1.6 Security

What does it mean for a cryptographic protocol to be "secure"? In the August 1977 issue of *Scientific American*, Martin Gardner wrote a column on the newly developed RSA public-key cryptosystem entitled "A new kind of cipher that would take millions of years to break" Included in the article was a challenge ciphertext, encrypted using a 512-bit RSA key. The challenge was solved 17 years later, on April 26, 1994 by factoring the given public key (the plaintext was "The Magic Words are Squeamish Ossifrage").

So what went wrong? The statement that the cipher would take millions of years to break probably referred to how long it would take to run the best factoring algorithm known at that time on the fastest computer available at that time. In the 17 intervening years, computers became much faster, and improved factoring algorithms were found. Furthermore, the computations performed to factor the key were not performed on a single comuputer; they were distributed among hundreds of computers around the world. And the amount of computation time required to factor the public key was in excess of 5000 MIPS-years, which is a lot of computing time! All of these factors led to the key being factored much sooner than anyone would have predicted in 1977.

This anecdote should suggest that any statements about cryptographic security should be scrutinized carefully, in particular, to ascertain what is being claimed and whether the claims are valid. Any discussion of cryptographic security requires a specification of an attack model, an adversarial goal, and a level of security. These terms are defined as follows:

attack model
: We will always assume that the adversary knows the protocol being used (this is called *Kerckhoff's Principle*) as well as the public key (if the system is a public-key system). We also assume that the adversary does not know any secret or private keys being used. Possible additional information provided to the adversary is specified in the attack model.

adversarial goal
: The adversarial goal specifies what it means to "break" the cryptosystem. In other words, what is the adversary attempting to do and what information is he trying to determine? How is the notion of a "successful attack" defined?

level of security
: The security level attempts to quantify the effort required to break the cryptosystem. Equivalently, what computational resources does the adversary have access to?

A statement of security for a cryptographic scheme will assert that a particular adversarial goal cannot be achieved in a specified attack model, given specified computational resources.

Here are some typical examples of attack models for cryptosystems:

key-only attack
The adversary is given the public key (if there is one).

known ciphertext attack
The adversary is given a string of ciphertext (all encrypted with the same unknown key).

known plaintext attack
The adversary is given a string of plaintext and the corresponding ciphertext.

chosen plaintext attack
The adversary chooses a string of plaintext and is then given the corresponding ciphertext.

chosen ciphertext attack
The adversary chooses a string of ciphertext and is then given the corresponding plaintext

We can make a couple of simple observations about the different attack models enumerated above. First, a key-only attack would not be relevant in the situation of a secret-key cryptosystem, because a secret-key cryptosystem does not have a public key. Furthermore, in the case of a public-key cryptosystem, a key-only attack is equivalent to a chosen plaintext attack, because the adversary can always encrypt any chosen plaintext using the public key.

The following are examples of possible adversarial goals in the case of a secret- or public-key cryptosystem:

complete break
The adversary determines the value of the private (or secret) key.

decryption of previously unseen ciphertexts
The adversary can decrypt a previously unseen ciphertext with some specified non-zero probability. In the context of a public-key cryptosystem, the scheme is said to provide *one-way encryption* if a previously unseen ciphertext cannot be decrypted, except with very small probability.

partial information
The adversary can determine some specific type of partial information about the plaintext, given a previously unseen ciphertext, with some specified non-zero probability.

distinguishability
This means that the adversary can distinguish between encryptions of two given plaintexts (*semantic security* means that the adversary cannot do this).

Again, a couple of comments are in order. Clearly, determining a private or public key means that the cryptosystem is irretrievably broken. The next notions include some types of less catastrophic attacks. For example, it might be possible decrypt 1% of all possible ciphertexts, even if the key is not known; or it might be possible to determine the least significant bit of the plaintext without decrypting the ciphertext.

The concept of semantic security is a very strong notion. It means that, when the adversary is given two possible plaintexts, say x_1 and x_2, and a ciphertext y which is an encryption of one of x_1 or x_2, he should not be able to guess which plaintext was encrypted with a probability significantly greater than $1/2$. In the case of a public-key cryptosystem, this property requires a randomized encryption operation: if the encryption operation is deterministic, then the adversary can just encrypt x_1 and x_2 using the public key, and see which one yields the ciphertext, y.

Security levels studied in cryptography include the following:

computational security
: This means that a specific algorithm to break the system is computationally infeasible. For example, an exahustive key search (trying every possible key, until the right one is found) is considered to be infeasible if the number of keys is large enough.

provable security
: This refers to a situation where breaking the cryptosystem can be reduced in a complexity-theoretic sense to solving some underlying (assumed difficult) mathematical problem, or a "simpler" cryptographic protocol.
For example, if we could prove that breaking a cryptosystem implies that a large composite number n can be factored, then we would have provable security. Provable security only establishes the security of a cryptosystem relative to some other problem, however. It does not give an absolute guarantee of security unless the relevant problem has a proof of infeasibility. Usually this is not the case; in the example mentioned above, we would know only that breaking the cryptosystem cannot be done without factoring n; we do not know for sure how difficult the factoring problem is because there is no proof known at the present tome that factoring is a difficult problem.

unconditional security
: This means that the cryptosystem cannot be broken, even with unlimited computational resources, because there is not enough information available to the adversary for him to be able to do so. Unconditional security is not generally achievable in public-key systems because the adversary can always try every possible key (given enough time). However, unconditional security is achievable for various types of secret key systems.

As we said earlier, a statement of security must specify the attack model, adversarial goal and the level of security in order to be meaningful. We illustrate with a few examples.

Fist, we look at the classical *One-time pad*.

Cryptosystem 1.8: *One-time pad*

Let $n \geq 1$ be an integer, and take $\mathcal{P} = \mathcal{C} = \mathcal{K} = (\mathbb{Z}_2)^n$. For $K \in (\mathbb{Z}_2)^n$, define $e_K(x)$ to be the vector sum modulo 2 of K and x (or, equivalently, the exclusive-or of the two associated bitstrings). So, if $x = (x_1, \ldots, x_n)$ and $K = (K_1, \ldots, K_n)$, then

$$e_K(x) = (x_1 + K_1, \ldots, x_n + K_n) \bmod 2.$$

Decryption is identical to encryption. If $y = (y_1, \ldots, y_n)$, then

$$d_K(y) = (y_1 + K_1, \ldots, y_n + K_n) \bmod 2.$$

Shannon proved that the *One-time pad* is unconditionally semantically secure in a known-ciphertext attack if the key is used only for a single encryption. Here, the attack model is that the adversary is given a single piece of ciphertext, y, as well as two possible plaintexts, x_1 and x_2. These are all bitstrings of a specified length. The adversary is supposed to determine whether y is an encryption of x_1 or x_2. This is clearly an impossible task, because either situation is possible under the given information. Namely, y is the encryption of x_i ($i = 1, 2$) if and only if $K = y \oplus x_i$. There is no way for the adversary to achieve his goal, regardless of how much computer time he uses. Therefore we achieve unconditional security against the specified attack.

Next, we provide an example of a provably secure scheme; we show that the *Rabin cryptosystem* achieves provable one-way encryption against a chosen-plaintext attack, assuming that factoring is infeasible.

The proof is a proof by contradiction. We assume that the adversary can find a decryption algorithm, say RABINDECRYPT. RABINDECRYPT is any hypothetical algorithm (or oracle) that performs Rabin decryption, returning one of the four possible plaintexts corresponding to a given valid ciphertext. Then, we will prove that RABINDECRYPT can be incorporated into a randomized algorithm, called RABINORACLEFACTORING, that factors the modulus, n, with probability at least $1/2$. (On average, n will be factored after two runs of the algorithm.) Becasue we are assuming that the adversary cannot factor n, we have a contradiction. This contradiction implies that the adversary cannot decrypt Rabin ciphertexts.

Stated more succinctly, the proof technique can be summarized as follows: if the adversary can decrypt Rabin ciphertexts, then he can factor n. Therefore, if it is infeasible to factor n, it must be infeasible for the adversary to decrpyt Rabin ciphertexts.

Here is the factoring algorithm:

Algorithm 1.9: RABINORACLEFACTORING(n)

external RABINDECRYPT
choose a random integer $r \in {\mathbb{Z}_n}^*$
$y \leftarrow r^2 \bmod n$
$x \leftarrow$ RABINDECRYPT(y)
if $x \equiv \pm r \pmod{n}$
  then return ("failure")
  else $\begin{cases} p \leftarrow \gcd(x+r, n) \\ q \leftarrow n/p \\ \textbf{return } (\text{"}n = p \times q\text{"}) \end{cases}$

Here is what is happening in this algorithm. First, a random r is encrypted to yield a ciphertext, y. Then RABINDECRYPT is consulted, and it supplies a plaintext x that is a decryption of y. Therefore it holds that $x^2 \equiv r^2 \pmod{n}$, and hence

$$n \mid (x+r)(x-r).$$

There are four plaintexts whose encryption yields the ciphertext y. Two of these are r and $-r \bmod n$. The probability that $x \neq r$ and $x \neq -r \bmod n$ is $1/2$. In this case, we have that

$$\begin{aligned} (x+r)(x-r) &\equiv 0 \pmod{pq} \\ x + r &\not\equiv 0 \pmod{pq} \\ x - r &\not\equiv 0 \pmod{pq} \end{aligned}$$

We conclude that exactly one of p and q divide $x+r$. Therefore, $\gcd(x+r, n)$, which can easily be computed using the EUCLIDEAN ALGORITHM, yields one of the two prime factors of n. Dividing this factor into n provides the second prime factor of n.

The discovery of an attack on a cryptosystem shows that a certain security goal is not achieved. For example, it is easy to see that the *RSA cryptosystem* does not provide computationally secure one-way encryption in the chosen-ciphertext model. We present a simple attack based on the following so-called "homomorphic property" of the RSA encryption function: Let $y_1, y_2 \in \mathbb{Z}_n$, and suppose

$$y_3 \equiv y_1 y_2 \pmod{n}.$$

Then it is easy to see that

$$d_K(y_1) d_K(y_2) \equiv d_K(y_3) \pmod{n}.$$

Here is the attack. The adversary, Oscar, is given a ciphertext, say y, and is asked to decrypt it. Oscar chooses $y_1, y_2 \in \mathbb{Z}_n$ such that $y \neq y_1, y_2$ and

$$y \equiv y_1 y_2 \pmod{n}.$$

Then, Oscar requests the values $x_1 = d_K(y_1)$ and $x_2 = d_K(y_2)$ (he is allowed to request decryptions of any ciphertext other than y in the chosen-ciphertext attack model). Finally, Oscar easily computes

$$x = x_1 x_2 \bmod n,$$

which is the decryption of y.

1.6.1 What Do Proofs of Security Mean?

A statement of security for a cryptographic scheme will assert that a particular adversarial goal cannot be achieved in a specified attack model, given that the adversary possesses specified computational resources. Whether a proof of security has any meaning in the "real world" depends on several factors:

1. Does the attack model specify all the information available to the adversary, as well as his computational resources? If the adversary has access to additional information, then all bets are off.
2. Can the adversary do "bad things" even if he cannot achieve the specified adversarial goal?
3. How realistic are the underlying assumptions in the attack model? (Note that this includes explicit, as well as "hidden" assumptions.)

Clearly, in order for a security proof to be significant/valid/realistic, we should try to prove that a *weak* adversarial goal cannot be attained in a *strong* attack model, and we should minimize the assumptions required, to the greatest extent possible.

Let's consider various types of assumptions, both implicit and explicit, that underly security proofs. These can be regarded as additional aspects of the attack model, in most cases.

black-box model

Cryptographic schemes are most often modelled as "black boxes". Their input-output behaviour is described precisely, but details of their implementation (in software or hardware) are not taken into consideration in security proofs in this model. Attacks in a black box model will work for any implementation of the scheme in question. On the other hand, a security proof in the black box model may not be valid if details of the implementation are known to the adversary. Side channel attacks provide illustrations of this; see Section 1.6.3.

intractibility assumptions

The presumed difficulty of certain computational problems are often used in "provable security" results for public-key schemes. Commonly used intractibility assumptions are based on the apparent difficulty of factoring large integers, or of solving the discrete logarithm problem in certain large abelian groups.

random oracle model

Many cryptographic schemes use hash functions (in practice, *SHA-1* is the most common). Many security proofs assume that the outputs of a hash function are truly random and unpredictable; this is the so-called *random oracle model*. A proof in the random oracle model presemes that the adversary cannot use any information about the actual hash function being used in the scheme in his attack. Stated another way, a security proof in the random oracle model shows that the adversary cannot achieve his goal without exploiting weaknesses of the hash function used in the scheme.

random number generators

Many randomized cryptographic schemes require the generation of random numbers. Most security proofs of such schemes assume implicitly that there is some ideal random number generator that outputs truly random numbers for use in the scheme. It would be better and more accurate to include this as an explicit assumption in the attack model. In analogy with the random oracle model, we might call this the *perfect random number generator model*. In practice, any scheme requiring random numbers will make use of a certain pseudorandom number generator. A security proof in the perfect random number generator model would show that the adversary cannot achieve his goal without exploiting weaknesses of the particular pseudorandom number generator used in the scheme.

The assumptions discussed above often arise in the context of "provable security". We have provided one example of a provably secure cryptosystem, namely, the *Rabin cryptosystem*. The proof of security for that scheme depended on the assumption that factoring the modulus, n, is infeasible. However, provable security in cryptography can take many other forms. Here are a few typical scenarios in which provable security results have been obtained.

1. Proving that cryptographic schemes are secure, provided that an underlying computational problem is intractable.
2. Proving that a secure basic scheme can be used to build a secure general scheme with increased functionality. For example, there is a well-known technique which can be used to build a secure hash function (with infinite domain) from a secure compression function (with finite domain).
3. Proving that a basic scheme providing a minimal level of security can be used to build a scheme with a higher level of security. For example, if the basic *RSA cryptosystem* is secure against decryption of previously unseen ciphertexts, then a certain enhanced scheme can be proven to be semantically secure.
4. Proving that a cryptographic protocol is secure if the underlying crypto-

graphic tools used in the protocol are secure. We will consider many example of these types of proofs in the rest of the book.

Finally, let's consider briefly the computational models that we have introduced. Unconditional security is a very desirable property precisely because no intractability assumptions are required. Unfortunately, unconditional security has some limitations in terms of its applicability: most public-key schemes cannot be unconditionally secure because there is always the attack which tests every possible private key, and confrims whether or not it is valid by using a certain public-key operation, in order to break the scheme. Unconditionally secure schemes can be designed in the secret-key context, however, as well as in situations where secret information is distributed among two or more people (we'll see examples of this later in the book). The other drawback of unconditionally secure schemes in general it that they tend to require a large amount of secret information, and/or they can be used for only a specified (small) number of operations with a given key.

We have already discussed the fact that provably secure schemes depend on assumptions which are usually not proven. Thus they provide proofs of security relative to something else. In general, provable security results are not unconditional.

Results concerning computational security usually are too specific to be of much use. For example, one can say plausibly that a secret key cryptosystem is secure against an exhaustive key search if the number of possible keys is 2^{128}, say. However, statements about computational security usually apply to very specific attacks, and cannot even begin to take into consideration the wide variety of potential attacks — this is why provable security can be of much more value.

1.6.2 Security of Signature Schemes and MACs

The adversarial goals when attacking a signature scheme or MAC relate to creating valid signatures or MACs for messages (as mentioned earlier, these are called forgeries).

In the case of signature schemes, the following types of attack models are commonly considered:

key-only attack

The adversary possesses Alice's public key, i.e., the verification function, ver_K.

known message attack

The adversary possesses a list of messages previously signed by Alice, say

$$(x_1, y_1), (x_2, y_2), \ldots,$$

where the x_i's are messages and the y_i's are Alice's signatures on these messages (so $y_i = \mathsf{sig}_K(x_i)$, $i = 1, 2, \ldots$).

chosen message attack

The adversary requests Alice's signatures on a list of messages. Therefore he chooses messages $x_1, x_2, \ldots$, and Alice supplies her signatures on these messages, namely, $y_i = \mathsf{sig}_K(x_i)$, $i = 1, 2, \ldots$.

There are several "reasonable" adversarial goals:

total break

The adversary is able to determine Alice's private key, i.e., the signing function sig_K. Therefore he can create valid signatures on any message.

selective forgery

With some non-negligible probability, the adversary is able to create a valid signature on a (random) message chosen by someone else. In other words, if the adversary is given a message x, then he can determine (with some probability) the signature y such that $\mathsf{ver}_K(x, y) = \mathrm{true}$. The message x should not be one that has previously been signed by Alice.

existential forgery

The adversary is able to create a valid signature for at least one message. In other words, the adversary can create a pair (x, y) where x is a message and $\mathsf{ver}_K(x, y) = \mathrm{true}$. The message x should not be one that has previously been signed by Alice.

We just present one brief example to illustrate a certain trivial attack on the *RSA signature scheme*. Suppose that this scheme is used without a hash function. It is a simple matter for an adversary to choose an arbitrary $y \in \mathbb{Z}_n$ and then compute $x = y^b \bmod n$. Clearly y is a valid signature on the message x, so this is a valid forgery. The attack is an existential forgery in the key-only model.

This attack is prevented by using a preimage resistant hash function.

1.6.3 Side Channel Attacks

The previous analyses treated the cryptosystems as "black boxes". Their input-output behaviour is specified (in terms of an unknown key) but no details about the implementations of the algorithms are specified. This is called, naturally enough, a *black box* model. In some scenarios, certain aspects of an implementation of the algorithms (either in hardware or in software) may be known. Often this is called a *grey box model*. Finally, if the software is run in a completely hostile environment, where all details of the computations can be monitored, this is called a *white box model*.

A proof of security in the black box model does not mean that a particular implementation of a cryptosystem is secure. *Side channel attacks* refer to attacks on particular implementations of cryptosystems in a grey box model. Here are some examples:

timing analysis

sometimes permits cryptosystems to be attacked by measuring the time required to perform encryptions or decryptions.

power analysis
sometimes permits cryptosystems implemented on smart cards to be attacked by measuring the power consumption during encryption or decryption

fault analysis
can be used to break cryptosystems if the implementation can be induced to produce incorrect decryptions in some predictable way.