: The function g is drawn at random from $\text{Perm}(D)$, namely via $g \xleftarrow{R} \text{Perm}(D)$. (So g is just a random permutation on D .)

World 1: The function g is drawn at random from F , namely $g \xleftarrow{R} F$. (This means that a key is chosen via $K \xleftarrow{R} \text{Keys}(F)$ and then g is set to F_K .)

In World 1, $g^{-1} = F_K^{-1}$ is the inverse of the chosen instance, while in World 0 it is the inverse of the chosen random permutation. As before the task facing the adversary A is to determine in which world it was placed based on the input-output behavior of its oracles.

Definition 3.6 Let $F: \text{Keys}(F) \times D \rightarrow D$ be a family of permutations, and let A be an algorithm that takes an oracle for a function $g: D \rightarrow D$, and also an oracle for the function $g^{-1}: D \rightarrow D$, and returns a bit. We consider two experiments:

$$\begin{array}{l|l} \text{Experiment } \mathbf{Exmt}_F^{\text{prp-cca-1}}(A) & \text{Experiment } \mathbf{Exmt}_F^{\text{prp-cca-0}}(A) \\ \hline K \stackrel{R}{\leftarrow} \text{Keys}(F) & g \stackrel{R}{\leftarrow} \text{Perm}(D) \\ b \leftarrow A^{F_K, F_K^{-1}} & b \leftarrow A^{g, g^{-1}} \\ \text{Return } b & \text{Return } b \end{array}$$

The *prp-cca-advantage* of A is defined as

$$\mathbf{Adv}_F^{\text{prp-cca}}(A) = \Pr \left[\mathbf{Exmt}_F^{\text{prp-cca-1}}(A) = 1 \right] - \Pr \left[\mathbf{Exmt}_F^{\text{prp-cca-0}}(A) = 1 \right] .$$

For any $t, q_e, \mu_e, q_d, \mu_d$ we define the *prp-cca-advantage* of F via

$$\mathbf{Adv}_F^{\text{prp-cca}}(t, q_e, \mu_e, q_d, \mu_d) = \max_A \{ \mathbf{Adv}_F^{\text{prp-cca}}(A) \}$$

where the maximum is over all A having time-complexity t , making at most q_e queries to the g oracle, the sum of the lengths of these queries being at most μ_e bits, and also making at most q_d queries to the g^{-1} oracle, the sum of the lengths of these queries being at most μ_d bits, ■

The intuition is similar to that for Definition 3.4. The difference is that here the adversary has more power: not only can it query g , but it can directly query g^{-1} . Conventions regarding resource measures also remain the same as before. However, we add some resource parameters. Specifically, since there are now two oracles, we count separately the number of queries, and total length of these queries, for each. Informally, a family F is a secure PRP under CCA if $\mathbf{Adv}_F^{\text{prp-cca}}(t, q_e, \mu_e, q_d, \mu_d)$ is “small” for “practical” values of the resource parameters.

3.4.3 Relations between the notions

If an adversary above does not query g^{-1} , the latter oracle may as well not be there, and the adversary is effectively mounting a chosen-plaintext attack. Thus we have the following:

Proposition 3.7 Let $F: \text{Keys}(F) \times D \rightarrow D$ be a family of permutations. Then

$$\mathbf{Adv}_F^{\text{prp-cpa}}(t, q, \mu) = \mathbf{Adv}_F^{\text{prp-cca}}(t, q, \mu, 0, 0)$$

for any t, q, μ . ■

3.5 Usage of PRFs and PRPs

We discuss some motivation for these notions of security.

3.5.1 The shared-random-function model

In symmetric (ie. shared-key) cryptography, Alice and Bob 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—one that takes the form of 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 in practice because the description of a random function is 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 $x \in \{0, 1\}^\ell$ and in one step be returned $f(x)$.

It turns out that the shared-random-function model is a very convenient one 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.

As an example, consider the CTR mode of operation discussed in Section 2.5.3. Consider the version where the initial vector is a counter. Consider replacing every invocation of E_K with an invocation of the random function f . (Assume $\ell = L$.) In that case, the mode of operation turns into the one-time-pad cryptosystem. The shared random key is just the random function f . As we have discussed, this is well known to meet a strong and well-defined notion of security. So, in the shared-random-function model, CTR mode is “good”. Well, it would be, if we had yet defined what “good” means!

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 or permutation 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 Alice and Bob, 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 restricted in resource usage.

In other words, instances of PRFs can be used in place of random functions in shared-key schemes. The definition of a PRF is crafted to make this possible for as wide a range of applications as possible. An instance of 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 resort to the formal definition of a pseudorandom function family and prove the security of our construct based on it. We will see how to do this later.

In this context we stress one important point. The security of a PRF relies 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.

Pseudorandom functions are an intriguing notion and a powerful tool that enable the following design paradigm. When 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 your scheme should still be secure.

3.5.2 Modeling block ciphers

One of the primary motivations for the notions of pseudorandom functions (PRFs) and pseudorandom permutations (PRPs) is to model block ciphers and thereby enable the security analysis of protocols that use block ciphers.

As discussed in Section 2.7, classically the security of DES or other block ciphers has been looked at only with regard to key recovery. That is, analysis of a block cipher F has focused on the following question: Given some number of input-output examples

$$(X_1, F_K(X_1)), \dots, (X_q, F_K(X_q))$$

where K is a random, unknown key, how hard is it to find K ? The block cipher is taken as “secure” if the resources required to recover the key are prohibitive. Yet, as we saw, even a cursory glance at common block cipher usages shows that hardness of key recovery is not *sufficient* for security. We had discussed wanting a “MASTER” security property of block ciphers under which natural usages of block ciphers could be proven secure. We suggest that this “MASTER” property is that the block cipher be a secure PRP, under either CPA or CCA.

We cannot prove that specific block ciphers have this property. The best we can do is assume they do, and then go on to use them. For quantitative security assessments, we would make specific conjectures about the advantage functions of various block ciphers. For example we might conjecture something like:

$$\mathbf{Adv}_{\text{DES}}^{\text{prp-cpa}}(t, q, 64q) \leq c_1 \cdot \frac{t/T_{\text{DES}}}{2^{55}} + c_2 \cdot \frac{q}{2^{40}}$$

Here T_{DES} is the time to do one DES computation on our fixed RAM model of computation, and c_1, c_2 are some constants. In other words, we are conjecturing that the best attacks are either exhaustive key search or linear cryptanalysis. We

might be bolder with regard to AES and conjecture something like

$$\mathbf{Adv}_{\text{AES}}^{\text{prp-cpa}}(t, q, 128q) \leq c_1 \cdot \frac{t/T_{\text{AES}}}{2^{128}} + c_2 \cdot \frac{q}{2^{128}}.$$

We could also make similar conjectures regarding the strength of block ciphers as PRPs under CCA rather than CPA.

More interesting is $\mathbf{Adv}_{\text{DES}}^{\text{prf}}(t, q)$. Here we cannot do better than assume that

$$\begin{aligned} \mathbf{Adv}_{\text{DES}}^{\text{prf}}(t, q, 64q) &\leq c_1 \cdot \frac{t/T_{\text{DES}}}{2^{55}} + \frac{q^2}{2^{64}} \\ \mathbf{Adv}_{\text{AES}}^{\text{prf}}(t, q, 128q) &\leq c_1 \cdot \frac{t/T_{\text{AES}}}{2^{128}} + \frac{q^2}{2^{128}}. \end{aligned}$$

This is due to the birthday attack discussed later. The second term in each formula arises simply because the object under consideration is a family of permutations.

We stress that these are all conjectures. There could exist highly effective attacks that break DES or AES as a PRF without recovering the key. So far, we do not know of any such attacks, but the amount of cryptanalytic effort that has focused on this goal is small. Certainly, to assume that a block cipher is a PRF is a much stronger assumption than that it is secure against key recovery. Nonetheless, the motivation and arguments we have outlined in favor of the PRF assumption stay, and our view is that if a block cipher is broken as a PRF then it should be considered insecure, and a replacement should be sought.

3.6 Example Attacks

Let us illustrate the models by providing adversaries that attack different function families in these models.

Example 3.8 We define a family of functions $F: \{0, 1\}^k \times \{0, 1\}^\ell \rightarrow \{0, 1\}^L$ as follows. We let $k = L\ell$ and view a k -bit key K as specifying an L row by ℓ column matrix of bits. (To be concrete, assume the first L bits of K specify the first column of the matrix, the next L bits of K specify the second column of the matrix, and so on.) The input string $X = X[1] \dots X[\ell]$ is viewed as a sequence of bits, and the value of $F(K, x)$ is the corresponding matrix vector product. That is

$$F_K(X) = \begin{bmatrix} K[1, 1] & K[1, 2] & \cdots & K[1, \ell] \\ K[2, 1] & K[2, 2] & \cdots & K[2, \ell] \\ \vdots & & & \vdots \\ K[L, 1] & K[L, 2] & \cdots & K[L, \ell] \end{bmatrix} \cdot \begin{bmatrix} X[1] \\ X[2] \\ \vdots \\ X[\ell] \end{bmatrix} = \begin{bmatrix} Y[1] \\ Y[2] \\ \vdots \\ Y[L] \end{bmatrix}$$

where

$$\begin{aligned}
Y[1] &= K[1, 1] \cdot x[1] \oplus K[1, 2] \cdot x[2] \oplus \dots \oplus K[1, \ell] \cdot x[\ell] \\
Y[2] &= K[2, 1] \cdot x[1] \oplus K[2, 2] \cdot x[2] \oplus \dots \oplus K[2, \ell] \cdot x[\ell] \\
&\vdots = \vdots \\
Y[L] &= K[L, 1] \cdot x[1] \oplus K[L, 2] \cdot x[2] \oplus \dots \oplus K[L, \ell] \cdot x[\ell].
\end{aligned}$$

Here the bits in the matrix are the bits in the key, and arithmetic is modulo two. The question we ask is whether F is a “secure” PRF. We claim that the answer is no. The reason is that one can design an adversary algorithm A that achieves a high advantage (close to 1) in distinguishing between the two worlds.

We observe that for any key K we have $F_K(0^\ell) = 0^L$. This is a weakness since a random function of ℓ -bits to L -bits is very unlikely to return 0^L on input 0^ℓ , and thus this fact can be the basis of a distinguishing adversary. Let us now show how the adversary works. Remember that as per our model it is given an oracle $g: \{0, 1\}^\ell \rightarrow \{0, 1\}^L$ and will output a bit. Our adversary D works as follows:

Adversary D^g

Let $Y \leftarrow g(0^\ell)$

If $Y = 0^L$ then return 1 else return 0

This adversary queries its oracle at the point 0^ℓ , and denotes by Y the ℓ -bit string that is returned. If $y = 0^L$ it bets that g was an instance of the family F , and if $y \neq 0^L$ it bets that g was a random function. Let us now see how well this adversary does. We claim that

$$\begin{aligned}
\Pr \left[\mathbf{Exmt}_F^{\text{prf-1}}(D) = 1 \right] &= 1 \\
\Pr \left[\mathbf{Exmt}_F^{\text{prf-0}}(D) = 1 \right] &= 2^{-L}.
\end{aligned}$$

Why? Look at Experiment $\mathbf{Exmt}_F^{\text{prf-1}}(D)$ as defined in Definition 3.4. Here $g = F_K$ for some K . In that case it is certainly true that $g(0^\ell) = 0^L$ so by the code we wrote for D the latter will return 1. On the other hand look at Experiment $\mathbf{Exmt}_F^{\text{prf-0}}(D)$ as defined in Definition 3.4. Here g is a random function. As we saw in Example 3.3, the probability that $g(0^\ell) = 0^L$ will be 2^{-L} , and hence this is the probability that D will return 1. Now as per Definition 3.4 we subtract to get

$$\begin{aligned}
\mathbf{Adv}_F^{\text{prf}}(D) &= \Pr \left[\mathbf{Exmt}_F^{\text{prf-1}}(D) = 1 \right] - \Pr \left[\mathbf{Exmt}_F^{\text{prf-0}}(D) = 1 \right] \\
&= 1 - 2^{-L}.
\end{aligned}$$

Now let t be the time complexity of D . This is $O(\ell + L)$ plus the time for one computation of F , coming to $O(\ell^2 L)$. The number of queries made by D is just one, and the total length of all queries is l . Thus we have

$$\mathbf{Adv}_F^{\text{prf}}(t, 1, \ell) = \max_A \{ \mathbf{Adv}_F^{\text{prf}}(A) \}$$

$$\begin{aligned} &\geq \mathbf{Adv}_F^{\text{prf}}(D) \\ &= 1 - 2^{-L} . \end{aligned}$$

The first inequality is true because the adversary D is one member of the set of adversaries A over which the maximum is taken, and hence the maximum advantage is at least that attained by D . Our conclusion is that the advantage function of F as a PRF is very high even for very low values of its resource parameter inputs, meaning F is very insecure as a PRF. ■

Example 3.9 . Suppose we are given a secure PRF $F: \{0,1\}^k \times \{0,1\}^\ell \rightarrow \{0,1\}^L$. We want to use F to design a PRF $G: \{0,1\}^k \times \{0,1\}^\ell \rightarrow \{0,1\}^{2L}$. The input length of G is the same as that of F but the output length of G is twice that of F . We suggest the following candidate construction: for every k -bit key K and every ℓ -bit input x

$$G_K(x) = F_K(x) \| F_K(\bar{x}) .$$

Here “ $\|$ ” denotes concatenation of strings, and \bar{x} denotes the bitwise complement of the string x . We ask whether this is a “good” construction. “Good” means that under the assumption that F is a secure PRF, G should be too. However, this is not true. Regardless of the quality of F , the construct G is insecure. Let us demonstrate this.

We want to specify an adversary attacking G . Since an instance of G maps ℓ bits to $2L$ bits, the adversary D will get an oracle for a function g that maps ℓ bits to $2L$ bits. In World 0, g will be chosen as a random function of ℓ bits to $2L$ bits, while in World 1, g will be set to G_K where K is a random k -bit key. The adversary must determine in which world it is placed. Our adversary works as follows:

Adversary D^g

Let $y_1 \leftarrow g(1^\ell)$

Let $y_2 \leftarrow g(0^\ell)$

Parse y_1 as $y_1 = y_{1,1} \| y_{1,2}$ with $|y_{1,1}| = |y_{1,2}| = L$

Parse y_2 as $y_2 = y_{2,1} \| y_{2,2}$ with $|y_{2,1}| = |y_{2,2}| = L$

If $y_{1,1} = y_{2,2}$ then return 1 else return 0

This adversary queries its oracle at the point 1^ℓ to get back y_1 and then queries its oracle at the point 0^ℓ to get back y_2 . Notice that 1^ℓ is the bitwise complement of 0^ℓ . The adversary checks whether the first half of y_1 equals the second half of y_2 , and if so bets that it is in World 1. Let us now see how well this adversary does. We claim that

$$\begin{aligned} \Pr \left[\mathbf{Exmt}_G^{\text{prf-1}}(D) = 1 \right] &= 1 \\ \Pr \left[\mathbf{Exmt}_G^{\text{prf-0}}(D) = 1 \right] &= 2^{-L} . \end{aligned}$$

Why? Look at Experiment $\mathbf{Exmt}_G^{\text{prf-1}}(D)$ as defined in Definition 3.4. Here $g = G_K$ for some K . In that case we have

$$\begin{aligned} G_K(1^\ell) &= F_K(1^\ell) \| F_K(0^\ell) \\ G_K(0^\ell) &= F_K(0^\ell) \| F_K(1^\ell) \end{aligned}$$

by definition of the family G . Notice that the first half of $G_K(1^\ell)$ is the same as the second half of $G_K(0^\ell)$. So D will return 1. On the other hand look at Experiment $\mathbf{Exmt}_G^{\text{prf-0}}(D)$ as defined in Definition 3.4. Here g is a random function. So the values $g(1^\ell)$ and $g(0^\ell)$ are both random and independent $2L$ bit strings. What is the probability that the first half of the first string equals the second half of the second string? It is exactly the probability that two randomly chosen L -bit strings are equal, and this is 2^{-L} . So this is the probability that D will return 1. Now as per Definition 3.4 we subtract to get

$$\begin{aligned} \mathbf{Adv}_G^{\text{prf}}(D) &= \Pr [\mathbf{Exmt}_G^{\text{prf-1}}(D) = 1] - \Pr [\mathbf{Exmt}_G^{\text{prf-0}}(D) = 1] \\ &= 1 - 2^{-L} . \end{aligned}$$

Now let t be the time complexity of D . This is $O(\ell + L)$ plus the time for two computations of G , coming to $O(\ell + L)$ plus the time for four computations of F . The number of queries made by D is two, and the total length of all queries is 2ℓ . Thus we have

$$\begin{aligned} \mathbf{Adv}_G^{\text{prf}}(t, 2, 2\ell) &= \max_A \{ \mathbf{Adv}_G^{\text{prf}}(A) \} \\ &\geq \mathbf{Adv}_G^{\text{prf}}(D) \\ &= 1 - 2^{-L} . \end{aligned}$$

Our conclusion is that the advantage function of G as a PRF is very high even for very low values of its resource parameter inputs, meaning G is very insecure as a PRF. ■

3.7 Security against key recovery

We have mentioned several times that security against key recovery is not sufficient as a notion of security for a block cipher. However it is certainly necessary: if key recovery is easy, the block cipher should be declared insecure. We have indicated that we want to adopt as notion of security for a block cipher the notion of a PRF or a PRP. If this is to be viable, it should be the case that any function family that is insecure under key recovery is also insecure as a PRF or PRP. In this section we verify this simple fact. Doing so will enable us to exercise the method of reductions.

We begin by formalizing security against key recovery. We consider an adversary that, based on input-output examples of an instance F_K of family F , tries to find

K . Its advantage is defined as the probability that it succeeds in finding K . The probability is over the random choice of K , and any random choices of the adversary itself.

We give the adversary oracle access to F_K so that it can obtain input-output examples of its choice. We do not constrain the adversary with regard to the method it uses. This leads to the following definition.

Definition 3.10 Let $F: \text{Keys}(F) \times D \rightarrow R$ be a family of functions, and let B be an algorithm that takes an oracle for a function $g: D \rightarrow R$, and outputs a string. We consider the experiment:

Experiment $\mathbf{Exmt}_F^{\text{kr}}(B)$
 $K \xleftarrow{R} \text{Keys}(F)$
 $K' \leftarrow B^{F_K}$
 If $K = K'$ then return 1 else return 0

The *kr-advantage* of B is defined as

$$\mathbf{Adv}_F^{\text{kr}}(B) = \Pr \left[\mathbf{Exmt}_F^{\text{kr}}(B) = 1 \right].$$

For any t, q, μ the *kr-advantage* of F is defined via

$$\mathbf{Adv}_F^{\text{kr}}(t, q, \mu) = \max_B \{ \mathbf{Adv}_F^{\text{kr}}(B) \}$$

where the maximum is over all B having time-complexity t and making at most q oracle queries, the sum of the lengths of these queries being at most μ bits. ■

This definition has been made general enough to capture all types of key-recovery attacks. Any of the classical attacks such as exhaustive key search, differential cryptanalysis or linear cryptanalysis correspond to different, specific choices of adversary B . They fall in this framework because all have the goal of finding the key K based on some number of input-output examples of an instance F_K of the cipher. To illustrate let us see what are the implications of the classical key-recovery attacks on DES for the value of the key-recovery advantage function of DES. Assuming the exhaustive search attack is always successful based on testing two examples leads to the fact that

$$\mathbf{Adv}_{\text{DES}}^{\text{kr}}(t, 2, 2 \cdot 64) = 1$$

for t being about 2^{55} times the time T_{DES} for one computation of DES. On the other hand, linear cryptanalysis implies that

$$\mathbf{Adv}_{\text{DES}}^{\text{kr}}(t, 2^{43}, 2^{43} \cdot 64) = 1$$

for t being about $2^{43} \cdot T_{\text{DES}}$. This gives us a couple of data points on the curve $\mathbf{Adv}_{\text{DES}}^{\text{kr}}(t, q, ql)$. For a more concrete example, let us look at the key-recovery advantage of the family of Example 3.8.

Example 3.11 Let $F: \{0, 1\}^k \times \{0, 1\}^l \rightarrow \{0, 1\}^L$ be the family of functions from Example 3.8. We saw that its prf-advantage was very high. Let us now compute its kr-advantage. The following adversary B recovers the key. We let e_j be the l -bit binary string having a 1 in position j and zeros everywhere else. We assume that the manner in which the key K defines the matrix is that the first L bits of K form the first column of the matrix, the next L bits of K form the second column of the matrix, and so on.

Adversary B^{F_K}

Let K' be the empty string

For $j = 1, \dots, l$ do

$y_j \leftarrow F_K(e_j)$

$K' \leftarrow K' \| y_j$

EndFor

Return K'

The adversary B invokes its oracle to compute the output of the function on input e_j . The result, y_j , is exactly the j -th column of the matrix associated to the key K . The matrix entries are concatenated to yield K' , which is returned as the key. Since the adversary always finds the key we have

$$\mathbf{Adv}_F^{\text{kr}}(B) = 1.$$

The time-complexity of this adversary is $t = O(l^2 L)$ since it makes $q = l$ calls to its oracle and each computation of F_K takes $O(lL)$ time. Thus

$$\mathbf{Adv}_F^{\text{kr}}(t, l, l^2) = 1.$$

The parameters here should still be considered small: l is 64 or 128, which is small for the number of queries. So F is insecure against key-recovery. Note however that F is less secure as a PRF than against key-recovery: its advantage function as a PRF had a value close to 1 for parameter values much smaller than those above. This leads into our next claim, which says that for any given parameter values, the kr-advantage of a family cannot be significantly more than its prf or prp-cpa advantage. ■

Now we claim that if a block cipher is a secure PRF or PRP then it is also secure against all key-recovery attacks. Put another way, the advantage of F with respect to key recovery cannot be much larger than its advantage as a PRF.

Proposition 3.12 Let $F: \{0, 1\}^k \times \{0, 1\}^l \rightarrow \{0, 1\}^L$ be a family of functions. Then for any t, q with $q < 2^l$ we have

$$\mathbf{Adv}_F^{\text{kr}}(t, q, ql) \leq \mathbf{Adv}_F^{\text{prf}}(t', q + 1, (q + 1)l) + \frac{1}{2^L}, \quad (3.1)$$

and furthermore, if $L = l$, then also

$$\mathbf{Adv}_F^{\text{kr}}(t, q, ql) \leq \mathbf{Adv}_F^{\text{prp-cpa}}(t', q + 1, (q + 1)l) + \frac{1}{2^{L-q}}, \quad (3.2)$$

where we set t' to be t plus the time for one computation of F . ■

The proof introduces the central idea of *reductions*. We will show a transformation $B \mapsto A_B$ of any kr-adversary B into a prf-adversary A_B such that

$$\mathbf{Adv}_F^{\text{kr}}(B) \leq \mathbf{Adv}_F^{\text{prf}}(A_B) + \frac{1}{2^L}$$

and also, if the resources used by B are t, q, ql , then those used by A_B are $t', q + 1, (q + 1)l$. We claim that barring manipulation, this proves the first equation of the claim. Indeed, by taking maximums on both sides, we will be able to get the equation in question, as we will see later.

The problem that adversary A_B is trying to solve is to determine whether its given oracle g is a random instance of F or a random function of l bits to L -bits. The idea behind a reduction is that A_B will run B as a subroutine and use B 's output to solve its own problem.

B is an algorithm that expects to be in a world where it gets an oracle F_K , and it tries to find K via queries to its oracle. For simplicity, first assume that B makes no oracle queries. Now, when A_B runs B , it produces some key K' . A_B can test K' by checking whether $F(K', x)$ agrees with $g(x)$ for some value x . If so, it bets that g was an instance of F , and if not it bets that g was random.

If B does make oracle queries, we must ask how A_B can run B at all. The oracle that B wants is not available. However, B is a piece of code, communicating with its oracle via a prescribed interface. If you start running B , at some point it will output an oracle query, say by writing this to some prescribed memory location, and stop. It awaits an answer, to be provided in another prescribed memory location. When that appears, it continues its execution. When it is done making oracle queries, it will return its output. Now when A_B runs B , it will itself supply the answers to B 's oracle queries. When B stops, having made some query, A will fill in the reply in the prescribed memory location, and let B continue its execution. B does not know the difference between this “simulated” oracle and the real oracle except in so far as it can glean this from the values returned.

The value that B expects in reply to query x is $F_K(x)$. That is not what A_B gives it. Instead, it returns $g(x)$, where g is A_B 's oracle. When A_B is in World 1, $g(x) = F_K(x)$, and so B is functioning as it would in its usual environment, and will return the key K with a probability equal to its kr-advantage. However when A_B is in World 0, g is a random function, and B is getting back values that bear little relation to the ones it is expecting. That does not matter. B is a piece of code that will run to completion and produce some output. When we are in World 0, we have no idea what properties this output will have. But it is some k -bit string, and A_B will test it as indicated above. It will fail the test with high probability as long as the test point x was not one that B queried, and A_B will make sure the latter is true via its choice of x . Let us now proceed to the actual proof.

Proof of Proposition 3.12: We prove the first equation and then briefly indicate how to alter the proof to prove the second equation.

We will show that given any adversary B whose resources are restricted to t, q, ql we can construct an adversary A_B , using resources $t', q + 1, (q + 1)l$, such that

$$\mathbf{Adv}_F^{\text{kr}}(B) \leq \mathbf{Adv}_F^{\text{prf}}(A_B) + \frac{1}{2^L}. \quad (3.3)$$

If this is true then we can establish Equation (3.3) as follows:

$$\begin{aligned} \mathbf{Adv}_F^{\text{kr}}(t, q, \mu) &= \max_B \{ \mathbf{Adv}_F^{\text{kr}}(B) \} \\ &\leq \max_B \{ \mathbf{Adv}_F^{\text{prf}}(A_B) + 2^{-L} \} \\ &\leq \max_A \{ \mathbf{Adv}_F^{\text{prf}}(A) + 2^{-L} \} \\ &= \mathbf{Adv}_F^{\text{prf}}(t, q + 1, (q + 1)l) + 2^{-L}. \end{aligned}$$

The maximum, in the case of B , is taken over all adversaries whose resources are t, q, ql . In the second line, we apply Equation (3.3). In the third line, we maximize over all A whose resources are $t, q + 1, (q + 1)l$. The inequality on the third line is true because this set includes all adversaries of the form A_B . The last line is simply by definition. So it remains to show how to design A_B so that Equation (3.3) holds. (This is the core of the argument, namely what is called the “reduction.”)

As per Definition 3.4, adversary A_B will be provided an oracle for a function $g: \{0, 1\}^l \rightarrow \{0, 1\}^L$, and will try to determine in which World it is. To do so, it will run adversary B as a subroutine. We provide the description followed by an explanation and analysis.

Adversary A_B^g

$i \leftarrow 0$

Run adversary B , replying to its oracle queries as follows

When B makes an oracle query x do

$i \leftarrow i + 1; x_i \leftarrow x$

$y_i \leftarrow g(x_i)$

Return y_i to B as the answer

Until B stops and outputs a key K'

Let x be an l bit string not in the set $\{x_1, \dots, x_q\}$

$y \leftarrow g(x)$

If $F(K', x) = y$ then return 1 else return 0

As indicated in the discussion preceding the proof, A_B is running B and itself providing answers to B 's oracle queries via the oracle g . When B has run to completion it returns some k -bit string K' , which A_B tests by checking whether $F(K'x)$ agrees with $g(x)$. Here x is a value different from any that B queried, and it is to ensure that such a value can be found that we require $q < 2^l$ in the statement of the

Proposition. Now we claim that

$$\begin{aligned}\Pr \left[\mathbf{Exmt}_F^{\text{prf-1}}(A_B) = 1 \right] &\geq \mathbf{Adv}_F^{\text{kr}}(B) \\ \Pr \left[\mathbf{Exmt}_F^{\text{prf-0}}(A_B) = 1 \right] &= 2^{-L}.\end{aligned}$$

We will justify these claims shortly, but first let us use them to conclude. Subtracting, as per Definition 3.4, we get

$$\begin{aligned}\mathbf{Adv}_F^{\text{prf}}(A_B) &= \Pr \left[\mathbf{Exmt}_F^{\text{prf-1}}(A_B) = 1 \right] - \Pr \left[\mathbf{Exmt}_F^{\text{prf-0}}(A_B) = 1 \right] \\ &\geq \mathbf{Adv}_F^{\text{kr}}(B) - 2^{-L}.\end{aligned}$$

Re-arranging terms gives us Equation (3.3). It remains to justify Equations (3.4) and (3.4).

Equation (3.4) is true because in $\mathbf{Exmt}_F^{\text{prf-1}}(A_B)$ the oracle g is F_K for some K , which is the oracle that B expects, and thus B functions as it does in $\mathbf{Adv}_F^{\text{kr}}(B)$. If B is successful, meaning the key K' it outputs equals K , then certainly A_B returns 1. (It is possible that A_B might return 1 even though B was not successful. This would happen if $K' \neq K$ but $F(K', x) = F(K, x)$. It is for this reason that $\Pr \left[\mathbf{Exmt}_F^{\text{prf-1}}(A_B) = 1 \right]$ is greater than or equal to $\mathbf{Adv}_F^{\text{kr}}(B)$ rather than merely equal to it.) Equation (3.4) is true because in $\mathbf{Exmt}_F^{\text{prf-0}}(A_B)$ the function g is random, and since x was never queried by B , the value $g(x)$ is unpredictable to B . Imagine that $g(x)$ is chosen only when x is queried to g . At that point, K' , and thus $F(K', x)$, is already defined. So $g(x)$ has a 2^{-L} chance of hitting this fixed point. Note this is true regardless of how hard B tries to make $F(K', x)$ be the same as $g(x)$.

For the proof of Equation (3.2) we seek a reduction $B \mapsto A_B$ with the property that

$$\mathbf{Adv}_F^{\text{kr}}(B) \leq \mathbf{Adv}_F^{\text{prp-cpa}}(A_B) + \frac{1}{2^L - q}. \quad (3.4)$$

The reduction is identical to the one given above, meaning the adversary A_B is the same. For the analysis we see that

$$\begin{aligned}\Pr \left[\mathbf{Exmt}_F^{\text{prp-cpa-1}}(A_B) = 1 \right] &= \mathbf{Adv}_F^{\text{kr}}(B) \\ \Pr \left[\mathbf{Exmt}_F^{\text{prp-cpa-0}}(A_B) = 1 \right] &\leq \frac{1}{2^L - q}.\end{aligned}$$

Subtracting yields

$$\begin{aligned}\mathbf{Adv}_F^{\text{prp-cpa}}(A_B) &= \Pr \left[\mathbf{Exmt}_F^{\text{prp-cpa-1}}(A_B) = 1 \right] - \Pr \left[\mathbf{Exmt}_F^{\text{prp-cpa-0}}(A_B) = 1 \right] \\ &\geq \mathbf{Adv}_F^{\text{kr}}(B) - \frac{1}{2^L - q}\end{aligned}$$

and re-arranging terms gives us Equation (3.4). The first equation above is true for the same reason as before. The second equation is true because in World 0 the map g is now a random permutation of l -bits to l -bits. So $g(x)$ assumes any random value except the values y_1, \dots, y_q , meaning there are $2^L - q$ things it could be. (Remember $L = l$ in this case.) ■

The following example illustrates that the converse of the above claim is far from true. The kr-advantage of a family can be significantly smaller than its prf or prp-cpa advantage, meaning that a family might be very secure against key recovery yet very insecure as a prf or prp, and thus not useful for protocol design.

Example 3.13 Define the block cipher $E: \{0, 1\}^k \times \{0, 1\}^l \rightarrow \{0, 1\}^l$ by $E_K(x) = x$ for all k -bit keys K and all l -bit inputs x . We claim that it is very secure against key-recovery but very insecure as a PRP under CPA. More precisely, we claim that for all values of t, q , however high,

$$\mathbf{Adv}_E^{\text{kr}}(t, q, ql) = 2^{-k},$$

and on the other hand

$$\mathbf{Adv}_E^{\text{prp-cpa}}(t, 1, l) \geq 1 - 2^{-l}$$

for $t = O(l)$. In other words, given an oracle for E_K , you may make as many queries as you want, and spend as much time as you like, before outputting your guess as to the value of K , yet your chance of getting it right is only 2^{-k} . On the other hand, using only a single query to a given oracle $g: \{0, 1\}^l \rightarrow \{0, 1\}^l$, and very little time, you can tell almost with certainty whether g is an instance of E or is a random function of l bits to l bits. Why are these claims true? Since E_K does not depend on K , an adversary with oracle E_K gets no information about K by querying it, and hence its guess as to the value of K can be correct only with probability 2^{-k} . On the other hand, an adversary can test whether $g(0^l) = 0^l$, and by returning 1 if and only if this is true, attain a prp-advantage of $1 - 2^{-l}$. ■

3.8 The birthday attack

Suppose $E: \{0, 1\}^k \times \{0, 1\}^l \rightarrow \{0, 1\}^l$ is a family of permutations, meaning a block cipher. If we are given an oracle $g: \{0, 1\}^l \rightarrow \{0, 1\}^l$ which is either an instance of E or a random function, there is a simple test to determine which of these it is. Query the oracle at distinct points x_1, x_2, \dots, x_q , and get back values y_1, y_2, \dots, y_q . You know that if g were a permutation, the values y_1, y_2, \dots, y_q must be distinct. If g was a random function, they may or may not be distinct. So, if they are distinct, bet on a permutation.

Surprisingly, this is pretty good distinguisher, as we will argue below. Roughly, it takes $q = \sqrt{2^l}$ queries to get an advantage that is quite close to 1. The reason is the birthday paradox. If you are not familiar with this, you may want to look at Appendix A, and then come back to the following.

This tells us that an instance of a block cipher can be distinguished from a random function based on seeing a number of input-output examples which is approximately $2^{l/2}$. This has important consequences for the security of block cipher based protocols.

Proposition 3.14 Let $E: \{0, 1\}^k \times \{0, 1\}^l \rightarrow \{0, 1\}^l$ be a family of permutations. Suppose q satisfies $2 \leq q \leq 2^{(l+1)/2}$. Then

$$\mathbf{Adv}_E^{\text{prf}}(t, q, ql) \geq 0.3 \cdot \frac{q(q-1)}{2^l},$$

where t is the time for q computations of E , plus $O(ql)$. ■

Proof of Proposition 3.14: The birthday attack is implemented by an adversary D who, given an oracle $g: \{0, 1\}^l \rightarrow \{0, 1\}^l$, works like this:

```

Adversary  $D^g$ 
For  $i = 1, \dots, q$  do
  Let  $x_i$  be the  $i$ -th  $l$ -bit string in lexicographic order
   $y_i \leftarrow g(x_i)$ 
End For
If  $y_1, \dots, y_q$  are all distinct then return 1, else return 0

```

We claim that

$$\mathbf{Adv}_E^{\text{prf}}(D) \geq 0.3 \cdot \frac{q(q-1)}{2^l},$$

from which the Proposition follows. Let us now justify this lower bound. Letting $N = 2^l$, we claim that

$$\Pr \left[\mathbf{Exmt}_E^{\text{prf-1}}(D) = 1 \right] = 1 \tag{3.5}$$

$$\Pr \left[\mathbf{Exmt}_E^{\text{prf-0}}(D) = 1 \right] = 1 - C(N, q). \tag{3.6}$$

Here $C(N, q)$, as defined in Appendix A, is the probability that some bin gets two or more balls in the experiment of randomly throwing q balls into N bins. We will justify these claims shortly, but first let us use them to conclude. Subtracting, we get

$$\begin{aligned} \mathbf{Adv}_E^{\text{prf}}(D) &= \Pr \left[\mathbf{Exmt}_E^{\text{prf-1}}(D) = 1 \right] - \Pr \left[\mathbf{Exmt}_E^{\text{prf-0}}(D) = 1 \right] \\ &= 1 - [1 - C(N, q)] \\ &= C(N, q) \\ &\geq 0.3 \cdot \frac{q(q-1)}{2^l}. \end{aligned}$$

The last line is by Proposition A.1. It remains to justify Equations (3.5) and (3.6). Equation (3.5) is clear because in World 1, $g = E_K$, and since E is a family of permutations, g is a permutation, and thus y_1, \dots, y_q are all distinct. Now, suppose D is in World 0, so that g is a random function of l bits to l bits. What is the probability that y_1, \dots, y_q are all distinct? Since g is a random function and x_1, \dots, x_q are distinct, y_1, \dots, y_q are random, independently distributed values in $\{0, 1\}^l$. Thus we are looking at the birthday problem. We are throwing q balls into $N = 2^l$ bins and asking what is the probability of there being no collisions, meaning no bin contains two or more balls. This is $1 - C(N, q)$, justifying Equation (3.6). ■

3.9 PRFs versus PRPs

When we come to analyses of block cipher based constructions, we will find a curious dichotomy. Analyses are considerably simpler and more natural assuming the block cipher is a PRF. Yet, PRPs are what most naturally model block ciphers. To bridge the gap, we relate the prf and prp-cpa advantage functions of a given block cipher. The following says, roughly, that the birthday attack is the best possible one. A particular family of permutations E may have prf-advantage that is greater than its prp-advantage, but only by an amount of $q(q-1)/2^{l+1}$, the collision probability term in the birthday attack.

Proposition 3.15 Suppose $E: \{0, 1\}^k \times \{0, 1\}^l \rightarrow \{0, 1\}^l$ is a family of permutations. Then

$$\mathbf{Adv}_E^{\text{prf}}(t, q, ql) \leq \frac{q(q-1)}{2^{l+1}} + \mathbf{Adv}_E^{\text{prp-cpa}}(t, q, ql)$$

for any t, q . ■

The proof is again by reduction, but a very simple one. A given prf-adversary A is mapped to prp-adversary A , meaning the adversary is unchanged. Accordingly, the following does not explicitly talk of reductions.

Proof: Let A be an adversary that takes an oracle for a function $g: \{0, 1\}^l \rightarrow \{0, 1\}^l$. Then we claim that

$$\mathbf{Adv}_E^{\text{prf}}(A) \leq \mathbf{Adv}_E^{\text{prp-cpa}}(A) + \frac{q(q-1)}{2^{l+1}}, \quad (3.7)$$

where q is the number of oracle queries made by A . The Proposition follows by taking maximums, so it remains to prove Equation (3.7).

Let B denote the adversary that first runs A to obtain an output bit b and then returns \bar{b} , the complement of b . Then

$$\mathbf{Adv}_E^{\text{prf}}(A) = \Pr[\mathbf{Exmt}_E^{\text{prf-1}}(A) = 1] - \Pr[\mathbf{Exmt}_E^{\text{prf-0}}(A) = 1]$$

$$\begin{aligned}
&= \left(1 - \Pr \left[\mathbf{Exmt}_E^{\text{prf-1}}(B) = 1 \right]\right) - \left(1 - \Pr \left[\mathbf{Exmt}_E^{\text{prf-0}}(B) = 1 \right]\right) \\
&= \Pr \left[\mathbf{Exmt}_E^{\text{prf-0}}(B) = 1 \right] - \Pr \left[\mathbf{Exmt}_E^{\text{prf-1}}(B) = 1 \right] \\
&= \Pr \left[\mathbf{Exmt}_E^{\text{prf-0}}(B) = 1 \right] - \Pr \left[\mathbf{Exmt}_E^{\text{prp-cpa-1}}(B) = 1 \right] \\
&= \Pr \left[\mathbf{Exmt}_E^{\text{prf-0}}(B) = 1 \right] - \Pr \left[\mathbf{Exmt}_E^{\text{prp-cpa-0}}(B) = 1 \right] \\
&\quad + \Pr \left[\mathbf{Exmt}_E^{\text{prp-cpa-0}}(B) = 1 \right] - \Pr \left[\mathbf{Exmt}_E^{\text{prp-cpa-1}}(B) = 1 \right] \\
&= \Pr \left[\mathbf{Exmt}_E^{\text{prf-0}}(B) = 1 \right] - \Pr \left[\mathbf{Exmt}_E^{\text{prp-cpa-0}}(B) = 1 \right] + \mathbf{Adv}_E^{\text{prp-cpa}}(A).
\end{aligned}$$

So it suffices to show that

$$\Pr \left[\mathbf{Exmt}_E^{\text{prf-0}}(B) = 1 \right] - \Pr \left[\mathbf{Exmt}_E^{\text{prp-cpa-0}}(B) = 1 \right] \leq \frac{q(q-1)}{2^{l+1}}. \quad (3.8)$$

Let $\Pr[\cdot]$ denote the probability in Experiment $\mathbf{Exmt}_E^{\text{prf-0}}(B)$, and let g denote the oracle in that experiment. Assume without loss of generality that all oracle queries of A —they are the same as those of B —are distinct. Let D denote the event that all the answers are distinct, and let \overline{D} denote the complement of event D . Then

$$\begin{aligned}
\Pr \left[\mathbf{Exmt}_E^{\text{prf-0}}(B) = 1 \right] &= \Pr [B^g = 1] \\
&= \Pr [B^g = 1 \mid D] \cdot \Pr [D] + \Pr [B^g = 1 \mid \overline{D}] \cdot \Pr [\overline{D}] \\
&\leq \Pr [B^g = 1 \mid D] + \Pr [\overline{D}] \\
&= \Pr \left[\mathbf{Exmt}_E^{\text{prp-cpa-0}}(B) = 1 \right] + \Pr [\overline{D}] \\
&\leq \Pr \left[\mathbf{Exmt}_E^{\text{prp-cpa-0}}(B) = 1 \right] + \frac{q(q-1)}{2^{l+1}}.
\end{aligned}$$

In the last step we used Proposition A.1. Re-arranging terms gives us Equation (3.8) and concludes the proof. ■

3.10 One-way functions

The framework for the Unix password-hashing scheme is this. We fix some function $h: \{0, 1\}^k \rightarrow \{0, 1\}^L$, which we call the *password hashing function*. A user U chooses a k -bit password K , and the system stores in the password file the value $y = h(K)$ together with the user's name U . When the user logs in he or she is prompted for a user name U and a password K . The system uses the user U to retrieve y , and then the system computes $h(K)$ and declares the user to be authentic if and only if this value equals y . The idea of this system—instead of storing (U, K) itself—is that a party who obtains (U, y) still can not gain trivial entry into the system: they must still find a K such that $h(K) = y$.

Assume the attacker gets access to the password file and hence to y . The attacker's task is thus to find K given y . (The attacker knows the function h , since this is public code. However we assume the attacker does not have any further powers, such as the use of trojan horses.) Security in this model would require that it be computationally infeasible to recover K from y . Thus h must be chosen to make this true.

A simple example choice of h is $h(K) = \text{DES}_K(0^{64})$. (The actual choice made by Unix is somewhat more complex, involving something called a "salt," which customizes the function h to each user U . It also involves iterating the block cipher a number of times. However this does not change the heart of the analysis, so let us stick with the fiction we have described.) In this example, $k = 56$ and $L = 64$.

We ask ourselves how secure is this scheme. The question boils down to asking how hard it would be to recover K given $y = \text{DES}_K(0^{64})$.

Obviously, the security of this scheme depends on the security of DES. If we want to prove anything meaningful about the security of the simplified password scheme, we must make some assumption about DES. We have suggested above that the appropriate assumption to make about a block cipher like DES is that it is a secure PRP. So we make this assumption and now ask what we can prove about the security of the simplified password scheme.

We know what we want to assume about DES, but we don't yet know exactly what security property we would like to target the password scheme as meeting. We need some model and definition for this. We target the requirement that the password-hashing function be *one-way*, meaning it is computationally infeasible to recover the pre-image of a range point. The formalization is more specific. Function $h: \{0, 1\}^k \rightarrow \{0, 1\}^L$ is one-way if it is hard, given y , to compute a point x' such that $h(x') = y$, when y was chosen by drawing x at random from $\{0, 1\}^k$ and setting $y = h(x)$. A definition to capture this notion of one-wayness appears below.

Definition 3.16 Let $h: \{0, 1\}^k \rightarrow \{0, 1\}^L$ be a function, and let I be an algorithm that on input an L -bit string returns a k -bit string. We consider the experiment:

Experiment $\mathbf{Exmt}_h^{\text{owf}}$
 $K \xleftarrow{R} \{0, 1\}^k ; y \leftarrow h(K)$
 $x \leftarrow I(y)$
 If $h(x) = y$ then return 1 else return 0

The *owf-advantage* of I is defined as

$$\mathbf{Adv}_h^{\text{owf}}(I) = \Pr \left[\mathbf{Exmt}_h^{\text{owf}} = 1 \right].$$

For any t the *owf-advantage* of I is defined via

$$\mathbf{Adv}_h^{\text{owf}}(t) = \max_I \{ \mathbf{Adv}_h^{\text{owf}}(I) \}$$

where the maximum is over all I having time-complexity t . ■

As usual, a one-way function is understood to be one for which $\mathbf{Adv}_h^{\text{owf}}(t)$ is “small” for practical values of t . We want to show that if h is defined via $h(K) = F_K(0^l)$ for a secure PRF $F: \{0, 1\}^k \times \{0, 1\}^l \rightarrow \{0, 1\}^L$ then h is one-way.

We remark that one must look carefully at the models to know how to interpret the impact of such a result on the actual password scheme. Showing that h is a one-way function amounts to saying that the password scheme is secure if passwords are randomly chosen k -bit keys where k is the block length of the block cipher. In real life, passwords are often not random, and in that case this result does not apply. However, our intent here is to illustrate an application of PRFs, not to explain the true security of the Unix password scheme.

Theorem 3.17 Let $F: \{0, 1\}^k \times \{0, 1\}^l \rightarrow \{0, 1\}^L$ be a family of functions, and define $h: \{0, 1\}^k \rightarrow \{0, 1\}^L$ via $h(K) = F(K, 0^l)$ for all $K \in \{0, 1\}^k$. Then we have

$$\mathbf{Adv}_h^{\text{owf}}(t) \leq \frac{1}{1 - 2^{k-L}} \cdot \mathbf{Adv}_F^{\text{prf}}(t, 1, l), \quad (3.9)$$

under the assumption that $k \leq L - 1$. ■

As per the theorem, $\mathbf{Adv}_h^{\text{owf}}(t)$ can only be marginally more than $\mathbf{Adv}_F^{\text{prf}}(t, 1, l)$. Specifically, $\mathbf{Adv}_h^{\text{owf}}(t)$ can be at most twice $\mathbf{Adv}_F^{\text{prf}}(t, 1, l)$, because $k \leq L - 1$ implies $1 - 2^{k-L} \leq 2$. So if F is secure, meaning $\mathbf{Adv}_F^{\text{prf}}(t, 1, l)$ is low, $\mathbf{Adv}_h^{\text{owf}}(t)$ is also low, and hence h is secure. It is thus a proof of security, showing that h is one-way if F is a secure PRF.

It is an open question what happens when $k \geq L$. We do not know whether, in this case, h is still one-way, and, if it is, whether this can be proved based solely on the assumption that F is a secure PRF. For DES we do have $k \leq L - 1$, but for AES we do not, so it is a relevant question. Answering these questions is a research problem and shows how quickly one reaches the research boundaries in this area.

Proof of Theorem 3.17: We associate to any adversary I attempting to invert h an adversary D_I attacking F such that

$$\mathbf{Adv}_F^{\text{prf}}(D_I) \leq \frac{1}{1 - 2^{k-L}} \cdot \mathbf{Adv}_h^{\text{owf}}(I). \quad (3.10)$$

Furthermore, D_I makes only one oracle query, this of length l bits, and has time-complexity t where t is the time-complexity of I . Taking maximums in the usual way yields Equation (3.9), so it remains to provide D_I such that Equation (3.10) is true. This adversary takes an oracle for a function $g: \{0, 1\}^l \rightarrow \{0, 1\}^L$ and works as follows:

Adversary D_I^g
 $y \leftarrow g(0^l)$
 $x \leftarrow I(y)$
 If $F(x, 0^l) = y$ then return 1 else return 0

The adversary queries its oracle g at 0^l to get back a value it calls y , and then applies the inverting algorithm I to y to get back a value x . If I successfully inverted h at y our adversary bets that g is an instance of F , and otherwise it bets that g is an instance of $\text{Rand}(l, L)$. To compute the advantage of this adversary it is convenient to set

$$\epsilon = \mathbf{Adv}_F^{\text{owf}}(I) .$$

Now we claim that

$$\Pr \left[\mathbf{Exmt}_F^{\text{prf-1}}(D_I) = 1 \right] = \epsilon \quad (3.11)$$

$$\Pr \left[\mathbf{Exmt}_F^{\text{prf-0}}(D_I) = 1 \right] \leq \frac{2^k}{2^L} \cdot \epsilon . \quad (3.12)$$

We will justify these claims shortly, but first let us use them to conclude. Subtracting, we have

$$\begin{aligned} \mathbf{Adv}_F^{\text{prf}}(D_I) &= \Pr \left[\mathbf{Exmt}_F^{\text{prf-1}}(D_I) = 1 \right] - \Pr \left[\mathbf{Exmt}_F^{\text{prf-0}}(D_I) = 1 \right] \\ &\geq \epsilon - \frac{2^k}{2^L} \cdot \epsilon \\ &= \left(1 - 2^{k-L} \right) \cdot \epsilon . \end{aligned}$$

Now, we divide both sides by $1 - 2^{k-L}$ to get

$$\epsilon \leq \frac{1}{1 - 2^{k-L}} \cdot \mathbf{Adv}_F^{\text{prf}}(D_I) ,$$

which is exactly Equation (3.10). However, there is a subtle point here that should be noted. This step is only correct if the quantity $1 - 2^{k-L}$ by which we are dividing is non-zero (otherwise we can't divide by it) and in fact positive (if it was negative, we would have to reverse the inequality). The fact that $1 - 2^{k-L}$ is positive is true by our assumption that $k \leq L - 1$. This is the *only* place we make use of this assumption, but it is crucial. It remains to justify Equations (3.11) and (3.12).

We claim that Experiment $\mathbf{Exmt}_F^{\text{prf-1}}(D_I)$ ends up faithfully mimicking Experiment $\mathbf{Exmt}_{h,I}^{\text{owf}}$. Indeed, Experiment $\mathbf{Exmt}_F^{\text{prf-1}}(D_I)$ begins by selecting a random k -bit key K , so that $y = F(K, 0^l)$. By definition of h this means that $y = h(K)$, so y is distributed the same way in the two experiments. Then, both experiments run I and return 1 if and only if I is successful, so the probability that they return 1 is the same. This justifies Equation (3.10).

Now suppose D_I is in World 0, meaning $g: \{0, 1\}^l \rightarrow \{0, 1\}^L$ is a random function. We want to upper bound the probability that $\mathbf{Exmt}_F^{\text{prf-0}}(D_I)$ returns 1. Since g is random, y will be uniformly distributed over $\{0, 1\}^L$. Thus we want to upper bound

$$\delta \stackrel{\text{def}}{=} \Pr \left[y \stackrel{R}{\leftarrow} \{0, 1\}^L ; x \leftarrow I(y) : F(x, 0^l) = y \right] . \quad (3.13)$$

The notation here means that we first pick y at random from $\{0, 1\}^L$, then set x to $I(y)$, and then ask what is the probability that $F(x, 0^l)$ equals y . Since the algorithm I might be randomized, the probability is not only over the choice of y , but also over the random coins tossed by I itself.

For simplicity we first prove Equation (3.12) in the case where I is deterministic, so that the probability in the computation of δ is only over the choice of y . In this case it is convenient to define the sets

$$\begin{aligned} X &= \{x \in \{0, 1\}^k : h(I(h(x))) = h(x)\} \\ Y &= \{y \in \{0, 1\}^L : h(I(y)) = y\}. \end{aligned}$$

We show the sequence of steps via which Equation (3.12) can be obtained, and then justify them:

$$\delta = \frac{|Y|}{2^L} \leq \frac{|X|}{2^L} = \frac{2^k \cdot \epsilon}{2^L}.$$

The fact that $\delta = |Y|/2^L$ follows from Equation (3.13) and the definition of Y . The last equality uses the analogous fact that $\epsilon = |X|/2^k$, and this can be justified by looking at Experiment $\mathbf{Exmt}_{h,I}^{\text{owf}}$ and the definition of set X above. The main claim used above is that $|Y| \leq |X|$. To see why this is true, let

$$h(X) = \{h(x) : x \in \{0, 1\}^k\} = \{y \in \{0, 1\}^L : \exists x \in X \text{ such that } h(x) = y\}.$$

This is called the *image* of X under h . Then observe two things, from which $|Y| \leq |X|$ follows:

$$|h(X)| \leq |X| \quad \text{and} \quad h(X) = Y.$$

The first of these is true simply because h is a function. (One x value yields exactly one y value under h . Some of these y values might be the same as x ranges over X , but certainly you can't get more y values than you have x values.) The second, that $h(X) = Y$, can be justified by looking at the definitions of the sets X and Y and observing two things: If $x \in X$ then $h(x) \in Y$ and if $y \in Y$ then there is some $x \in X$ such that $h(x) = y$.

That completes the proof for the case where I is deterministic. Let us now briefly indicate why Equation (3.12) remains true when I is a randomized algorithm.

In this case, when I is run on input y , it tosses coins to get a random string R , and bases its computation on both y and R , returning a value x that is a function of both of y and R . Thus, there are many different possible x values that it might return on input y . We have no idea exactly how I uses R or how it performs its computation, but we can still assess the probabilities we need to assess. For any $y \in \{0, 1\}^L$ and any $x \in \{0, 1\}^k$ we let

$$P_y(x) = \Pr[R \leftarrow \{0, 1\}^r : I(y; R) = x].$$

In other words, having fixed x, y , we ask what is the probability that I , on input y , would output x . The probability is over the coin toss sequence R of I , and this has been made explicit. We are letting r be the number of coins that I tosses and

letting $I(y; R)$ denote the output of I on input y and coins R . Note that this output is a single x value. (Towards understanding this it may be helpful to note that the case of I being deterministic corresponds to the following: for every y there is a unique x such that $P_y(x) = 1$, and for all other values of x we have $P_y(x) = 0$.)

Now for any $y \in \{0, 1\}^L$ we let

$$\begin{aligned} h^{-1}(y) &= \{x \in \{0, 1\}^k : h(x) = y\} \\ Y^* &= \{y \in \{0, 1\}^L : h^{-1}(y) \neq \emptyset\}. \end{aligned}$$

Thus $h^{-1}(y)$ is the set of all pre-images of y under h , while Y^* is the image of $\{0, 1\}^k$ under h , meaning the set of all range points that possess some pre-image under h . Notice that for any $y \in Y^*$ we have $|h^{-1}(y)| \geq 1$. Thus for any $y \in Y^*$ we have

$$\frac{1}{2^L} \leq \frac{|h^{-1}(y)|}{2^L} = \frac{2^k}{2^L} \cdot \frac{|h^{-1}(y)|}{2^k}. \quad (3.14)$$

We show the sequence of steps via which Equation (3.12) can be obtained, and then justify them:

$$\begin{aligned} \delta &= \sum_{y \in \{0, 1\}^L} \left(\sum_{x \in h^{-1}(y)} P_y(x) \right) \cdot \frac{1}{2^L} \\ &= \sum_{y \in Y^*} \left(\sum_{x \in h^{-1}(y)} P_y(x) \right) \cdot \frac{1}{2^L} \\ &\leq \sum_{y \in Y^*} \left(\sum_{x \in h^{-1}(y)} P_y(x) \right) \cdot \frac{2^k}{2^L} \cdot \frac{|h^{-1}(y)|}{2^k} \\ &= \frac{2^k}{2^L} \cdot \sum_{y \in Y^*} \left(\sum_{x \in h^{-1}(y)} P_y(x) \right) \cdot \frac{|h^{-1}(y)|}{2^k} \\ &= \frac{2^k}{2^L} \cdot \sum_{y \in \{0, 1\}^L} \left(\sum_{x \in h^{-1}(y)} P_y(x) \right) \cdot \frac{|h^{-1}(y)|}{2^k} \\ &= \frac{2^k}{2^L} \cdot \epsilon. \end{aligned}$$

The equation for δ used in the first line comes about by looking at the the probability that I succeeds for a given value of y , and then summing this over all y -values, weighted by the probability 2^{-L} of that y value being chosen. We then restrict the sum to values $y \in Y^*$ based on the fact that the terms corresponding to values $y \notin Y^*$ in the previous sum are just zero. Once this is done we can apply Equation (3.14) to obtain the inequality. We then factor $2^k/2^L$ out of the sum. We extend the sum to cover values $y \notin Y^*$ based again on the fact that the corresponding new terms are simply zero. In the last sum, we are summing the probability that I succeeds for a

given value of y , weighted by the probability that y would be produced under the experiment of choosing x at random and setting $y = h(x)$, namely as in Experiment $\mathbf{Exmt}_{h,I}^{\text{owf}}$, and thus recover ϵ . ■

3.11 Pseudorandom generators

3.12 Historical notes

The basic notion of pseudorandom functions is due to Goldreich, Goldwasser and Micali [16]. In particular these authors introduced the important notion of distinguishers. The notion of a pseudorandom permutation is due to Luby and Rackoff [22]. These works are in the complexity-theoretic or “asymptotic” setting, where one considers an infinite sequence of families rather than just one family, and defines security by saying that polynomial-time adversaries have “negligible” advantage. The approach used here, motivated by the desire to model block ciphers, is called “concrete security,” and originates with [2]. Definitions 3.4 and 3.5 are from [2], as are Propositions 3.14 and 3.15. The materiel of Section 3.10 is a concrete security adaptation of results from [23].

3.13 Exercises and problems

Exercise 3.1 Let $E: \{0,1\}^k \times \{0,1\}^n \rightarrow \{0,1\}^n$ be a secure PRP. Consider the PRP $E': \{0,1\}^k \times \{0,1\}^{2n} \rightarrow \{0,1\}^{2n}$ defined by

$$E'_K(xx') = E_K(x) E_K(x \oplus x')$$

where $|x| = |x'| = n$. Show that E' is not a secure PRP

Exercise 3.2 Consider the following block cipher $E: \{0,1\}^3 \times \{0,1\}^2 \rightarrow \{0,1\}^2$:

key	0	1	2	3
0	0	1	2	3
1	3	0	1	2
2	2	3	0	1
3	1	2	3	0
4	0	3	2	1
5	1	0	3	2
6	2	1	0	3
7	3	2	1	0

(The eight possible keys are the eight rows, and each row shows where the points to which 0, 1, 2, and 3 map.) Compute the maximal advantage an adversary can get (a) with one query, (b) with four queries, and (c) with two queries.

Exercise 3.3 Let $D, R \subseteq \{0, 1\}^*$ with D finite. Let $f : D \rightarrow R$. Consider the following definition for the success of an adversary I in breaking f as a one-way function:

$$\mathbf{Adv}_f^{\text{owf}}(I) = \Pr[X \stackrel{R}{\leftarrow} D : I(f(X)) = X]$$

Is this a good definition for the security of a one-way function? Why or why not.

Problem 3.1 Suppose you are given a PRF $F: \{0, 1\}^k \times \{0, 1\}^n \rightarrow \{0, 1\}^n$. Design a PRF $G: \{0, 1\}^{2k} \times \{0, 1\}^n \rightarrow \{0, 1\}^{2n}$ which is secure as long as F is secure. Analyze the security of G in terms of the security of F .

Problem 3.2 Present a secure construction for the problem of Example 3.9. That is, given a PRF $F: \{0, 1\}^k \times \{0, 1\}^n \rightarrow \{0, 1\}^n$, construct a PRF $G: \{0, 1\}^k \times \{0, 1\}^n \rightarrow \{0, 1\}^{2n}$ which is a secure PRF as long as F is secure.

Problem 3.3 Design a block cipher $E : \mathcal{K} \times \{0, 1\}^{128} \rightarrow \{0, 1\}^{128}$ that is secure (up to a large number of queries) against non-adaptive adversaries, but is completely insecure (even for two queries) against an adaptive adversary. (A non-adaptive adversary reads all her questions M_1, \dots, M_q , in advance, getting back $E_K(M_1), \dots, E_K(M_q)$. An adaptive adversary is the sort we have dealt with throughout: each query may depend on prior answers.)

Problem 3.4 Let $a[i]$ denote the i -th bit of a binary string a , where $1 \leq i \leq |a|$. The *inner product* of n -bit binary strings a, b is

$$\langle a, b \rangle = a[1]b[1] \oplus a[2]b[2] \oplus \dots \oplus a[n]b[n].$$

A family of functions $F: \{0, 1\}^k \times \{0, 1\}^l \rightarrow \{0, 1\}^L$ is said to be *inner-product preserving* if for every $K \in \{0, 1\}^k$ and every distinct $x_1, x_2 \in \{0, 1\}^l - \{0^l\}$ we have

$$\langle F(K, x_1), F(K, x_2) \rangle = \langle x_1, x_2 \rangle.$$

Prove that if F is inner-product preserving then

$$\mathbf{Adv}_F^{\text{prf}}(t, 2, 2l) \geq \frac{1}{2} \cdot \left(1 + \frac{1}{2^L}\right)$$

for $t = q \cdot T_F + O(\mu)$, where T_F denotes the time to perform one computation of F . Explain in a sentence why this shows that if F is inner-product preserving then F is not a secure PRF.

Problem 3.5 Let $E: \{0, 1\}^k \times \{0, 1\}^l \rightarrow \{0, 1\}^l$ be a block cipher. The *two-fold cascade* of E is the block cipher $E^{(2)}: \{0, 1\}^{2k} \times \{0, 1\}^l \rightarrow \{0, 1\}^l$ defined by

$$E^{(2)}(K_1 \| K_2, x) = E(K_1, E(K_2, x))$$

for all $K_1, K_2 \in \{0, 1\}^k$ and all $x \in \{0, 1\}^l$. (Here “||” stands for concatenation of strings.) Prove that

$$\mathbf{Adv}_{E^{(2)}}^{\text{prp-cpa}}(t, q, lq) \leq \mathbf{Adv}_E^{\text{prp-cpa}}(t, q, lq)$$

for all t, q . Explain in a sentence why this shows that if E is a secure PRP then so is $E^{(2)}$.

Problem 3.6 Give a construction to show that $F : \{0, 1\}^{2n} \times \{0, 1\}^n \rightarrow \{0, 1\}^n$ can be a good PRF (secure in the sense of $\mathbf{Adv}_F^{\text{prf}}$) and yet the function $f(X) = F_X(0)$ is not a secure one-way function.

Problem 3.7 Let $D, R \subseteq \{0, 1\}^*$ with D finite. Let $f : D \rightarrow R$ be a function. Suppose there is a probabilistic adversary I that, in time t , obtains advantage $\epsilon = \mathbf{Adv}_f^{\text{owf}}(I)$. Show that there is a deterministic adversary I' with essentially the same running time as I such that $\epsilon = \mathbf{Adv}_f^{\text{owf}}(I')$.

Problem 3.8 Let A be an adversary that makes at most q total queries to its two oracles, f and g , where $f, g : \{0, 1\}^n \rightarrow \{0, 1\}^n$. Assume that A never asks the same query X to both of its oracles. Define

$$\mathbf{Adv}(A) = \Pr[\pi \leftarrow \text{Perm}(n) : A^{\pi(\cdot), \pi(\cdot)} = 1] - \Pr[\pi, \pi' \leftarrow \text{Perm}(n) : A^{\pi(\cdot), \pi'(\cdot)} = 1].$$

Prove a good upper bound for $\mathbf{Adv}(A)$, say $\mathbf{Adv}(A) \leq q^2/2^n$.

Chapter 4

SYMMETRIC ENCRYPTION

A symmetric encryption scheme (also called a shared-key encryption scheme) enables parties in possession of a shared secret key to achieve the goal of data privacy. This is the canonical goal of cryptography.

4.1 A framework for both encryption and message authentication

The symmetric setting considers two parties who share a key and will use this key to imbue communicated data with various security attributes. The main security goals are privacy and authenticity of the communicated data. Chapter 4 looks at privacy, Chapter 6 looks at authenticity, and Chapter 7 looks at providing both together. Chapters 2 and 5 describe primitives we shall use.

The type of object we will consider we call an *encapsulation scheme*. An encapsulation scheme specifies an encapsulation algorithm, which tells the sender how to process her data as a function of the key to produce the object that is actually transmitted. It also specifies a decapsulation algorithm which tells the receiver how to retrieve the original data from the transmission while possibly also performing some verification. Finally, there is a key generation algorithm, which produces a key that the parties need to share. The formal description follows.

Definition 4.1 A *symmetric encapsulation scheme* $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ consists of three algorithms, as follows:

- The *key generation* algorithm \mathcal{K} is a randomized algorithm that returns a string K . We let $\text{Keys}(\mathcal{SE})$ denote the set of all strings that have non-zero probability of being output by \mathcal{K} . The members of this set are called *keys*. We write $K \stackrel{R}{\leftarrow} \mathcal{K}$ for the operation of executing \mathcal{K} and letting K denote the key returned.

- The *encapsulation* algorithm \mathcal{E} takes a key $K \in \text{Keys}(\mathcal{SE})$ and a *plaintext* $M \in \{0, 1\}^*$ to return a *ciphertext* $C \in \{0, 1\}^* \cup \{\perp\}$. This algorithm might be randomized or stateful. We write $C \stackrel{R}{\leftarrow} \mathcal{E}_K(M)$.
- The deterministic *decapsulation* algorithm \mathcal{D} takes a key $K \in \text{Keys}(\mathcal{SE})$ and a ciphertext $C \in \{0, 1\}^* \cup \{\perp\}$ to return some $M \in \{0, 1\}^* \cup \{\perp\}$. We write $M \leftarrow \mathcal{D}_K(C)$.

We require that for any key $K \in \text{Keys}(\mathcal{SE})$ and any message $M \in \{0, 1\}^*$, if $\mathcal{E}_K(M)$ returns a ciphertext $C \neq \perp$ then $\mathcal{D}_K(C) = M$. ■

When discussing privacy, it is conventional to call an encapsulation scheme an encryption scheme. The encapsulation algorithm is called the encryption algorithm, and the decapsulation algorithm is called the decryption algorithm.

The key generation algorithm, as the definition indicates, is randomized. It takes no inputs. When it is run, it flips coins internally and uses these to select a key K . Typically, the key is just a random string of some length, in which case this length is called the *key length* of the scheme. When two parties want to use the scheme, it is assumed they are in possession of K generated via \mathcal{K} . How they came into joint possession of this key K in such a way that the adversary did not get to know K is not our concern here; it is an assumption we make.

Once in possession of a shared key, the parties can encapsulate data for transmission. To encapsulate plaintext M , the sender (or encapsulator) runs the encapsulation algorithm with key K and input M to get back a string we call the ciphertext.

The encapsulation algorithm may be either randomized or stateful. If randomized, it flips coins and uses those to compute its output on a given input K, M . Each time the algorithm is invoked, it flips coins anew, and in particular invoking it twice on the same inputs may not yield the same response both times. If the encapsulation algorithm is stateful, its operation depends on a global variable such as a counter, which is updated upon each invocation of the encapsulation algorithm. Thus, the encapsulator maintains state that is initialized in some pre-specified way. When the encapsulation algorithm is invoked on inputs K, M , it computes a ciphertext based on K, M and the current state. It then updates the state, and the new state value is stored. (The receiver does not maintain matching state and, in particular, decapsulation does not require access to any global variable or call for any synchronization between parties.)

When there is no such counter or global variable, the scheme is *stateless*. In stateful schemes the encapsulation algorithm typically does not flip coins internally. (It is still OK to call it a randomized algorithm. It just happens to not make use of its source of random bits.) In stateless schemes, randomization is essential to security, as we will see.

Once a ciphertext C is computed, it is transmitted to the receiver. The latter can recover the message by running the decapsulation algorithm with the same key used to create the ciphertext, namely via $M \leftarrow \mathcal{D}_K(C)$. The decapsulation algorithm is

neither randomized nor stateful.

Many encapsulation schemes restrict the set of strings that they are willing to encapsulate. (For example, perhaps the algorithm can only encapsulate plaintexts of length a positive multiple of some block length n , and can only encapsulate plaintexts of length up to so maximum length.) These kinds of restrictions are captured by having the encapsulation algorithm return the special symbol \perp when fed a message not meeting the required restriction. In a stateless scheme, there is typically a set of strings, called the *plaintext space*, such that $\mathcal{E}_K(M) \neq \perp$ for all K and all M in the plaintext space. In a stateful scheme, whether or not $\mathcal{E}_K(M)$ returns \perp depends not only on M but also possibly on the value of the state variable. For example, when a counter is being used, it is typical that there is a limit to the number of encapsulations performed, and when the counter reaches a certain value the encapsulation algorithm returns \perp no matter what message it is fed.

4.2 Some encryption schemes

In the remainder of this chapter, we refer to an encapsulation scheme as an encryption scheme. The encapsulation algorithm is called the encryption algorithm, and the decapsulation algorithm is called the decryption algorithm. Let us begin with a few examples.

Scheme 4.2 [One-time-pad encryption] The one-time-pad encryption scheme $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ is stateful and deterministic. The key generation algorithm simply returns a random k -bit string K , where the key-length k is a parameter of the scheme, so that the key space is $\text{Keys}(\mathcal{SE}) = \mathcal{K}$. The encryptor maintains a counter ctr which is initially zero. The encryption and decryption algorithms operate as follows:

<pre> Algorithm $\mathcal{E}_K(M)$ Let static $ctr \leftarrow 0$ Let $m \leftarrow M$ If $ctr + m > k$ then return \perp $C \leftarrow M \oplus K[ctr .. ctr + m - 1]_1$ $ctr \leftarrow ctr + m$ Return $\langle ctr, C \rangle$ </pre>	<pre> Algorithm $\mathcal{D}_K(\langle ctr, C \rangle)$ Let $m \leftarrow M$ If $ctr + m > k$ then return \perp $M \leftarrow C \oplus K[ctr .. ctr + m - 1]_1$ Return M </pre>
---	--

Here $X[i .. j]_1$ denotes the i -th through j -th bit of the binary string X . By $\langle ctr, C \rangle$ we mean a string that encodes the number ctr and the string C . As the number ctr is in $[0..2^n - 1]$ the most natural encoding is to write ctr using n bits and then prefix this to C . Conventions are established so that every string y is regarded as encoding some ctr, C . The encryption algorithm XORs the message bits with key bits, starting with the key bit indicated by the current counter value. The counter is then incremented by the length of the message. Key bits are not reused, and thus if not enough key bits are available to encrypt a message, the encryption

algorithm returns \perp . Note that the ciphertext returned includes the value of the counter. This is to enable decryption. (Recall that the decryption algorithm, as per Definition 4.1, must be stateless and deterministic, so we do not want it to have to maintain a counter as well.) ■

The following schemes rely either on a family of permutations (ie. a block cipher) or a family of functions. It is convenient if the length of the message to be encrypted is a positive multiple of a block length associated to the family. Accordingly, the encryption algorithm returns \perp if this is not the case. In practice, however, one would first pad the message appropriately so that the padded message always had length a positive multiple of the block length, and apply the encryption algorithm to the padded message. The padding function should be injective and easily invertible.

Scheme 4.3 [ECB mode] Let $E: \mathcal{K} \times \{0, 1\}^n \rightarrow \{0, 1\}^n$ be a block cipher. Operating it in ECB (Electronic Code Book) mode yields a stateless symmetric encryption scheme, $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$. The key generation algorithm simply returns a random key for the block cipher, meaning it picks a random string $K \xleftarrow{R} \mathcal{K}$ and returns it. The encryption and decryption algorithms are as follows:

<p>Algorithm $\mathcal{E}_K(M)$</p> <p>If $M \notin \{n, 2n, 3n, \dots\}$ then return \perp</p> <p>Parse M as n-bit $M[1] \cdots M[m]$</p> <p>For $i \leftarrow 1$ to m do</p> <p style="padding-left: 20px;">$C[i] \leftarrow E_K(M[i])$</p> <p>EndFor</p> <p>$C \leftarrow C[1] \cdots C[m]$</p> <p>Return C</p>	<p>Algorithm $\mathcal{D}_K(C)$</p> <p>If $C \notin \{n, 2n, 3n, \dots\}$ then return \perp</p> <p>Parse C as n-bit $C[1] \cdots C[m]$</p> <p>For $i \leftarrow 1$ to m do</p> <p style="padding-left: 20px;">$M[i] \leftarrow E_K^{-1}(C[i])$</p> <p>EndFor</p> <p>$M \leftarrow M[1] \cdots M[m]$</p> <p>Return M</p>
---	--

Parse M as n -bit $M[1] \cdots M[m]$ means to set $m = |M|/n$ and, $in \in [1..m]$, to set $M[i] = M[i]_n$, where $M[i]_n$ means the i -th n -bit block of M . Similarly for parsing C into $C[1] \cdots C[m]$. Notice that that time the encryption algorithm did not make any random choices. (That does not mean we are not allowed to call it a randomized algorithm; it is simply a randomized algorithm that happened to choose to not make random choices.) ■

The next scheme, cipher-block chaining (CBC), is the most popular mode, used pervasively in practice.

Scheme 4.4 [CBC\$ mode] Let $E: \mathcal{K} \times \{0, 1\}^n \rightarrow \{0, 1\}^n$ be a block cipher. Operating it in CBC mode with random IV yields a stateless symmetric encryption scheme, $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$. The key generation algorithm simply returns a random key for the block cipher, $K \xleftarrow{R} \mathcal{K}$. The encryption and decryption algorithms are as follows:

<p>Algorithm $\mathcal{E}_K(M)$</p> <p>If $M \notin \{n, 2n, 3n, \dots\}$ then return \perp</p> <p>Parse M as n-bit $M[1] \cdots M[m]$</p> <p>$C[0] \leftarrow \text{IV} \stackrel{R}{\leftarrow} \{0, 1\}^n$</p> <p>For $i \leftarrow 1$ to m do</p> <p style="padding-left: 20px;">$C[i] \leftarrow E_K(C[i-1] \oplus M[i])$</p> <p>EndFor</p> <p>$C \leftarrow C[1] \cdots C[m]$</p> <p>Return $\langle \text{IV}, C \rangle$</p>	<p>Algorithm $\mathcal{D}_K(\langle \text{IV}, C \rangle)$</p> <p>If $C \notin \{n, 2n, 3n, \dots\}$ then return \perp</p> <p>Parse C as n-bit $C[1] \cdots C[m]$</p> <p>$C[0] \leftarrow \text{IV}$</p> <p>For $i \leftarrow 1$ to m do</p> <p style="padding-left: 20px;">$M[i] \leftarrow E_K^{-1}(C[i]) \oplus C[i-1]$</p> <p>EndFor</p> <p>$M \leftarrow M[1] \cdots M[m]$</p> <p>Return M</p>
--	---

Parsing C as $C[0] \cdots C[m]$ means that we divide it into n -bit blocks and number them starting at 0. The IV (“initialization vector”) is $C[0]$, which is chosen at random by the encryption algorithm. This choice is made independently each time the algorithm is invoked. ■

For the following schemes it is useful to introduce some notation. If $n \geq 1$ and $i \geq 0$ are integers then we let $[i]_n$ (read “number to an n -bit string”) denote the n -bit string which is the binary representation of integer $i \bmod 2^n$. If we use a number $i \geq 0$ in a context for which a string $I \in \{0, 1\}^n$ is required, it is understood that we mean to replace i by $I = [i]_n$.

The CTR (counter) modes that follow are not much used, to the best of our knowledge, but perhaps wrongly so. We will see later that they have good security properties. In contrast to CBC, the encryption and decryption procedures are parallelizable, which can be exploited to speed up these processes in the presence of hardware support. There are two variants of the mode, one random and the other stateful, and, as we will see later, their security properties are different.

Scheme 4.5 [CTR\$ mode] Let $F: \mathcal{K} \times \{0, 1\}^n \rightarrow \{0, 1\}^\ell$ be a family of functions (not necessarily a family of permutations). Operating it in CTR mode with random starting point is a stateless symmetric encryption scheme, $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$, which we call CTR\$ mode. The key generation algorithm simply returns a random key for F , meaning that the key-generation algorithm for the encryption scheme is just the key generation algorithm for the function family F . The encryption and decryption algorithms are as follows:

<p>Algorithm $\mathcal{E}_K(M)$</p> <p>If $M \notin \{\ell, 2\ell, \dots, \ell 2^n\}$ then return \perp</p> <p>$m \leftarrow M /\ell$</p> <p>$r \stackrel{R}{\leftarrow} [0..2^n - 1]$</p> <p>$P \leftarrow F_K(r+1)F_K(r+2) \cdots F_K(r+m)$</p> <p>$C \leftarrow M \oplus P$</p> <p>Return $\langle r, C \rangle$</p>	<p>Algorithm $\mathcal{D}_K(\langle r, C \rangle)$</p> <p>If $C \notin \{\ell, 2\ell, \dots, \ell 2^n\}$ then return \perp</p> <p>$m \leftarrow C /\ell$</p> <p>$P \leftarrow F_K(r+1)F_K(r+2) \cdots F_K(r+m)$</p> <p>$M \leftarrow C \oplus P$</p> <p>Return M</p>
--	--

In this mode the random value r chosen by the encryption algorithm is an integer in the range $0, \dots, 2^n - 1$. It is used to define a sequence of values on which F_K is applied to produce a “pseudo one-time pad” to which the data is XORed. The random value is included in the ciphertext in order to enable decryption. The natural way to encode r and C is to write the former number as an n -bit string and then prepend this to C . ■

We now give the counter-based version of CTR mode.

Scheme 4.6 [CTRC mode] Let $F: \mathcal{K} \times \{0, 1\}^n \rightarrow \{0, 1\}^\ell$ be a family of functions. Operating it in CTR mode with counter starting point is a stateful symmetric encryption scheme, $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$, which we call CTCRC. The key generation algorithm simply returns a random key for F , meaning that the key generation algorithm for the encryption scheme is also \mathcal{K} . The encryptor maintains a counter ctr which is initially zero. The encryption and decryption algorithms are as follows:

<pre> Algorithm $\mathcal{E}_K(M)$ static $ctr \leftarrow 0$ If $M \notin \{\ell, 2\ell, 3\ell, \dots\}$ then return \perp $m \leftarrow M /\ell$ If $ctr + m > 2^n$ then return \perp $P \leftarrow F_K(ctr)F_K(ctr + 1) \cdots F_K(ctr + m - 1)$ $C \leftarrow M \oplus P$ $ctr \leftarrow ctr + m$ Return $\langle ctr - m, C \rangle$ </pre>	<pre> Algorithm $\mathcal{D}_K(\langle i, C \rangle)$ If $C \notin \{\ell, 2\ell, 3\ell, \dots\}$ then return \perp $m \leftarrow C /\ell$ If $ctr + m > 2^n$ then return \perp $P \leftarrow F_K(ctr)F_K(ctr + 1) \cdots F_K(ctr + m - 1)$ $M \leftarrow P \oplus C$ Return M </pre>
---	--

Position index ctr is not allowed to wrap around: the encryption algorithm returns \perp if this would happen. The position index is included in the ciphertext in order to enable decryption. The encryption algorithm updates the position index upon each invocation, and begins with this updated value the next time it is invoked. ■

We will return to the security of these schemes after we have developed the appropriate notions.

4.3 Issues in security

Let us fix a symmetric encryption scheme $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$. Two parties share a key K for this scheme, this key having been generated as $K \xleftarrow{R} \mathcal{K}$. The adversary does not a priori know K . We now want to explore the issue of what security (in this case, privacy) of the scheme might mean.

The adversary is assumed able to capture any ciphertext that flows on the channel between the two parties. It can thus collect ciphertexts, and try to glean something from them. Our first question is: what exactly does “glean” mean? What

tasks, were the adversary to accomplish them, would make us declare the scheme insecure? And, correspondingly, what tasks, were the adversary unable to accomplish them, would make us declare the scheme secure?

It is easier to think about *insecurity* than security, because we can certainly identify adversary actions that indubitably imply the scheme is insecure. For example, if the adversary can, from a few ciphertexts, derive the underlying key K , it can later decrypt anything it sees, so if the scheme allowed easy key recovery from a few ciphertexts it is definitely insecure. Yet, an absence of easy key recovery is not enough for the scheme to be secure; maybe the adversary can do something else.

One might want to say something like: given C , the adversary has no idea what M is. This however cannot be true, because of what is called a priori information. Often, something about the message is known. For example, it might be a packet with known headers. Or, it might be an English word. So the adversary, and everyone else, has some information about the message even before it is encrypted.

One might also try to say that what we want is: given ciphertext C , the adversary can't easily recover the plaintext M . But actually, this isn't good enough. The reason is that the adversary might be able to figure out *partial information* about M . 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 m , 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 were 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.

As an example, consider the ECB encryption scheme of Example 4.3. Given the ciphertext, can an eavesdropping adversary figure out the message? Hard to see how, since it does not know K , and if F is a “good” block cipher, then it ought to have a hard time inverting F_K without knowledge of the underlying key. Nonetheless this is not a good scheme. Consider just the case $n = 1$ of a single block message. Suppose I have just two messages, 0^n for “buy” and 1^n 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 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 an encryption of a particular message may be different each time. In terms of our setup it means \mathcal{E} is a *probabilistic* or *stateful* algorithm. That’s why we defined symmetric encryption schemes, above, to allow these types of algorithms.

The reason this is dramatic is that it goes in many ways against the historical or popular notion of encryption. Encryption was once thought of as a code, a fixed mapping of plaintexts to ciphertexts. But this is not the contemporary viewpoint. A single plaintext should have many possible ciphertexts (depending on the random choices or the state of the encryption algorithm). Yet it must be possible to decrypt. How is this possible? We have seen several examples above.

Let us now start looking at privacy more formally. We will begin with the information-theoretic notion of perfect privacy introduced by Shannon, and analyze the one-time pad scheme in this light. Perfect security, however, requires a key as long as the total amount of data encrypted, and this is not usually practical. So we then look at a notion of “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. This is the important notion for us and will be used to analyze the security of schemes such as those presented above.

4.4 Indistinguishability under chosen-plaintext attack

We have already discussed the issues in Section 4.3 above and will now distill a formal definition of security.

4.4.1 Definition

Consider an adversary (not in possession of the secret key) who chooses two messages of the same length, M_0 and M_1 . Then, one is encrypted and the ciphertext is given to the adversary. As a first cut, the scheme is to be considered secure if the adversary

has a hard time telling which message was encrypted.

We will give the adversary a little more power, letting her choose a whole sequence of messages. First, a “challenge” bit b is chosen at random. Now the adversary chooses a sequence of pairs of messages, $(M_1^0, M_1^1), \dots, (M_q^0, M_q^1)$, where, in each pair, the two messages have the same length. We give to the adversary a sequence of ciphertexts C_1, \dots, C_q , where $C_i \leftarrow \mathcal{E}_K(M_i^b)$. Note that in these encryptions, the encryption algorithm uses fresh coins, or an updated state, each time. The adversary gets the sequence of ciphertexts and must guess the bit b to win. In other words, the adversary is trying to determine whether the sender sent M_1^0, \dots, M_q^0 or M_1^1, \dots, M_q^1 .

To further empower the adversary, we let it choose the sequence of message pairs via a *chosen plaintext attack*. This means that the adversary chooses the first pair, then receives C_1 , then chooses the second pair, receives C_2 , and so on.

Let us now formalize this. We fix some encryption scheme $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$. It could be either stateless or stateful. We consider an adversary A . It is a program which has access to an oracle to which it can provide as input any pair (M^0, M^1) of equal-length messages. The oracle will return a ciphertext. We will consider two possible ways in which this ciphertext is computed by the oracle, corresponding to two possible “worlds” in which the adversary “lives”. To do this, first define the *left-or-right encryption oracle* $\mathcal{E}_K(\text{LR}(\cdot, \cdot, b))$, as follows:

Oracle $\mathcal{E}_K(\text{LR}(M_0, M_1, b)) \quad // \quad b \in \{0, 1\} \text{ and } M_0, M_1 \in \{0, 1\}^*$
 $C \leftarrow \mathcal{E}_K(M_b)$
 Return C

The oracle encrypts one of the messages, the choice of which being made according to the bit b . Now the two worlds are as follows:

world 0: The oracle provided to the adversary is $\mathcal{E}_K(\text{LR}(\cdot, \cdot, 0))$. So, whenever the adversary makes a query (M_0, M_1) to its oracle, the oracle computes $C \stackrel{R}{\leftarrow} \mathcal{E}_K(M_0)$, and returns C as the answer.

world 1: The oracle provided to the adversary is $\mathcal{E}_K(\text{LR}(\cdot, \cdot, 1))$. So, whenever the adversary makes a query (M_0, M_1) to its oracle, the oracle computes $C \stackrel{R}{\leftarrow} \mathcal{E}_K(M_1)$, and returns C as the answer.

We call the first world (or oracle) the “left” world (or oracle), and we call the second world (or oracle) the “right” world (or oracle). The problem for the adversary is, after talking to its oracle for some time, to tell which of the two oracles it was given. Before we pin this down, let us further clarify exactly how the oracles operate.

Think of an oracle as a subroutine to which A has access. Adversary A can make an oracle query (M_0, M_1) by calling the subroutine with arguments (M_0, M_1) . In one step, the answer is then returned. Adversary A has no control on how the answer is computed, nor can A see the inner workings of the subroutine, which will typically depend on secret information that A is not provided. Adversary A has

only an interface to the subroutine—the ability to call it as a black-box, and get back an answer.

First assume the given symmetric encryption scheme \mathcal{SE} is stateless. The oracle, in either world, is probabilistic, because it calls the encryption algorithm. Recall that this algorithm is probabilistic. Above, when we say $C \stackrel{R}{\leftarrow} \mathcal{E}_K(M_b)$, it is implicit that the oracle picks its own random coins implicitly and uses them to compute ciphertext C .

The random choices of the encryption function are somewhat “under the rug” here, not being explicitly represented in the notation. But these random bits should not be forgotten. They are central to the meaningfulness of the notion, as also the security of the schemes.

If the given symmetric encryption scheme \mathcal{SE} was stateful, the oracles, in either world, become stateful, too. (Think of a subroutine that maintains a “static” variable across successive calls.) In oracle (either one) begins with a state value initialized to a value specified by the encryption scheme. For example, in CTRC mode, the state is an integer ctr that is initialized to 0. Now, each time the oracle is invoked, it computes $\mathcal{E}_K(M_b)$ according to the specification of algorithm \mathcal{E} . The algorithm may, as a side-effect, update the state, and upon the next invocation of the oracle, the new state value will be used.

We clarify that the choice of which world we are in is made once, at the before the adversary starts to interact with the oracle. In world 0, *all* message pairs sent to the oracle are answered by the oracle encrypting the left message in the pair, while in world 1, all message pairs are answered by the oracle encrypting the right message in the pair. The choice of which does not flip-flop from oracle query to oracle query.

We consider an encryption scheme to be “secure against chosen-plaintext attack” if a “reasonable” adversary cannot obtain “significant” advantage in distinguishing the cases $b = 0$ and $b = 1$ given access to the oracle, where reasonable reflects its resource usage. The technical notion is called indistinguishability under chosen-plaintext attack, denoted IND-CPA.

Before presenting it we need to discuss a subtle point. There are certain queries that an adversary can make to its lr-encryption oracle which will definitely enable it to learn the value of the hidden bit b (meaning figure out in which world it is) but which we consider illegitimate. One is to query the oracle with messages M_0, M_1 of different lengths. We do not ask that encryption hide the length of the plaintext, and indeed common schemes reveal this because the length of the ciphertext depends on the length of the plaintext, so an adversary making such a query could easily win. Another, less obvious attack is for the adversary to make a query M_0, M_1 of equal-length messages such that $\mathcal{E}_K(M_0) \neq \perp$ and $\mathcal{E}_K(M_1) = \perp$. (If the scheme is stateless, this means M_0 is in the plaintext space and M_1 is not.) For some schemes, it is easy for the adversary to find such messages. However, the response of the lr-encryption oracle then gives away the bit b . We have chosen to deal with these issues by simply disallowing the adversary from making such queries. That is, let us

say that an adversary is *illegitimate* if (for some coins it might be provided and for some sequence of oracle responses it might be given) it either makes an lr-encryption query consisting of two messages of different lengths or it makes an lr-encryption query M_0, M_1 for which $\mathcal{E}_K(M_0) = \perp$ or $\mathcal{E}_K(M_1) = \perp$. The adversary is legitimate if it is not illegitimate.

The issue of legitimacy can, once discussed, be forgotten, since in all our reductions and results we will have only legitimate adversaries. But we do have to deal with this issue in the definition.

Definition 4.7 Let $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a symmetric encryption scheme, let $b \in \{0, 1\}$, and let A be an algorithm that has access to an oracle that takes input a pair of strings and returns a string. We consider the following experiment:

Experiment $\mathbf{Exmt}_{\mathcal{SE}}^{\text{ind-cpa-}b}(A)$
 $K \xleftarrow{R} \mathcal{K}$
 $b' \leftarrow A^{\mathcal{E}_K(\text{LR}(\cdot, \cdot, b))}$
 Return b'

The *IND-CPA advantage* of A is defined as

$$\mathbf{Adv}_{\mathcal{SE}}^{\text{ind-cpa}}(A) = \Pr \left[\mathbf{Exmt}_{\mathcal{SE}}^{\text{ind-cpa-1}}(A) = 1 \right] - \Pr \left[\mathbf{Exmt}_{\mathcal{SE}}^{\text{ind-cpa-0}}(A) = 1 \right]$$

if A is legitimate, and 0 otherwise. For any t, q, μ we define the *IND-CPA advantage* of \mathcal{SE} via

$$\mathbf{Adv}_{\mathcal{SE}}^{\text{pr-cpa}}(t, q, \mu) = \max_A \{ \mathbf{Adv}_{\mathcal{SE}}^{\text{ind-cpa}}(A) \}$$

where the maximum is over all legitimate A having time-complexity t , making to the oracle at most q queries the sum of whose lengths is at most μ bits. ■

We discuss some important conventions. The *time-complexity* mentioned above is the worst case total execution time of A , regardless of A 's coins or the answers returned by A 's oracle queries, plus the size of the code of the adversary A , in some fixed RAM model of computation. This convention for measuring time complexity is the same as used in other parts of these notes. Another convention we make is that the length of a query M_0, M_1 to a left-or-right encryption oracle is defined as $|M_0|$. (We can assume this equals $|M_1|$ since the adversary is assumed to be legitimate.) This convention is used in measuring the parameter μ .

If $\mathbf{Adv}_{\mathcal{SE}}^{\text{ind-cpa}}(A)$ is small (meaning close to zero), it means that A is outputting 1 about as often in world 0 as in world 1, meaning it is not doing a good job of telling which world it is in. If this quantity is large (meaning close to one) then the adversary A is doing well, meaning our scheme \mathcal{SE} is not secure.

Informally, for symmetric encryption scheme \mathcal{SE} to be secure against chosen plaintext attack, the IND-CPA advantage of an adversary must be small, no matter what strategy the adversary tries. However, we expect that the advantage grows as

the adversary invests more effort in the process. To capture this we have defined the advantage function $\mathbf{Adv}_{\mathcal{SE}}^{\text{pr-cpa}}(\cdot, \cdot, \cdot)$ as above. This is a function associated to any symmetric encryption scheme \mathcal{SE} . This function is fixed once we fix the encryption scheme. The resources of the adversary we have chosen to use in the parameterization are three. First, its time-complexity, measured according to the convention above. Second, the number of oracle queries, or the number of message pairs the adversary asks of its oracle. These messages may have different lengths, and our third parameter is the sum of all these lengths, denoted μ , again measured according to the convention above. The IND-CPA advantage function of the scheme measures the maximum probability that the security of the scheme \mathcal{SE} can be compromised by an adversary using the indicated resources.

4.4.2 Alternative interpretation of lr-advantage

Why is the $\mathbf{Adv}_{\mathcal{SE}}^{\text{ind-cpa}}(A)$ called the “advantage” of the adversary? We can view the task of the adversary as trying to guess which world it is in. A trivial guess is for the adversary to return a random bit. In that case, it has probability $1/2$ of being right. Clearly, it has not done anything damaging in this case. The advantage of the adversary measures how much better than this it does at guessing which world it is in, namely the excess over $1/2$ of the adversary’s probability of guessing correctly. In this subsection we will see how the above definition corresponds to this alternative view, a view that lends some extra intuition to the definition and is also useful in later usages of the definition.

As usual we fix a symmetric encryption scheme $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$. We now consider the following game, or experiment.

Experiment $\mathbf{Exmt}_{\mathcal{SE}}^{\text{ind-cpa}'}(A)$
 Pick a bit b at random
 Let $K \xleftarrow{R} \mathcal{K}$
 $b' \leftarrow A^{\mathcal{E}_K(\text{LR}(\cdot, b))}$
 If $b = b'$ return 1 else return 0

Here, A is run with an oracle for world b , where the bit b is chosen at random. A eventually outputs a bit b' , its guess as to the value of b . The experiment returns 1 if A ’s guess is correct. Thus

$$\Pr \left[\mathbf{Exmt}_{\mathcal{SE}}^{\text{ind-cpa}'}(A) = 1 \right]$$

is the probability that A correctly guesses which world it is in. (The probability is over the initial choice of world as given by the bit b , the choice of K , the random choices of $\mathcal{E}_K(\cdot)$ if any, and the coins of A if any.) This value is $1/2$ when the adversary does not deserve any advantage, since one can guess b correctly by a strategy as simple as “always answer zero” or “answer with a random bit.” So we re-scale the value and define

$$\mathbf{Adv}_{\mathcal{SE}}^{\text{ind-cpa}'}(A) = 2\Pr \left[\mathbf{Exmt}_{\mathcal{SE}}^{\text{ind-cpa}'}(A) = 1 \right] - 1$$

The following proposition says that this rescaled advantage is exactly the same measure as before.

Proposition 4.8 Let \mathcal{SE} be a symmetric encryption scheme and let A be an adversary. Then

$$\mathbf{Adv}_{\mathcal{SE}}^{\text{ind-cpa}'}(A) = \mathbf{Adv}_{\mathcal{SE}}^{\text{ind-cpa}}(A).$$

■

Proof of Proposition 4.8: We let $\Pr[\cdot]$ be the probability of event “ \cdot ” in the experiment $\mathbf{Exmt}_{\mathcal{SE}}^{\text{ind-cpa}'}(A)$, and refer below to quantities in this experiment. The claim of the Proposition follows by a straightforward calculation:

$$\begin{aligned} & \Pr[\mathbf{Exmt}_{\mathcal{SE}}^{\text{ind-cpa}'}(A) = 1] \\ &= \Pr[b = g] \\ &= \Pr[b = b' \mid b = 1] \cdot \Pr[b = 1] + \Pr[b = b' \mid b = 0] \cdot \Pr[b = 0] \\ &= \Pr[b = b' \mid b = 1] \cdot \frac{1}{2} + \Pr[b = b' \mid b = 0] \cdot \frac{1}{2} \\ &= \Pr[b' = 1 \mid b = 1] \cdot \frac{1}{2} + \Pr[b' = 0 \mid b = 0] \cdot \frac{1}{2} \\ &= \Pr[b' = 1 \mid b = 1] \cdot \frac{1}{2} + (1 - \Pr[b' = 1 \mid b = 0]) \cdot \frac{1}{2} \\ &= \frac{1}{2} + \frac{1}{2} \cdot (\Pr[b' = 1 \mid b = 1] - \Pr[b' = 1 \mid b = 0]) \\ &= \frac{1}{2} + \frac{1}{2} \cdot (\Pr[\mathbf{Exmt}_{\mathcal{SE}}^{\text{ind-cpa-1}}(A) = 1] - \Pr[\mathbf{Exmt}_{\mathcal{SE}}^{\text{ind-cpa-0}}(A) = 1]) \\ &= \frac{1}{2} + \frac{1}{2} \cdot \mathbf{Adv}_{\mathcal{SE}}^{\text{ind-cpa}}(A). \end{aligned}$$

We began by expanding the quantity of interest via standard conditioning. The term of $1/2$ in the third line emerged because the choice of b is made at random. In the fourth line we noted that if we are asking whether $b = b'$ given that we know $b = 1$, it is the same as asking whether $b' = 1$ given $b = 1$, and analogously for $b = 0$. In the fifth line and sixth lines we just manipulated the probabilities and simplified. The next line is important; here we observed that the conditional probabilities in question are exactly the success probabilities in the real and random games respectively. That meant we had recovered the advantage, as desired. ■

4.5 Examples of chosen-plaintext attacks

We illustrate the use of the model in finding attacks by providing an attack on ECB mode, and also a general attack on deterministic, stateless schemes.

4.5.1 Attack on ECB

Let us fix a block cipher $E: \mathcal{K} \times \{0, 1\}^n \rightarrow \{0, 1\}^n$. The ECB symmetric encryption scheme $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ was described as Scheme 4.3. Suppose an adversary sees a ciphertext $C = \mathcal{E}_K(M)$ corresponding to some unknown plaintext M , encrypted under the key K also unknown to the adversary. Can the adversary recover M ? Not easily, if E is a “good” block cipher. For example if E is AES, it seems quite infeasible. Yet, we have already discussed how infeasibility of recovering plaintext from ciphertext is not an indication of security. ECB has other weaknesses. Notice that if two plaintexts M and M' agree in the first block, then so do the corresponding ciphertexts. So an adversary, given the ciphertexts, can tell whether or not the first blocks of the corresponding plaintexts are the same. This is loss of partial information about the plaintexts, and is not permissible in a secure encryption scheme.

It is a test of our definition to see that the definition captures these weaknesses and also finds the scheme insecure. It does. To show this, we want to show that there is an adversary that has a high IND-CPA advantage while using a small amount of resources. This is what the following proposition says.

Proposition 4.9 Let $E: \mathcal{K} \times \{0, 1\}^n \rightarrow \{0, 1\}^n$ be a block cipher, and $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ the corresponding ECB symmetric encryption scheme as described in Scheme 4.3. Then

$$\text{Adv}_{\mathcal{SE}}^{\text{pr-cpa}}(t, 1, 2n) = 1$$

for $t = O(n)$ plus the time for two applications of the block cipher. ■

The advantage of this adversary is 1 even though it uses hardly any resources: just one query, and not a long one at that. That is clearly an indication that the scheme is insecure.

Proof of Proposition 4.9: We will present an adversary algorithm A , having time-complexity t , making 1 query to its oracle, this query being of length $2n$, and having

$$\text{Adv}_{\mathcal{SE}}^{\text{ind-cpa}}(A) = 1.$$

The Proposition follows.

Remember the adversary A is given a lr-encryption oracle $\mathcal{E}_K(\text{LR}(\cdot, \cdot, b))$ which takes input a pair of messages, and returns an encryption of either the left or the right message in the pair, depending on the value of b . The goal of A is to determine the value of b . Our adversary works like this:

Adversary $A^{\mathcal{E}_K(\text{LR}(\cdot, \cdot, b))}$

$M_1 \leftarrow 0^{2n}; M_0 \leftarrow 0^n \| 1^n$

$C[1]C[2] \leftarrow \mathcal{E}_K(\text{LR}(M_0, M_1, b))$

If $C[1] = C[2]$ then return 1 else return 0

The adversary's single oracle query is the pair of messages M_0, M_1 . Since each of them is two blocks long, so is the ciphertext computed according to the ECB scheme. Now, we claim that

$$\begin{aligned} \Pr \left[\mathbf{Exmt}_{\mathcal{SE}}^{\text{ind-cpa-1}}(A) = 1 \right] &= 1 \text{ and} \\ \Pr \left[\mathbf{Exmt}_{\mathcal{SE}}^{\text{ind-cpa-0}}(A) = 1 \right] &= 0. \end{aligned}$$

Hence $\mathbf{Adv}_{\mathcal{SE}}^{\text{ind-cpa}}(A) = 1 - 0 = 1$. And A achieved this advantage by making just one oracle query, whose length, which as per our conventions is just the length of M_0 , is $2n$ bits. So $\mathbf{Adv}_{\mathcal{SE}}^{\text{pr-cpa}}(t, 1, 2n) = 1$.

Why are the two equations claimed above true? You have to return to the definitions of the quantities in question, and trace through the experiments defined there. In world 1, meaning $b = 1$, the oracle returns $C[1]C[2] = E_K(0^n) \| E_K(0^n)$, so $C[1] = C[2]$ and A returns 1. In world 0, meaning $b = 0$, the oracle returns $C[1]C[2] = E_K(0^n)E_K(1^n)$. Since E_K is a permutation, $C[1] \neq C[2]$. So A returns 0 in this case. ■

As an exercise, try to analyze the same adversary as an adversary against CBC or CTR modes, and convince yourself that the adversary will not get a high advantage.

There is an important feature of this attack that must be emphasized. Namely, ECB is an insecure encryption scheme *even if the underlying block cipher E is highly secure*. The weakness is not in the tool being used, the block cipher, but in the manner we are using it. It is the ECB mechanism that is at fault. Even a good tool is useless if you don't use it well.

This is the kind of design flaw that we want to be able to spot and eradicate. Our goal is to find symmetric encryption schemes that are secure as long as the underlying block cipher is secure. In other words, the scheme has no inherent flaw. As long as you use good ingredients, the recipe produces a good meal. If you don't use good ingredients? Well, that is your problem.

4.5.2 Deterministic, stateless schemes are insecure

ECB mode is deterministic and stateless, so that if the same message is encrypted twice, the same ciphertext is returned. It turns out that this property, in general, results in an insecure scheme, and provides perhaps a better understanding of why ECB fails. Let us state the general fact more precisely.

Proposition 4.10 Let $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a deterministic, stateless symmetric encryption scheme. Assume there is an integer m such that the plaintext space of the scheme contains two distinct strings of length m . Then

$$\mathbf{Adv}_{\mathcal{SE}}^{\text{pr-cpa}}(t, 2, 2m) = 1$$

for $t = O(m)$ plus the time for two encryptions. ■

The requirement being made on the message space is minimal; typical schemes have message spaces containing all strings of lengths between some minimum and maximum length, possibly restricted to strings of some given multiples. Note that this Proposition applies to ECB and is enough to show the latter is insecure (but Proposition 4.9 shows something a little stronger because there there is only one query rather than two).

Proof of Proposition 4.10: We will present an adversary algorithm A , having time-complexity t , making 2 queries to its oracle, each query being of length m , and having

$$\mathbf{Adv}_{\mathcal{SE}}^{\text{ind-cpa}}(A) = 1.$$

The Proposition follows.

Remember the adversary A is given a lr-encryption oracle $\mathcal{E}_K(\text{LR}(\cdot, \cdot, b))$ which takes input a pair of messages, and returns an encryption of either the left or the right message in the pair, depending on the value of b . The goal of A is to determine the value of b . Our adversary works like this:

Adversary A^f

Let X, Y be distinct, m -bit strings in the plaintext space

$C_1 \leftarrow f(X, Y)$

$C_2 \leftarrow F(Y, Y)$

If $C_1 = C_2$ then return 1 else return 0

Now, we claim that

$$\Pr \left[\mathbf{Exmt}_{\mathcal{SE}}^{\text{ind-cpa-1}}(A) = 1 \right] = 1$$

$$\Pr \left[\mathbf{Exmt}_{\mathcal{SE}}^{\text{ind-cpa-0}}(A) = 1 \right] = 0.$$

Hence $\mathbf{Adv}_{\mathcal{SE}}^{\text{ind-cpa}}(A) = 1 - 0 = 1$. And A achieved this advantage by making two oracle query, each of whose length, which as per our conventions is just the length of the first message, is m bits. So $\mathbf{Adv}_{\mathcal{SE}}^{\text{pr-cpa}}(t, 2, 2m) = 1$.

Why are the two equations claimed above true? In world 1, meaning $b = 1$, the oracle returns $C_1 = \mathcal{E}_K(Y)$ and $C_2 = \mathcal{E}_K(Y)$, and since the encryption function is deterministic and stateless, $C_1 = C_2$, so A returns 1. In world 0, meaning $b = 0$, the oracle returns $C_1 = \mathcal{E}_K(X)$ and $C_2 = \mathcal{E}_K(Y)$, and since it is required that decryption be able to recover the message, it must be that $C_1 \neq C_2$. So A returns 0. ■

4.5.3 Attack on CBC encryption with a counter IV

4.6 Security against plaintext recovery

In Section 4.3 we noted a number of security properties that are necessary but not sufficient for security. For example, it should be computationally infeasible for an

adversary to recover the key from a few plaintext-ciphertext pairs, or to recover a plaintext from a ciphertext. A test of our definition is that it implies these properties, in the sense that a scheme that is secure in the sense of our definition is also secure against key-recovery or plaintext-recovery.

The situation is analogous to what we saw in the case of PRFs. There we showed that a secure PRF is secure against key-recovery. In order to have some variation, this time we choose a different property, namely plaintext recovery. We formalize this, and then show if there was an adversary B capable of recovering the plaintext from a given ciphertext, then this would enable us to construct an adversary A that broke the scheme in the IND-CPA sense, meaning figured out which of the two worlds it is in. But if the scheme is secure in the IND-CPA sense, that latter adversary could not exist. Hence, neither could the former.

The idea of this argument illustrates how we convince ourselves that the above definition is good, and captures all the properties we might want for security against chosen plaintext attack. Take some other property that you feel a secure scheme should have: infeasibility of key recovery from a few plaintext-ciphertext pairs; infeasibility of predicting the XOR of the plaintext bits; etc. Imagine there was an adversary B that was successful at this task. We claim this would enable us to construct an adversary A that broke the scheme in the left-or-right sense, and hence B does not exist if the scheme is secure in the left-or-right sense. More precisely, we would use the advantage function of the scheme to bound the probability that adversary B succeeds. Assuming the advantage function is small at the specified parameter values, so is the chance that adversary B succeeds.

Let us now go through the plaintext recovery example in detail. The task facing the adversary will be to decrypt a ciphertext which was formed by encrypting a randomly chosen challenge message of some length m . In the process we want to give the adversary the ability to see plaintext-ciphertext pairs, and capture this by giving it access to an encryption oracle. This encryption oracle is not the Ir-encryption oracle we saw above: instead, it simply takes input a single message M and returns a ciphertext $C \stackrel{R}{\leftarrow} \mathcal{E}_K(M)$ computed by encrypting M . To capture providing the adversary with a challenge ciphertext, we choose a random m -bit plaintext M , compute $C \stackrel{R}{\leftarrow} \mathcal{E}_K(M)$, and give C to the adversary. The adversary wins if it can output the plaintext M corresponding to the ciphertext C .

For simplicity we assume the encryption scheme is stateless, and that $\{0, 1\}^m$ is a subset of the plaintext space associated to the scheme. As usual, when either the encryption or the challenge oracle invoke the encryption function, it is implicit that they respect the randomized nature of the encryption function, meaning the latter tosses coins anew upon each invocation of the oracle.

Definition 4.11 Let $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a stateless symmetric encryption scheme whose plaintext space includes $\{0, 1\}^m$, and let B be an algorithm that has access to an oracle. We consider the following experiment:

Experiment $\mathbf{Exmt}_{\mathcal{SE}}^{\text{pr-cpa}}(B)$
 $K \xleftarrow{R} \mathcal{K}$
 $M' \xleftarrow{R} \{0, 1\}^m$
 $C \xleftarrow{R} \mathcal{E}_K(M')$
 $M \leftarrow B^{\mathcal{E}_K(\cdot)}(C)$
 If $M = M'$ then return 1 else return 0

The *pr-advantage* of B is defined as

$$\mathbf{Adv}_{\mathcal{SE}}^{\text{pr-cpa}}(B) = \Pr \left[\mathbf{Exmt}_{\mathcal{SE}}^{\text{pr-cpa}}(B) = 1 \right] .$$

For any t, q, μ we define the *pr-advantage* of \mathcal{SE} via

$$\mathbf{Adv}_{\mathcal{SE}}^{\text{pr-cpa}}(t, q, \mu) = \max_B \{ \mathbf{Adv}_{\mathcal{SE}}^{\text{pr-cpa}}(B) \}$$

where the maximum is over all B having time-complexity t , making to the encryption oracle at most q queries the sum of whose lengths is at most μ bits. ■

In the experiment above, B is executed with its oracle and challenge ciphertext C . The adversary B wins if it can correctly decrypt C , and in that case the experiment returns 1. In the process, the adversary can make encryption oracle queries as it pleases.

The following Proposition says that the probability that an adversary successfully recovers a plaintext from a challenge ciphertext cannot exceed the IND-CPA advantage of the scheme (with resource parameters those of the plaintext recovery adversary) plus the chance of simply guessing the plaintext. In other words, security in the IND-CPA sense implies security against plaintext recovery.

Proposition 4.12 Let $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a stateless symmetric encryption scheme whose plaintext space includes $\{0, 1\}^m$. Then

$$\mathbf{Adv}_{\mathcal{SE}}^{\text{pr-cpa}}(t, q, \mu) \leq \mathbf{Adv}_{\mathcal{SE}}^{\text{pr-cpa}}(t, q + 1, \mu + m) + \frac{1}{2^m}$$

for any t, q, μ . ■

The reason this is true is quite simple. If an adversary B were capable of decrypting the challenge ciphertext we could easily build an adversary A_B that, using B as a subroutine, would be able to tell whether it is in world 0 or world 1. In other words, it is a reduction.

Proof of Proposition 4.12: We will show that given any adversary B whose resources are restricted to t, q, μ we can construct an adversary A_B , using resources $t, q + 1, \mu + m$, such that

$$\mathbf{Adv}_{\mathcal{SE}}^{\text{pr-cpa}}(B) \leq \mathbf{Adv}_{\mathcal{SE}}^{\text{ind-cpa}}(A_B) + \frac{1}{2^m} . \quad (4.1)$$

The Proposition follows by the usual maximization process.

As per Definition 4.1, adversary A_B will be provided a lr-encryption oracle, and will try to determine in which world it is. To do so, it will run adversary B as a subroutine. We provide the description followed by an explanation and analysis.

Adversary $A_B^{f(\cdot, \cdot)}$

$M_0 \xleftarrow{R} \{0, 1\}^m ; M_1 \xleftarrow{R} \{0, 1\}^m$

$C \leftarrow f(M_0, M_1)$

Run adversary $B(C)$, replying to its oracle queries as follows

When B makes an oracle query X do

$Y \leftarrow f(X, X)$

Return Y to B as the answer

Until B stops and outputs a plaintext M

If $M = M_1$ then return 1 else return 0

Here A_B is running B and itself providing answers to B 's oracle queries. To make the challenge ciphertext C for B , adversary A_B chooses random messages M_0 and M_1 and uses its lr-oracle to get the encryption C of one of them. When B makes an encryption oracle query X , adversary A_B needs to return $\mathcal{E}_K(X)$. It does this by invoking its lr-encryption oracle, setting both messages in the pair to X , so that regardless of the value of the bit b , the ciphertext returned is an encryption of X , just as B wants. When B outputs a plaintext M , adversary A_B tests whether $M = M_1$ and if so bets that it is in world 1. Else it bets that it is in world 0. Now we claim that

$$\begin{aligned} \Pr \left[\mathbf{Exmt}_{\mathcal{SE}}^{\text{ind-cpa-1}}(A_B) = 1 \right] &\geq \mathbf{Adv}_{\mathcal{SE}}^{\text{pr-cpa}}(B) \\ \Pr \left[\mathbf{Exmt}_{\mathcal{SE}}^{\text{ind-cpa-0}}(A_B) = 1 \right] &\leq 2^{-m} . \end{aligned}$$

We will justify these claims shortly, but first let us use them to conclude. Subtracting, as per Definition 4.1, we get

$$\begin{aligned} \mathbf{Adv}_{\mathcal{SE}}^{\text{ind-cpa}}(A_B) &= \Pr \left[\mathbf{Exmt}_{\mathcal{SE}}^{\text{ind-cpa-1}}(A_B) = 1 \right] - \Pr \left[\mathbf{Exmt}_{\mathcal{SE}}^{\text{ind-cpa-0}}(A_B) = 1 \right] \\ &\geq \mathbf{Adv}_{\mathcal{SE}}^{\text{pr-cpa}}(B) - 2^{-m} . \end{aligned}$$

Re-arranging terms gives us Equation (4.1). It remains to justify Equations (4.2) and (4.2).

Adversary B will return the $M = \mathcal{D}_K(C)$ with probability at least $\mathbf{Adv}_{\mathcal{SE}}^{\text{pr-cpa}}(B)$. In world 1, ciphertext C is an encryption of M_1 , so this means that $M = M_1$ with probability at least $\mathbf{Adv}_{\mathcal{SE}}^{\text{pr-cpa}}(B)$, and thus Equation (4.2) is true. Now assume A_B is in world 0. In that case, A_B will return 1 only if B returns $M = M_1$. But B is given no information about M_1 , since C is an encryption of M_0 and M_1 is chosen

randomly and independently of M_0 . It is simply impossible for B to output M_1 with probability greater than 2^{-m} . Thus Equation (4.2) is true. ■

Similar arguments can be made to show that other desired security properties of a symmetric encryption scheme follow from this definition. For example, is it possible that some adversary B , given some plaintext-ciphertext pairs and then a challenge ciphertext C , can compute the XOR of the bits of $M = \mathcal{D}_K(C)$? Or the sum of these bits? Or the last bit of M ? Its probability of doing any of these cannot be more than marginally above $1/2$ because were it so, we could design an adversary A that won the left-or-right game using resources comparable to those used by B . We leave as an exercise the formulation and working out of other such examples along the lines of Proposition 4.12.

Of course one cannot exhaustively enumerate all desirable security properties. But you should be moving towards being convinced that our notion of left-or-right security covers all the natural desirable properties of security under chosen plaintext attack. Indeed, we err, if anything, on the conservative side. There are some attacks that might in real life be viewed as hardly damaging, yet our definition declares the scheme insecure if it succumbs to one of these. That is all right; there is no harm in making our definition a little demanding. What is more important is that if there is any attack that in real life would be viewed as damaging, then the scheme will fail the left-or-right test, so that our formal notion too declares it insecure.

4.7 Security of CTR encryption

Let $F: \mathcal{K} \times \{0, 1\}^n \rightarrow \{0, 1\}^\ell$ be a family of functions. The CTR symmetric encryption scheme comes in two variants: the randomized (stateless) one of Scheme 4.5 and the counter-based (stateful) one of Scheme 4.6. Both are secure against chosen-plaintext attack, but, interestingly, the counter version is more secure than the randomized version. We will first state the main theorems about the schemes, discuss them, and then prove them. For the counter version we have:

Theorem 4.13 Let $F: \mathcal{K} \times \{0, 1\}^n \rightarrow \{0, 1\}^\ell$ be a family of functions and let $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be the corresponding CTRE symmetric encryption scheme as described in Scheme 4.6. Then for any t, q, μ with $\mu < \ell 2^n$ we have

$$\mathbf{Adv}_{\mathcal{SE}}^{\text{pr-cpa}}(t, q, \mu) \leq 2 \cdot \mathbf{Adv}_F^{\text{prf}}(t, q', nq'),$$

where $q' = \mu/\ell$. ■

And for the randomized version:

Theorem 4.14 Let $F: \mathcal{K} \times \{0, 1\}^n \rightarrow \{0, 1\}^\ell$ be a family of functions and let $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be the corresponding CTR\$ symmetric encryption scheme as described in Scheme 4.5. Then for any t, q, μ with $\mu < \ell 2^n$ we have

$$\mathbf{Adv}_{\mathcal{SE}}^{\text{pr-cpa}}(t, q, \mu) \leq 2 \cdot \mathbf{Adv}_F^{\text{prf}}(t, q', nq') + \frac{\mu(q-1)}{\ell 2^n},$$

where $q' = \mu/\ell$. ■

This kind of result is what this whole approach is about. Namely, we are able to provide provable guarantees of security of some higher level cryptographic construct (in this case, a symmetric encryption scheme) based on the assumption that some building block (in this case an underlying block cipher treated as a PRF) is secure. They are the first example of the “punch-line” we have been building towards. So it is worth pausing at this point and trying to make sure we really understand what these theorems are saying and what are their implications.

If we want to entrust our data to some encryption mechanism, we want to know that this encryption mechanism really provides privacy. If it is ill-designed, it may not. We saw this happen with ECB. Even if we used a secure block cipher, the design flaws of ECB mode made it an insecure encryption scheme.

Flaws are not apparent in CTR at first glance. But maybe they exist. It is very hard to see how one can be convinced they do *not* exist, when one cannot possibly exhaust the space of all possible attacks that could be tried. Yet this is exactly the difficulty that the above theorems circumvent. They are saying that CTR mode *does not have design flaws*. They are saying that as long as you use a good block cipher, you are *assured* that nobody will break your encryption scheme. One cannot ask for more, since if one does not use a good block cipher, there is no reason to expect security anyway. We are thus getting a conviction that *all attacks fail* even though we do not even know exactly how these attacks operate. That is the power of the approach.

Now, one might appreciate that the ability to make such a powerful statement takes work. It is for this that we have put so much work and time into developing the definitions: the formal notions of security that make such results meaningful. For readers who have less experience with definitions, it is worth knowing, at least, that the effort is worth it. It takes time and work to understand the notions, but the payoffs are big: you actually have the ability to get guarantees of security.

How, exactly, are the theorems saying this? The above discussion has pushed under the rug the quantitative aspect that is an important part of the results. It may help to look at a concrete example.

Example 4.15 Let us suppose that F is AES. So the key size is $k = 128$ and the block size is $n = \ell = 128$. Suppose I want to encrypt $q = 2^{40}$ messages, each $128 * 2^3$ bits long, so that I am encrypting a total of $\mu = 2^{50}$ bits of data. Can I do this securely using counter-mode CTR? What is the chance that an adversary figures out something about my data? Well, if the adversary has $t = 2^{60}$ computing cycles, then by definition its chance is not more than $\mathbf{Adv}_{\mathcal{SE}}^{\text{pr-cpa}}(t, q, \mu)$. That has nothing to do with the theorem: it is just our definitions, which say that this is the maximum probability of being able to break the encryption scheme in these given resources. So the question of whether the scheme is secure for my chosen parameters boils down to asking what is the value of $\mathbf{Adv}_{\mathcal{SE}}^{\text{pr-cpa}}(t, q, \mu)$. A priori, we have no idea. But now, we appeal to Theorem 4.13, which says that this chance is at most $2 \cdot \mathbf{Adv}_F^{\text{prf}}(t, q', 128q')$,

where q' is as given in the theorem. Namely $q' = \mu/L = 2^{50}/128 = 2^{43}$. So the question is, what is the value of $\mathbf{Adv}_F^{\text{prf}}(t, q', 128q')$ with these values of t, q' ?

Thus, what the theorem has done is reduce the question of estimating the probability of loss of privacy from the encryption scheme to the question of estimating the pseudorandomness of AES. As per Section 3.5.2, one might conjecture that

$$\mathbf{Adv}_{\text{AES}}^{\text{prf}}(t, q', 128q') = c_1 \cdot \frac{t/T_{\text{AES}}}{2^{128}} + \frac{(q')^2}{2^{128}},$$

where T_{AES} is the time to do one AES computation on our fixed RAM model of computation. Now plug in $t = 2^{60}$ and $q' = 2^{43}$ and take into account what we computed above. We get

$$\begin{aligned} \mathbf{Adv}_{\mathcal{SE}}^{\text{pr-cpa}}(t, q, \mu) &\leq 2 \cdot \mathbf{Adv}_{\text{AES}}^{\text{prf}}(t, q', 128q') \\ &\leq 2c_1 \cdot \frac{t/T_{\text{AES}}}{2^{128}} + \frac{2(q')^2}{2^{128}} \\ &= \frac{2^{61}}{2^{128}} \cdot \frac{c_1}{T_{\text{AES}}} + \frac{2^{43 \cdot 2 + 1}}{2^{128}} \\ &= \frac{1}{2^{67}} \cdot \frac{c_1}{T_{\text{AES}}} + \frac{1}{2^{41}} \\ &\leq \frac{1}{2^{41}}. \end{aligned}$$

In the last step, we made the (very reasonable) assumption that c_1/T_{AES} is at most 2^{26} . Thus, the chance the adversary gets any information about our encrypted data is about 2^{-41} , even though we allow this adversary computing time up to 2^{60} , and are encrypting 2^{50} bits of data. This is a very small chance, and we can certainly live with it. It is in this sense that we say the scheme is secure. ■

Example 4.16 You are encouraged to work out another example along the following lines. Don't assume F is AES, but rather assume it is an even better PRF. It still has $k = n = \ell = 128$, but assume it is not a permutation, so that there are no birthday attacks; specifically, assume

$$\mathbf{Adv}_F^{\text{prf}}(t, q', 128q') = c_1 \cdot \frac{t/T_{\text{AES}}}{2^{128}} + c_1 \cdot \frac{q}{2^{128}}.$$

Now, consider both the counter-based CTR scheme and the randomized one. In the theorems, the difference is the $\mu(q-1)/L2^l$ term. Try to see what kind of difference this makes. For each scheme, consider how high you can push q, μ, t and still have some security left. For which scheme can you push them higher? Which scheme is thus "more secure"? ■

These examples illustrate how to use the theorems to figure out how much security you will get from the CTR encryption scheme in some application.

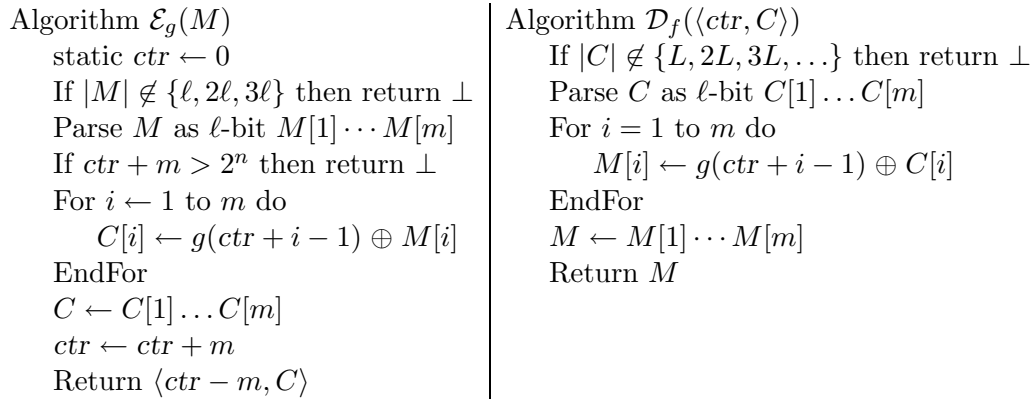


Figure 4.1: Version $\mathcal{SE}[G] = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ of the CTRC scheme parameterized by a family of functions G .

4.7.1 Proof of Theorem 4.13

The paradigm used is quite general in many of its aspects, and we will use it again, not only for encryption schemes, but for other kinds of schemes that are based on pseudorandom functions.

An important observation regarding the CTR scheme is that the encryption and decryption operations do not need direct access to the key K , but only access to a subroutine, or oracle, that implements the function F_K . This is important because one can consider what happens when F_K is replaced by some other function. To consider such replacements, we reformulate the scheme. We introduce a scheme that takes as a parameter any given family of functions G having domain $\{0, 1\}^n$ and range $\{0, 1\}^\ell$. As we will see later the cases of interest are $G = F$ and $G = \text{Rand}(n, \ell)$. Let us first however describe this parameterized scheme. In the rest of this proof, $\mathcal{SE}[G] = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ denotes the symmetric encryption scheme defined as follows. The key generation algorithm simply returns a random instance of G , meaning that it picks a function $g \xleftarrow{R} G$ from family G at random, and views g as the key. The encryptor maintains a counter ctr which is initially zero. The encryption and decryption algorithms are shown in Figure 4.1. In the figure, parsing M means that we divide it into ℓ -bit (not n -bit!) blocks and let $M[i]$ denote the i -th such block. The encryption algorithm updates the counter upon each invocation, and begins with this updated value the next time it is invoked. As the description indicates, the scheme is exactly CTRC, except that function g is used in place of F_K . This seemingly cosmetic change of viewpoint is quite useful, as we will see.

We observe that the scheme in which we are interested, and which the theorem is about, is simply $\mathcal{SE}[F]$ where F is our given family of functions as per the theorem. Now, the proof breaks into two parts. The first step removes F from the picture, and looks instead at an “idealized” version of the scheme. Namely we consider the

scheme $\mathcal{SE}[\text{Rand}(n,\ell)]$. Here, a random function g of n -bits to ℓ -bits is being used where the original scheme would use F_K . We then assess an adversary's chance of breaking this idealized scheme. We argue that this chance is actually zero. This is the main lemma in the analysis.

This step is definitely a thought experiment. No real implementation can use a random function in place of F_K because even storing such a function takes an exorbitant amount of memory. But this analysis of the idealized scheme enables us to focus on any possible weaknesses of the CTR mode itself, as opposed to weaknesses arising from properties of the underlying block cipher. We can show that this idealized scheme is secure, and that means that the mode itself is good.

It then remains to see how this “lifts” to a real world, in which we have no ideal random functions, but rather want to assess the security of the scheme $\mathcal{SE}[F]$ that uses the given family F . Here we exploit the notion of pseudorandomness to say that the chance of an adversary breaking the $\mathcal{SE}[F]$ can differ from its chance of breaking the ideal-world scheme $\mathcal{SE}[\text{Rand}(n,\ell)]$ by an amount not exceeding the probability of breaking the pseudorandomness of F .

Lemma 4.17 Let A be any IND-CPA adversary attacking $\mathcal{SE}[\text{Rand}(n,\ell)]$. Then

$$\text{Adv}_{\mathcal{SE}[\text{Rand}(n,\ell)]}^{\text{ind-cpa}}(A) = 0.$$

■

The lemma considers an arbitrary adversary. Let us say this adversary has time-complexity t , makes q queries to its lr-encryption oracle, these totaling μ bits. The lemma does not care about the values of t , q , or μ . (Recall, however, that $\mu \leq \ell 2^n$; after that maximal number of bits, the encryption mechanism will “shut up” and be of no use.) It says that adversary has zero advantage, meaning no chance at all of breaking the scheme. The fact that no restriction is made on t indicates that the result is information-theoretic: it holds regardless of how much computing time the adversary invests.

Of course, this lemma refers to the idealized scheme, namely the one where the function g being used by the encryption algorithm is random. But remember that ECB was insecure even in this setting. (The attacks we provided for ECB work even if the underlying cipher E is $\text{Perm}(n)$, the family of all permutations on n -bit strings.) So the statement is not content-free; it is saying something quite meaningful and important about the CTR mode. It is not true of all modes.

We postpone the proof of the lemma. Instead we will first see how to use it to conclude the proof of the theorem. The argument here is quite simple and generic.

The lemma tells us that the CTRC encryption scheme is (very!) secure when g is a random function. But we are interested in the case where g is an instance of our given family F . So our worry is that the actual scheme $\mathcal{SE}[F]$ is insecure even though the idealized scheme $\mathcal{SE}[\text{Rand}(n,\ell)]$ is secure. In other words, we worry that there might be an adversary having large IND-CPA advantage in attacking $\mathcal{SE}[F]$,

even though we know that its advantage in attacking $\mathcal{SE}[\text{Rand}(n,\ell)]$ is zero. But we claim that this is not possible if F is a secure PRF. Intuitively, the existence of such an adversary indicates that F is not approximating $\text{Rand}(n,\ell)$ since there is some detectable event, namely the success probability of some adversary in a certain experiment, that happens with high probability when F is used and with low probability when $\text{Rand}(n,\ell)$ is used. To concretize this intuition, let A be a IND-CPA adversary attacking $\mathcal{SE}[F]$. We associate to A a distinguisher D_A that is given oracle access to a function $g: \{0,1\}^n \rightarrow \{0,1\}^\ell$ and is trying to determine which world it is in, where in world 0 g is a random instance of $\text{Rand}(n,\ell)$ and in world 1 g is a random instance of F . We suggest the following strategy to the distinguisher. It runs A , and replies to A 's oracle queries in such a way that A is attacking $\mathcal{SE}[\text{Rand}(n,\ell)]$ in D_A 's world 0, and A is attacking $\mathcal{SE}[F]$ in D_A 's world 1. The reason it is possible for D_A to do this is that it can execute the encryption algorithm $\mathcal{E}_g(\cdot)$ of Figure 4.1, which simply requires access to the function g . If the adversary A wins, meaning it correctly identifies the encryption oracle, D_A bets that g is an instance of F ; otherwise, D_A bets that g is an instance of $\text{Rand}(n,\ell)$.

We stress the key point that makes this argument work. It is that the encryption function of the CTRC scheme invokes the function F_K purely as an oracle. If it had, instead, made direct some direct use of the key K , the paradigm above would not work. The full proof follows.

Proof of Theorem 4.13: Let A be any IND-CPA adversary attacking $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$. Assume A makes q oracle queries totaling μ bits, and has time-complexity t . We will design a distinguisher D_A such that

$$\text{Adv}_{\mathcal{SE}}^{\text{ind-cpa}}(A) \leq 2 \cdot \text{Adv}_F^{\text{prf}}(D_A). \quad (4.2)$$

Furthermore, D_A will make μ/ℓ oracle queries and have time-complexity t . Now, the statement of Theorem 4.13 follows as usual, by taking maximums. So the main thing is to provide the distinguisher for which Equation (4.2) is true. This distinguisher uses A as a subroutine.

Remember that D_A takes an oracle $g: \{0,1\}^n \rightarrow \{0,1\}^\ell$. This oracle is either drawn at random from F or from $\text{Rand}(n,\ell)$ and D_A does not know which. To find out, D_A will use A . But remember that A too gets an oracle, namely an lr-encryption oracle. From A 's point of view, this oracle is simply a subroutine: A can write, at some location, a pair of messages, and is returned a response by some entity it calls its oracle. When D_A runs A as a subroutine, it is D_A that will “simulate” the lr-encryption oracle for A , meaning D_A will provide the responses to any oracle queries that A makes. Here is the description of D_A :

Distinguisher D_A^g

$b \xleftarrow{R} \{0,1\}$

Run adversary A , replying to its oracle queries as follows

When A makes an oracle query (M_0, M_1) do

$C \xleftarrow{R} \mathcal{E}_g(M_b)$
 Return C to A as the answer
 Until A stops and outputs a bit b'
 If $b' = b$ then return 1 else return 0

Here $\mathcal{E}_g(\cdot)$ denotes the encryption function of the generalized CTRC scheme that we defined in Figure 4.1. The crucial fact we are exploiting here is that this function can be implemented given an oracle for g . Distinguisher D_A itself picks the challenge bit b representing the choice of worlds for A , and then sees whether or not A succeeds in guessing the value of this bit. If it does, it bets that g is an instance of F , and otherwise it bets that g is an instance of $\text{Rand}(n, \ell)$. For the analysis, we claim that

$$\Pr \left[\mathbf{Exmt}_F^{\text{prf-1}}(D_A) = 1 \right] = \frac{1}{2} + \frac{1}{2} \cdot \mathbf{Adv}_{\mathcal{SE}[F]}^{\text{ind-cpa}}(A) \quad (4.3)$$

$$\Pr \left[\mathbf{Exmt}_F^{\text{prf-0}}(D_A) = 1 \right] = \frac{1}{2} + \frac{1}{2} \cdot \mathbf{Adv}_{\mathcal{SE}[\text{Rand}(n, \ell)]}^{\text{ind-cpa}}(A). \quad (4.4)$$

We will justify these claims shortly, but first let us use them to conclude. Subtracting, as per Definition 3.4, we get

$$\begin{aligned} \mathbf{Adv}_F^{\text{prf}}(D_A) &= \Pr \left[\mathbf{Exmt}_F^{\text{prf-1}}(D_A) = 1 \right] - \Pr \left[\mathbf{Exmt}_F^{\text{prf-0}}(D_A) = 1 \right] \\ &= \frac{1}{2} \cdot \mathbf{Adv}_{\mathcal{SE}[F]}^{\text{ind-cpa}}(A) - \frac{1}{2} \cdot \mathbf{Adv}_{\mathcal{SE}[\text{Rand}(n, \ell)]}^{\text{ind-cpa}}(A) \\ &= \frac{1}{2} \cdot \mathbf{Adv}_{\mathcal{SE}[F]}^{\text{ind-cpa}}(A). \end{aligned} \quad (4.5)$$

The last inequality was obtained by applying Lemma 4.17, which told us that the term $\mathbf{Adv}_{\mathcal{SE}[\text{Rand}(n, \ell)]}^{\text{ind-cpa}}(A)$ was simply zero. Re-arranging terms gives us Equation (4.2). Now let us check the resource usage. Each computation $\mathcal{E}_g(M_b)$ requires $|M_b|/\ell$ applications of g , and hence the total number of queries made by D_A to its oracle g is μ/ℓ . The time-complexity of D_A equals that of A once one takes into account the convention that time-complexity refers to the time of the entire underlying experiment. It remains to justify Equations (4.3) and (4.4).

Distinguisher D_A returns 1 when $b = b'$, meaning that IND-CPA adversary A correctly identified the world b in which it was placed, or, in the language of Section 4.4.2, made the “correct guess.” The role played by D_A ’s world is simply to alter the encryption scheme for which this is true. When D_A is in world 1, the encryption scheme, from the point of view of A , is $\mathcal{SE}[F]$, and when D_A is in world 0, the encryption scheme, from the point of view of A , is $\mathcal{SE}[\text{Rand}(n, \ell)]$. Thus, using the notation from Section 4.4.2, we have

$$\begin{aligned} \Pr \left[\mathbf{Exmt}_F^{\text{prf-1}}(D_A) = 1 \right] &= \Pr \left[\mathbf{Exmt}_{\mathcal{SE}[F]}^{\text{ind-cpa}'}(A) = 1 \right] \\ \Pr \left[\mathbf{Exmt}_F^{\text{prf-0}}(D_A) = 1 \right] &= \Pr \left[\mathbf{Exmt}_{\mathcal{SE}[\text{Rand}(n, \ell)]}^{\text{ind-cpa}'}(A) = 1 \right]. \end{aligned}$$

To obtain Equations (4.3) and (4.4) we can now apply Proposition 4.8. ■

For someone unused to PRF-based proofs of security the above may seem complex, but the underlying idea is actually very simple, and will be seen over and over again. It is simply that one can view the experiment of the IND-CPA adversary attacking the encryption scheme as information about the underlying function g being used, and if the adversary has more success in the case that g is an instance of F than that g is an instance of $\text{Rand}(n, \ell)$, then we have a distinguishing test between F and $\text{Rand}(n, \ell)$. Let us now prove the lemma about the security of the idealized CTRC scheme.

Proof of Lemma 4.17: The intuition is simple. When g is a random function, its value on successive counter values yields a one-time pad, a truly random and unpredictable sequence of bits. As long as the number of data bits encrypted does not exceed $\ell 2^n$, we invoke g only on distinct values in the entire encryption process. And if an encryption would result in more queries than this, the algorithm simply shuts up, so we can ignore this. The outputs of g are thus random. Since the data is XORed to this sequence, the adversary gets no information whatsoever about it.

Now, we must make sure that this intuition carries through in our setting. Our lemma statement makes reference to our notions of security, so we must use the setup in Section 4.4.1. The adversary A has access to an lr-encryption oracle. Since the scheme we are considering is $\mathcal{SE}[\text{Rand}(n, \ell)]$, the oracle is $\mathcal{E}_g(\text{LR}(\cdot, \cdot, b))$, where the function \mathcal{E}_g was defined in Figure 4.1, and g is a random instance of $\text{Rand}(n, \ell)$, meaning a random function.

The adversary makes some number q of oracle queries. Let $(M_{i,0}, M_{i,1})$ be the i -th query, and let m_i be the number of blocks in $M_{i,0}$. (This is the same as the number of blocks in $M_{i,1}$.) Let $M_{i,c}[j]$ be the value of the j -th ℓ -bit block of $M_{i,b}$ for $b \in \{0, 1\}$. Let C'_i be the response returned by the oracle to query $(M_{i,0}, M_{i,1})$. It consists of a value that encodes the counter value, together with m_i blocks of ℓ bits each, $C_i[1] \dots C_i[m_i]$. Pictorially:

$$\begin{aligned} M_{1,b} &= M_{1,b}[1]M_{1,b}[1] \dots M_{1,b}[m_1] \\ C_1 &= \langle 0, C_1[1] \dots C_1[m_1] \rangle \\ M_{2,b} &= M_{2,b}[1]M_{2,b}[2] \dots M_{2,b}[m_2] \\ C_2 &= \langle m_1, C_2[1] \dots C_2[m_2] \rangle \\ &\vdots \\ &\vdots \\ M_{q,b} &= M_{q,b}[1]M_{q,b}[2] \dots M_{q,b}[m_q] \\ C_q &= \langle m_1 + \dots + m_{q-1}, C_q[1] \dots C_q[m_q] \rangle \end{aligned}$$

What kind of distribution do the outputs received by A have? We claim that the $m_1 + \dots + m_q$ values $C_i[j]$ ($i = 1, \dots, q$ and $j = 1, \dots, m_i$) are randomly and independently distributed, not only of each other, but of the queried messages and

<pre> Algorithm $\mathcal{E}_g(M)$ If $M \notin \{\ell, 2\ell, \dots, \ell 2^n\}$ then return \perp Parse M as ℓ-bit $M[1] \cdots M[m]$ $r \xleftarrow{R} [0..2^n - 1]$ For $i \leftarrow 1$ to m do $C[i] \leftarrow g(r + i - 1) \oplus M[i]$ EndFor $C \leftarrow C[1] \cdots C[m]$ Return $\langle r, C \rangle$ </pre>	<pre> Algorithm $\mathcal{D}_f(\langle r, C \rangle)$ If $C \notin \{\ell, 2\ell, \dots, \ell 2^n\}$ then return \perp Parse C as ℓ-bit $C[1] \cdots C[m]$ For $i \leftarrow 1$ to m do $M[i] \leftarrow g(r + i - 1) \oplus C[i]$ EndFor $M \leftarrow M[1] \cdots M[m]$ Return M </pre>
--	---

Figure 4.2: Version $\mathcal{SE}[G] = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ of the CTR\$ scheme parameterized by a family of functions G .

the bit b , and moreover this is true in both worlds. Why? Here is where we use a crucial property of the CTR mode, namely that it XORs data with the value of g on a counter. We observe that according to the scheme

$$C_i[j] = g([m_1 + \cdots + m_{i-1} + j]_i) \oplus \begin{cases} M_{i,1}[j] & \text{if we are in world 1} \\ M_{i,0}[j] & \text{if we are in world 0.} \end{cases}$$

Now, we can finally see that the idea we started with is really the heart of it. The values on which g is being applied above are all distinct. So the outputs of g are all random and independent. It matters not, then, what we XOR these outputs with; what comes back is just random.

This tells us that any given output sequence from the oracle is equally likely in both worlds. Since the adversary determines its output bit based on this output sequence, its probability of its returning 1 must be the same in both worlds,

$$\Pr \left[\mathbf{Exmt}_{\mathcal{SE}[\text{Rand}(n,\ell)]}^{\text{ind-cpa-1}}(A) = 1 \right] = \Pr \left[\mathbf{Exmt}_{\mathcal{SE}[\text{Rand}(n,\ell)]}^{\text{ind-cpa-0}}(A) = 1 \right].$$

Hence A 's IND-CPA advantage is zero. ■

4.7.2 Proof of Theorem 4.14

The proof of Theorem 4.14 re-uses a lot of what we did for the proof of Theorem 4.13 above. We first look at the scheme when g is a random function, and then use the pseudorandomness of the given family F to deduce the theorem. As before we associate to a family of functions G having domain $\{0,1\}^n$ and range $\{0,1\}^\ell$ a parameterized version of the CTR\$ scheme, $\mathcal{SE}[G] = (\mathcal{K}, \mathcal{E}, \mathcal{D})$. The key generation algorithm simply returns a random instance of G , meaning picks a function $g \xleftarrow{R} G$ from family G at random, and views g as the key, and the encryption and decryption algorithms are shown in Figure 4.2. Here is the main lemma.

Lemma 4.18 Let A be any IND-CPA adversary attacking $\mathcal{SE}[\text{Rand}(n,\ell)]$. Then

$$\mathbf{Adv}_{\mathcal{SE}[\text{Rand}(n,\ell)]}^{\text{ind-cpa}}(A) \leq \frac{\mu(q-1)}{\ell 2^n},$$

where q is the number of oracle queries made by A and μ is the total length of these queries. ■

The proof of Theorem 4.14 given this lemma is easy at this point because it is almost identical to the above proof of Theorem 4.13. So let us finish that first, and then go on to prove Lemma 4.18.

Proof of Theorem 4.14: Let A be any IND-CPA adversary attacking $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$. Assume A makes q oracle queries totaling μ bits, and has time-complexity t . We will design a distinguisher D_A such that

$$\mathbf{Adv}_{\mathcal{SE}}^{\text{ind-cpa}}(A) \leq 2 \cdot \mathbf{Adv}_F^{\text{prf}}(D_A) + \frac{\mu(q-1)}{\ell 2^n}.$$

Furthermore, D_A will make μ/ℓ oracle queries and have time-complexity t . Now, the statement of Theorem 4.14 follows as usual, by taking maximums.

The code for D_A is the same as in the proof of Theorem 4.13. However note that the underlying algorithm $\mathcal{E}_g(\cdot)$ has changed, now being the one of Figure 4.2 rather than that of Figure 4.1. For the analysis, the only change is that the term

$$\mathbf{Adv}_{\mathcal{SE}[\text{Rand}(n,\ell)]}^{\text{ind-cpa}}(A)$$

in Equation (4.5), rather than being zero, is upper bounded as per Lemma 4.18, and thus

$$\mathbf{Adv}_F^{\text{prf}}(D_A) \geq \frac{1}{2} \cdot \mathbf{Adv}_{\mathcal{SE}[F]}^{\text{ind-cpa}}(A) - \frac{1}{2} \cdot \frac{\mu(q-1)}{\ell 2^n}. \quad (4.6)$$

The rest is as before. ■

The above illustrates how general and generic was the “simulation” argument of the proof of Theorem 4.13. Indeed it adapts easily not only to the randomized version of the scheme but also to the use of pseudorandom functions in many other schemes, even for different tasks like message authentication. The key point that makes it work is that the scheme itself invokes g as an oracle.

Before we prove Lemma 4.18, we will analyze a certain probabilistic game. The problem we isolate here is purely probabilistic; it has nothing to do with encryption or even cryptography.

Lemma 4.19 Let n, q, ℓ be positive integers, and let $m_1, \dots, m_q < 2^n$ also be positive integers. Suppose we pick q integers r_1, \dots, r_q from $[0..2^n - 1]$ uniformly and

independently at random. We consider the following $m_1 + \dots + m_q$ numbers:

$$\begin{array}{ccccccc} r_1 + 1, & r_1 + 2, & \dots, & r_1 + m_1 & & & \\ r_2 + 1, & r_2 + 2, & \dots, & r_2 + m_2 & & & \\ \vdots & & & & & & \vdots \\ r_q + 1, & r_q + 2, & \dots, & r_q + m_q, & & & \end{array}$$

where the addition is performed modulo 2^n . We say that a *collision* occurs if some two (or more) numbers in the above table are equal. Then

$$\Pr[\text{Col}] \leq \frac{(q-1)(m_1 + \dots + m_q)}{2^n},$$

where Col denotes the event that a collision occurs. ■

Proof of Lemma 4.19: As with many of the probabilistic settings that arise in this area, this is a question about some kind of “balls thrown in bins” setting, related to the birthday problem studied in Appendix A. Indeed a reader may find it helpful to study that appendix first.

Think of having 2^n bins, numbered $0, 1, \dots, 2^n - 1$. We have q balls, numbered $1, \dots, q$. For each ball we choose a random bin which we call r_i . We choose the bins one by one, so that we first choose r_1 , then r_2 , and so on. When we have thrown in the first ball, we have defined the first row of the above table, namely the values $r_1 + 1, \dots, r_1 + m_1$. Then we pick the assignment r_2 of the bin for the second ball. This defines the second row of the table, namely the values $r_2 + 1, \dots, r_2 + m_2$. A collision occurs if any value in the second row equals some value in the first row. We continue, up to the q -th ball, each time defining a row of the table, and are finally interested in the probability that a collision occurred somewhere in the process. To upper bound this, we want to write this probability in such a way that we can do the analysis step by step, meaning view it in terms of having thrown, and fixed, some number of balls, and seeing whether there is a collision when we throw in one more ball. To this end let Col_i denote the event that there is a collision somewhere in the first i rows of the table, for $i = 1, \dots, q$. Let NoCol_i denote the event that there is no collision in the first i rows of the table, for $i = 1, \dots, q$. Then by conditioning we have

$$\begin{aligned} \Pr[\text{Col}] &= \Pr[\text{Col}_q] \\ &= \Pr[\text{Col}_{q-1}] + \Pr[\text{Col}_q \mid \text{NoCol}_{q-1}] \cdot \Pr[\text{NoCol}_{q-1}] \\ &\leq \Pr[\text{Col}_{q-1}] + \Pr[\text{Col}_q \mid \text{NoCol}_{q-1}] \\ &\leq \vdots \\ &\leq \Pr[\text{Col}_1] + \sum_{i=2}^q \Pr[\text{Col}_i \mid \text{NoCol}_{i-1}] \end{aligned}$$

$$= \sum_{i=2}^q \Pr [\text{Col}_i \mid \text{NoCol}_{i-1}] .$$

Thus we need to upper bound the chance of a collision upon throwing the i -th ball, given that there was no collision created by the first $i - 1$ balls. Then we can sum up the quantities obtained and obtain our bound.

We claim that for any $i = 2, \dots, q$ we have

$$\Pr [\text{Col}_i \mid \text{NoCol}_{i-1}] \leq \frac{(i-1)m_i + m_{i-1} + \dots + m_1}{2^n} . \quad (4.7)$$

Let us first see why this proves the lemma and then return to justify it. From the above and Equation (4.7) we have

$$\begin{aligned} \Pr [\text{Col}] &\leq \sum_{i=2}^q \Pr [\text{Col}_i \mid \text{NoCol}_{i-1}] \\ &\leq \sum_{i=2}^q \frac{(i-1)m_i + m_{i-1} + \dots + m_1}{2^n} \\ &= \frac{(q-1)(m_1 + \dots + m_q)}{2^n} . \end{aligned}$$

How did we do the last sum? The term m_i occurs with weight $i - 1$ in the i -th term of the sum, and then with weight 1 in the j -th term of the sum for $j = i + 1, \dots, q$. So its total weight is $(i - 1) + (q - i) = q - 1$.

It remains to prove Equation (4.7). To get some intuition about it, begin with the cases $i = 1, 2$. When we throw in the first ball, the chance of a collision is zero, since there is no previous row with which to collide, so that is simple. When we throw in the second, what is the chance of a collision? The question is, what is the probability that one of the numbers $r_2 + 1, \dots, r_2 + m_2$ defined by the second ball is equal to one of the numbers $r_1 + 1, \dots, r_1 + m_1$ already in the table? View r_1 as fixed. Observe that a collision occurs if and only if $r_1 - m_2 + 1 \leq r_2 \leq r_1 + m_1 - 1$. So there are $(r_1 + m_1 - 1) - (r_1 - m_2 + 1) + 1 = m_1 + m_2 - 1$ choices of r_2 that could yield a collision. This means that $\Pr [\text{Col}_2 \mid \text{NoCol}_1] \leq (m_2 + m_1 - 1)/2^n$.

We need to extend this argument as we throw in more balls. So now suppose $i - 1$ balls have been thrown in, where $2 \leq i \leq q$, and suppose there is no collision in the first $i - 1$ rows of the table. We throw in the i -th ball, and want to know what is the probability that a collision occurs. We are viewing the first $i - 1$ rows of the table as fixed, so the question is just what is the probability that one of the numbers defined by r_i equals one of the numbers in the first $i - 1$ rows of the table. A little thought shows that the worst case (meaning the case where the probability is the largest) is when the existing $i - 1$ rows are well spread-out. We can upper bound the collision probability by reasoning just as above, except that there are $i - 1$ different intervals

to worry about rather than just one. The i -th row can intersect with the first row, or the second row, or the third, and so on, up to the $(i - 1)$ -th row. So we get

$$\begin{aligned} \Pr[\text{Col}_i \mid \text{NoCol}_{i-1}] &\leq \frac{(m_i + m_1 - 1) + (m_i + m_2 - 1) + \cdots + (m_i + m_{i-1} - 1)}{2^n} \\ &= \frac{(i - 1)m_i + m_{i-1} + \cdots + m_1 - (i - 1)}{2^n}, \end{aligned}$$

and Equation (4.7) follows by just dropping the negative term in the above. ■

Let us now extend the proof of Lemma 4.17 to prove Lemma 4.18.

Proof of Lemma 4.18: Recall that the idea of the proof of Lemma 4.17 was that when g is a random function, its value on successive counter values yields a one-time pad. This holds whenever g is applied on some set of distinct values. In the counter case, the inputs to g are always distinct. In the randomized case they may not be distinct. The approach is to consider the event that they are distinct, and say that in that case the adversary has no advantage; and on the other hand, while it may have a large advantage in the other case, that case does not happen often. We now flush all this out in more detail.

The adversary makes some number q of oracle queries. Let $(M_{i,0}, M_{i,1})$ be the i -th query, and let m_i be the number of blocks in $M_{i,0}$. (This is the same as the number of blocks in $M_{i,1}$.) Let $M_{i,b}[j]$ be the value of the j -th ℓ -bit block of $M_{i,b}$ for $b \in \{0, 1\}$. Let C'_i be the response returned by the oracle to query $(M_{i,0}, M_{i,1})$. It consists of the encoding of a number $r_i \in [0..2^n - 1]$ and a m_i -block message $C_i = C_i[1] \cdots C_i[m_i]$. Pictorially:

$$\begin{aligned} M_{1,b} &= M_{1,b}[1]M_{1,b}[1] \cdots M_{1,b}[m_1] \\ C_1 &= \langle r_1, C_1[1] \cdots C_1[m_1] \rangle \\ \\ M_{2,b} &= M_{2,b}[1]M_{2,b}[2] \cdots M_{2,b}[m_2] \\ C_2 &= \langle r_2, C_2[1] \cdots C_2[m_2] \rangle \\ \\ &\vdots \quad \quad \quad \vdots \\ \\ M_{q,b} &= M_{q,b}[1]M_{q,b}[2] \cdots M_{q,b}[m_q] \\ C_q &= \langle r_q, C_q[1] \cdots C_q[m_q] \rangle \end{aligned}$$

Let **NoCol** be the event that the following $m_1 + \dots + m_q$ values are all distinct:

$$\begin{array}{ccccccc} r_1 + 1, & r_1 + 2, & \dots, & r_1 + m_1 & & & \\ r_2 + 1, & r_2 + 2, & \dots, & r_2 + m_2 & & & \\ \vdots & & & & & & \vdots \\ r_q + 1, & r_q + 2, & \dots, & r_q + m_q & & & \end{array}$$

Let **Col** be the complement of the event **NoCol**, meaning the event that the above table contains at least two values that are the same. It is useful for the analysis to introduce the following shorthand:

$$\Pr_0[\cdot] = \text{The probability of event “.” in world 0}$$

$$\Pr_1[\cdot] = \text{The probability of event “.” in world 1 .}$$

We will use the following three claims, which are proved later. The first claim says that the probability of a collision in the above table does not depend on which world we are in.

Claim 1: $\Pr_1[\text{Col}] = \Pr_0[\text{Col}]$. \square

The second claim says that A has zero advantage in winning the left-or-right game in the case that no collisions occur in the table. Namely, its probability of outputting one is identical in these two worlds under the assumption that no collisions have occurred in the values in the table.

Claim 2: $\Pr_0[A = 1 \mid \text{NoCol}] = \Pr_1[A = 1 \mid \text{NoCol}]$. \square

We can say nothing about the advantage of A if a collision does occur in the table. It might be big. However, it will suffice to know that the probability of a collision is small. Since we already know that this probability is the same in both worlds (Claim 1) we bound it just in world 0:

Claim 3: $\Pr_0[\text{Col}] \leq \frac{\mu(q-1)}{\ell 2^n}$. \square

Let us see how these put together complete the proof of the lemma, and then go back and prove them.

Proof of Lemma given Claims: It is a simple conditioning argument:

$$\begin{aligned} & \mathbf{Adv}_{\mathcal{SE}[\text{Rand}(n,\ell)]}^{\text{ind-cpa}}(A) \\ &= \Pr_1[A = 1] - \Pr_0[A = 1] \\ &= \Pr_1[A = 1 \mid \text{Col}] \cdot \Pr_1[\text{Col}] + \Pr_1[A = 1 \mid \text{NoCol}] \cdot \Pr_1[\text{NoCol}] \\ &\quad - \Pr_0[A = 1 \mid \text{Col}] \cdot \Pr_0[\text{Col}] - \Pr_0[A = 1 \mid \text{NoCol}] \cdot \Pr_0[\text{NoCol}] \end{aligned}$$

Using Claim 1 and Claim 2, the above equals

$$\begin{aligned} &= (\Pr_1[A = 1 \mid \text{Col}] - \Pr_0[A = 1 \mid \text{Col}]) \cdot \Pr_0[\text{Col}] \\ &\leq \Pr_0[\text{Col}] . \end{aligned}$$

In the last step we simply bounded the parenthesized expression by 1. Now apply Claim 3, and we are done. \square

It remains to prove the three claims.

Proof of Claim 1: The event NoCol depends only on the random values r_1, \dots, r_q chosen by the encryption algorithm $\mathcal{E}_g(\cdot)$. These choices, however, are made in exactly the same way in both worlds. The difference in the two worlds is what message is encrypted, not how the random values are chosen. \square

Proof of Claim 2: Given the event NoCol, we have that, in either game, the function g is evaluated at a new point each time it is invoked. (Here we use the assumption that $\mu < \ell 2^n$, since otherwise there may be wraparound in even a single query.) Thus the output is randomly and uniformly distributed over $\{0, 1\}^\ell$, independently of anything else. That means the reasoning from the counter-based scheme as given in Lemma 4.17 applies. Namely we observe that according to the scheme

$$C_i[j] = g(r_i + j) \oplus \begin{cases} M_{i,1}[j] & \text{if we are in world 1} \\ M_{i,0}[j] & \text{if we are in world 0.} \end{cases}$$

Thus each cipher block is a message block XORed with a random value. A consequence of this is that each cipher block has a distribution that is independent of any previous cipher blocks and of the messages. \square

Proof of Claim 3: This follows from Lemma 4.19. We simply note that $m_1 + \dots + m_q = \mu/\ell$. \square

This concludes the proof. \blacksquare

4.8 Security of CBC encryption

Define indistinguishability from random bits, IND\$-CPA, and show that it implies IND-CPA. Then show the security of CBC using the game approach. That is, construct games 1 and 2 where game 1 returns random bits in response to each query and game 2 returns CBC-encrypted text, under a random permutation, and where these two games are identical until some flag bad gets set to true. Bound the probability that this happens.

4.9 Other characterizations of IND-CPA security

To be written—define (1) real-or-random notion; (2) find-then-guess notion; (3) semantic security. Then prove equivalences.

4.10 Indistinguishability under chosen-ciphertext attack

So far we have considered privacy under chosen-plaintext attack. Sometimes we want to consider privacy when the adversary is capable of mounting a stronger type of attack, namely a chosen-ciphertext attack. In this type of attack, an adversary has access to a decryption oracle. It can feed this oracle a ciphertext and get back the corresponding plaintext.

How might such a situation arise? One situation one could imagine is that an adversary at some point gains temporary access to the equipment performing decryption. It can feed the equipment ciphertexts and see what plaintexts emerge. (We assume it cannot directly extract the key from the equipment, however.)

If an adversary has access to a decryption oracle, security at first seems moot, since after all it can decrypt anything it wants. To create a meaningful notion of security, we put a restriction on the use of the decryption oracle. To see what this is, let us look closer at the formalization. As in the case of chosen-plaintext attacks, we consider two worlds:

world 0: The adversary is provided the oracle $\mathcal{E}_K(\text{LR}(\cdot, \cdot, 0))$ as well as the oracle $\mathcal{D}_K(\cdot)$.

world 1: The adversary is provided the oracle $\mathcal{E}_K(\text{LR}(\cdot, \cdot, 1))$ as well as the oracle $\mathcal{D}_K(\cdot)$.

The adversary's goal is the same as in the case of chosen-plaintext attacks: it wants to figure out which world it is in. There is one easy way to do this. Namely, query the lr-encryption oracle on two distinct, equal length messages M_0, M_1 to get back a ciphertext C , and now call the decryption oracle on C . If the message returned by the decryption oracle is M_0 then the adversary is in world 0, and if the message returned by the decryption oracle is M_1 then the adversary is in world 1. The restriction we impose is simply that this call to the decryption oracle is not allowed. More generally, call a query C to the decryption oracle *illegitimate* if C was previously returned by the lr-encryption oracle; otherwise a query is *legitimate*. We insist that only legitimate queries are allowed. In the formalization below, the experiment simply returns 0 if the adversary makes an illegitimate query. (We clarify that a query C is legitimate if C is returned by the lr-encryption oracle *after* C was queried to the decryption oracle.)

This restriction still leaves the adversary with a lot of power. Typically, a successful chosen-ciphertext attack proceeds by taking a ciphertext C returned by the lr-encryption oracle, modifying it into a related ciphertext C' , and querying the decryption oracle with C' . The attacker seeks to create C' in such a way that its decryption tells the attacker what was the message underlying M . We will see this illustrated in Section 4.11 below.

The model we are considering here might seem quite artificial. If an adversary has access to a decryption oracle, how can we prevent it from calling the decryption oracle on certain messages? The restriction might arise due to the adversary's

having access to the decryption equipment for a limited period of time. We imagine that after it has lost access to the decryption equipment, it sees some ciphertexts, and we are capturing the security of these ciphertexts in the face of previous access to the decryption oracle. Further motivation for the model will emerge when we see how encryption schemes are used in protocols. We will see that when an encryption scheme is used in many authenticated key-exchange protocols the adversary effectively has the ability to mount chosen-ciphertext attacks of the type we are discussing. For now let us just provide the definition and exercise it.

Definition 4.20 Let $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a symmetric encryption scheme, let $b \in \{0, 1\}$, and let A be an algorithm that has access to two oracles and returns a bit. We consider the following experiment:

Experiment $\mathbf{Exmt}_{\mathcal{SE}}^{\text{ind-cca-}b}(A)$
 $K \xleftarrow{R} \mathcal{K}$
 $b \leftarrow A^{\mathcal{E}_K(\text{LR}(\cdot, \cdot, b)), \mathcal{D}_K(\cdot)}$
 If A queried $\mathcal{D}_K(\cdot)$ on a ciphertext previously returned by $\mathcal{E}_K(\text{LR}(\cdot, \cdot, b))$
 then return 0
 else return b

The *ind-cca advantage* of A is defined as

$$\mathbf{Adv}_{\mathcal{SE}}^{\text{ind-cca}}(A) = \Pr[\mathbf{Exmt}_{\mathcal{SE}}^{\text{ind-cca-1}}(A) = 1] - \Pr[\mathbf{Exmt}_{\mathcal{SE}}^{\text{ind-cca-0}}(A) = 1].$$

For any $t, q_e, \mu_e, q_d, \mu_d$ we define the *ind-cca advantage* of \mathcal{SE} via

$$\mathbf{Adv}_{\mathcal{SE}}^{\text{ind-cca}}(t, q_e, \mu_e, q_d, \mu_d) = \max_A \{\mathbf{Adv}_{\mathcal{SE}}^{\text{ind-cca}}(A)\}$$

where the maximum is over all A having time-complexity t , making to the lr-encryption oracle at most q_e queries the sum of whose lengths is at most μ_e bits, and making to the decryption oracle at most q_d queries the sum of whose lengths is at most μ_d bits. ■

The conventions with regard to resource measures are the same as those used in the case of chosen-plaintext attacks. In particular, the length of a query M_0, M_1 to the lr-encryption oracle is the length is defined as the length of M_0 , and the time-complexity is the execution time of the entire experiment plus the size of the code of the adversary.

We consider an encryption scheme to be “secure against chosen-ciphertext attack” if a “reasonable” adversary cannot obtain “significant” advantage in distinguishing the cases $b = 0$ and $b = 1$ given access to the oracles, where reasonable reflects its resource usage. The technical notion is called indistinguishability under chosen-ciphertext attack, denoted IND-CCA.

4.11 Example chosen-ciphertext attacks

Chosen-ciphertext attacks are powerful enough to break all the standard modes of operation, even those like CTR and CBC that are secure against chosen-plaintext attack. The one-time pad scheme is also vulnerable to a chosen-ciphertext attack: our notion of perfect security only took into account chosen-plaintext attacks. Let us now illustrate a few chosen-ciphertext attacks.

4.11.1 Attack on CTR\$

Let $F: \mathcal{K} \times \{0, 1\}^n \rightarrow \{0, 1\}^\ell$ be a family of functions and let $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be the associated CTR\$ symmetric encryption scheme as described in Scheme 4.5. The weakness of the scheme that makes it susceptible to a chosen-ciphertext attack is the following. Say $\langle r, C[1] \rangle$ is a ciphertext of some ℓ -bit message M , and we flip bit i of $C[1]$, resulting in a new ciphertext $\langle r, C'[1] \rangle$. Let M' be the message obtained by decrypting the new ciphertext. Then M' equals M with the i -th bit flipped. (You should check that you understand why.) Thus, by making a decryption oracle query of $\langle r, C'[1] \rangle$ one can learn M' and thus M . In the following, we show how this idea can be applied to break the scheme in our model by figuring out in which world an adversary has been placed.

Proposition 4.21 Let $F: \mathcal{K} \times \{0, 1\}^n \rightarrow \{0, 1\}^\ell$ be a family of functions and let $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be the corresponding CTR\$ symmetric encryption scheme as described in Scheme 4.5. Then

$$\mathbf{Adv}_{\mathcal{SE}}^{\text{ind-cca}}(t, 1, \ell, 1, n + \ell) = 1$$

for $t = O(n + \ell)$ plus the time for one application of F . ■

The advantage of this adversary is 1 even though it uses hardly any resources: just one query to each oracle. That is clearly an indication that the scheme is insecure.

Proof of Proposition 4.21: We will present an adversary algorithm A , having time-complexity t , making 1 query to its lr-encryption oracle, this query being of length ℓ , making 1 query to its decryption oracle, this query being of length $n + \ell$, and having

$$\mathbf{Adv}_{\mathcal{SE}}^{\text{ind-cca}}(A) = 1.$$

The Proposition follows.

Remember the the lr-encryption oracle $\mathcal{E}_K(\text{LR}(\cdot, \cdot, b))$ takes input a pair of messages, and returns an encryption of either the left or the right message in the pair, depending on the value of b . The goal of A is to determine the value of b . Our adversary works like this:

Adversary $A^{en(\cdot), de(\cdot)}$

$$M_0 \leftarrow 0^\ell; M_1 \leftarrow 1^\ell$$

$$\langle r, C[1] \rangle \leftarrow en(M_0, M_1)$$

$$C'[1] \leftarrow C[1] \oplus 1^\ell$$

$$C' \leftarrow \langle r, C'[1] \rangle$$

$$M \leftarrow de(C')$$

If $M = M_0$ then return 1 else return 0

The adversary's single lr-encryption oracle query is the pair of distinct messages M_0, M_1 , each one block long. It is returned a ciphertext $\langle r, C[1] \rangle$. It flips the bits of $C[1]$ to get $C'[1]$ and then feeds the ciphertext $\langle r, C'[1] \rangle$ to the decryption oracle. It bets on world 1 if it gets back M_0 , and otherwise on world 0. Notice that $\langle r, C'[1] \rangle \neq \langle r, C[1] \rangle$, so the decryption query is legitimate. Now, we claim that

$$\Pr \left[\mathbf{Exmt}_{\mathcal{SE}}^{\text{ind-cca-1}}(A) = 1 \right] = 1$$

$$\Pr \left[\mathbf{Exmt}_{\mathcal{SE}}^{\text{ind-cca-0}}(A) = 1 \right] = 0.$$

Hence $\mathbf{Adv}_{\mathcal{SE}}^{\text{ind-cpa}}(A) = 1 - 0 = 1$. And A achieved this advantage by making just one lr-encryption oracle query, whose length, which as per our conventions is just the length of M_0 , is ℓ bits, and just one decryption oracle query, whose length is $n + \ell$ bits (assuming an encoding of $\langle r, X \rangle$ as $n + |X|$ -bits). So $\mathbf{Adv}_{\mathcal{SE}}^{\text{pr-cpa}}(t, 1, \ell, 1, n + \ell) = 1$.

Why are the two equations claimed above true? You have to return to the definitions of the quantities in question, as well as the description of the scheme itself, and walk it through. In world 1, meaning $b = 1$, let $\langle r, C[1] \rangle$ denote the ciphertext returned by the lr-encryption oracle. Then

$$C[1] = F_K(r + 1) \oplus M_1 = F_K(r + 1) \oplus 1^\ell.$$

Now notice that

$$\begin{aligned} M &= \mathcal{D}_K(C[0]C'[1]) \\ &= F_K(r + 1) \oplus C'[1] \\ &= F_K(r + 1) \oplus C[1] \oplus 1^\ell \\ &= F_K(r + 1) \oplus (F_K(r + 1) \oplus 1^\ell) \oplus 1^\ell \\ &= 0^\ell \\ &= M_0. \end{aligned}$$

Thus, the decryption oracle will return M_0 , and thus A will return 1. In world 0, meaning $b = 0$, let $\langle r, C[1] \rangle$ denote the ciphertext returned by the lr-encryption oracle. Then

$$C[1] = F_K(r + 1) \oplus M_0 = F_K(r + 1) \oplus 0^\ell.$$

Now notice that

$$\begin{aligned}
 M &= \mathcal{D}_K(\langle r, C'[1] \rangle) \\
 &= F_K(r+1) \oplus C'[1] \\
 &= F_K(r+1) \oplus C[1] \oplus 1^\ell \\
 &= F_K(r+1) \oplus (F_K(r+1) \oplus 0^\ell) \oplus 1^\ell \\
 &= 1^\ell \\
 &= M_1 .
 \end{aligned}$$

Thus, the decryption oracle will return M_1 , and thus A will return 0, meaning will return 1 with probability zero. ■

An attack on CTRC (cf. Scheme 4.6) is similar, and is left to the reader.

4.11.2 Attack on CBC

Let $E: \mathcal{K} \times \{0,1\}^n \rightarrow \{0,1\}^n$ be a block cipher and let $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be the associated CBC symmetric encryption scheme as described in Scheme 4.4. The weakness of the scheme that makes it susceptible to a chosen-ciphertext attack is the following. Say $\langle IV, C[1] \rangle$ is a ciphertext of some n -bit message M , and we flip bit i of the IV, resulting in a new ciphertext $\langle IV', C[1] \rangle$. Let M' be the message obtained by decrypting the new ciphertext. Then M' equals M with the i -th bit flipped. (You should check that you understand why by looking at Scheme 4.4.) Thus, by making a decryption oracle query of $\langle IV', C[1] \rangle$ one can learn M' and thus M . In the following, we show how this idea can be applied to break the scheme in our model by figuring out in which world an adversary has been placed.

Proposition 4.22 Let $E: \mathcal{K} \times \{0,1\}^n \rightarrow \{0,1\}^n$ be a block cipher and let $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be the corresponding CBC\$ encryption scheme as described in Scheme 4.4. Then

$$\mathbf{Adv}_{\mathcal{SE}}^{\text{ind-cca}}(t, 1, n, 1, 2n) = 1$$

for $t = O(n)$ plus the time for one application of F . ■

The advantage of this adversary is 1 even though it uses hardly any resources: just one query to each oracle. That is clearly an indication that the scheme is insecure.

Proof of Proposition 4.22: We will present an adversary A , having time-complexity t , making 1 query to its lr-encryption oracle, this query being of length n , making 1 query to its decryption oracle, this query being of length $2n$, and having

$$\mathbf{Adv}_{\mathcal{SE}}^{\text{ind-cca}}(A) = 1 .$$

The Proposition follows.

Remember the the lr-encryption oracle $\mathcal{E}_K(\text{LR}(\cdot, \cdot, b))$ takes input a pair of messages, and returns an encryption of either the left or the right message in the pair, depending on the value of b . The goal of A is to determine the value of b . Our adversary works like this:

Adversary $A^{en(\cdot, \cdot), de(\cdot)}$
 $M_0 \leftarrow 0^l; M_1 \leftarrow 1^n$
 $\langle \text{IV}, C[1] \rangle \leftarrow en(M_0, M_1)$
 $\text{IV}' \leftarrow \text{IV} \oplus 1^n; C' \leftarrow \langle \text{IV}', C[1] \rangle$
 $M \leftarrow de(C')$
 If $M = M_0$ then return 1 else return 0

The adversary's single lr-encryption oracle query is the pair of distinct messages M_0, M_1 , each one block long. It is returned a ciphertext $\langle \text{IV}, C[1] \rangle$. It flips the bits of the IV to get a new IV, IV' and then feeds the ciphertext $\langle \text{IV}', C[1] \rangle$ to the decryption oracle. It bets on world 1 if it gets back M_0 , and otherwise on world 0. It is important that $\langle \text{IV}', C[1] \rangle \neq \langle \text{IV}, C[1] \rangle$ so the decryption oracle query is legitimate. Now, we claim that

$$\begin{aligned} \Pr \left[\mathbf{Exmt}_{\mathcal{SE}}^{\text{ind-cca-1}}(A) = 1 \right] &= 1 \\ \Pr \left[\mathbf{Exmt}_{\mathcal{SE}}^{\text{ind-cca-0}}(A) = 1 \right] &= 0. \end{aligned}$$

Hence $\mathbf{Adv}_{\mathcal{SE}}^{\text{ind-cca}}(A) = 1 - 0 = 1$. And A achieved this advantage by making just one lr-encryption oracle query, whose length, which as per our conventions is just the length of M_0 , is n bits, and just one decryption oracle query, whose length is $2n$ bits. So $\mathbf{Adv}_{\mathcal{SE}}^{\text{ind-cca}}(t, 1, n, 1, 2n) = 1$.

Why are the two equations claimed above true? You have to return to the definitions of the quantities in question, as well as the description of the scheme itself, and walk it through. In world 1, meaning $b = 1$, the lr-encryption oracle returns $\langle \text{IV}, C[1] \rangle$ with

$$C[1] = E_K(\text{IV} \oplus M_1) = E_K(\text{IV} \oplus 1^n).$$

Now notice that

$$\begin{aligned} M &= \mathcal{D}_K(\langle \text{IV}', C[1] \rangle) \\ &= E_K^{-1}(C[1]) \oplus \text{IV}' \\ &= E_K^{-1}(E_K(\text{IV} \oplus 1^n)) \oplus \text{IV}' \\ &= (\text{IV} \oplus 1^n) \oplus \text{IV}'[0] \\ &= (\text{IV} \oplus 1^n) \oplus (\text{IV} \oplus 1^n) \\ &= 0^n \\ &= M_0. \end{aligned}$$

Thus, the decryption oracle will return M_0 , and thus A will return 1. In world 0, meaning $b = 0$, the Ir-encryption oracle returns $\langle \text{IV}, C[1] \rangle$ with

$$C[1] = E_K(\text{IV} \oplus M_0) = E_K(\text{IV} \oplus 0^l).$$

Now notice that

$$\begin{aligned} M &= \mathcal{D}_K(C'[0]C[1]) \\ &= E_K^{-1}(C[1]) \oplus \text{IV}' \\ &= E_K^{-1}(E_K(\text{IV} \oplus 0^n)) \oplus \text{IV}' \\ &= (\text{IV} \oplus 0^n) \oplus \text{IV}'[0] \\ &= (\text{IV} \oplus 0^n) \oplus (\text{IV} \oplus 1^n) \\ &= 1^n \\ &= M_1. \end{aligned}$$

Thus, the decryption oracle will return M_1 , and thus A will return 0, meaning will return 1 with probability zero. ■

4.12 Historical Notes

The pioneering work on the theory of encryption is that of Goldwasser and Micali [17], with refinements by [25, 12]. This body of work is however in the asymmetric (ie. public key) setting, and uses the asymptotic framework of polynomial-time adversaries and negligible success probabilities. The treatment of symmetric encryption we are using is from [3]. In particular Definition 4.1 and the concrete security framework are from [3]. The analysis of the CTR mode encryption schemes, as given in Theorems 4.13 and 4.14, is also from [3]. The approach taken to the analysis of CBC mode is new.

4.13 Exercises and Problems

Exercise 4.1 Revise the definition of CTRC mode so as to not make the assumption that plaintexts are a positive multiple of ℓ bits.

Problem 4.1 Formalize a notion of security against key-recovery for symmetric encryption schemes, and prove an analogue of Proposition 4.12.

Problem 4.2 Let $l \geq 1$ and $m \geq 2$ be integers, and let $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a given symmetric encryption scheme whose associated plaintext space is $\{0, 1\}^n$, meaning one can only encrypt messages of length n . In order to be able to encrypt longer messages, says ones of mn bits for $m \geq 1$, we define a new symmetric encryption scheme $\mathcal{SE}^{(m)} = (\mathcal{K}, \mathcal{E}^{(m)}, \mathcal{D}^{(m)})$ having the same key generation algorithm as that of \mathcal{SE} , plaintext space $\{0, 1\}^{mn}$, and encryption and decryption algorithms as follows:

<pre> Algorithm $\mathcal{E}_K^{(m)}(M)$ Parse M as n-bit $\langle M[1], \dots, M[m] \rangle$ For $i \leftarrow 1$ to m do $C[i] \leftarrow \mathcal{E}_K(M[i])$ EndFor $C \leftarrow C[1] \cdots C[m]$ Return C </pre>	<pre> Algorithm $\mathcal{D}_K^{(m)}(C)$ Parse C as n-bit $C[1] \cdots C[m]$ For $i \leftarrow 1$ to m do $M[i] \leftarrow \mathcal{D}_K(C[i])$ If $M[i] = \perp$ then return \perp EndFor $M \leftarrow M[1] \cdots M[m]$ Return M </pre>
--	---

Here M is mn bits long. For encryption, M is broken into a sequence of blocks $M = M[1] \dots M[m]$, each block being n -bits long, and each block is then separately encrypted. For decryption, C is parsed as a sequence of m strings, each n bits, and each is separately decrypted. If any component ciphertexts $C[i]$ is invalid (meaning \mathcal{D}_K returns \perp for it) then the entire ciphertext is declared invalid.

(a) Show that

$$\mathbf{Adv}_{\mathcal{SE}^{(m)}}^{\text{ind-cca}}(t, 1, mn, 1, mn) = 1$$

for some small t .

(b) Show that

$$\mathbf{Adv}_{\mathcal{SE}^{(m)}}^{\text{ind-cpa}}(t, q, mnq) \leq \mathbf{Adv}_{\mathcal{SE}}^{\text{ind-cpa}}(t, mq, mnq)$$

for any t, q .

Part (a) says that $\mathcal{SE}^{(m)}$ is insecure against chosen-ciphertext attack. Note this is true regardless of the security properties of \mathcal{SE} , which may itself be secure against chosen-ciphertext attack. Part (b) says that if \mathcal{SE} is secure against chosen-plaintext attack, then so is $\mathcal{SE}^{(m)}$.

Problem 4.3 Consider the problem of trying to define the strongest notion of encryption that is achievable by a stateless, deterministic scheme. Syntactically, an encryption scheme is a triple of algorithms $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$, as before, but now \mathcal{E} is a deterministic function taking a key K and a message M and producing a ciphertext C .

- (a) Define an appropriate measure of advantage $\mathbf{Adv}_{\mathcal{SE}}^{\text{cipher}}(A)$.
- (b) Describe some contexts in which a deterministic scheme is and is not appropriate.

Chapter 5

HASH FUNCTIONS

5.1 Notions of security for hash-function families

Under a unified framework, define: universal hash functions, UOWHFs, collision-resistant hash functions.

5.2 The hash function SHA-1

5.3 The Merkle-Damgård result

5.4 Collision-resistant hash functions are one-way

5.5 UOWHFs

Give the BR and the Shoup constructions

5.6 Universal hash functions

5.7 Exercises and Problems

Exercise 5.1 Let $H : \mathcal{K} \times \{0, 1\}^a \rightarrow \{0, 1\}^n$ be an ϵ -AU hash-function family. Construct from H an ϵ -AU hash-function family $H' : \mathcal{K} \times \{0, 1\}^{2a} \rightarrow \{0, 1\}^{2n}$.

Exercise 5.2 Let $H : \mathcal{K} \times \{0, 1\}^a \rightarrow \{0, 1\}^n$ be an ϵ -AU hash-function family. Construct from H an ϵ^2 -AU hash-function family $H' : \mathcal{K}^2 \times \{0, 1\}^a \rightarrow \{0, 1\}^{2n}$.

Chapter 6

MESSAGE AUTHENTICATION

In this chapter we address message authentication, the second major goal in cryptography. In most people’s minds, privacy is the goal most strongly associated to cryptography. But message authentication is arguably even more important. Indeed you may may or may not care if some particular message you send out stays private, but you almost certainly do want to be sure of the originator of each message that you act on. Message authentication is what buys you that guarantee.

Message authentication allows one party—the Sender—to send a message to another party—the Receiver—in such a way that if the message is modified en route, then the Receiver will almost certainly detect this. Message authentication is also called “data-origin authentication,” since it authenticates the point-of-origin for each message. Message authentication is said to protect the “integrity” of messages, ensuring that each that is received and deemed acceptable is arriving in the same condition that it was sent out—with no bits inserted, missing, or modified.

Here we’ll be looking at the shared-key setting for message authentication (remember that the public-key setting is the problem addressed by *digital signature*). In this case the Sender and the Receiver share a secret key, K , which they’ll use to authenticate their transmissions. We’ll define the message authentication goal and we’ll describe some different ways to achieve it. As usual, we’ll be careful to pin down the problem we’re working to solve.

6.1 The Setting

It is often crucial for an agent who receives a message to be sure who sent it out. If a hacker can call into his bank’s central computer and produce deposit transactions that *appear* to be coming from a branch office, easy wealth is just around the corner. If an unprivileged user can interact over the network with his company’s mainframe in such a way that the machine *thinks* that the packets it is receiving are coming from the system administrator, then all the machine’s access control mechanisms

Figure 6.1: A message-authentication scheme. Sender S wants to send a message M to receiver R in such a way that R will be sure that M came from S . They share key K . Adversary A controls the communication channel. Sender S sends an authenticated version of M , M' , which adversary A may or may not pass on. On receipt of a message \overline{M} , receiver R either recovers a message that S really sent, or else R gets an indication that \overline{M} is inauthentic.

are for naught. If an Internet interlouter can provide bogus financial data to on-line investors, he can make data *seem* to have come from a reputable source when it does not, perhaps inducing an enemy to make a disastrous investment.

In all of these cases the risk is that an adversary A —the Forger—will create messages that look like they come from some other party, S , the (legitimate) Sender. The attacker will send a message M to R —the Receiver—under S 's identity. The Receiver R will be tricked into believing that M originates with S . Because of this wrong belief, R may act on M in a way that is somehow inappropriate.

The rightful Sender S could be one of many different kinds of entities, like a person, a corporation, a network address, or a particular process running on a particular machine. As the receiver R , you might know that it is S that supposedly sent you the message M for a variety of reasons. For example, the message M might be tagged by an identifier which somehow names S . Or it might be that the manner in which M arrives is a route currently dedicated to servicing traffic from S .

Here we're going to be looking at the case when S and R already share some secret key, K . How S and R came to get this shared secret key is a separate question, one that we deal with it in Chapter ??.

Authenticating messages may be something done for the benefit of the Receiver R , but the Sender S will certainly need to help out—he'll have to *authenticate* each of his messages. See Figure 6.1. To authenticate a message M using the key K the legitimate Sender will apply some “message-authenticating function” \mathcal{S} to K and M , giving rise an “authenticated message” M' . The sender S will transmit the authenticated message M' to the receiver R . Maybe the Receiver will get R —and then again, maybe not. The problem is that an adversary A controls the channel on which messages are being sent. Let's let \overline{M} be the message that the Receiver actually gets. The receiver R , on receipt of \overline{M} , will apply some “message-recovery function” to K and \overline{M} . We want that this should yield one of two things: (1) the original message M , or else (2) an indication that \overline{M} should not be regarded as

Figure 6.2: A message authentication code (MAC). A MAC is a special-case of a message-authentication scheme, where the authenticated message is the original message M together with a tag Tag . The adversary controls the channel, so we can not be sure that M and Tag reach their intended destination. Instead, the Receiver gets $\overline{M}, \overline{T}$. The Receiver will apply a verification function to K, \overline{M} and \overline{T} to decide if \overline{M} should be regarded as the transmitted message, M , or as the adversary's creation.

authentic.

Usually the authenticated message M' is just the original message M together with a fixed-length “tag.” The tag serves to validate the authenticity of the message M . In this case we call the message-authentication scheme a *message authentication code*, or *MAC*. See Figure 6.2

When the Receiver decides that a message he has received is inauthentic what should he do? The Receiver might want to just ignore the bogus message: perhaps it was just noise on the channel. Or perhaps taking action will do more harm than good, opening up new possibilities for denial-of-service attacks. Or the Receiver want to take more decisive actions, like tearing down the channel on which the message was received and informing some human being of apparent mischief. The proper course of action is dictated by the circumstances and the security policy of the Receiver.

Unlike encryption, adversarial success in violating the authenticity of messages demands an active attack: to succeed, the adversary has to get some bogus data to the receiver R . If the attacker just watches S and R communicate she hasn't won this game.

In some communication scenerios it may be difficult for the adversary to get her own messages to the receiver R —she might not *really* control the communication channel. For example, it may be difficult for an adversary to drop her own messages onto a dedicated phone line or network link. In other environments it may be trivial, no harder than dropping a packet onto the Internet. Since we don't know what are the characteristics of the Sender—Receiver channel it is best to assume the worst and think that the adversary has plenty of power over the communications media (and even some power over influencing what messages are legitimately sent out).

We wish to emphasize that the authentication problem is very different from

the encryption problem. We are not worried about secrecy of the message M . Our concern is in whether the adversary can profit by injecting new messages into the communications stream, not whether she understands the contents of the communication. Indeed, as we shall see, encryption provides no ready solution for message authentication.

6.2 Encryption does not provide authenticity

We know how to encrypt data so as to provide privacy, and something often suggested—and even done—is to encrypt as a way to provide data authenticity, too. Fix a symmetric encryption scheme $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$, and let parties S and R share a key K for this scheme. When S wants to send a message M to R , she encrypts it, transferring a ciphertext $M' = C$ generated via $C \stackrel{R}{\leftarrow} \mathcal{E}_K(M)$. The receiver B decrypts it and, if it makes sense, he regards the recovered message $M = \mathcal{D}_K(C)$ as authentic.

The argument that this works is as follows. Suppose, for example, that S transmits an ASCII message M_{100} which indicates that R should please transfer \$100 from the checking account of S to the checking account of some other party, A . The adversary A wants to change the amount from the \$100 to \$900. Now if M_{100} had been sent in the clear, A can easily modify it. But if M_{100} is encrypted so that ciphertext C_{100} is sent, how is A to modify C_{100} so as to make S recover the different message M_{900} ? The adversary A does not know the key K , so she cannot just encrypt M_{900} on her own. The privacy of C_{100} already rules out that C_{100} can be profitably tampered with.

The above argument is completely wrong. To see the flaws let's first look at a counter-example. If we encrypt M_{100} using a one time pad, then all the adversary has to do is to XOR the byte of the ciphertext C_{100} which encodes the character "1" with the XOR of the bytes which encode "1" and "9". That is, when we one-time pad encrypt, the privacy of the transmission does *not* make it difficult for the adversary to tamper with ciphertext so as to produce related ciphertexts.

There are many possible reactions to this counter-example. Let's look at some.

What you should *not* conclude is that one-time pad encryption is unsound. The goal of encryption was to provide privacy, and nothing we have said has suggested that one-time pad encryption does not. Faulting an encryption scheme for not providing authenticity is like faulting a screwdriver because you can not use it to cut vegetables. There is no reason to expect a tool designed to solve one problem to be effective at solving another.

You should *not* conclude that the example is contrived, and that you'd fare far better with any other encryption method. One-time-pad encryption is not at all contrived. And other methods of encryption, like CBC encryption, are only marginally better at protecting message integrity. This will be explored in the exercises.

You should *not* conclude that the failure stemmed from a failure to add "redundancy" before the message was encrypted. Adding redundancy is something

like this: before the Sender S encrypts his data he pads it with some known, fixed string, like 128 bits of zeros. When the receiver decrypts the ciphertext he checks whether the decrypted string ends in 128 zeros. He rejects the transmission if it does not. Such an approach can, and almost always will, fail. For example, the added redundancy does absolutely nothing in our one-time pad example.

What you *should* conclude is that encrypting a message was never an appropriate approach for protecting its authenticity. With hindsight, this is pretty clear. The fact that data is encrypted need not prevent an adversary from being able to make the receiver recover data different from that which the sender had intended. Indeed with most encryption schemes *any* ciphertext will decrypt to *something*, so even a random transmission will cause the receiver to receive something different from what the Sender intended, which was not to send any message at all. Now perhaps the random ciphertext will look like garbage to the receiver, or perhaps not. Since we do not know what the the Receiver intends to do with his data it is impossible to say.

Since encryption was not designed for authenticating messages, it very rarely does. We emphasize this because the belief that good encryption, perhaps after adding redundancy, already provides authenticity, is not only voiced, but even printed in books or embedded into security systems. These authors or programmers do not understand how the cryptographic community has separated out our various goals, with encryption being the tool for achieving privacy, and for achieving that goal alone. Happily, we have other tools for achieving message authenticity.

Good cryptographic design is goal-oriented. One must understand and formalize our goal. Only then do we have the basis on which to design and evaluate potential solutions. Accordingly, our next step is to come up with a definition for a message-authentication scheme and its security.

6.3 Syntax of message-authentication schemes

A message authentication scheme \mathcal{MA} has three components. The first is the function \mathcal{K} which generates the shared keys for the Sender and the Receiver. The second is the function \mathcal{S} that takes a key K and a message M and produces an authenticated message M' . The third is the function \mathcal{R} that takes a key K and what is supposed to be an authenticated message M' , and returns either an underlying message M or else an indication that the message M' should not be rejected.

There are some details to take care of. We pin these down in the following definition.

Definition 6.1 A *message-authentication scheme* \mathcal{MA} consists of three functions, $\mathcal{MA} = (\mathcal{K}, \mathcal{S}, \mathcal{R})$, and associated sets $\text{Key}(\mathcal{MA})$, $\text{Msgs}(\mathcal{MA}) \subseteq \{0, 1\}^*$, and $\text{Auth}(\mathcal{MA}) \subseteq \{0, 1\}^*$, as follows:

- The *key-generation function* \mathcal{K} is a probabilistic function which takes no inputs and returns a value $K \in \text{Key}(\mathcal{MA})$, called a *key*. We write $K \stackrel{R}{\leftarrow} \mathcal{K}$ to

denote the choosing of K by computing \mathcal{K} .

- The **message-authenticating function** \mathcal{S} is a deterministic, probabilistic, or stateful function which takes a key $K \in \text{Key}(\mathcal{MA})$ and a string $M \in \{0, 1\}^*$ and returns a value M' , where $M' \in \text{Auth}(\mathcal{MA})$ if $M \in \text{Msgs}(\mathcal{MA})$ and $M' = \text{ERROR}$ if $M \notin \text{Msgs}(\mathcal{MA})$. We write $M' \stackrel{R}{\leftarrow} \mathcal{S}_K(M)$ to denote the choosing of M' by computing \mathcal{S} on K and M .
- The **message-recovery function** \mathcal{R} is a deterministic function which takes a key $K \in \text{Key}(\mathcal{MA})$ and a string $M' \in \{0, 1\}^*$ and returns a value M , where $M \in \text{Msgs}(\mathcal{MA})$ or $M = \text{REJECT}$. We write $M \leftarrow \mathcal{R}_K(M')$ to denote the choosing of M by computing \mathcal{S} on K and M' .

We require that if $K \in \text{Key}(\mathcal{MA})$ and $M' \stackrel{R}{\leftarrow} \mathcal{R}_K(M)$ and $M' \neq \text{ERROR}$, then $\mathcal{S}_K(M') = M$. ■ ■

As we indicated already, a message-authentication code (MAC) is the special case of a message-authentication scheme in which the authenticated message M' consists of M together with a fixed-length string, Tag . Usually the length of the tag is between 32 and 128 bits. MACs of 32 bits, 64 bits, 96 bits, and 128 bits are common.

It could be confusing, but it is very common practice to call the tag itself a MAC. That is, the scheme itself is called MAC, but so too is the computed tag. It's not really a problem, you'll be able to keep them straight.

Since MACs are so important let us take the time to specialize Definition 6.1 for MACs.

Definition 6.2 [MAC] A **message-authentication code** Π consists of three functions, $\Pi = (\mathcal{K}, \text{MAC}, \text{VF})$, and associated sets $\text{Key}(\Pi)$, $\text{Msgs}(\Pi) \subseteq \{0, 1\}^*$, and $\text{Tags}(\Pi) = \{0, 1\}^{\tau(\Pi)}$, for some number $\tau(\Pi) \geq 1$, as follows:

- The **key-generation function** \mathcal{K} is a probabilistic function which takes no inputs and returns a value $K \in \text{Key}(\Pi)$, called a *key*. We write $K \stackrel{R}{\leftarrow} \mathcal{K}$ to denote the choosing of K by computing \mathcal{K} .
- The **MAC-generation function** MAC is a deterministic, probabilistic, or stateful function which takes a key $K \in \text{Key}(\Pi)$ and a string $M \in \{0, 1\}^*$ and returns a value Tag , called the “MAC” or “tag,” where $\text{Tag} \in \text{Tags}(\Pi)$ if $M \in \text{Msgs}(\Pi)$ and $\text{Tag} = \text{ERROR}$ otherwise. We write $\text{Tag} \stackrel{R}{\leftarrow} \text{MAC}_K(M)$ to denote the choosing of Tag by computing MAC on K and M .
- The **MAC-verification function** VF is a deterministic function which takes a key $K \in \text{Key}(\Pi)$, a string $M \in \{0, 1\}^*$, and a string $\text{Tag} \in \{0, 1\}^*$, and returns a value of ACCEPT or REJECT . We write $b \leftarrow \text{VF}_K(M, \text{Tag})$ to denote the choosing of b by computing VF on K , M , and Tag .

We require that if $K \in \text{Key}(\Pi)$ and $M \in \text{Msgs}$ and $\text{Tag} \stackrel{R}{\leftarrow} \text{MAC}_K(M)$ then $\text{VF}_K(M, \text{Tag}) = \text{ACCEPT}$. If $M \notin \text{Msgs}$ or $\text{Tag} \notin \text{Tags}$ then $\text{VF}_K(M, \text{Tag}) = \text{REJECT}$. ■ ■

Let us pause and make a few comments about Definitions 6.1 and 6.2. First, we emphasize that, so far, we have only defined MAC and message-authentication scheme “syntax”—we haven’t yet said anything formal about security. Of course any viable message-authentication scheme will require some security properties. We’ll get there in a moment. But first we needed to pin down exactly what exactly is the type of objects we’re talking about.

Next the reader should notice that we said that, in both definitions, Key was a set—we didn’t say it was a set of strings. The added generality lets us say things like “the key is a pair of functions, h and f ,” or “the key contains an infinite sequence of numbers, each between 0 and $2^{32} - 1$.” Now since K might not be a string we can’t very well speak of $(\mathcal{K}, \mathcal{S}, \mathcal{R})$, or $(\mathcal{K}, \text{MAC}, \text{VF})$, as algorithms: algorithms map strings to strings. So we called these things functions. Recall that we made the same definitional choice when we defined encryption schemes, and for the same reason—it’s just too convenient to sometimes speak of message-authentication schemes which depend on infinite objects. Now in any “practical” message-authentication scheme the set of keys $\text{Key}(\mathcal{MA})$ will need to be a finite set, and the functions \mathcal{K} , MAC and VF will all be given by algorithms—hopefully, quite efficient ones! But allowing greater generality is a useful intermediate step. Not to worry, we’ll have only strings and algorithms before we’re through.

Note that our definitions we didn’t permit stateful message recovery or MAC-verification. Stateful functions for the Receiver can be problematic because of the possibility of messages not reaching their destination—it is too easy for the Receiver to be in a state different from the one that we’d like. All the same, stateful MAC verification functions are essential for detecting “replay attacks,” and are therefore important tools. We will eventually allow stateful verification. We take up this issue in Section ??.

When we defined encryption, it was essential for security that the encryption function be probabilistic or stateful—you couldn’t do well at achieving our strong notion of security with a deterministic encryption function. But this isn’t true for message authentication: it is no problem for the message-authenticating function (or MAC-generation function) to be deterministic. In fact, most MACs do use deterministic MAC-generation functions. In this case, MAC verification is invariably accomplished by having the Verifier compute the correct tag for the received message M (using the MAC-generation function) and checking that it matches the received tag. That is, the MAC-verification function is simply the following:

Function $\text{VF}_K(M, \text{Tag})$
 $\text{Tag}' \leftarrow \text{MAC}_K(M)$
 If $\text{Tag} = \text{Tag}'$ then return ACCEPT else return REJECT.

For a deterministic MAC we’ll only specify the key-generation function and the MAC-generation function: the MAC-verification function is then understood to be the one just described. That is, a deterministic MAC is specified with a pair of functions, $\Pi = (\mathcal{K}, \text{MAC})$, and not a triple of functions, $\Pi = (\mathcal{K}, \text{MAC}, \text{VF})$.

Figure 6.3: *The CBC MAC, here illustrated with a message M of four blocks, $M = M_1M_2M_3M_4$.*

6.4 Example message-authentication schemes

Before getting to the business of defining security for message-authentication schemes let us look at some important examples of message authentication schemes. We will be concrete, to give you a feel for the type of message-authentication codes which people use.

Example 6.3 [CBC MAC] Let $E: \{0, 1\}^k \times \{0, 1\}^n \rightarrow \{0, 1\}^n$ be a block cipher, such RC6, and let $m \geq 1$ be a number. The CBC MAC uses the block cipher E to generate an n -bit MAC for messages whose lengths are some multiple of n . More explicitly, the m -fold CBC MAC over E is the deterministic message authentication code $\text{CBC}^m(E)$ in which the tag of an m -block message is the last block of ciphertext obtained by processing the message in CBC mode with zero IV. In more detail, the scheme $\text{CBC}^m(E) = (\mathcal{K}, \text{MAC})$ is defined as follows. The key space is $\text{Key} = \{0, 1\}^k$. The message space is $\text{Msgs} = \{0, 1\}^{nm}$. The tag space is $\text{Tags} = \{0, 1\}^n$. The key-generation algorithm is the algorithm which picks a random k -bit string K and outputs it. The MAC-generation algorithm takes a message M and does the following:

Algorithm $\text{MAC}_K(M)$

Divide M into n -bit blocks, $M = M_1 \cdots M_m$

$C_0 \leftarrow 0^n$

For $i = 1, \dots, m$ do $C_i \leftarrow E_K(C_{i-1} \oplus M_i)$

Return C_m

See Figure 6.2.

Since the MAC is deterministic, the MAC-verification algorithm is understood. It just checks, on input (K, M, Tag) , if $\text{Tag} = \text{MAC}_K(M)$.

As we will see later, the choice of message space is important for the security of the CBC MAC. If we had taken the message space to be $\text{Msgs} = \cup_{m \geq 1} \{0, 1\}^{nm}$, as

may seem natural, this MAC will be insecure. But if the length of the messages is restricted to some single fixed value, as above, the scheme is secure. We will address security in Section ??.

Example 6.4 [HMAC] *Describe HMAC.*

Example 6.5 [Encrypting a MAC'ed Message] Both of the examples given above were MACs. Here is an example of a message-authentication scheme which is not a MAC. ...

6.5 Towards a Definition of Security

Rather than put down a definition for security as though it fell from the sky, let us spend some time to build up our intuition about what properties a message-authentication code should have to deserve to be called “secure”. Let’s concentrate on MACs, as it will be easy to lift our definition to general message-authentication schemes.

The goal that we seek to achieve with a MAC is to be able to detect any attempt by the adversary to modify the transmitted data. We don’t want the adversary to be able to produce messages that the Receiver will deem authentic—only the Sender should be able to do this. That is, we don’t want that the adversary A to be able to create a pair (M, Tag) such that $VF_K(M, Tag) = 1$, but M did not originate with the Sender S . Such a pair (M, Tag) is called a *forgery*. If the adversary can make such a pair, she is said to have forged.

In some discussions of security people assume that the adversary’s goal is to recover the secret key K . Certainly if she could do this, it would be a disaster, since she could then forge anything. It is important to understand, however, that an adversary might be able to forge *without* being able to recover the key, and if all we asked was for the adversary to be unable to recover the key, we’d be asking too little. Forgery is what counts, not key recovery.

Now it should be admitted right away that some forgeries might be useless to the adversary. For example, maybe the adversary can forge, but she can only forge strings that look random; meanwhile, suppose that all “good” messages are supposed to have a certain format. Should this really be viewed as a forgery? The answer is *yes*. If checking that the message of a certain format was really a part of validating the message, then that should have been considered as part of the message-authentication scheme. In the absence of this, it is not for us to make assumptions about how the messages are formatted or interpreted. We really have no idea. Good protocol design means the security is guaranteed no matter what is the application. Asking that the adversary be unable to forge “meaningful” messages, whatever that might mean, would again be asking too little.

In our adversary’s attempt to forge a message we could consider various attacks. The simplest setting is that the adversary wants to forge a message even though she has never seen any transmission sent by the Sender. In this case the adversary must

concoct a pair (M, Tag) which passes the verification test, even though she hasn't obtained any information to help. This is called a *no-message attack*. It often falls short of capturing the capabilities of realistic adversaries, since an adversary who can inject bogus messages onto the communications media can probably see valid messages as well. We should let the adversary use this information.

Suppose the Sender sends the transmission (M, Tag) consisting of some message M and its legitimate tag Tag . The Receiver will certainly accept this—we demanded that. Now at once a simple attack comes to mind: the adversary can just repeat this transmission, (M, Tag) , and get the Receiver to accept it once again. This attack is unavoidable, so far, in that we required in the syntax of a MAC for the MAC-verification functions to be stateless. If the Verifier accepted (M, Tag) once, he's bound to do it again.

What we have just described is called a *replay attack*. The adversary sees a valid (M, Tag) from the Sender, and at some later point in time she re-transmits it. Since the Receiver accepted it the first time, he'll do so again.

Should a replay attack count as a valid forgery? In real life it usually should. Say the first message was “Transfer \$1000 from my account to the account of party A .” Then party A may have a simple way to enriching herself: she just keeps replaying this same MAC'ed message, happily watching her bank balance grow.

It is important 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* already produced by the Sender. We will see later that we can always achieve security against replay attacks by simple means; that is, we can take any MAC which is not secure against replay attacks and modify it—after making the Verifier stateful—so that it will be secure against replay attacks. At this point, not worrying about replay attacks results in a cleaner problem definition. And it leads us to a more modular protocol-design approach—that is, we cut up the problem into sensible parts (“basic security” and then “replay security”) solving them one by one.

If the adversary wants to be successful she can take a valid pair (M, Tag) and use it to concoct pair valid pair (M', Tag') such that $M \neq M'$. If she can do this, she has won. This is sometimes called a *substitution attack*. In a substitution attack the adversary takes a single message and tag, (M, Tag) , and uses them in her attempt to forge.

Of course there is no reason to think that the adversary will be limited to seeing only one example message. Realistic adversaries may see millions of authenticated messages, and still it should be hard for them to forge.

For some MACs the adversary's ability to forge will grow with q_s —that is, her forgery probability can be expected to grow with the number of examples of legitimately authenticated messages. Likewise, in some security systems the number of valid (M, Tag) pairs that the adversary can obtain may be architecturally limited. (For example, a stateful Signer may be unwilling to MAC more than a certain number of messages.) So when we give our quantitative treatment of security we will

treat q_s as an important adversarial resource.

How exactly do all these tagged messages arise? We could think of there being some distribution on messages that the Sender will authenticate, but in some settings it is even possible for the adversary to influence which messages are tagged. In the worst case, imagine that the adversary *herself* chooses which messages get authenticated. That is, the adversary chooses a message, gets its MAC, chooses another message, gets its MAC, and so forth. Then she tries to forge. This is called an *adaptive chosen-message attack*. It is the same type of attack that we concentrated on in defining secure encryption.

At first glance it may seem like an adaptive chosen-message attack is unrealistically generous to our adversary; after all, if an adversary could really obtain a valid MAC for *any* message she wanted, wouldn't that make moot the whole point of authenticating messages be useless? In fact, there are several good arguments for allowing the adversary such a strong capability. First, we will see examples—higher-level protocols that use MACs—where adaptive chosen-message attacks are quite realistic. Second, recall our general principles. We want to design schemes which are secure in *any* usage. This requires that we make worst-case notions of security, so that when we err in realistically modelling adversarial capabilities, we err on the side of caution, allowing the adversary more power than she might really have. Since eventually we will design schemes that meet our stringent notions of security, we only gain when we assume our adversary to be strong.

As an example of a simple scenario in which an adaptive chosen-message attack is realistic, imagine that the Sender S is forwarding messages to a Receiver R . The Sender receives messages from any number of third parties, A_1, \dots, A_n . The Sender gets a piece of data M from party A_i along a secure channel, and then the Sender transmits to the Receiver $\langle i \rangle \| M \| MAC_K(\langle i \rangle \| M)$. This is the Sender's way of attesting to the fact that he has received message M from party A_i . Now if one of these third parties, say A_1 , wants to play an adversarial role, she will ask the Sender to forward her adaptively-chosen messages M_1, M_2, \dots to the Receiver. If, based on what she sees, she can learn the key K , or even if she can learn to forge message of the form $\langle 2 \rangle \| M$, so as to produce a valid $\langle 2 \rangle \| M \| MAC_K(\langle 2 \rangle \| M)$, then the intent of the protocol will have been defeated, even though most it has correctly used a MAC.

So far we have said that we want to give our adversary the ability to obtain MACs for messages of her choosing, and then we want to look at whether or not she can forge: produce a valid (M, Tag) where she never asked the adversary to MAC M . But we should recognize that a realistic adversary might be able to produce lots of candidate forgeries, and she may be content if any of these turn out to be valid. We can model this possibility by giving the adversary the capability to tell if a prospective (M, Tag) pair is valid, and saying that the adversary forges if she ever finds an (M, Tag) pair that is. We'll rule out messages that the adversary already knows a tag for.

Whether or not a real adversary can try lots of possible forgeries depends on

Figure 6.4: *The model for a message authentication code. Adversary A has access to a MAC-generation oracle and a MAC-verification oracle. The adversary wants to get the MAC-verification oracle to accept some (M, Tag) for which she didn't earlier ask the MAC-generation oracle for M .*

the context. Suppose the Verifier is going to tear down a connection the moment he detects an invalid tag. Then it is unrealistic to try to use this Verifier to help you determine if a candidate pair (M, Tag) is valid—one mistake, and you're done for. In this case, thinking of there being a single attempt to forge a message is quite adequate.

On the other hand, suppose that a Verifier just ignores any improperly tagged message, while she responds in some noticeably different way if she receives a properly authenticated message. In this case a quite reasonable adversarial strategy may be ask the Verifier about the validity of a large number of candidate (M, Tag) pairs. The adversary hopes to find at least one that is valid. When the adversary finds such an (M, Tag) pair, we'll say that she has won.

Let us summarize. To be fully general, we will give our adversary two different capabilities. The first adversarial capability is to obtain a MAC M for any message that she chooses. We will call this a signing query. The adversary will make some number of them, q_s . The second adversarial capability is to find out if a particular pair (M, Tag) is valid. We will call this a verification query. The adversary will make some number of them, q_v . Our adversary is said to succeed—to forge—if she ever makes a verification query (M, Tag) and gets a return value of ACCEPT even though the message M is not a message that the adversary already knew a tag for by virtue of an earlier signing query. Let us now proceed more formally.

6.6 Definition of security

Let $\mathcal{MA} = (\mathcal{K}, \text{MAC}, \text{VF})$ be an arbitrary message authentication scheme. We will formalize a quantitative notion of security against adaptive chosen-message attack. We begin by describing the model.

We begin by distilling out the model from the intuition we have described. There is no need, in the model, to think of the Sender and the Verifier as animate entities. The purpose of the Sender, from the adversary’s point of view, is to authenticate messages. So we will embody the Sender as an oracle that the adversary can use to authenticate any message M . This “signing oracle,” as we will call it, is our way to provide the adversary black-box access to the function $\text{MAC}_K(\cdot)$. Likewise, the purpose of the Verifier, from the adversary’s point of view, is to have something to whom to send attempted forgeries. So we will embody the Verifier as an oracle that the adversary can use to see if a candidate pair (M, Tag) is valid. This “verification oracle,” as we will call it, is our way to provide the adversary black-box access to the function $\text{VF}_K(\cdot)$. Thus, when we become formal, the cast of characters—the Sender, Verifier, and Adversary—gets reduced to just the adversary, running with her oracles. The Sender and Verifier have vanished—reduced to oracles, poor things.

Here, in detail, is how the game is run—the experiment which defines whether or not the adversary wins when she attacks the message-authentication code $\mathcal{MA} = (\mathcal{K}, \text{MAC}, \text{VF})$.

Definition 6.6 [MAC Security] Let $\mathcal{MA} = (\mathcal{K}, \text{MAC}, \text{VF})$ be a message authentication code, and let A be an adversary. Let $\text{Adv}_{\Pi}^{\text{mac}}(A)$ denote the probability that A succeeds in the following experiment:

Let $K \xleftarrow{R} \mathcal{K}$
 Run $A^{\text{MAC}_K(\cdot), \text{VF}_K(\cdot)}$
 If A ever asked a MAC-verification query $\text{VF}_K(M, \text{Tag})$,
 getting a return value of ACCEPT, and A did not earlier ask
 MAC-generation query $\text{MAC}_K(M)$, then A *succeeds*;
 else A *fails*.

Let $q_s, q_v, m, t \geq 0$ be numbers. Then we let

$$\text{Adv}_{\Pi}^{\text{mac}}(q_s, q_v, m, t) = \max_A \{ \text{Adv}_{\Pi}^{\text{mac}}(A) \}$$

where the maximum is over all adversaries A that make at most q_s signing queries, at most q_v verification queries, each signing and verification query is of length at most m , and the adversary’s total running time is t . ■■

Let us discuss the above definition. Fix a MAC scheme Π . Then we associate to any adversary A its “advantage,” or “success probability.” We denote this value as $\text{Adv}_{\Pi}^{\text{mac}}(A)$. It’s just the chance that A manages to forge. The probability is over the choice of key K , any probabilistic choices that MAC might make, and the probabilistic choices, if any, that the adversary A makes. The insecurity of the MAC itself, which we also denote with Adv , is the success probability of the “cleverest” possible adversary, amongst all adversaries restricted to specified computational resources.

As usual, there is a certain amount of arbitrariness as to which resources we measure. Certainly it is important to separate the oracle queries (q_s and q_v) from the time. In practice, signing queries correspond to messages sent by the legitimate sender, and obtaining these is probably more difficult than just computing on one's own. Verification queries correspond to messages the adversary hopes the Verifier will accept, so finding out if she does accept these queries again requires interaction. Some system architectures may effectively limit q_s and q_v . No system architecture can limit t —that is limited primarily by the adversary's budget.

We emphasize that there are contexts in which you are happy with a MAC that makes forgery impractical when $q_v = 1$ and $q_s = 0$ (an “impersonation attack”) and there are contexts in which you are happy when forgery is impractical when $q_v = 1$ and $q_s = 1$ (a “substitution attack”). But it is perhaps more common that you'd like for forgery to be impractical even when q_s is large, like 2^{50} , and maybe when q_v is large, too.

The maximal length of each message provided to an oracle, m , is a resource that could well be parameterized in different ways. For example, we could, alternatively, have looked at the total length of all queries, m . Or we could look at the $(q_s + q_v)$ -vector which specifies the length of each of the oracle queries. The point is simply that getting lots of bits of MACed message may be more difficult than getting a few bits. Perhaps the Signer won't MAC messages that a gigabytes long—and no doubt sending such long messages takes longer than sending short ones. We choose to parameterize by maximal message length simply because it is convenient for the results we will show.

In some MAC schemes security does not depend on the adversary's running time, t , being bounded. In this case the scheme is said to be “information-theoretically secure.” Likewise, in some MAC schemes security does not degrade with m or q_s .

Naturally the key K is not directly given to the adversary, and neither are any random choices or counter used by the MAC-generation algorithm. The adversary sees these things only to the extent that they are reflected in the answers to her oracle queries.

6.7 Example schemes

Mihir, I'm unconvinced that this is really needed, but maybe examples never hurt....

Let us examine some example message authentication schemes and use the definition to assess their strengths and weaknesses. We fix a PRF $F: \{0, 1\}^k \times \{0, 1\}^l \rightarrow \{0, 1\}^L$. Our first scheme $\mathcal{MA}_1 = (\mathcal{K}, \text{MAC}, \mathcal{V})$ works like this—

Algorithm $\text{MAC}_K(M)$ Divide M into l bit blocks, $M = x_1 \dots x_n$ For $i = 1, \dots, n$ do $y_i \leftarrow F_K(x_i)$ $\sigma \leftarrow y_1 \oplus \dots \oplus y_n$ Return σ	Algorithm $\mathcal{V}_K(M, \sigma)$ Divide M into l bit blocks, $M = x_1 \dots x_n$ For $i = 1, \dots, n$ do $y_i \leftarrow F_K(x_i)$ $\sigma' \leftarrow y_1 \oplus \dots \oplus y_n$ If $\sigma = \sigma'$ then return 1 else return 0
--	--

Now let us try to assess the security of this message authentication scheme.

Suppose the adversary wants to forge the tag of a certain given message M . A priori it is unclear this can be done. The adversary is not in possession of the secret key K , so cannot compute F_K and hence will have a hard time computing σ . However, remember that the notion of security we have defined says that the adversary is successful as long as it can produce a correct tag for *some* message, not necessarily a given one. We now note that even without a chosen-message attack (in fact without seeing any examples of correctly tagged data) the adversary can do this. It can choose a message M consisting of two equal blocks, say $M = x||x$ where x is some l -bit string, set $\sigma \leftarrow 0^l$, and output M, σ . Notice that $\mathcal{V}_K(M, \sigma) = 1$ because $F_K(x) \oplus F_K(x) = 0^l = \sigma$. So the adversary is successful. In more detail, the adversary is:

Adversary $A_1^{\text{MAC}_K(\cdot)}$
 Let x be some l -bit string
 Let $M \leftarrow x||x$
 Let $\sigma \leftarrow 0^l$
 Return (M, σ)

Then $\text{Adv}^{\text{ma}}(\mathcal{MA}_1, A_1) = 1$. Furthermore A_1 makes no oracle queries, uses $t = O(l)$ time, and outputs an l -bit message in its forgery, so we have shown that

$$\text{Adv}_{(\cdot)}^{\text{mac-frg}} \mathcal{MA}_1; t, 0, l) = 1.$$

That is, the scheme \mathcal{MA}_1 is totally insecure.

There are many other attacks. For example we note that if $\sigma = F_K(M_1) \oplus F_K(M_2)$ is the tag of M_1M_2 then σ is also the correct tag of M_2M_1 . So it is possible, given the tag of a message, to forge the tag of a new message formed by permuting the blocks of the old message. We leave it to the reader to specify the corresponding adversary and compute its advantage.

Let us now try to strengthen the scheme to avoid these attacks. Instead of applying F_K to a data block, we will first prefix the data block with its index. To do this we pick some parameter m with $1 \leq m \leq l - 1$, and write the index as an m -bit string. The message authentication scheme $\mathcal{MA}_1 = (\mathcal{K}, \text{MAC}, \mathcal{V})$ looks like this:

Algorithm $\text{MAC}_K(M)$

Divide M into $l - m$ bit blocks, $M = x_1 \dots x_n$
 For $i = 1, \dots, n$ do $y_i \leftarrow F_K(\langle i \rangle \| x_i)$
 $\sigma \leftarrow y_1 \oplus \dots \oplus y_n$
 Return σ

Algorithm $\mathcal{V}_K(M, \sigma)$

Divide M into $l - m$ bit blocks, $M = x_1 \dots x_n$
 For $i = 1, \dots, n$ do $y_i \leftarrow F_K(\langle i \rangle \| x_i)$
 $\sigma' \leftarrow y_1 \oplus \dots \oplus y_n$
 If $\sigma = \sigma'$ then return 1 else return 0

As the code indicates, we divide M into smaller blocks: not of size l , but of size $l - m$. Then we prefix the i -th message block with the value i itself, the block index, written in binary. Above $\langle i \rangle$ denotes the integer i written as a binary string of m bits. It is to this padded block that we apply F_K before taking the XOR.

Note that encoding of the block index i as an m -bit string is only possible if $i < 2^m$. This means that we cannot authenticate a message M having more than 2^m blocks. That is, the message space is confined to strings of length at most $(l - m)(2^m - 1)$, and, for simplicity, of length a multiple of $l - m$ bits. However this is hardly a restriction in practice since a reasonable value of m , like $m = 32$, is large enough that typical messages fall in the message space, and since l is typically at least 64, we have at least 32 bits left for the data itself.

Anyway, the question we are really concerned with is the security. Has this improved with respect to \mathcal{MA}_1 ? Begin by noticing that the attacks we found on \mathcal{MA}_1 no longer work. For example take the adversary A_1 above. (It needs a minor modification to make sense in the new setting, namely the chosen block x should not be of length l but of length $l - m$. Consider this modification made.) What is its success probability when viewed as an adversary attacking \mathcal{MA}_2 ? The question amounts to asking what is the chance that $\mathcal{V}_K(M, \sigma) = 1$ where \mathcal{V} is the verification algorithm of our amended scheme and M, σ is the output of A_1 . The verification algorithm will compute $\sigma' = F_K(\langle 1 \rangle \| x) \oplus F_K(\langle 2 \rangle \| x)$ and test whether this equals 0^l , the value of σ output by A . This happens only when

$$F_K(\langle 1 \rangle \| x) = F_K(\langle 2 \rangle \| x),$$

and this is rather unlikely. For example if we are using a block cipher it never happens because F_K is a permutation. Even when F is not a block cipher, this event has very low probability as long as F is a good PRF; specifically, $\mathbf{Adv}^{\text{ma}}(\mathcal{MA}_2, A_1)$ is at most $\mathbf{Adv}_F^{\text{prf}}(t, 2)$ where $t = O(l)$. (A reader might make sure they see why this bound is true.) So the attack has very low success probability.

Similar arguments show that the second attack discussed above, namely that based on permuting of message blocks, also has low success against the new scheme. Why? In the new scheme

$$\begin{aligned} \text{MAC}_K(M_1 M_2) &= F_K(\langle 1 \rangle \| M_1) \oplus F_K(\langle 2 \rangle \| M_2) \\ \text{MAC}_K(M_2 M_1) &= F_K(\langle 1 \rangle \| M_2) \oplus F_K(\langle 2 \rangle \| M_1). \end{aligned}$$

These are unlikely to be equal for the same reasons discussed above. As an exercise, a reader might upper bound the probability that these values are equal in terms of the value of the insecurity of F at appropriate parameter values.

However, \mathcal{MA}_2 is still insecure. The attacks however require a more non-trivial usage of the chosen-message attacking ability. The adversary will query the tagging oracle at several related points and combine the responses into the tag of a new message. We call it A_2^-

Adversary $A_2^{\text{MAC}_K(\cdot)}$

Let x_1, x'_1 be distinct, $l - m$ bit strings, and let x_2, x'_2 be distinct $l - m$ bit strings
 $\sigma_1 \leftarrow \text{MAC}_K(x_1x_2)$; $\sigma_2 \leftarrow \text{MAC}_K(x_1x'_2)$; $\sigma_3 \leftarrow \text{MAC}_K(x'_1x_2)$
 $\sigma \leftarrow \sigma_1 \oplus \sigma_2 \oplus \sigma_3$
 Return $(x'_1x'_2, \sigma)$

We claim that $\text{Adv}^{\text{ma}}(\mathcal{MA}_2, A_2) = 1$. Why? This requires two things. First that $\mathcal{V}_K(x'_1x'_2, \sigma) = 1$, and second that $x'_1x'_2$ was never a query to $\text{MAC}_K(\cdot)$ in the above code. The latter is true because we insisted above that $x_1 \neq x'_1$ and $x_2 \neq x'_2$, which together mean that $x'_1x'_2 \notin \{x_1x_2, x_1x'_2, x'_1x_2\}$. So now let us check the first claim. We use the definition of the tagging algorithm to see that

$$\begin{aligned}\sigma_1 &= F_K(\langle 1 \rangle \| x_1) \oplus F_K(\langle 2 \rangle \| x_2) \\ \sigma_2 &= F_K(\langle 1 \rangle \| x_1) \oplus F_K(\langle 2 \rangle \| x'_2) \\ \sigma_3 &= F_K(\langle 1 \rangle \| x'_1) \oplus F_K(\langle 2 \rangle \| x_2) .\end{aligned}$$

Now look how A_2 defined σ and do the computation; due to cancellations we get

$$\begin{aligned}\sigma &= \sigma_1 \oplus \sigma_2 \oplus \sigma_3 \\ &= F_K(\langle 1 \rangle \| x'_1) \oplus F_K(\langle 2 \rangle \| x'_2) .\end{aligned}$$

This is indeed the correct tag of $x'_1x'_2$, meaning the value σ' that $\mathcal{V}_K(x'_1x'_2, \sigma)$ would compute, so the latter algorithm returns 1, as claimed. In summary we have shown that

$$\text{Adv}_{\text{}}^{\text{mac-frag}} \mathcal{MA}_2; t, 3, 4(l - m) = 1 ,$$

where $t = O(l)$. So the scheme \mathcal{MA}_2 is also totally insecure.

Later we will see how a slight modification of the above actually yields a secure scheme. For the moment however we want to stress a feature of the above attacks. Namely that these attacks *did not cryptanalyze the PRF F* . The cryptanalysis of the message authentication schemes did not care anything about the structure of F ; whether it was DES, RC6, or anything else. They found weaknesses in the message authentication schemes themselves. In particular, the attacks work just as well when F_K is a random function, or a “perfect” cipher. This illustrates again the point we have been making, about the distinction between a tool (here the PRF) and its usage. We need to make better usage of the tool, and in fact to tie the security of the scheme to that of the underlying tool in such a way that attacks like those illustrated here are provably impossible under the assumption that the tool is secure.

6.8 The PRF-as-a-MAC Paradigm

Pseudorandom functions make good MACs, and constructing a MAC in this way is an excellent approach. Here we show why PRFs are good MACs, and determine the concrete security of the underlying reduction. The following shows that the reduction is almost tight—security hardly degrades at all.

Note that when we think of a PRF as a MAC it is important that the domain of the PRF be whatever one wants as the domain of the MAC. So such a PRF probably can't be realized as a block cipher. It may have to be realized by a PRF that allows for inputs of many different lengths, since you might want to MAC messages of many different lengths. As yet we haven't demonstrated that we can make such PRFs. But we will.

Let us restate the definition of a PRF, to make sure that the variable domain is clear.

Definition 6.7 A **pseudorandom function** (PRF) is a function $F : \text{Key} \times \text{Message} \rightarrow \{0, 1\}^n$ where Key is a finite set (or else it comes endowed with a probability measure), and Message is a nonempty set of strings. ■

To make a MAC from a PRF F we are simply setting the key generator to be the algorithm that samples from Key , and we set $\text{MAC}_a(M) = F_a(M)$. The MAC is deterministic, so we don't have to separately specify a MAC-verification function. For notational convenience, we will not distinguish between F as a PRF and the F -induced MAC scheme.

Proposition 6.8 Let $F : \text{Key} \times \text{Message} \rightarrow \{0, 1\}^n$ be a PRF. Suppose that there exists an adversary A_{mac} that, running in time t_{mac} , asking q_{mac} queries, these totalling μ_{mac} bits, forges with probability $\epsilon_{\text{mac}} = \mathbf{Adv}_F^{\text{mac}}(A_{\text{mac}})$. Then there exists an adversary A_{prf} that, running in time t_{prf} , asking q_{prf} queries, these totalling μ_{prf} bits, distinguishes a random instance of F from a random function with advantage $\epsilon_{\text{prf}} = \mathbf{Adv}_F^{\text{prf}}(A_{\text{prf}})$ where

$$t_{\text{prf}} = \tilde{O}(t_{\text{mac}}), \quad q_{\text{prf}} = q_{\text{mac}} + 1, \quad \mu_{\text{prf}} = \mu_{\text{mac}}, \quad \text{and} \quad \epsilon_{\text{prf}} = \epsilon_{\text{mac}} - 2^{-n}.$$

Proof: Adversary A_{prf} has an oracle f . Let A_{prf} work as follows

Run A_{mac} .
 When A_{mac} makes a query, x , to its (MAC) oracle g , return $f(x)$.
 Finally A_{mac} halts, outputting a pair (x^*, σ^*) .
 If x^* was not already asked of f , and $f(x^*) = \sigma^*$,
 then output 1, otherwise output 0.

With the obvious shorthand,

$$\mathbf{Adv}_F^{\text{prf}}(A_{\text{prf}}) = \Pr[A_{\text{prf}}^{F_a(\cdot)} = 1] - \Pr[A_{\text{prf}}^{\rho} = 1]$$

$$\begin{aligned} &\leq \Pr[A_{\text{mac}}^{F_a(\cdot)} \text{ forges}] - 2^{-n} \\ &= \epsilon_{\text{mac}} - 2^{-n} \end{aligned}$$

The running time and query complexity of A_{prf} are clearly as claimed: the \tilde{O} overhead is for checking if x^* is a new query, while the $+1$ in the query complexity accounts for asking $f(x^*)$. Recall that, by our convention, μ_{mac} already includes the length of x^* . ■ ■

6.9 Making a PRF from a PRF and a Universal Hash Function

We have shown that one paradigm for making a good MAC is to make something stronger: a good PRF. Unfortunately, out-of-the-box PRFs usually operate on strings of some fixed length, like 128 bits. That's almost certainly not the domain that we want for our MAC's message space. In this section we describe a simple paradigm for extending the domain of a PRF by using a universal hash-function family. Several MACs can be seen as instances of this approach.

Theorem 6.9 Let $H : \text{Key}(H) \times \text{Message} \rightarrow \{0, 1\}^n$ be a δ -AU hash-function family. Let $F : \text{Key}(F) \times \{0, 1\}^n \rightarrow \{0, 1\}^s$ be $\epsilon_F(t, q)$ -secure, as a PRF; that is, $\epsilon_F(t, q) = \text{Adv}_F^{\text{prf}}(t, q)$. Define the PRF $FH : (\text{Key}(F) \times \text{Key}(H)) \times \text{Message} \rightarrow \{0, 1\}^s$ by $FH_{(a,k)}(x) = F_a(H_k(x))$. Then FH is (t', q', μ, ϵ') -secure, as a PRF, where \dots . ■

To be completed.

Let us give a concrete example of this approach. We saw in Chapter 5 that one could hash a sequence of words $M_{m-1} \dots M_0$ using a key $k \in \{0, 2^{89} - 1\}$ by computing $H_k(M) = (k^m + M_{m-1}k^{m-1} + \dots + M_1k + M_0) \bmod (2^{89} - 1)$. If the message is limited to $2^{32} - 1$ words, say, then this hash function family has collision probability bounded by $\frac{2^{32}-1}{2^{89}-1} > 2^{-57}$. So to MAC a message M , compute $H_k(M)$, encode this into a 128-bit string, and apply AES_a , yielding the desired authentication tag.

6.10 An XOR Scheme

Eliminated. Plan to completely revise, directly proving the Bernstein Bernstein-variant of our XOR MAC (that's the version where you encipher the XOR of the PRF outputs), which has a trivial proof in the above framework.

6.11 The EMAC Construction

We wish to show that if $M, M' \in (\{0, 1\}^n)^+$ are distinct strings then $\Pr_{\pi}[\text{CBC}_{\pi}(M) = \text{CBC}_{\pi}(M')]$ is small. By "small" we mean a slowly growing function of $m = |M|/n$

Figure 6.5: A fragment of the CBC construction showing the labeling convention used in the proof of Lemma ??.

and $m' = |M'|/n$. Formally, for $n, m, m' \geq 1$, define the *collision probability* of the CBC MAC to be

$$V_n(m, m') \stackrel{\text{def}}{=} \max_{M \in \{0,1\}^{nm}, M' \in \{0,1\}^{nm'}, M \neq M'} \{ \Pr[\pi \xleftarrow{R} \text{Perm}(n) : \text{CBC}_\pi(M) = \text{CBC}_\pi(M')] \} .$$

(The character “V” is meant to suggest collisions.)

Lemma 6.10 [CBC MAC Collision Bound] Let $n, m, m' \geq 1$. Then

$$V_n(m, m') \leq \frac{2.5(m + m')^2}{2^n} .$$

■

Proof: Although M and M' are distinct, they may share some common prefix. Let k be the index of the last block in which M and M' agree. (If M and M' have unequal first blocks then $k = 0$.)

Each particular permutation π is equally likely among all permutations from $\{0, 1\}^n$ to $\{0, 1\}^n$. In our analysis, we will view the selection of π as an incremental procedure. This will be equivalent to selecting π uniformly at random. In particular, we view the computation of $\text{CBC}_\pi(M)$ and $\text{CBC}_\pi(M')$ as playing the game given in Figure 6.6. Here the notation M_i indicates the i th block of M . We initially set each range point of π as **undefined**; the notation $\text{Domain}(\pi)$ represents the set of points x where $\pi(x)$ is no longer **undefined**. We use $\text{Range}(\pi)$ to denote the set of points $\pi(x)$ which are no longer **undefined**; we use $\overline{\text{Range}}(\pi)$ to denote $\{0, 1\}^n - \text{Range}(\pi)$.

During the game, the X_i are those values produced after XORing with the current message block, M_i , and the Y_i values are $\pi(X_i)$. See Figure 6.5.

We are concerned with the probability that π will cause $\text{CBC}_\pi(M) = \text{CBC}_\pi(M')$, which will occur in our game iff $Y_m = Y'_{m'}$. Since π is invertible, this occurs iff $X_m = X'_{m'}$. As we shall see, this condition will cause *bad* = true in our game. However, we actually set *bad* to true in many other cases in order to simplify the analysis.

```

1:  $bad \leftarrow false$ ;   for all  $x \in \{0,1\}^n$  do  $\pi(x) \leftarrow undefined$ ;    $X_1 \leftarrow M_1$ ;    $X'_1 \leftarrow M'_1$ ;    $BAD \leftarrow \{X_1, X'_1\}$ 

2: for  $i \leftarrow 1$  to  $k$  do
3:   if  $X_i \in \text{Domain}(\pi)$  then  $Y_i \leftarrow Y'_i \leftarrow \pi(X_i)$ 
4:   else  $Y_i \leftarrow Y'_i \xleftarrow{R} \overline{\text{Range}}(\pi)$ ;    $\pi(X_i) \leftarrow Y_i$ 
5:     if  $i < m$  then  $X_{i+1} \leftarrow Y_i \oplus M_{i+1}$ 
6:     if  $X_{i+1} \in BAD$  then  $bad \leftarrow true$  else  $BAD \leftarrow BAD \cup \{X_{i+1}\}$ 
7:     if  $i < m'$  then  $X'_{i+1} \leftarrow Y'_i \oplus M'_{i+1}$ 
8:     if  $X'_{i+1} \in BAD$  then  $bad \leftarrow true$  else  $BAD \leftarrow BAD \cup \{X'_{i+1}\}$ 

9: for  $i \leftarrow k+1$  to  $m$  do
10:  if  $X_i \in \text{Domain}(\pi)$  then  $Y_i \leftarrow \pi(X_i)$ 
11:  else  $Y_i \xleftarrow{R} \overline{\text{Range}}(\pi)$ ;    $\pi(X_i) \leftarrow Y_i$ 
12:    if  $i < m$  then  $X_{i+1} \leftarrow Y_i \oplus M_{i+1}$ 
13:    if  $X_{i+1} \in BAD$  then  $bad \leftarrow true$  else  $BAD \leftarrow BAD \cup \{X_{i+1}\}$ 

14: for  $i \leftarrow k+1$  to  $m'$  do
15:  if  $X'_i \in \text{Domain}(\pi)$  then  $Y'_i \leftarrow \pi(X'_i)$ 
16:  else  $Y'_i \xleftarrow{R} \overline{\text{Range}}(\pi)$ ;    $\pi(X'_i) \leftarrow Y'_i$ 
17:    if  $i < m$  then  $X'_{i+1} \leftarrow Y'_i \oplus M'_{i+1}$ 
18:    if  $X'_{i+1} \in BAD$  then  $bad \leftarrow true$  else  $BAD \leftarrow BAD \cup \{X'_{i+1}\}$ 

```

Figure 6.6: Game used in the proof of Lemma ???. The algorithm gives one way to compute the CBC MAC of distinct messages $M = M_1 \cdots M_m$ and $M' = M'_1 \cdots M'_{m'}$. These messages are identical up to block k , but different afterwards. The computed MACs are Y_m and $Y_{m'}$, respectively.

The idea behind the variable *bad* is as follows: throughout the program (lines 4, 11, and 16) we randomly choose a range value for π at some **undefined** domain point. Since π has not yet been determined at this point, the selection of our range value will be an independent uniform selection: there is no dependence on any prior choice. If the range value for π were already determined by some earlier choice, the analysis would become more involved. We avoid the latter condition by setting *bad* to true whenever such interdependencies are detected. The detection mechanism works as follows: throughout the processing of M and M' we will require π be evaluated at $m + m'$ domain points X_1, \dots, X_m and $X'_1, \dots, X'_{m'}$. If all of these domain points are distinct (ignoring duplications due to any common prefix of M and M'), we can rest assured that we are free to assign their corresponding range points without constraint. We maintain a set *BAD* to track which domain points have already been determined; initially X_1 and X'_1 are the only such points, since future values will depend on random choices not yet made. Of course if $k > 0$ then $X_1 = X'_1$ and *BAD* contains only one value. Next we begin randomly choosing range points; if ever any such choice leads to a value already contained in the *BAD* set, we set the flag *bad* to true.

We now bound the probability of the event that *bad* = true by analyzing our game. The variable *bad* can be set true in lines 6, 8, 13, and 18. In each case it is required that some Y_i was selected such that $Y_i \oplus M_{i+1} \in \text{BAD}$ (or possibly that some Y'_i was selected such that $Y'_i \oplus M'_{i+1} \in \text{BAD}$). The set *BAD* begins with at most 2 elements and then grows by 1 with each random choice of Y_i or Y'_i . We know that on the i th random choice in the game the *BAD* set will contain at most $i + 1$ elements. And so each random choice of Y_i (resp. Y'_i) from the co-range of π will cause $Y_i \oplus M_{i+1}$ (resp. $Y'_i \oplus M'_{i+1}$) to be in *BAD* with probability at most $(i + 1)/(N - i + 1)$. We have already argued that in the absence of *bad* = true each of the random choices we make are independent. We make $m - 1$ choices of Y_i to produce X_2 through X_m and $m' - 1$ choices of Y'_i to determine X'_2 through $X'_{m'}$ and so we can compute

$$\Pr[\text{bad} = \text{true}] \leq \sum_{i=1}^{m-1+m'-1} \frac{i+1}{N-i+1}.$$

Using the fact that $m, m' \leq N/4$, we can bound the above by

$$\sum_{i=1}^{m+m'-2} \frac{i+1}{N-i} \leq \frac{2}{N} \sum_{i=1}^{m+m'-2} i+1 \leq \frac{(m+m')^2}{N}.$$

This completes the proof. ■

6.12 The HMAC Construction

6.13 The UMAC Construction

Sketch UMAC, particularly improving security with the use of counters (the “standard” Wegman-Carter method), and the hash function NH). Old material follows.

Today the most effective paradigm for fast message authentication is based on the use of “almost xor universal hash functions”. The design of these hash functions receives much attention and has resulted in some very fast ones, so that universal hash based MACs are the fastest MACs around. Let us begin by describing the tool, and then seeing how it can be used for message authentication.

6.13.1 Almost xor universal hash functions

Let $H: \text{Keys}(H) \times \text{Dom}(H) \rightarrow \{0, 1\}^L$ be a family of functions. We think of them as hash functions because the domain $\text{Dom}(H)$ of any individual function H_K is typically large, being the message space of the desired message authentication scheme.

Fix any two points a_1, a_2 in the domain $\text{Dom}(H)$ of the family, the only restriction on them being that they are not allowed to be equal. Also fix a point b in the range $\{0, 1\}^L$ of the family. With H fixed, we can associate to these three points a probability

$$\begin{aligned} \text{UHColPr}_H(a_1, a_2, b) &= \Pr \left[K \stackrel{R}{\leftarrow} \text{Keys}(H) : H_K(a_1) \oplus H_K(a_2) = b \right] \\ &= \Pr \left[h \stackrel{R}{\leftarrow} H : h(a_1) \oplus h(a_2) = b \right] , \end{aligned}$$

the two expressions above being equal by definition. We are interested in keeping this probability low for all choices of a_1, a_2, b . The quality of H as an almost xor universal family, which we call the insecurity of H , is accordingly measured by the maximum value of this probability, the maximum being over the choices of a_1, a_2, b .

Definition 6.11 Let $H: \text{Keys}(H) \times \text{Dom}(H) \rightarrow \{0, 1\}^L$ be a family of functions. Let

$$\mathbf{Adv}^{\text{uh}}(H) = \max_{a_1, a_2, b} \left\{ \Pr \left[K \stackrel{R}{\leftarrow} \text{Keys}(H) : H_K(a_1) \oplus H_K(a_2) = b \right] \right\} ,$$

the maximum being over all *distinct* points $a_1, a_2 \in \text{Dom}(H)$ and all strings $b \in \{0, 1\}^L$. ■■

The smaller the value of $\mathbf{Adv}^{\text{uh}}(H)$, the better the quality of H as an almost xor-universal function. We say that H is a *xor-universal hash function* if $\mathbf{Adv}(H) = 2^{-L}$. (We will see later that this is the lowest possible value of the insecurity.)

The simplest example is the family of all functions.

Proposition 6.12 The family $R^{l,L}$ of all functions of l -bits to L -bits is xor-universal, meaning $\mathbf{Adv}^{\text{uh}}(R^{l,L}) = 2^{-L}$.

Proof: With distinct $a_1, a_2 \in \{0, 1\}^l$, and $b \in \{0, 1\}^L$ fixed, we clearly have

$$\Pr \left[h \stackrel{R}{\leftarrow} R^{l,L} : h(a_1) \oplus h(a_2) = b \right] = 2^{-L}$$

because h is a random function. ■ ■

Another source of examples is polynomials over finite fields.

Example 6.13 Identify $\{0, 1\}^l$ with $\text{GF}(2^l)$, the finite field of 2^l elements. We fix an irreducible, degree l polynomial over $\text{GF}(2)$ so as to be able to do arithmetic over the field. The hash function H we define takes as key a pair α, β of points in $\{0, 1\}^l$ such that $\alpha \neq 0$. The domain is $\{0, 1\}^l$ and the range is $\{0, 1\}^L$ where $L \leq l$. We define the function by

$$H_{\alpha, \beta}(x) = [\alpha x + \beta]_{1\dots L}.$$

That is, with key α, β and input $x \in \{0, 1\}^l$, first compute, in the finite field, the value $\alpha x + \beta$. View this as an l -bit string, and output the first L bits of it. ■

Proposition 6.14 The family $H: \text{Keys}(H) \times \{0, 1\}^l \rightarrow \{0, 1\}^L$ defined above, where $L \leq l$ and $\text{Keys}(H)$ is the set of all pairs (a, b) of l -bit strings such that $a \neq 0$, is a xor-universal hash function.

Proof: We need to show that $\mathbf{Adv}(H) = 2^{-L}$. Accordingly fix $a_1, a_2 \in \{0, 1\}^l$ such that $a_1 \neq a_2$, and fix $b \in \{0, 1\}^L$. Fix any key for the function, meaning any $\alpha \neq 0$ and any β . Notice that $y = \alpha x + \beta$ iff $x = \alpha^{-1}(y - \beta)$. (The arithmetic here is over the finite field, and we are using the assumption that $\alpha \neq 0$.) This means that the map of $\text{GF}(2^l)$ to $\text{GF}(2^l)$ given by $x \mapsto \alpha x + \beta$ is a permutation. The proposition follows from this. ■ ■

It is useful to interpret the almost xor-universal measure in another, more dynamic way. Imagine that the choice of the points a_1, a_2, b is made by an adversary. This adversary C knows that H is the target family. It clunks along for a while and then outputs some distinct values $a_1, a_2 \in \text{Dom}(H)$, and a value $b \in \{0, 1\}^L$. Now a key K is chosen at random, defining the function $H_K: \text{Dom}(H) \rightarrow \{0, 1\}^L$, and we test whether or not $H_K(a_1) \oplus H_K(a_2) = b$. If so, the adversary C wins. We denote the probability that the adversary wins by $\mathbf{Adv}^{\text{uh}}(H, C)$. We then claim that this probability is at most $\mathbf{Adv}^{\text{uh}}(H)$.

The reason is that there is a single best strategy for the adversary, namely to choose points a_1, a_2, b which maximize the probability $\text{UHColPr}_H(a_1, a_2, b)$ defined above. This should be relatively clear, at least for the case when the adversary is deterministic. But the claim is true even when the adversary is probabilistic,

meaning that the triple of points it outputs can be different depending on its own coin tosses. (In such a case, the probability defining $\mathbf{Adv}^{\text{uh}}(C)$ is taken over the choice of K and also the coin tosses of C .) We justify this claim in Proposition 6.15 below. We thus have two, equivalent ways of thinking about $\mathbf{Adv}^{\text{uh}}(H)$, one more “static” and the other more “dynamic”. Depending on the setting, we may benefit more from one view than another.

Before stating and proving Proposition 6.15, however, let us emphasize some features of this notion. A key feature of the game is that the steps must follow a particular order: *first* the adversary chooses points a_1, a_2, b , *then* K is chosen at random and the function H_K is defined. The adversary is *not* allowed to choose a_1, a_2, b as a function of K ; it must first commit to them, and then there is some probability of its winning the game.

This notion differs from others we have considered in that there is no computational restriction on the adversary. Namely, it can run for as long as it likes in deciding how to choose a_1, a_2, b , and the security condition is true nonetheless. Thus, it is a purely information theoretic notion.

Here now is the promised bound.

Proposition 6.15 Let $H\text{Keys}(H) \times \text{Dom}(H) \rightarrow \{0, 1\}^L$ be a family of functions and C a (possibly probabilistic) algorithm that outputs a triple a_1, a_2, b such that a_1, a_2 are distinct points in $\text{Dom}(H)$ and $b \in \{0, 1\}^L$. Then

$$\mathbf{Adv}^{\text{uh}}(H, C) \leq \mathbf{Adv}^{\text{uh}}(H).$$

Proof: Remember that to say C is probabilistic means that it has as an auxiliary input a sequence ρ of random bits of some length r , and uses them in its computation. Depending on the value of r , the output triple of C will change. We can denote by $a_1(\rho), a_2(\rho), b(\rho)$ the triple that C outputs when its coins are ρ . For any particular value of ρ it is clear from Definition 6.11 that

$$\begin{aligned} & \Pr \left[K \stackrel{R}{\leftarrow} \text{Keys}(H) : H_K(a_1(\rho)) \oplus H_K(a_2(\rho)) = b(\rho) \right] \\ & \leq \max_{a_1, a_2, b} \left\{ \Pr \left[K \stackrel{R}{\leftarrow} \text{Keys}(H) : H_K(a_1) \oplus H_K(a_2) = b \right] \right\} \\ & = \mathbf{Adv}^{\text{uh}}(H). \end{aligned}$$

Using this we get

$$\begin{aligned} \mathbf{Adv}^{\text{uh}}(H, C) &= \Pr \left[\rho \stackrel{R}{\leftarrow} \{0, 1\}^r ; K \stackrel{R}{\leftarrow} \text{Keys}(H) : H_K(a_1(\rho)) \oplus H_K(a_2(\rho)) = b(\rho) \right] \\ &= \sum_{\rho \in \{0, 1\}^r} \Pr \left[K \stackrel{R}{\leftarrow} \text{Keys}(H) : H_K(a_1(\rho)) \oplus H_K(a_2(\rho)) = b(\rho) \right] \cdot 2^{-r} \\ &\leq \sum_{\rho \in \{0, 1\}^r} \mathbf{Adv}^{\text{uh}}(H) \cdot 2^{-r} \\ &= \mathbf{Adv}^{\text{uh}}(H). \end{aligned}$$

The first equality is by definition of $\mathbf{Adv}^{\text{uh}}(H, C)$. In the second line we used the fact that the coins of C are chosen at random from the set of all strings of length r . In the third line, we used the above observation. ■ ■

How low can $\mathbf{Adv}^{\text{uh}}(H)$ go? We claim that the lowest possible value is 2^{-L} , the value achieved by a xor-universal family. The following justifies this claim.

Proposition 6.16 Let $H\text{Keys}(H) \times \text{Dom}(H) \rightarrow \{0, 1\}^L$ be a family of functions. Then

$$\mathbf{Adv}^{\text{uh}}(H) \geq 2^{-L} .$$

Proof: Fix two distinct points $a_1, a_2 \in \text{Dom}(H)$, and for any fixed key $K \in \text{Keys}(H)$ let

$$c(K) = \Pr \left[b \stackrel{R}{\leftarrow} \{0, 1\}^L : H_K(a_1) \oplus H_K(a_2) = b \right] .$$

Then $c(K) = 2^{-L}$. Why? With K, a_1, a_2 all fixed, $H_K(a_1) \oplus H_K(a_2)$ is some fixed value, call it b' . The above is then just asking what is the probability that $b = b'$ if we pick b at random, and this of course is 2^{-L} .

Now consider the adversary C that picks b at random from $\{0, 1\}^L$ and outputs the triple a_1, a_2, b . (Note this adversary is probabilistic, because of its random choice of b .) Then

$$\begin{aligned} \mathbf{Adv}^{\text{uh}}(H, C) &= \Pr \left[b \stackrel{R}{\leftarrow} \{0, 1\}^L ; K \stackrel{R}{\leftarrow} \text{Keys}(H) : H_K(a_1) \oplus H_K(a_2) = b \right] \\ &= \sum_{K \in \text{Keys}(H)} c(K) \cdot \Pr \left[K' \leftarrow \text{Keys}(H) : K' = K \right] \\ &= \sum_{K \in \text{Keys}(H)} 2^{-L} \cdot \Pr \left[K' \leftarrow \text{Keys}(H) : K' = K \right] \\ &= 2^{-L} \cdot 1 . \end{aligned}$$

Thus we have been able to present an adversary C such that $\mathbf{Adv}^{\text{uh}}(H, C) = 2^{-L}$. From Proposition 6.15 it follows that $\mathbf{Adv}^{\text{uh}}(H) \geq 2^{-L}$. ■ ■

6.13.2 The corresponding MACs

Let $H: \text{Keys}(H) \times \text{Plaintexts} \rightarrow \{0, 1\}^L$ be a family of hash functions, and let $F: \{0, 1\}^k \times \{0, 1\}^l \rightarrow \{0, 1\}^L$ be a PRF. We associate to them the *xor-universal hash based MACs*. There are two such MACs; one stateful (using counters) and deterministic, the other stateless and randomized. The key will be a pair of strings, K_1, K_2 , where the first subkey is for H and the second is for F . (We call them the *hashing* and *masking* keys respectively.) In both cases, the basic paradigm is

the same. The message is first hashed to a string x using H_{K_1} , and this value is then “encrypted” by XORing with $F_{K_2}(s)$ to yield a value τ , where s is some point chosen by the sender. The tag contains τ , but also s so as to permit verification. The difference in the two version is in how s is selected. In the counter version it is a counter, and in the randomized version a random number chosen anew with each application of the tagging algorithm.

Here now is the full description of the counter-based version of the scheme, $\mathcal{C}\text{-UHM}^{H,F} = (\mathcal{K}, \text{MAC}, \mathcal{V})$ –

Algorithm $\text{MAC}_{K_1, K_2}(M)$ $x \leftarrow H_{K_1}(M)$ $\tau \leftarrow F_{K_2}(\text{ctr}) \oplus x$ $\sigma \leftarrow (\text{ctr}, \tau)$ $\text{ctr} \leftarrow \text{ctr} + 1$ Return σ	Algorithm $\mathcal{V}_{K_1, K_2}(M, \sigma)$ Parse σ as (s, τ) $x' \leftarrow F_{K_2}(s) \oplus \tau$ $x \leftarrow H_{K_1}(M)$ If $x = x'$ then return 1 else return 0
--	--

The randomized version $\mathcal{R}\text{-UHM}^{H,F} = (\mathcal{K}, \text{MAC}, \mathcal{V})$ is like this–

Algorithm $\text{MAC}_{K_1, K_2}(M)$ $x \leftarrow H_{K_1}(M)$ $r \stackrel{R}{\leftarrow} \{0, 1\}^l$ $\tau \leftarrow F_{K_2}(r) \oplus x$ $\sigma \leftarrow (r, \tau)$ Return σ	Algorithm $\mathcal{V}_{K_1, K_2}(M, \sigma)$ Parse σ as (s, τ) $x' \leftarrow F_{K_2}(s) \oplus \tau$ $x \leftarrow H_{K_1}(M)$ If $x = x'$ then return 1 else return 0
---	--

Lemma 6.17 Let $H: \text{Keys}(H) \times \text{Plaintexts} \rightarrow \{0, 1\}^L$ be a family of functions, and A an adversary attacking the message authentication scheme $\mathcal{C}\text{-UHM}^{H, R^{l,L}}$. Then for any q, μ with $q < 2^l$ we have

$$\mathbf{Adv}^{\text{ma}}(\mathcal{C}\text{-UHM}^{H, R^{l,L}}, A) \leq \mathbf{Adv}^{\text{uh}}(H).$$

■

Proof of Lemma 6.17: The adversary A makes a sequence M_1, \dots, M_q of queries to its $\text{MAC}_{K_1, K_2}(\cdot)$ oracle, and these are answered according to the above scheme. Pictorially:

$$\begin{array}{rcl} M_1 & \implies & \sigma_1 = (s_1, \tau_1) \\ M_2 & \implies & \sigma_2 = (s_2, \tau_2) \\ \vdots & & \vdots \\ M_q & \implies & \sigma_q = (s_q, \tau_q) \end{array}$$

Here $s_i = \langle i - 1 \rangle$ is simply the (binary representation of the) counter value, and $\tau_i = f(s_i) \oplus h(M_i)$, where $h = H_{K_1}$ is the hash function instance in use, and $f = R_{K_2}^{l,L}$ is the random function specified by the second key. Following this chosen-message attack, A outputs a pair M, σ where $\sigma = (s, \tau)$. We may assume wlog that

$M \notin \{M_1, \dots, M_q\}$. We know that A will be considered successful if $\mathcal{V}_{K_1, K_2}(M, \sigma) = 1$. We wish to upper bound the probability of this event.

Let **New** be the event that $s \notin \{s_1, \dots, s_q\}$, and **Old** the complement event, namely that $s = s_i$ for some value of $i \in \{1, \dots, q\}$. Let $\Pr[\cdot]$ denote the probability of event “.” in the experiment $\text{ForgeExp}(\mathcal{C}\text{-}\mathcal{U}\mathcal{H}\mathcal{M}^{H, R^{l, L}}, A)$. We consider

$$\begin{aligned} p_1 &= \Pr[\mathcal{V}_{K_1, K_2}(M, \sigma) = 1 \mid \text{Old}] \\ p_2 &= \Pr[\mathcal{V}_{K_1, K_2}(M, \sigma) = 1 \mid \text{New}] \\ q &= \Pr[\text{New}] . \end{aligned}$$

We will use the following two claims.

Claim 1: $p_1 \leq \mathbf{Adv}^{\text{uh}}(H)$.

Claim 2: $p_2 \leq 2^{-L}$.

We will prove these claims later. Let us first check that they yield the desired result:

$$\begin{aligned} \mathbf{Adv}^{\text{ma}}(\mathcal{C}\text{-}\mathcal{U}\mathcal{H}\mathcal{M}^{H, R^{l, L}}, A) &= \Pr[\mathcal{V}_{K_1, K_2}(M, \sigma) = 1] \\ &= p_1 q + p_2(1 - q) \\ &\leq \mathbf{Adv}^{\text{uh}}(H) \cdot q + 2^{-L} \cdot (1 - q) \\ &\leq \mathbf{Adv}^{\text{uh}}(H) \cdot q + \mathbf{Adv}^{\text{uh}}(H) \cdot (1 - q) \\ &\leq \mathbf{Adv}^{\text{uh}}(H) . \end{aligned}$$

The first line is simply by definition of the success probability. The second line is obtained by conditioning. In the third line we used the claims. In the fourth line we used Proposition 6.16.

It remains to prove the claims. We begin with the second.

Proof of Claim 2: Since the queries of the adversary did not result in the function f being evaluated on the point s , the value $f(s)$ is uniformly distributed from the point of view of A . Or, remember the dynamic view of random functions; we can imagine that f gets specified only as it is queried. Since the tagging oracle (as invoked by A) has not applied f at s , we can imagine that the coins to determine $f(s)$ are tossed after the forgery is created. With that view it is clear that

$$p_2 = \Pr[f(s) \oplus h(M) = \tau] = 2^{-L} .$$

Note that here we did not use anything about the hash function; the claim is true due only to the randomness of f . \square

Proof of Claim 1:

Adversary C

```

Initialize counter  $ctr$  to 0
For  $i = 1, \dots, q$  do
     $A \rightarrow M_i$ 
     $\tau_i \xleftarrow{R} \{0, 1\}^L$ ;  $s_i \leftarrow \langle ctr \rangle$ ;  $\sigma_i \leftarrow (s_i, \tau_i)$ 
     $A \leftarrow \sigma_i$ ;  $ctr \leftarrow ctr + 1$ 
 $A \rightarrow M, \sigma$ 
Parse  $\sigma$  as  $(s, \tau)$ 
If  $s \notin \{s_1, \dots, s_q\}$  then FAIL
Else let  $i$  be such that  $s = s_i$ 
Let  $b \leftarrow \tau_i \oplus \tau$  and return  $M, M_i, b$ 

```

We claim that $\mathbf{Adv}^{\text{uh}}(H, C) = p_1$.

■

Theorem 6.18 Let $H: \text{Keys}(H) \times \text{Plaintexts} \rightarrow \{0, 1\}^L$ be a family of functions, and let $F: \{0, 1\}^k \times \{0, 1\}^l \rightarrow \{0, 1\}^L$ be a PRF. Then for any t, q, μ we have

$$\mathbf{Adv}_{\mathcal{C}\text{-}\mathcal{U}\mathcal{H}\mathcal{M}^{H,F}; t, q, \mu}^{\text{mac-frg}} \leq \mathbf{Adv}^{\text{uh}}(H) + \mathbf{Adv}_F^{\text{prf}}(t', q + 1)$$

where $t' = t + O(\mu)$. ■

6.14 Problems

Problem 6.1 Consider the following variant of the CBC MAC, intended to allow one to MAC messages of arbitrary length. The construction uses a block cipher $E: \{0, 1\}^k \times \{0, 1\}^n \rightarrow \{0, 1\}^n$, which you should assume to be secure. The domain for the MAC is $(\{0, 1\}^n)^+$. To MAC M under key K compute $\text{CBC}_K(M || |M|)$, where $|M|$ is the length of M , written in n bits. Of course K has k bits. Show that this MAC is completely insecure: break it with a constant number of queries.

Problem 6.2 Consider the following variant of the CBC MAC, intended to allow one to MAC messages of arbitrary length. The construction uses a block cipher $E: \{0, 1\}^k \times \{0, 1\}^n \rightarrow \{0, 1\}^n$, which you should assume to be secure. The domain for the MAC is $(\{0, 1\}^n)^+$. To MAC M under key (K, K') compute $\text{CBC}_K(M) \oplus K'$. Of course K has k bits and K' has n bits. Show that this MAC is completely insecure: break it with a constant number of queries.

Problem 6.3 Let $\text{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a symmetric encryption scheme and let $\text{MA} = (\mathcal{K}', \text{MAC}, \text{VF})$ be a message authentication code. Alice (A) and Bob (B) share a secret key $K = (K1, K2)$ where $K1 \leftarrow \mathcal{K}$ and $K2 \leftarrow \mathcal{K}'$. Alice wants to send messages to Bob in a private and authenticated way. Consider her sending each of the following as a means to this end. For each, say whether it is a secure way or not, and briefly justify your answer. (In the cases where the method is good, you don't have to give a proof, just the intuition.)

- (a) $M, \text{MAC}_{K_2}(\mathcal{E}_{K_1}(M))$
- (b) $\mathcal{E}_{K_1}(M, \text{MAC}_{K_2}(M))$
- (c) $\text{MAC}_{K_2}(\mathcal{E}_{K_1}(M))$
- (d) $\mathcal{E}_{K_1}(M), \text{MAC}_{K_2}(M)$
- (e) $\mathcal{E}_{K_1}(M), \mathcal{E}_{K_1}(\text{MAC}_{K_2}(M))$
- (f) $C, \text{MAC}_{K_2}(C)$ where $C = \mathcal{E}_{K_1}(M)$
- (g) $\mathcal{E}_{K_1}(M, A)$ where A encodes the identity of Alice; B decrypts the received ciphertext C and checks that the second half of the plaintext is “ A ”.

In analyzing these schemes, you should assume that the primitives have the properties guaranteed by their definitions, but no more; for an option to be good it must work for *any* choice of a secure encryption scheme and a secure message authentication scheme.

Now, out of all the ways you deemed secure, suppose you had to choose one to implement for a network security application. Taking performance issues into account, do all the schemes look pretty much the same, or is there one you would prefer?

Problem 6.4 Refer to problem 4.3. Given a block cipher $E : \mathcal{K} \times \{0, 1\}^n \rightarrow \{0, 1\}^n$, construct a cipher (a “deterministic encryption scheme”) with message space $\{0, 1\}^*$ that is secure in the sense that you defined. (*Hint*: you now know how to construct from E a pseudorandom function with domain $\{0, 1\}^*$.)

6.15 References and Related Work

Chapter 7

AUTHENTICATED ENCRYPTION

Chapter 8

NUMBER-THEORETIC BACKGROUND

8.1 The basic groups

We let $\mathbf{Z} = \{\dots, -2, -1, 0, 1, 2, \dots\}$ denote the set of integers. We let $\mathbf{Z}_+ = \{1, 2, \dots\}$ denote the set of positive integers and $\mathbf{N} = \{0, 1, 2, \dots\}$ the set of non-negative integers.

8.1.1 Integers mod N

If a, b are integers, not both zero, then their greatest common divisor, denoted $\gcd(a, b)$, is the largest integer d such that d divides a and d divides b . If $\gcd(a, b) = 1$ then we say that a and b are relatively prime. If a, N are integers with $N > 0$ then there are unique integers r, q such that $a = Nq + r$ and $0 \leq r < N$. We call r the remainder upon division of a by N , and denote it by $a \bmod N$. We note that the operation $a \bmod N$ is defined for both negative and non-negative values of a , but only for positive values of N . (When a is negative, the quotient q will also be negative, but the remainder r must always be in the indicated range $0 \leq r < N$.) If a, b are any integers and N is a positive integer, we write $a \equiv b \pmod{N}$ if $a \bmod N = b \bmod N$. We associate to any positive integer N the following two sets:

$$\begin{aligned}\mathbf{Z}_N &= \{0, 1, \dots, N-1\} \\ \mathbf{Z}_N^* &= \{i \in \mathbf{Z} : 1 \leq i \leq N-1 \text{ and } \gcd(i, N) = 1\}\end{aligned}$$

The first set is called the set of integers mod N . Its size is N , and it contains exactly the integers that are possible values of $a \bmod N$ as a ranges over \mathbf{Z} . We define the Euler Phi (or totient) function $\varphi: \mathbf{Z}_+ \rightarrow \mathbf{N}$ by $\varphi(N) = |\mathbf{Z}_N^*|$ for all $N \in \mathbf{Z}_+$. That is, $\varphi(N)$ is the size of the set \mathbf{Z}_N^* .

8.1.2 Groups

Let G be a non-empty set, and let \cdot be a binary operation on G . This means that for every two points $a, b \in G$, a value $a \cdot b$ is defined.

Definition 8.1 Let G be a non-empty set and let \cdot denote a binary operation on G . We say that G is a *group* if it has the following properties:

1. **CLOSURE:** For every $a, b \in G$ it is the case that $a \cdot b$ is also in G .
2. **ASSOCIATIVITY:** For every $a, b, c \in G$ it is the case that $(a \cdot b) \cdot c = a \cdot (b \cdot c)$.
3. **IDENTITY:** There exists an element $\mathbf{1} \in G$ such that $a \cdot \mathbf{1} = \mathbf{1} \cdot a = a$ for all $a \in G$.
4. **INVERTIBILITY:** For every $a \in G$ there exists a unique $b \in G$ such that $a \cdot b = b \cdot a = \mathbf{1}$.

The element b in the invertibility condition is referred to as the inverse of the element a , and is denoted a^{-1} . ■

In any group, we can define an exponentiation operation which associates to any $a \in G$ and any integer i a group element we denote a^i , defined as follows. If $i = 0$ then a^i is defined to be $\mathbf{1}$, the identity element of the group. If $i > 0$ then

$$a^i = \underbrace{a \cdot a \cdots a}_i .$$

If i is negative, then we define $a^i = (a^{-1})^{-i}$. Put another way, let $j = -i$, which is positive, and set

$$a^i = \underbrace{a^{-1} \cdot a^{-1} \cdots a^{-1}}_j .$$

With these definitions in place, we can manipulate exponents in the way in which we are accustomed with ordinary numbers. Namely, identities such as the following hold for all $a \in G$ and all $i, j \in \mathbf{Z}$:

$$\begin{aligned} a^{i+j} &= a^i \cdot a^j \\ (a^i)^j &= a^{ij} \\ a^{-i} &= (a^i)^{-1} \\ a^{-i} &= (a^{-1})^i . \end{aligned}$$

We will use this type of manipulation frequently without explicit explanation.

It is customary in group theory to call the size of a group G its *order*. That is, the order of a group G is $|G|$, the number of elements in it. We will often make use of the following basic fact. It says that if any group element is raised to the power the order of the group, the result is the identity element of the group.

Fact 8.2 Let G be a group and let $m = |G|$ be its order. Then $a^m = \mathbf{1}$ for all $a \in G$. ■

This means that computation in the group indices can be done modulo m . That is, for all $a \in G$ and all $i \in \mathbf{Z}$ we have

$$a^i = a^{i \bmod m}$$

where $m = |G|$ and the mod operation is defined for all $i \in \mathbf{Z}$ as above. (A reader may want to make sure they see why Fact 8.2 implies this.)

If G is a group, a set $S \subseteq G$ is called a subgroup if it is a group in its own right, under the same operation as that under which G is a group. If we already know that G is a group, there is a simple way to test whether S is a subgroup: it is one if and only if $x \cdot y^{-1} \in S$ for all $x, y \in S$. Here y^{-1} is the inverse of y in G .

Fact 8.3 Let G be a group and let S be a subgroup of G . Then the order of S divides the order of G . ■

We now return to the sets we defined above and remark on their group structure. Let N be a positive integer. The operation of addition modulo N takes input any two integers a, b and returns $(a + b) \bmod N$. The operation of multiplication modulo N takes input any two integers a, b and returns $ab \bmod N$.

Fact 8.4 Let N be a positive integer. Then \mathbf{Z}_N is a group under addition modulo N , and \mathbf{Z}_N^* is a group under multiplication modulo N . ■

In \mathbf{Z}_N , the identity element is 0 and the inverse of a is $-a \bmod N = N - a$. In \mathbf{Z}_N^* , the identity element is 1 and the inverse of a is a $b \in \mathbf{Z}_N^*$ such that $ab \equiv 1 \pmod{N}$. It may not be obvious why such a b even exists, but it does. We do not prove the above fact here.

8.2 Algorithms

Figure 8.1 summarizes some basic algorithms involving numbers. These algorithms are used to implement public-key cryptosystems, and thus their running time is an important concern. We begin with a discussion about the manner in which running time is measured, and then go on to discuss the algorithms, some very briefly, some in more depth.

8.2.1 Bit operations and binary length

In a course or text on algorithms, we learn to analyze the running time of an algorithm as a function of the size of its input. The inputs are typically things like graphs, or arrays, and the measure of input size might be the number of nodes in the graph or the length of the array. Within the algorithm we often need to perform arithmetic operations, like addition or multiplication of array indices. We typically assume these have $O(1)$ cost. The reason this assumption is reasonable is that the numbers in question are small and the cost of manipulating them is negligible

Algorithm	Input	Output
INT-DIV	a, N ($N > 0$)	(q, r) with $a = Nq + r$
MOD	a, N ($N > 0$)	$a \bmod N$
EXT-GCD	a, b ($(a, b) \neq (0, 0)$)	(d, \bar{a}, \bar{b}) with $d = \gcd(a, b)$
MOD-ADD	a, b, N ($a, b \in \mathbf{Z}_N$)	$(a + b) \bmod N$
MOD-MULT	a, b, N ($a, b \in \mathbf{Z}_N$)	$ab \bmod N$
MOD-INV	a, N ($a \in \mathbf{Z}_N^*$)	$b \in \mathbf{Z}_N^*$ with $ab \equiv 1 \pmod{N}$
MOD-EXP	a, n, N ($a \in \mathbf{Z}_N$)	$a^n \bmod N$
EXP_G	a, n ($a \in G$)	$a^n \in G$

Figure 8.1: **Some basic algorithms and their running time.** Unless otherwise indicated, an input value is an integer and the running time is the number of bit operations. G denotes a group.

compared to costs proportional to the size of the array or graph on which we are working.

In contrast, the numbers arising in cryptographic algorithms are large, having magnitudes like 2^{512} or 2^{1024} . The arithmetic operations on these numbers are the main cost of the algorithm, and the costs grow as the numbers get bigger.

The numbers are provided to the algorithm in binary, and the size of the input number is thus the number of bits in its binary representation. We call this the length, or binary length, of the number, and we measure the running time of the algorithm as a function of the binary lengths of its input numbers. In computing the running time, we count the number of bit operations performed.

Let $b_{k-1} \dots b_1 b_0$ be the binary representation of a positive integer a , meaning b_0, \dots, b_{k-1} are bits such that $b_{k-1} = 1$ and $a = 2^{k-1}b_{k-1} + 2^{k-2}b_{k-2} + \dots + 2^1b_1 + 2^0b_0$. Then the binary length of a is k , and is denoted $|a|$. Notice that $|a| = k$ if and only if $2^{k-1} \leq a < 2^k$. If a is negative, we let $|a| = |-a|$, and assume that an additional bit or two is used to indicate to the algorithm that the input is negative.

8.2.2 Integer division and mod algorithms

We define the integer division function as taking input two integers a, N , with $N > 0$, and returning the quotient and remainder obtained by dividing a by N . That is, the function returns (q, r) such that $a = qN + r$ with $0 \leq r < N$. We denote by INT-DIV an algorithm implementing this function. The algorithm uses the standard division method we learned way back in school, which turns out to run in time proportional to the product of the binary lengths of a and N .

We also want an algorithm that implements the mod function, taking integer inputs a, N with $N > 0$ and returning $a \bmod N$. This algorithm, denoted MOD, can be implemented simply by calling INT-DIV(a, N) to get (q, r) , and then returning just the remainder r .

8.2.3 Extended GCD algorithm

Suppose a, b are integers, not both 0. A basic fact about the greatest common divisor of a and b is that it is the smallest positive element of the set

$$\{ a\bar{a} + b\bar{b} : \bar{a}, \bar{b} \in \mathbf{Z} \}$$

of all integer linear combinations of a and b . In particular, if $d = \gcd(a, b)$ then there exist integers \bar{a}, \bar{b} such that $d = a\bar{a} + b\bar{b}$. (Note that either \bar{a} or \bar{b} could be negative.)

Example 8.5 The gcd of 20 and 12 is $d = \gcd(20, 12) = 4$. We note that $4 = 20(2) + (12)(-3)$, so in this case $\bar{a} = 2$ and $\bar{b} = -3$. ■

Besides the gcd itself, we will find it useful to be able to compute these weights \bar{a}, \bar{b} . This is what the extended-gcd algorithm EXT-GCD does: given a, b as input, it returns (d, \bar{a}, \bar{b}) such that $d = \gcd(a, b) = a\bar{a} + b\bar{b}$. The algorithm itself is an extension

of Euclid's classic algorithm for computing the gcd, and the simplest description is a recursive one. We now provide it, and then discuss the correctness and running time. The algorithm takes input any integers a, b , not both zero.

```

Algorithm EXT-GCD( $a, b$ )
If  $b = 0$  then return  $(a, 1, 0)$ 
Else
   $(q, r) \leftarrow \text{INT-DIV}(a, b)$ 
   $(d, x, y) \leftarrow \text{EXT-GCD}(b, r)$ 
   $\bar{a} \leftarrow y$ 
   $\bar{b} \leftarrow x - qy$ 
  Return  $(d, \bar{a}, \bar{b})$ 
EndIf

```

The base case is when either $b = 0$. If $b = 0$ then we know by assumption that $a \neq 0$, so $\text{gcd}(a, b) = a$, and since $a = a(1) + b(0)$, the weights are 1 and 0. If $b \neq 0$ then we can divide by it, and we divide a by it to get a quotient q and remainder r . For the recursion, we use the fact that $\text{gcd}(a, b) = \text{gcd}(b, r)$. The recursive call thus yields $d = \text{gcd}(a, b)$ together with weights x, y such that $d = bx + ry$. Noting that $a = bq + r$ we have

$$d = bx + ry = bx + (a - bq)y = ay + b(x - qy) = a\bar{a} + b\bar{b},$$

confirming that the values assigned to \bar{a}, \bar{b} are correct.

The running time of this algorithm is $O(|a| \cdot |b|)$, or, put a little more simply, the running time is quadratic in the length of the longer number. This is not so obvious, and proving it takes some work. We do not provide this proof here.

8.2.4 Algorithms for modular addition and multiplication

The next two algorithms in Figure 8.1 are the ones for modular addition and multiplication. To compute $(a + b) \bmod N$, we first compute $c = a + b$ using the usual algorithm we learned way back in school, which runs in time linear in the binary representations of the numbers. We might imagine that it now takes quadratic time to do the mod operation, but in fact if $c > N$, the mod operation can be simply executed by subtracting N from c , which takes only linear time, which is why the algorithm as a whole takes linear time. For multiplication mod N , the process is much the same. First compute $c = ab$ using the usual algorithm, which is quadratic time. This time we do the mod by invoking $\text{MOD}(c, N)$. (The length of c is the sum of the lengths of a and b , and so c is not small as in the addition case, so a shortcut to the mod as we saw there does not seem possible.)

8.2.5 Algorithm for modular inverse

The next algorithm in Figure 8.1 is for computation of the multiplicative inverse of a in the group \mathbf{Z}_N^* . Namely, on input $N > 0$ and $a \in \mathbf{Z}_N^*$, algorithm MOD-INV

returns b such that $ab \equiv 1 \pmod{N}$. The method is quite simple:

```

Algorithm MOD-INV( $a, N$ )
( $d, \bar{a}, \bar{N}$ )  $\leftarrow$  EXT-GCD( $a, N$ )
 $b \leftarrow \bar{a} \bmod N$ 
Return  $b$ 

```

The cost is $O(|a| \cdot |N|)$ because this is the cost of the invoked algorithms. Now let us see why the algorithm is correct. Since $a \in \mathbf{Z}_N^*$ we know that $\gcd(a, N) = 1$. The EXT-GCD algorithm thus guarantees that $d = 1$ and $1 = a\bar{a} + N\bar{N}$. Since $N \bmod N = 0$, we have $1 \equiv a\bar{a} \pmod{N}$, and thus $b = \bar{a} \bmod N$ is the right value to return.

8.2.6 Exponentiation algorithm

We will be using exponentiation in various different groups, so it is useful to look at it at the group level. Let G be a group and let $a \in G$. Given an integer $n \in \mathbf{Z}$ we want to compute the group element a^n as defined in Section 8.1.2. The naive method, assuming for simplicity $n \geq 0$, is to execute

```

 $y \leftarrow \mathbf{1}$ 
For  $i = 1, \dots, n$  do  $y \leftarrow y \cdot a$  EndFor
Return  $y$ 

```

This might at first seem like a satisfactory algorithm, but actually it is very slow. The number of group operations required is n , and the latter can be as large as the order of the group. Since we are often looking at groups containing about 2^{512} elements, exponentiation by this method is not feasible. In the language of complexity theory, the problem is that we are looking at an exponential time algorithm. This is because the running time is exponential in the binary length $|n|$ of the input n . So we seek a better algorithm. We illustrate the idea of fast exponentiation with an example.

Example 8.6 Suppose the binary length of n is 5, meaning the binary representation of n has the form $b_4b_3b_2b_1b_0$. Then

$$\begin{aligned} n &= 2^4b_4 + 2^3b_3 + 2^2b_2 + 2^1b_1 + 2^0b_0 \\ &= 16b_4 + 8b_3 + 4b_2 + 2b_1 + b_0 . \end{aligned}$$

Our exponentiation algorithm will proceed to compute the values $y_5, y_4, y_3, y_2, y_1, y_0$

in turn, as follows:

$$\begin{aligned}
 y_5 &= \mathbf{1} \\
 y_4 &= y_5^2 \cdot a^{b_4} = a^{b_4} \\
 y_3 &= y_4^2 \cdot a^{b_3} = a^{2b_4+b_3} \\
 y_2 &= y_3^2 \cdot a^{b_2} = a^{4b_4+2b_3+b_2} \\
 y_1 &= y_2^2 \cdot a^{b_1} = a^{8b_4+4b_3+2b_2+b_1} \\
 y_0 &= y_1^2 \cdot a^{b_0} = a^{16b_4+8b_3+4b_2+2b_1+b_0} .
 \end{aligned}$$

Two group operations are required to compute y_i from y_{i+1} , and the number of steps equals the binary length of n , so the algorithm is fast. ■

In general, we let $b_{k-1} \dots b_1 b_0$ be the binary representation of n , meaning b_0, \dots, b_{k-1} are bits such that $n = 2^{k-1}b_{k-1} + 2^{k-2}b_{k-2} + \dots + 2^1b_1 + 2^0b_0$. The algorithm proceeds as follows given any input $a \in G$ and $n \in \mathbf{Z}$:

```

Algorithm EXPG( $a, n$ )
If  $n < 0$  then  $a \leftarrow a^{-1}$  and  $n \leftarrow -n$  EndIf
Let  $b_{k-1} \dots b_1 b_0$  be the binary representation of  $n$ 
 $y \leftarrow \mathbf{1}$ 
For  $i = k - 1$  downto 0 do
     $y \leftarrow y^2 \cdot a^{b_i}$ 
End For
Output  $y$ 

```

The algorithm uses two group operations per iteration of the loop: one to multiply y by itself, another to multiply the result by a^{b_i} . (The computation of a^{b_i} is without cost, since this is just a if $b_i = 1$ and $\mathbf{1}$ if $b_i = 0$.) So its total cost is $2k = 2|n|$ group operations. (We are ignoring the cost of the one possible inversion in the case $n < 0$.) (This is the worst case cost. We observe that it actually takes $|n| + W_H(n)$ group operations, where $W_H(n)$ is the number of ones in the binary representation of n .)

We will typically use this algorithm when the group G is \mathbf{Z}_N^* and the group operation is multiplication modulo N , for some positive integer N . We have denoted this algorithm by MOD-EXP in Figure 8.1. (The input a is not required to be relatively prime to N even though it usually will be, so is listed as coming from \mathbf{Z}_N .) In that case, each group operation is implemented via MOD-MULT and takes $O(|N|^2)$ time, so the running time of the algorithm is $O(|n| \cdot |N|^2)$. Since n is usually in \mathbf{Z}_N , this comes to $O(|N|^3)$. The salient fact to remember is that modular exponentiation is a cubic time algorithm.

8.3 Cyclic groups and generators

Let G be a group, let $\mathbf{1}$ denote its identity element, and let $m = |G|$ be the order of G . If $g \in G$ is any member of the group, the *order* of g is defined to be the least positive integer n such that $g^n = \mathbf{1}$. We let

$$\langle g \rangle = \{ g^i : i \in \mathbf{Z}_n \} = \{ g^0, g^1, \dots, g^{n-1} \}$$

denote the set of group elements generated by g . A fact we do not prove, but is easy to verify, is that this set is a subgroup of G . The order of this subgroup (which, by definition, is its size) is just the order of g . Fact 8.3 tells us that the order n of g divides the order m of the group. An element g of the group is called a *generator* of G if $\langle g \rangle = G$, or, equivalently, if its order is m . If g is a generator of G then for every $a \in G$ there is a unique integer $i \in \mathbf{Z}_m$ such that $g^i = a$. This i is called the discrete logarithm of a to base g , and we denote it by $\text{DLog}_{G,g}(a)$. Thus, $\text{DLog}_{G,g}(\cdot)$ is a function that maps G to \mathbf{Z}_m , and moreover this function is a bijection, meaning one-to-one and onto. The function of \mathbf{Z}_m to G defined by $i \mapsto g^i$ is called the discrete exponentiation function, and the discrete logarithm function is the inverse of the discrete exponentiation function.

Example 8.7 Let $p = 11$, which is prime. Then $Z_{11}^* = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$ has order $p - 1 = 10$. Let us find the subgroups generated by group elements 2 and 5. We raise them to the powers $i = 0, \dots, 9$. We get:

i	0	1	2	3	4	5	6	7	8	9
$2^i \text{ mod } 11$	1	2	4	8	5	10	9	7	3	6
$5^i \text{ mod } 11$	1	5	3	4	9	1	5	3	4	9

Looking at which elements appear in the row corresponding to 2 and 5, respectively, we can determine the subgroups these group elements generate:

$$\begin{aligned} \langle 2 \rangle &= \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\} \\ \langle 5 \rangle &= \{1, 3, 4, 5, 9\} . \end{aligned}$$

Since $\langle 2 \rangle$ equals \mathbf{Z}_{11}^* , the element 2 is a generator. Since a generator exists, \mathbf{Z}_{11}^* is cyclic. On the other hand, $\langle 5 \rangle \neq \mathbf{Z}_{11}^*$, so 5 is not a generator. The order of 2 is 10, while the order of 5 is 5. Note that these orders divide the order 10 of the group. The table also enables us to determine the discrete logarithms to base 2 of the different group elements:

a	1	2	3	4	5	6	7	8	9	10
$\text{DLog}_{\mathbf{Z}_{11}^*,2}(a)$	0	1	8	2	4	9	7	3	6	5

Later we will see a way of identifying all the generators given that we know one of them. ■

The discrete exponentiation function is conjectured to be one-way (meaning the discrete logarithm function is hard to compute) for some cyclic groups G . Due to this fact we often seek cyclic groups for cryptographic usage. Here are three sources of such groups. We will not prove any of the facts below; their proofs can be found in books on algebra.

Fact 8.8 Let p be a prime. Then the group \mathbf{Z}_p^* is cyclic. ■

The operation here is multiplication modulo p , and the size of this group is $\varphi(p) = p - 1$. This is the most common choice of group in cryptography.

Fact 8.9 Let G be a group and let $m = |G|$ be its order. If m is a prime number, then G is cyclic. ■

In other words, any group having a prime number of elements is cyclic. Note that it is not for this reason that Fact 8.8 is true, since the order of \mathbf{Z}_p^* (where p is prime) is $p - 1$, which is even if $p \geq 3$ and 1 if $p = 2$, and is thus never a prime number.

The following is worth knowing if you have some acquaintance with finite fields. Recall that a such a field is a set F equipped with two operations, an addition and a multiplication. The identity element of the addition is denoted 0. When this is removed from the field, what remains is a group under multiplication. This group is always cyclic.

Fact 8.10 Let F be a finite field, and let $F^* = F - \{0\}$. Then F^* is a cyclic group under the multiplication operation of F . ■

A finite field of order m exists if and only if $m = p^n$ for some prime p and integer $n \geq 1$. The finite field of order p is exactly \mathbf{Z}_p , so the case $n = 1$ of Fact 8.10 implies Fact 8.8. Another interesting special case of Fact 8.10 is when the order of the field is 2^n , meaning $p = 2$, yielding a cyclic group of order $2^n - 1$.

When we want to use a cyclic group G in cryptography, we will often want to find a generator for it. The process used is to pick group elements in some appropriate way, and then test each chosen element to see whether it is a generator. One thus has to solve two problems. One is how to test whether a given group element is a generator, and the other is what process to use to choose the candidate generators to be tested.

Let $m = |G|$ and let $\mathbf{1}$ be the identity element of G . The obvious way to test whether a given $g \in G$ is a generator is to compute the values g^1, g^2, g^3, \dots , stopping at the first j such that $g^j = \mathbf{1}$. If $j = m$ then g is a generator. This test however can require up to m group operations, which is not efficient, given that the groups of interest are large, so we need better tests.

The obvious way to choose candidate generators is to cycle through the entire group in some way, testing each element in turn. Even with a fast test, this can take a long time, since the group is large. So we would also like better ways of picking candidates.

We address these problems in turn. Let us first look at testing whether a given $g \in G$ is a generator. One sees quickly that computing all powers of g as in g^1, g^2, g^3, \dots is not necessary. For example if we computed g^8 and found that this is not $\mathbf{1}$, then we know that $g^4 \neq \mathbf{1}$ and $g^2 \neq \mathbf{1}$ and $g \neq \mathbf{1}$. More generally, if we know that $g^j \neq \mathbf{1}$ then we know that $g^i \neq \mathbf{1}$ for all i dividing j . This tells us that it is better to first compute high powers of g , and use that to cut down the space of exponents that need further testing. The following Proposition pinpoints the optimal way to do this. It identifies a set of exponents m_1, \dots, m_n such that one need only test whether $g^{m_i} \neq \mathbf{1}$ for $i = 1, \dots, n$. As we will argue later, this set is quite small.

Proposition 8.11 Let G be a cyclic group and let $m = |G|$ be the size of G . Let $p_1^{\alpha_1} \cdots p_n^{\alpha_n}$ be the prime factorization of m and let $m_i = m/p_i$ for $i = 1, \dots, n$. Let $g \in G$. Then g is a generator of G if and only if

$$\text{For all } i = 1, \dots, n: \quad g^{m_i} \neq \mathbf{1}, \quad (8.1)$$

where $\mathbf{1}$ is the identity element of G . ■

Proof of Proposition 8.11: First suppose that g is a generator of G . Then we know that the smallest positive integer j such that $g^j = \mathbf{1}$ is $j = m$. Since $0 < m_i < m$, it must be that $g^{m_i} \neq \mathbf{1}$ for all $i = 1, \dots, n$.

Conversely, suppose g satisfies the condition of Equation (8.1). We want to show that g is a generator. Let j be the order of g , meaning the smallest positive integer such that $g^j = \mathbf{1}$. Then we know that j must divide the order m of the group, meaning $m = dj$ for some integer $d \geq 1$. This implies that $j = p_1^{\beta_1} \cdots p_n^{\beta_n}$ for some integers β_1, \dots, β_n satisfying $0 \leq \beta_i \leq \alpha_i$ for all $i = 1, \dots, n$. If $j < m$ then there must be some i such that $\beta_i < \alpha_i$, and in that case j divides m_i , which in turn implies $g^{m_i} = \mathbf{1}$ (because $g^j = \mathbf{1}$). So the assumption that Equation (8.1) is true implies that j cannot be strictly less than m , so the only possibility is $j = m$, meaning g is a generator. ■

The number n of terms in the prime factorization of m cannot be more than $\lg(m)$, the binary logarithm of m . (This is because $p_i \geq 2$ and $\alpha_i \geq 1$ for all $i = 1, \dots, n$.) So, for example, if the group has size about 2^{512} , then at most 512 tests are needed. So testing is quite efficient. One should note however that it requires knowing the prime factorization of m .

Let us now consider the second problem we discussed above, namely how to choose candidate group elements for testing. There seems little reason to think that trying all group elements in turn will yield a generator in a reasonable amount of time. Instead, we consider picking group elements at random, and then testing them. The probability of success in any trial is $|\text{Gen}(G)|/|G|$. So the expected number of trials before we find a generator is $|G|/|\text{Gen}(G)|$. To estimate the efficacy of this method, we thus need to know the number of generators in the group. The

following Proposition gives a characterization of the generator set which in turn tells us its size.

Proposition 8.12 Let G be a cyclic group and let g be a generator of G . Then

$$\text{Gen}(G) = \{ g^i \in G : i \in \mathbf{Z}_m^* \},$$

and the number of generators of G is

$$|\text{Gen}(G)| = \varphi(m),$$

where $m = |G|$ is the size of G . ■

Proof of Proposition 8.12: The second equation follows immediately from the first:

$$|\text{Gen}(G)| = \left| \{ g^i \in G : i \in \mathbf{Z}_m^* \} \right| = |\mathbf{Z}_m^*| = \varphi(m).$$

We now prove the first equation. First, we show that if $i \in \mathbf{Z}_m^*$ then $g^i \in \text{Gen}(G)$. Second, we show that if $i \in \mathbf{Z}_m - \mathbf{Z}_m^*$ then $g^i \notin \text{Gen}(G)$.

So first suppose $i \in \mathbf{Z}_m^*$, and let $h = g^i$. We want to show that h is a generator of G . It suffices to show that the only possible value of $j \in \mathbf{Z}_m$ such that $h^j = \mathbf{1}$ is $j = 0$, so let us now show this. Let $j \in \mathbf{Z}_m$ be such that $h^j = \mathbf{1}$. Since $h = g^i$ we have

$$\mathbf{1} = h^j = g^{ij \bmod m}.$$

Since g is a generator, it must be that $ij \equiv 0 \pmod{m}$, meaning m divides ij . But $i \in \mathbf{Z}_m^*$ so $\gcd(i, m) = 1$. So it must be that m divides j . But $j \in \mathbf{Z}_m$ and the only member of this set divisible by m is 0, so $j = 0$ as desired.

Next, suppose $i \in \mathbf{Z}_m - \mathbf{Z}_m^*$ and let $h = g^i$. To show that h is not a generator it suffices to show that there is some non-zero $j \in \mathbf{Z}_m$ such that $h^j = \mathbf{1}$. Let $d = \gcd(i, m)$. Our assumption $i \in \mathbf{Z}_m - \mathbf{Z}_m^*$ implies that $d > 1$. Let $j = m/d$, which is a non-zero integer in \mathbf{Z}_m because $d > 1$. Then the following shows that $h^j = \mathbf{1}$, completing the proof:

$$h^j = g^{ij} = g^{i \cdot m/d} = g^{m \cdot i/d} = (g^m)^{i/d} = \mathbf{1}^{i/d} = \mathbf{1}.$$

We used here the fact that d divides i and that $g^m = \mathbf{1}$. ■

Example 8.13 Let us determine all the generators of the group \mathbf{Z}_{11}^* . Let us first use Proposition 8.11. The size of \mathbf{Z}_{11}^* is $m = \varphi(11) = 10$, and the prime factorization of 10 is $2^1 \cdot 5^1$. Thus, the test for whether a given $a \in \mathbf{Z}_{11}^*$ is a generator is that $a^2 \not\equiv 1 \pmod{11}$ and $a^5 \not\equiv 1 \pmod{11}$. Let us compute $a^2 \bmod 11$ and $a^5 \bmod 11$ for all group elements a . We get:

a	1	2	3	4	5	6	7	8	9	10
$a^2 \bmod 11$	1	4	9	5	3	3	5	9	4	1
$a^5 \bmod 11$	1	10	1	1	1	10	10	10	1	10

The generators are those a for which the corresponding column has a no entry equal to 1, meaning in both rows, the entry for this column is different from 1. So

$$\text{Gen}(\mathbf{Z}_{11}^*) = \{2, 6, 7, 8\}.$$

Now, let us use Proposition 8.12 and double-check that we get the same thing. We saw in Example 8.7 that 2 was a generator of \mathbf{Z}_{11}^* . As per Proposition 8.12, the set of generators is

$$\text{Gen}(\mathbf{Z}_{11}^*) = \{2^i \bmod 11 : i \in \mathbf{Z}_{10}^*\}.$$

This is because the size of the group is $m = 10$. Now, $\mathbf{Z}_{10}^* = \{1, 3, 7, 9\}$. The values of $2^i \bmod 11$ as i ranges over this set can be obtained from the table in Example 8.7 where we computed all the powers of 2. So

$$\begin{aligned} \{2^i \bmod 11 : i \in \mathbf{Z}_{10}^*\} &= \{2^1 \bmod 11, 2^3 \bmod 11, 2^7 \bmod 11, 2^9 \bmod 11\} \\ &= \{2, 6, 7, 8\}. \end{aligned}$$

This is the same set we obtained above via Proposition 8.11.

If we try to find a generator by picking group elements at random and then testing using Proposition 8.11, each trial has probability of success $\varphi(10)/10 = 4/10$, so we would expect to find a generator in $10/4$ trials. We can optimize slightly by noting that 1 and -1 can never be generators, and thus we only need pick candidates randomly from $\mathbf{Z}_{11}^* - \{1, 10\}$. In that case, each trial has probability of success $\varphi(10)/8 = 4/8 = 1/2$, so we would expect to find a generator in 2 trials. ■

When we want to work in a cyclic group in cryptography, the most common choice is to work over \mathbf{Z}_p^* for a suitable prime p . The algorithm for finding a generator would be to repeat the process of picking a random group element and testing it, halting when a generator is found. In order to make this possible we choose p in such a way that the prime factorization of the order $p-1$ of \mathbf{Z}_p^* is known. In order to make the testing fast, we choose p so that $p-1$ has few prime factors. Accordingly, it is common to choose p to equal $2q+1$ for some prime q . In this case, the prime factorization of $p-1$ is 2^1q^1 , so we need raise a candidate to only two powers to test whether or not it is a generator. In choosing candidates, we optimize slightly by noting that 1 and -1 are never generators, and accordingly pick the candidates from $\mathbf{Z}_p^* - \{1, p-1\}$ rather than from \mathbf{Z}_p^* . So the algorithm is as follows:

Algorithm FIND-GEN(p)

$q \leftarrow (p-1)/2$

found $\leftarrow 0$

While (found $\neq 1$) do

$g \xleftarrow{R} \mathbf{Z}_p^* - \{1, p-1\}$

If $(g^2 \bmod p \neq 1)$ and $(g^q \bmod p \neq 1)$ then found $\leftarrow 1$

EndWhile

Return g

Proposition 8.11 tells us that the group element g returned by this algorithm is always a generator of \mathbf{Z}_p^* . By Proposition 8.12, the probability that an iteration of the algorithm is successful in finding a generator is

$$\frac{|\text{Gen}(\mathbf{Z}_p^*)|}{|\mathbf{Z}_p^*| - 2} = \frac{\varphi(p-1)}{p-3} = \frac{\varphi(2q)}{2q-2} = \frac{q-1}{2q-2} = \frac{1}{2}.$$

Thus the expected number of iterations of the while loop is 2.

8.4 Squares and non-squares

An element a of a group G is called a *square*, or *quadratic residue* if it has a square root, meaning there is some $b \in G$ such that $b^2 = a$ in G . We let

$$\text{QR}(G) = \{g \in G : g \text{ is quadratic residue in } G\}$$

denote the set of all squares in the group G . We leave to the reader to check that this set is a subgroup of G .

We are mostly interested in the case where the group G is \mathbf{Z}_N^* for some integer N . An integer a is called a *square mod N* or *quadratic residue mod N* if $a \bmod N$ is a member of $\text{QR}(\mathbf{Z}_N^*)$. If $b^2 \equiv a \pmod{N}$ then b is called a square-root of a mod N . An integer a is called a *non-square mod N* or *quadratic non-residue mod N* if $a \bmod N$ is a member of $\mathbf{Z}_p^* - \text{QR}(\mathbf{Z}_N^*)$. We will begin by looking at the case where $N = p$ is a prime. In this case we define a function $J_p: \mathbf{Z}_p^* \rightarrow \{-1, 1\}$ by

$$J_p(a) = \begin{cases} 1 & \text{if } a \text{ is a square mod } p \\ -1 & \text{otherwise.} \end{cases}$$

for all $a \in \mathbf{Z}_p^*$. We call $J_p(a)$ the *Legendre symbol* of a . Thus, the Legendre symbol is simply a compact notation for telling us whether or not its argument is a square modulo p .

Before we move to developing the theory, it may be useful to look at an example.

Example 8.14 Let $p = 11$, which is prime. Then $\mathbf{Z}_{11}^* = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$ has order $p-1 = 10$. A simple way to determine $\text{QR}(\mathbf{Z}_{11}^*)$ is to square all the group elements in turn:

a	1	2	3	4	5	6	7	8	9	10
$a^2 \bmod 11$	1	4	9	5	3	3	5	9	4	1

The squares are exactly those elements that appear in the second row, so

$$\text{QR}(\mathbf{Z}_{11}^*) = \{1, 3, 4, 5, 9\}.$$

The number of squares is 5, which we notice equals $(p-1)/2$. This is not a coincidence, as we will see. Also notice that each square has exactly two different square roots. (The square roots of 1 are 1 and 10; the square roots of 3 are 5 and 6; the square roots of 4 are 2 and 9; the square roots of 5 are 4 and 7; the square roots of 9 are 3 and 8.)

Since 11 is prime, we know that \mathbf{Z}_{11}^* is cyclic, and as we saw in Example 8.7, 2 is a generator. (As a side remark, we note that a generator must be a non-square. Indeed, if $a = b^2$ is a square, then $a^5 = b^{10} = 1$ modulo 11 because 10 is the order of the group. So $a^j = 1$ modulo 11 for some positive $j < 10$, which means a is not a generator. However, not all non-squares need be generators.) Below, we reproduce from that example the table of discrete logarithms of the group elements. We also add below it a row providing the Legendre symbols, which we know because, above, we identified the squares. We get:

a	1	2	3	4	5	6	7	8	9	10
$\text{DLog}_{\mathbf{Z}_{11}^*, 2}(a)$	0	1	8	2	4	9	7	3	6	5
$J_{11}(a)$	1	-1	1	1	1	-1	-1	-1	1	-1

We observe that the Legendre symbol of a is 1 if its discrete logarithm is even, and -1 if the discrete logarithm is odd, meaning the squares are exactly those group elements whose discrete logarithm is even. It turns out that this fact is true regardless of the choice of generator. ■

As we saw in the above example, the fact that \mathbf{Z}_p^* is cyclic is useful in understanding the structure of the subgroup of quadratic residues $\text{QR}(\mathbf{Z}_p^*)$. The following Proposition summarizes some important elements of this connection.

Proposition 8.15 Let $p \geq 3$ be a prime and let g be a generator of \mathbf{Z}_p^* . Then

$$\text{QR}(\mathbf{Z}_p^*) = \{ g^i : i \in \mathbf{Z}_{p-1} \text{ and } i \text{ is even} \}, \tag{8.2}$$

and the number of squares mod p is

$$|\text{QR}(\mathbf{Z}_p^*)| = \frac{p-1}{2}.$$

Furthermore, every square mod p has exactly two different square roots mod p . ■

Proof of Proposition 8.15: Let

$$E = \{ g^i : i \in \mathbf{Z}_{p-1} \text{ and } i \text{ is even} \}.$$

We will prove that $E = \text{QR}(\mathbf{Z}_p^*)$ by showing first that $E \subseteq \text{QR}(\mathbf{Z}_p^*)$ and second that $\text{QR}(\mathbf{Z}_p^*) \subseteq E$.

To show that $E \subseteq \text{QR}(\mathbf{Z}_p^*)$, let $a \in E$. We will show that $a \in \text{QR}(\mathbf{Z}_p^*)$. Let $i = \text{DLog}_{\mathbf{Z}_p^*, g}(a)$. Since $a \in E$ we know that i is even. Let $j = i/2$ and note that $j \in \mathbf{Z}_{p-1}$. Clearly

$$(g^j)^2 \equiv g^{2j \bmod p-1} \equiv g^{2j} \equiv g^i \pmod{p},$$

so g^j is a square root of $a = g^i$. So a is a square.

To show that $\text{QR}(\mathbf{Z}_p^*) \subseteq E$, let b be any element of \mathbf{Z}_p^* . We will show that $b^2 \in E$.

Let $j = \text{DLog}_{\mathbf{Z}_p^*, g}(b)$. Then

$$b^2 \equiv (g^j)^2 \equiv g^{2j \bmod p-1} \equiv g^{2j} \pmod{p},$$

the last equivalence being true because $p - 1$ is even. This shows that $b^2 \in E$.

The number of even integers in \mathbf{Z}_{p-1} is exactly $(p - 1)/2$ since $p - 1$ is even. The claim about the size of $\text{QR}(\mathbf{Z}_p^*)$ thus follows from Equation (8.2). It remains to justify the claim that every square mod p has exactly two square roots mod p . This can be seen by a counting argument, as follows.

Suppose a is a square mod p . Let $i = \text{DLog}_{\mathbf{Z}_p^*, g}(a)$. We know from the above that i is even. Let $x = i/2$ and let $y = x + (p - 1)/2 \bmod (p - 1)$. Then g^x is a square root of a . Furthermore

$$(g^y)^2 \equiv g^{2y} \equiv g^{2x+(p-1)} \equiv g^{2x} g^{p-1} \equiv a \cdot 1 \equiv a \pmod{p},$$

so g^y is also a square root of a . Since i is an even number in \mathbf{Z}_{p-1} and $p - 1$ is even, it must be that $0 \leq x < (p - 1)/2$. It follows that $(p - 1)/2 \leq y < p - 1$. Thus $x \neq y$. This means that a has at least two square roots. This is true for each of the $(p - 1)/2$ squares mod p . So the only possibility is that each of these squares has exactly two square roots. ■

Suppose we are interested in knowing whether or not a given $a \in \mathbf{Z}_p^*$ is a square mod p , meaning we want to know the value of the Legendre symbol $J_p(a)$. Proposition 8.15 tells us that

$$J_p(a) = (-1)^{\text{DLog}_{\mathbf{Z}_p^*, g}(a)},$$

where g is any generator of \mathbf{Z}_p^* . This however is not very useful in computing $J_p(a)$, because it requires knowing the discrete logarithm of a , which is hard to compute. The following Proposition says that the Legendre symbols of a modulo an odd prime p can be obtained by raising a to the power $(p - 1)/2$, and helps us compute the Legendre symbol.

Proposition 8.16 Let $p \geq 3$ be a prime. Then

$$J_p(a) \equiv a^{\frac{p-1}{2}} \pmod{p}$$

for any $a \in \mathbf{Z}_p^*$. ■

Now one can determine whether or not a is a square mod p by running the algorithm MOD-EXP on inputs $a, (p - 1)/2, p$. If the algorithm returns 1 then a is a square mod p , and if it returns $p - 1$ (which is the same as $-1 \bmod p$) then a is a non-square mod p . Thus, the Legendre symbol can be computed in time cubic in the length of p .

Towards the proof of Proposition 8.16, we begin with the following lemma which is often useful in its own right.

Lemma 8.17 Let $p \geq 3$ be a prime. Then

$$g^{\frac{p-1}{2}} \equiv -1 \pmod{p}$$

for any generator g of \mathbf{Z}_p^* . ■

Proof of Lemma 8.17: We begin by observing that 1 and -1 are both square roots of 1 mod p , and are distinct. (It is clear that squaring either of these yields 1, so they are square roots of 1. They are distinct because -1 equals $p - 1$ mod p , and $p - 1 \neq 1$ because $p \geq 3$.) By Proposition 8.15, these are the only square roots of 1. Now let

$$b = g^{\frac{p-1}{2}} \pmod{p}.$$

Then $b^2 \equiv 1 \pmod{p}$, so b is a square root of 1. By the above b can only be 1 or -1 . However, since g is a generator, b cannot be 1. (The smallest positive value of i such that g^i is 1 mod p is $i = p - 1$.) So the only choice is that $b \equiv -1 \pmod{p}$, as claimed. ■

Proof of Proposition 8.16: By definition of the Legendre symbol, we need to show that

$$a^{\frac{p-1}{2}} \equiv \begin{cases} 1 \pmod{p} & \text{if } a \text{ is a square mod } p \\ -1 \pmod{p} & \text{otherwise.} \end{cases}$$

Let g be a generator of \mathbf{Z}_p^* and let $i = \text{DLog}_{\mathbf{Z}_p^*, g}(a)$. We consider separately the cases of a being a square and a being a non-square.

Suppose a is a square mod p . Then Proposition 8.15 tells us that i is even. In that case

$$a^{\frac{p-1}{2}} \equiv (g^i)^{\frac{p-1}{2}} \equiv g^{i \cdot \frac{p-1}{2}} \equiv (g^{p-1})^{i/2} \equiv 1 \pmod{p},$$

as desired.

Now suppose a is a non-square mod p . Then Proposition 8.15 tells us that i is odd. In that case

$$a^{\frac{p-1}{2}} \equiv (g^i)^{\frac{p-1}{2}} \equiv g^{i \cdot \frac{p-1}{2}} \equiv g^{(i-1) \cdot \frac{p-1}{2} + \frac{p-1}{2}} \equiv (g^{p-1})^{(i-1)/2} \cdot g^{\frac{p-1}{2}} \equiv g^{\frac{p-1}{2}} \pmod{p}.$$

However Lemma 8.17 tells us that the last quantity is -1 modulo p , as desired. ■

The following Proposition says that $ab \pmod{p}$ is a square if and only if either both a and b are squares, or if both are non-squares. But if one is a square and the other is not, then $ab \pmod{p}$ is a non-square. This can be proved by using either Proposition 8.15 or Proposition 8.16. We use the latter in the proof. You might try, as an exercise, to reprove the result using Proposition 8.15 instead.

Proposition 8.18 Let $p \geq 3$ be prime. Then

$$J_p(ab \pmod{p}) = J_p(a) \cdot J_p(b)$$

for all $a, b \in \mathbf{Z}_p^*$. ■

Proof of Proposition 8.18: Using Proposition 8.16 we get

$$J_p(ab \pmod{p}) \equiv (ab)^{\frac{p-1}{2}} \equiv a^{\frac{p-1}{2}} b^{\frac{p-1}{2}} \equiv J_p(a) \cdot J_p(b) \pmod{p}.$$

The two quantities we are considering both being either 1 or -1 , and equal modulo p , must then be actually equal. ■

A quantity of cryptographic interest is the Diffie-Hellman (DH) key. Having fixed a cyclic group G and generator g for it, the DH key associated to elements $X = g^x$ and $Y = g^y$ of the group is the group element g^{xy} . The following Proposition tells us that the DH key is a square if either X or Y is a square, and otherwise is a non-square.

Proposition 8.19 Let $p \geq 3$ be a prime and let g be a generator of \mathbf{Z}_p^* . Then

$$J_p(g^{xy} \bmod p) = 1 \quad \text{if and only if} \quad J_p(g^x \bmod p) = 1 \text{ or } J_p(g^y \bmod p) = 1,$$

for all $x, y \in \mathbf{Z}_{p-1}$ ■

Proof of Proposition 8.19: By Proposition 8.15, it suffices to show that

$$xy \bmod (p - 1) \text{ is even} \quad \text{if and only if} \quad x \text{ is even or } y \text{ is even}.$$

But since $p - 1$ is even, $xy \bmod (p - 1)$ is even exactly when xy is even, and clearly xy is even exactly if either x or y is even. ■

With a cyclic group G and generator g of G fixed, we will be interested in the distribution of the DH key g^{xy} in G , under random choices of x, y from \mathbf{Z}_m , where $m = |G|$. One might at first think that in this case the DH key is a random group element. The following proposition tells us that in the group \mathbf{Z}_p^* of integers modulo a prime, this is certainly not true. The DH key is significantly more likely to be a square than a non-square, and in particular is thus not even almost uniformly distributed over the group.

Proposition 8.20 Let $p \geq 3$ be a prime and let g be a generator of \mathbf{Z}_p^* . Then

$$\Pr \left[x \stackrel{R}{\leftarrow} \mathbf{Z}_{p-1}; y \stackrel{R}{\leftarrow} \mathbf{Z}_{p-1} : J_p(g^{xy}) = 1 \right]$$

equals $3/4$. ■

Proof of Proposition 8.20: By Proposition 8.20 we need only show that

$$\Pr \left[x \stackrel{R}{\leftarrow} \mathbf{Z}_{p-1}; y \stackrel{R}{\leftarrow} \mathbf{Z}_{p-1} : J_p(g^x) = 1 \text{ or } J_p(g^y) = 1 \right]$$

equals $3/4$. The probability in question is $1 - \alpha$ where

$$\begin{aligned} \alpha &= \Pr \left[x \stackrel{R}{\leftarrow} \mathbf{Z}_{p-1}; y \stackrel{R}{\leftarrow} \mathbf{Z}_{p-1} : J_p(g^x) = -1 \text{ and } J_p(g^y) = -1 \right] \\ &= \Pr \left[x \stackrel{R}{\leftarrow} \mathbf{Z}_{p-1} : J_p(g^x) = -1 \right] \cdot \Pr \left[y \stackrel{R}{\leftarrow} \mathbf{Z}_{p-1} : J_p(g^y) = -1 \right] \\ &= \frac{|\text{QR}(\mathbf{Z}_p^*)|}{|\mathbf{Z}_p^*|} \cdot \frac{|\text{QR}(\mathbf{Z}_p^*)|}{|\mathbf{Z}_p^*|} \\ &= \frac{(p-1)/2}{p-1} \cdot \frac{(p-1)/2}{p-1} \end{aligned}$$

$$\begin{aligned}
&= \frac{1}{2} \cdot \frac{1}{2} \\
&= \frac{1}{4}.
\end{aligned}$$

Thus $1 - \alpha = 3/4$ as desired. Here we used Proposition 8.15 which told us that $|\text{QR}(\mathbf{Z}_p^*)| = (p - 1)/2$. ■

The above Propositions, combined with Proposition 8.16 (which tells us that quadratic residuosity modulo a prime can be efficiently tested), will later lead us to pinpoint weaknesses in certain cryptographic schemes in \mathbf{Z}_p^* .

8.5 Groups of prime order

A group of prime order is a group G whose order $m = |G|$ is a prime number. Such a group is always cyclic. These groups turn out to be quite useful in cryptography, so let us take a brief look at them and some of their properties.

An element h of a group G is called *non-trivial* if it is not equal to the identity element of the group.

Proposition 8.21 Suppose G is a group of order q where q is a prime, and h is any non-trivial member of G . Then h is a generator of G . ■

Proof of Proposition 8.21: It suffices to show that the order of h is q . We know that the order of any group element must divide the order of the group. Since the group has prime order q , the only possible values for the order of h are 1 and q . But h does not have order 1 since it is non-trivial, so it must have order q . ■

A common way to obtain a group of prime order for cryptographic schemes is as a subgroup of a group of integers modulo a prime. We pick a prime p having the property that $q = (p - 1)/2$ is also prime. It turns out that the subgroup of quadratic residues modulo p then has order q , and hence is a group of prime order. The following proposition summarizes the facts for future reference.

Proposition 8.22 Let $q \geq 3$ be a prime such that $p = 2q + 1$ is also prime. Then $\text{QR}(\mathbf{Z}_p^*)$ is a group of prime order q . Furthermore, if g is any generator of \mathbf{Z}_p^* , then $g^2 \bmod p$ is a generator of $\text{QR}(\mathbf{Z}_p^*)$. ■

Note that the operation under which $\text{QR}(\mathbf{Z}_p^*)$ is a group is multiplication modulo p , the same operation under which \mathbf{Z}_p^* is a group.

Proof of Proposition 8.22: We know that $\text{QR}(\mathbf{Z}_p^*)$ is a subgroup, hence a group in its own right. Proposition 8.15 tells us that $|\text{QR}(\mathbf{Z}_p^*)|$ is $(p - 1)/2$, which equals q in this case. Now let g be a generator of \mathbf{Z}_p^* and let $h = g^2 \bmod p$. We want to show that h is a generator of $\text{QR}(\mathbf{Z}_p^*)$. As per Proposition 8.21, we need only show that

h is non-trivial, meaning $h \neq 1$. Indeed, we know that $g^2 \not\equiv 1 \pmod{p}$, because g , being a generator, has order p and our assumptions imply $p > 2$. ■

Example 8.23 Let $q = 5$ and $p = 2q + 1 = 11$. Both p and q are primes. We know from Example 8.14 that

$$\text{QR}(\mathbf{Z}_{11}^*) = \{1, 3, 4, 5, 9\}.$$

This is a group of prime order 5. We know from Example 8.7 that 2 is a generator of \mathbf{Z}_p^* . Proposition 8.22 tells us that $4 = 2^2$ is a generator of $\text{QR}(\mathbf{Z}_{11}^*)$. We can verify this by raising 4 to the powers $i = 0, \dots, 4$:

i	0	1	2	3	4
$4^i \pmod{11}$	1	4	5	9	3

We see that the elements of the last row are exactly those of the set $\text{QR}(\mathbf{Z}_{11}^*)$. ■

Let us now explain what we perceive to be the advantage conferred by working in a group of prime order. Let G be a cyclic group, and g a generator. We know that the discrete logarithms to base g range in the set \mathbf{Z}_m where $m = |G|$ is the order of G . This means that arithmetic in these exponents is modulo m . If G has prime order, then m is prime. This means that *any non-zero exponent has an inverse modulo m* . In other words, in working in the exponents, we can divide. It is this that turns out to be useful.

As an example illustrating how we use this, let us return to the problem of the distribution of the DH key that we looked at in Section 8.4. Recall the question is that we draw x, y independently at random from \mathbf{Z}_m and then ask how g^{xy} is distributed over G . We saw that when $G = \mathbf{Z}_p^*$ for a prime $p \geq 3$, this distribution was noticeably different from uniform. In a group of prime order, the distribution of the DH key, in contrast, is very close to uniform over G . It is not quite uniform, because the identity element of the group has a slightly higher probability of being the DH key than other group elements, but the deviation is small enough to be negligible for groups of reasonably large size. The following proposition summarizes the result.

Proposition 8.24 Suppose G is a group of order q where q is a prime, and let g be a generator of G . Then for any $Z \in G$ we have

$$\Pr \left[x \xleftarrow{R} \mathbf{Z}_q; y \xleftarrow{R} \mathbf{Z}_q : g^{xy} = Z \right] = \begin{cases} \frac{1}{q} \left(1 - \frac{1}{q}\right) & \text{if } Z \neq \mathbf{1} \\ \frac{1}{q} \left(2 - \frac{1}{q}\right) & \text{if } Z = \mathbf{1}, \end{cases}$$

where $\mathbf{1}$ denotes the identity element of G . ■

Proof of Proposition 8.24: First suppose $Z = \mathbf{1}$. The DH key g^{xy} is $\mathbf{1}$ if and only if either x or y is 0 modulo q . Each is 0 with probability $1/q$ and these probabilities are independent, so the probability that either x or y is 0 is $2/q - 1/q^2$, as claimed.

Now suppose $Z \neq \mathbf{1}$. Let $z = \text{DLog}_{G,g}(Z)$, meaning $z \in \mathbf{Z}_q^*$ and $g^z = Z$. We will have $g^{xy} \equiv Z \pmod{p}$ if and only if $xy \equiv z \pmod{q}$, by the uniqueness of the discrete logarithm. For any fixed $x \in \mathbf{Z}_q^*$, there is exactly one $y \in \mathbf{Z}_q$ for which $xy \equiv z \pmod{q}$, namely $y = x^{-1} \pmod{q}$, the multiplicative inverse of x in the group \mathbf{Z}_q^* . (Here we are making use of the fact that q is prime, since otherwise the inverse of x modulo q may not exist.) Now, suppose we choose x at random from \mathbf{Z}_q . If $x = 0$ then, regardless of the choice of $y \in \mathbf{Z}_q$, we will not have $xy \equiv z \pmod{q}$, because $z \not\equiv 0 \pmod{q}$. On the other hand, if $x \neq 0$ then there is exactly $1/q$ probability that the randomly chosen y is such that $xy \equiv z \pmod{q}$. So the probability that $xy \equiv z \pmod{q}$ when both x and y are chosen at random in \mathbf{Z}_q is

$$\frac{q-1}{q} \cdot \frac{1}{q} = \frac{1}{q} \left(1 - \frac{1}{q}\right)$$

as desired. Here, the first term is because when we choose x at random from \mathbf{Z}_q , it has probability $(q-1)/q$ of landing in \mathbf{Z}_q^* . ■

8.6 Historical Notes

8.7 Exercises and Problems

Chapter 9

ASYMMETRIC ENCRYPTION

Chapter 10

DIGITAL SIGNATURES

Chapter 11

KEY DISTRIBUTION

Chapter 12

THE ASYMPTOTIC APPROACH

Chapter 13

INTERACTIVE PROOFS AND ZERO KNOWLEDGE

Chapter 14

MORE PROTOCOLS

Part I

Appendices

Appendix A

THE BIRTHDAY PROBLEM

The setting is that we have q balls. View them as numbered, $1, \dots, q$. We also have N bins, where $N \geq q$. We throw the balls at random into the bins, one by one, beginning with ball 1. At random means that each ball is equally likely to land in any of the N bins, and the probabilities for all the balls are independent. A collision is said to occur if some bin ends up containing at least two balls. We are interested in $C(N, q)$, the probability of a collision.

The birthday paradox is the case where $N = 365$. We are asking what is the chance that, in a group of q people, there are two people with the same birthday, assuming birthdays are randomly and independently distributed over the days of the year. It turns out that when q hits $\sqrt{365}$ the chance of a birthday collision is already quite high, around $1/2$.

This fact can seem surprising when first heard. The reason it is true is that the collision probability $C(N, q)$ grows roughly proportional to q^2/N . This is the fact to remember. The following gives a more exact rendering, providing both upper and lower bounds on this probability.

Proposition A.1 Let $C(N, q)$ denote the probability of at least one collision when we throw $q \geq 1$ balls at random into $N \geq q$ buckets. Then

$$C(N, q) \leq \frac{q(q-1)}{2N}.$$

Also

$$C(N, q) \geq 1 - e^{-q(q-1)/2N},$$

and

$$C(N, q) \geq 0.3 \cdot \frac{q(q-1)}{N}$$

for $1 \leq q \leq \sqrt{2N}$. ■

In the proof we will find the following inequalities useful to make estimates.

Proposition A.2 The inequality

$$\left(1 - \frac{1}{e}\right) \cdot x \leq 1 - e^{-x} \leq x.$$

is true for any real number x with $0 \leq x \leq 1$. ■

Proof of Proposition A.1: Let C_i be the event that the i -th ball collides with one of the previous ones. Then $\Pr[C_i]$ is at most $(i-1)/N$, since when the i -th ball is thrown in, there are at most $i-1$ different occupied slots and the i -th ball is equally likely to land in any of them. Now

$$\begin{aligned} C(N, q) &= \Pr[C_1 \vee C_2 \vee \cdots \vee C_q] \\ &\leq \Pr[C_1] + \Pr[C_2] + \cdots + \Pr[C_q] \\ &\leq \frac{0}{N} + \frac{1}{N} + \cdots + \frac{q-1}{N} \\ &= \frac{q(q-1)}{2N}. \end{aligned}$$

This proves the upper bound. For the lower bound we let D_i be the event that there is no collision after having thrown in the i -th ball. If there is no collision after throwing in i balls then they must all be occupying different slots, so the probability of no collision upon throwing in the $(i+1)$ -st ball is exactly $(N-i)/N$. That is,

$$\Pr[D_{i+1} | D_i] = \frac{N-i}{N} = 1 - \frac{i}{N}.$$

Also note $\Pr[D_1] = 1$. The probability of no collision at the end of the game can now be computed via

$$\begin{aligned} 1 - C(N, q) &= \Pr[D_q] \\ &= \Pr[D_q | D_{q-1}] \cdot \Pr[D_{q-1}] \\ &\quad \vdots \\ &= \prod_{i=1}^{q-1} \Pr[D_{i+1} | D_i] \\ &= \prod_{i=1}^{q-1} \left(1 - \frac{i}{N}\right). \end{aligned}$$

Note that $i/N \leq 1$. So we can use the inequality $1 - x \leq e^{-x}$ for each term of the above expression. This means the above is not more than

$$\prod_{i=1}^{q-1} e^{-i/N} = e^{-1/N - 2/N - \cdots - (q-1)/N} = e^{-q(q-1)/2N}.$$

Putting all this together we get

$$C(N, q) \geq 1 - e^{-q(q-1)/2N},$$

which is the second inequality in Proposition A.1. To get the last one, we need to make some more estimates. We know $q(q-1)/2N \leq 1$ because $q \leq \sqrt{2N}$, so we can use the inequality $1 - e^{-x} \geq (1 - e^{-1})x$ to get

$$C(N, q) \geq \left(1 - \frac{1}{e}\right) \cdot \frac{q(q-1)}{2N}.$$

A computation of the constant here completes the proof. ■

Appendix B

PROBABILITY THEORY

Bibliography

- [1] MIHIR BELLARE. Practice-oriented provable security. Available via <http://www-cse.ucsd.edu/users/mihir/crypto-papers.html>.
- [2] M. BELLARE, J. KILIAN AND P. ROGAWAY. The security of the cipher block chaining message authentication code. *Journal of Computer and System Sciences* , Vol. 61, No. 3, Dec 2000, pp. 362–399.
- [3] M. BELLARE, A. DESAI, E. JOKIPII, AND P. ROGAWAY. A Concrete Security Treatment of Symmetric Encryption: Analysis of the DES Modes of Operation. *Proceedings of the 38th Symposium on Foundations of Computer Science*, IEEE, 1997.
- [4] M. BELLARE AND O. GOLDBREICH. On defining proofs of knowledge. *Advances in Cryptology – CRYPTO '92*, Lecture Notes in Computer Science Vol. 740, E. Brickell ed., Springer-Verlag, 1992.
- [5] M. BELLARE, R. IMPAGLIAZZO AND M. NAOR. Does parallel repetition lower the error in computationally sound protocols? *Proceedings of the 38th Symposium on Foundations of Computer Science*, IEEE, 1997.
- [6] G. BRASSARD, D. CHAUM, AND C. CRÉPEAN. Minimum Disclosure Proofs of knowledge. *Journal of Computer and System Sciences*, Vol. 37, No. 2, 1988, pp. 156–189.
- [7] Data Encryption Standard. FIPS PUB 46, Appendix A, Federal Information Processing Standards Publication, January 15, 1977, US Dept. of Commerce, National Bureau of Standards.
- [8] J. DAEMEN AND V. RIJMEN. AES proposal: Rijndael. <http://csrc.nist.gov/encryption/aes/rijndael/Rijndael.pdf>.
- [9] W. DIFFIE AND M. HELLMAN. New directions in cryptography. *IEEE Trans. Info. Theory*, Vol. IT-22, No. 6, November 1976, pp. 644–654.
- [10] U. FEIGE, A. FIAT, AND A. SHAMIR. Zero-Knowledge Proofs of Identity. *Journal of Cryptology*, Vol. 1, 1988, pp. 77–94.

- [11] U. FEIGE, AND A. SHAMIR. Witness Indistinguishability and Witness Hiding Protocols. *Proceedings of the 22nd Annual Symposium on the Theory of Computing*, ACM, 1990.
- [12] O. GOLDREICH. A uniform complexity treatment of encryption and zero-knowledge. *Journal of Cryptology*, Vol. 6, 1993, pp. 21-53.
- [13] O. GOLDREICH AND H. KRAWCZYK. On the Composition of Zero-Knowledge Proof Systems. *SIAM Journal on Computing*, Vol. 25, No. 1, 1996, pp. 169–192.
- [14] O. GOLDREICH, S. MICALI, AND A. WIGDERSON. Proofs that Yields Nothing but Their Validity, or All Languages in NP Have Zero-Knowledge Proof Systems. *Journal of the ACM*, Vol. 38, No. 1, July 1991, pp. 691–729.
- [15] O. GOLDREICH AND Y. OREN. Definitions and Properties of Zero-Knowledge Proof Systems. *Journal of Cryptology*, Vol. 7, No. 1, 1994, pp. 1–32.
- [16] O. GOLDREICH, S. GOLDWASSER AND S. MICALI. How to construct random functions. *Journal of the ACM*, Vol. 33, No. 4, 1986, pp. 210–217.
- [17] S. GOLDWASSER AND S. MICALI. Probabilistic encryption. *J. of Computer and System Sciences*, Vol. 28, April 1984, pp. 270–299.
- [18] S. GOLDWASSER, S. MICALI AND C. RACKOFF. The knowledge complexity of interactive proof systems. *SIAM J. of Comp.*, Vol. 18, No. 1, pp. 186–208, February 1989.
- [19] S. GOLDWASSER, S. MICALI AND R. RIVEST. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal of Computing*, Vol. 17, No. 2, pp. 281–308, April 1988.
- [20] A. JOUX AND R. LERCIER. Computing a discrete logarithm in $GF(p)$, p a 120 digits prime, <http://www.medicis.polytechnique.fr/~lercier/english/dlog.html>.
- [21] D. KAHN. *The Codebreakers; The Comprehensive History of Secret Communication from Ancient Times to the Internet*. Scribner, Revised edition, December 1996.
- [22] M. LUBY AND C. RACKOFF. How to construct pseudorandom permutations from pseudorandom functions. *SIAM J. Comput*, Vol. 17, No. 2, April 1988.
- [23] M. LUBY AND C. RACKOFF. A study of password security. *Advances in Cryptology – CRYPTO '87*, Lecture Notes in Computer Science Vol. 293, C. Pomerance ed., Springer-Verlag, 1987.

- [24] C. LUND, L. FORTNOW, H. KARLOFF AND N. NISAN. Algebraic Methods for Interactive Proof Systems. *Journal of the ACM*, Vol. 39, No. 4, 1992, pp. 859–868.
- [25] S. MICALI, C. RACKOFF AND R. SLOAN. The notion of security for probabilistic cryptosystems. *SIAM J. of Computing*, April 1988.
- [26] M. NAOR AND M. YUNG, Public-key cryptosystems provably secure against chosen ciphertext attacks. *Proceedings of the 22nd Annual Symposium on the Theory of Computing*, ACM, 1990.
- [27] A. ODLYZKO. The rise and fall of knapsack cryptosystems. Available via <http://www.research.att.com/~amo/doc/cnt.html>.
- [28] C. RACKOFF AND D. SIMON. Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack. *Advances in Cryptology – CRYPTO '91*, Lecture Notes in Computer Science Vol. 576, J. Feigenbaum ed., Springer-Verlag, 1991.
- [29] RONALD RIVEST, MATT ROBSHAW, RAY SIDNEY, AND YIQUIN YIN. The RC6 Block Cipher. Available via <http://theory.lcs.mit.edu/~rivest/publications.html>.
- [30] R. RIVEST, A. SHAMIR, AND L. ADLEMAN. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Communications of the ACM*, Vol. 21, No. 2, February 1978, pp. 120–126.
- [31] A. SHAMIR. $IP = PSPACE$. *Journal of the ACM*, Vol. 39, No. 4, 1992, pp. 869–877.
- [32] D. WEBER AND T. DENNY. The solution of McCurley's discrete log challenge. *Advances in Cryptology – CRYPTO '98*, Lecture Notes in Computer Science Vol. 1462, H. Krawczyk ed., Springer-Verlag, 1998.