Symmetric Cryptography Basics

Chanathip Namprempre, Ph.D.

Symmetric Cryptography Basics

Chanathip Namprempre

October 2011

Contents at a glance

Contents at a glance · iii
Contents · iv
List of Figures · vii
List of Tables · x
Preface · xi
Acknowledgments · xiii
1 Introduction · 1
2 Symmetric Encryption: A First Look · 9
3 Building Block I: Block ciphers · 19
4 Building Block II: Pseudorandom Number Generators · 53
5 Block-Cipher-Based Symmetric Encryption · 61
6 Building Block III: Hash functions · 101
7 Message Authentication Codes · 115
8 Authenticated Encryption · 139
Bibliography · 161
Index · 169

Contents

Contents at a glance iii

Contents iv

List of Figures vii

List of Tables x

Preface xi

Acknowledgments xiii

1 Introduction 1

1.1 What cryptography is about 1

1.2 What cryptography is not about 3

1.3 Two main settings for cryptography 4

1.4 Our approach to cryptography 5

1.5 Conclusion 6

1.6 Exercises 7

2 Symmetric Encryption: A First Look 9

2.1 Caesar cipher: an ancient symmetric encryption scheme 11

2.2 One-time pad: a perfect symmetric encryption scheme 12

2.3 Conclusion 16

2.4 Credits 16

2.5 Exercises 16

3 Building Block I: Block ciphers 19

3.1 Motivation 19

3.2 What it is 19

3.3 Constructions 21
3.4 What security means 30
3.5 Attacks against block ciphers and PRFs 45
3.6 Conclusion 48
3.7 Credits 48
3.8 Exercises 49
4 Building Block II: Pseudorandom Number Generators 53
4.1 Motivation 53
4.2 What it is 54
4.3 Gaps between the model and practical constructions 55
4.4 Attacks against PRNGs 56
4.5 Conclusion 58
4.6 Credits 58
4.7 Exercises 58
5 Block-Cipher-Based Symmetric Encryption 61
5.1 Motivation 61
5.2 What it is: syntax and correctness condition 62
5.3 What security means: privacy 67
5.4 Constructions: modes of operation 71
5.5 Attacks against block-cipher-based symmetric encryption schemes 82
5.6 Examples of security proofs for symmetric encryption 90
5.7 Conclusion 96
5.8 Credits 97
5.9 Exercises 97
6 Building Block III: Hash functions 101
6.1 Motivation 101
6.2 What it is 102
6.3 What security means 103
6.4 Hash function design 109

6.5 Conclusion . . . 111
6.6 Credits . . . 112
6.7 Exercises . . . 112
7 Message Authentication Codes . . . 115
7.1 Motivation . . . 115
7.2 What it is: syntax and correctness condition . . . 116
7.3 What security means: unforgeability . . . 118
7.4 Constructions . . . 122
7.5 Attacks against MAC schemes . . . 131
7.6 Conclusion . . . 133
7.7 Credits . . . 134
7.8 Exercises . . . 134
8 Authenticated Encryption . . . 139
8.1 Motivation . . . 139
8.2 What it is: syntax and correctness condition . . . 139
8.3 What security means: privacy and authenticity . . . 140
8.4 Constructions based on the generic composition paradigm . . . 144
8.5 Other approaches . . . 158
8.6 Conclusion . . . 159
8.7 Exercises . . . 159
Bibliography . . . 161
Index . . . 169

List of Figures

2.1 An analogy for symmetric encryption. 9

2.2 The key for locking and unlocking the box must be agreed upon by the communicating parties *before* the communication takes place. . 10

2.3 A (likely historically inaccurate) depiction of the use of caesar cipher. 12

3.1 A whimsical depiction of what a block cipher does in the forward direction. 20

3.2 Encipherment via a block cipher (E, E^{-1}). 20

3.3 The structure of DES. 22

3.4 The initial permutation and the inverse initial permutation. 23

(a) IP 23

(b) IP^{-1} 23

3.5 Double DES 25

(a) forward double DES computation 25

(b) backward double DES computation 25

3.6 Triple DES 26

(a) Forward triple DES computation 26

(b) backward triple DES computation 26

3.7 DESX 28

(a) forward DESX computation 28

(b) backward DESX computation 28

3.8 All possible functions with the signature $\{0,1\} \times \{0,1\} \to \{0,1\}$. 31

3.9 All possible functions with the signature $\{0,1\} \times \{0,1\} \to \{0,1\}$ now viewed as being indexed by the first component of the input. . 32
3.10 A typical experiment for security definitions in modern cryptography. 36
3.11 An experiment for the PRP-CPA security definition for block ciphers. The behavior of the permutation g depends on whether the coin flip in the experiment results in a head or a tail. 37
3.12 An experiment for the PRP-CCA security definition for block ciphers. The behaviors of the permutations g and g^{-1} depend on whether the coin flip in the experiment results in a head or a tail. . 41
5.1 An experiment for the IND-CPA security definition for symmetric encryption. 71
5.2 An experiment for the IND-CCA security definition for symmetric encryption. 72
5.3 The encryption algorithm for the electronic codebook mode (ECB). 74
5.4 The encryption algorithm for the cipher block chaining mode (CBC). 75
5.5 The encryption algorithm for the cipher feedback mode (CFB). . . 77
5.6 The encryption algorithm for the output feedback mode (OFB). . . 78
5.7 The encryption algorithm for the counter mode (CTR). 80
5.8 The main idea for mounting a chosen-ciphertext attack against CBCr. 89
5.9 Subroutine `SimEnc` used by B. 94
6.1 A way to think about hash functions. The digest is supposed to capture the essense of the object being hashed. 101
6.2 A hash function H. 102
6.3 Roughly, preimage resistance means that it should be difficult to figure out what was hashed simply by looking at the hash value. . . . 103
6.4 For collision resistant hash functions, one would expect that, in most cases, hashing two distinct input values would result in two different hash values. 104

6.5 A collision can always occur, however, since a hash function maps values from a large set to those in a smaller set. 104
6.6 The Merkle-Damgard construciton. 109
7.1 Another way to think about hash functions. A document is hashed to obtain its digest as a bit string using a hash function depicted here as a funnel. 117
7.2 In order to compute a digest using the tagging algorithm of a message authentication scheme, one needs to supply both an input message and the secret key. In effect, the secret key specifies which function to use to compute the digest. 118
7.3 The tagging algorithm for CBC MAC. 122
7.4 The tagging algorithm for EMAC. 124
7.5 The tagging algorithm for XCBC MAC. 130
8.1 A knight delivers an important message. The message is kept secret (as conveyed by the use of an opaque envelope) and authenticated (as conveyed by the use of a special stamp of approval). 140
8.2 If a symmetric encryption was a car and a MAC scheme was a boat, we can just put the two together in a “black-box” manner and hope that we get an amphibious vehicle, can’t we? 145
8.3 Combining a car and a boat in this manner seems unlikely to result in a vehicle that could function both on the road and in a lake. . . . 145
8.4 Combining a car and a boat in this manner seems promising. But would it actually work? The devil is most likely in the details. . . . 146
8.5 A depiction of the *Encrypt-and-MAC* composition method. 149
8.6 A depiction of the *MAC-then-encrypt* composition method. 152
8.7 A depiction of the *Encrypt-then-MAC* composition method. 155

List of Tables

5.1 A summary of the power and goal of any adversary mounting chosen-plaintext and chosen-ciphertext attacks against a symmetric encryption scheme. 73

7.1 A summary of the power and goal of any adversary mounting chosen-message attacks against a MAC scheme. 121

8.1 Summary of results under the assumption that the underlying encryption scheme is IND-CPA and the underlying MAC scheme is weakly unforgeable. 148

8.2 Summary of results under the assumption that the underlying encryption scheme is IND-CPA and the underlying MAC scheme is strongly unforgeable. 148

8.3 Summary of security results for the *Encrypt-and-MAC* composition method. 150

8.4 Summary of security results for the *MAC-then-Encrypt* composition method. 153

8.5 Summary of security results for the *Encrypt-then-MAC* composition method. 156

Preface

It is a fact that the more we rely on computer systems, the more important it becomes to secure them. Perpetrators today have more incentives than ever to subvert the security mechanisms so as to exploit flaws in systems for their own gain. Whether it be systems that control critical infrastructure such as power grids, air traffic control systems, and financial systems, or more down-to-earth systems such as email and other application services, the damage that can occur can range anywhere from mere annoyance on a personal level to a catastrophe on an international level.

While it is true that most incidents we hear about every couple of days have to do with social engineering and exploits of software bugs, it is also clear that one would be hard pressed to protect computer and network systems without the use of cryptography. Without cryptographers, one would not have the Advanced Encryption Standard (to use as a basis for encrypting data to keep them secret), the SHA-3 hash function (to hash passwords to mitigate damage that could occur were an attacker to gain access to the database of user information), and the various encryption modes (to secure communication between web browsers and web servers during, say, an online banking transaction).

Cryptography as a field has come a long way from an art form to a science. The history of cryptography is full of fascinating stories involving ingenious characters. (See The Codebreakers [29], for example.) Many textbooks and recreational science books abound explaining back stories behind various machines used in wars and detailing how various ciphers work. It was not until relatively recently that we started to see textbooks focusing more on analysis and more systematic treatments of cryp-

tosystems. This is the basis of what is often referred to as *modern cryptography*.

This book aims to be in the category of modern cryptography with an emphasis on symmetric cryptosystems. Material in the book is organized so that one can get to concrete, well-known cryptographic primitives as soon as possible. Consequently, the book is not arranged in a strictly bottom-up approach, as one might expect in a book with a heavy mathematical slant. Rather, the building blocks and their applications are interleaved throughout the book. This feature should hopefully make it more convenient for instructors to use the book as a text for a class on the subject.

Chanathip Namprempre, Klong Luang, Patumtani. October 2011.

Acknowledgments

Any praise for this book should be attributed to all my teachers. I am especially indebted to Mihir Bellare for giving me a chance and helping me get my start in cryptography. His dedication to mentoring and his generosity in offering his students not only professional guidance but friendship as well have inspired me to do the same for my students. I am also deeply grateful to Phillip Rogaway for his kindness, wisdom, and professional companionship, which has helped keep me motivated to continue with cryptography research in Thailand.

I would also like to thank many people who made this book possible. I thank all my students for, intentionally or accidentally, providing me insights into how to actually make the book useful to people who do not already know the material.

Finally, I would like to thank my family for their support and encouragement. I thank my children for their wonderful smiles and infectious laughs. I thank my husband for putting them to bed and keeping them there every night so that I could get this book done. And most importantly, I thank him for all the professional, intellectual, and emotional support he has unyieldingly provided me over the years all while talking nerdy to me.

1

Introduction

We first specify the scope of cryptography then discuss the various problems that can be addressed by advances in cryptography and our approach in exploring the field.

1.1 What cryptography is about

A very precise definition of cryptography is due to Ronald Rivest, one of the most prominent researchers in the area. He defines cryptography as "communication in the presence of adversaries." This implies many things including, for example,

- that there are parties trying to communicate,
- that there should be something that distinguishes the communicating parties from adversaries, and
- that each of the communicating parties is trying to achieve its goal.

As an example, consider the most well-known use of cryptography, namely, to carry out private communication. Suppose two people would like to exchange messages in such a way that *only* the two of them learn the contents of the messages. Clearly,

- these two people are the communicating parties;
- these two people should know something (or some things) that no one else does; and
- the goal of both people is to communicate in private.

We discuss each of these aspects in turn.

1.1.1 There must be communicating parties.

This may appear rather self-evident, but there are some points worth noting. First, the term "communicate" here is not limited to message transmission but includes other related concepts. Examples range from complex but apparently useful problems such as how to use information technology to help improve a voting system and how to buy goods and services online to quirky-sounding, seemingly pointless problems to solve such as how to flip a coin in a well.

Also, basic textbook communication problems are often presented as involving only two communicating parties, e.g. how can the sender send a piece of data to the receiver so that the data (and, in some applications, the fact that it is sent) remains private, authenticated, non-repudiable, undeniable, and so on. However, there are many other problems that involve fewer or more than two communicating parties. For example, a person storing files on a server so that only he can learn the contents of the files later on can be said to be communicating with himself.[1] An online meeting can involve a large group of more than two people. A typical electronic commerce transaction often involve many parties, e.g. the seller, the buyer, the credit card issuer, the acquirer, and so on.

1.1.2 The communicating parties must have an information advantage over the adversary.

Consider a simple example of a person storing personal files on a server. In order to ensure that no one else can obtain the contents of the files, he must possess something no one else does where possessing may mean knowing some information (such as the correct secret bitstring or password) or having some physical object (such as the correct smartcard with the correct information on it) or some data (such as the correct fingerprint). Such an information advantage distinguishes a legitimate party from an advsersary.

[1] Different genders will be used throughout this book without any discerning pattern.

On a related note, we emphasize that the consensus among cryptographers and security professionals is to limit the information advantage to secret data or physical objects. The practice of keeping the computational processes involved in the communication secret (i.e. known only to the legitimate parties but not the adversary) as a means for the legitimate parties to obtain an information advantage — the so-called "security through obscurity" approach — has fallen out of favor and should be abandoned.

1.1.3 There is a goal for each communicating party.

Not surprisingly, privacy is not the only possible goal for communication. Other common goals include ensuring that the data being exchanged have not been altered, ensuring that a person who has created and published a certain piece of data actually knows the contents represented by the data without requiring him to tell anyone else what those contents are until the time is right, allowing a person who has the privilege to access some resource to communicate that privilege to a verifier in such a way that no other person can, and so on.

Cryptography is about devising methods that would allow various communication goals to be achieved in the presence of adversaries. A **cryptographer** is concerned with designing the algorithms or protocols for doing so. A **cryptanalyst** is concerned with how to derail such efforts.

1.2 What cryptography is not about

Much of the communication taking place in the present day is facilitated by computational devices (e.g. computers, smartphones, smartcards, dedicated hardware, etc.), and data generated as part of the communication often get transmitted over public networks such as the Internet. For example, to ensure that a bank customer is allowed to withdraw cash from an Automated Teller Machine (ATM), the machine needs to ask the customer for his Personal Identification Number (PIN) so that it can check the

PIN against the customer's target bank account, the information regarding which is collected through the customer's ATM card. In this scenario, the ATM is the computational device, and the network is what the bank uses to link the ATM to the bank's network. This connection may be a direct link or a dial-up modem over a telephone line, for example. It is common that an attacker, rather than targeting the algorithms being used to verify the identity of the customer, instead tries to collect the PIN and to copy data from the ATM card. Many methods are used to achieve this goal. For example, an attacker may add a card reading device over the ATM card slot along with a small camera to watch the customer as she types in the PIN. Such attacks are known as "skimming scams" and have proved effective. In this type of attacks, the cryptographic algorithms used to protect the communication have not been broken. Rather, the environment has been compromised.

There are many other examples of such threasts. Computer viruses, spyware, and even simple attacks such as cajoling a PIN out of a bank customer all have successfully prevented legitimate parties from conducting their business. Such attempts do threaten to prevent the communicating parties from achieving their goals and thus need be addressed. However, cryptography generally leaves them to computer security experts and social engineering experts to solve and focuses instead on the algorithm and protocol design aspect, an equally important part of the problem.

1.3 Two main settings for cryptography

Often, in designing cryptographic algorithms or protocols, cryptographers will make certain "set-up assumptions" about the setting in which the legitimate parties in the system are to operate. As an example, consider the case of a customer accessing an ATM. The set-up assumption made here is that the customer and the ATM know the correct PIN corresponding to the customer (or more precisely, to the customer's ATM card that is being inserted into the slot). This is achieved by having the customer choose his own PIN and tell the bank what it is at the time he applies for or picks up

the card from the bank. (Note that telling the bank does not imply telling a bank personnel. A currently common practice is to have the customer type in the PIN on a keypad by himself without a bank personnel looking over his shoulder.)

The two most well-known set-up assumptions give rise to the terms symmetric-key and asymmetric-key cryptography. A **symmetric-key** cryptosystem is one in which it is assumed that the legitimate parties are in possession of a shared secret *before* they start running the algorithms or the protocols. The shared secret gives the information advantage to legitimate parties over an adversary. The name symmetric-key comes from the facts that the secret information is often referred to as a **key** and that the information shared by all of the parties involved is the same, namely the secret key. In contrast, an **asymmetric-key** cryptosystem is one in which each legitimate party is assumed to hold a secret key while other parties are assumed to hold the corresponding **public key** leading to an asymmetry in the setup, thus the term. In a given application, one often ends up using both symmetric and asymmetric cryptosystems rather one or the other.

1.4 Our approach to cryptography

This book focuses on symmetric-key cryptography. Taking a bottom-up approach to our study, we will start with the various low-level building blocks in symmetric-key cryptography. Once we explore their interfaces and their security properties, we then focus on how they can be used to build higher-level primitives. These primitives are then used to construct even higher-level protocols. The following table shows where some example symmetric-key cryptographic constructs fall in this respect.

Abstraction level	Cryptographic constructs
Low-level building blocks	block ciphers
	pseudorandom function families
	pseudorandom number generators
	hash functions
Primitives	symmetric encryption
	message authentication codes
High-level protocols	key exchange

Note that where constructs fall in the table is not dictated by a strict rule. For example, consider hash functions. They are used as an underlying primitive for constructing certain message authentication codes. Thus, in the table, they are classified as a low-level building block. However, a recent trend is to construct hash functions out of block ciphers [35, 15, 45, 40], and thus they could be classified in the second level, namely as a primitive.

1.5 Conclusion

Cryptography is a field of study that is concerned with secure communication in the presence of an adversary. This book focuses on symmetric cryptography, a setting in which the communicating parties possess some shared secret before they start to communicate. We emphasize that while cryptography is a necessary ingredient in securing computer and network systems, it is far from being a sufficient component. Many other issues such as how to safeguard systems against social engineering, fraud, human errors, and poorly written software must be considered in ensuring security.

1.6 Exercises

Exercise 1.1. What is an adversary? Give examples of adversaries in different contexts. For example, what would an adversary against a financial system want to do? What about one against a system deployed for the military?

Exercise 1.2. What is an information advantage? Again, give examples in different contexts.

Exercise 1.3. What are the goals for the "communicating parties" in the context of an Automated Teller Machine (ATM) exchanging data with a bank server? What is an information advantage that the parties have over an adversary in this context?

Exercise 1.4. What is the "security through obscurity" approach? Why is it a bad idea?

2

Symmetric Encryption: A First Look

The most well-known goal in cryptography is to allow two people to communicate in private. In the symmetric setting, this requires that before the two parties can communicate, they must somehow have arrived at a shared piece of information that they, *and only they*, know. This piece of information is referred to as the **secret key** or a **private key**. Intuitively, sending a message to a friend using symmetric encryption is akin to putting the message into an opaque locked box then sending along on a conveyor belt for your friend to pick up. This can only work if you and your friend each have a key whose blade—the part that slides into the lock—looks exactly the same. Figure 2.1 depicts this scenario.

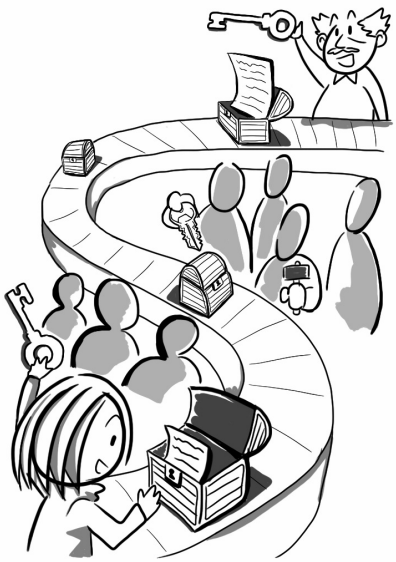


Figure 2.1: An analogy for symmetric encryption.

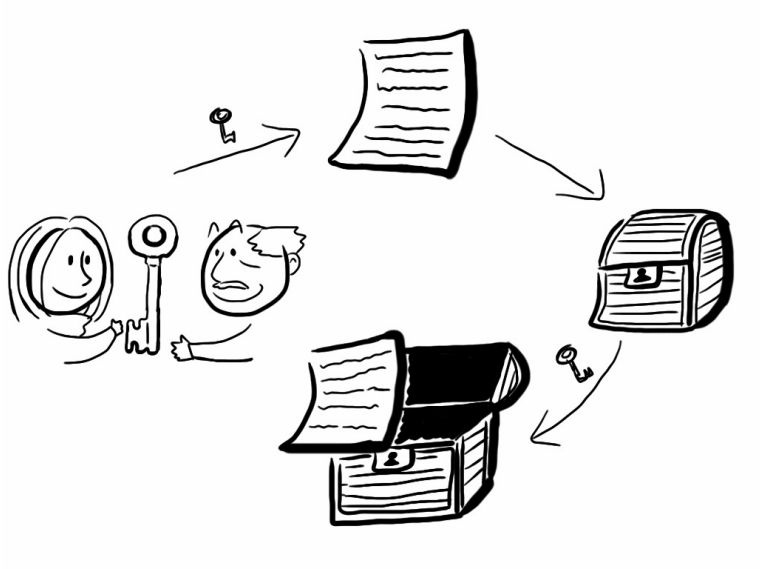


Figure 2.2: The key for locking and unlocking the box must be agreed upon by the communicating parties *before* the communication takes place.

The similarities between using symmetric encryption and a locked box as depicted in Figure 2.1 are easy to see. The two parties have the key that can lock and unlock the box. (Each party has its own copy.) Other observers or adversaries only see the opague box travel by on the conveyor belt just as is the case in digital communication over a public network such as the Internet. (There are many simple tools that allow network users to see contents placed on the shared network even though those contents may be meant for others. See the Wireshark network protocol analyzer [47], for example.) An adversary may want to try his hands on unlocking the box, but if he holds a key that is different from the correct key, then he should have little hope to unlock the box. One thing that cannot be prevented, however, is an adversary who aims to destroy the box with a hammer. This type of attack is akin to packet drop attacks on computer networks. Packet drop attacks often occur in the form of malicious routers dropping packets or nodes in mobile ad hoc networks (MANET) dropping packets rather than forwarding them.

However, one subtle difference between the digital world and the locked-box analog depicted in Figure 2.1 is that the opaque locked box conceals more aspects of

its contents than most symmetric encryption schemes can. Specifically, an observer cannot see any aspect of the contents of the box while most symmetric encryption schemes in use today reveal at least one aspect of the transmitted message, namely its length.

Another aspect that is clear from Figure 2.1 but is worth explicitly pointing out is the fact that, since the key used to lock and unlock the box must look exactly the same, the sender and the receiver must be in possession of the key that looks exactly the same *before* the communication happens. Figure 2.2 emphasizes that the key must be agreed upon by the two communicating parties before they were to exchange messages via the locked box.

2.1 Caesar cipher: an ancient symmetric encryption scheme

A very old and well-known symmetric encryption scheme is **caesar cipher**, so named after its most famous user Julius Caesar.[1] See Figure 2.3 for a (most likely historically inaccurate) depiction of the use of this cipher. It entails replacing each character in the message that the sender wants to send, often referred to as the **plaintext**, with a character that appears a fixed number of positions after it in the alphabetical order. This number can be considered to be the secret key. Once every character in the plaintext has been processed, the result, referred to as the **ciphertext**, is then sent to the intended recipient. For some reason, Caesar always used three as the secret key.

For example, to send the message `war`[2], Caesar would instead send `zdu`. To send `hello`, he would instead send `khoor`.

Thus, Caesar cipher is considered a **shift cipher** where each character is shifted by three. Other integers (modulo 26 to be efficient) would have worked as secret keys.

[1] There is evidence suggesting that Julius Caesar probably did not invent the cipher.

[2] Julius Caesar most likely did not use English characters. However, we use them here for familiarity.

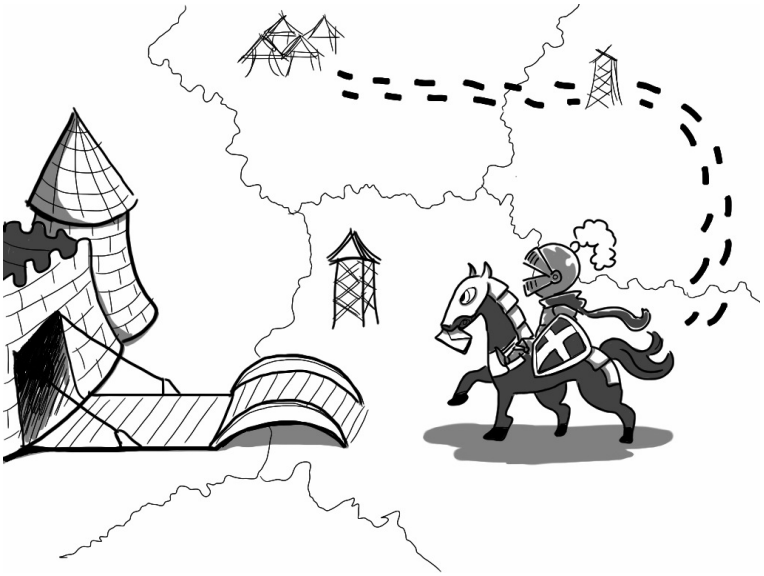


Figure 2.3: A (likely historically inaccurate) depiction of the use of caesar cipher.

Shift ciphers in general suffer from plenty of weaknesses. For example, notice that when a character in the plaintext repeats, so does a character in the ciphertext. If a long piece of exposition is enciphered in this way, one could analyze the frequencies of the characters in the ciphertext to try to glean information about the plaintext. For example, in English, it is well-known that the letter e occurs most frequently. Thus, if one were to analyze a textbook that has been enciphered with a shift cipher, it would be wise to assume that the letter that occurs most frequently in the ciphertext most likely corresponds to the letter e. Counting the distance from e to this character would directly yield a secret key that is likely to be correct. One can then further verify this guess by analyzing the second most frequently occuring character in the ciphertext, and so on.

2.2 One-time pad: a perfect symmetric encryption scheme

Classical ciphers such as shift ciphers have long gone out of favor. A more modern but almost equally well-known way to communicate in private is to use **one-time**

pad. In this scheme, the messages are assumed to be bitstrings. (If they are not, then they are first encoded as such before they get processed by the scheme.) Furthermore, before the sender and the receiver can communicate, they are assumed to already have a shared secret key K which is also a bitstring. Suppose that K is n bits long and that $\oplus$ denotes bitwise exclusive-or. Then, for the sender to send a plaintext M of length n bits, he would instead send $M \oplus K$ as the ciphertext. Upon receiving a ciphertext C, the recipient would compute $C \oplus K$. If C indeed originated from M, the recipient should then end up with $C \oplus K = (M \oplus K) \oplus K = M$ as desired.

One-time pad is known to be a *perfect* encryption scheme since it provides *perfect secrecy*. This means that it cannot be "broken" no matter how much computational resources (e.g. time, memory, money, brain power) the adversary is allowed. Of course, we must first specify what perfect secrecy means. Again, we assume below that the message space is $\{0,1\}^n$ where n is a positive integer.

Definition 2.1: Perfect secrecy

Let n be a positive integer. Let K be a shared secret, and let M and C be random variables denoting the value of the plaintext and that of the ciphertext during the experiment in which a plaintext is first chosen according to some public distribution then gets encrypted thus producing the ciphertext using the shared secret K. A symmetric encryption scheme provides **perfect secrecy** iff, for any $a, b \in \{0,1\}^n$, it is the case that $\Pr[M = a \mid C = b] = \Pr[M = a]$.

In other words, having seen a ciphertext gives one no further information on what the underlying message is. Notice that perfect security does not mean that the adversary knows nothing about the underlying message. Rather, the emphasis is that the adversary's knowledge about the underlying message before and after having seen the ciphertext remains the same. This is to capture the fact that often times some information about the message is known a priori (for example, the message may be an answer to a multiple-choice question), and thus, asking that the adversary knows

nothing of the message would be asking for too much.

We now show that the symmetric encryption scheme OTP, when used appropriately, provides perfect secrecy.

Theorem 2.2: One-time pad provides perfect secrecy.

Let $\mathsf{OTP} = (\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec})$ be the symmetric encryption scheme described above (and defined more formally in Example 5.2.1). Then, OTP encryption provides perfect secrecy.

Proof of Theorem 2.2. Let $a, b \in \{0,1\}^n$. We show that Definition 2.1 holds for OTP. First, we apply Bayes' Theorem to obtain

$$\Pr[M = a \mid C = b] = \frac{\Pr[M = a \wedge C = b]}{\sum_{a \in \{0,1\}^n} \Pr[M = a \wedge C = b]} \tag{2.1}$$

Then, we focus on the quantity $\Pr[M = a \wedge C = b]$ and try to simplify it. Let K be the random variable denoting the value taken on by the shared secret.

$$\begin{aligned} \Pr[M = a \wedge C = b] &= \Pr[M = a \wedge K = (a \oplus b)] \\ &= \Pr[M = a] \cdot \Pr[K = (a \oplus b)] \\ &= \Pr[M = a] \cdot 2^{-n} \end{aligned} \tag{2.2}$$

Substituting Equation (2.2) into Equation (2.1), we obtain

$$\begin{aligned} \Pr[M = a \mid C = b] &= \frac{\Pr[M = a] \cdot 2^{-n}}{\sum_{a \in \{0,1\}^n} \Pr[M = a] \cdot 2^{-n}} \\ &= \frac{\Pr[M = a] \cdot 2^{-n}}{2^{-n} \cdot \sum_{a \in \{0,1\}^n} \Pr[M = a]} \\ &= \Pr[M = a] \end{aligned}$$

as desired. □

Notice, however, that in the experiment in Definition 2.1 only one plaintext is chosen. Thus, Theorem 2.2 says nothing about what would happen if OTP were to be used to encrypt more than one plaintext. In fact, if one were to use OTP as described (and

as stated in Example 5.2.1). Namely, first the secret key K is chosen and then shared between the communicating parties before any real communication occurs. Then, K is used to encrypt more than one plaintext. Suppose two plaintexts M_0 and M_1 are encrypted, we would have that

$$C_0 = K \oplus M_0 \qquad \text{and} \qquad C_1 = K \oplus M_1 \ .$$

Thus,

$$C_0 \oplus C_1 = (K \oplus M_0) \oplus (K \oplus M_1) = M_0 \oplus M_1 \ .$$

Therefore, an adversary who has intercepted two ciphertexts from the communication line (or the air in the case of wireless communication) will learn the value of the bit-wise exclusive-or of the underlying plaintexts corresponding to these two ciphertexts even though this piece of information was not known prior to the communication, thus violating the perfect secrecy property. This kind of information leakage is unacceptable for real communications.

Consequently, if one were to achieve perfect secrecy through one-time pad, one must not reuse the secret key (that is, the secret keys used must be generated independently). Considering this restriction together with the fact that, for each encryption, the number of bits needed for the secret key that must be shared a priori is equal to the length of the plaintext, we can see that perfect secrecy through one-time pad is hard to obtain in practice.

2.2.1 Example: an obvious failure to provide perfect secrecy

Consider an encryption scheme composed of an encryption and a decryption algorithm that does nothing to their inputs. Specifically, suppose the encryption algorithm ignores any secret key provided and simply outputs the plaintext without performing any encryption on it. Similarly, the decryption algorithm ignores the secret key and simply outputs the ciphertext as the decryption. Suppose the message space is $\{0,1\}^{16}$, and suppose that the plaintext distribution is such that all possible messages

are equally likely. We can show that this obviously insecure encryption scheme does not have the perfect secrecy property per Definition 2.1.

First, consider a specific plaintext, say, 0^{16} and a specific ciphertext, say, 1^{16}. Clearly, the conditional probability

$$\Pr\left[M = 0^{16} \mid C = 1^{16}\right] = 0\,.$$

But $\Pr[M = 0^{16}]$ is 2^{-16}. Thus, we have that

$$\Pr\left[M = 0^{16} \mid C = 1^{16}\right] \neq \Pr[M = 0^{16}]\,,$$

and the perfect secrecy property fails to hold.

2.3 Conclusion

We introduce simple symmetric encryption schemes in this chapter. The schemes explored here are the Caesar cipher and one-time pad. Caesar cipher is a simple shift cipher while one-time pad encryption involves simply bitwise exclusive-oring the input message with the keystream. The former is completely insecure while the latter achieves perfect secrecy as long as the keystream is never reused. Although the condition under which one-time pad achieves perfect secrecy makes the scheme impractical, cryptographers have devised ways to get close to perfect secrecy as we will see in Chapter 5.

2.4 Credits

Most of the notations used in this chapter and throughout this book are due to Bellare and Rogaway.

2.5 Exercises

Exercise 2.1. What is One Time Pad? If it cannot be used in practice, why should we care about how it works?

Exercise 2.2. What does it mean to say that an encryption scheme provides "perfect secrecy"? Explain in your own words.

Exercise 2.3. Consider the One-Time Pad encryption. Suppose we encrypt M_0 and M_1 using the same secret key and consequently obtaining $C_0 = 11011010$ and $C_1 = 10100111$, respectively. Suppose we know that $M_1 = 10001001$. Compute M_0. Show your work. Be neat.

Exercise 2.4. What happens when one uses Caesar cipher to encrypt a single message more than once? What are the security implications in this scenario?

Exercise 2.5. Consider an obviously insecure encryption scheme composed of the following encryption and decryption algorithms. Suppose the encryption algorithm works in the same way as the One Time Pad encryption except that the key bits are output along with the ciphertext (say, for the same number of bits).

1. Explain how the decryption algorithm works.
2. Suppose the message space is $\{0,1\}^{16}$, and suppose that the plaintext distribution is such that all possible messages are equally likely. Show that this encryption scheme does not have the perfect secrecy property per Definition 2.1.

Exercise 2.6. Let the distribution of the message space be as follows:

- $\Pr[M = 0^{64}] = \frac{1}{2}$
- $\Pr[M = 1^{64}] = \frac{1}{2}$
- $\Pr[M \neq 0^{64} \wedge M \neq 1^{64}] = 0$

Suppose K is chosen at random from $\{0,1\}^{64}$. Let $C = M \oplus K$. Compute the following probabilities.

1. $\Pr[M = 01^{63}]$
2. $\Pr[C = 0^{64}]$
3. $\Pr[C = 0^{60}1^4]$

Exercise 2.7. Consider the following experiment:

Experiment Exp

$K \xleftarrow{\$} \{0,1\}^{64}$; $M_1 \xleftarrow{\$} \{0,1\}^{64}$; $M_2 \xleftarrow{\$} \{0,1\}^{64}$

$C_1 \leftarrow K \oplus M_1$; $C_2 \leftarrow K \oplus M_2$

Compute the following probabilities.

1. $\Pr[M_1 = M_2]$
2. $\Pr[M_1 \oplus C_1 = M_2 \oplus C_2]$
3. $\Pr\left[M_1 = 0^{64} \mid C_1 = 1^{64}\right]$
4. $\Pr\left[M_1 = 0^{64} \wedge M_2 = 1^{64} \mid C_1 = 0^{63}1 \wedge C_2 = 1^{63}0\right]$
5. $\Pr\left[M_1 = 0^{64} \wedge M_2 = 1^{64} \mid C_1 = 0^{63}1 \wedge C_2 = 110^{62}\right]$

3

Building Block I: Block ciphers

3.1 Motivation

As discussed in the last chapter, classical ciphers have many weaknesses, and the one-time pad encryption scheme is impractical. Therefore, the current state of the art in symmetric encryption relies on a different construct, namely *block ciphers*. In this chapter, we first discuss intuitively what block ciphers are and roughly how they are used for encryption (more detailed discussions on the latter are in the next chapter). Then, we turn our attention to the mathematical objects used to model block ciphers along with the security properties provided by them. Doing so will allow us to be more precise about what is expected of block ciphers.

3.2 What it is

A **block cipher** can be thought of as a pair of deterministic algorithms: one for the "forward" direction and the other for the "backward" direction. We will refer to the former simply as the block cipher and the latter as the inverse block cipher. For both directions, given an input string, it outputs a string that does not appear to have any relationship with the input. Furthermore, the scrambling process (the forward direction) and the unscrambling process (the backward direction) require a secret key. Without the key, it is difficult to compute an output from the input, and vice versa. Figure 3.1 is an admittedly whimsical depiction of a block cipher in the forward direction.

Figure 3.1: A whimsical depiction of what a block cipher does in the forward direction.

Figure 3.2: Encipherment via a block cipher (E, E^{-1}). The left hand side shows the enciphering process while the right hand side shows the deciphering process.

A natural use of block ciphers is to use them for encryption. Suppose that the sender and the receiver have as a shared secret key a bitstring K and agree to use a block cipher (E, E^{-1}) to communicate. In order to encrypt a plaintext M, the sender runs E with K and M as inputs to obtain a ciphertext as the output. In order to decrypt a ciphertext C, the sender runs E^{-1} with K and C as inputs to obtain the plaintext as the output. Figure 3.2 depicts this usage scenario.

With this usage scenario in mind, we see that the desired properties of block ciphers include at least the following:

- Given both inputs, it should be easy to compute the output. The legitimate parties will be performing these actions. Thus, it should be possible for them to do so efficiently.
- If one has only a ciphertext but not the secret key used to generate it, it should be hard to compute the underlying plaintext.

- In fact, if one has only a ciphertext but not the secret key used to generate it, it should be hard to compute any partial information regarding the plaintext. Examples of partial information include the first bit, the last bit, the parity, whether the plaintext can be decoded into a sentence in English, and so on.

As it turns out, block ciphers are also used for other purposes as good block ciphers offer many other additional properties as well.

We emphasize that simply applying a block cipher to an input message is not equivalent to encrypting it. An encryption scheme is much more than a block cipher both in terms of the construction and the security analysis. The term "encipher" and "decipher" are used to emphasize this difference. For many examples of how block ciphers can be used to encrypt messages, see Chapter 5.

Designing a block cipher is not for the faint of heart. The subtleties and difficulties involved make the list of block ciphers being used in practice today a rather short one. The two most commonly used ones are called DES and AES.

3.3 Constructions

3.3.1 Data Encryption Standard (DES): How it works

DES takes two bitstrings, one of length 56 bits and the other of length 64 bits, as inputs and outputs a bitstring of length 64 bits. We give an overview of how DES works here. Further details can be found in the original Federal Information Processing Standards Publication (FIPS PUB) 46 [21]. We first focus on the forward direction. The first input of DES is the key where as the second one is the plaintext. The output is often called the ciphertext. DES operates in rounds, meaning that there is a pattern that gets repeated within the computation of DES as depicted in Figure 3.3. First, the plaintext gets permuted through a fixed table called the *initial permutation* (IP) shown in Figure 3.4(a). The output obtained after the permutation then goes through 16 rounds of the so-called **feistel cipher**, which is a particular structure commonly

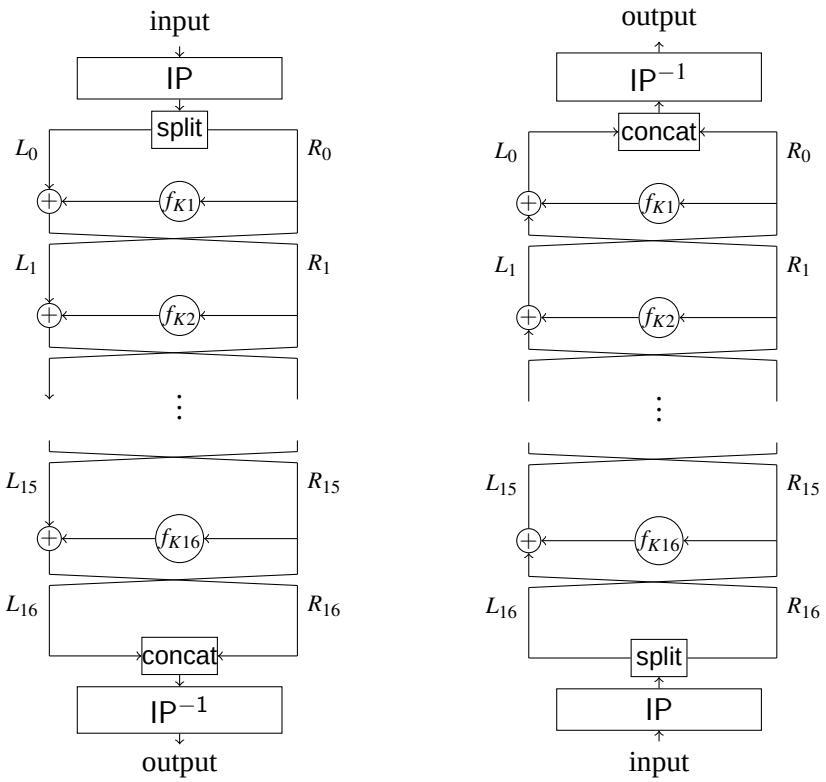


Figure 3.3: The structure of DES. The computation in the "forward" and "backward" directions are shown on the left and the right, respectively. The input and the output are each 64 bits long. The key is 56 bits long but gets expanded by the key scheduling algorithm (not depicted here) into 16 subkeys $K1, \ldots, K16$. Given a bitstring as input, split returns two bitstrings of equal length. Given two bitstrings as input, concat concatenates them in the order in which they are received. We write f_K to denote that the first input of f is K.

used in block cipher design. The bitstring that emerges after the last round then goes through another fixed table called the *inverse initial permutation* (IP^{-1}), shown in Figure 3.4(b), yielding the final output. Within each round, the input is divided into two halves of equal lengths. Two types of computation are performed in each round: an xor and the **cipher function** (f). The cipher function takes two inputs: the right half of the input of the current round and the 48-bit **subkey** for the current round. The latter is computed from the original 56-bit secret key through a process called **key scheduling** which in effect expands the 56-bit secret key into 16 subkeys each of length 48 bits.

58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

(a) IP

40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

(b) IP^{-1}

Figure 3.4: The initial permutation and the inverse initial permutation. If a number i appears in the j-th position in the table, then this means that the bit in the i-th position of the input should appear in the j-th position in the output. For example, the table IP specifies that the 58-th bit in the input should appear as the first bit of the output. Thus, the numbers in the tables are integers between 1 and 64, inclusive. Notice that since both tables are permutations, the numbers in the table do not repeat. In addition, the two tables are inverses of each other.

The right hand side of Figure 3.3 depicts the computation in the backward direction of DES. Notice that, although the algorithms embodied by the left and the right hand side of Figure 3.3 are different, the components needed to execute them are the same. This reflects the fact that one of the design criteria for DES is to be efficient (in terms of both time and space) when implemented in hardware.

3.3.2 Complementation property of DES

It is well known that the following property holds true for DES. For any $K \in \{0,1\}^{56}$ and any $M \in \{0,1\}^{64}$, it is the case that

$$C = \mathsf{DES}_K(M) \quad \text{if and only if} \quad \overline{C} = \mathsf{DES}_{\overline{K}}(\overline{M}) \; .$$

This property can be exploited in an exhaustive key search attack against DES to reduce the amount of work required by a half.

3.3.3 DES weak and semi-weak keys

DES has four so-called *weak keys*. These are the keys that give rise to the following undesirable property: using any of the weak keys in the forward direction once on any plaintext and again on the result yields the original plaintext. These keys are 0x0101010101010101, 0xFEFEFEFEFEFEFEFE, 0xE0E0E0E0F1F1F1F1, and 0x1F1F1F1F0E0E0E0E. For K being any one of these weak keys, it is the case that for any $M \in \{0,1\}^{64}$,

$$\mathsf{DES}_K(\mathsf{DES}_K(M)) = M\ .$$

DES also has six pairs of the so-called *semi-weak keys*. These are the keys that give rise to the following undesirable property: using one in the forward direction is the same as using its pair in the backward direction. These pairs are

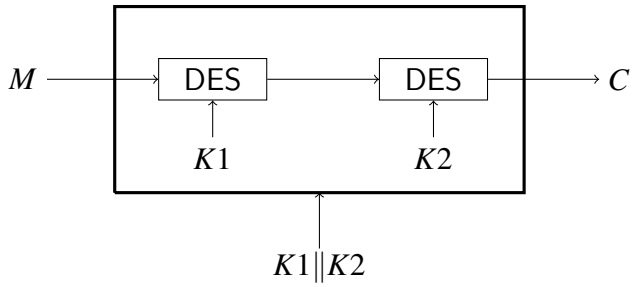
- 0x011F011F010E010E and 0x1F011F010E010E01
- 0x01E001E001F101F1 and 0xE001E001F101F101
- 0x01FE01FE01FE01FE and 0xFE01FE01FE01FE01
- 0x1FE01FE00EF10EF1 and 0xE01FE01FF10EF10E
- 0x1FFE1FFE0EFE0EFE and 0xFE1FFE1FFE0EFE0E
- 0xE0FEE0FEF1FEF1FE and 0xFEE0FEE0FEF1FEF1.

For each semi-weak key pair $(K1, K2)$, it is the case that

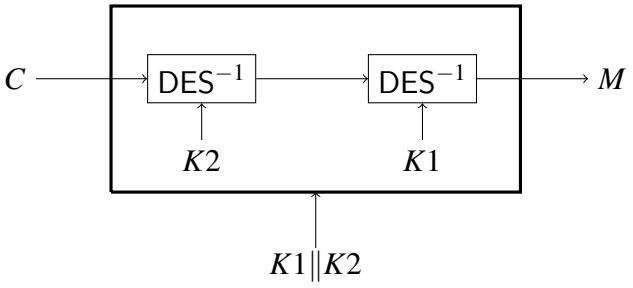
$$\mathsf{DES}_{K1} = \mathsf{DES}_{K2}^{-1}\ .$$

3.3.4 Double and Triple DES

Attempts have been made to extend the effective key length for DES, resulting in the double DES and triple DES constructions. Double DES, commonly denoted 2DES,



(a) forward double DES computation



(b) backward double DES computation

Figure 3.5: The forward and backward double DES computation. The thick frames emphasize the fact that the end result is a single block cipher with the forward and backward computation as illustrated. Here, the strings $K1$ and $K2$ are of length 64 bits each.

computes the forward direction as follows. For any 112-bit key $K1\|K2$ and any 64-bit plaintext M,

$$2\mathsf{DES}_{K1\|K2}(M) = \mathsf{DES}_{K2}(\mathsf{DES}_{K1}(M)) \ .$$

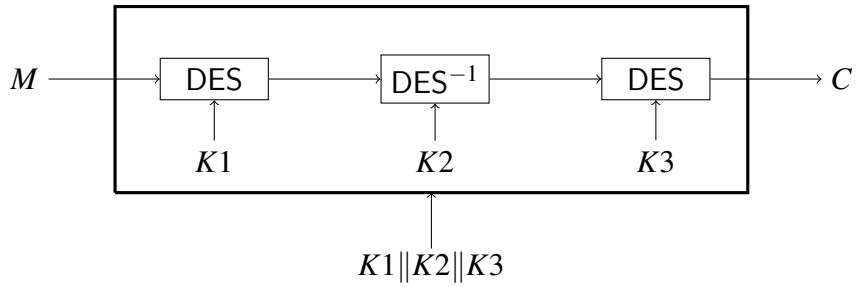
Figure 3.5 illustrates the computation in both directions of double DES.

Similarly, triple DES, commonly denoted 3DES, computes the forward direction as follows [36]. For any 168-bit key $K1\|K2$ and any 64-bit plaintext M,

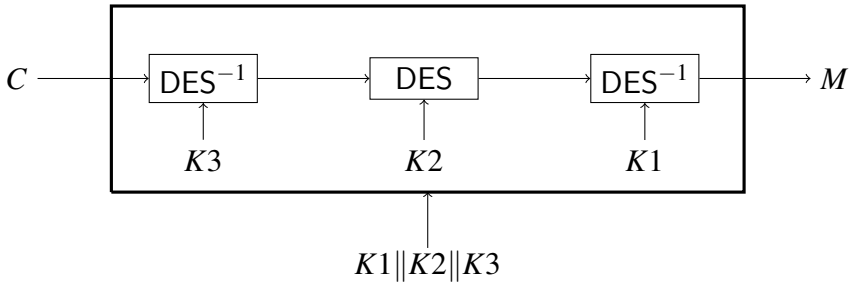
$$3\mathsf{DES}_{K1\|K2\|K3}(M) = \mathsf{DES}_{K3}(\mathsf{DES}^{-1}_{K2}(\mathsf{DES}_{K1}(M))) \ .$$

Figure 3.6 illustrates the computation in both directions of triple DES.

The NIST standard specifies two keying options for triple DES. For option 1, the bitstrings $K1, K2, K3$ are "unique" keys. Specifically, this means that $K1 \neq K2, K2 \neq K3$, and $K3 \neq K1$. For option 2, only the bitstrings $K1$ and $K2$ are required to be different. Suppose we denote the option 2 construction as 3DES$'$. The



(a) Forward triple DES computation



(b) backward triple DES computation

Figure 3.6: The forward and backward triple DES computation. The thick frames emphasize the fact that the end result is a single block cipher with the forward and backward computation as illustrated. Here, the strings $K1, K2$, and $K3$ are all of length 64 bits each.

forward direction of $3\mathsf{DES}'$ is then computed as follows. For any 112-bit key $K1\|K2$ and any 64-bit plaintext M,

$$3\mathsf{DES}'_{K1\|K2}(M) = \mathsf{DES}_{K1}(\mathsf{DES}^{-1}_{K2}(\mathsf{DES}_{K1}(M))) \; .$$

for any 112-bit key $K1\|K2$ and any 64-bit plaintext M.

Remark 3.3:

We find the requirement that certain parts of the key be different from the other problematic. This is because, in security definitions, the experiments are almost always defined so that the secret keys are chosen uniformly at random. Clearly, choosing a 112-bit string completely at random does not ensure that its first 56 bits and second 56 bits will be different. Consequently, strictly speaking, security results obtained with respect to the de facto security definitions (namely, the

notions of pseudorandom functions and pseudorandom permutations studied extensively in Sections 3.4.5 and 3.4.6) do not apply to the triple DES construction as defined in the NIST standard.

3.3.5 DESX

Another approach used to strengthen DES is to do the following:

1. lengthen the secret key by 128 additional bits,
2. split the secret key into three pieces of length 64, 56, and 64,
3. exclusive-or the first piece with the input string before applying DES to the result,
4. finally exclusive-or the output with the last piece.

The exclusive-or computation before and after the block cipher application is often called "whitening." This construction is known as DESX and was designed by Ronald Rivest.

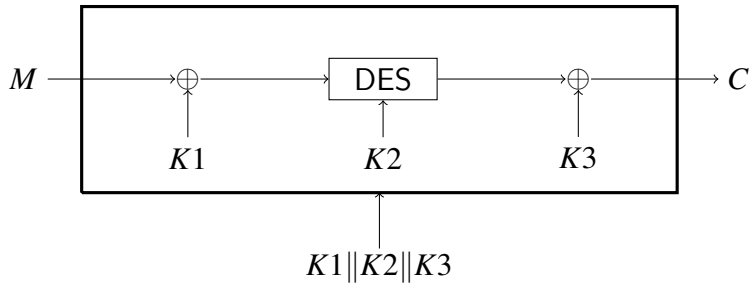
More precisely, DESX works as follows. Let $K1\|K2\|K3$ be the secret key of length 64, 56, and 64 each, respectively. Then, the forward computation can be summarized as follows: for any 64-bit input string M,

$$\mathsf{DESX}_{K1\|K2\|K3}(M) = K1 \oplus \mathsf{DES}_{K2}(M) \oplus K3 \; .$$

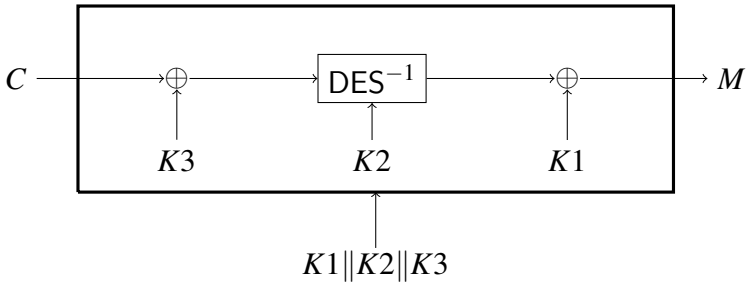
Figure 3.7 illustrates both the forward and backward computation of DESX.

3.3.6 The obsolescence of DES

There were many reasons the community of cryptographers and security experts felt the need to come up with a replacement for DES, which was invented in the 70s. One reason is that the key size for DES is too small. Recall that DES has key size of 56 bits and block size of 64 bits. With the current state of the art, it is possible to exhaustively search for a secret key in a relatively short time. To do so, we can obtain a certain number of input-output pairs and try computing DES on the inputs using all



(a) forward DESX computation



(b) backward DESX computation

Figure 3.7: The forward and backward DESX computation. The thick frames emphasize the fact that the end result is a single block cipher with the forward and backward computation as illustrated. Here, the strings $K1, K2$, and $K3$ are of length 64, 56, and 64 bits, respectively.

the possible keys until we find one that is consistent with the input-output pairs that we have. Another important reason for replacing DES is that the block size for DES is too short. Why short block size leads to vulnerability will become clear after we explore the security definition for block ciphers.

3.3.7 Advanced Encryption Standard (AES)

The replacement for DES came in 2002. The Advanced Encryption Standard (AES) is a result of a contest for a block cipher to replace DES. The winner of the contest was a block cipher named Rijndael. It is parameterizable. The standard specifies the block ciphers AES-128, AES-192, and AES-256 where the numbers specify, for each block cipher, the numbers of bits of an input (which is also equal to the numbers of bits of a secret key).

We give an overview of AES-128 here. Further details can be found in FIPS PUB 197 [1]. Like DES, AES operates in rounds. Unlike DES, AES has a clear algebraic structure. For simplicity, however, we give an overview of AES here in terms of simple bitstring manipulation rather than as algebraic computation. For a description of the latter kind, see for example [41].

There are a total of 11 rounds in AES-128. The first and the last rounds consist of fewer operations than the middle 9 rounds. Similar to DES, given two bitstrings, the input and the secret key (although in the case of AES they are of equal length, specifically 128 bits for AES-128), the computation first processes the secret key through key scheduling to produce 11 subkeys each of length 128 bits then processes the input through the following operations in order: SubBytes, ShiftRows, MixColumns, and AddRoundKey. First, the input is written as a sequence of bytes which are then arranged into a 4x4 array and can be viewed as a matrix. While the input is processed through the rounds, it is referred to as the **state array**. During the first round, only AddRoundKey is applied to the state array. During the last round, only SubBytes, ShiftRows, and AddRoundKey are performed. During the middle 9 rounds, all 4 operations are performed. We describe each of the operation only briefly here.

SubBytes: The bytes in the input get substituted via a table lookup. This table is fixed and is often referred to as the **s-box**.

ShiftRows: Each row of state array is shifted by a different number of positions depending on the row number. Specifically, the first row is not shifted at all (i.e. zero positions). The second row is shifted to the left one byte. (By shifted, we mean that, starting from the left most byte, the first byte becomes the last byte, and all the bytes in the middle move to the left by one byte.) The third and the fourth rows are then shifted to the left two and three bytes, respectively.

MixColumns: Each column of the state array is multiplied by a fixed matrix then replaced by the result of the multiplication.

AddRoundKey: The state array is bit-wise exclusive-ored with the subkey for the

current round.

The key scheduling process also involves simple operations such as byte shifting, byte substitution, and bit-wise exclusive-or. See [1] for details.

3.4 What security means

3.4.1 A close look at families of functions and families of permutations

Since a block cipher is modelled as a family of permutations, we start by investigating what a family of permutations looks like. We begin with families of functions. Recall that a function f mapping from a domain A to a co-domain B has the signature

$$f\colon\ A \to B\ .$$

Consider a function F mapping from the domain $\{0,1\} \times \{0,1\}$ to the co-domain $\{0,1\}$. The signature of F is then

$$F\colon\ \{0,1\} \times \{0,1\} \to \{0,1\}\ .$$

We can write out all possible functions that has this signature. There are 16 of them in all as shown in Figure 3.8.

Another way to view the group or *family* of 16 functions with the signature $\{0,1\} \times \{0,1\} \to \{0,1\}$ is to first fix the first component of the input, call it, say, K, then consider the resulting function from $\{0,1\}$ to $\{0,1\}$ as being indexed by K. Figure 3.9 depicts this view.

Consider, for instance, the family f_8 shown in Figure 3.9. Indexed by $K = 0$, the family f_8 yields the function that maps 0 to 1 and 1 to 0. Indexed by $K = 1$, the family yields the function that maps both 0 and 1 to 0.

Note that, in order to be explicit, we have written both the input and output in full so far. For example, in Figure 3.8, f_8 is written as

x	f_0(x)
(0,0)	0
(0,1)	0
(1,0)	0
(1,1)	0

x	f_1(x)
(0,0)	0
(0,1)	0
(1,0)	0
(1,1)	1

x	f_2(x)
(0,0)	0
(0,1)	0
(1,0)	1
(1,1)	0

x	f_3(x)
(0,0)	0
(0,1)	0
(1,0)	1
(1,1)	1

x	f_4(x)
(0,0)	0
(0,1)	1
(1,0)	0
(1,1)	0

x	f_5(x)
(0,0)	0
(0,1)	1
(1,0)	0
(1,1)	1

x	f_6(x)
(0,0)	0
(0,1)	1
(1,0)	1
(1,1)	0

x	f_7(x)
(0,0)	0
(0,1)	1
(1,0)	1
(1,1)	1

x	f_8(x)
(0,0)	1
(0,1)	0
(1,0)	0
(1,1)	0

x	f_9(x)
(0,0)	1
(0,1)	0
(1,0)	0
(1,1)	1

x	f_{10}(x)
(0,0)	1
(0,1)	0
(1,0)	1
(1,1)	0

x	f_{11}(x)
(0,0)	1
(0,1)	0
(1,0)	1
(1,1)	1

x	f_{12}(x)
(0,0)	1
(0,1)	1
(1,0)	0
(1,1)	0

x	f_{13}(x)
(0,0)	1
(0,1)	1
(1,0)	0
(1,1)	1

x	f_{14}(x)
(0,0)	1
(0,1)	1
(1,0)	1
(1,1)	0

x	f_{15}(x)
(0,0)	1
(0,1)	1
(1,0)	1
(1,1)	1

Figure 3.8: All possible functions with the signature $\{0,1\} \times \{0,1\} \to \{0,1\}$.

x	f_8(x)
(0,0)	1
(0,1)	0
(1,0)	0
(1,1)	0

However, if we view the two input bits as representing a binary number then list the output bit in, say, increasing order of the inputs, then we could simply write $(1,0,0,0)$ to represent f_8. Using this convention, a function f: $\{0,1\}^1 \times \{0,1\}^2 \to \{0,1\}^2$ that is written as $(10,00,11,10,11,01,11,00)$ is one that maps $(1,00)$ to 11.

Returning our attention to the families of functions with the signature $\{0,1\} \times \{0,1\} \to \{0,1\}$. Notice that, fixing, say, $K = 0$, the family f_0 maps both 0 and 1 to

K = 0		**K = 1**			**K = 0**		**K = 1**		
x	y	x	y		x	y	x	y	
0	0	0	0	$\Longleftarrow f_0$	0	1	0	0	$\Longleftarrow f_8$
1	0	1	0		1	0	1	0	
x	y	x	y		x	y	x	y	
0	0	0	0	$\Longleftarrow f_1$	0	1	0	0	$\Longleftarrow f_9$
1	0	1	1		1	0	1	1	
x	y	x	y		x	y	x	y	
0	0	0	1	$\Longleftarrow f_2$	0	1	0	1	$\Longleftarrow f_{10}$
1	0	1	0		1	0	1	0	
x	y	x	y		x	y	x	y	
0	0	0	1	$\Longleftarrow f_3$	0	1	0	1	$\Longleftarrow f_{11}$
1	0	1	1		1	0	1	1	
x	y	x	y		x	y	x	y	
0	0	0	0	$\Longleftarrow f_4$	0	1	0	0	$\Longleftarrow f_{12}$
1	1	1	0		1	1	1	0	
x	y	x	y		x	y	x	y	
0	0	0	0	$\Longleftarrow f_5$	0	1	0	0	$\Longleftarrow f_{13}$
1	1	1	1		1	1	1	1	
x	y	x	y		x	y	x	y	
0	0	0	1	$\Longleftarrow f_6$	0	1	0	1	$\Longleftarrow f_{14}$
1	1	1	0		1	1	1	0	
x	y	x	y		x	y	x	y	
0	0	0	1	$\Longleftarrow f_7$	0	1	0	1	$\Longleftarrow f_{15}$
1	1	1	1		1	1	1	1	

Figure 3.9: All possible functions with the signature $\{0,1\} \times \{0,1\} \to \{0,1\}$ now viewed as being indexed by the first component of the input.

0, thus is not a permutation. (Recall that a permutation over a set is a bijection from the set to itself.) The families f_1 through f_3 behave the same way. Similarly, fixing $K = 0$, the families f_{12} through f_{15} map both 0 and 1 to 1.

Definition 3.4: Family of Permutations

Let k and n be positive integers. A family of functions with the signature $\{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$ is considered to be a *family of permutations* if and only if, for all $K \in \{0,1\}^k$, the function indexed by K is a permutation.

Thus, among the families f_0 through f_{15} from Figure 3.8, the only families that are families of permutations are f_5, f_6, f_9, and f_{10}.

3.4.2 Modelling block ciphers as pseudorandom permutation families

A block cipher is modelled as a family of permutations. One chooses a particular permutation out of the family by specifying the secret key. For concreteness, we consider DES as an example. The key and the input are 56 and 64 bits long, respectively. Fixing the secret key to, say, 0^{56}, one ends up with a permutation mapping 64-bit inputs to 64-bit outputs. We can write the signature of DES as

$$\mathsf{DES} : \{0,1\}^{56} \times \{0,1\}^{64} \to \{0,1\}^{64} .$$

The simplest, albiet insecure, use of block ciphers for encryption is as follows. Suppose the block cipher being used is $E : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$ where k is the length in bits of each key and l is the length in bits of each input and output. Respectively, these lengths are often referred to as the **key size** and the **block size** of the block cipher. First, the communicating parties agree on a secret key K chosen at random from the set of all possible keys. Then, when the sender wants to send an input x, he sends instead $y = E(K,x)$. Often, we write this as $y = E_K(x)$ to stress the fact that the key K is picked first, then the permutation indexed by K is used on many

inputs. We also write $E_K(\cdot)$ to denote the permutation indexed by K and $E_K^{-1}(\cdot)$ to denote the inverse of the permutation indexed by K.

3.4.3 Example: A block cipher with small parameters

Consider a block cipher E: $\{0,1\}^2 \times \{0,1\}^2 \to \{0,1\}^2$ specified as follows:

$$E_{00} = (11,10,01,00) \qquad E_{01} = (01,11,00,10)$$
$$E_{10} = (01,10,11,00) \qquad E_{11} = (10,01,11,00)$$

Then, for the decipherment to work, the inverse block cipher E^{-1}: $\{0,1\}^2 \times \{0,1\}^2 \to \{0,1\}^2$ must be

$$E_{00}^{-1} = (11,10,01,00) \qquad E_{01}^{-1} = (10,00,11,01)$$
$$E_{10}^{-1} = (11,00,01,10) \qquad E_{11}^{-1} = (11,01,00,10)$$

Suppose parties A and B have agreed on 01 as the secret key. The party A would encipher an input 01 as 11. The party B would decipher 11 as 01.

Suppose an adversary is somehow able to obtain a pair of message and the corresponding ciphertext as $(10,01)$, then it would know without a doubt that the underlying secret key must be 00. But if the message-ciphertext pair is $(01,10)$, then it would know only that the underlying secret key is either 00 or 10 but would not be able to tell which is the one used by A and B.

3.4.4 All possible permutations and functions

Fundamental to the framework in which block cipher security is defined is the concept of choosing a permutation at random out of all possible permutations. We investigate this and a closely related concept in this section.

We denote, for the rest of this book, by $\mathsf{Perm}(n)$ the set of all possible permutations over the set of n-bit strings. Consider, for example, $\mathsf{Perm}(2)$, the permutations over the set of 2-bit strings. We list them all here using the notation discussed in Section 3.4.1.

(00,01,10,11)	(01,00,10,11)	(10,00,01,11)	(11,00,01,10)
(00,01,11,10)	(01,00,11,10)	(10,00,11,01)	(11,00,10,01)
(00,10,01,11)	(01,10,00,11)	(10,01,00,11)	(11,01,00,10)
(00,10,11,01)	(01,10,11,00)	(10,01,11,00)	(11,01,10,00)
(00,11,01,10)	(01,11,00,10)	(10,11,00,01)	(11,10,00,01)
(00,11,10,01)	(01,11,10,00)	(10,11,01,00)	(11,10,01,00)

Similarly, we also denote by $\mathsf{Func}(m,n)$ the set of all possible functions from $\{0,1\}^m$ to $\{0,1\}^n$. For example, consider $\mathsf{Func}(2,1)$, the functions from the set of 2-bit strings to a bit. We list them all here for clarity.

(0,0,0,0)	(0,0,0,1)	(0,0,1,0)	(0,0,1,1)
(0,1,0,0)	(0,1,0,1)	(0,1,1,0)	(0,1,1,1)
(1,0,0,0)	(1,0,0,1)	(1,0,1,0)	(1,0,1,1)
(1,1,0,0)	(1,1,0,1)	(1,1,1,0)	(1,1,1,1)

3.4.5 Security definitions for block ciphers

In modern cryptography, one does not merely look to intuition or experience to decide whether a construct, for example a symmetric encryption scheme or a block cipher, is secure. Rather, one relies on carefully-formulated, well-accepted security definitions and rigorous analyses of whether there is an attack that uses "reasonable" amount of "resources" yet achieve high probability of success against the construct in question under the experiments defined as part of the relevant security definitions. (The term in quotes will be discussed further later on in this chapter.) If there is no such attack, one strive to craft a *reductionist argument* to vouch for the security of the construct in question based on some (computational) hardness assumption.

The experiments under which one uses to analyze the security of a construct are often referred to as "games." Typically, a game takes on the following structure. Some setup steps are carried out. Then, an adversary is invoked. Often, the adversary can interact with some component(s) during the game. By the end of the game, the

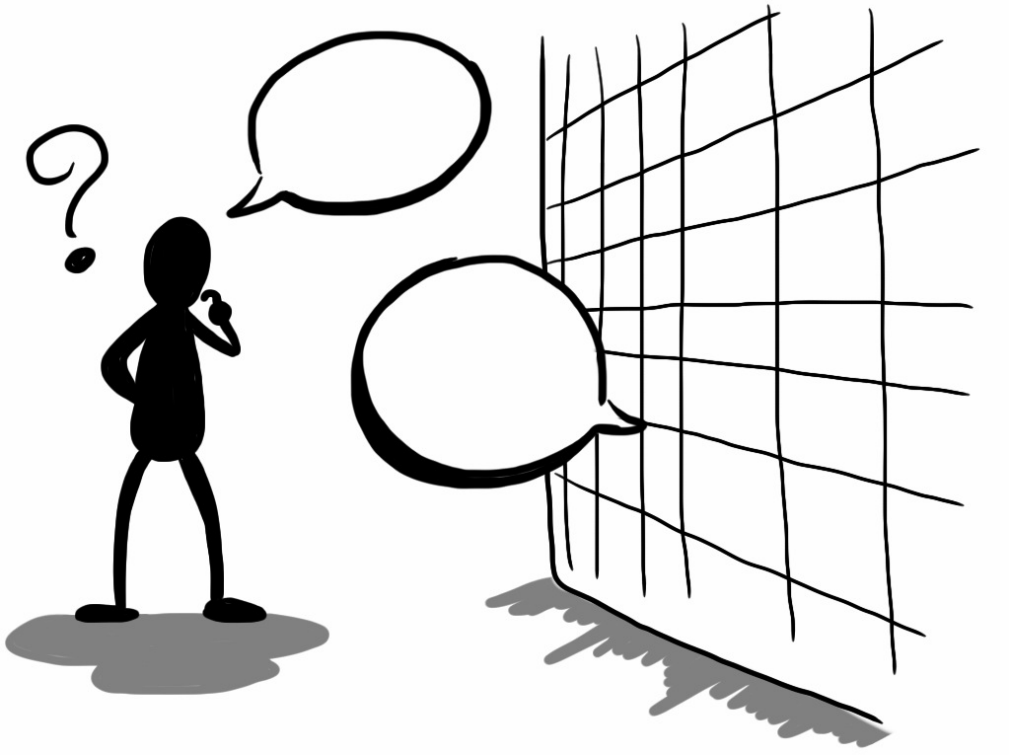


Figure 3.10: A typical experiment for security definitions in modern cryptography.

adversary is supposed to beat the game by accomplishing some well-defined goal. A typical goal is to guess a mysterious bit or to come up with a piece of data that will pass a certain test without having known any secret. Figure 3.10 illustrates this concept.

Let us now turn our attention back to block ciphers. Since block ciphers are used for many purposes, there are many security properties that they are expected to deliver. The consensus is that the property that would imply all the relevant properties is the following: for a good block cipher, once a secret key is chosen at random from the set of all possible keys, the permutation indexed by the key should behave like a "random permutation," i.e. any given input should be mapped to a "random" output that has not been mapped to any other input.

Imagine the following experiment depicted in Figure 3.11. A boy sits in front of a wall with an input device and a screen. (Admittedly, the figure depicts a rather gender and age neutral individual. We choose a specific gender and age range for

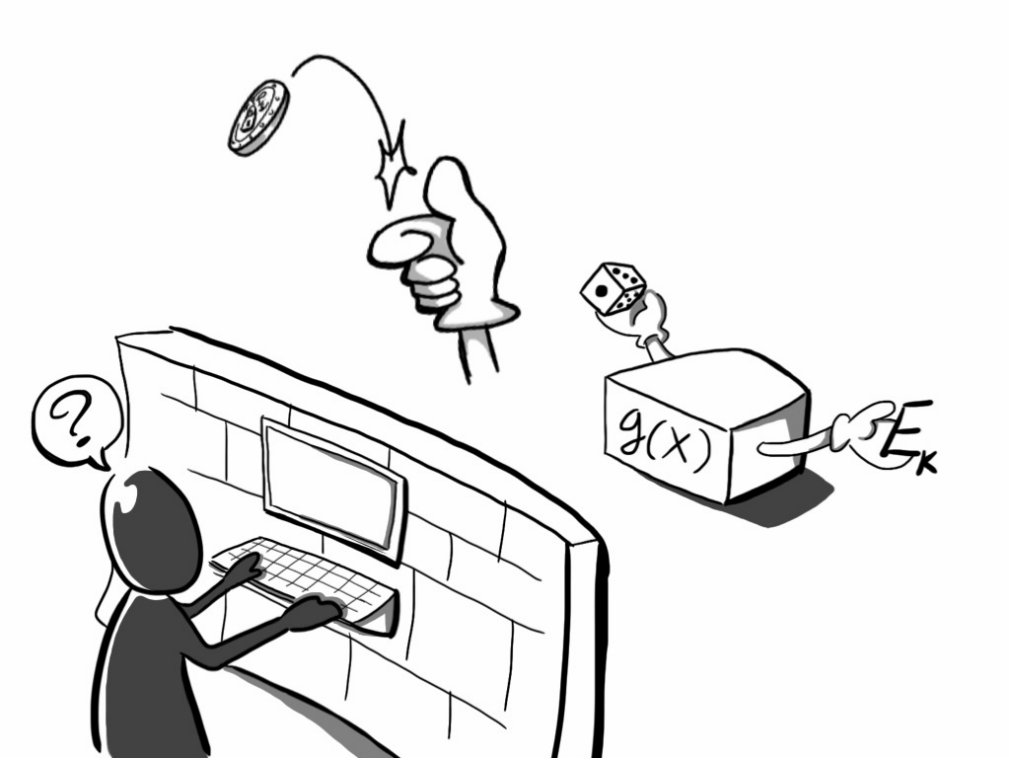


Figure 3.11: An experiment for the PRP-CPA security definition for block ciphers. The behavior of the permutation g depends on whether the coin flip in the experiment results in a head or a tail.

brevity in our discussions.) The input device is for him to submit an input value while the screen is for him to receive the corresponding output. Behind the wall sits a box. The experiment begins with a fair coin being flipped.

If the coin comes up heads, then the experiment continues as follows. A secret key is chosen and is fixed throughout the rest of the experiment. The box then computes the output of a block cipher given the secret key and an input submitted by the boy. The output is then shown on the screen. The boy may submit many inputs and consequently receive many corresponding outputs.

If the coin comes up tails, then a permutation is chosen at random from all possible permutations (of strings of a certain length).

If the block cipher is a good one, the boy should have a hard time telling whether the coin had initially come up heads or tails at the beginning of the experiment. The

underlying idea is that a good block cipher, with a fixed but initially randomly chosen key, should behave like a permutation chosen at random from all possible permutations (of strings of a certain length).

Note that in the case that the coin comes up tails, if there are too many possible permutations to choose from, it may be computationally difficult to actually choose one at random. For example, consider permutations over the set of 64-bit strings. The number of all such permutations is the number of ways we can order 2^{64} bitstrings in a sequence. Thus, there are $2^{64}!$ such permutations. Directly choosing one at random out of such a large set is difficult. (Consider, for example, the task of storing a single sequence of 2^{64} elements each of length 64 bits. This requires a lot of space already.) Instead, we dynamically build a table as the boy submits his queries. If he submits an input x that he has not submitted before, we pick 64 bits at random, record the resulting sequence of bits in our table to remember that it is the output corresponding to the input x, and return it as the output. If the boy submits an input that he has submitted before, we simply lookup x in the table and return the output value previously recorded there. This table has the size proportional to the total number of queries that the boy submits during the experiment.

The following definition specifies this experiment more precisely and succinctly. We write $x \xleftarrow{\$} S$ to denote that an element is chosen at random from the set S and then assigned to x. Also, an **oracle** is simply an algorithm that can be invoked by other algorithms. It internally maintains its own states, if any.

Definition 3.5: Security notion for block ciphers: PseudoRamdom family of Permutations secure against Chosen-Plaintext Attacks (PRP-CPA)

Let k, n be positive integers, and let $E : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$ be a family of permutations. Let A be an adversary with access to an oracle. We define the following subroutines, experiment, and advantage function.

Subroutines

Subroutine `Initialize`

$b \xleftarrow{\$} \{0,1\}$

If $b = 1$ then $K \xleftarrow{\$} \{0,1\}^k$ else $p \xleftarrow{\$} \mathsf{Perm}(n)$

Subroutine $\mathtt{g}(x)$

If $b = 1$ then return $E_K(x)$ else return $p(x)$

Subroutine $\mathtt{Finalize}(d)$

Return (d = b)

Experiment

Experiment $\mathbf{Exp}_E^{\text{prp-cpa}}(A)$

`Initialize`

$d \xleftarrow{\$} A^{\mathtt{g}}$

Return $\mathtt{Finalize}(d)$

We define the **prp-cpa advantage** of an adversary A mounting a chosen-plaintext attack against E as

$$\mathbf{Adv}_E^{\text{prp-cpa}}(A) = 2 \cdot \Pr[\mathbf{Exp}_E^{\text{prp-cpa}}(A) \Rightarrow \text{true}] - 1 \ . \tag{3.1}$$

In this definition, the bit b is the coin. An adversary is the boy. The oracle g is the box. The boy tries to figure out what is behind the wall by interacting with the box. The analogs between the definitional components and the components in the boy-and-box game for chosen-plaintext attacks can be summarized as follows. All variables in the table are as defined in Definition 3.5.

definitional component	interpretation
adversary A	the boy
oracle g	the box
challenge bit b	the result of the initial coin toss
guess bit d	the boy's guess as to what the result of the initial coin toss was

When mounting a chosen-ciphertext attack, the boy is given two boxes, one for the permutation g and the other for the inverse permutation g^{-1}. The boy tries to figure out what is behind the wall by interacting with the box. See Figure 3.12 for a depiction and Definition 3.6 for details. The analogs between the definitional components and the components in the boy-and-box game for chosen-ciphertext attacks can be summarized as follows. All variables in the table are as defined in Definition 3.6.

definitional component	interpretation
adversary A	the boy
oracle $\mathsf{g}, \mathsf{g}^{-1}$	the boxes
challenge bit b	the result of the initial coin toss
guess bit d	the boy's guess as to what the result of the initial coin toss was

Definition 3.6: Security notion for block ciphers: PseudoRamdom family of Permutations secure against Chosen-Ciphertext Attacks (PRP-CCA)

Let k, n be positive integers, and let $E : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$ be a family of permutations. Let A be an adversary with access to two oracles. We define the following subroutines, experiment, and advantage function. Here, the permutation p^{-1} denotes the inverse permutation of the permutation p.

Figure 3.12: An experiment for the PRP-CCA security definition for block ciphers. The behaviors of the permutations g and g^{-1} depend on whether the coin flip in the experiment results in a head or a tail.

Subroutines

Subroutine `Initialize`

$b \xleftarrow{\$} \{0,1\}$

If $b = 1$ then $K \xleftarrow{\$} \{0,1\}^k$ else $p \xleftarrow{\$} \mathsf{Perm}(n)$

Subroutine $\mathsf{g}(x)$

If $b = 1$ then return $E_K(x)$ else return $p(x)$

Subroutine $\mathsf{g}^{-1}(x)$

If $b = 1$ then return $E_K^{-1}(x)$ else return $p^{-1}(x)$

Subroutine `Finalize`(d)

Return (d = b)

Experiment

Experiment $\mathbf{Exp}_E^{\text{prp-cca}}(A)$

`Initialize`

$d \xleftarrow{\$} A^{\mathsf{g},\mathsf{g}^{-1}}$

Return `Finalize`(d)

We define the **prp-cca advantage** of an adversary A mounting a chosen-ciphertext attack against E as

$$\mathbf{Adv}_E^{\text{prp-cca}}(A) = 2 \cdot \Pr[\mathbf{Exp}_E^{\text{prp-cca}}(A) \Rightarrow \mathsf{true}] - 1 \ . \tag{3.2}$$

An advantage function measures how easy it is to tell whether the oracle g is a permutation randomly chosen from the block cipher or from the set of all possible permutations over the set $\{0,1\}^n$. The closer the advantage is to one, the easier it is, and the less secure the block cipher is considered to be.

In Definitions 3.5 and 3.6, the acronym "prp" stands for the term **pseudorandom permutation**. In particular, a family of permutations that is secure under one or both of these definitions is often referred to as a secure pseudorandom permutation family. We stress that, although we use the term *secure*, it does not have an absolute meaning. Rather, there is a degree of security of a block cipher as measured by the advantage of any adversary with a "reasonable" amount of resources. A secure block cipher is one that, for any adversary A whose resource usage is reasonable (polynomially-related to the key size, the block size, and the time it takes to compute an output of a block cipher given the inputs, for example), the advantage of A is small.

Remark 3.7:

We comment further on the intuition behind the main components of the PRP-CPA and PRP-CCA notions here. Let $E\colon \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$ be the

block cipher whose security level is being analyzed. Recall that in the games in Definitions 3.5 and 3.6, the adversary tries to tell whether it is being given access to a permutation chosen at random from the family of permutations defined by the block cipher or to a permutation chosen at random from all possible permutations mapping n bits to n bits.

Consider for example a block cipher with the same block size and key size as DES. Specifically, let E: $\{0,1\}^{56} \times \{0,1\}^{64} \rightarrow \{0,1\}^{64}$ be the block cipher in question. Consider all possible permutations mapping 64 bits to 64 bits. There are a total of $2^{64}!$ such permutations. Suppose that E is a family of 2^{56} distinct permutations (each indexed by a 56-bit key). Clearly, $2^{64}!$ is much much larger than 2^{56}. Yet, if E is a PRP-CPA or PRP-CCA secure block cipher, it should be difficult for an adversary to tell whether it is interacting with a permutation chosen at random from the huge set of $2^{64}!$ members or a permutation chosen from the much smaller set of 2^{56} members as dictated by the specification of the block cipher E.

3.4.6 A related primitive: pseudorandom function family

A block cipher is a family of permutations. A closely-related concept is that of a family of functions. For certain applications (see Chapter 7, for example), we only need the latter. We present the security definition for pseudorandom functions, which is analogous to that for pseudorandom permutations except of course that the object of interest is a family of functions rather than a family of permutations. Given the close similarity between the PRP-CPA and the PRF definitions, Figure 3.11 can also be viewed as a depiction of the experiment for the PRF security definition.

In particular, a bit is chosen at the beginning of the game. If this challenge bit is one, then a function is chosen at random from the family of functions being analyzed. Otherwise, a function is chosen at random from all possible functions mapping appropriate bitstrings to bitstrings of appropriate lengths. The attacker's goal is to tell

whether the challenge bit is one or zero. It is allowed to submit inputs to see corresponding outputs during the game.

Definition 3.8: Security notions for block ciphers: PseudoRamdom family of Functions (PRF)

Let k, n be positive integers, and let $F : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$ be a family of permutations. Let A be an adversary with access to an oracle. We define the following subroutines, experiment, and advantage function.

Subroutines

Subroutine `Initialize`

$b \xleftarrow{\$} \{0,1\}$

If $b = 1$ then $K \xleftarrow{\$} \{0,1\}^k$ else $f \xleftarrow{\$} \mathsf{Func}(m,n)$

Subroutine $\mathtt{g}(x)$

If $b = 1$ then return $F_K(x)$ else return $f(x)$

Subroutine $\mathtt{Finalize}(d)$

Return (d = b)

Experiment

Experiment $\mathbf{Exp}^{\mathrm{prf}}_F(A)$

`Initialize`

$d \xleftarrow{\$} A^{\mathtt{g}}$

Return $\mathtt{Finalize}(d)$

We define the **prf advantage** of an adversary A attacking F as

$$\mathbf{Adv}^{\text{prf}}_F(A) = 2 \cdot \Pr[\mathbf{Exp}^{\text{prf}}_F(A) \Rightarrow \mathsf{true}] - 1 \,. \tag{3.3}$$

Notice that, since some functions in F may not have inverses, chosen-ciphertext attacks are not applicable here.

3.5 Attacks against block ciphers and PRFs

Now we consider some attacks against some families of permutations and families of functions. These examples show how the definitions above can be used to make the degrees of (in)security of PRPs and PRFs under consideration concrete.

3.5.1 Example: An obviously insecure block cipher: the family of the identity function

Consider a family of permutations $F : \{0,1\}^{56} \times \{0,1\}^{64} \to \{0,1\}^{64}$ defined as follows: for any $K \in \{0,1\}^{56}$ and any $x \in \{0,1\}^{64}$,

$$F_K(x) = x \,.$$

Clearly, this family of permutations is completely insecure because, for any key K, the permutation F_K is in fact the identity function on $\{0,1\}^{64}$. For concreteness, we define an attack explicitly as follows.

Adversary A^{g}

$y \leftarrow \mathsf{g}(0^{64})$

If $y = 0^{64}$ then return 1 else return 0

In short, the adversary A submits one query and looks at the output. If it looks exactly the same as the input, then A bets that the oracle g is in fact chosen from the family F. Otherwise, A bets that g is a permutation chosen at random from the set of all possible permutations over $\{0,1\}^{64}$. We analyze the prp-cpa advantage of A by first

computing the quantity $\Pr[\mathbf{Exp}_F^{\text{prp-cpa}}(A) \Rightarrow \mathsf{true}]$. Let b be the random variable for the challenge bit b chosen in the experiment. We have that

$$\begin{aligned}
&\Pr[\mathbf{Exp}_F^{\text{prp-cpa}}(A) \Rightarrow \mathsf{true}] \\
&= \Pr\left[\mathbf{Exp}_F^{\text{prp-cpa}}(A) \Rightarrow \mathsf{true} \mid b = 1\right] \cdot \Pr[b = 1] \\
&\quad + \Pr\left[\mathbf{Exp}_F^{\text{prp-cpa}}(A) \Rightarrow \mathsf{true} \mid b = 0\right] \cdot \Pr[b = 0] \\
&= 1 \cdot \frac{1}{2} + \Pr\left[\mathbf{Exp}_F^{\text{prp-cpa}}(A) \Rightarrow \mathsf{true} \mid b = 0\right] \cdot \frac{1}{2} && (3.4) \\
&= \frac{1}{2} + \left(1 - \frac{1}{2^{64}}\right) \cdot \frac{1}{2} && (3.5) \\
&= 1 - \frac{1}{2^{65}} \,. && (3.6)
\end{aligned}$$

Equation (3.4) holds because g is simply an identity function when $b = 1$. Thus, if $b = 1$, then the output that A receives will be 0^{64} and A will always return 1. Equation (3.5) holds because, when $b = 0$, the oracle g is a permutation chosen at random from all possible permutations over $\{0,1\}^{64}$. Thus, the chance that g would map 0^{64} to 0^{64}, hence misleading A to return 1 rather than the correct answer, is 2^{-64}. Substituting the quantity in Equation (3.6) into Equation (3.1) in Definition 3.5, we obtain

$$\mathbf{Adv}_F^{\text{prp-cpa}}(A) = 2 \cdot \left[1 - \frac{1}{2^{65}}\right] - 1 = 1 - \frac{1}{2^{64}} \,,$$

which is very close to one. Thus, A is able to achieve a high probability of success with a small amount of resources. Specifically, A only submits one query of length 64 bits. Thus, as a pseudorandom permutation family, F is considered completely insecure.

3.5.2 Example: A block cipher that completely reveals the secret key

Consider a family of permutations $F : \{0,1\}^{64} \times \{0,1\}^{64} \to \{0,1\}^{64}$ defined as follows: for any $K \in \{0,1\}^{64}$ and any $x \in \{0,1\}^{64}$,

$$F_K(x) = K \oplus x \,.$$

This family of permutations is completely insecure because, for any key K, the permutation $F_K(0^{64}) = K$. For concreteness, we define an attack explicitly as follows.

Adversary A^g

$y \leftarrow g(0^{64})$; $y' \leftarrow g(1^{64})$

If $y' = F_y(1^{64})$ then return 1 else return 0

In short, the adversary A submits one query containing only zeros and verifies whether the output is in fact the secret key by checking whether g's output on any input different from 0^{64} equals the output of F with the suspected value as the key applied to the same input. If they are equal, then A bets that the oracle g is in fact chosen from the family F. Otherwise, A bets that g is a permutation chosen at random from the set of all possible permutations over $\{0,1\}^{64}$. We analyze the prp-cpa advantage of A by first computing the quantity $\Pr[\mathbf{Exp}_F^{\text{prp-cpa}}(A) \Rightarrow \mathsf{true}]$. Let b be the random variable for the challenge bit b chosen in the experiment. We have that

$$
\begin{aligned}
&\Pr[\mathbf{Exp}_F^{\text{prp-cpa}}(A) \Rightarrow \mathsf{true}] \\
&= \Pr\left[\mathbf{Exp}_F^{\text{prp-cpa}}(A) \Rightarrow \mathsf{true} \mid b = 1\right] \cdot \Pr[b = 1] \\
&\quad + \Pr\left[\mathbf{Exp}_F^{\text{prp-cpa}}(A) \Rightarrow \mathsf{true} \mid b = 0\right] \cdot \Pr[b = 0] \\
&= 1 \cdot \frac{1}{2} + \Pr\left[\mathbf{Exp}_F^{\text{prp-cpa}}(A) \Rightarrow \mathsf{true} \mid b = 0\right] \cdot \frac{1}{2} && (3.7) \\
&= \frac{1}{2} + \left(1 - \frac{1}{2^{64}}\right) \cdot \frac{1}{2} && (3.8) \\
&= 1 - \frac{1}{2^{65}} \,. && (3.9)
\end{aligned}
$$

To see that Equation (3.7) holds, first notice that, as discussed, we have that $F_K(0^{64}) = K$ for any K. Therefore, A obtains the secret key through y. If $b = 1$, then g is a permutation indexed by y from the family F. Thus $y' = F_y(1^{64})$, and A always returns 1. On the other hand, if $b = 0$, then the response y' from the second query to g is a random value in $\{0,1\}^{64}$ and thus would equal a particular value with probability 2^{-64}. Thus, A would return 0 in this case with probability $1 - 2^{-64}$ yielding Equation (3.8). Substituting the quantity in Equation (3.9) into Equation (3.1) in

Definition 3.5, we obtain

$$\mathbf{Adv}_F^{\text{prp-cpa}}(A) = 2 \cdot \left[1 - \frac{1}{2^{65}}\right] - 1 = 1 - \frac{1}{2^{64}} ,$$

which is very close to one. Thus, A is able to achieve a high probability of success with a small amount of resources. Specifically, A only submits two queries of length 64 bits each. Thus, as a pseudorandom permutation family, F is considered completely insecure.

3.6 Conclusion

Block ciphers are one of the most crucial ingredients in symmetric cryptography. Many constructs are designed using block ciphers as building blocks. The most popular block ciphers are DES and AES, both of which emerge through the standardization processes spearheaded by the National Institute of Standards and Technology (NIST). We present in this chapter the security definitions for block ciphers. These are the PRP-CPA, PRP-CCA, and PRF security notions. Keeping in mind the games defined in these definitions, cryptanalysts pick apart concrete block ciphers to evaluate the levels of security they provide. Cryptographers devise algorithms and protocols using block ciphers assumed to be secure under these definitions. Often, the black-box manner in which the design of higher-level primitives and protocols is done allows block ciphers that have become obsolete to be replaced by new ones whenever possible (when the number of bits of the input and the output are compatible or parameterized, for example). The modes of operation discussed in Chapter 5 exemplify how this paradigm is practiced.

3.7 Credits

The provable security movement started with Silvio Micali and Shafi Goldwasser's paper entitled *Probabilistic Encryption* [27]. The trend of defining the syntax and the correctness condition for cryptographic primitives and protocols was made popular

by Mihir Bellare and Phillip Rogaway. So is the idea of writing out an experiment to measure an adversary's success probability as done here to quantify the level of security of a cryptographic scheme. The use of subroutines in the experiment is also due to their work on code-based game playing proofs [13]. The difference is that we explicit spell out the experiment here while they explain it in the prose.

The use of the term "encipher" to emphasize the difference between applying a block cipher in the forward direction and encrypting a message is from the Asiacrypt paper [12] by Mihir Bellare and Phillip Rogaway.

3.8 Exercises

Exercise 3.1. Let $E\colon \{0,1\}^2 \times \{0,1\}^2 \to \{0,1\}^2$ be the following maps.

$$E_{00} = (00,10,01,01) \qquad E_{01} = (00,01,10,11)$$
$$E_{10} = (10,11,01,00) \qquad E_{11} = (01,10,11,00)$$

Is E a block cipher? Explain in detail.

Exercise 3.2. Let $E\colon \{0,1\}^2 \times \{0,1\}^2 \to \{0,1\}^2$ be the following maps.

$$E_{00} = (01,10,11,00) \qquad E_{01} = (11,01,10,00)$$
$$E_{10} = (10,00,11,01) \qquad E_{11} = (00,10,11,01)$$

1. Suppose the secret key $K = 10$ and the input $M = 10$. Then, what is $E_K(M)$?
2. Specify the set of keys consistent with the input-output pair $(00,10)$.
3. Specify the set of keys consistent with the input-output pair $(10,11)$.
4. Specify the set of keys consistent with the input-output pairs $(10,11),(01,10)$.

Exercise 3.3. Consider $E\colon \{0,1\}^{56} \times \{0,1\}^{64} \to \{0,1\}^{64}$ defined as follows. For any $K \in \{0,1\}^{56}$ and any $M \in \{0,1\}^{64}$,

$$E_K(M) = \mathsf{DES}(K,0^{64}) \oplus M\ .$$

1. Define E^{-1}.
2. Prove that E is insecure under PRP-CPA. You need to write the pseudocode for an adversary, analyze its advantage, and specify its resource usage. The less the resource usage and the higher the advantage, the better.

Exercise 3.4. Recall that 8 S-boxes are used as part of the DES block cipher. The description of an S-box as studied in class is spelled out here for your convenience.

> Each S-box is described by a table as partially shown shown below. Read these tables as follows. $\mathbf{S}_i$ takes a 6-bit input. Write it as $b_1b_2b_3b_4b_5b_6$. Read $b_3b_4b_5b_6$ as an integer in the range 0,..., 15, naming a column in the table describing $\mathbf{S}_i$. Let b_1b_2 name a row in the table describing $\mathbf{S}_i$. Take the row b_1b_2, column $b_3b_4b_5b_6$ entry of the table of $\mathbf{S}_i$ to get an integer in the range 0,..., 15. The output of $\mathbf{S}_i$ on input $b_1b_2b_3b_4b_5b_6$ is the 4-bit string corresponding to this table entry.

The S-box $\mathbf{S}_1$ from a figure in the lecture notes slides (page 16) is shown below.

		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
0	1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
1	0	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
1	1	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

1. Is $\mathbf{S}_1$ a function? Explain your answer.
2. Is $\mathbf{S}_1$ a permutation? Explain your answer.

Exercise 3.5. Write one family of functions with the signature $\{0,1\}^2 \times \{0,1\}^2 \to \{0,1\}^3$. Then, answer the following questions.

1. For your family of functions above, what is the output when the key is 01 and the input is 11?
2. For your family of functions above, what is the output when the key is 11 and the input is 11?

Exercise 3.6. Write one family of permutations with the signature $\{0,1\}^2 \times \{0,1\}^3 \to \{0,1\}^3$. Then, answer the following questions.

1. For your family of functions above, what is the output when the key is 01 and the input is 011?
2. For your family of functions above, what is the output when the key is 11 and the input is 011?

Exercise 3.7. Answer the following questions.

1. Can there be a family of functions with the signature $\{0,1\}^5 \times \{0,1\}^4 \to \{0,1\}^2$? Explain.
2. Can there be a family of functions with the signature $\{0,1\}^5 \times \{0,1\}^2 \to \{0,1\}^4$? Explain.
3. Can there be a family of permutations with the signature $\{0,1\}^5 \times \{0,1\}^4 \to \{0,1\}^2$? Explain.
4. Can there be a family of permutations with the signature $\{0,1\}^5 \times \{0,1\}^2 \to \{0,1\}^4$? Explain.

Exercise 3.8. Let k, l be non-negative integers. Let $E\colon\ \{0,1\}^k \times \{0,1\}^l \to \{0,1\}^l$ be a family of permutations. Let $F\colon\ \{0,1\}^k \times \{0,1\}^{2l} \to \{0,1\}^{2l}$ be defined as follows: for any $K \in \{0,1\}^k$ and any $M \in \{0,1\}^{2l}$,

$$F_K(M) = E_K(M[1]) \| E_K(M[2])$$

where $M = M[1]M[2]$ and $|M[1]| = |M[2]| = l$.

1. Is F a secure pseudorandom permutation family against chosen-ciphertext attacks (PRP-CCA) as per Definition 3.6?
2. Prove your answer. If your answer is yes, you need to show a reduction and derive the relationship between the insecurity of E and that of F obtained from the reduction. If your answer is no, you need to write the pseudocode for an adversary, analyze its advantage, and specify its resource usage. The less the resource usage and the higher the advantage, the higher the grades.

4

Building Block II: Pseudorandom Number Generators

4.1 Motivation

Randomness is one of the most crucial ingredients in many cryptosystems. Many cryptographic algorithms are randomized. Secret keys in symmetric cryptography are almost always chosen uniformly at random from the set of all possible strings of a certain length. Consequently, generating random strings for various uses as required by the algorithms is a task that security-minded programmers must be able to perform. Yet, there are not many sources of randomness in nature. Furthermore, as we know, computers are very much a deterministic machine. (A computer that behaves nondeterministically during the boot sequence, while could be a source of excitement, is not too desirable for someone who wants to get work done with it.) Thus, a common approach to solve this problem is to start with a (possibly short) random bitstring then "stretch" it to a desired length, all the while limiting the degradation in the degree of randomness obtained. An algorithm for this task is called a Pseudorandom Number Generator (PRG or PRNG). Note that the view we take in this book is that a PRNG takes a bitstring as an input and outputs another bitstring. Specifically, it does not generate any number. However, there are PRNGs that do generate sequences of numbers, for example, that designed by Shamir [44].

4.2 What it is

As discussed above, a PRG is quite different from PRPs and PRFs. A PRG G has the following signature:

$$G : \{0,1\}^m \to \{0,1\}^n$$

where n and m are integers and $n > m$. In other words, given a bitstring, a PRG outputs another bitstring that is longer than the input. Notice that, unlike PRPs and PRFs, a PRG does not take a key as an input. Thus, if one knows the input, the output can be easily computed. Typically, the input is a truly random seed, and the PRG is considered to "stretch" it to obtain a longer bitstring that is still "reasonably random." Below, we define what this means more precisely.

Definition 4.9: Security notion for PseudoRandom number Generators

Let m, n be positive integers, and let $G : \{0,1\}\{0,1\}^m \to \{0,1\}^n$ be a pseudorandom number generator. Let A be an adversary. We define the following subroutines, experiment and advantage function.

Subroutines

Subroutine `Initialize`

$b \xleftarrow{\$} \{0,1\}$

If $b = 1$ then $x \xleftarrow{\$} \{0,1\}^m$; $y \leftarrow G(x)$ else $y \xleftarrow{\$} \{0,1\}^n$

Return y

Subroutine `Finalize`(d)

Return (d = b)

Experiment

Experiment $\mathbf{Exp}^{\text{prg}}_{G}(A)$

$y \xleftarrow{\$}$ `Initialize`

$d \xleftarrow{\$} A(y)$

Return `Finalize`

We define the **prg advantage** of an adversary A attacking G as

$$\mathbf{Adv}^{\text{prg}}_{G}(A) = 2 \cdot \Pr[\mathbf{Exp}^{\text{prg}}_{G}(A) \Rightarrow \mathsf{true}] - 1 \ . \tag{4.1}$$

Conceptually, the experiment works as follows. An adversary A attacking a PRG G mapping m bits to n bits is presented with a string y of length n bits. The adversary A has to decide whether y is

- a string obtained by applying G to a random seed or
- a truly random string.

If G is a good PRG, it should be difficult for any A with reasonable amount of resources to answer correctly with a high probability.

4.3 Gaps between the model and practical constructions

As specified in the syntax, a PRG takes in a fixed length seed and produces a string of longer pseudorandom bits. This view is reasonable except that many PRGs used in practical applications do not always fit into this mold. We discuss some examples here.

1. RC4, the most widely used stream cipher, which should be possible to regard as a pseudorandom number generator as it is often used to generate keystreams (for instance, in the Wired Equivalent Protocol (WEP)), takes in a variable-length input as a seed. Furthermore, in WEP, RC4 is given an *initial vector* (IV)

as an auxiliary input. Predictably enough, the way the IV is handled, together with other issues with WEP's use of RC4, lead to a widely known, practical attack against WEP. [25]. We emphasize, however, that the weaknesses of WEP stem from the way it uses RC4 and not from any flaw in RC4 itself.

2. The PRG specified in the ANSI 9.17 standard [4] actually takes 3 inputs: a secret key, the current state, and an auxiliary piece of information (specifically, a timestamp). It is not immediately clear which input should be considered the random seed (the variable x in Definition 4.9) and how to treat the other two inputs. As explored precisely and concretely by Desai, Hevia, and Yin [22], the choice one makes in this regard has profound implications for the security of the construction. Their Eurocrypt 2002 paper addresses this issue by proposing a different security definition that they believe addresses this gap better.
3. The PRG specified in the FIPS 186 standard [23] is similar to that specified in the ANSI 9.17 standard discussed above in that the PRG also takes 3 inputs. One crucial difference between the two PRGs is that the ANSI 9.17 PRG uses a block cipher while the FIPS 186 PRG uses a hash function as the main component in the construction. The implications in terms of the gaps between the simple model for PRG and the FIPS 186 construction are similar to those mentioned above for the ANSI 9.17 PRG. Desai et al. also address the FIPS 186 PRG in their work [22].

4.4 Attacks against PRNGs

4.4.1 Example: Completely insecure PRG

Consider a PRG $G : \{0,1\}^{32} \to \{0,1\}^{64}$ defined as follows: for any $x \in \{0,1\}^{32}$,

$$G(x) = x||x\ .$$

Intuitively, we can easily see that G is a bad PRG because, to stretch the random seed it has received as an input into an output that is twice as long, G simply repeats the input.

Consequently, this makes the output far from random-looking. For concreteness, we define an attack explicitly as follows.

Adversary $A(y)$

Parse y as 32-bit blocks $y[1]y[2]$

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

In short, if the two halves of the challenge string y are the same, A bets that y is an output of G. Otherwise, it bets that y is a string chosen at random from $\{0,1\}^{64}$. We now compute the advantage of A. Let b be the random variable for the challenge bit b chosen in the experiment. We have that

$$\begin{aligned}
&\Pr[\mathbf{Exp}^{\text{prg}}_G(A) \Rightarrow \mathsf{true}] \\
&= \Pr\left[\mathbf{Exp}^{\text{prg}}_G(A) \Rightarrow \mathsf{true} \mid b = 1\right] \cdot \Pr[b = 1] \\
&\quad + \Pr\left[\mathbf{Exp}^{\text{prg}}_G(A) \Rightarrow \mathsf{true} \mid b = 0\right] \cdot \Pr[b = 0] \\
&= 1 \cdot \frac{1}{2} + \Pr\left[\mathbf{Exp}^{\text{prg}}_G(A) \Rightarrow \mathsf{true} \mid b = 0\right] \cdot \frac{1}{2} && (4.2) \\
&= \frac{1}{2} + \left(1 - \frac{1}{2^{32}}\right) \cdot \frac{1}{2} && (4.3) \\
&= 1 - \frac{1}{2^{33}} \,. && (4.4)
\end{aligned}$$

To see that Equation (4.2) holds, first notice that, as discussed, for any x, we have that $G(x) = y[1]y[2]$ where, if $|y[1]| = |y[2]| = 32$, then $y[1] = y[2]$. Thus, if $b = 1$, then $y = G(x)$ will have this property, and thus A will always answer correctly. Otherwise, y is a 64-bit string chosen at random from $\{0,1\}^{64}$ and thus would have a probability of only 2^{-32} to have this property. Thus, A will answer incorrectly with probability 2^{-32} yielding Equation (4.3). Substituting the quantity in Equation (4.4) into the equation in Definition 4.9, we obtain

$$\mathbf{Adv}^{\text{prg}}_G(A) = 2 \cdot \left[1 - \frac{1}{2^{33}}\right] - 1 = 1 - \frac{1}{2^{32}} \,,$$

which is very close to one. Thus, A is able to achieve a high probability of success with a small amount of resources. Thus, as a PRG, G is considered completely insecure.

4.5 Conclusion

Most cryptosystems require the use of random bit strings in one form or another. Since sources of true randomness that can be conveniently used in computer systems are few and far between, pseudorandom bit sequence generators are often needed to stretch the few bits that can be squeezed out of the sources of truly random bits into bit strings that are long enough as required by the specifications of cryptographic algorithms and protocols. The way we model a PRNG in this chapter and the corresponding security definition is a simple one and is by no means the only way. See [22] and [19], for example, for alternatives.

4.6 Credits

The study of provably secure PRG began with Shamir's proposal in 1981 of a pseudorandom sequence generator which generates a sequence of numbers rather than sequence of bits [44]. Blum and Micali [18] proposed pseudorandom bit sequence generator in 1984. Many other PRG soon followed [18, 17, 2, 30, 48].

4.7 Exercises

Exercise 4.1. Why is it insufficient for a PRG to simply be one-way? Can you think of a one-way function (a function that is hard to invert) but is clearly insecure as a PRG?

Exercise 4.2. What would happen if the adversary knows all of the bits of the seed input to the PRG? What if it knows half of the bits?

Exercise 4.3. Consider a PRG $G : \{0,1\}^{32} \to \{0,1\}^{64}$ defined as follows: for any $x \in \{0,1\}^{32}$,

$$G(x) = x \oplus x \| x \, .$$

Is G a secure PRG? Prove your answer.

Exercise 4.4. Let k and l be non-negative integers. Let $E\colon \{0,1\}^k \times \{0,1\}^l \to \{0,1\}^l$ be a secure block cipher. Consider a PRG $G : \{0,1\}^l \to \{0,1\}^{2l}$ defined as follows: for any $x \in \{0,1\}^l$,

$$G(x) = E_{0^k}(x) \| E_{0^k}(x) \ .$$

Is G a secure PRG? Prove your answer.

Exercise 4.5. Let k and l be non-negative integers. Let $E\colon \{0,1\}^k \times \{0,1\}^l \to \{0,1\}^l$ be a secure block cipher. Consider a PRG $G : \{0,1\}^l \to \{0,1\}^{2l}$ defined as follows: for any $x \in \{0,1\}^l$,

$$G(x) = E_{0^k}(x) \| E_{0^k}(\overline{x}) \ .$$

Is G a secure PRG? Prove your answer.

Exercise 4.6. Let l be a non-negative integer. Let $E\colon \{0,1\}^l \times \{0,1\}^l \to \{0,1\}^l$ be a secure block cipher. Consider a PRG $G : \{0,1\}^l \to \{0,1\}^{2l}$ defined as follows: for any $x \in \{0,1\}^l$,

$$G(x) = E_x(x) \| E_x(\overline{x}) \ .$$

Is G a secure PRG? Prove your answer.

5

Block-Cipher-Based Symmetric Encryption

5.1 Motivation

Block ciphers would be adequate for encryption if all messages were of a fixed size equal to the block size. However, in real applications, messages are often of different sizes. Furthermore, one message may be shorter than the block size of the block cipher we decide to use while another message may be longer. For applications in which these scenarios may occur, one need to use an appropriate **mode of operation**. Specifically, if the messages to be encrypted may be shorter than the block size, we need to use an approprite mode that specifies how to pad short messages properly. Similarly, if the messages may be longer than the block size, the mode we choose must specify how to divide up long messages into blocks and how to process them properly. The end goal is to preserve the security offered by the block cipher as much as possible while taking other application-specific needs into consideration.

In this chapter, we first describe how encryption schemes are modelled and the security notions that is widely accepted as appropriate for encryption in general purpose applications. Then, we describe the modes of operation defined in FIPS PUB 81 and one other related mode and analyze the security of some of the modes.

5.2 What it is: syntax and correctness condition

We have now discussed two different ways of achieving private communication through the use of symmetric encryption schemes. The methods are different, but as one can see, there is also a pattern emerging. We specify here what they have in common. In fact, we go further to specify the features any symmetric encryption scheme must have. These characteristics are specified through mathematical statements and are collectively referred to as the **syntax** of symmetric encryption schemes.

To determine whether a *thing*[1] under the consideration is a symmetric encryption scheme or not, one looks at the syntax and the accompanying correctness condition and figures out whether the thing in question follows them.

Some background and notation.

Before proceeding further, we need to discuss some basic terminology related to algorithms and to specify some notations to help us stay precise while readable at the same time.

An algorithm specifies a sequence of steps that get executed in order when the algorithm is run. It takes zero or more inputs and returns zero or more outputs. A **deterministic algorithm** A is one that behaves deterministically: given particular inputs, A always returns the same outputs no matter how many times it is run on those same inputs. An algorithm that internally flips coins may not behave in this way and thus is called a **non-deterministic** or **randomized algorithm**. (The internal coin flips may alternatively be modelled as providing the algorithm with random bits as an additional input.) Orthogonal to whether an algorithm is deterministic is whether it is stateful or stateless. A **stateful algorithm** is one that maintains internal states across multiple invocations. (Think a static variable in the C programming language.) A **stateless algorithm** is one that does not.

[1] If one has contempt toward such an informal word, one can replace it with the phrase *mathematical object* here.

Now we introduce some notations. If x is a string, then $|x|$ denotes its length (i.e. the number of symbols that appear in x). Suppose A is a deterministic algorithm, we write $x \leftarrow A(y)$ to denote that A is run with y as the input and x as the output. If A is a randomized algorithm, then we write $x \xleftarrow{\$} A(y)$ to denote the same thing except that the added dollar sign is there to emphasize that A is randomized, and that, as a result, its output may change depending on the values of the coin flips of A in addition to its input y. We denote the set of all possible outputs that may result from the execution of A with y as the input as $[A(y)]$. Here and throughout the rest of the book, unless otherwise specified, the plaintext and ciphertext spaces are assumed to be $\{0,1\}^*$ for simplicity.

The following specify the syntax and the correctness condition for symmetric encryption schemes. All data being manipulated as inputs and outputs of algorithms are bitstrings.

Definition 5.10: Syntax of symmetric encryption schemes.

A **symmetric encryption scheme** SE is a triple of algorithms $(\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec})$.

- The randomized **key generation algorithm** KG takes no input and returns the secret key K as the output.
- The possibly randomized and/or stateful **encryption algorithm** Enc takes a secret key K and a plaintext M as inputs and returns either the ciphertext C or the special symbol $\perp$ indicating a failure as the output.
- The deterministic and stateless **decryption algorithm** Dec takes a secret key K and a ciphertext C as inputs and returns either the plaintext M or the special symbol $\perp$ indicating a failure as the output.

The correctness condition requires that, for any key $K \in [\mathsf{KG}]$, any plaintext M, it must be the case that

$$\Pr\left[C = \perp \vee \mathsf{Dec}(K,C) = M \ : \ C \xleftarrow{\$} \mathsf{Enc}(K,M)\right] = 1\ .$$

We make the following observations.[2]

First, to state the obvious, an encryption scheme is not one but three algorithms. There are more to an encryption scheme than the encryption algorithm. To define an encryption scheme, one must specify how the secret keys are to be chosen and how a ciphertext is to be decrypted.

Second, as a general rule of thumb, the key generation algorithm of a good encryption scheme should not be deterministic. Technically, Caesar cipher violates this condition as the secret key is always three. Alternatively, one can view Caesar cipher as a shift cipher whose key generation algorithm outputs an integer chosen at random from integers between 1 and 26.

Third, the encryption and the decryption algorithms are allowed to return a special symbol $\perp$ indicating a failure. An example of an encryption failure is a scenario where the encryption algorithm that uses a counter that gets incremented for each plaintext finds that the counter has exceeded the capacity (e.g. a 32-bit counter can only count up to at most 2^{32} depending on the number representation used). In this case, the algorithm will return $\perp$. For decryption, $\perp$ may be an output indicating that the ciphertext decrypts to a plaintext that is of a wrong format.

Fourth, we require above that the decryption algorithm of an encryption scheme be deterministic. This means that, given a secret key and a ciphertext, it always outputs a particular value as the message corresponding to the ciphertext. It is conceivable that the decryption algorithm of an encryption scheme would do this with a probability that, although is high, is not one. All practical encryption schemes are, however, deterministic.

Fifth, the correctness condition can be read as follows. No matter which secret key K ends up being used and what plaintext M gets encrypted, either the encryption of M results in a failure or it does not. In the latter case, decrypting the resulting ciphertext with the same secret key K ought to yield the original plaintext M.

[2] Since these observations are relevant to both symmetric and asymmetric encryption schemes, we simply use the term “encryption schemes” rather than symmetric encryption schemes even though we have not specified the syntax of asymmetric encryption schemes at this point.

Sixth, the correctness condition says absolutely nothing about the security of the scheme. A scheme can satisfy the correctness requirement while being completely insecure. Consider as an example a scheme $(\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec})$ for which KG simply outputs a bitstring of length 128 chosen at random from all possible strings of length 128 while Enc and Dec simply return their second input (namely, the plaintext in the case of Enc and the ciphertext in the case of Dec) as the output. Clearly, this scheme is completely insecure, yet it is defined according to the syntax and meets the correctness requirement.

Seventh, we emphasize that all three algorithms that make up the scheme are public. Specifically, everyone knows how they work and can write programs to execute them if they wish. The only secret component that provides the information advantage to the legitimate parties over the adversary is the shared secret key.

It is a simple exercise to write the shift cipher as a symmetric encryption scheme following the syntax specified above and to verify that the correctness condition is met. As a different example, we do so for one-time pad.

5.2.1 Example: The one-time pad symmetric encryption scheme

Let n be an integer specifying the size of the plaintexts to be sent. The one-time pad OTP is a triple of algorithms $(\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec})$ defined as follows:

Algorithm KG

$K \xleftarrow{\$} \{0,1\}^n$; Return K

Algorithm $\mathsf{Enc}(K, M)$

If $|M| \neq n$ then return $\perp$

$C \leftarrow K \oplus M$; Return C

Algorithm $\mathsf{Dec}(K, C)$

If $|C| \neq n$ then return $\perp$

$M \leftarrow K \oplus C$; Return M

The correctness condition is met because, for any $K, M \in \{0,1\}^n$, we have that $K \oplus (K \oplus M) = M$.

The following example shows something that looks very much like a symmetric encryption scheme but is not.

5.2.2 Example: Not a symmetric encryption scheme

Let n be an integer specifying the size of the plaintexts to be sent. The one-time pad OTP is a triple of algorithms $(\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec})$ defined as follows:

Algorithm KG

$K \xleftarrow{\$} \{0,1\}^n$; Return K

Algorithm $\mathsf{Enc}(K, M)$

If $M = 0^n$ then Return 1^n

If $|M| \neq n$ then return $\perp$

$C \leftarrow K \oplus M$; Return C

Algorithm $\mathsf{Dec}(K, C)$

If $C = 1^n$ then Return 0^n

If $|C| \neq n$ then return $\perp$

$M \leftarrow K \oplus C$; Return M

At first glance, the differences between this scheme and OTP seem to be rather reasonable. However, notice that whenever the recipient receives the ciphertext 1^n, there is no way for him to tell whether the plaintext the sender meant to send is 0^n or $K \oplus 1^n$. If K happens to be 1^n, then this would not be a problem. But for any other value of K, the ciphertext cannot be decrypted unambigously.

In other words, the correctness condition is not met. For example, suppose that $K = 0^n$ and that $M = 1^n$. We would then have that $\Pr\left[C = \perp \vee \mathsf{Dec}(K, C) = 1^n \ : \ C \xleftarrow{\$} \mathsf{Enc}(K, 1^n)\right] = 0 \neq 1$. This is true be-

cause the ciphertext sent is $C = K \oplus M = 0^n \oplus 1^n = 1^n$ but $\mathsf{Dec}(K, 1^n) = 0^n \neq 1^n$. In fact, other values of K and M would also work as long as $K \oplus M = 1^n$ but $M \neq 0^n$.

5.3 What security means: privacy

In this section, we focus on how to determine the level of security offered by a symmetric encryption scheme. To do so, we need to first focus on what it means for a symmetric encryption scheme to provide privacy. Recall that the parties involved in the problem of how to communicate privately are the sender, the receiver, and the adversary. The sender and the receiver who are using a particular encryption scheme to communicate are supposed to behave as specified by the scheme. The crux of the problem then lies in the adversary. In particular, two questions need be answered.

- What all is the adverary allowed to do? This defines the limit of its power.
- What is the adversary trying to accomplish in the end? This defines its goal.

The approach taken in modern cryptography is to first precisely define an experiment in which the answers to these questions are explicitly specify then measure the probability of success of the adversary operating in this experiment in achieving its goal.

The following definitions are commonly accepted as appropriately capturing the concept of privacy that a symmetric encryption scheme should provide if it were to be used within the context of other practical applications. We note however that, even though these definitions are widely accepted, they are not the only ones. Further refinements are still being devised and debated.

Definition 5.11: Privacy notion for symmetric encryption schemes: Indistinguishability against Chosen-Plaintext Attacks

Let $\mathsf{SE} = (\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec})$ be a symmetric encryption scheme, and let A be an adversary with access to an oracle. We define the following subroutines, experi-

ment, and advantage function.

Subroutines

Subroutine `Initialize`

$b \xleftarrow{\$} \{0,1\}$; $K \xleftarrow{\$} \mathsf{KG}$

Subroutine $\texttt{Enc}(M_0, M_1)$

If $|M_0| \neq |M_1|$ then return $\perp$

Return $\mathsf{Enc}_K(M_b)$

Subroutine $\texttt{Finalize}(d)$

Return $(d = b)$

Experiment

Experiment $\mathbf{Exp}^{\text{ind-cpa}}_{\mathsf{SE}}(A_1)$

`Initialize`

$d \xleftarrow{\$} A^{\texttt{Enc}}$

Return $\texttt{Finalize}(d)$

We define the **ind-cpa advantage** of an adversary A mounting a chosen-ciphertext attack against SE as

$$\mathbf{Adv}^{\text{ind-cpa}}_{\mathsf{SE}}(A) = 2 \cdot \Pr[\mathbf{Exp}^{\text{ind-cpa}}_{\mathsf{SE}}(A) \Rightarrow \mathsf{true}] - 1 \ . \tag{5.1}$$

Definition 5.12: Privacy notion for symmetric encryption schemes: Indistinguishability against Chosen-Ciphertext Attacks

Let $\mathsf{SE} = (\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec})$ be a symmetric encryption scheme, and let A be an adversary with access to an oracle. We define the following subroutines, experiment, and advantage function.

Subroutines

Subroutine `Initialize`

$b \xleftarrow{\$} \{0,1\}$; $K \xleftarrow{\$} \mathsf{KG}$; $S \leftarrow \emptyset$

Subroutine `Enc`(M_0, M_1)

If $|M_0| \neq |M_1|$ then return $\perp$

$C \xleftarrow{\$} \mathsf{Enc}(K, M_b)$; $S \leftarrow S \cup \{C\}$

Return C

Subroutine `Dec`(C)

If $C \in S$ then return $\perp$

Return $\mathsf{Dec}(K, C)$

Subroutine `Finalize`(d)

Return $(d = b)$

Experiment

Experiment $\mathbf{Exp}^{\text{ind-cca}}_{\mathsf{SE}}(A_2)$

`Initialize`

$d \xleftarrow{\$} A^{\texttt{Enc},\texttt{Dec}}$

Return `Finalize`(d)

We define the **ind-cca advantage** of an adversary A mounting a chosen-

ciphertext attack against SE as

$$\mathbf{Adv}^{\text{ind-cca}}_{\mathsf{SE}}(A) = 2 \cdot \Pr[\mathbf{Exp}^{\text{ind-cca}}_{\mathsf{SE}}(A) \Rightarrow \mathsf{true}] - 1 \ . \tag{5.2}$$

The general idea for the game against chosen-plaintext attacks as depicted in Figure 5.1 is to let the adversary submit a number of message pairs, see the encryption of only one of the messages in each pair, then output its guess as to whether the messages that were encrypted were always the left or the right ones. For chosen-ciphertext attacks, depicted in Figure 5.2, the adversary is additionally given the ability to ask for decryption of ciphertext of its choice. To prevent a trivial attack, however, if it ever asks to see the decryption of any ciphertext that it has obtained as a result of previous queries to the encryption oracle, then it gets no additional information (as represented by the special symbol $\perp$) in return. The check for this condition is in the first line of the subroutine Dec in Definition 5.12. There, we check to see whether the set S which contains all the ciphertexts returned so far by the encryption oracle contains the ciphertext of which the adversary asks to see the decryption and return $\perp$ if so.

Another check worth explaining is the length-checking step in the subroutines Enc in both experiments. The code there ensures that the messages submitted to the encryption oracle are of equal length. The reason that we refuse to return the ciphertext if the messages in each pair submitted by the adversary are of different lengths is that most encryption schemes do not attempt to hide the length of the plaintexts. Attacking these schemes by submitting messages of different lengths to the encryption oracle would have easily broken these schemes if this check was not included.

A word on terminology. For the IND-CPA security notion, we use the term *chosen-plaintext* attacks because the adversary can choose message pairs to submit to the encryption oracle. For the IND-CCA security notion, we use the term *chosen-ciphertext* attacks because the adversary additionally has access to the decryption oracle and can choose what ciphertexts to submit to it. The abbreviation IND stands for *indistinguishability* and refers to the fact that the adversary's goal is to distinguish

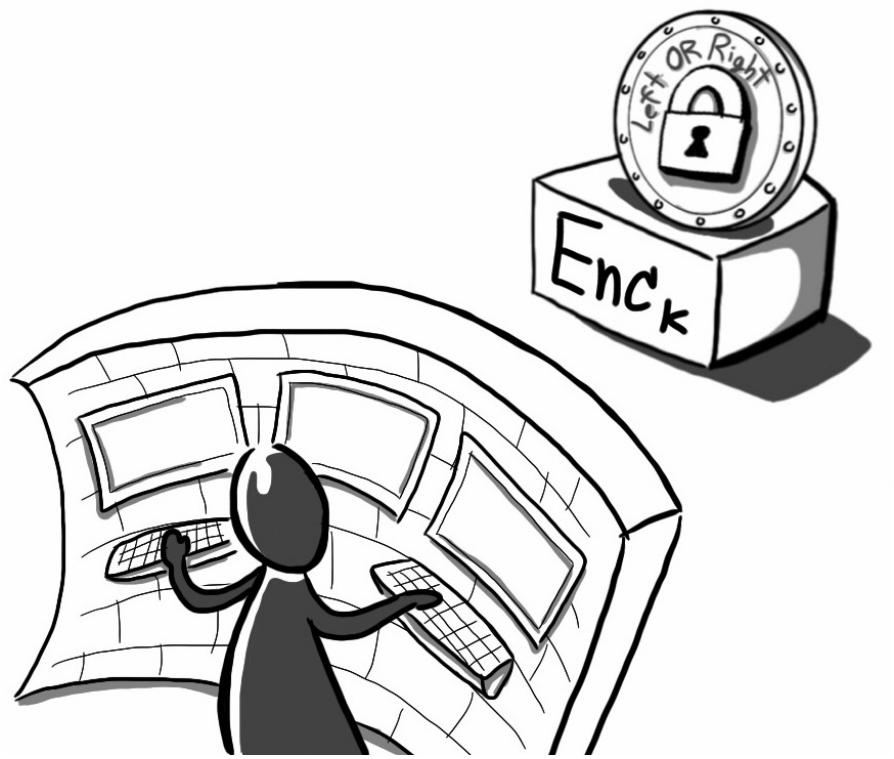


Figure 5.1: An experiment for the IND-CPA security definition for symmetric encryption.

whether the ciphertexts that it receives from the encryption oracle are the results of always encrypting the left or the right messages. Table 5.1 summarizes the power and the goal of adversaries mounting chosen-plaintext and chosen-ciphertext attacks against the privacy of any symmetric encryption scheme.

5.4 Constructions: modes of operation

FIPS PUB 81 defines four well-known modes. We describe each of them in turn. We also describe one popular mode called the counter mode. Throughout the rest of this chapter, if M is a bitstring, then we write $M[i]$ to denote the i-th block of M where the block size is dictated by the block size of the block cipher being employed. If the length of M is not a multiple of the block size, then all of the blocks except for the last one will have the length equal to the block size while the last block will contain

Figure 5.2: An experiment for the IND-CCA security definition for symmetric encryption.

the remaining bits and thus be shorter. FIPS PUB 81 leaves it up to the application developer to decide how to pad the last block if necessary. (The document suggests as an example that one pads the last block with random bits.) Below, we assume for simplicity that every plaintext is of length a multiple of the block size.

In all of the modes discussed here, the shared secret key chosen a priori is used as the key to the block cipher throughout the exchanges between a pair of sender and receiver. Many of the following modes make use of a bitstring called an initialization vector (IV). For these modes, to encrypt a plaintext, one needs to first set the value of the IV. FIPS PUB 81 does not explicitly specify to what value one should set the IV for each mode. We will discuss the various ways to set the IV in Section 5.4.6.

Notion	The adversary's power	The adversary's goal
IND-CPA	Submit message pairs to the encryption oracle and see the encryption of only one message in each pair. The oracle either always encrypts the left message or always encrypts the right message.	Guess whether the left or the right messages are encrypted
IND-CCA	Same as above but additionally can submit ciphertext to the decryption oracle to be encrypted. The oracle will return $\perp$ as the answer, however, if the ciphertext has previously been returned by the encryption oracle.	Guess whether the left or the right messages are encrypted

Table 5.1: A summary of the power and goal of any adversary mounting chosen-plaintext and chosen-ciphertext attacks against a symmetric encryption scheme.

5.4.1 Electronic Codebook Mode

The simplest mode is the Electronic Codebook (ECB) mode. To encrypt a long message, one simply divides up the message into blocks, enciphers each block independently using a single secret key, then concatenates the results to obtain the ciphertext. Figure 5.3 depicts the encryption operation for ECB. The diagram for the decryption operation is left as an exercise for the reader. We explicitly specify the encryption and decryption algorithms of ECB in Construction 5.13.

Construction 5.13: ECB

Let k and m be positive integers, and let $E : \{0,1\}^k \times \{0,1\}^m \to \{0,1\}^m$ be the underlying block cipher. The encryption scheme $\mathsf{ECB} = (\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec})$ consists of the following algorithms. The key generation algorithm KG simply returns a bitstring chosen at random from $\{0,1\}^k$. The encryption and decryption

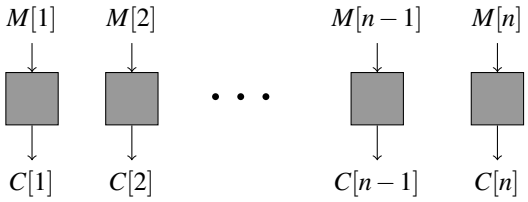


Figure 5.3: The encryption algorithm for the electronic codebook mode (ECB). The gray boxes denote the permutation E_K where E is the underlying block cipher and K is the shared secret key. It is assumed here that the message length is a multiple of the block size.

algorithms are as follows:

Algorithm $\mathsf{Enc}(K, M)$

If $(|M| = 0)$ or $(|M| \bmod m \neq 0)$ then return $\perp$

Parse M into m-bit blocks $M[1], \ldots, M[n]$

For $i = 1$ to n do

$C[i] \leftarrow E(K, M[i])$

Return $C[1] \| \ldots \| C[n]$

Algorithm $\mathsf{Dec}(K, C)$

If $(|C| = 0)$ or $(|C| \bmod m \neq 0)$ then return $\perp$

Parse C into m-bit blocks $C[1], \ldots, C[n]$

For $i = 1$ to n do

$M[i] \leftarrow E^{-1}(K, C[i])$

Return $M[1] \| \ldots \| M[n]$

ECB has long fallen out of favor mainly because it preserves too much of the structure of the plaintext. For example, a message containing two repeated blocks will encrypt to a ciphertext containing two repeated blocks. Encrypting a single message twice yields a single ciphertext both times. Moving ciphertext blocks around results in the plaintext blocks being moved around in the same way after decryption. These characteristics are undesirable when privacy is the goal.

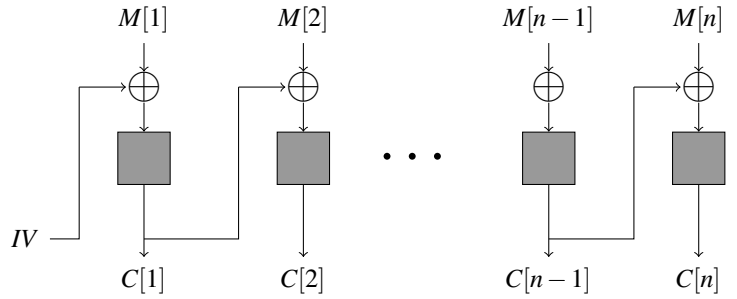


Figure 5.4: The encryption algorithm for the cipher block chaining mode (CBC). The gray boxes denote the permutation E_K where E is the underlying block cipher and K is the shared secret key. IV denotes the initial vector. It is assumed here that the message length is a multiple of the block size.

5.4.2 Cipher Block Chaining Mode

The most popular mode is the Cipher Block Chaining (CBC) mode. To encrypt the first block of the plaintext, one exclusive-ors it with the IV, which FIPS PUB 81 refers to as the randomizing block implying that its value is to be chosen (pseudo)randomly, then passes the result through as an input to the block cipher indexed by the shared secret key. To encrypt the rest of the plaintext, one proceeds similarly except that the IV is replaced by the previous ciphertext block. The ciphertext transmitted to the receiver is the concatenation of the IV and all of the ciphertext blocks.

We specify the encryption and decryption algorithms of CBC explicitly in Construction 5.14. Note that in this construction, the initial vector is chosen at random from $\{0,1\}^m$ where m is the block size.

Construction 5.14: CBC with a random IV

Let k and m be positive integers, and let $E : \{0,1\}^k \times \{0,1\}^m \to \{0,1\}^m$ be the underlying block cipher. The encryption scheme $\mathsf{CBCr} = (\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec})$ consists of the following algorithms. The key generation algorithm KG simply returns a bitstring chosen at random from $\{0,1\}^k$. The encryption and decryption algorithms are as follows:

Algorithm $\mathsf{Enc}(K, M)$

If $(|M| = 0)$ or $(|M| \bmod m \neq 0)$ then return $\perp$

Parse M into m-bit blocks $M[1], \ldots, M[n]$

$C[0] \leftarrow IV \xleftarrow{\$} \{0,1\}^m$

For $i = 1$ to n do

$C[i] \leftarrow E(K, C[i-1] \oplus M[i])$

Return $C[0] \| \ldots \| C[n]$

Algorithm $\mathsf{Dec}(K, C)$

If $(|C| = 0)$ or $(|C| \bmod m \neq 0)$ then return $\perp$

Parse C into m-bit blocks $C[0], \ldots, C[n]$

For $i = 1$ to n do

$M[i] \leftarrow E^{-1}(K, C[i]) \oplus C[i-1]$

Return $M[1] \| \ldots \| M[n]$

FIPS PUB 81 recognizes the fact that, if one fixes the secret key and encrypts a particular message twice, there is a chance that the values of the IV for the two encryptions will end up repeating, thus causing the resulting ciphertext to also repeat. Consequently, it recommends that if the implementor is concerned about such a scenario, he can start the IV with a counter which gets incremented for each encryption.

5.4.3 Cipher Feedback Mode

The cipher feedback (CFB) mode and the rest of the modes presented in this chapter have a similar structure. Specifically, to compute the i-th ciphertext block, we exclusive-or the i-th message block with the output of the i-th application of the block cipher. These modes only differ in what they use as input for each block cipher application.

For the computation of the i-th ciphertext block, the CFB mode uses the $i-1$-st ciphertext block as the input to the block cipher. For the first block, it uses an

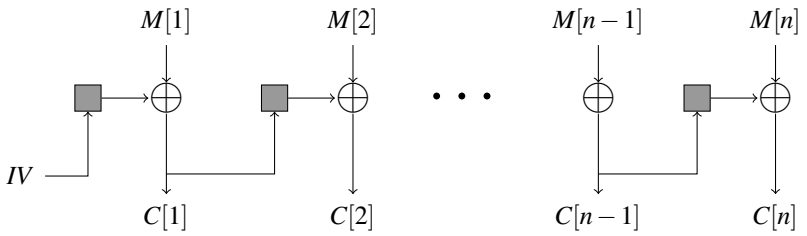


Figure 5.5: The encryption algorithm for the cipher feedback mode (CFB). The gray boxes denote the permutation E_K where E is the underlying block cipher and K is the shared secret key. IV denotes the initial vector. It is assumed here that the message length is a multiple of the block size.

IV. (Again, FIPS PUB 81 does not specify what value to use as the IV here.) The ciphertext transmitted to the receiver is the concatenation of the IV and all of the ciphertext blocks. Note that the CFB mode as specified in FIPS PUB 81 allows the "feedback" to be shorter than one block.

Construction 5.15: CFB with a random IV

Let k and m be positive integers, and let $E : \{0,1\}^k \times \{0,1\}^m \to \{0,1\}^m$ be the underlying block cipher. The encryption scheme $\mathsf{CFBr} = (\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec})$ consists of the following algorithms. The key generation algorithm KG simply returns a bitstring chosen at random from $\{0,1\}^k$. The encryption and decryption algorithms are as follows:

Algorithm $\mathsf{Enc}(K, M)$

If $(|M| = 0)$ or $(|M| \bmod m \neq 0)$ then return $\perp$

Parse M into m-bit blocks $M[1], \ldots, M[n]$

$C[0] \leftarrow IV \xleftarrow{\$} \{0,1\}^m$

For $i = 1$ to n do

$C[i] \leftarrow E(K, C[i-1]) \oplus M[i]$

Return $C[0] \| \ldots \| C[n]$

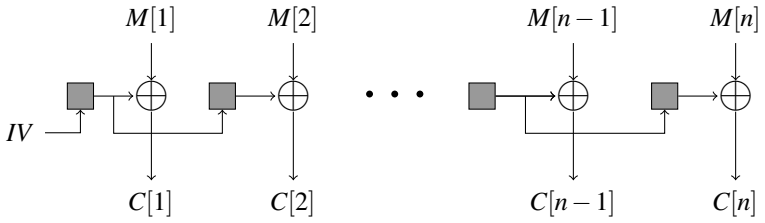


Figure 5.6: The encryption algorithm for the output feedback mode (OFB). The gray boxes denote the permutation E_K where E is the underlying block cipher and K is the shared secret key. IV denotes the initial vector. It is assumed here that the message length is a multiple of the block size.

Algorithm $\mathsf{Dec}(K, C)$

If $(|C| = 0)$ or $(|C| \bmod m \neq 0)$ then return $\perp$

Parse C into m-bit blocks $C[0], \ldots, C[n]$

For $i = 1$ to n do

$M[i] \leftarrow E(K, C[i-1]) \oplus C[i]$

Return $M[1] \| \ldots \| M[n]$

5.4.4 Output Feedback Mode

Similar to CFB, the output feedback (OFB) mode specifies that the i-th ciphertext block be computed by exclusive-oring the i-th message block with the output of the i-th application of the block cipher. For the OFB mode, this output is in turn computed using the result of the $i-1$-st block cipher application as the input. For the first block cipher application, it uses an IV as the input. (Again, FIPS PUB 81 does not specify what value to use as the IV here.) The ciphertext transmitted to the receiver is the concatenation of the IV and all of the ciphertext blocks. Note that the OFB mode as specified in FIPS PUB 81 allows the "feedback" to be shorter than one block.

Construction 5.16: OFB with a random IV

Let k and m be positive integers, and let $E : \{0,1\}^k \times \{0,1\}^m \to \{0,1\}^m$ be

the underlying block cipher. The encryption scheme $\mathsf{OFBr} = (\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec})$ consists of the following algorithms. The key generation algorithm KG simply returns a bitstring chosen at random from $\{0,1\}^k$. The encryption and decryption algorithms are as follows:

Algorithm $\mathsf{Enc}(K, M)$

If $(|M| = 0)$ or $(|M| \bmod m \neq 0)$ then return $\perp$

Parse M into m-bit blocks $M[1], \ldots, M[n]$

$P[0] \leftarrow IV \xleftarrow{\$} \{0,1\}^m$

For $i = 1$ to n do

$P[i] \leftarrow E(K, P[i-1])$; $C[i] \leftarrow P[i] \oplus M[i]$

Return $C[0] \| \ldots \| C[n]$

Algorithm $\mathsf{Dec}(K, C)$

If $(|C| = 0)$ or $(|C| \bmod m \neq 0)$ then return $\perp$

Parse C into m-bit blocks $C[0], \ldots, C[n]$

$P[0] \leftarrow C[0]$

For $i = 1$ to n do

$P[i] \leftarrow E(K, P[i-1])$; $M[i] \leftarrow P[i] \oplus C[i]$

Return $M[1] \| \ldots \| M[n]$

5.4.5 Counter Mode

Similar to OFB, the counter (CTR) mode specifies that the i-th ciphertext block be computed by exclusive-oring the i-th message block with the output of the i-th application of the block cipher. For the CTR mode, this output is in turn computed by applying the block cipher to a (bitstring representation of a) counter. The counter starts at one and gets incremented by one for every application of the block cipher. Note that the encryption algorithm for CTR is a stateful one: the value of the counter is maintained across the encryption of multiple messages. The ciphertext transmitted to

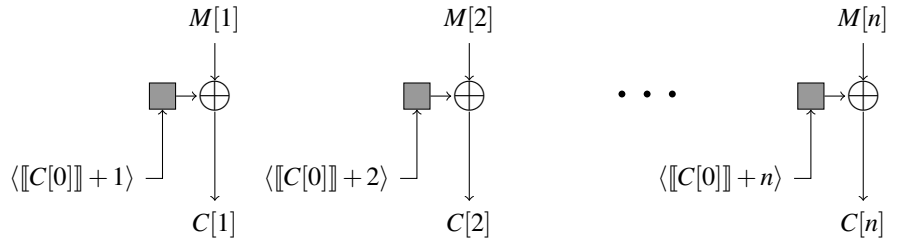


Figure 5.7: The encryption algorithm for the counter mode (CTR). The gray boxes denote the permutation E_K where E is the underlying block cipher and K is the shared secret key. $C[0]$ denotes the binary representation of the counter value for the current message. It is assumed here that the message length is a multiple of the block size.

the receiver is the concatenation of the (bitstring representing the) first counter value used for the current message and all of the ciphertext blocks. Here and for the rest of this book, if x is an integer, then we write $\langle x \rangle$ to denote its bitstring representation (its length is not explicitly specified as it is usually clear from the context). Also, if y is a bitstring, then we write $[\![y]\!]$ to denote the integer value obtained by interpreting y as an unsigned binary number.

Construction 5.17: CTR with a random IV

Let k, m and s be positive integers, and let $E : \{0,1\}^k \times \{0,1\}^m \to \{0,1\}^s$ be the underlying block cipher. The encryption scheme $\mathsf{CTRr} = (\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec})$ consists of the following algorithms. The key generation algorithm KG simply returns a bitstring chosen at random from $\{0,1\}^k$. The encryption and decryption algorithms are as follows:

Algorithm $\mathsf{Enc}(K, M)$

If $(|M| = 0)$ or $(|M| \bmod s \neq 0)$ then return $\perp$

Parse M into s-bit blocks $M[1], \ldots, M[n]$

$C[0] \leftarrow IV \xleftarrow{\$} \{0,1\}^s$

For $i = 1$ to n do

$C[i] \leftarrow E(K, \langle [\![C[0]]\!] + i \rangle) \oplus M[i]$

Return $C[0]\|\ldots\|C[n]$

Algorithm $\mathsf{Dec}(K,C)$

If $(|C| = 0)$ or $(|C| \bmod s \neq 0)$ then return $\perp$

Parse C into s-bit blocks $C[0],\ldots,C[n]$

For $i = 1$ to n do

$M[i] \leftarrow E(K, \langle [\![C[0]]\!] + i\rangle) \oplus C[i]$

Return $M[1]\|\ldots\|M[n]$

5.4.6 Setting the initial vector to a counter value

Constructions 5.14–5.17 specify that the initial vector is to be chosen at random, as denoted by the letter 'r' at the end of the names of the resulting encryption schemes. An alternative is to set the initial vector to a counter value. That is, the encryption algorithm becomes a stateful algorithm with the counter as the state to be maintained across invocations. In the C programming language, this can be achieved via the modifier `static` in the variable declaration. The counter value starts at a fixed, predetermined value then gets incremented for each message that gets encrypted. Specifically, for the CBC, CFB, and OFB modes, the code

$$C[0] \leftarrow IV \xleftarrow{\$} \{0,1\}^m$$

in Constructions 5.14–5.15 is replaced by the following code:

$$\text{static } ctr \leftarrow 0\,;\, C[0] \leftarrow \langle ctr\rangle\,;\, ctr \leftarrow ctr + 1.$$

For the CTR mode (Construction 5.17), the same piece of code is replaced by the following code:

$$\text{static } ctr \leftarrow 0\,;\, C[0] \leftarrow \langle ctr\rangle\,;\, ctr \leftarrow ctr + n\,;\, \text{if } ctr > 2^m - 1 \text{ then return } \perp.$$

For the OFB mode (Construction 5.16), the code

$$P[0] \leftarrow IV \xleftarrow{\$} \{0,1\}^m$$

is replaced by the following code:

$$\text{static } ctr \leftarrow 0\,;\, P[0] \leftarrow \langle ctr \rangle\,;\, ctr \leftarrow ctr + 1.$$

We append the letter 'c' to the names of the resulting encryption schemes to signify that the encryption algorithms use counters as IVs. Thus, applying the above-mentioned changes, we name the resulting schemes $\mathsf{CBCc}, \mathsf{CFBc}, \mathsf{OFBc}$, and CTRc. Consequently, we have just defined a total of 9 schemes. We investigate only some of them here.

5.5 Attacks against block-cipher-based symmetric encryption schemes

In order to see how to measure the level of security offered by a symmetric encryption scheme, we analyze some example schemes in this section. We first start with a scheme that is clearly insecure, namely one that ignores the secret key and simply outputs the plaintext as the ciphertext.

5.5.1 Example: An identity function makes an insecure encryption scheme.

Let k be positive integers. The encryption scheme $\mathsf{SE}_1 = (\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec})$ consists of the following algorithms. The key generation algorithm KG simply returns a bitstring chosen at random from $\{0,1\}^k$. The encryption and decryption algorithms are as follows:

Algorithm $\mathsf{Enc}(K, M)$

  Return M

Algorithm $\mathsf{Dec}(K, C)$

  Return C

An adversary mounting a chosen-plaintext attack against SE_1 can simply submit two different plaintexts to the encryption oracle and look at the result to tell directly

whether the left or the right one was "encrypted." Explicitly, the adversary A's pseudocode is as follows:

Adversary A

$C \xleftarrow{\$} \mathtt{Enc}(0,1)$

If $(C = 0)$ then return 0 else return 1

We analyze the ind-cpa advantage of A by first computing the quantity $\Pr[\mathbf{Exp}^{\text{ind-cpa}}_{\mathsf{SE}_1}(A) \Rightarrow \mathsf{true}]$. Let b be the random variable for the challenge bit b chosen in the experiment (through the subroutine `Initialize`). We have that

$$
\begin{aligned}
&\Pr[\mathbf{Exp}^{\text{ind-cpa}}_{\mathsf{SE}_1}(A) \Rightarrow \mathsf{true}] \\
&= \Pr\left[\mathbf{Exp}^{\text{ind-cpa}}_{\mathsf{SE}_1}(A) \Rightarrow \mathsf{true} \mid b = 1\right] \cdot \Pr[b = 1] \\
&\quad + \Pr\left[\mathbf{Exp}^{\text{ind-cpa}}_{\mathsf{SE}_1}(A) \Rightarrow \mathsf{true} \mid b = 0\right] \cdot \Pr[b = 0] \\
&= \Pr[\mathtt{Enc}(0,1) = 1 \mid b = 1] \cdot \Pr[b = 1] \\
&\quad + \Pr[\mathtt{Enc}(0,1) = 0 \mid b = 0] \cdot \Pr[b = 0] \qquad (5.3) \\
&= \frac{1}{2} \cdot 1 + \frac{1}{2} \cdot 1 \qquad (5.4) \\
&= 1 \,.
\end{aligned}
$$

To see why Equations 5.3 and 5.4 hold, notice that A simply returns the ciphertext obtained from the encryption oracle. This ciphertext, however, always has the same value as the bit b. Substituting 1 into Equation (5.1), we obtain

$$\mathbf{Adv}^{\text{ind-cpa}}_{\mathsf{SE}_1}(A) = 2 \cdot 1 - 1 = 1 \,.$$

Thus, A has an ind-cpa advantage of 1 even though it submits only one encryption query of the length only one bit. The scheme SE_1 is completely insecure.

5.5.2 Example: An encryption scheme that completely reveals the secret key

Let m be positive integers, and let $E : \{0,1\}^m \times \{0,1\}^m \to \{0,1\}^m$ be the underlying block cipher. The encryption scheme $\mathsf{SE}_2 = (\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec})$ consists of the following

algorithms. The key generation algorithm KG simply returns a bitstring chosen at random from $\{0,1\}^m$. The encryption and decryption algorithms are as follows:

Algorithm $\mathsf{Enc}(K,M)$

If $(|M| = 0)$ or $(|M| \bmod m \neq 0)$ then return $\perp$

Parse M into m-bit blocks $M[1],\ldots,M[n]$

For $i = 1$ to n do

$C[i] \leftarrow K \oplus M[i]$

Return $C[1]\|\ldots\|C[n]$

Algorithm $\mathsf{Dec}(K,C)$

If $(|C| = 0)$ or $(|C| \bmod m \neq 0)$ then return $\perp$

Parse C into m-bit blocks $C[1],\ldots,C[n]$

For $i = 1$ to n do

$M[i] \leftarrow K \oplus C[i]$

Return $M[1]\|\ldots\|M[n]$

Observe that here the encryption of 0^m is simply the secret key! Thus, an adversary A can exploit this weakness as follows:

Adversary A

$K \xleftarrow{\$} \mathtt{Enc}(0^m,0^m)$; $C \xleftarrow{\$} \mathtt{Enc}(0^m,1^m)$

If $E_K^{-1}(C) = 0^m$ then return 0 else return 1

There are many ways to attack this scheme, but we present the one above to highlight the fact that, since the constituent algorithms of an encryption scheme is public, there is nothing preventing an adversary from computing the algorithms itself. In this case, since the adversary has obtained the secret key, it can simply decrypts any ciphertext as it wishes.

We analyze the ind-cpa advantage of A by first computing the quantity $\Pr[\mathbf{Exp}^{\text{ind-cpa}}_{\mathsf{SE}_2}(A) \Rightarrow \mathsf{true}]$. Let b be the random variable for the challenge bit b cho-

sen in the experiment (through the subroutine `Initialize`), and let d be the decision A outputs. We have that

$$\begin{aligned}
&\Pr[\mathbf{Exp}^{\text{ind-cpa}}_{\mathsf{SE}_2}(A) \Rightarrow \mathsf{true}] \\
&= \Pr\left[\mathbf{Exp}^{\text{ind-cpa}}_{\mathsf{SE}_2}(A) \Rightarrow \mathsf{true} \mid b = 1\right] \cdot \Pr[b = 1] \\
&\quad + \Pr\left[\mathbf{Exp}^{\text{ind-cpa}}_{\mathsf{SE}_2}(A) \Rightarrow \mathsf{true} \mid b = 0\right] \cdot \Pr[b = 0] \\
&= \Pr[d = 1 \mid b = 1] \cdot \Pr[b = 1] + \Pr[d = 0 \mid b = 0] \cdot \Pr[b = 0] \\
&= \Pr\left[E_{C_1}^{-1}(\mathtt{Enc}(0^m, 1^m)) \neq 0^m \mid b = 1\right] \cdot \Pr[b = 1] \\
&\quad + \Pr\left[E_{C_1}^{-1}(\mathtt{Enc}(0^m, 1^m)) = 0^m \mid b = 0\right] \cdot \Pr[b = 0] \qquad (5.5) \\
&= \frac{1}{2} \cdot 1 + \frac{1}{2} \cdot 1 \qquad (5.6) \\
&= 1 \,.
\end{aligned}$$

Let K be the secret key chosen in the experiment (through `Initialize`). Equations 5.5 and 5.6 hold because, regardless of the value of b,

$$C_1 = \mathtt{Enc}(0^m, 0^m) = K \oplus 0^m = K \,.$$

Thus, $E_{C_1}^{-1}(\mathtt{Enc}(0^m, 1^m)) = 0^m$ if $b = 0$, and $E_{C_1}^{-1}(\mathtt{Enc}(0^m, 1^m)) = 1^m$ otherwise. Substituting 1 into Equation (5.1), we obtain

$$\mathbf{Adv}^{\text{ind-cpa}}_{\mathsf{SE}_2}(A) = 2 \cdot 1 - 1 = 1 \,.$$

Thus, A has an ind-cpa advantage of 1 even though it submits only two encryption queries of the length only m bits each. The scheme SE_2 is completely insecure.

5.5.3 Example: ECB is insecure.

Consider the encryption scheme in the ECB mode as specified in Construction 5.13. There are many different ways to attack this scheme. We present only one. Observe that in this case, if a message contains two repeating blocks (i.e. the first and the second blocks are exactly the same), then the ciphertext will also contain two repeating blocks. Thus, an adversary mounting a chosen-plaintext attack against ECB can

simply submit to the encryption oracle a pair of messages where the left, say, are repeating blocks while the right are two distinct blocks and look at the pattern in the ciphertext to determine whether the left or the right one got encrypted. Explicitly, the adversary A's pseudocode is as follows:

Adversary A

$C \xleftarrow{\$} \mathtt{Enc}(0^m 0^m, 0^m 1^m)$

Parse C as two m-bit blocks $C[1]C[2]$

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

We analyze the ind-cpa advantage of A by first computing the quantity $\Pr[\mathbf{Exp}^{\text{ind-cpa}}_{\mathsf{ECB}}(A) \Rightarrow \mathsf{true}]$. Let b be the random variable for the challenge bit b chosen in the experiment (through the subroutine `Initialize`), and let d be the decision A outputs. We have that

$$
\begin{aligned}
&\Pr[\mathbf{Exp}^{\text{ind-cpa}}_{\mathsf{ECB}}(A) \Rightarrow \mathsf{true}] \\
&= \Pr\left[\mathbf{Exp}^{\text{ind-cpa}}_{\mathsf{ECB}}(A) \Rightarrow \mathsf{true} \mid b = 1\right] \cdot \Pr[b = 1] \\
&\quad + \Pr\left[\mathbf{Exp}^{\text{ind-cpa}}_{\mathsf{ECB}}(A) \Rightarrow \mathsf{true} \mid b = 0\right] \cdot \Pr[b = 0] \\
&= \Pr[d = 1 \mid b = 1] \cdot \Pr[b = 1] + \Pr[d = 0 \mid b = 0] \cdot \Pr[b = 0] \\
&= \Pr[C[1] \neq C[2] \mid b = 1] \cdot \Pr[b = 1] \\
&\quad + \Pr[C[1] = C[2] \mid b = 0] \cdot \Pr[b = 0] \qquad (5.7) \\
&= \frac{1}{2} \cdot 1 + \frac{1}{2} \cdot 1 \qquad (5.8) \\
&= 1 \,.
\end{aligned}
$$

Let K be the secret key chosen in the experiment (through `Initialize`). Equations 5.7 and 5.8 hold because,

- if $b = 0$, then $C[1] = C[2] = E_K(0^m)$, and
- otherwise, $C[1] = E_K(0^m)$ but $C[2] = E_K(1^m) \neq C[1]$.

Substituting 1 into Equation (5.1), we obtain

$$\mathbf{Adv}^{\text{ind-cpa}}_{\mathsf{ECB}}(A) = 2 \cdot 1 - 1 = 1 \ .$$

Thus, A has an ind-cpa advantage of 1 even though it submits only one encryption query of the length only $2m$ bits. The scheme ECB is completely insecure.

5.5.4 Example: CTR mode with the counter repeatedly getting reset

Consider the encryption scheme in the CTR mode as specified in Construction 5.17 except that the code

$$C[0] \leftarrow IV \xleftarrow{\$} \{0,1\}^m$$

is replaced by the following code:

$$ctr \leftarrow 0 \,;\, C[0] \leftarrow \langle ctr \rangle .$$

Specifically, the counter is reset to 0 for each encryption. (Perhaps the programmer forgets to make *ctr* a static variable.) We refer to this encryption scheme as CTR'.

There are many ways to attack CTR'. We present only one. Observe that, since CTR' is both stateless and deterministic, encrypting a single message twice will yield the same ciphertext both times. Thus, an adversary mounting a chosen-plaintext attack against CTR' can simply submit to the encryption oracle two pairs of messages such that they have the same value on the left, say, and different values on the right. If the ciphertexts obtained from the oracle have the same values both times, then the encryption oracle must have encrypted messages on the left. Otherwise, it must have encrypted messages on the right. Explicitly, the adversary A's pseudocode is as follows:

Adversary A

$C_1 \xleftarrow{\$} \mathtt{Enc}(0^m, 0^m) \,;\, C_2 \xleftarrow{\$} \mathtt{Enc}(0^m, 1^m)$

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

We analyze the ind-cpa advantage of A by first computing the quantity $\Pr[\mathbf{Exp}^{\text{ind-cpa}}_{\mathsf{CTR}'}(A) \Rightarrow \mathsf{true}]$. Let b be the random variable for the challenge bit b chosen in the experiment (through the subroutine `Initialize`). We have that

$$\begin{aligned}
&\Pr[\mathbf{Exp}^{\text{ind-cpa}}_{\mathsf{ECB}}(A) \Rightarrow \mathsf{true}] \\
&= \Pr\left[\mathbf{Exp}^{\text{ind-cpa}}_{\mathsf{ECB}}(A) \Rightarrow \mathsf{true} \mid b = 1\right] \cdot \Pr[b = 1] \\
&\quad + \Pr\left[\mathbf{Exp}^{\text{ind-cpa}}_{\mathsf{ECB}}(A) \Rightarrow \mathsf{true} \mid b = 0\right] \cdot \Pr[b = 0] \\
&= \Pr[d = 1 \mid b = 1] \cdot \Pr[b = 1] + \Pr[d = 0 \mid b = 0] \cdot \Pr[b = 0] \\
&= \Pr[C_1 \neq C_2 \mid b = 1] \cdot \Pr[b = 1] \\
&\quad + \Pr[C_1 = C_2 \mid b = 0] \cdot \Pr[b = 0] \qquad (5.9) \\
&= \frac{1}{2} \cdot 1 + \frac{1}{2} \cdot 1 \qquad (5.10) \\
&= 1 \ .
\end{aligned}$$

Let K be the secret key chosen in the experiment (through `Initialize`). Equations 5.9 and 5.10 hold because,

- if $b = 0$, then $C_1 = C_2 = \mathsf{Enc}(K, 0^m)$, and
- otherwise, $C_1 = \mathsf{Enc}(K, 0^m)$ but $C_2 = Enc(K, 1^m) \neq C_1$.

Substituting 1 into Equation (5.1), we obtain

$$\mathbf{Adv}^{\text{ind-cpa}}_{\mathsf{CTR}'}(A) = 2 \cdot 1 - 1 = 1 \ .$$

Thus, A has an ind-cpa advantage of 1 even though it submits only two encryption queries of the length only m bits each. The scheme CTR' is completely insecure.

In fact, the technique used Example 5.5.4 is a general one. Specifically, it can be used to attack any symmetric encryption scheme whose encryption algorithm is deterministic and stateless.

5.5.5 Example: CBC with a random IV

Consider the encryption scheme $\mathsf{CTRr} = (\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec})$ in the CBC mode with a random IV as specified in Construction 5.17. We want to mount a chosen-ciphertext

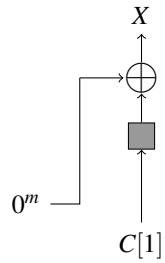


Figure 5.8: The main idea for mounting a chosen-ciphertext attack against CBCr. By setting IV to 0^m, we can apply E_K^{-1} to any value $C[1]$ of our choice by submitting $0^m C[1]$ to the decryption oracle. Specifically, $X = E_K^{-1}(C[1])$.

attack against CTRr. Observe that, if we submit a 2-block ciphertext in which the first block contains only zeros to the decryption oracle, we effectively can ask to see E_K^{-1} applied to any value of our choice. This is illustrated in Figure 5.8. The following simple attack uses this observation.

Adversary A

$C[0]C[1] \xleftarrow{\$} \mathtt{Enc}(0^m, 1^m)$

$X \leftarrow \mathtt{Dec}(0^m C[1])$; $M \leftarrow X \oplus C[0]$

If $M = 0^m$ then return 0 else return 1

The adversary first submits a pair of distinct messages to the encryption oracle, then replaces the first block of the resulting ciphertext with all zeros and submits this modified ciphertext to the decryption oracle. Finally, it takes the output from the decryption oracle and directly compute the underlying message from it.

Now we analyze A's probability of success. Let K be the value of the secret key chosen in the experiment (through `Initialize`). If the encryption oracle always encrypts the left message, then $C[1] = E_K(C[0])$ and consequently $C[0] = E_K^{-1}(C[1])$. Now, we also have that $X = E_K^{-1}(C[1])$. So $X = C[0]$. Thus, $M = X \oplus C[0] = 0^m$, and A always returns 0.

If the encryption oracle always encrypts the right message, then $C[1] = E_K(C[0] \oplus 1^m)$ and consequently $C[0] = E_K^{-1}(C[1]) \oplus 1^m$. Now, we also have that

$X = E_K^{-1}(C[1])$. So $X \oplus C[0] = 1^m \neq 0^m$, and A always returns 1.

Let b be the random variable for the challenge bit b chosen in the experiment (through the subroutine `Initialize`). We have that

$$\begin{aligned}
&\Pr[\mathbf{Exp}^{\text{ind-cca}}_{\mathsf{CTRr}}(A) \Rightarrow \mathsf{true}] \\
&= \frac{1}{2} \cdot \Pr\left[\mathbf{Exp}^{\text{ind-cca}}_{\mathsf{CTRr}}(A) \Rightarrow \mathsf{true} \mid b = 1\right] \\
&\quad + \frac{1}{2} \cdot \Pr\left[\mathbf{Exp}^{\text{ind-cca}}_{\mathsf{CTRr}}(A) \Rightarrow \mathsf{true} \mid b = 0\right] \\
&= \frac{1}{2} \cdot 1 + \frac{1}{2} \cdot 1 = 1
\end{aligned}$$

Substituting 1 into Equation (5.1), we obtain

$$\mathbf{Adv}^{\text{ind-cca}}_{\mathsf{CTRr}}(A) = 2 \cdot 1 - 1 = 1 \ .$$

Thus, A has an ind-cca advantage of 1 even though it submits only one encryption query of the length m bits and one decryption query of length $2m$ bits. The scheme CTRr is not ind-cca secure.

5.6 Examples of security proofs for symmetric encryption

We have seen how to explicitly describe attacks against symmetric encryption schemes and how to measure the effectiveness of the attacks. In this section, we explore some examples of how to "prove" that a block-cipher-based scheme is secure as long as the underlying block cipher is secure. The term "prove" is in quotes here because the approach we take in writing security proofs in this book, namely that of the "provable security" paradigm [5], is considered by some to only provide arguments rather than a proof. Specifically, we show that a scheme is secure as long as its building blocks are secure where the term "secure" has different meanings depending on what particular constructs are being considered.

Provable security was first introduced by Goldwasser and Micali in [27]. It encompasses both design and analysis of cryptographic constructs. In this approach,

one designs a construct based on computationally hard problems such as factoring large composite integers. These hard problems are treated as *atomic primitives*, and the security of the desired construct is based upon the computational hardness of the primitives. Before the construct can be analyzed, however, one must determine what it means for the construct to be considered "secure." This requires precisely defined *security definitions* and *adversary models*. The former should capture the security objectives that the construct is to achieve while the latter should capture the settings in which adversaries operate. The goal here is to capture the settings in which the construct will be deployed in the real-world.

Once security definitions and adversary models are in place, one "proves" security of the desired construct via a *reduction* from the hardness of the underlying primitives, similar to the way one reduces the satisfiability problem (SAT) to a problem to prove that the problem is NP-complete. In this approach, one does not prove that a construct is secure. Rather, one provides a reduction of the security of the construct from that of its underlying primitives. This technique allows us to arrive at a powerful conclusion: the *only* way to defeat the desired construct in the prescribed models is to break the underlying primitives. Thus, as long as the underlying primitives have not been broken (i.e. have not been shown to be solvable in a reasonable amount of time), we know that the construct is secure under the prescribed security definitions and adversary models.

5.6.1 Security proofs for CTRc

In this section, we prove the security of the scheme CTRc. First, we explicitly specify the scheme in its entirety here for convenience. Then, the theorem that follows says that the encryption scheme in the CTR mode in which the counter is a static counter is secure as long as the underlying family of functions is secure.

Construction 5.18: CTR with a counter value

Let k, m and s be positive integers, and let $E : \{0,1\}^k \times \{0,1\}^m \to \{0,1\}^s$ be the underlying family of functions. The encryption scheme $\mathsf{CTRc} = (\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec})$ consists of the following algorithms. The key generation algorithm KG simply returns a bitstring chosen at random from $\{0,1\}^k$. The encryption and decryption algorithms are as follows:

Algorithm $\mathsf{Enc}(K, M)$

If $(|M| = 0)$ or $(|M| \bmod s \neq 0)$ then return $\perp$

Parse M into s-bit blocks $M[1], \ldots, M[n]$

static $ctr \leftarrow 0$; $C[0] \leftarrow \langle ctr \rangle$; $ctr \leftarrow ctr + n$;

If $ctr > 2^m - 1$ then return $\perp$

For $i = 1$ to n do

$C[i] \leftarrow E(K, \langle [\![C[0]]\!] + i \rangle) \oplus M[i]$

Return $C[0] \| \ldots \| C[n]$

Algorithm $\mathsf{Dec}(K, C)$

If $(|C| = 0)$ or $(|C| \bmod s \neq 0)$ then return $\perp$

Parse C into s-bit blocks $C[0], \ldots, C[n]$

For $i = 1$ to n do

$M[i] \leftarrow E(K, \langle [\![C[0]]\!] + i \rangle) \oplus C[i]$

Return $M[1] \| \ldots \| M[n]$

Theorem 5.19: Security of CTRc

Let k, m, and s be positive integers, and let $E : \{0,1\}^m \times \{0,1\}^m \to \{0,1\}^s$ be a family of functions. Let $\mathsf{CTRc} = (\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec})$ be the encryption scheme per Construction 5.18. Then, for any adversary A running in time t making at most q `Enc` queries of length totalling at most μ bits, we can construct an adversary B

such that

$$\mathbf{Adv}^{\text{ind-cpa}}_{\mathsf{CTRc}}(A) \leq 2 \cdot \mathbf{Adv}^{\text{prf}}_{E}(B) \tag{5.11}$$

and B runs in time $O(t)$ and makes at most μ/s `Enc` queries of length totalling at most μ bits.

One way to interpret this theorem is the following. First, suppose that the underlying family of functions E is secure. This means that the advantage of any adversary attacking E using a reasonable amount of resources is small. Equation (5.11) then implies that the advantage of any adversary attacking CTRc using a reasonable amount of resources must also be small.

We prove this theorem using the following plan.

1. Given A, we construct B whose strategy is simply to run A using its oracle `g` to simulate the encryption algorithm so that it can generate answers to A's `Enc` queries.
2. We argue that, if `g` is chosen at random from E, then B's probability of success (in $\mathbf{Exp}^{\text{prf}}_{E}(B)$) is at least as much as that of A (in $\mathbf{Exp}^{\text{ind-cpa}}_{\mathsf{CTRc}}(A)$).
3. We argue that, on the other hand, if `g` is chosen at random from the set of all possible functions mapping $\{0,1\}^m$ to $\{0,1\}^s$, then B's probability of success is at least 1/2.

Proof of Theorem 5.19

Let A be an adversary mounting a chosen-plaintext attack against the privacy of CTRc. We first define a subroutine used by B in Figure 5.9 then define B as follows.

Adversary B

$b \xleftarrow{\$} \{0,1\}$

Run A, answering its oracle queries as follows:

Subroutine $\texttt{SimEnc}(M_0, M_1, b)$
  If $|M_0| \neq |M_1|$ then return $\perp$
  $M \leftarrow M_b$
  If $(|M| = 0)$ or $(|M| \bmod s \neq 0)$ then return $\perp$
  Parse M into s-bit blocks $M[1], \ldots, M[n]$
  static $ctr \leftarrow 0$; $C[0] \leftarrow \langle ctr \rangle$; $ctr \leftarrow ctr + n$;
  If $ctr > 2^m - 1$ then return $\perp$
  For $i = 1$ to n do
    $C[i] \leftarrow \mathsf{g}(\langle [\![C[0]]\!] + i \rangle) \oplus M[i]$
  Return $C[0] \| \ldots \| C[n]$

Figure 5.9: Subroutine `SimEnc` used by B.

On query $\texttt{Enc}(M_0, M_1)$:

Return $\texttt{SimEnc}(M_0, M_1, b)$ to A

Until A halts and outputs d

If $d = b$ then return 1 else return 0

Our goal is to obtain Equation (5.11). Recall that from Definitions 3.8 and 5.11,

$$\mathbf{Adv}^{\text{prf}}_{E}(B) = 2 \cdot \Pr[\mathbf{Exp}^{\text{prf}}_{E}(B) \Rightarrow \mathsf{true}] - 1 \ , \text{ and}$$
$$\mathbf{Adv}^{\text{ind-cpa}}_{\mathsf{CTRc}}(A) = 2 \cdot \Pr[\mathbf{Exp}^{\text{ind-cpa}}_{\mathsf{CTRc}}(A) \Rightarrow \mathsf{true}] - 1 \ ,$$

respectively. Thus, for Equation (5.11) to hold, we need to show that

$$\Pr[\mathbf{Exp}^{\text{prf}}_{E}(B) \Rightarrow \mathsf{true}] \ \geq \ \frac{1}{2} \cdot \Pr[\mathbf{Exp}^{\text{ind-cpa}}_{\mathsf{CTRc}}(A) \Rightarrow \mathsf{true}] + \frac{1}{4} \ . \tag{5.12}$$

For brevity, we refer to the experiments $\mathbf{Exp}^{\text{ind-cpa}}_{\mathsf{CTRc}}(A)$ and $\mathbf{Exp}^{\text{prf}}_{E}(B)$ as A's experiment and B's experiment, respectively. Consider B's experiment. Let "g `real`" denote the event in which g is a function chosen at random from E, and let "g `rand`" denote the event in which g is a function chosen at random from the set of all possible functions mapping $\{0,1\}^m$ to $\{0,1\}^s$. Now, we consider the quantity on the left of

Equation (5.12).

$$\begin{aligned}
&\Pr[\mathbf{Exp}^{\text{prf}}_E(B) \Rightarrow \texttt{true}] \\
&= \Pr[\texttt{g real}] \cdot \Pr\left[\mathbf{Exp}^{\text{prf}}_E(B) \Rightarrow \texttt{true} \mid \texttt{g real}\right] \\
&\quad + \Pr[\texttt{g rand}] \cdot \Pr\left[\mathbf{Exp}^{\text{prf}}_E(B) \Rightarrow \texttt{true} \mid \texttt{g rand}\right] \\
&= \frac{1}{2} \cdot \Pr\left[\mathbf{Exp}^{\text{prf}}_E(B) \Rightarrow \texttt{true} \mid \texttt{g real}\right] \\
&\quad + \frac{1}{2} \cdot \Pr\left[\mathbf{Exp}^{\text{prf}}_E(B) \Rightarrow \texttt{true} \mid \texttt{g rand}\right] \qquad (5.13)
\end{aligned}$$

and make the following claims:

Claim 5.20:

$$\Pr\left[\mathbf{Exp}^{\text{prf}}_E(B) \Rightarrow \texttt{true} \mid \texttt{g real}\right] = \Pr[\mathbf{Exp}^{\text{ind-cpa}}_{\mathsf{CTRc}}(A) \Rightarrow \texttt{true}]$$

Claim 5.21:

$$\Pr\left[\mathbf{Exp}^{\text{prf}}_E(B) \Rightarrow \texttt{true} \mid \texttt{g rand}\right] \geq \frac{1}{2}$$

Applying both claims to Equation (5.13), we obtain Equation (5.12) as desired. Now, we justify the claims.

Claim 5.20 holds because, when "`g real`" occurs, B in fact simulates A exactly as A's experiment. Thus, the probability that B outputs 1 (which is the correct answer in this case) is the same as the probability that A guesses correctly whether B encrypts the left or the right messages. This probability is the probability that A succeeds in A's experiment.

Now we justify Claim 5.21. The probability that B outputs 0 (which is the correct answer in this case) is the same as the probability that A guesses incorrectly whether B encrypts the left or the right messages. We will argue that, even after seeing B's answers to its queries, A learns absolutely nothing about whether the encryption was

done on the left or the right messages. This in turn means that A can do no better than guessing the value of the bit b chosen by B. So the probability that A guesses incorrectly when g `rand` occurs is at least 1/2.

To see why A learns nothing useful from B's answers to its queries, notice that the inputs to the block cipher never repeat throughout the experiment. Specifically, if the j-th encryption query contains $l[j]$ blocks (for $1 \leq j \leq q$), then the counter *ctr* will have the values

$$\begin{array}{llll} 1, & 2, & \ldots, & l[1], \\ l[1]+1, & l[2]+2, & \ldots, & l[1]+l[2], \\ \vdots & & & \\ \sum_{j=1}^{q-1} l[j]+1, & \sum_{j=1}^{q-1} l[j]+2, & \ldots, & \sum_{j=1}^{q} l[j]\,. \end{array}$$

Since the encryption algorithm ensures that *ctr* never exceeds $2^m - 1$, its values will never repeat. Now, notice that, when g is a function chosen at random from the set of all functions mapping $\{0,1\}^m$ to $\{0,1\}^s$, the fact that the inputs to g never repeat means that its outputs are independent. Thus, it matters not whether B encrypts the left or the right messages because the answers always contain only blocks of random, independent values regardless. In short, A's view of the world is the same in both cases.

It remains to justify B's resource usage. The running time of B is taken up by some bookkeeping and the time it takes to submit queries to the oracle g. The number of queries that B submits to g is the number of blocks of plaintexts to be encrypted. This is at most μ/s. The total number of bits submitted by B is equal to that submitted by A which in turn is equal to μ. Thus, the theorem follows.

5.7 Conclusion

One of the most widely used primitives in cryptography is symmetric encryption schemes based on block ciphers or the so-called modes of operation. In this chapter, we present the security definitions for symmetric encryption, namely the indistin-

guishability against chosen-plaintext attacks (IND-CPA) and the indistinguishability against chosen-ciphertext attacks (IND-CCA) security notions. It is standard practice nowadays that any serious proposal of modes of operation for symmetric encryption include provable security results with respect to at least the IND-CPA notion. It must be emphasized, however, that often these security results hold under particular conditions, and care must be taken during implementation to ensure that these conditions are met. Specifically, for many of the existing modes, an initialization vector (IV) is a crucial ingredient in that the way it is chosen directly affects the security guarantees that an implementation of a mode can actually provide. Often, there are many different natural choices for how to choose an IV. Making appropriate choices is often tricky and must be carried out with care.

5.8 Credits

The analysis of the security of CTRc is due to Mihir Bellare, Anand Desai, Eric Jokipii, and Phillip Rogaway [8].

5.9 Exercises

Exercise 5.1. Why is security against key recovery insufficient for encryption schemes? Explain in one paragraph.

Exercise 5.2. What property does the correctness condition for symmetric encryption schemes capture? Explain in your own words.

Exercise 5.3. Suppose that in your application, the message that needs to be encrypted is always of 128 bits. Explain why it would be a bad idea to simply use (the forward direction of) AES-128 to encrypt each message instead of using one of the better modes of operation.

Exercise 5.4. What is an “initialization vector”? Why does it matter so much to the security of a symmetric encryption scheme?

Exercise 5.5. Consider the symmetric encryption scheme $\mathsf{SE} = (\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec})$ whose key generation algorithm and encryption algorithm work as follows:

Algorithm KG	Algorithm $\mathsf{Enc}(K, M)$
$K \xleftarrow{\$} \{0,1\}^{56}$	If $\lvert M\rvert \neq 64$ then return $\perp$
Return K	Return $\mathsf{DES}(K, M \oplus 1^{64})$

Write the decryption algorithm so that SE satisfies the correctness condition for symmetric encryption schemes.

Exercise 5.6. Suppose x is m bits long. Let $\overline{x} = x \oplus 1^m$. Consider an symmetric encryption scheme $\mathsf{SE} = (\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec})$ whose key generation algorithm and encryption algorithm work as follows:

Algorithm KG	Algorithm $\mathsf{Enc}(K, M)$
$K \xleftarrow{\$} \{0,1\}^{64}$	If $M = 1^{64}$ then $C \leftarrow M \oplus K$ else $C \leftarrow M \oplus \overline{K}$
Return K	Return C

Explain what happens when $C = \overline{K}$.

Exercise 5.7. Let k and l be non-negative integers. Let $E\colon \{0,1\}^k \times \{0,1\}^l \to \{0,1\}^l$ be a family of permutations. Let $\mathsf{SE} = (\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec})$ be the CBC encryption scheme with 0^l being the initialization vector (IV). We specify the definitions of the algorithms of SE in full here to avoid ambiguity. The key generation algorithm simply outputs a randomly-chosen bit string of length k. The encryption and decryption algorithms work as follows.

Algorithm $\mathsf{Enc}_K(M)$

If $|M| \bmod l \neq 0$ then return $\perp$

Parse M as l-bit blocks $M[1] \ldots M[n]$

$C[0] \leftarrow 0^l$

For $i = 1$ to n do

$C[i] \leftarrow E_K(M[i] \oplus C[i-1])$

Return $C[0] \ldots C[n]$

Algorithm $\mathsf{Dec}_K(C)$

If $|C| \bmod l \neq 0$ then return $\perp$

Parse C as l-bit blocks $C[0] \ldots C[n]$

For $i = 1$ to n do

$M[i] \leftarrow E_K^{-1}(C[i]) \oplus C[i-1])$

Return $M[1] \ldots M[n]$

1. Is SE a secure symmetric encryption scheme under the IND-CPA definition?
2. Prove your answer. If your answer is yes, you need to show a reduction and derive the relationship between the insecurity of E and that of SE obtained from the reduction. If your answer is no, you need to write the pseudocode for an adversary, analyze its advantage, and specify its resource usage. The less the resource usage and the higher the advantage, the better.

6

Building Block III: Hash functions

6.1 Motivation

Hash functions are one of the most versatile cryptographic primitives. Not to be confused with hash tables in the study of data structures, cryptographic hash functions compress data down to what is often referred to as its digest. If the data is a password, the digest is often stored in the database for later authentication of the user who owns the password. If the data is a document, the digest is then authenticated (symmetrically by generating a message authentication code or asymmetrically by generating a digital signature) for later verification. Figure 6.1 illustrates the general concept of a hash function.

In this chapter, we discuss the different models for hash functions in theory and in practice and the corresponding security notions. Then, we describe some popular

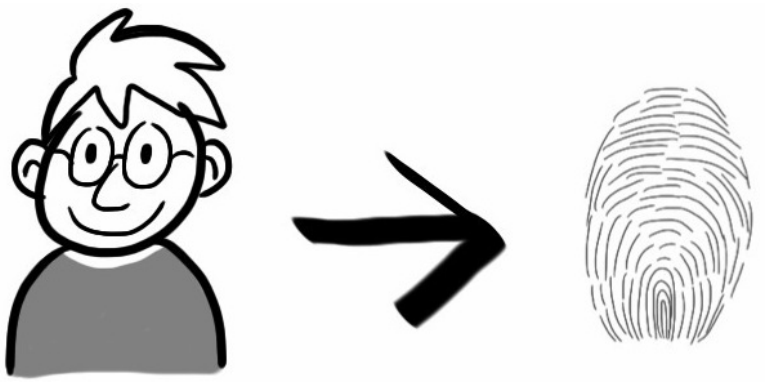


Figure 6.1: A way to think about hash functions. The digest is supposed to capture the essense of the object being hashed.

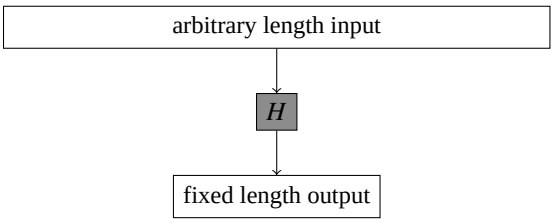


Figure 6.2: A hash function H. The output is often called a message digest.

hash functions and present attacks against simple examples and well-known hash functions.

6.2 What it is

A hash function takes an arbitrary length input and returns a fixed length output. Figure 6.2 illustrates this simple concept. Often, the input is longer than the output, and thus, it is often said that a hash function generates **message digests**. In practice, there is a limit, albeit a large one, on the length of the input. Some well-known hash functions include the MD4, MD5, RIPEMD, SHA-1, and SHA-2.

In practice, a hash function has one input, namely the message, and one output, namely the digest. In modern cryptography theory, however, hash functions have traditionally been modelled as families of functions. Specifically, a hash function takes two inputs, namely a "key" and an input message, and returns the digest of the message. We emphasize that although the first input is referred to as a "key," it is *not* a secret value but a public one. In current literature, a concrete theorem statement concerning the security of a cryptographic construct based on a hash function almost always refers to the hash function as a family of functions rather than a single function. This means that the SHA-1 hash, for example, has the signature $\{0,1\}^{160} \times \{0,1\}^{<2^{64}} \rightarrow \{0,1\}^{160}$. (More details on SHA-1 can be found in Section 6.4.2).

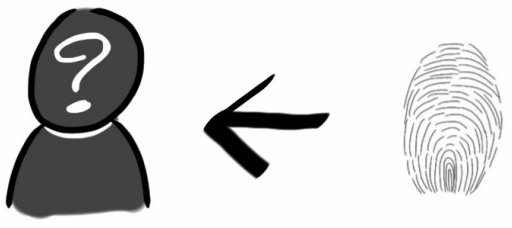


Figure 6.3: Roughly, preimage resistance means that it should be difficult to figure out what was hashed simply by looking at the hash value.

6.3 What security means

Since hash functions are so versatile, many security properties are expected out of them. The following are the most widely known ones. Let H be a hash function.

1. Preimage resistance. Given a hash value h, it should be difficult to find a preimage x such that $H(x) = h$. Figure 6.3 illustrates this concept.

2. Second preimage resistance (also known as weak collision resistance). Given a preimage x, it should be difficult to find another preimage y such that $x \neq y$ but $H(x) = H(y)$.

3. Collision resistance (also known as strong collision resistance). It should be difficult to find preimages x and y such that $x \neq y$ but $H(x) = H(y)$. Since a hash function maps elements from a large set into one in a much smaller set, it is natural that a collision would occur. Figure 6.4 illustrates a scenario where mapping two points in the domain does not result in a collision while Figure 6.5 illustrates a scenario where a collision does occur.

These security definitions, although prevalent, are ambiguous in many ways. For example, when the hash function in question is modelled as a family of functions, it is subject to interpretation whether the definitions are defined with respect to a function chosen by the adversary or a function chosen at random from the family. Similarly, it is unclear whether the target points are chosen by the adversary or are chosen at random.

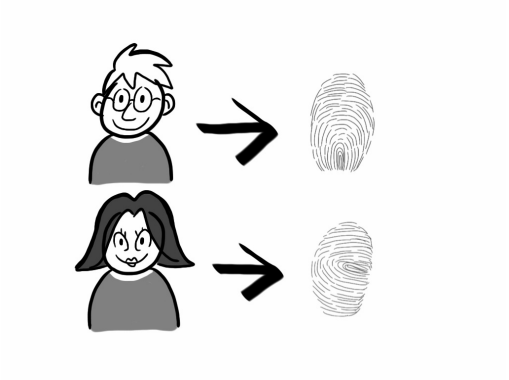


Figure 6.4: For collision resistant hash functions, one would expect that, in most cases, hashing two distinct input values would result in two different hash values.

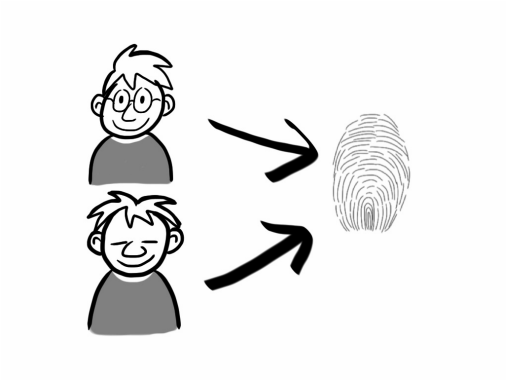


Figure 6.5: A collision can always occur, however, since a hash function maps values from a large set to those in a smaller set.

We will follow Phillip Rogaway's approach in modelling hash functions as single functions here. The reason is that this approach gets us closer to how hash functions are defined and used in practice in contrast to the approach of modelling hash functions as families of functions. We state three security definitions explicitly as games here.

Definition 6.22: Resistance against pre-image attacks for hash functions

Let m, n be integers such that $m > n$. Let $\mathsf{H} : \{0,1\}^m \to \{0,1\}^n$ be a hash function, and let A be an adversary. We define the following subroutines, exper-

iments, and advantage function.

Subroutines

Subroutine `Initialize`

$x \xleftarrow{\$} \{0,1\}^m$; $h \leftarrow \mathsf{H}(x)$

Return h

Subroutine `Finalize`(x')

Return $(\mathsf{H}(x') = h)$

Experiment

Experiment $\mathbf{Exp}^{\text{pre}}_{\mathsf{H}}(A)$

$h \leftarrow$ `Initialize`

$x' \xleftarrow{\$} A(h)$

Return `Finalize`(x')

We define the **pre-image advantage** of an adversary A mounting a pre-image attack against H as

$$\mathbf{Adv}^{\text{pre}}_{\mathsf{H}}(A) = \Pr[\mathbf{Exp}^{\text{pre}}_{\mathsf{H}}(A) \Rightarrow \mathsf{true}] \,. \tag{6.1}$$

Definition 6.23: Resistance against second pre-image attacks for hash functions

Let m, n be integers such that $m > n$. Let $\mathsf{H} : \{0,1\}^m \to \{0,1\}^n$ be a hash function, and let A be an adversary. We define the following subroutines, experiments, and advantage function.

Subroutines

Subroutine `Initialize`

$x \xleftarrow{\$} \{0,1\}^m$; Return x

Subroutine `Finalize`(x')

Return $(\mathsf{H}(x) = \mathsf{H}(x') \wedge x \neq x')$

Experiment

Experiment $\mathbf{Exp}^{\text{sec}}_{\mathsf{H}}(A)$

$x \leftarrow$ `Initialize`

$x' \xleftarrow{\$} A(x)$

Return `Finalize`(x')

We define the **second pre-image advantage** of an adversary A mounting a second pre-image attack against H as

$$\mathbf{Adv}^{\text{sec}}_{\mathsf{H}}(A) = \Pr\left[\mathbf{Exp}^{\text{sec}}_{\mathsf{H}}(A) \Rightarrow \text{true}\right]. \tag{6.2}$$

Definition 6.24: Collision resistance of hash functions

Let m, n be integers such that $m > n$. Let $\mathsf{H} : \{0,1\}^m \to \{0,1\}^n$ be a hash function, and let A be an adversary. We define the following subroutines, experiments, and advantage function.

Subroutines

Subroutine `Initialize`

Return

Subroutine `Finalize`(x, x')

Return $(\mathsf{H}(x) = \mathsf{H}(x') \wedge x \neq x')$

Experiment

Experiment $\mathbf{Exp}^{\text{coll}}_{\mathsf{H}}(A)$

`Initialize`

$(x, x') \xleftarrow{\$} A$

Return `Finalize`(x, x')

We define the **collision advantage** of an adversary A mounting a collision attack against H as

$$\mathbf{Adv}^{\text{coll}}_{\mathsf{H}}(A) = \Pr[\mathbf{Exp}^{\text{coll}}_{\mathsf{H}}(A) \Rightarrow \text{true}] \,. \tag{6.3}$$

6.3.1 Going beyond the standard model

Hash functions are among the most versatile cryptographic building blocks around. Many common cryptographic schemes and protocols, including encryption, digital signatures, and identity authentication schemes, rely on hash functions. Proofs attesting the security of these schemes are often devised in the following manner. Suppose a scheme S is constructed based on a hash function H. Suppose further that, for S to be secure, an appropriate security property that we need from H to make the proof go through is P. Then, the security statement about the security of S under the notion Q often takes on the following form:

If H is secure under the notion P, then S is secure under the notion Q.

As it turns out, for many different schemes S, assuming that the underlying hash functions are preimage resistant or collision resistant is not sufficient to make the proofs go through. When this happens, researchers writing security proofs for the schemes often end up either handcrafting a new security definition P to capture the properties of the underlying hash function that would be sufficient in proving the security of S or coming up with ways to get around the problem completely. Here we the most commonly used approach, namely the random oracle model.

The random oracle model

There are many cryptographic constructs in the literature that use hash functions as building blocks but that so far cannot be proven secure assuming only that the underlying hash functions are collision resistant or preimage resistant. Many of these constructs get around the problem by using the **random oracle model** proposed by Bellare and Rogaway [11]. In this model, the adversary may not invoke the underlying hash function directly even though it is public. Instead, the hash function is modelled as an oracle that can be queried by any party in the game. Additionally, the way this oracle works is to reply with a value chosen uniformly at random from all possible outputs whenever the input is a value that has never been queried before. For old input values, the oracle simply outputs the old answer as a reply. This allows the hash function to be appropriately programmed in the proof and makes many of the proofs possible.

A proof in the random oracle model however does not guarantee security. Specifically, many articles show schemes that are constructed based on hash functions so that the schemes (1) can be proven secure in the random oracle model but (2) are completely insecure no matter how the underlying hash functions are instantiated. Although these schemes are often contrived and handcrafted specifically to obtain this result, they show that a proof in the random oracle is not as strong a result as one would hope and should be considered a heuristic.

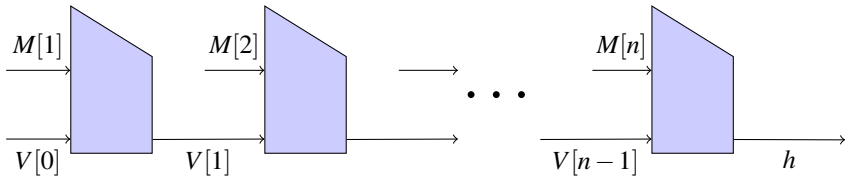


Figure 6.6: The Merkle-Damgard construciton. The trapezoids represent a compression function. The initial vector $V[0]$ is a fixed, public value. The messages $M[1], \ldots, M[n]$ are fixed-length blocks of the padded input message. The output is the output h of the compression function on the last round.

Nonetheless, the random oracle model remains a useful and popular approach for designing cryptographic constructs. In fact, for some constructs, the only known provably secure constructions are in the random oracle model.

6.4 Hash function design

The most common approach to designing hash functions is to use what is commonly referred to as the **Merkle-Damgard** construction. In this approach, one uses a **compression function** as a building block and builds a hash function from it. The hope is that if the underlying compression function offers good security properties, then so will the resulting hash function.

As the name suggests, a compression function essentially compresses its data input. Specifically, given a fixed-length message block to be compressed and a fixed-length **chaining variable** whose length is shorter than the message block, its output is of the same length as that of the chaining variable. Starting from some initial value for the chaining variable, the Merkle-Damgard construction chops the input message into blocks and processes each of the message blocks in turn. If the input message does not end on the block boundary, it is first padded before being processed. The initial value for the chaining variable is often referred to as the **initial vector**. A depiction of this process is shown in Figure 6.6.

The two most common ways of constructing a compression function are to build

it from scratch specifically for the purpose of constructing a particular hash function or to build it using a block cipher as the building block. Most of the popular hash functions use the former approach. We will describe some of them briefly here.

6.4.1 MD-5

MD5 uses a compression function that takes as inputs a 512-bit message block and a 128-bit chaining variable and outputs a 128-bit value. The compression function operates in 4 rounds each of which consists of a sequence of 16 steps. The execution of each step involves elementary operations such as modular addition, bit shifting, logical operations such as exclusive or, negation, and, and or.

The padding scheme used by MD5 works as follows. Let l be the length, specified as a 64-bit binary number, of the message M to be hashed. If M is longer than 2^{64} bits, then l is the low-order 64 bits of the actual length of M. Then, the input M is padded with $10\cdots0||l$ so that its length is a multiple of 512. The Merkle-Damgard construction is then applied. Specifically, the padded message is chopped into blocks of 512 bits. The chunks are then processed from left to right. The main operation in each round is to apply the compression function to the 512-bit chunk for the current round concatenated with the output from the previous iteration. As usual, the initial vector is a fixed, publicly known constant. The resulting hash function maps $\{0,1\}^*$ to $\{0,1\}^{160}$.

6.4.2 SHA-1

We will now provide a brief description of the SHA-1 hash function [42, 43], one of the most well-known hash functions based on the Merkle-Damgard construction. Attacks against it have been published and improved upon since 2005, however, Its replacement is being sought through a competition run by the US National Institute of Standards and Technology (NIST) in search of what is to be the SHA-3 hash function.

SHA-1 uses a compression function that takes as inputs a 512-bit message block

and a 160-bit chaining variable and outputs a 160-bit value. The compression function operates in 4 rounds each of which consists of a sequence of 20 steps. The execution of each step involves elementary operations such as modular addition, bit shifting, logical operations such as exclusive or, negation, and, and or.

The padding scheme used by SHA-1 works as follows. Let l be the length, specified as a 64-bit binary number, of the message M to be hashed. The input M is first padded with $10\cdots0||l$ so that it is a multiple of 512. The Merkle-Damgard construction is then applied. Specifically, the padded message is chopped into blocks of 512 bits. The chunks are then processed from left to right. The main operation in each round is to apply the compression function to the 512-bit chunk for the current round concatenated with the output from the previous iteration. As usual, the initial vector is a fixed, publicly known constant. The resulting hash function SHA-1 maps $\{0,1\}^{<2^{64}}$ to $\{0,1\}^{160}$.

6.5 Conclusion

Hash functions are another commonly used cryptographic primitive. Most web applications that require user login often use hash functions to protect the users' passwords. The most common security properties that are expect of hash functions are the preimage security (also known as one-wayness) and collision resistance. We explore these concepts in this chapter. We emphasize, however, that, in today's climate, one would need more than a straight application of a hash function (e.g. $\mathsf{H}(\mathsf{password})$) to secure passwords. As is the case with any other application of low-level cryptographic primitives, the task of designing a protocol that uses a hash function to implement secure password protection or to implement secure password-based authentication requires careful consideration about the power and the goal of the adversaries. For an example of how the secure password hashing problem was treated in modern cryptography, see the paper by David Wagner and Ian Goldberg [46].

6.6 Credits

In this book, hash functions are modelled as keyless following an approach advocated by Phillip Rogaway [37]. This is in contrast to the conventional practice of modelling a hash function as being a family of functions indexed by the key bits. See [37] for discussions comparing the two approaches and the justifications for moving away from the conventional model to the keyless model.

6.7 Exercises

Exercise 6.1. Explain why a hash function cannot possibly be a permutation.

Exercise 6.2. Why is pre-image resistance for hash functions considered desirable? Describe an application of hash functions that would be negatively affected if the hash function used is easily invertible. Explain the negative affects resulting from the lack of pre-image resistance of the hash function.

Exercise 6.3. Why is a hash collision considered undesirable? Describe an application of hash functions that would be negatively affected if the hash function used is not collision resistant. Explain the negative affects resulting from the lack of collision resistance of the hash function.

Exercise 6.4. Consider the following statement.

> A hash function that is insecure against pre-image attacks (see Definition 6.22) cannot possibly be collision resistant (see Definition 6.24).

Is this statement correct? Prove your answer.

Exercise 6.5. Consider the following statement.

> A hash function that is insecure against second pre-image attacks (see Definition 6.23) cannot possibly be collision resistant (see Definition 6.24).

Is this statement correct? Prove your answer.

Exercise 6.6. Let k, l, and L be non-negative integers. Let $E\colon \{0,1\}^k \times \{0,1\}^l \to \{0,1\}^L$ be a family of functions. Let $\mathsf{H}\colon \{0,1\}^* \to \{0,1\}^L$ be a public hash function and padwithzeros be a subroutine defined as follows:

Algorithm $\mathsf{H}(M)$

If $|M| \bmod l \neq 0$ then $M \leftarrow \mathsf{padwithzeros}(M)$

Parse M as l-bit blocks $M[1] \ldots M[n]$

For $i = 1$ to n do $Y[i] \leftarrow E_{0^k}(M[i])$

$T \leftarrow Y[1] \oplus \ldots \oplus Y[n]$

Return T

Subroutine $\mathsf{padwithzeros}(M)$

$m \leftarrow l - (|M| \bmod l)$

Return $M \| 0^m$

1. Is H a secure hash function under the collision resistance definition (Definition 6.24)?

2. Prove your answer. If your answer is yes, you need to show a reduction and derive the relationship between the insecurity of E and that of H obtained from the reduction. If your answer is no, you need to write the pseudocode for an adversary, analyze its advantage, and specify its resource usage. The less the resource usage and the higher the advantage, the better.

Exercise 6.7. Let $\mathsf{H}_1\colon \{0,1\}^* \to \{0,1\}^{160}$ and $\mathsf{H}_2\colon \{0,1\}^* \to \{0,1\}^{160}$ be collision-resistant, public hash functions as per Definition 6.24. Consider $\mathsf{H}_3\colon \{0,1\}^* \to \{0,1\}^{320}$ defined as follows: for any $M \in \{0,1\}^*$,

$$\mathsf{H}_3(M) = \mathsf{H}_1(M) \| \mathsf{H}_2(M) \ .$$

Must H_3 necessarily be collision-resistant as per Definition 6.24? Briefly explain your answer. No proof needed.

Exercise 6.8. Let k, n be positive integers, and let $F : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$ be a pseudorandom family of functions. Consider the following hash function H based on F.

Algorithm $\mathsf{H}(M)$

If $|M| \bmod n \neq 0$ then return $\perp$

Parse M as n-bit blocks $M[1] \| \cdots \| M[m]$

Return $F_{0^k}(M[1]) \oplus \cdots \oplus F_{0^k}(M[m])$

Do the following problems.

1. Find a collision for H. Specifically, find two different messages M_1 and M_2 that hash to the same value.
2. Show in detail how M_1 and M_2 in your answer above hash to the same value.

7

Message Authentication Codes

As previously discussed, cryptography is not only about keeping secrets. In this chapter, we study a construct used to accomplish a different goal: how to ensure the authenticity of the transmitted data. In this chapter, we introduce message authentication codes, a cryptographic construct used to achieve authenticity in the symmetric setting. Specifically, we specify its syntax and correctness condition, look at some example constructions, and analyze the security of some of them.

7.1 Motivation

Consider two parties, say, Alice and Bob, who are trying to communicate in such a way that, when Alice wants to send a message to Bob, she wants to make sure that Bob has a way to check that the data he received is as she intended. If an adversary, say, Eve, modifies Alice's message in transit or concocts another message herself and sends it to Bob, then Bob should be able to tell right away that something is amiss.

Even though the privacy goal may be the first goal that comes to mind when people mention cryptography, authenticity (also referred to as data integrity) is in fact a more common goal: it is often desirable to be able to be sure that the data one receives is in fact as the sender has intended. Public announcements, for example, are data that are intended to be made public (and so privacy is not the goal) but must necessarily be verifiable both in terms of their contents and the sources from which they originate.

In the symmetric setting, a Message Authentication Code (MAC) scheme is a construct that can help achieve authenticity. Given the secret key and a message to be sent, the sender runs an algorithm to generate a tag corresponding to this particular message. She then sends both the message and the tag to the recipient(s), who can then use the tag and the secret key to verify whether the message is in fact as the sender intended to send or has been forged or modified.

7.2 What it is: syntax and correctness condition

Now we specify the syntax and correctness condition for MAC schemes. All data being manipulated as inputs and outputs of algorithms are bitstrings.

Definition 7.25: Syntax of MAC schemes

A **message authentication code** scheme MA is a triple of algorithms $(\mathsf{KG}, \mathsf{Tag}, \mathsf{Vf})$.

- The randomized **key generation algorithm** KG takes no input and returns the secret key K as the output.
- The possibly randomized and/or stateful **tagging algorithm** Tag takes a secret key K and a message M as inputs and returns either the tag T or the special symbol $\perp$ indicating a failure as the output.
- The deterministic and stateless **verification algorithm** Vf takes a secret key K, a message M, and a tag T as inputs and returns 1 if the verification has succeeded. It returns 0 otherwise.

The correctness condition requires that, for any key $K \in [\mathsf{KG}]$, any message M, it must be the case that

$$\Pr\left[T = \perp \vee \mathsf{Vf}(K, M, T) = 1 \ : \ T \xleftarrow{\$} \mathsf{Tag}(K, M) \right] = 1 \ .$$

Figure 7.1: Another way to think about hash functions. A document is hashed to obtain its digest as a bit string using a hash function depicted here as a funnel.

The correctness condition can be read as follows. No matter which secret key K ends up being used and what plaintext M gets tagged, either the tagging algorithm returns a failure or it does not. In the latter case, verifying the resulting tag against M with the same secret key K ought to yield 1 indicating success. We say that a tag T is **valid** for M under a secret key K if $\mathsf{Vf}(K,M,T) = 1$.

Similar to the case of symmetric encryption schemes, notice that the correctness condition says nothing of the security of MAC schemes. A scheme can satisfy the correctness requirement while being completely insecure.

7.2.1 MAC versus Hash

A tag generated by computing a MAC from a secret key and an input message is sometimes considered a digest of sort. In this way, a MAC is very similar to a hash. Recall that a hash function is a function that, given an input message, outputs a digest as a bit string of a fixed length. Figure 7.1 recalls the concept of a hash function.

Similarly, a tagging algorithm can be thought of as a way to compute a digest of an input message except that this time a secret key is used to specify which function is to be used to compute the short tag. Figure 7.2 illustrates this concept in contrast to Figure 7.1.

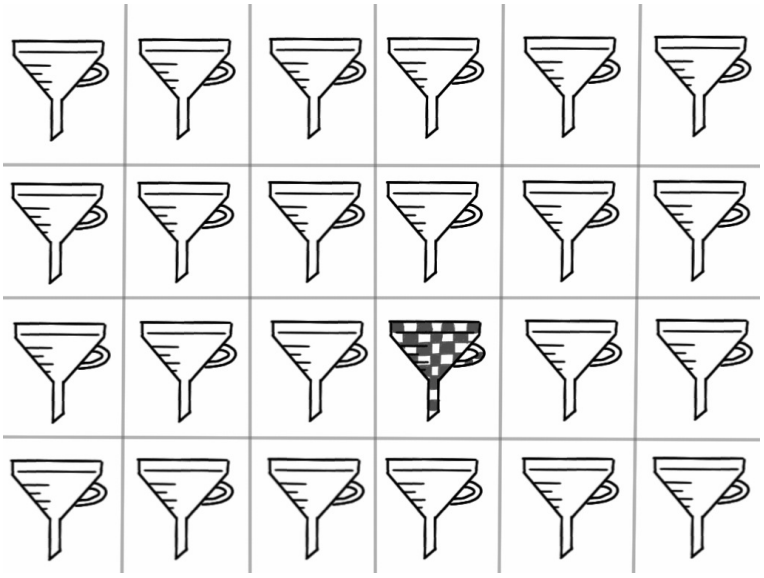


Figure 7.2: In order to compute a digest using the tagging algorithm of a message authentication scheme, one needs to supply both an input message and the secret key. In effect, the secret key specifies which function to use to compute the digest.

7.3 What security means: unforgeability

We define a security notion for MAC schemes in this section. Consider an adversary A, also known as a **forger**. The power we give A is the ability to obtain a tag for any message of its choice (under a particular secret key K). The adversary's goal is to forge a valid tag for any message that is not among the ones whose tags it has previously asked for. The following definition states this more precisely.

Definition 7.26: Security notion for MAC schemes: Weak unforgeability against Chosen-Message Attacks

Let $\mathsf{MA} = (\mathsf{KG}, \mathsf{Tag}, \mathsf{Vf})$ be a MAC scheme, and let A be an adversary with access to two oracles. We define the following subroutines, experiment, and advantage function.

Subroutines

Subroutine `Initialize`

$K \xleftarrow{\$} \mathsf{KG}$; $S \leftarrow \emptyset$; win $\leftarrow$ false

Subroutine $\texttt{Tag}(M)$

$S \leftarrow S \cup \{M\}$; Return $\mathsf{Tag}(K, M)$

Subroutine $\texttt{Vf}(M, T)$

$v \leftarrow \mathsf{Vf}(M, T)$

If $v = 1$ and $M \notin S$ then win $\leftarrow$ true

Return v

Subroutine `Finalize`

Return win

Experiment

Experiment $\mathbf{Exp}^{\text{wuf-cma}}_{\mathsf{MA}}(A)$

`Initialize`

$A^{\texttt{Tag},\texttt{Vf}}$

Return `Finalize`

We define the **wuf-cma advantage** of an adversary A mounting a chosen-message attack against MA as

$$\mathbf{Adv}^{\text{wuf-cma}}_{\mathsf{MA}}(A) = \Pr[\,\mathbf{Exp}^{\text{wuf-cma}}_{\mathsf{MA}}(A) \Rightarrow \text{true}\,] \,. \tag{7.1}$$

Definition 7.27: Security notion for MAC schemes: Strong unforgeability against Chosen-Message Attacks

Let $\mathsf{MA} = (\mathsf{KG}, \mathsf{Tag}, \mathsf{Vf})$ be a MAC scheme, and let A be an adversary with access to two oracles. We define the following subroutines, experiment, and advantage function.

Subroutines

Subroutine `Initialize`

$K \xleftarrow{\$} \mathsf{KG}$; $S \leftarrow \emptyset$; $\mathsf{win} \leftarrow \mathsf{false}$

Subroutine $\mathtt{Tag}(M)$

$T \xleftarrow{\$} \mathsf{Tag}(K, M)$; $S \leftarrow S \cup \{(M, T)\}$; Return $\mathsf{Tag}(K, M)$

Subroutine $\mathtt{Vf}(M, T)$

$v \leftarrow \mathsf{Vf}(M, T)$

If $v = 1$ and $(M, T) \notin S$ then $\mathsf{win} \leftarrow \mathsf{true}$

Return v

Subroutine `Finalize`

Return win

Experiment

Experiment $\mathbf{Exp}^{\text{suf-cma}}_{\mathsf{MA}}(A)$

`Initialize`

$A^{\mathtt{Tag},\mathtt{Vf}}$

Return `Finalize`

We define the **suf-cma advantage** of an adversary A mounting a chosen-message

Notion	The adversary's power	The adversary's goal
WUF-CMA	Submit messages to the tagging oracle and obtains the corresponding tags.	Forge a valid tag on a message that it has never submitted to the tagging oracle before.
SUF-CMA	Submit messages to the tagging oracle and obtains the corresponding tags.	Forge a valid tag on a message M so that the tag has never been returned by the tagging oracle as a tag for M before.

Table 7.1: A summary of the power and goal of any adversary mounting chosen-message attacks against a MAC scheme.

attack against MA as

$$\mathbf{Adv}^{\text{suf-cma}}_{\mathsf{MA}}(A) = \Pr[\mathbf{Exp}^{\text{suf-cma}}_{\mathsf{MA}}(A) \Rightarrow \mathsf{true}] \,. \tag{7.2}$$

The security notions defined here are called **unforgeability against chosen-message attacks**. There are two flavors. For both flavors, an adversary is allowed to ask to see tags of messages of its choices by submitting them to a *tagging oracle* one by one. In order to break weak unforgeability (WUF-CMA), the adversary needs to generate a valid tag for a message that it has never submitted to the tagging oracle. The message-tag pair output by the adversary to beat the game is often called a *forgery*. In order to break strong unforgeability (SUF-CMA), the adversary only needs to generate a new valid message-tag pair. Specifically, the message part of the forgery is allowed to be one of the ones previously submitted to the tagging oracle if the tag part is not what the tagging oracle returns as a result. Table 7.1 summarizes the power and the goal of any adversary mounting chosen-message attacks against the unforgeability of any MAC scheme.

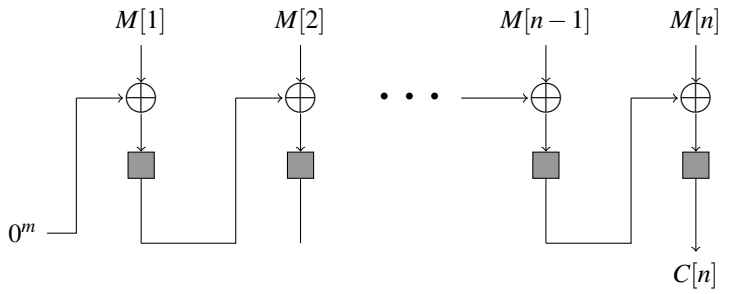


Figure 7.3: The tagging algorithm for CBC MAC. The gray boxes denote the permutation E_K where E is the underlying block cipher and K is the shared secret key. It is assumed here that the message length is a *fixed* multiple of the block size.

7.4 Constructions

7.4.1 CBC MAC

One common way to construct a MAC scheme is to use a block cipher as a building block. The most well-known MAC scheme constructed in this way is the CBC MAC. Note that, since the inverse operation on the block cipher is not relevant in this case, the following construction only assumes that the building block is a family of functions rather than a family of permutations.

Construction 7.28: CBC MAC

Let k, m, and n be positive integers, and let $E : \{0,1\}^k \times \{0,1\}^m \to \{0,1\}^m$ be a family of functions. The MAC scheme $\mathsf{CBC\text{-}MAC} = (\mathsf{KG}, \mathsf{Tag}, \mathsf{Vf})$ consists of the following algorithms. The key generation algorithm KG simply returns a bitstring chosen at random from $\{0,1\}^k$. The tagging and verification algorithms are as follows:

Algorithm $\mathsf{Tag}(K, M)$

If $(|M| = 0)$ or $(|M| \bmod m \neq 0)$ then return $\perp$

Parse M into m-bit blocks $M[1], \ldots, M[n]$

$C[0] \leftarrow IV \leftarrow 0^m$

For $i = 1$ to n do

$C[i] \leftarrow E(K, C[i-1] \oplus M[i])$

Return $C[n]$

Algorithm $\mathsf{Vf}(K, M, T)$

$T' \leftarrow \mathsf{Tag}(K, M)$

If $T' = \perp$ or $T' \neq T$ then return 0

Return 1

In short, the CBC MAC tagging algorithm is very similar to the CBC encryption algorithm except that

1. the number of message blocks to be tagged is fixed as a parameter of the scheme;
2. the initial vector is a fixed bitstring; and
3. only the last block of what would be the ciphertext in the encryption algorithm is output as the tag.

The verification algorithm simply recomputes the tag, checks whether the recomputed tag equals the input tag, and accept the message-tag pair as being authentic only if so. This strategy works here because the tagging algorithm is deterministic.

The scheme is illustrated in Figure 7.3. It was proven secure in [9]. However, as pointed out earlier, the scheme requires that the length of the messages be a fixed number of blocks. The variant of CBC MAC that allows the message length to vary is known as **SMAC** and is described in the ISO/IEC standard for CBC-MACs (ISO/IEC 9797-1: 1999 [20]).

The attack against SMAC is well known. One way researchers have tried to prevent this attack is to process the output from the last block cipher application further rather than simply outputting it as a tag. Different approaches to doing so give rise to different schemes. We describe some of them here.

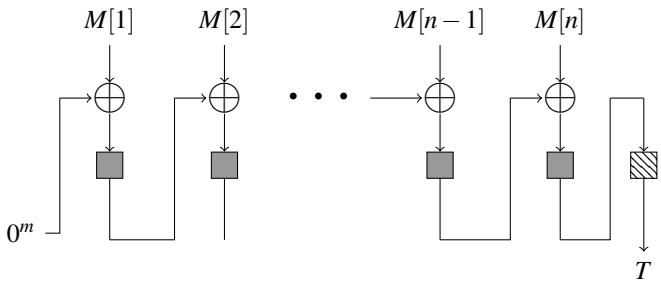


Figure 7.4: The tagging algorithm for EMAC. The gray boxes denote the permutation E_{K_1} where E is the underlying block cipher. The black box denotes the permutation E_{K_2}. The string $K_1 \| K_2$ is the shared secret key.

7.4.2 EMAC

The idea behind the Encrypted Message Authentication Code (EMAC) is, as the name suggests, to "encrypt" the output of the last application of the block cipher with one additional block cipher application, albeit using a different key. The scheme is described in the ISO/IEC standard for CBC-MACs [20]. Figure 7.4 illustrates the structure of the tagging algorithm. We describe the scheme in detail here.

Construction 7.29: EMAC

Let k and m be positive integers, and let $E : \{0,1\}^k \times \{0,1\}^m \to \{0,1\}^m$ be a family of functions. The MAC scheme $\mathsf{EMAC} = (\mathsf{KG}, \mathsf{Tag}, \mathsf{Vf})$ consists of the following algorithms. The key generation algorithm KG simply returns a bit-string chosen at random from $\{0,1\}^{2k}$. The tagging and verification algorithms are as follows:

Algorithm $\mathsf{Tag}(K, M)$

Cut K in half to obtain K_1 and K_2

Parse M into m-bit blocks $M[1], \ldots, M[n]$

where the last block can contain fewer than or equal to m bits

If $|M[n]| < m$ then pad $M[n]$ so that $|M[n]| = m$

$C[0] \leftarrow IV \leftarrow 0^m$

For $i = 1$ to n do

$C[i] \leftarrow E(K_1, C[i-1] \oplus M[i])$

$T \leftarrow E(K_2, C[n])$

Return T

Algorithm $\mathsf{Vf}(K, M, T)$

$T' \leftarrow \mathsf{Tag}(K, M)$

If $T' = \perp$ or $T' \neq T$ then return 0

Return 1

There are some choices to be made in regards to how to pad the last block $M[n]$ out to m bits. For this purpose, there are two commonly used padding schemes.

1. Append a single one followed by as many zeros as necessary so that the last block becomes m bits long.
2. Prepend to $M[n]$ an entire block of the binary representation of the length of $M[n]$. Then append to $M[n]$ as many zeros as necessary so that the last block becomes m bits long.

EMAC comes equipped with provable security results [34]. If the block size is sufficiently large, then EMAC is considered secure. If the block size is too small, then a simple meet-in-the-middle key recovery attack can be used to attack the scheme.

7.4.3 ARMAC

The idea behind the ANSI Retail MAC (ARMAC) is to "triple-encrypt" the intermediate value from the processing of the last message block before outputting the result as a tag. The scheme is described in the ISO/IEC standard for CBC-MACs [20]. We describe it in detail here. The padding scheme can be either of the two described in Section 7.4.2.

Construction 7.30: ARMAC

Let k and m be positive integers, and let $E : \{0,1\}^k \times \{0,1\}^m \to \{0,1\}^m$ be a family of functions. The MAC scheme $\mathsf{ARMAC} = (\mathsf{KG}, \mathsf{Tag}, \mathsf{Vf})$ consists of the following algorithms. The key generation algorithm KG simply returns a bit-string chosen at random from $\{0,1\}^{2k}$. The tagging and verification algorithms are as follows:

Algorithm $\mathsf{Tag}(K, M)$

Cut K in half to obtain K_1 and K_2

Parse M into m-bit blocks $M[1], \ldots, M[n]$

where the last block can contain fewer than or equal to m bits

If $|M[n]| < m$ then pad $M[n]$ so that $|M[n]| = m$

$C[0] \leftarrow IV \leftarrow 0^m$

For $i = 1$ to n do

$C[i] \leftarrow E(K_1, C[i-1] \oplus M[i])$

$T \leftarrow E(K_1, E^{-1}(K_2, C[n]))$

Return T

Algorithm $\mathsf{Vf}(K, M, T)$

$T' \leftarrow \mathsf{Tag}(K, M)$

If $T' = \perp$ or $T' \neq T$ then return 0

Return 1

ARMAC was designed with DES in mind as the underlying block cipher. It is considered inefficient because of the two additional block cipher applications at the end for a tag computation of each message. Furthermore, an implementation of ARMAC needs both the block cipher and the inverse block cipher. This is considered undesirable as it could increase the hardware cost or additional code in software, which could in turn increase the number of programming errors and the amount of code to be developed and tested. Since the advent of AES, which has a larger block size than

DES, attention has shifted to other more efficient schemes.

7.4.4 RMAC

The idea behind the Randomized MAC (RMAC) is similar to EMAC. Specifically, the output of the last application of the block cipher is "encrypted" with one additional block cipher application, albeit using a different key exclusive-ored with a random string that changes for every message. The scheme is proposed to NIST for consideration as part of the standardization process to find secure message authentication codes after AES has been standardized. We describe the scheme in detail here. The padding scheme is similar to the first one described in Section 7.4.2 except that extra care is taken when the message is already of length multiple of the block length of the underlying block cipher. See the article "RMAC: A randomized MAC beyond the birthday paradox limit" by Éliane Jaulmes, Antoine Joux and Frédéric Valette [24] for details. The description below uses a padding scheme that is the simpler one of the two proposed in [24].

Construction 7.31: RMAC

Let k and m be positive integers, and let $E : \{0,1\}^k \times \{0,1\}^m \to \{0,1\}^m$ be a family of functions. The MAC scheme $\mathsf{RMAC} = (\mathsf{KG}, \mathsf{Tag}, \mathsf{Vf})$ consists of the following algorithms. The key generation algorithm KG simply returns a bit-string chosen at random from $\{0,1\}^{2k}$. The tagging and verification algorithms are as follows:

Algorithm $\mathsf{Tag}(K, M)$

Cut K in half to obtain K_1 and K_2

Parse M into m-bit blocks $M[1], \ldots, M[n]$

where the last block can contain fewer than or equal to m bits

If $|M[n]| < m$ then $n' \leftarrow n$; pad $M[n]$ so that $|M[n]| = m$

Else $n' \leftarrow n+1$; $M[n+1] \leftarrow 10^{m-1}$

$R \xleftarrow{\$} \{0,1\}^k$

$C[0] \leftarrow IV \leftarrow 0^m$

For $i = 1$ to n' do

$C[i] \leftarrow E(K_1, C[i-1] \oplus M[i])$

$T \leftarrow R \| E(K_2 \oplus R, C[n'])$

Return T

Algorithm $\mathsf{Vf}(K, M, T)$

$T' \leftarrow \mathsf{Tag}(K, M)$

If $T' = \perp$ or $T' \neq T$ then return 0

Return 1

RMAC was included in NIST special publication 800-38B in 2002, but many attacks against it were published soon after. One very simple attack employs the fact that tagging a particular message twices allows the second part of the key, namely K_2, to be exhaustively searched in 2^k computation of the inverse block cipher operations. We describe this attack in more detail in Example 7.5.2. See Knudsen and Kohno, FSE 2003 [32] for details on other attacks against RMAC.

7.4.5 XCBC

XCBC uses a slightly different approach in dealing with the variable-length message issue than that used by other MAC schemes we have discussed so far. The idea is to "whiten" the output of the last application of the block cipher by exclusive-oring it with a random string before sending the result to one additional block cipher application. There are two choices for the random string used in the whitening step. Which one is used depends on whether the last block of the message has been padded. These two strings are chosen at random and are treated as being a part of the secret key (that is, they are fixed over the lifetime of the secret key used for the block cipher). We describe the scheme in detail here.

Construction 7.32: XCBC

Let k, m, and s be positive integers, and let $E : \{0,1\}^k \times \{0,1\}^m \to \{0,1\}^s$ be a family of functions. The MAC scheme $\mathsf{XCBC} = (\mathsf{KG}, \mathsf{Tag}, \mathsf{Vf})$ consists of the following algorithms. The key generation algorithm KG simply returns a bitstring chosen at random from $\{0,1\}^{k+2m}$. The tagging and verification algorithms are as follows:

Algorithm $\mathsf{Tag}(K, M)$

Parse K as one k-bit block and two m-bit blocks $K_1 \| K_2 \| K_3$, respectively

Parse M into m-bit blocks $M[1], \ldots, M[n]$

where the last block can contain fewer than or equal to m bits

If $|M[n]| = m$

then $B \leftarrow M[n]$

else $B \leftarrow M[n] \| 10 \cdots 0$ so that $|B| = m$

$C[0] \leftarrow IV \leftarrow 0^m$

For $i = 1$ to $n-1$ do

$C[i] \leftarrow E(K_1, C[i-1] \oplus M[i])$

If $|M[n]| = m$

then $C[n] \leftarrow E(K_1, C[n-1] \oplus B \oplus K_2)$

else $C[n] \leftarrow E(K_1, C[n-1] \oplus B \oplus K_3)$

Return $C[n]$

Algorithm $\mathsf{Vf}(K, M, T)$

$T' \leftarrow \mathsf{Tag}(K, M)$

If $T' = \perp$ or $T' \neq T$ then return 0

Return 1

The scheme is illustrated in Figure 7.5. It is very similar to CBC MAC except the handling of the last block. If the last block is shorter than the block size, then it is padded by $10 \cdots 0$ until it is. Once the last block has been massaged so that its length

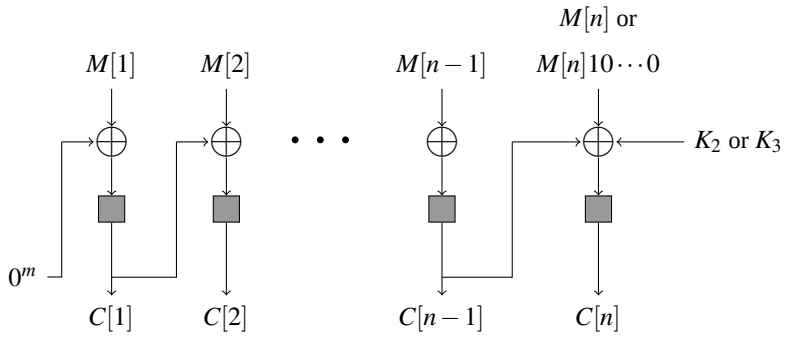


Figure 7.5: The tagging algorithm for XCBC MAC. The gray boxes denote the permutation E_K where E is the underlying block cipher and K is the shared secret key. If the last block $M[n]$ is too short, it is padded by $10\cdots0$ until its length equals the block size. If the last block has not been padded, then K_2 is additionally exclusive-ored with the message. Otherwise, K_3 is used instead.

equals the block size, the result is exclusive-ored with an additional key. There are two choices for the key to be used here, one for the case where the last block did not need padding and another in case that it did. These key bits are chosen independently of each other as denoted by setting the entire key to be of the length $k+2m$ and picking off three different portions to use for three different purposes. This scheme was proven secure in [16].

7.4.6 HMAC

The most well-known MAC scheme that uses a hash function as a building block is HMAC (Hash-based Message Authentication Codes) [7]. It requires that the underlying hash function is one constructed using the Merkle-Damgard construction (e.g. SHA1). The HMAC scheme is defined as shown in Construction 7.33 below.

Construction 7.33:

Let H be the underlying hash function, and suppose that, in the construction of H, the block length is b. The key generation algorithm KG simply returns a bitstring chosen at random from $\{0,1\}^b$. The tagging and verification algorithms are as

follows:

Algorithm $\mathsf{Tag}(K, M)$

$\mathsf{opad} \leftarrow$ x36 repeated to get b bits total

$\mathsf{ipad} \leftarrow$ x5c repeated to get b bits total

Return $H(K \oplus \mathsf{opad} \| H(K \oplus \mathsf{ipad}))$

Algorithm $\mathsf{Vf}(K, M, T)$

$T' \leftarrow \mathsf{Tag}(K, M)$

If $T' = \perp$ or $T' \neq T$ then return 0

Return 1

Bellare et al. proved in [7] that HMAC is secure assuming that the underlying compression function is a PRF and the hash function is weakly collision-resistant. This result is later improved upon by Bellare in [6]. Specifically, the second assumption is removed, and thus, we have that HMAC is a secure MAC scheme assuming that the underlying compression function is a PRF.

7.5 Attacks against MAC schemes

In this section, we show some example attacks against some simple MAC schemes.

We begin with an attack against the scheme in Construction 7.28, namely the CBC MAC scheme. Although CBC MAC is secure assuming that the underlying block cipher is secure, this security is contingent upon the restriction that the input messages can only be of a particular fixed length. Here we show an attack against the scheme when the input messages are allowed to be of various lengths. The idea behind this attack is simply to first obtain a tag for some message, then to use as a forgery a different message that also has the same tag. To compute the message that would pass the verification when considered with the same tag, the attack manipulates the message so that the input value to the block cipher is repeated. Here are the details.

7.5.1 Example: CBC MAC for varying message lengths

An adversary A can attack CBC-MAC defined in Construction 7.28 as follows:

Adversary A

$T \xleftarrow{\$} \mathtt{Tag}(0^m)$

$\mathtt{Vf}(0^m \| T, T)$

Let K be the secret key unknown to the adversary A. To see that this attack works, observe that, since T is the tag of the message 0^m, we know that

$$T = E_K(0^m \oplus 0^m) .$$

Now, consider the message $0^m \| T$. The tag is

$$E_K(E_K(0^m) \oplus T) = E_K(T \oplus T) = E_K(0^m) = T .$$

So the tag for the message $0^m \| T$ is also T, and thus, the pair $(0^m \| T, T)$ is a valid forgery.

The success probability of A is 1 while its resource usage is very low. Specifically, A submits only one query to the tagging oracle and one query to the verification oracle. Notice that in Example 7.5.1, it matters not whether the IV used is 0^m. One can still attack CBC MAC in a similar way.

7.5.2 Example: Exhaustive key search against RMAC

Let $\langle x \rangle$ denote the k-bit binary representation of the integer x. An adversary A can attack RMAC defined in Construction 7.31 as follows:

Adversary A

$M \leftarrow 0^{m-1}$

$R_1 \| T_1 \xleftarrow{\$} \mathtt{Tag}(M)$; $R_2 \| T_2 \xleftarrow{\$} \mathtt{Tag}(M)$

For $i = 1$ to 2^k do

$K \leftarrow \langle i \rangle$; $C_1 \leftarrow E^{-1}_{K \oplus R_1}(T_1)$; $C_2 \leftarrow E^{-1}_{K \oplus R_2}(T_2)$

If $C_1 = C_2$ then break

$K_2 \leftarrow K$

For $i = 1$ to 2^k do

$K \leftarrow \langle i \rangle$; If $E_K(0^{m-1}1) = C_1$ then break

$K_1 \leftarrow K$

$R \| T \xleftarrow{\$} \mathsf{Tag}(K_1 \| K_2, 1^m)$

$\mathtt{Vf}(1^m, R \| T)$

The first for loop exhaustively search for the second part of the secret key, namely K_2, while the second for loop exhaustively search for the first part of the secret key, namely K_1. Once the complete key is obtained, A generates the tag by itself on a new message using the tagging algorithm with the newly obtained key. It then submits the forgery to the verification oracle and rests assured that it wins.

The success probability of A is 1 while its resource usage is at most 2^k inverse block cipher computations and at most 2^k block cipher computations. It also submits only two queries to the tagging oracle and one query to the verification oracle.

7.6 Conclusion

Mention the word cryptography, and most people think about encryption and the goal of keeping secrets. This chapter explores an equally, if not more, important goal of guaranteeing authenticity of data (whether at rest or in transit). The main tool for achieving this goal in the symmetric setting is message authentication codes (MACs). The idea is to attach to the data being protected a tag that can be used to verify the authenticity of the data. In the symmetric setting, the entity who generates the tag must share a secret key with the entity who verifies it. We point out that, while this mechanism allows the verification of the authenticity of data, it does not provide a property known as *non-repudiation*, the ability to be able to hold the entity who generates the data accountable for having done so since the verifier could have just as easily generated the tag using the same shared key. For non-repudiation, one needs

digital signatures, a cryptographic primitive designed to be used in the asymmetric setting.

The security definitions relevant to message authentication codes are the notion of unforgeability. There are two slightly different definitions along this line, namely the weak (WUF-CMA) and strong (SUF-CMA) unforgeability. We explore many practical schemes that achieve these properties in this chapter.

7.7 Credits

Construction 7.28 was proven secure in [9]. The idea of considering a family hash functions when analyzing security is due to Damgard. More complete taxonomy of security definitions for hash functions is in [39]. The attack against RMAC is due to Knudsen and Kohno [32].

7.8 Exercises

Exercise 7.1. How does a MAC scheme differ from a symmetric encryption scheme in each of the following contexts?

1. syntax
2. security properties targetted
3. applications

Exercise 7.2. What property does the correctness condition for MAC schemes capture? Explain in your own words.

Exercise 7.3. Let k be a non-negative integer. Let H: $\{0,1\}^* \to \{0,1\}^k$ be a public hash function secure under the collision resistance definition (Definition 6.24). Consider a message authentication code $\mathsf{MA} = (\mathsf{KG}, \mathsf{Tag}, \mathsf{Vf})$ defined as follows:

Algorithm KG	Algorithm $\mathsf{Tag}_K(M)$	Algorithm $\mathsf{Vf}_K(M,T)$
$K \xleftarrow{\$} \{0,1\}^k$	Return $\mathsf{H}(M) \oplus K$	If $\mathsf{H}(M) \oplus K = T$
Return K		then return 1 else return 0

1. Is MA a secure MAC scheme under the weak unforgeability against chosen-message attacks (WUF-CMA defined in Definition 7.26)?
2. Prove your answer. If your answer is yes, you need to show a reduction and derive the relationship between the insecurity of H and that of MA obtained from the reduction. If your answer is no, you need to write the pseudocode for an adversary, analyze its advantage, and specify its resource usage. The less the resource usage and the higher the advantage, the better.

Exercise 7.4. Let k, l be non-negative integers. Let $E\colon \{0,1\}^k \times \{0,1\}^l \to \{0,1\}^l$ be a family of permutations. Let $\mathsf{MA} = (\mathsf{KG}, \mathsf{Tag}, \mathsf{Vf})$ be a MAC scheme inspired by the CTR encryption scheme with random initial counter value. We specify the definitions of the algorithms of MA in full here. The key generation algorithm simply outputs a randomly-chosen bit string of length k. The tagging and verification algorithms work as follows.

Algorithm $\mathsf{Tag}_K(M)$

If $|M| \bmod l \neq 0$ then return $\perp$

Parse M as l-bit blocks $M[1] \ldots M[n]$

static $ctr \xleftarrow{\$} \{0,1\}^l$

$C[0] \leftarrow \langle ctr \rangle$

For $i = 1$ to n do $C[i] \leftarrow E_K(\langle [\![C[0]]\!] + i \rangle) \oplus M[i]$

$T \leftarrow C[1] \oplus \ldots \oplus C[n]$

$ctr \leftarrow ctr + n$

Return $(M, C[0], T)$

Algorithm $\mathsf{Vf}_K(M, C_0, T)$

If $|M| \bmod l \neq 0$ then return $\perp$

Parse M as l-bit blocks $M[1] \ldots M[n]$

$C[0] \leftarrow C_0$

For $i = 1$ to n do $C[i] \leftarrow E_K(\langle [\![C[0]]\!] + i \rangle) \oplus M[i]$

$T' \leftarrow C[1] \oplus \ldots \oplus C[n]$

If $T' = T$ then return 1 else return 0

1. Is MA a weakly unforgeable MAC scheme as per Definition 7.26 ?
2. Prove your answer. If your answer is yes, you need to show a reduction and derive the relationship between the insecurity of E and that of MA obtained from the reduction. If your answer is no, you need to write the pseudocode for an adversary, analyze its advantage, and specify its resource usage. The less the resource usage and the higher the advantage, the better.

Exercise 7.5. Let k, n be positive integers. Consider the following fixed-length MAC for messages of length $2n-2$ using a pseudorandom family of functions $E : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$. Consider a message authentication code $\mathsf{MA} = (\mathsf{KG}, \mathsf{Tag}, \mathsf{Vf})$ defined as follows. The key generation algorithm KG simply outputs a randomly-chosen bit string of length k. The tagging algorithm works as follows.

Algorithm $\mathsf{Tag}_K(M)$

$M_0 \| M_1 \leftarrow M$ where $|M_0| = |M_1| = n-1$

Return $E_K(0\|M_0) \oplus E_K(1\|M_1)$

Do the following problems.

1. Specify in full detail how the verification algorithm works.
2. Is this MAC scheme weakly unforgeable as per Definition 7.26 ?
3. Prove your answer. If your answer is yes, you need to show a reduction and derive the relationship between the insecurity of E and that of MA obtained from the reduction. If your answer is no, you need to write the pseudocode for an adversary, analyze its advantage, and specify its resource usage. The less the resource usage and the higher the advantage, the better.
4. Is this MAC scheme strongly unforgeable as per Definition 7.27 ?
5. Prove your answer. If your answer is yes, you need to show a reduction and derive the relationship between the insecurity of E and that of MA obtained

from the reduction. If your answer is no, you need to write the pseudocode for an adversary, analyze its advantage, and specify its resource usage. The less the resource usage and the higher the advantage, the better.

6. Are your answers for the weak unforgeability and strong unforgeability security any different?

Exercise 7.6. Let k, n be positive integers. Consider the following fixed-length MAC for messages of length $2n-2$ using a pseudorandom family of functions $E : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$. Consider a message authentication code $\mathsf{MA} = (\mathsf{KG}, \mathsf{Tag}, \mathsf{Vf})$ defined as follows. The key generation algorithm KG simply outputs a randomly-chosen bit string of length k. The tagging algorithm works as follows.

Algorithm $\mathsf{Tag}_K(M)$

$M_0 \| M_1 \leftarrow M$ where $|M_0| = |M_1| = n$

Return $E_K(M_0) \oplus E_K(E_K(M_1))$

Do the following problems.

1. Specify in full detail how the verification algorithm works.
2. Is this MAC scheme weakly unforgeable as per Definition 7.26 ?
3. Prove your answer. If your answer is yes, you need to show a reduction and derive the relationship between the insecurity of E and that of MA obtained from the reduction. If your answer is no, you need to write the pseudocode for an adversary, analyze its advantage, and specify its resource usage. The less the resource usage and the higher the advantage, the better.
4. Is this MAC scheme strongly unforgeable as per Definition 7.27 ?
5. Prove your answer. If your answer is yes, you need to show a reduction and derive the relationship between the insecurity of E and that of MA obtained from the reduction. If your answer is no, you need to write the pseudocode for an adversary, analyze its advantage, and specify its resource usage. The less the resource usage and the higher the advantage, the better.

6. Are your answers for the weak unforgeability and strong unforgeability security any different?

8

Authenticated Encryption

8.1 Motivation

Encryption has no doubt long been the most prominent goal in communication security. However, in the cryptography and security communities alike, people have come to realized that authenticity is equally important. In fact, there are many examples in real world usage of cryptography that prove the point that "encryption without integrity checking is all but useless [14]."

In this chapter, we investigate a cryptographic construct whose aim is to simultaneously offer *both* privacy *and* authenticity properties. It is called an *authenticated encryption scheme*. The chapter begins with the syntax and correctness condition of authenticated encryption schemes. Then, security definitions pertinent to authenticated encrytion schemes are explored and presented. Finally, the most common methodology in designing authenticated encryption schemes, namely the *generic composition paradigm*, is studied.

8.2 What it is: syntax and correctness condition

We use the term *authenticated encryption (AE) scheme* to refer to a symmetric encryption scheme whose goal is to simultaneously provide *both* privacy *and* authenticity. Figure 8.1 depicts a possible application of authenticated encryption.

The syntax of an AE scheme is exactly the same as that of a symmetric encryption scheme. Specifically, an AE scheme is made up of three algorithms: key generation,

Figure 8.1: A knight delivers an important message. The message is kept secret (as conveyed by the use of an opaque envelope) and authenticated (as conveyed by the use of a special stamp of approval).

encryption, and decryption. The key generation algorithm takes no input and returns a secret key. The encryption algorithm takes a secret key and the message to be encrypted and returns either a ciphertext or a distinguished symbol $\perp$ to indicate that it refuses to encrypt the message. The decryption algorithm takes a secret key and a ciphertext and returns either the decryption of the latter or $\perp$ to indicate that the input ciphertext is rejected. The correctness condition for AE schemes is also the same as that of symmetric encryption schemes: a ciphertext either decrypts to $\perp$ or the original message when the decryption and encryption operations are performed using the same secret key. The precise definition for AE schemes is exactly the same as that for symmetric encryption schemes as per Definition 5.10 in Chapter 5.

8.3 What security means: privacy and authenticity

The privacy security definitions for AE schemes are indistinguishability against chosen-plaintext attacks (IND-CPA) and indistinguishability against chosen-ciphertext attacks (IND-CCA) and are defined in exactly the same manner as for symmetric encryption schemes as per Definitions 5.11 and 5.12. The precise definitions for authenticity are as specified in Definitions 8.34 and 8.35 here.

The general idea is to let the adversary see the encryption of a number of messages of its choice. The adversary is also allowed to check whether any ciphertext

of its choice decrypts to a valid message. The adversary's goal is to forge a "valid" ciphertext where term "valid" here means that the message is not $\perp$.

In practical terms, similar to the rationale behind chosen-plaintext attacks, the adversary's ability to see the encryption of messages captures the fact that in practice it typically is not hard to fool someone to encrypt messages and snoop the resulting ciphertexts. Here we additionally give the adversary an ability to test whether a ciphertext is valid. This ability may be realized in practice by observing the behavior of the decryptor. For example, a server may tear down a connection to conserve its resources once it obtains a ciphertext and finds that the latter is invalid, and it may be clear to an outside observer that this has happened.

In order to violate the integrity of plaintexts, the adversary needs to find a valid ciphertext that decrypts to a "new" message where the term "new" means that the message has never been queried to the encryption oracle before. The game defined as part of Definition 8.34 spells all this out in detail.

Definition 8.34: Authenticity notion for authenticated encryption schemes: Integrity of Plaintext

Let $\mathsf{AE} = (\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec})$ be an authenticated encryption scheme, and let A be an adversary with access to an oracle. We define the following subroutines, experiment, and advantage function.

Subroutines

Subroutine `Initialize`

$K \xleftarrow{\$} \mathsf{KG}$; $S \leftarrow \emptyset$; $\mathsf{win} \leftarrow \mathsf{false}$

Subroutine $\texttt{Enc}(M)$

$S \leftarrow S \cup \{M\}$

Return $\mathsf{Enc}(K, M)$

Subroutine $\texttt{Vf}(C)$

$M \leftarrow \mathsf{Dec}(K, C)$

If $M \neq \perp$ and $M \notin S$ then $\mathsf{win} = \mathsf{true}$

If $M = \perp$ then return 0 else return 1

Subroutine `Finalize`

Return win

Experiment

Experiment $\mathbf{Exp}^{\text{int-ptxt}}_{\mathsf{AE}}(A)$

`Initialize`

$d \xleftarrow{\$} A^{\texttt{Enc},\texttt{Vf}}$

Return `Finalize`

We define the **int-ptxt advantage** of an adversary A mounting an attack against the integrity of plaintexts of AE as

$$\mathbf{Adv}^{\text{int-ptxt}}_{\mathsf{AE}}(A) = \Pr[\,\mathbf{Exp}^{\text{int-ptxt}}_{\mathsf{AE}}(A) \Rightarrow \mathsf{true}\,] \ . \tag{8.1}$$

In contrast to integrity of plaintexts, an adversary whose goal is to violate the integrity of ciphertexts only needs to find a valid ciphertext that is itself new. Similar to the

case of integrity of plaintexts, a "new" ciphertext is one that the adversary has never received as a result of any previous query to the encryption oracle. The game defined as part of Definition 8.35 spells all this out in detail.

Definition 8.35: Authenticity notion for authenticated encryption schemes: Integrity of Ciphertext

Let $\mathsf{AE} = (\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec})$ be an authenticated encryption scheme, and let A be an adversary with access to an oracle. We define the following subroutines, experiment, and advantage function.

Subroutines

Subroutine `Initialize`

$K \xleftarrow{\$} \mathsf{KG}$; $S \leftarrow \emptyset$; $\mathsf{win} \leftarrow \mathsf{false}$

Subroutine $\texttt{Enc}(M)$

$C \xleftarrow{\$} \mathsf{Enc}(K, M)$; $S \leftarrow S \cup \{C\}$

Return C

Subroutine $\texttt{Vf}(C)$

$M \leftarrow \mathsf{Dec}(K, C)$

If $M \neq \perp$ and $C \notin S$ then $\mathsf{win} = \mathsf{true}$

If $M = \perp$ then return 0 else return 1

Subroutine `Finalize`

Return win

Experiment

Experiment $\mathbf{Exp}^{\text{int-ctxt}}_{\mathsf{AE}}(A)$

`Initialize`

$d \xleftarrow{\$} A^{\mathtt{Enc},\mathtt{Vf}}$

Return `Finalize`

We define the **int-ctxt advantage** of an adversary A mounting an attack against integrity of ciphertexts of AE as

$$\mathbf{Adv}^{\text{int-ctxt}}_{\mathsf{AE}}(A) = \Pr[\,\mathbf{Exp}^{\text{int-ctxt}}_{\mathsf{AE}}(A) \Rightarrow \mathsf{true}\,]\,. \tag{8.2}$$

8.4 Constructions based on the generic composition paradigm

As we have studied in Chapters 5 and 7) on symmetric encryption and MAC schemes, respectively, many schemes have been devised to provide privacy and authenticity separately. One common approach to design an authenticated encryption scheme in order to achieve both privacy and authenticity simultaneously is hence to combine symmetric encryption and MAC schemes, treating each construct as a black box. The process of generically combining a symmetric encryption scheme and a MAC scheme to obtain an authenticated encryption scheme can be likened to combining a car and a boat without taking either of them apart to obtain a amphibious vehicle. Figure 8.2 illustrates this concept.

At first glance, putting the two together should be a simple matter, and one would hope that the resulting scheme would "just work." However, as Figures 8.3 and 8.4 illustrate, some combination does not work while others may look more promising.

The problem of analyzing security properties that result from the three most obvious ways to combine a symmetric encryption scheme and a MAC scheme is known

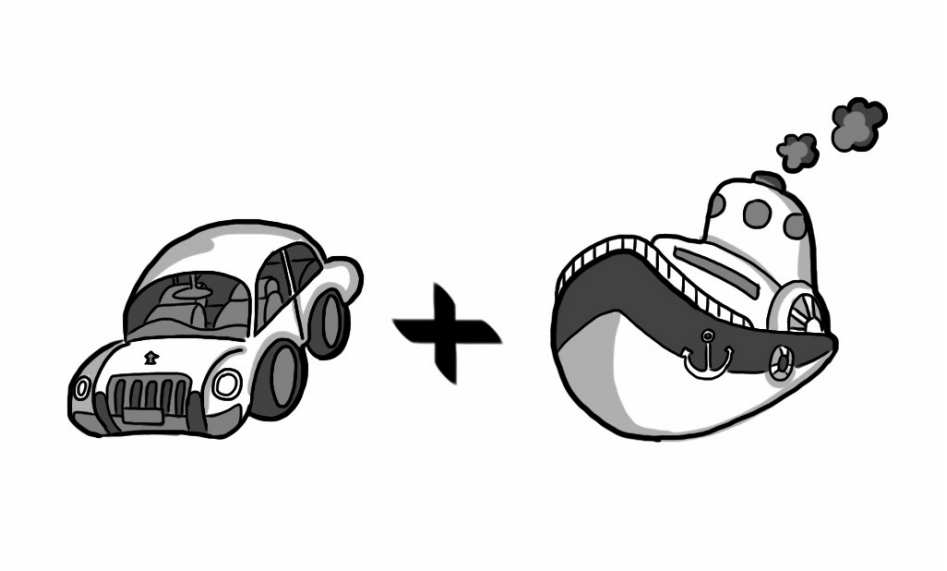


Figure 8.2: If a symmetric encryption was a car and a MAC scheme was a boat, we can just put the two together in a "black-box" manner and hope that we get an amphibious vehicle, can't we?

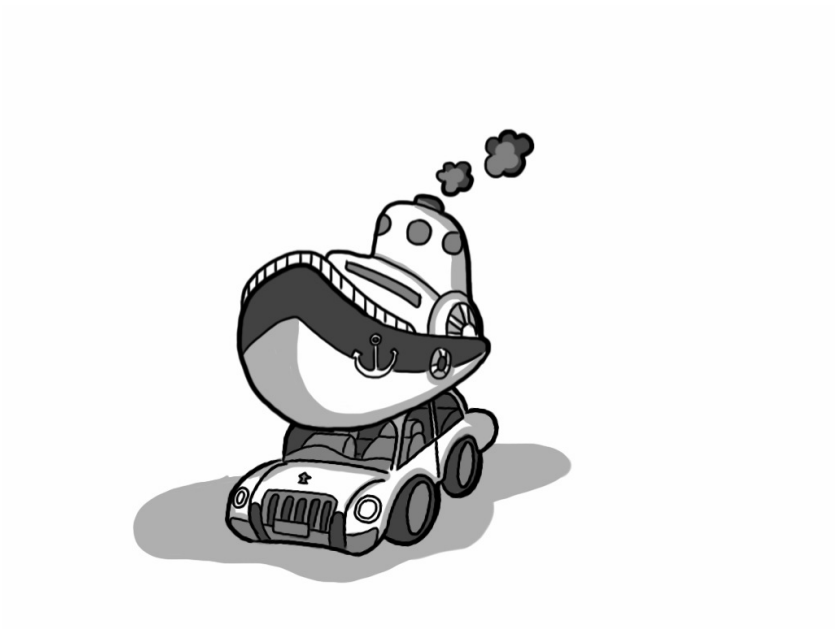


Figure 8.3: Combining a car and a boat in this manner seems unlikely to result in a vehicle that could function both on the road and in a lake.

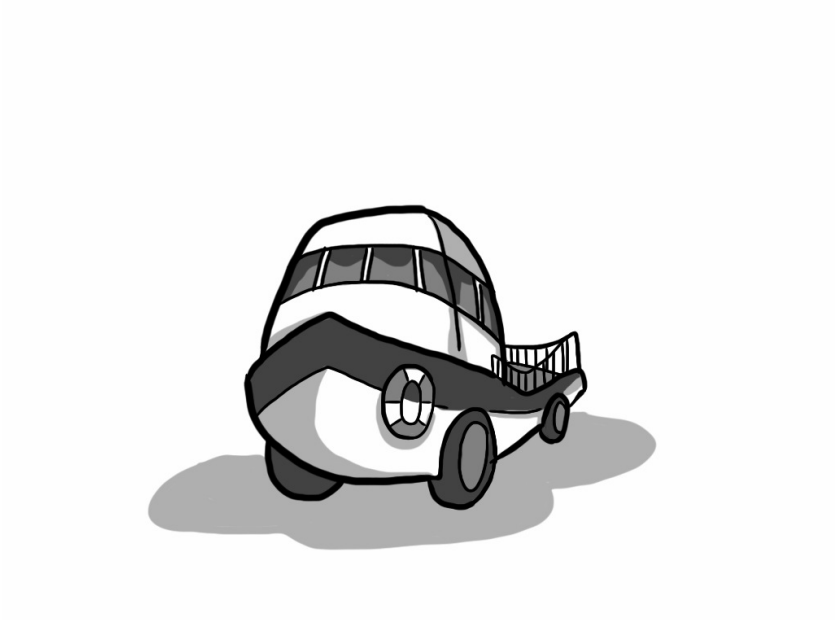


Figure 8.4: Combining a car and a boat in this manner seems promising. But would it actually work? The devil is most likely in the details.

as the order of encryption and MAC problem. The approach is known as the *generic composition paradigm*.

Let $\mathsf{SE} = (\mathsf{KG}_e, \mathsf{Enc}, \mathsf{Dec})$ be a symmetric encryption scheme, and let $\mathsf{MA} = (\mathsf{KG}_m, \mathsf{Tag}, \mathsf{Vf})$ be a MAC scheme. The order of encryption and MAC problem refers to the idea that, to combine the encryption and MAC algorithms when encrypting a message, one can perform the two operations in parallel or one before the other. (The decryption algorithm can easily be defined accordingly.) The goal is to obtain both privacy and authenticity simultaneously. This means that the target goal is to meet either INT-CTXT $\wedge$ IND-CPA or INT-PTXT $\wedge$ IND-CPA.

The three composition methods are described in more detail here. Throughout, we denote the key for encryption by K_e and the key for message authentication by K_m.

Encrypt-and-MAC: In this method, we first encrypt the plaintext then append a MAC of the plaintext. The decryption algorithm first decrypts to get the plaintext and then verifies the tag. In short,

$$\overline{\mathsf{Enc}}(\langle K_e, K_m \rangle, M) = \mathsf{Enc}(K_e, M) \| \mathsf{Tag}(K_m, M) \ .$$

MAC-then-encrypt: In this method, we compute a MAC, then append it to the plaintext, and finally encrypt them together. The decryption algorithm first decrypts to get the plaintext and candidate tag, and then verifies the tag. In short,

$$\overline{\mathsf{Enc}}(\langle K_e, K_m\rangle, M) = \mathsf{Enc}(K_e, M \| \mathsf{Tag}(K_m, M)) \ .$$

Encrypt-then-MAC: In this method, we first encrypt the plaintext to get a ciphertext C and then append a MAC of C. The decryption algorithm first verifies the tag and then decrypts C. In short,

$$\overline{\mathsf{Enc}}(\langle K_e, K_m, M) = C \| \mathsf{Tag}(K_m, C) \quad \text{where} \quad C = \mathsf{Enc}(K_e, M) \ .$$

8.4.1 Security results

We are interested in the security properties obtained through each of the generic composition method. Specifically, assuming that the underlying symmetric encryption and MAC schemes are secure in their respective senses, we ask what security properties the authenticated encryption scheme put together according to each generic composition method would give us. Table 8.1 and Table 8.2 summarize the results. The results in Table 8.1 are obtained under the assumptions that the underlying symmetric encryption scheme provides indistinguishability against chosen-plaintext attacks (IND-CPA) and that the underlying MAC scheme is weakly unforgeable (WUF-CMA). The results in Table 8.2 are obtained under the assumptions that the underlying symmetric encryption scheme provides indistinguishability against chosen-plaintext attacks (IND-CPA) and that the underlying MAC scheme is strongly unforgeable (SUF-CMA).

In both tables, the term *secure* means that, assuming only that the underlying symmetric encryption and the underlying MAC schemes are secure, then the resulting authenticated encryption scheme in question is secure. The term *insecure* means that there exist some secure symmetric encryption and some secure MAC schemes such that, when put together as dictated by the generic composition method in question, results in an authenticated encryption scheme that is insecure.

Type	Privacy		Integrity	
	IND-CPA	IND-CCA	INT-PTXT	INT-CTXT
E&M	insecure	insecure	secure	insecure
MtE	secure	insecure	secure	insecure
EtM	secure	insecure	secure	insecure

Table 8.1: Summary of results under the assumption that the underlying encryption scheme is IND-CPA and the underlying MAC scheme is weakly unforgeable.

Type	Privacy		Integrity	
	IND-CPA	IND-CCA	INT-PTXT	INT-CTXT
E&M	insecure	insecure	secure	insecure
MtE	secure	insecure	secure	insecure
EtM	secure	secure	secure	secure

Table 8.2: Summary of results under the assumption that the underlying encryption scheme is IND-CPA and the underlying MAC scheme is strongly unforgeable.

Note that the results presented here are generic. For example, they do not imply that *all* authenticated encryption schemes that employ the *encrypt-and-MAC* method are insecure under indistinguishability against chosen-plaintext attacks. In other words, there are specific authenticated encryption schemes that are secure even though they employ the *encrypt-and-MAC* method. See [10] for concrete examples.

8.4.2 Security analysis of generic composition methods

We now proceed with the security analysis for the authenticated encryption schemes obtained via generic composition methods. Throughout this section, $\mathsf{SE} = (\mathsf{KG}_e, \mathsf{Enc}, \mathsf{Dec})$ is an underlying symmetric encryption scheme which is IND-CPA secure, $\mathsf{MA} = (\mathsf{KG}_m, \mathsf{Tag}, \mathsf{Vf})$ is an underlying MAC scheme which is WUF-CMA or SUF-CMA secure, and $\overline{\mathsf{SE}} = (\overline{\mathsf{K}}, \overline{\mathsf{E}}, \overline{\mathsf{D}})$ is the resulting scheme obtained through the composition methods.

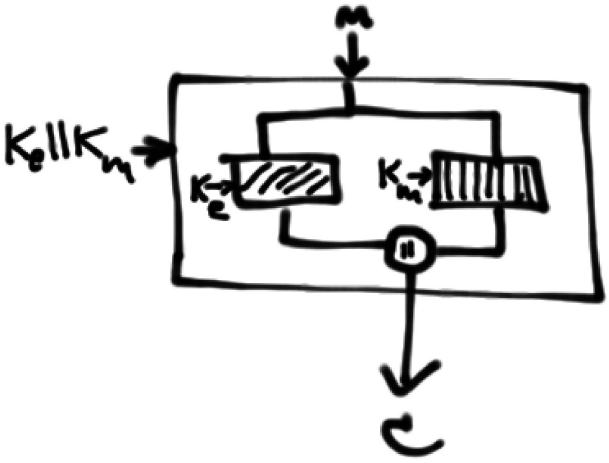


Figure 8.5: A depiction of the *Encrypt-and-MAC* composition method.

Encrypt-and-MAC

Figure 8.5 depicts the structure of the *Encrypt-and-MAC* method. Construction 8.36 describes this method in detail. Table 8.3 summarizes the security results.

Construction 8.36: Encrypt-and-MAC composition method

Let $\mathsf{SE} = (\mathsf{KG}_e, \mathsf{Enc}, \mathsf{Dec})$ be a symmetric encryption scheme, and let $\mathsf{MA} = (\mathsf{KG}_m, \mathsf{Tag}, \mathsf{Vf})$ be a message authentication scheme. We define a composite scheme $\overline{\mathsf{SE}} = (\overline{\mathsf{K}}, \overline{\mathsf{E}}, \overline{\mathsf{D}})$ as follows.

Algorithm $\overline{\mathsf{K}}$

$K_e \xleftarrow{\$} \mathsf{KG}_e$

$K_m \xleftarrow{\$} \mathsf{KG}_m$

Return $\langle K_e, K_m \rangle$

Algorithm $\overline{\mathsf{E}}(\langle K_e, K_m \rangle, M)$

$C' \leftarrow \mathsf{Enc}(K_e, M)$

$\tau \leftarrow \mathsf{Tag}(K_m, M)$

$C \leftarrow C' \| \tau$

Security notion	Assuming weak MAC	Assuming strong MAC
IND-CPA	×	×
IND-CCA	×	×
INT-PTXT	√	√
INT-CTXT	×	×

Table 8.3: Summary of security results for the *Encrypt-and-MAC* composition method.

Return C

Algorithm $\overline{\mathsf{D}}(\langle K_e, K_m\rangle, C)$

Parse C as $C' \| \tau$

$M \leftarrow \mathsf{Dec}(K_e, C')$

$v \leftarrow \mathsf{Vf}(K_m, M, \tau)$

If $v = 1$ return M else return $\perp$.

The *Encrypt-and-MAC* composition method does not preserve privacy because the MAC could reveal information about the plaintext. The following makes this precise.

Theorem 8.37:

(Encrypt-and-MAC method is not IND-CPA secure) Given a IND-CPA secure symmetric encryption scheme SE and a WUF-CMA (resp. SUF-CMA) secure message authentication scheme MA, we can construct a message authentication scheme MA' such that MA' is WUF-CMA (resp. SUF-CMA) secure, but the composite scheme $\overline{\mathsf{SE}}$ formed by the *Encrypt-and-MAC* composition method based on SE and MA' is *not* IND-CPA secure.

Since IND-CCA implies IND-CPA, this means that this composition method is also not IND-CCA secure.

More generally, since most MACs are deterministic, the *Encrypt-and-MAC* method, which exposes the tag part outright, it is easy to tell if a particular message

has been encrypted twice. Thus, the following theorem follows.

Theorem 8.38:

(Encrypt-and-MAC method is IND-CPA insecure for any deterministic MAC) Let SE be a IND-CPA secure symmetric encryption scheme, and let MA be a deterministic WUF-CMA or SUF-CMA secure message authentication scheme. Then, the composite scheme $\overline{\mathsf{SE}}$ obtained from SE and MA by the *Encrypt-and-MAC* composition method is *not* IND-CPA secure.

The *Encrypt-and-MAC* composition method does preserve integrity of plaintexts. It inherits the integrity of the MAC in a direct way, with no degradation in security. This is independent of the symmetric encryption scheme: whether the latter is secure or not does not affect the integrity of the composite scheme.

Theorem 8.39:

(Encrypt-and-MAC method is INT-PTXT secure) Let SE be a symmetric encryption scheme, let MA be a message authentication scheme, and let $\overline{\mathsf{SE}}$ be the encryption scheme obtained from SE and MA via the *Encrypt-and-MAC* composition method. Then, if MA is WUF-CMA secure, then $\overline{\mathsf{SE}}$ is INT-PTXT secure. Specifically, given any int-ptxt adversary A against $\overline{\mathsf{SE}}$, we can construct a wuf-cma forger F such that

$$\mathbf{Adv}^{\text{int-ptxt}}_{\overline{\mathsf{SE}}}(A) \leq \mathbf{Adv}^{\text{wuf-cma}}_{\mathsf{MA}}(F) \,.$$

Furthermore, F uses the same amount of resources as A.

The same is true if the MAC is SUF-CMA secure.

However, the *Encrypt-and-MAC* composition method is insecure when it comes to integrity of ciphertexts. The reason: for some secure encryption schemes, it is possible to change the ciphertext without affecting the decryption. Thus, the tag part remains valid even if the ciphertext is changed. A scheme with this property does not

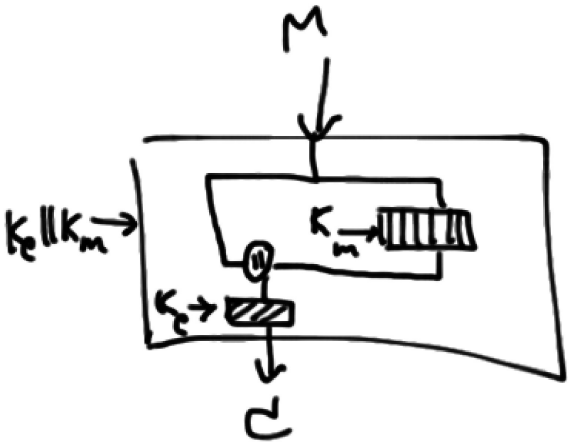


Figure 8.6: A depiction of the *MAC-then-encrypt* composition method.

provide the integrity of ciphertexts.

Theorem 8.40:

(Encrypt-and-MAC method is not INT-CTXT secure) Given a IND-CPA secure symmetric encryption scheme SE and a WUF-CMA or SUF-CMA secure message authentication scheme MA, we can construct a symmetric encryption scheme SE' such that SE' is IND-CPA secure, but the composite scheme $\overline{\mathsf{SE}}$ formed by the *Encrypt-and-MAC* composition method based on SE' and MA is *not* INT-CTXT secure.

8.4.3 MAC-then-Encrypt

Figure 8.6 depicts the structure of the *MAC-then-encrypt* method. Construction 8.41 describes this method in detail. Table 8.4 summarizes the security results.

Construction 8.41: MAC-then-encrypt composition method

Let $\mathsf{SE} = (\mathsf{KG}_e, \mathsf{Enc}, \mathsf{Dec})$ be a symmetric encryption scheme, and let $\mathsf{MA} = (\mathsf{KG}_m, \mathsf{Tag}, \mathsf{Vf})$ be a message authentication scheme. We define a composite scheme $\overline{\mathsf{SE}} = (\overline{\mathsf{K}}, \overline{\mathsf{E}}, \overline{\mathsf{D}})$ as follows.

Security notion	Assuming weak MAC	Assuming strong MAC
IND-CPA	$\surd$	$\surd$
IND-CCA	$\times$	$\times$
INT-PTXT	$\surd$	$\surd$
INT-CTXT	$\times$	$\times$

Table 8.4: Summary of security results for the *MAC-then-Encrypt* composition method.

Algorithm $\overline{\mathsf{K}}$

$K_e \xleftarrow{\$} \mathsf{KG}_e$

$K_m \xleftarrow{\$} \mathsf{KG}_m$

Return $\langle K_e, K_m \rangle$

Algorithm $\overline{\mathsf{E}}(\langle K_e, K_m \rangle, M)$

$\tau \leftarrow \mathsf{Tag}(K_m, M)$

$C \leftarrow \mathsf{Enc}(K_e, M \| \tau)$

Return C

Algorithm $\overline{\mathsf{D}}(\langle K_e, K_m \rangle, C)$

$M' \leftarrow \mathsf{Dec}(K_e, C)$

Parse M' as $M \| \tau$

$v \leftarrow \mathsf{Vf}(K_m, M, \tau)$

If $v = 1$ return M else return $\perp$

The *MAC-then-encrypt* composition method preserves both privacy against chosen-plaintext attack and integrity of plaintexts, as stated in the following theorem.

Theorem 8.42:

(MAC-then-encrypt method is both INT-PTXT secure and IND-CPA secure) Let MA be a message authentication scheme, and let SE be a symmetric encryption scheme secure against chosen-plaintext attacks. Let $\overline{\mathsf{SE}}$ be the encryption scheme obtained from SE and MA via the *MAC-then-encrypt* composi-

tion method. Then, if MA is WUF-CMA secure, then $\overline{\mathsf{SE}}$ is INT-PTXT secure. Furthermore, if SE is IND-CPA secure, then so is $\overline{\mathsf{SE}}$. Specifically, given any int-ptxt adversary I against $\overline{\mathsf{SE}}$, we can construct a wuf-cma forger F such that Equation (8.3) holds. Similarly, any ind-cpa adversary A against $\overline{\mathsf{SE}}$, we can construct an ind-cpa adversary A_p such that Equation (8.4) holds.

$$\mathbf{Adv}^{\text{int-ptxt}}_{\overline{\mathsf{SE}}}(I) \leq \mathbf{Adv}^{\text{wuf-cma}}_{\mathsf{MA}}(F) \tag{8.3}$$

$$\mathbf{Adv}^{\text{ind-cpa}}_{\overline{\mathsf{SE}}}(A) \leq \mathbf{Adv}^{\text{ind-cpa}}_{\mathsf{SE}}(A_p)\,. \tag{8.4}$$

Furthermore, F and I use the same amount of resources while A_p and A use the same amount of resources except that each encryption query of A_p is l bits longer than that of A where l is the length of a tag in the scheme MA.

This composition method does not preserve the integrity of ciphertexts for the same reason as in the case of the *Encrypt-and-MAC plaintext* method. In terms of privacy against chosen-ciphertext attacks, this composition method also fails. The reason is the same as before: one can query two messages to the left-or-right encryption oracle, modify part of the response, submit the result to the decryption oracle as a valid query, and completely determine whether the given encryption oracle is a left or a right one.

Furthermore, the *MAC-then-encrypt* composition method does not provide integrity of ciphertexts for the same reason as that for the *Encrypt-and-MAC* method: there are secure encryption schemes for which changing a ciphertext does not affect its decryption.

8.4.4 Encrypt-then-MAC

Figure 8.7 depicts the structure of the *Encrypt-then-MAC* method. Construction 8.43 describes this method in detail. Table 8.5 summarizes the security results.

Construction 8.43: Encrypt-then-MAC composition method

Let $\mathsf{SE} = (\mathsf{KG}_e, \mathsf{Enc}, \mathsf{Dec})$ be a symmetric encryption scheme, and let $\mathsf{MA} =$

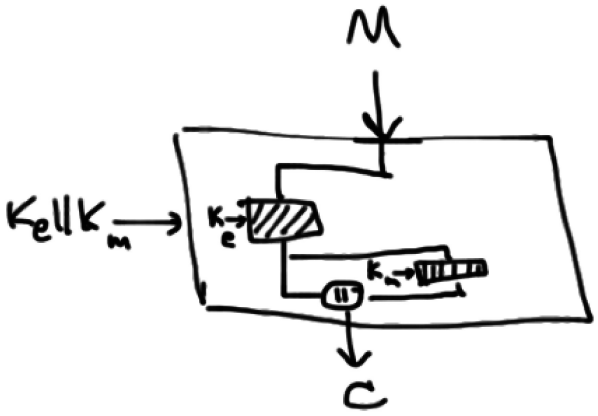


Figure 8.7: A depiction of the *Encrypt-then-MAC* composition method.

$(\mathsf{KG}_m, \mathsf{Tag}, \mathsf{Vf})$ be a message authentication scheme. We define a composite scheme $\overline{\mathsf{SE}} = (\overline{\mathsf{K}}, \overline{\mathsf{E}}, \overline{\mathsf{D}})$ as follows.

Algorithm $\overline{\mathsf{K}}$

$K_e \xleftarrow{\$} \mathsf{KG}_e$; $K_m \xleftarrow{\$} \mathsf{KG}_m$

Return $\langle K_e, K_m \rangle$

Algorithm $\overline{\mathsf{E}}(\langle K_e, K_m \rangle, M)$

$C' \leftarrow \mathsf{Enc}(K_e, M)$; $\tau' \leftarrow \mathsf{Tag}(K_m, C')$; $C \leftarrow C' \| \tau'$

Return C

Algorithm $\overline{\mathsf{D}}(\langle K_e, K_m \rangle, C)$

Parse C as $C' \| \tau'$

$M \leftarrow \mathsf{Dec}(K_e, C')$; $v \leftarrow \mathsf{Vf}(K_m, C', \tau')$

If $v = 1$ return M else return $\perp$

The following theorem states that the *Encrypt-then-MAC* composition method is IND-CPA and INT-PTXT secure assuming that the base MAC scheme is weakly unforgeable.

Theorem 8.44:

(Encrypt-then-MAC method is both IND-CPA secure and INT-PTXT se-

Security notion	Assuming weak MAC	Assuming strong MAC
IND-CPA	$\surd$	$\surd$
IND-CCA	$\times$	$\surd$
INT-PTXT	$\surd$	$\surd$
INT-CTXT	$\times$	$\surd$

Table 8.5: Summary of security results for the *Encrypt-then-MAC* composition method.

cure) Let SE be a symmetric encryption scheme, and let MA be a message authentication scheme. Let $\overline{\mathsf{SE}}$ be the authenticated encryption scheme obtained from SE and MA via the *Encrypt-then-MAC* composition method. Then, if MA is WUF-CMA secure, then $\overline{\mathsf{SE}}$ is INT-PTXT secure. And if SE is IND-CPA secure, then so is $\overline{\mathsf{SE}}$. Specifically, given any ind-cpa adversary A against $\overline{\mathsf{SE}}$, we can construct an ind-cpa adversary A_p such that Equation (8.5) holds. Similarly, given any int-ptxt adversary I against $\overline{\mathsf{SE}}$, we can construct a wuf-cma forger F such that Equation (8.6) holds.

$$\mathbf{Adv}^{\text{ind-cpa}}_{\overline{\mathsf{SE}}}(A) \leq \mathbf{Adv}^{\text{ind-cpa}}_{\mathsf{SE}}(A_p) \tag{8.5}$$

$$\mathbf{Adv}^{\text{int-ptxt}}_{\overline{\mathsf{SE}}}(I) \leq \mathbf{Adv}^{\text{wuf-cma}}_{\mathsf{MA}}(F)\,. \tag{8.6}$$

Furthermore, A and A_p use the same amount of resources while I and F use the same amount of resources except that each tagging query of F is l bits longer than that of I where l is the difference in bits of the length of a ciphertext and that of a plaintext in the scheme SE.

However, a weakly unforgeable base MAC scheme is not enough to obtain neither IND-CCA nor INT-CTXT security when the base MAC scheme is only assumed to be weakly unforgeable.

Theorem 8.45 below implies that the *Encrypt-then-MAC* composition method is IND-CPA, IND-CCA, INT-PTXT, and INT-CTXT secure assuming a strongly unforgeable base MAC scheme. Note that the theorem does not state explicitly that the

composition method is INT-PTXT secure even though it is since INT-CTXT security implies INT-PTXT security.

Theorem 8.45:

(Encrypt-then-MAC method with a SUF-CMA secure MAC is INT-CTXT, IND-CPA, and IND-CCA secure) Let SE be a symmetric encryption scheme, and let MA be a message authentication scheme. Let $\overline{\mathsf{SE}}$ be the authenticated encryption scheme obtained from SE and MA via the *Encrypt-then-MAC* composition method. Then, if MA is SUF-CMA secure, then $\overline{\mathsf{SE}}$ is INT-CTXT secure. If SE is IND-CPA secure, then so is $\overline{\mathsf{SE}}$. And if we have both of the previous conditions, then $\overline{\mathsf{SE}}$ is IND-CCA secure. Specifically, given any ind-cpa adversary A_{cpa}, we can construct an ind-cpa adversary A such that Equation (8.7) holds. Similarly, given any int-ctxt adversary I, we can construct an suf-cma forger F such that Equation (8.8) holds. Moreover, given any ind-cca adversary A_{cca}, we can construct an suf-cma forger F' and an ind-cpa adversary A' such that Equation (8.9) holds.

$$\mathbf{Adv}^{\text{ind-cpa}}_{\overline{\mathsf{SE}}}(A_{\text{cpa}}) \leq \mathbf{Adv}^{\text{ind-cpa}}_{\mathsf{SE}}(A) \tag{8.7}$$

$$\mathbf{Adv}^{\text{int-ctxt}}_{\overline{\mathsf{SE}}}(I) \leq \mathbf{Adv}^{\text{suf-cma}}_{\mathsf{MA}}(F) \tag{8.8}$$

$$\mathbf{Adv}^{\text{ind-cca}}_{\overline{\mathsf{SE}}}(A_{\text{cca}}) \leq 2 \cdot \mathbf{Adv}^{\text{suf-cma}}_{\mathsf{MA}}(F') + \mathbf{Adv}^{\text{ind-cpa}}_{\mathsf{SE}}(A') \tag{8.9}$$

Furthermore, let l be the difference in bits of the length of a ciphertext and that of a plaintext in the scheme SE. Then, A_{cpa} and A use the same amount of resources. Similarly, I and F use the same amount of resources except that each tagging query of F is l bits longer than each encryption query I. Moreover, F' and A' use the same amount of resources as A_{cca} except that each tagging query of F' is l bits longer than each encryption query of A_{cca}.

8.5 Other approaches

Putting together a symmetric encryption scheme and a MAC scheme in a black-box manner to obtain an authenticated encryption scheme is natural considering the fact that there are many existing symmetric encryption and MAC schemes to choose from. However, applying a generic composition method means that one needs to make at least two passes over the plaintext in order to generate the ciphertext. Moreover, for the *MAC-then-encrypt* and *Encrypt-then-MAC* composition methods, the two passes must be done in sequence (as opposed to in parallel). For this, and many other reasons, researchers have designed dedicated authenticated encryption schemes that will provide both privacy and authenticity more efficiently.

These schemes include the RPC mode of Katz and Yung [31], the IACBC mode of Jutla [28], the OCB mode of Rogaway, Bellare, Black, and Kravetz [38], the GCM mode of McGrew and Viega [33], and the XCBC mode of Gligor and Donescu [26] to name a few.

Besides dedicated schemes and schemes obtained by generically combining symmetric encrytion schemes with MAC schemes, there are other ways to obtain authenticated encryption schemes. One approach is to "encrypt with redundancy" and the other is to "encode then encipher."

The "encrypt with redundancy" method [3] refers to the following approach. For encryption, one first adds some "redundancy" to the plaintext (perhaps by appending a fixed string), then encrypts the plaintext with the encryption algorithm of the underlying symmetric encryption scheme. For decryption, one first decrypts the ciphertext, then verifies that the resulting string contains the proper redundancy in the right places and in the right form.

The "encode then encipher" method [12] refers to the following approach. For encryption, one simply "encodes" the plaintext, meaning applying a keyless transformation to it, then "enciphers" the result, meaning applying, say, a block cipher.

8.6 Conclusion

It is natural to hope that one can obtain both privacy and authenticity simultaneously merely by encrypting the data. A typical argument is that it would be hard to forge a ciphertext that decrypts to a "meaningful" message without knowing the secret key. A similar argument is that one can format the plaintext a certain way so that, upon decrypting the corresponding ciphertext, the resulting decryption can be verified that it follows the correct format. These arguments fall into the general approachs referred to as encrypt with redundancy [3] and encode then encipher [12]. It has been found, however, that done generically, they do not yield an encryption scheme that provides both privacy and authenticity simultaneously.

The prevalence of beliefs such as these and the necessity of achieving these goals in many applications led researchers to formalize authenticated encryption, a construct with the goal of providing both privacy and authenticity simultaneously, to formalize the security properties that such a construct should provide, and to explore ways in which we can properly design such constructs. The security definitions covering the privacy aspect are the indistinguishability against chosen-plaintext and indistinguishability against chosen-ciphertext notions as for regular symmetric encryption schemes. The security definitions covering the authenticity aspect, however, emerged. They are the notions of integrity of plaintexts (INT-PTXT) and integrity of ciphertexts (INT-CTXT). There are many practical constructs in use today that provides security under these notions. These are the various dedicated modes of operation and the authenticated encryption schemes obtained by combining existing symmetric encryption schemes and MAC schemes in a black-box manner.

8.7 Exercises

Exercise 8.1. Explain in your own words the difference between integrity of plaintext and integrity of ciphertext. Can you think of an application in which one would

care about one but not the other?

Exercise 8.2. What is the role of the set S in the game in Definition 8.34?

Exercise 8.3. What is the role of the set S in the game in Definition 8.35?

Exercise 8.4. Discuss efficiency trade-offs among the three generic composition methods. Can you think of applications or contexts in which one would be preferred over the others for efficiency reasons?

Exercise 8.5. What does it mean for a generic composition method to be "secure"?

Exercise 8.6. What does it mean for a generic composition method to be "insecure"?

Exercise 8.7. Does the ECB mode discussed in Section 5.4.1 provide authenticity? Prove your answer.

Exercise 8.8. Does the CBC mode discussed in Section 5.4.2 provide authenticity? Prove your answer.

Exercise 8.9. Show that the counter mode discussed in Section 5.4.5 does not provide authenticity. In particular, show that it is insecure under the INT-PTXT definition.

Exercise 8.10. Consider the following statement.

> The *Encrypt-and-MAC* composition method never yields a secure authenticated encryption scheme.

Is this statement true or false. Explain your answer.

Bibliography

[1] Advanced Encryption Standard. National Institute of Standards and Technology, NIST FIPS PUB 197, U.S. Department of Commerce, November 2001.

[2] Werner Alexi, Benny Chor, Oded Goldreich, and Claus-Peter Schnorr. RSA and Rabin functions: Certain parts are as hard as the whole. *SIAM Journal on Computing*, 17(2):194–209, 1988.

[3] Jee Hea An and Mihir Bellare. Does encryption with redundancy provide authenticity? In Birgit Pfitzmann, editor, *Advances in Cryptology – EUROCRYPT 2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 512–528, Innsbruck, Austria, May 6–10, 2001. Springer-Verlag, Berlin, Germany.

[4] ANSI X9.17 (revised). American National Standard for Financial Institution Key Management (Wholesale), American Bankers Association, 1985.

[5] Mihir Bellare. Practice-oriented provable-security. In *Proceedings of the First International Workshop on Information Security*, ISW '97, pages 221–231, London, UK, UK, 1998. Springer-Verlag, Berlin, Germany.

[6] Mihir Bellare. New proofs for NMAC and HMAC: Security without collision-resistance. In Cynthia Dwork, editor, *Advances in Cryptology – CRYPTO 2006*, volume 4117 of *Lecture Notes in Computer Science*, pages 602–619, Santa Barbara, CA, USA, August 20–24, 2006. Springer-Verlag, Berlin, Germany.

[7] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Keying hash functions for message authentication. In Neal Koblitz, editor, *Advances in Cryptology – CRYPTO'96*, volume 1109 of *Lecture Notes in Computer Science*, pages 1–15, Santa Barbara, CA, USA, August 18–22, 1996. Springer-Verlag, Berlin, Germany.

[8] Mihir Bellare, Anand Desai, Eric Jokipii, and Phillip Rogaway. A concrete security treatment of symmetric encryption. In *38th Annual Symposium on Foundations of Computer Science*, pages 394–403, Miami Beach, Florida, October 19–22, 1997. IEEE Computer Society Press.

[9] Mihir Bellare, Joe Kilian, and Phillip Rogaway. The security of the cipher block chaining message authentication code. In Yvo Desmedt, editor, *Advances in Cryptology – CRYPTO'94*, volume 839 of *Lecture Notes in Computer Science*, pages 341–358, Santa Barbara, CA, USA, August 21–25, 1994. Springer-Verlag, Berlin, Germany.

[10] Mihir Bellare, Tadayoshi Kohno, and Chanathip Namprempre. Breaking and provably repairing the SSH authenticated encryption scheme: A case study of the encode-then-encrypt-and-mac paradigm. *ACM Transactions on Information and System Security (TISSEC)*, 7(2):206–241, 2004.

[11] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM CCS 93: 1st Conference on Computer and Communications Security*, pages 62–73, Fairfax, Virginia, USA, November 3–5, 1993. ACM Press.

[12] Mihir Bellare and Phillip Rogaway. Encode-then-encipher encryption: How to exploit nonces or redundancy in plaintexts for efficient cryptography. In Tatsuaki Okamoto, editor, *Advances in Cryptology – ASIACRYPT 2000*, volume

1976 of *Lecture Notes in Computer Science*, pages 317–330, Kyoto, Japan, December 3–7, 2000. Springer-Verlag, Berlin, Germany.

[13] Mihir Bellare and Phillip Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In Serge Vaudenay, editor, *Advances in Cryptology – EUROCRYPT 2006*, volume 4004 of *Lecture Notes in Computer Science*, pages 409–426, St. Petersburg, Russia, May 28 – June 1, 2006. Springer-Verlag, Berlin, Germany.

[14] Steven M. Bellovin. Problem areas for the IP security protocols. In *Proceedings of the 6th conference on USENIX Security Symposium, Focusing on Applications of Cryptography - Volume 6*, SSYM'96, pages 21–21, Berkeley, CA, USA, 1996. USENIX Association.

[15] John Black, Martin Cochran, and Thomas Shrimpton. On the impossibility of highly-efficient blockcipher-based hash functions. In Ronald Cramer, editor, *Advances in Cryptology – EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 526–541. Springer-Verlag, Berlin, Germany, May 22–26, 2005.

[16] John Black and Phillip Rogaway. CBC MACs for arbitrary-length messages: The three-key constructions. In Mihir Bellare, editor, *Advances in Cryptology – CRYPTO 2000*, volume 1880 of *Lecture Notes in Computer Science*, pages 197–215, Santa Barbara, CA, USA, August 20–24, 2000. Springer-Verlag, Berlin, Germany.

[17] Lenore Blum, Manuel Blum, and Mike Shub. A simple unpredictable pseudo-random number generator. *SIAM Journal on Computing*, 15(2):364–383, May 1986.

[18] Manuel Blum and Silvio Micali. How to generate cryptographically strong sequences of pseudorandom bits. *SIAM Journal on Computing*, 13(4):850–864, 1984.

[19] Alexandra Boldyreva and Virendra Kumar. A new pseudorandom generator from collision-resistant hash functions. In Orr Dunkelman, editor, *Topics in Cryptology – CT-RSA 2012*, volume 7178 of *Lecture Notes in Computer Science*, pages 187–202, San Francisco, CA, USA, February 2012. Springer-Verlag, Berlin, Germany.

[20] Information technology – Security techniques – Message Authentication Codes (MACs) – Part 1: Mechanisms using a block cipher. ISO/IEC 9797-1, 1999.

[21] Data Encryption Standard. National Bureau of Standards, NBS FIPS PUB 46, U.S. Department of Commerce, January 1977.

[22] Anand Desai, Alejandro Hevia, and Yiqun Lisa Yin. A practice-oriented treatment of pseudorandom number generators. In Lars R. Knudsen, editor, *Advances in Cryptology – EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 368–383, Amsterdam, The Netherlands, April 28 – May 2, 2002. Springer-Verlag, Berlin, Germany.

[23] Digital Signature Standard. National Institute of Standards and Technology, NIST FIPS PUB 186, U.S. Department of Commerce, May 1994.

[24] Éliane Jaulmes, Antoine Joux, and Frédéric Valette. On the security of randomized CBC-MAC beyond the birthday paradox limit: A new construction. In Joan Daemen and Vincent Rijmen, editors, *Fast Software Encryption 2002*, volume 2365 of *Lecture Notes in Computer Science*, Leuven, Belgium, 2002. Springer-Verlag, Berlin, Germany.

[25] Scott R. Fluhrer, Itsik Mantin, and Adi Shamir. Weaknesses in the key scheduling algorithm of RC4. In *8th Annual International Workshop on Selected Areas in Cryptography*, SAC '01, pages 1–24, London, UK, UK, 2001. Springer-Verlag, Berlin, Germany.

[26] Virgil D. Gligor and Pompiliu Donescu. Fast encryption and authentication: XCBC encryption and XECB authentication modes. In Mitsuru Matsui, editor, *Fast Software Encryption 2001*, volume 2355 of *Lecture Notes in Computer Science*, pages 92–108, Yokohama, Japan, 2001. Springer-Verlag, Berlin, Germany.

[27] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28:270–299, 1984.

[28] Charanjit S. Jutla. Encryption modes with almost free message integrity. In Birgit Pfitzmann, editor, *Advances in Cryptology – EUROCRYPT 2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 529–544, Innsbruck, Austria, May 6–10, 2001. Springer-Verlag, Berlin, Germany.

[29] David Kahn. *The Codebreakers: The Comprehensive History of Secret Communication from Ancient Times to the Internet*. Scribner, 1996.

[30] Burton S. Kaliski Jr. A pseudo-random bit generator based on elliptic lograrithms. In Andrew M. Odlyzko, editor, *Advances in Cryptology – CRYPTO'86*, volume 263 of *Lecture Notes in Computer Science*, pages 84–103, Santa Barbara, CA, USA, August 1986. Springer-Verlag, Berlin, Germany.

[31] Jonathan Katz and Moti Yung. Unforgeable encryption and chosen ciphertext secure modes of operation. In Bruce Schneier, editor, *Fast Software Encryption 2000*, volume 1978 of *Lecture Notes in Computer Science*, pages 284–299,

New York, NY, USA, 2000. Springer-Verlag, Berlin, Germany.

[32] Lars Knudsen and Tadayoshi Kohno. Analysis of RMAC. In Thomas Johansson, editor, *Fast Software Encryption 2003*, volume 2887 of *Lecture Notes in Computer Science*, pages 182–191, Lund, Sweden, 2003. Springer-Verlag, Berlin, Germany.

[33] David A. McGrew and John Viega. The security and performance of the Galois/Counter Mode (GCM) of operation. In *INDOCRYPT*, pages 343–355, 2004.

[34] Erez Petrank and Charles Rackoff. CBC MAC for real-time data sources. *Journal of Cryptology*, 13(3):315–338, 2000.

[35] Bart Preneel. Design principles for dedicated hash functions. In Ross J. Anderson, editor, *Fast Software Encryption'93*, volume 809 of *Lecture Notes in Computer Science*, pages 71–82, Cambridge, UK, 1993. Springer-Verlag, Berlin, Germany.

[36] Recommendation for the Triple Data Encryption Algorithm (TDEA) block cipher. National Institute of Standards and Technology, NIST Special Publication 800-67, U.S. Department of Commerce, January 2012.

[37] Phillip Rogaway. Formalizing human ignorance. In *VIETCRYPT*, pages 211–228, 2006.

[38] Phillip Rogaway, Mihir Bellare, and John Black. OCB: A block-cipher mode of operation for efficient authenticated encryption. *ACM Transactions on Information and System Security (TISSEC)*, 6(3):365–403, 2003.

[39] Phillip Rogaway and Thomas Shrimpton. Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-

preimage resistance, and collision resistance. In Bimal K. Roy and Willi Meier, editors, *Fast Software Encryption 2004*, volume 3017 of *Lecture Notes in Computer Science*, pages 371–388, New Delhi, India, 2004. Springer-Verlag, Berlin, Germany.

[40] Phillip Rogaway and John P. Steinberger. Constructing cryptographic hash functions from fixed-key blockciphers. In David Wagner, editor, *Advances in Cryptology – CRYPTO 2008*, pages 433–450. Springer-Verlag, Berlin, Germany, August 2008.

[41] Joachim Rosenthal. A polynomial description of the Rijndael advanced encryption standard. *Journal of Algebra and its Applications*, 2(2):223–236, 2003.

[42] Secure Hash Standard. National Institute of Standards and Technology, NIST FIPS PUB 180-1, U.S. Department of Commerce, April 1995.

[43] Secure Hash Standard. National Institute of Standards and Technology, NIST FIPS PUB 180, U.S. Department of Commerce, May 1993.

[44] Adi Shamir. On the generation of cryptographically strong pseudorandom sequences. *ACM Transactions on Computer Systems*, 1(1):38–44, February 1983.

[45] Thomas Shrimpton and Martijn Stam. Building a collision-resistant compression function from non-compressing primitives. In *35rd International Colloquium on Automata, Languages and Programming – ICALP 2008*, volume 5126 of *Lecture Notes in Computer Science*, pages 643–654. Springer-Verlag, Berlin, Germany, 2008.

[46] David Wagner and Ian Goldberg. Proofs of security for the unix password hashing algorithm. In Tatsuaki Okamoto, editor, *Advances in Cryptology – ASI-*

ACRYPT 2000, volume 1976 of *Lecture Notes in Computer Science*, pages 560–572, Kyoto, Japan, December 3–7, 2000. Springer-Verlag, Berlin, Germany.

[47] Wireshark. `http://www.wireshark.org`.

[48] Andrew C. Yao. Theory and applications of trapdoor functions. In *23rd Annual Symposium on Foundations of Computer Science*, pages 80–91, Chicago, Illinois, November 3–5, 1982. IEEE Computer Society Press.

Index

Advanced Encryption Standard, 28
AES, 28
ANSI 9.17, 56
ARMAC, 125
asymmetric-key cryptography, 5
authenticated encryption
    example, 144
    security definition, 141
    syntax, 140

block cipher, 19
    attack, 45
    example, 21, 28, 34
    security definition, 35

Caesar cipher, 11
CBC, 75
CBC MAC, 122
CFB, 76
ciphertext, 11
collision resistance, 106
CTR, 79

Data Encryption Standard, 21
dedicated authenticated encryption, 158
DES, 21
DESX, 27
deterministic algorithm, 62
double DES, 24

ECB, 73
EMAC, 124
encode then encipher, 158
encrypt with redundancy, 158
Encrypt-and-MAC, 146, 149
Encrypt-then-MAC, 147, 154

family of permutations, 33
FIPS 186, 56

generic composition paradigm, 144

hash function, 102
    collision resistance, 106
    example, 109
    pre-image security, 104

second pre-image security, 105
security definition, 103
HMAC, 130

IND-CCA, 68
IND-CPA, 67
information advantage, 2
INT-CTXT, 143
INT-PTXT, 141

MAC, 115
MAC-then-encrypt, 147, 152
MD5, 110
message authentication code, 115
attack, 131
correctness condition, 116
example, 122
security definition, 118
syntax, 116
message digest, 102
modes of operation
CBC, 75
CFB, 76
CTR, 79
ECB, 73
OFB, 78

non-deterministic algorithm, 62

OFB, 78
one-time pad, 13

perfect secrecy, 13
plaintext, 11
pre-image security, 104
PRG, 54
private key, 9
provable security, 90
pseudorandom number generator, 54
attack, 56
example, 55
security definition, 54
public key, 5

random oracle model, 108
RMAC, 127

second pre-image security, 105
secret key, 9
security definition
IND-CCA, 68
IND-CPA, 67
INT-CTXT, 143
INT-PTXT, 141
PRF, 44
PRG, 54
PRP-CCA, 40
PRP-CPA, 38
SUF-CMA, 120
WUF-CMA, 118
SHA-1, 110
shift cipher, 11

SUF-CMA, 120
symmetric encryption, 9, 63
    attack, 82
    correctness condition, 63
    example, 71
    security definition, 67
    security proof, 91
    syntax, 63
symmetric-key cryptography, 5

triple DES, 25

weak keys, 24
WUF-CMA, 118

XCBC MAC, 128

About the author

Chanathip Namprempre is a faculty member in the Faculty of Engineering, Thammasat University, Thailand. Her research interests include cryptography, computer and network security, and distributed systems. She received her S.B. in Computer Science and Engineering and M.Eng. in Electrical Engineering and Computer Science from Massachusetts Institute of Technology in 1996 and 1997, respectively. She received her Ph.D. in Computer Science from University of California, San Diego in 2002.

Symmetric Cryptography Basics

It is a fact that the more we rely on computer systems, the more important it becomes to secure them. Perpetrators today have more incentives than ever to subvert the security mechanisms so as to exploit flaws in systems for their own gain. While it is true that most incidents reported in the news have to do with social engineering and exploits of software bugs, it is also clear that one would be hard pressed to protect computer and network systems without the use of cryptography, a field of study that is concerned with secure communication in the presence of an adversary. The focus of this book is on symmetric cryptosystems, a setting in which the communicating parties possess some shared secret before they start to communicate. The approach is to cover concrete, well-known cryptographic primitives as soon as possible without sacrificing fundamental concepts in modern cryptography.

Chanathip Namprempre, Ph.D.
Faculty of Engineering, Thammasat University
THAILAND