Chapter 1

BEGINNING WITH A SIMPLE COMMUNICATION GAME

We begin this book with a simple example of applying cryptography to solve a simple problem. This example of cryptographic application serves three purposes from which we will unfold the topics of this book:

- To provide an initial demonstration on the effectiveness and practicality of using cryptography for solving subtle problems in applications

- To suggest an initial hint on the foundation of cryptography

- To begin our process of establishing a required mindset for conducting the development of cryptographic systems for information security

To begin with, we shall pose a trivially simple problem and then solve it with an equally simple solution. The solution is a two-party game which is very familiar to all of us. However, we will realize that our simple game soon becomes troublesome when our game-playing parties are physically remote from each other. The physical separation of the game-playing parties eliminates the basis for the game to be played fairly. The trouble then is, the game-playing parties cannot trust the other side to play the game fairly.

The need for a fair playing of the game for remote players will "inspire" us to strengthen our simple game by protecting it with a shield of armor. Our strengthening method follows the long established idea for protecting communications over open networks: hiding information using cryptography.

After having applied cryptography and reached a quality solution to our first security problem, we shall conduct a series of discussions on the quality criteria for cryptographic systems (§1.2). The discussions will serve as a background and cultural introduction to the areas in which we research and develop technologies for protecting sensitive information.

1.1 A Communication Game

Here is a simple problem. Two friends, Alice and Bob[a], want to spend an evening out together, but they cannot decide whether to go to the cinema or the opera. Nevertheless, they reach an agreement to let a coin decide: playing a coin tossing game which is very familiar to all of us.

Alice holds a coin and says to Bob, "You pick a side then I will toss the coin." Bob does so and then Alice tosses the coin in the air. Then they both look to see which side of the coin landed on top. If Bob's choice is on top, Bob may decide where they go; if the other side of the coin lands on top, Alice makes the decision.

In the study of communication procedures, a multi-party-played game like this one can be given a "scientific sounding" name: protocol. A protocol is a well-defined procedure running among a plural number of participating entities. We should note the importance of the plurality of the game participants; if a procedure is executed entirely by one entity only then it is a procedure and cannot be called a protocol.

1.1.1 Our First Application of Cryptography

Now imagine that the two friends are trying to run this protocol over the telephone. Alice offers Bob, "You pick a side. Then I will toss the coin and tell you whether or not you have won." Of course Bob will not agree, because he cannot verify the outcome of the coin toss.

However we can add a little bit of cryptography to this protocol and turn it into a version workable over the phone. The result will become a cryptographic protocol, our first cryptographic protocol in this book! For the time being, let us just consider our "cryptography" as a mathematical function $f(x)$ which maps over the integers and has the following magic properties:

Property 1.1: Magic Function f

I) *For every integer x, it is easy to compute $f(x)$ from x, while given any value $f(x)$ it is impossible to find any information about a pre-image x, e.g., whether x is an odd or even number.*

[a]They are the most well-known figures in the area of cryptography, cryptographic protocols and information security; they will appear in most of the cryptographic protocols in this book.

Protocol 1.1: Coin Flipping Over Telephone

PREMISE
Alice and Bob have agreed:

i) a "magic function" f with properties specified in Property 1.1

ii) an even number x in $f(x)$ represents HEADS and the other case represents TAILS

(∗ Caution: due to (ii), this protocol has a weakness, see Exercise 1.2 ∗)

1. Alice picks a large random integer x and computes $f(x)$;
   she reads $f(x)$ to Bob over the phone;
2. Bob tells Alice his guess of x as even or odd;
3. Alice reads x to Bob;
4. Bob verifies $f(x)$ and sees the correctness/incorrectness of his guess.

II) It impossible to find a pair of integers (x, y) satisfying $x \neq y$ and $f(x) = f(y)$.

In Property 1.1, the adjectives "easy" and "impossible" have meanings which need further explanations. Also because these words are related to a degree of difficulty, we should be clear about their quantifications. However, since for now we view the function f as a magic one, it is safe for us to use these words in the way they are used in the common language. In Chapter 4 we will provide mathematical formulations for various uses of "easy" and "impossible" in this book. One important task for this book is to establish various quantitative meanings for "easy," "difficult" or even "impossible." In fact, as we will eventually see in the final technical chapter of this book (Chapter 19) that in our final realization of the coin-flipping protocol, the two uses of "impossible" for the "magic function" in Property 1.1 will have very different quantitative measures.

Suppose that the two friends have agreed on the magic function f. Suppose also that they have agreed that, e.g., an even number represents HEADS and an odd number represents TAILS. Now they are ready to run our first cryptographic protocol, Prot 1.1, over the phone.

It is not difficult to argue that Protocol "Coin Flipping Over Telephone" works quite well over the telephone. The following is a rudimentary "security analysis." (Warning: the reason for us to quote "security analysis" is because our analysis provided here is far from adequate.)

1.1.1.1 A Rudimentary "Security Analysis"

First, from "Property II" of f, Alice is unable to find two different numbers x and y, one is odd and the other even (this can be expressed as $x \neq y \pmod 2$) such that $f(x) = f(y)$. Thus, once having read the value $f(x)$ to Bob over the phone (Step 1), Alice has committed to her choice of x and cannot change her mind. That's when Alice has completed her coin flipping.

Secondly, due to "Property I" of f, given the value $f(x)$, Bob cannot determine whether the pre-image used by Alice is odd or even and so has to place his guess (in Step 2) as a real guess (i.e., an uneducated guess). At this point, Alice can convince Bob whether he has guessed right or wrong by revealing her pre-image x (Step 3). Indeed, Bob should be convinced if his own evaluation of $f(x)$ (in Step 4) matches the value told by Alice in Step 1 and if he believes that the properties of the agreed function hold. Also, the coin-flipping is fair if x is taken from an adequately large space so Bob could not have a guessing advantage, that is, some strategy that gives him a greater than 50-50 chance of winning.

We should notice that in our "security analysis" for Prot 1.1 we have made a number of simplifications and omissions. As a result, the current version of the protocol is far from a concrete realization. Some of these simplifications and omissions will be discussed in this chapter. However, necessary techniques for a proper and concrete realization of this protocol and methodologies for analyzing its security will be the main topics for the remainder of the whole book. We shall defer the proper and concrete realization of Prot 1.1 (more precisely, the "magic function" f) to the final technical chapter of this book (Chapter 19). There, we will be technically ready to provide a formal security analysis on the concrete realization.

1.1.2 An Initial Hint on Foundations of Cryptography

Although our first protocol is very simple, it indeed qualifies as a cryptographic protocol because the "magic function" the protocol uses is a fundamental ingredient for modern cryptography: **one-way function**. The two magic properties listed in Property 1.1 pose two **computationally intractable** problems, one for Alice, and the other for Bob.

From our rudimentary security analysis for Prot 1.1 we can claim that the existence of one-way function implies a possibility for secure selection of recreation venue. The following is a reasonable generalization of this claim:

The existence of a one-way function implies the existence of a secure cryptographic system.

It is now well understood that the converse of this claim is also true:

The existence of a secure cryptographic system implies the existence of a one-way function.

It is widely believed that one-way function does exist. Therefore we are optimistic on securing our information. Our optimism is often confirmed by our everyday experience: many processes in our world, mathematical or otherwise, have a one-way property. Consider the following phenomenon in physics (though not an extremely precise analogy for mathematics): it is an easy process for a glass to fall on the floor and break into pieces while dispersing a certain amount of energy (e.g., heat, sound or even some dim light) into the surrounding environment. The reverse process, recollecting the dispersed energy and using it to reintegrate the broken pieces back into a whole glass, must be a very hard problem if not impossible. (If possible, the fully recollected energy could actually bounce the reintegrated glass back to the height where it started to fall!)

In Chapter 4 we shall see a class of mathematical functions which provide the needed one-way properties for modern cryptography.

1.1.3 Basis of Information Security: More than Computational Intractability

We have just claimed that information security requires certain mathematical properties. Moreover, we have further made an optimistic assertion in the converse direction: mathematical properties imply (i.e., guarantee) information security.

However, in reality, the latter statement is not unconditionally true! Security in real world applications depends on many real world issues. Let us explain this by continuing using our first protocol example.

We should point out that many important issues have not been considered in our rudimentary security analysis for Prot 1.1. In fact, Prot 1.1 itself is a much simplified specification. It has omitted some details which are important to the security services that the protocol is designed to offer. The omission has prevented us from asking several questions.

For instance, we may ask: has Alice really been forced to stick to her choice of x? Likewise, has Bob really been forced to stick to his even-odd guess of x? By "forced," we mean whether voice over telephone is sufficient for guaranteeing the strong mathematical property to take effect. We may also ask whether Alice has a good random number generator for her to acquire the random number x. This quality can be crucially important in a more serious application which requires making a fair decision.

All these details have been omitted from this simplified protocol specification and therefore they become hidden assumptions (more on this later). In fact, if this protocol is used for making a more serious decision, it should include some *explicit* instructions. For example, both participants may consider recording the other party's voice when the value $f(x)$ and the even/odd guess are pronounced over the phone, and replay the record in case of dispute.

Often cryptographic systems and protocols, in particular, those introduced by a textbook on cryptography, are specified with simplifications similar to the case in Protocol "Coin Flipping Over Telephone." Simplifications can help to achieve presentation clarity, especially when some agreement may be thought of as obvious. But sometimes a hidden agreement or assumption may be subtle and can be exploited to result in a surprising consequence. This is somewhat ironic to the "presentation clarity" which is originally intended by omitting some details. A violation of an assumption of a security system may allow an attack to be exploited and the consequence can be the nullification of an intended service. It is particularly difficult to notice a violation of a hidden assumption. In §1.2.5 we shall provide a discussion on the importance of explicit design and specification of cryptographic systems.

A main theme of this book is to explain that security for real world applications has many application related subtleties which must be considered seriously.

1.1.4 Modern Role of Cryptography: Ensuring Fair Play of Games

Cryptography was once a preserve of governments. Military and diplomatic organizations used it to keep messages secret. Nowadays, however, cryptography has a modernized role in addition to keeping secrecy of information: ensuring fair play of "games" by a much enlarged population of "game players." That is part of the reasons why we have chosen to begin this book on cryptography with a communication game.

Deciding on a recreation venue may not be seen as a serious business, and so doing it via flipping a coin over the phone can be considered as just playing a small communication game for fun. However, there are many communications "games" which must be taken much more seriously. With more and more business and e-commerce activities being and to be conducted electronically over open communications networks, many cases of our communications involve various kinds of "game playing." (In the Preface of this book we have listed various business and services examples which can be conducted or offered electronically over open networks; all of them involve some interactive actions of the participants by following a set of rules, which can be viewed as "playing communication games".) These "games" can be very important!

In general, the "players" of such "games" are physically distant from each other and they communicate over open networks which are notorious for lack of security. The physical distance combined with the lack of security may help and/or encourage some of the "game players" (some of whom can even be uninvited) to try to defeat the rule of game in some clever way. The intention for defeating the rule of game is to try to gain some unentitled advantage, such as causing disclosure of confidential information, modification of data without detection, forgery of false evidence, repudiation of an obligation, damage of accountability or trust, reduction of availability or nullification of services, and so on. The importance of our modern communications in business, in the conduct of commerce and in providing services (and many more others, such as securing missions of companies, personal information, military actions and state affairs) mean that no unentitled advantage should be gained to a player who does not conform the rule of game.

In our development of the simple "Coin-Flipping-Over-Telephone" cryptographic protocol, we have witnessed the process whereby an easy-to-sabotage communication game evolves to a cryptographic protocol and thereby offers desired security services. Our example demonstrates the effectiveness of cryptography in maintaining the order of "game playing." Indeed, the use of cryptography is an effective and the *only practical* way to ensure secure communications over open computers and communications networks. Cryptographic protocols are just communication procedures armored with the use of cryptography and thereby have protective functions designed to keep communications in good order. The endless need for securing communications for electronic commerce, business and services coupled with another need for anticipating the ceaseless temptation of "breaking the rules of the game" have resulted in the existence of many cryptographic systems and protocols, which form the subject matter of this book.

1.2 Criteria for Desirable Cryptographic Systems and Protocols

We should start by asking a fundamental question:

What is a good cryptographic system/protocol?

Undoubtedly this question is not easy to answer! One reason is that there are many answers to it depending on various meanings the word *good* may have. It is a main task for this book to provide comprehensive answers to this fundamental question. However, here in this first chapter we should provide a few initial answers.

1.2.1 Stringency of Protection Tuned to Application Needs

Let us begin with considering our first cryptographic protocol we designed in §1.1.1.

We can say that Protocol "Coin Flipping Over Telephone" is good in the sense that it is conceptually very simple. Some readers who may already be familiar with many practical one-way hash functions, such as SHA-1 (see §10.3.1), might further consider that the function $f(x)$ is also easy to implement even in a pocket calculator. For example, an output from SHA-1 is a bit string of length of 160 bits, or 20 bytes (1 byte = 8 bits); using the hexadecimal encoding scheme (see

Example 5.17) such an output can be encoded into 40 hexadecimal characters[b] and so it is just not too tedious for Alice (Bob) to read (and jot down) over the phone. Such an implementation should also be considered sufficiently secure for Alice and Bob to decide their recreation venue: if Alice wants to cheat, she faces a non-trivial difficulty in order to find $x \neq y \pmod 2$ with $f(x) = f(y)$; likewise, Bob will also have to face a non-trivial difficulty, that is, given $f(x)$, to determine whether x is even or odd.

However, our judgement on the quality of Protocol "Coin Flipping Over Telephone" realized using SHA-1 is based on a level of non-seriousness that the game players expect on the consequence of the game. In many more serious applications (e.g., one which we shall discuss in §1.2.4), a fair coin-flipping primitive for cryptographic use will in general require much stronger one-way and commitment-binding properties than a practical one-way hash function, such as SHA-1, can offer. We should notice that a function with the properties specified in Property 1.1, if we take the word "impossible" literally, is a *completely secure* one-way function. Such a function is not easily implementable. Worse, even its very existence remains an open question (even though we are optimistic about the existence, see our optimistic view in §1.1.2, we shall further discuss the condition for the existence of a one-way function in Chapter 4). Therefore, for more serious applications of fair coin-flipping, practical hash functions won't be considered good; much more stringent cryptographic techniques are necessary. On the other hand, for deciding a recreation venue, use of heavyweight cryptography is clearly unnecessary or overkill.

We should point out that there are applications where a too-strong protection will even prevent an intended security service from functioning properly. For example, Rivest and Shamir propose a micropayment scheme, called MicroMint [242], which works by making use of a known deficiency in an encryption algorithm to their advantage. That payment system exploits a reasonable assumption that only a resourceful service provider (e.g., a large bank or financial institute) is able to prepare a large number of "collisions" under a practical one-way function, and do so economically. This is to say that the service provider can compute k distinct numbers $(x_1, x_2, \ldots, x_k)$ satisfying

$$f(x_1) = f(x_2) = \cdots = f(x_k).$$

The numbers x_1, x_2, …, x_k, are called collision under the one-way function f. A pair of collisions can be checked efficiently since the one-way function can be evaluated efficiently, they can be considered to have been issued by the resourceful service provider and hence can represent a certified value. The Data Encryption Standard (DES, see §7.6) is suggested as a suitable algorithm for implementing such a one-way function ([242]) and so to achieve a relatively small output space (64 binary bits). Thus, unlike in the normal cryptographic use of one-way functions

[b]Hexadecimal characters are those in the set $\{0, 1, 2, \ldots, 9, \mathrm{A}, \mathrm{B}, \ldots, \mathrm{F}\}$ representing the 16 cases of 4-bit numbers.

where a collision almost certainly constitutes a successful attack on the system (for example, in the case of Protocol "Coin Flipping Over Telephone"), in MicroMint, collisions are used in order to enable a fancy micropayment service! Clearly, a strong one-way function with a significantly larger output space (i.e., $\gg 64$ bits, such as SHA-1 with 160 bits) will nullify this service even for a resourceful service provider (in §3.6 we will study the computational complexity for finding collisions under a hash function).

Although it is understandable that using heavyweight cryptographic technologies in the design of security systems (for example, wrapping with layers of encryption, arbitrarily using digital signatures, calling for online services from a trusted third party or even from a large number of them) may provide a better feeling that a stronger security may have been achieved (it may also ease the design job), often this feeling only provides a false sense of assurance. Reaching the point of overkill with unnecessary armor is undesirable because in so doing it is more likely to require stronger security assumptions and to result in a more complex system. A complex system can also mean an increased difficulty for security analysis (hence more likelihood to be error-prone) and secure implementation, a poorer performance, and a higher overhead cost for running and maintenance.

It is more interesting and a more challenging job to design cryptographic or security systems which use only necessary techniques while achieving adequate security protection. This is an important element for cryptographic and security systems to qualify as *good*.

1.2.2 Confidence in Security Based on Established "Pedigree"

How can we be confident that a cryptographic algorithm or a protocol is secure? Is it valid to say that an algorithm is secure because nobody has broken it? The answer is, unfortunately, *no*. In general, what we can say about an unbroken algorithm is merely that we do not know how to break it yet. Because in cryptography, the meaning of a broken algorithm sometimes has quantitative measures; if such a measure is missing from an unbroken algorithm, then we cannot even assert whether or not an unbroken algorithm is more secure than a known broken one.

Nevertheless, there are a few exceptions. In most cases, the task of breaking a cryptographic algorithm or a scheme boils down to solving some mathematical problems, such as to find a solution to an equation or to invert a function. These mathematical problems are considered "hard" or "intractable." A formal definition for "hard" or "intractable" will be given in Chapter 4. Here we can informally, yet safely, say that a mathematical problem is intractable if it cannot be solved by any known methods within a reasonable length of time.

There are a number of well-known intractable problems that have been frequently used as standard ingredients in modern cryptography, in particular, in public-key or asymmetric cryptography (see §8.3—§8.14). For example, in public-key cryptography, intractable problems include the integer factorization problem, the discrete logarithm problem, the Diffie-Hellman problem, and a few associated problems (we will define and discuss these problems in Chapter 8). These problems can be referred to as established "pedigree" ones because they have sustained a long history of study by generations of mathematicians and as a result, they are now trusted as really hard with a high degree of confidence.

Today, a standard technique for establishing a high degree of confidence in security of a cryptographic algorithm is to conduct a formal proof which demonstrates that an attack on the algorithm can lead to a solution to one of the accepted "pedigree" hard problems. Such a proof is an efficient mathematical transformation, or a sequence of such transformations, leading from an attack on an algorithm to a solution to a hard problem. Such an efficient transformation is called a reduction which "reduces" an attack to a solution to a hard problem. Since we are highly confident that the resultant solution to the hard problem is unlikely to exist (especially under the time cost measured by the attack and the reduction transformation), we will be able to derive a measurable confidence that the alleged attack should not exist. This way of security proof is therefore named "reduction to contradiction:" an easy solution to a hard problem.

Formally provable security, in particular under various powerful attacking model called *adaptive attacks*, forms an important criterion for cryptographic algorithms and protocols to be regarded as *good*. We shall use *fit-for-application security* to name security qualities which are established through formal and reduction-to-contradiction approach under powerful attacking models.

As an important topic of this book, we shall study fit-for-application security for many cryptographic algorithms and protocols.

1.2.3 Practical Efficiency

When we say that a mathematical problem is efficient or is efficiently solvable, we basically assert that the problem is solvable in time which can be measured by a polynomial in the size of the problem. A formal definition for efficiency, which will let us provide precise measures of this assertion, will be provided in Chapter 4.

Without looking into quantitative details of this assertion for the time being, we can roughly say that this assertion divides all the problems into two classes: tractable and intractable. This division plays a fundamental role in the foundations for modern cryptography: a complexity-theoretically based one. Clearly, a cryptographic algorithm must be designed such that it is tractable on the one hand and so is usable by a legitimate user, but is intractable on the other hand and so constitutes a difficult problem for a non-user or an attacker to solve.

We should however note that this assertion for solubility covers a vast span of quantitative measures. If a problem's computing time for a legitimate user is measured by a huge polynomial, then the "efficiency" is in general impractical, i.e., can have no value for a practical use. Thus, an important criterion for a cryptographic algorithm being *good* is that it should be *practically efficient* for a legitimate user. In specific, the polynomial that measures the resource cost for the user should be small (i.e., have a small degree, the degree of a polynomial will be introduced in Chapter 4).

In Chapter 14 we will discuss several pioneering works on provably strong public-key cryptosystems. These works propose public-key encryption algorithms under a common motivation that many basic versions of public-key encryption algorithms are insecure (we name those insecure schemes "textbook crypto" because most text-books in cryptography introduce them up to their basic and primitive versions; they will be introduced in Part III of this book). However, most pioneering works on provably strong public-key cryptosystems resort to a bit-by-bit encryption method, [125, 210, 241], some even take extraordinary steps of adding proofs of knowledge on the correct encryption of each individual bit [210] plus using public-key authentication framework [241]. While these early pioneering works are important in providing insights to achieve strong security, the systems they propose are in general too inefficient for applications. After Chapter 14, we will further study a series of subsequent works following the pioneering ones on probably strongly secure public-key cryptosystems and digital signature schemes. The cryptographic schemes proposed by these latter works propose have not only strong security, but also practical efficiency. They are indeed very good cryptographic schemes.

A cryptographic protocol is not only an algorithm, it is also a communication procedure which involves transmitting of messages over computer networks between different protocol participants under a set of agreed rules. So a protocol has a further dimension for efficiency measure: the number of communication interactions which are often called communication rounds. Usually a step of communication is regarded to be more costly than a step of local computation (typically an execution of a set of computer instructions, e.g. a multiplication of two numbers on a computing device). Therefore it is desirable that a cryptographic protocol should have few communication rounds. The standard efficiency criterion for declaring an algorithm as being efficient is if its running time is bounded by a small polynomial in the size of the problem. If we apply this efficiency criterion to a protocol, then an efficient protocol should have its number of communication rounds bounded by a polynomial of an *extremely* small degree: a constant (degree 0) or at most a linear (degree 1) function. A protocol with communication rounds exceeding a linear function should not be regarded as practically efficient, that is, no *good* for any practical use.

In §18.2.3 we will discuss some zero-knowledge proof protocols which have communication rounds measured by non-linear polynomials. We should note that those protocols were not proposed for real applications; instead, they have importance in the theory of cryptography and computational complexity. In Chapter 18 we will witness much research effort for designing practically efficient zero-knowledge protocols.

1.2.4 Use of Practical and Available Primitives and Services

A level of security which is good for one application needn't be good enough for another. Again, let us use our coin-flipping protocol as an example. In §1.2.1 we have agreed that, if implemented with the use of a practical one-way hash function, Protocol "Coin Flipping Over Telephone" is good enough for Alice and Bob to decide their recreation venue over the phone. However, in many cryptographic applications of a fair coin-flipping primitive, security services against cheating and/or for fairness are at much more stringent levels; in some applications the stringency must be in an absolute sense.

For example, in Chapter 18 we will discuss a zero-knowledge proof protocol which needs random bit string input and such random input must be mutually trusted by both proving/verification parties, or else serious damages will occur to one or both parties. In such zero-knowledge proof protocols, if the two communication parties do not have access to, or do not trust, a third-party-based service for supplying random numbers (such a service is usually nicknamed "random numbers from the sky" to imply its impracticality) then they have to generate their mutually trusted random numbers, bit-by-bit via a fair coin-flipping protocol. Notice that here the need for the randomness to be generated in a bit-by-bit (i.e., via fair coin-flipping) manner is in order to satisfy certain requirements, such as the correctness and zero-knowledge-ness of the protocol. In such a situation, a level of practically good (e.g., in the sense of using a practical hash function in Protocol "Coin Flipping Over Telephone") is most likely to be inadequate.

A challenging task in applied research on cryptography and cryptographic protocols is to build high quality security services from *practical* and *available* cryptographic primitives. Once more, let us use a coin-flipping protocol to make this point clear. The protocol is a remote coin-flipping protocol proposed by Blum [43]. Blum's protocol employs a *practically secure* and *easily implementable* "one-way" function but achieves a high-quality security in a *very strong* fashion which can be expressed as:

- First, it achieves a quantitative measure on the difficulty against the coin flipping party (e.g., Alice) for cheating, i.e., for preparing a pair of collision $x \neq y$ satisfying $f(x) = f(y)$. Here, the difficulty is quantified by that for factoring a large composite integer, i.e., that for solving a "pedigree" hard problem.
- Second, there is *absolutely no way* for the guessing party to have a guessing strategy biased away from the 50-50 chance. This is in terms of a complete security.

Thus, Blum's coin-flipping protocol is *particularly good* in the sense of having achieved a strong security while using only practical cryptographic primitives. As a strengthening and concrete realization for our first cryptographic protocol, we will describe Blum's coin-flipping protocol as the final cryptographic protocol of this book.

Several years after the discovery of public-key cryptography [97, 98, 246], it became gradually apparent that several basic and best-known public-key encryption algorithms (we will refer to them as "textbook crypto") generally have two kinds of weakness: (i) they leak partial information about the message encrypted; (ii) they are extremely vulnerable to active attacks (see Chapter 14). These weaknesses mean that "textbook crypto" are not fit for applications. Early approaches to a general fix for the weaknesses in "textbook crypto" invariantly apply bit-by-bit style of encryption and even apply zero-knowledge proof technique at bit-by-bit level as a means to prevent active attacks, plus authentication framework. These results, while valuable in the development of provably secure public-key encryption algorithms, are not suitable for most encryption applications since the need for zero-knowledge proof or for authentication framework is not practical for the case of encryption algorithms.

Since the successful initial work of using a randomized padding scheme in the strengthening of a public key encryption algorithm [24], a general approach emerges which strengthens popular textbook public-key encryption algorithms into ones with provable security by using popular primitives such as hash functions and pseudo-random number generators. These strengthened encryption schemes are practical since they use practical primitives such as hash functions, and consequently their efficiency is similar to the underlying "textbook crypto" counterparts. Due to this important quality element, some of these algorithms enhanced from using practical and popular primitives become public-key encryption and digital signature standards. We shall study several such schemes in Chapters 15 and 16.

Designing cryptographic schemes, protocols and security systems using available and popular techniques and primitives is also desirable in the sense that such results are more likely to be secure as they attract a wider interest for public scrutiny.

1.2.5 Explicitness

In the late 1960's, software systems grew very large and complex. Computer programmers began to experience a crisis, the so-called "software crisis." Large and complex software systems were getting more and more error prone, and the cost of debugging a program became far in excess of the cost of the program design and development. Soon computer scientists discovered a few perpetrators who helped to set-up the crisis which resulted from bad programming practice. Bad programming practice includes:

- Arbitrary use of the GOTO statement (jumping up and down seems very convenient)
- Abundant use of global variables (causing uncontrolled change of their values,

e.g., in an unexpected execution of a subroutine)

- The use of variables without declaration of their types (implicit types can be used in Fortran, so, for example, a real value may be truncated to an integer one without being noticed by the programmer)
- Unstructured and unorganized large chunk of codes for many tasks (can be thousands of lines a piece)
- Few commentary lines (since they don't execute!)

These were a few "convenient" things for a programmer to do, but had proved to be capable of causing great difficulties in program debugging, maintenance and further development. Software codes designed with these "convenient" features can be just too obscure to be comprehensible and maintained. Back then it was not uncommon that a programmer would not be able to to understand a piece of code s/he had written merely a couple of months or even weeks ago.

Once the disastrous consequences resulting from the bad programming practice were being gradually understood, *Program Design Methodology* became a subject of study in which *being explicit* became an important principle for programming. Being explicit includes limiting the use of GOTO and global variables (better not to use them at all), explicit (via mandatory) type declaration for any variables, which permits a compiler to check type flaws systematically and automatically, modularizing programming (dividing a large program into many smaller parts, each for one task), and using abundant (as clear as possible) commentary material which are texts inside a program and documentation outside.

A security system (cryptographic algorithm or protocol) includes program parts implemented in software and/or hardware, and in the case of protocol, the program parts run on a number of separate hosts (or a number of programs concurrently and interactively running on these hosts). The explicitness principle for software engineering applies to a security system's design by default (this is true in particular for protocols). However, because a security system is assumed to run in a hostile environment in which even a legitimate user may be malicious, a designer of such systems must also be explicit about many additional things. Here we list three important aspects to serve as general guidelines for security system designers and implementors. (In the rest of the book we will see many attacks on algorithms and protocols due to being implicit in design or specification of these systems.)

1. **Be explicit about all assumptions needed.**

   A security system operates by interacting with an environment and therefore it has a set of requirements which must be satisfied by that environment. These requirements are called assumptions (or premises) for a system to run. A violation of an assumption of a protocol may allow the possibility of exploiting an attack on the system and the consequence can be the nullification of some intended services. It is particularly difficult to notice a violation of an assumption which has not been clearly specified (a hidden assumption). Therefore all assumptions of a security system should be made explicit.

For example, it is quite common that a protocol has an implicit assumption or expectation that a computer host upon which the protocol runs can supply good random numbers, but in reality few desktop machines or hand-held devices are capable of satisfying this assumption. A so-called low-entropy attack is applicable to protocols using a poor random source. A widely publicized attack on an early implementation of the Secure Sockets Layer (SSL) Protocol (an authentication protocol for World Wide Web browser and server, see §12.5) is a well-known example of the low-entropy attack [123].

Explicit identification and specification of assumptions can also help the analysis of complex systems. DeMillo et al. (Chapter 4 of [91]), DeMillo and Merritt [92] suggest a two-step approach to cryptographic protocol design and analysis, which are listed below (after a modification by Moore [204, 205]):

i) Identify **all** assumptions made in the protocol.
ii) For each assumption in step (i), determine the effect on the security of the protocol if that assumption were violated.

2. **Be explicit about exact security services to be offered.**

A cryptographic algorithm/protocol provides certain security services. Examples of some important security services include: confidentiality (a message cannot be comprehended by a non-recipient), authentication (a message can be recognized to confirm its integrity or its origin), non-repudiation (impossibility for one to deny a connection to a message), proof of knowledge (demonstration of evidence without disclosing it), and commitment (e.g., a service offered to our first cryptographic protocol "Coin Flipping Over Telephone" in which Alice is forced to stick to a string without being able to change).

When designing a cryptographic protocol, the designer should be very clear regarding exactly what services the protocol intends to serve and should explicitly specify them as well. The explicit identification and specification will not only help the designer to choose correct cryptographic primitives or algorithms, but also help an implementor to correctly implement the protocol. Often, an identification of services to the refinement level of the general services given in these examples is not adequate, and further refinement of them is necessary. Here are a few possible ways to further refine some of them:

Confidentiality	$\Rightarrow$ privacy, anonymity, invisibility, indistinguishability
Authentication	$\Rightarrow$ data-origin, data-integrity, peer-entity
Non-repudiation	$\Rightarrow$ message-issuance, message-receipt

Proof of knowledge ⇒ knowledge possession, knowledge structure

A misidentification of services in a protocol design can cause misuse of cryptographic primitives, and the consequence can be a security flaw in the protocol. In Chapter 2 and Chapter 11 we will see disastrous examples of security flaws in authentication protocols due to misidentification of security services between confidentiality and authentication.

There can be many more kinds of security services with more ad hoc names (e.g., message freshness, non-malleability, forward secrecy, perfect zero-knowledge, fairness, binding, deniability, receipt freeness, and so on). These may be considered as derivatives or further refinement from the general services that we have listed earlier (a derivative can be in terms of negation, e.g., deniability is a negative derivative from non-repudiation). Nevertheless, explicit identification of them is often necessary in order to avoid design flaws.

3. **Be explicit about special cases in mathematics.**

As we have discussed in §1.2.2, some hard problems in computational complexity theory can provide a high confidence in the security of a cryptographic algorithm or protocol. However, often a hard problem has some special cases which are not hard at all. For example, we know that the problem of factorization of a large composite integer is in general very hard. However the factorization of a *large* composite integer $N = PQ$ where Q is the next prime number of a *large* prime number P is not a hard problem at all! One can do so efficiently by computing $\lfloor\sqrt{N}\rfloor$ ($\lfloor\cdot\rfloor$ is called the floor function and denotes the integer part of $\cdot$) and followed by a few trial divisions around that number to pinpoint P and Q.

Usual algebraic structures upon which cryptographic algorithms work (such as groups, rings and fields, to be studied in Chapter 5) contain special cases which produce exceptionally easy problems. Elements of small multiplicative orders (also defined in Chapter 5) in a multiplicative group or a finite field provide such an example; an extreme case of this is when the base for the Diffie-Hellman key exchange protocol (see §8.3) is the unity element in these algebraic structures. Weak cases of elliptic curves, e.g., "supersingular curves" and "anomalous curves," form another example. The discrete logarithm problem on "supersingular curves" can be reduced to the discrete logarithm problem on a finite field, known as the Menezes-Okamoto-Vanstone attack [197] (see §13.3.4.1). An "anomalous curve" is one with the number of points on it being equal to the size of the underlying field, which allows a polynomial time solution to the discrete logarithm problem on the curve, known as the attack of Satoh-Araki [252], Semaev [258] and Smart [278].

An easy special case, if not understood by an algorithm/protocol designer and/or not clearly specified in an algorithm/protocol specification, may easily go into an implementation and can thus be exploited by an attacker. So an algorithm/protocol designer must be aware of the special cases in mathematics, and should explicitly specify the procedures for the implementor to eliminate such cases.

It is not difficult to list many more items for explicitness (for example, a key-management protocol should stipulate explicitly the key-management rules, such as separation of keys for different usages, and the procedures for proper key disposal, etc.). Due to the specific nature of these items we cannot list all of them here. However, explicitness as a general principle for cryptographic algorithm/protocol design and specification will be frequently raised in the rest of the book. In general, the more explicitly an algorithm/protocol is designed and specified, the easier it is for the algorithm/protocol to be analyzed; therefore the more likely it is for the algorithm/protocol to be correctly implemented, and the less likely it is for the algorithm/protocol to suffer an unexpected attack.

1.2.6 Openness

Cryptography was once a preserve of governments. Military and diplomatic organizations used it to keep messages secret. In those days, most cryptographic research was conducted behind closed doors; algorithms and protocols were secrets. Indeed, governments did, and they still do, have a valid point in keeping their cryptographic research activities secret. Let us imagine that a government agency publishes a cipher. We should only consider the case that the cipher published is provably secure; otherwise the publication can be too dangerous and may actually end up causing embarrassment to the government. Then other governments may use the provably secure cipher and consequently undermine the effectiveness of the code-breakers of the government which published the cipher.

Nowadays, however, cryptographic mechanisms have been incorporated in a wide range of civilian systems (we have provided a non-exhaustive list of applications in the very beginning of this chapter). Cryptographic research for civilian use should take an open approach. Cryptographic algorithms do use secrets, but these secrets should be confined to the cryptographic keys or keying material (such as passwords or PINs); the algorithms themselves should be made public. Let's explore the reasons for this stipulation.

In any area of study, quality research depends on the open exchange of ideas via conference presentations and publications in scholarly journals. However, in the areas of cryptographic algorithms, protocols and security systems, open research is more than just a common means to acquire and advance knowledge. An important function of open research is public expert examination. Cryptographic algorithms, protocols and security systems have been notoriously error prone. Once a cryptographic research result is made public it can be examined by a large number of experts. Then the opportunity for finding errors (in design or maybe in security analysis) which may have been overlooked by the designers will be greatly increased. In contrast, if an algorithm is designed and developed in secret, then in order to keep the secret, only few, if any, experts can have access to and examine the details. As a result the chance for finding errors is decreased. A worse scenario can be that a designer may know an error and may exploit it secretly.

It is now an established principle that cryptographic algorithms, protocols, and security systems for civilian use must be made public, and must go through a lengthy public examination process. Peer review of a security system should be conducted by a hostile expert.

1.3 Chapter Summary

In this chapter we began with an easy example of applied cryptography. The three purposes served by the example are:

i) Showing the effectiveness of cryptography in problem solving

ii) Aiming for a fundamental understanding of cryptography

iii) Emphasizing the importance of non-textbook aspects of security

They form the main topics to be developed in the rest of this book.

We then conducted a series of discussions which served the purpose for an initial background and cultural introduction to the areas of study. Our discussions in these directions are by no means of complete. Several other authors have also conducted extensive study on principles, guidelines and culture for the areas of cryptography and information security. The following books form good further reading material: Schneier [254], Gollmann [129] and Anderson [14]. Schneier's monthly distributed "Crypto-Gram Newsletters" are also good reading material. To subscribe for receiving the newsletters, send an email to `schneier@counterpane.com`.

Exercises

1.1 What is the difference between a protocol and an algorithm?

1.2 In Prot 1.1 Alice can decide HEADS or TAILS. This may be an unfair advantage for some applications. Modify the protocol so that Alice can no longer have this advantage.

Hint: let a correct guess decide the side.

1.3 Let function f map from the space of 200-bit integers to that of 100-bit ones with the following mapping rule:

$$f(x) \overset{\text{def}}{=} \begin{array}{l}(\text{the most significant 100 bits of } x) \oplus \\ (\text{the least significant 100 bits of } x)\end{array}$$

here "$\oplus$" denotes bit-by-bit XOR operation, i.e.,

$$a \oplus b = \begin{cases} 0 & \text{if } a = b \\ 1 & \text{otherwise} \end{cases}$$

i) Is f efficient?

ii) Does f have the "Magic Property I"?

iii) Does f have the "Magic Property II"?

iv) Can this function be used in Prot 1.1?

1.4 Is an unbroken cryptographic algorithm more secure than a known broken one? If not, why?

1.5 Complex systems are error-prone. Give an additional reason for a complex security system to be even more error-prone.