Cryptography

Jonathan Katz, University of Maryland, College Park, MD 20742.

1 Introduction

Cryptography is a vast subject, addressing problems as diverse as e-cash, remote authentication, fault-tolerant distributed computing, and more. We cannot hope to give a comprehensive account of the field here. Instead, we will narrow our focus to those aspects of cryptography most relevant to the problem of *secure communication*. Broadly speaking, secure communication encompasses two complementary goals: the **secrecy** and **integrity** of communicated data. These terms can be illustrated using the simple example of a user A attempting to transmit a message m to a user B over a public channel. In the simplest sense, techniques for data secrecy ensure that an eavesdropping adversary (i.e., an adversary who sees all communication occurring on the channel) cannot learn any information about the underlying message m. Viewed in this way, such techniques protect against a *passive* adversary who listens to — but does not otherwise interfere with — the parties' communication. Techniques for data integrity, on the other hand, protect against an *active* adversary who may arbitrarily modify the information sent over the channel or may inject new messages of his own. Security in this setting requires that any such modifications or insertions performed by the adversary will be detected by the receiving party.

In the cases of both secrecy and integrity, two different assumptions regarding the initial set-up of the communicating parties can be considered. In the **private-key setting** (also known as the "shared-key," "secret-key," or "symmetric-key" setting), which was the setting used exclusively for cryptography until the mid-1970s, parties A and B are assumed to have shared some secret information — a **key** — in advance. This key, which is completely unknown to the adversary, is then used to secure their future communication. (We do not comment further on how such a key might be generated and shared; for our purposes, it is simply an assumption of the model.) Techniques for secrecy in this setting are called **private-key encryption** schemes, and those for data integrity are termed **message authentication codes (MACs)**.

In the **public-key setting**, one (or both) of the parties generates a pair of keys: a *public key* that is widely disseminated, and an associated *private key* which is kept secret. The party generating these keys may now use them as a receiver to ensure message secrecy using a **public-key encryption** scheme, or as a sender to provide data integrity using a **digital signature scheme**. Table 1 gives an overview of these different cryptographic tools.

In addition to showcasing the above primitives, and explaining how they should be used and how they can be constructed, the treatment here will also introduce a bit of the methodology of modern (i.e., post-1980s) cryptography. This includes an emphasis on formal *definitions* of security that pin down exactly what goals a scheme is intended to achieve; precisely stated *assumptions* (if needed) regarding the hardness of certain mathematical problems; and rigorous *proofs* of security that a cryptographic construction meets some definition of security given a particular assumption. This

	Private-key setting	**Public-key setting**
Secrecy	Private-key encryption (Section 2)	Public-key encryption (Section 4)
Integrity	Message authentication codes (Section 3)	Digital signature schemes (Section 5)

Table 1: Overview of the topics covered in this survey.

approach to designing and analyzing cryptosystems is much preferred to the heuristic, "ad-hoc" approach used in the past.

We warn the reader in advance that it is not the intention of this survey to cover the precise details of schemes used in practice today, nor will the survey be comprehensive. Rather, the aim of the survey is to provide the reader with an appreciation for the problem of secure communication along with an explanation of the core techniques used to address it. The reader seeking further details is advised to consult the references listed at the end of this chapter.

2 The Private-Key Setting and Private-Key Encryption

We begin by discussing the private-key setting, where two parties share a random, secret key k that will be used to secure their future communication. Let us jump right in by defining the syntax of private-key encryption. Formally, a private-key encryption scheme consists of a pair of algorithms $(\mathsf{Enc}, \mathsf{Dec})$. The encryption algorithm Enc takes as input a key k and a message m (sometimes also called the **plaintext**), and outputs an encrypted version of the message called the **ciphertext** that will be denoted by c. The decryption algorithm Dec takes as input a key and a ciphertext, and outputs a message. The basic correctness requirement is that for every key k and message m (in some allowed set of messages) we have $m = \mathsf{Dec}_k(\mathsf{Enc}_k(m))$. Since it will come up later, we mention here that Enc may be a *randomized* algorithm, so that multiple ciphertexts can potentially be output when encrypting a given message with some key, in which case the preceding correctness condition is required to hold with probability 1. Without loss of generality, we may assume Dec is deterministic.

A private-key encryption scheme is used in the following way. Let k be the key shared by two parties A and B. For A to send a message m to B, she first **encrypts** the message by computing $c \leftarrow \mathsf{Enc}_k(m)$ (we use "$\leftarrow$" to explicitly indicate that Enc may be randomized); the ciphertext c is then transmitted over the public channel to B. Upon receiving this ciphertext, B **decrypts** the ciphertext to recover the message by computing $m := \mathsf{Dec}_k(c)$.

A private-key encryption scheme can also be used by a single party A to encrypt data (say, on a hard drive) that will be accessed by A herself at a later point in time. Here, A encrypts the data using a key k that she stores securely somewhere else. When A later wants to read the data, she can recover it by decrypting using k. Here, the "channel" is the hard drive itself and, rather than being separated in *space*, encryption and decryption are now separated in *time*. Everything we say in what follows will apply to either usage of private-key encryption. (Message authentication codes, discussed in Section 3, can also be used for either of these canonical applications.)

The above discussion says nothing about the security provided by the encryption scheme. We consider this aspect in the following sections.

2.1 Perfect Secrecy

The goal of a private-key encryption scheme is to ensure the secrecy of m from an eavesdropping adversary $\mathcal{A}$ who views c, but does not know k. How should secrecy be defined formally?

A natural first attempt would be to say that an encryption scheme is secure if $\mathcal{A}$ cannot recover the message m in its entirety (assuming, say, m is chosen uniformly). A little thought shows that such a definition is inadequate: What if the distribution of m is not uniform? And surely we would not consider secure a scheme that always leaks the first bit of the message (without revealing anything about the rest of m)!

A better definition, introduced by Shannon [38] and termed *perfect secrecy*, is that the ciphertext c should leak no information whatsoever about m, regardless of its distribution. This is formalized by requiring that the *a posteriori* distribution of m (after $\mathcal{A}$ observes the ciphertext) should be equivalent to the *a priori* distribution of m (reflecting the adversary's prior knowledge about the distribution of the message). Namely, a scheme $(\mathsf{Enc}, \mathsf{Dec})$ is perfectly secret if for any distribution $\mathcal{M}$ over the space of possible messages, any message m, and any ciphertext c, it holds that

$$\Pr[M = m \mid C = c] = \Pr[M = m],$$

where M (resp., C) denotes the random variable taking the value of the actual message of the sender (resp., actual ciphertext transmitted).

Before giving an example of a scheme satisfying the above definition, we stress that the adversary (implicit in the above) is assumed to know the full details of the encryption scheme being used by the honest parties. (This is called *Kerckhoffs's principle* [29].) It is a mistake to require the details of a cryptosystem scheme to be hidden in order for it to be secure, and modern schemes are designed to be secure even when the full details of all algorithms are publicly known. The only thing unknown to the adversary is the key itself. This highlights the necessity of choosing the key at random, and the importance of keeping it completely secret.

Perfect secrecy can be achieved by the **one-time pad** encryption scheme, which works as follows. Let ℓ be the length of the message to be transmitted, where the message is viewed as a binary string. (Note that all potential messages are assumed to have the same length.) The parties share in advance a uniformly distributed, ℓ-bit key $k \in \{0,1\}^\ell$. To encrypt message m the sender computes $c := m \oplus k$, where $\oplus$ represents bit-wise exclusive-or. Decryption is performed by setting $m := c \oplus k$. Clearly, decryption always recovers the original message.

To see that this scheme is perfectly secret, fix any initial distribution over messages and let K be the random variable denoting the key. For any message m and observed ciphertext c, we have

$$\begin{aligned}&\Pr[M = m \mid C = c] = \\ &\qquad \frac{\Pr[C = c \mid M = m]\Pr[M = m]}{\Pr[C = c]} = \frac{\Pr[C = c \mid M = m]\Pr[M = m]}{\sum_{m'} \Pr[C = c \mid M = m'] \cdot \Pr[M = m']},\end{aligned} \tag{1}$$

where the summation is over all possible messages m'. Moreover, for any c, m' (including $m' = m$) we have

$$\Pr[C = c \mid M = m'] = \Pr[K = c \oplus m'] = 2^{-\ell},$$

since k is a uniform ℓ-bit string. Substituting into (1), we have $\Pr[M = m \mid C = c] = \Pr[M = m]$ as desired.

Although the one-time pad is perfectly secret, it is of limited value in practice. For one, *the length of the shared key is equal to the length of the message.* Thus, the scheme becomes impractical when long messages are to be sent. Second, it is easy to see that the scheme provides secrecy *only when a given key is used to encrypt a single message* (hence the name "one-time pad"). This will not do in typical scenarios where A and B wish to share a single key that they can use to send multiple messages. Unfortunately, it can be shown that both these limitations are inherent for schemes achieving perfect secrecy.

2.2 Computational Secrecy

At the end of the previous section we observed some fundamental limitations of perfect secrecy. To obtain reasonable solutions, we thus need to (slightly) relax our definition of secrecy. This is not too bad, however, since perfect secrecy may be considered to be unnecessarily strong: it requires *absolutely no information* about m to be leaked, even to an *all-powerful* eavesdropper. Arguably, it would be sufficient to leak a *tiny amount of information* and to restrict attention to eavesdroppers having some *bounded amount of computing power.* To take some concrete numbers, we may be satisfied with an encryption scheme that leaks a most 2^{-60} bits of information (on average) to any eavesdropper that invests at most 100 years of computational effort (on a standard desktop PC, for instance). Definitions of this latter sort are termed **computational**, to distinguish them from notions (like perfect secrecy) that are **information-theoretic** in nature. As initiated in the work of [11, 36, 33, 35, 44, 9, 22] and others, computational security is now the default way security is defined for cryptographic primitives.

A drawback of computational notions of security is that, given the current state of our knowledge, proofs that a given scheme satisfies any such definition must necessarily rely on (unproven) assumptions regarding the computational hardness of certain problems. Our confidence in the security of a particular scheme can be no better than our confidence in the underlying assumption(s) it is based on.

We illustrate the above by introducing a basic definition of computational secrecy. Fix some $t, \varepsilon \geq 0$. Private-key encryption scheme $(\mathsf{Enc}, \mathsf{Dec})$ is (t, ε)-*indistinguishable* if for every eavesdropper $\mathcal{A}$ running in time at most t, and for all (equal-length) messages m_0, m_1, we have

$$\left|\Pr\left[\mathcal{A}(\mathsf{Enc}_k(m_b)) = b\right] - \frac{1}{2}\right| \leq \varepsilon.$$

The probability above is taken over uniform choice of the key k and the bit b, as well as any randomness used by Enc. In words: we choose a random key k, encrypt one of m_0 or m_1 (each with equal probability), and give the resulting ciphertext to $\mathcal{A}$; the scheme is indistinguishable if any $\mathcal{A}$ running in time t cannot determine which message was encrypted with probability ε-better than a random guess.

Perfect secrecy can be shown to be equivalent to $(\infty, 0)$-indistinguishability, and so the above is a natural relaxation of perfect secrecy. Taking t bounded and ε strictly positive exactly corresponds to our intuitive idea of relaxing perfect secrecy by placing a bound t on the running time of $\mathcal{A}$, and being content with possibly allowing a tiny amount ε of "information leakage."

2.3 Security Against Chosen-Plaintext Attacks

Rather than give a construction satisfying the above definition, we immediately introduce an even stronger definition that is better suited for practical applications. Thus far, our security definitions have been restricted to consideration of an adversary who eavesdrops on a *single* ciphertext and, as we have mentioned, this restriction is essential in the context of perfect secrecy. As noted earlier, however, the honest parties would prefer to encrypt multiple messages using the same shared key; we would like to guarantee secrecy in this setting as well. Moreover, it may be the case that the adversary already knows some of the messages being encrypted; in fact, the adversary might even be able to influence some of the messages the parties send. This latter scenario, where the adversary can cause the parties to encrypt plaintexts of the adversary's choice, is termed a **chosen-plaintext attack**. The one-time pad is trivially insecure against a chosen-plaintext attack: given a ciphertext $c = k \oplus m$ for a known message m, the adversary can easily recover the key k.

To capture the above attack scenarios, we modify the previous definition by additionally giving $\mathcal{A}$ access to an *encryption oracle* $\mathsf{Enc}_k(\cdot)$. This oracle allows the adversary to obtain the encryption of any message(s) of its choice using the key k shared by the parties. This oracle is meant to model the real-world capabilities of an adversary who can control what messages get encrypted by the parties; of course, if the adversary has only partial control over what messages get encrypted, then this only makes the adversary weaker. We say that an encryption scheme $(\mathsf{Enc}, \mathsf{Dec})$ is (t, ε)-*indistinguishable against a chosen-plaintext attack* (or (t, ε)-*CPA secure*) if for every adversary $\mathcal{A}$ running in time at most t

$$\left| \Pr\left[\mathcal{A}^{\mathsf{Enc}_k(\cdot)}(\mathsf{Enc}_k(m_b)) = b \right] - \frac{1}{2} \right| \leq \varepsilon,$$

where the probability space is as before.

It is not entirely obvious that CPA-security implies security when multiple messages are encrypted, but this can be shown to be the case [27].

Any easy, but important, observation is that any CPA-secure encryption scheme must have randomized encryption. This is true even if only security for multiple encrypted messages is desired: if the encryption scheme is deterministic, then given any two ciphertexts the adversary can tell whether or not they are encryptions of the same message, an undesirable leakage of information.

2.4 Block Ciphers

As a step toward constructing a CPA-secure private-key encryption scheme, we first introduce an important primitive for this application. A *keyed function* $F : \{0,1\}^n \times \{0,1\}^\ell \to \{0,1\}^\ell$ is an efficiently computable function that maps two inputs to a single output; we treat the first input to F as a key that will be chosen at random and then fixed, and define $F_k(x) = F(k, x)$. (We have assumed for simplicity that the input and output of F_k have the same length ℓ, but this is not essential.) We call n the *key length* and ℓ the *block length.* Informally, F is pseudorandom if the function F_k, for a randomly chosen k, is "indistinguishable" from a completely random function f with the same domain and range. That is, consider an adversary $\mathcal{A}$ who can send inputs to and receive outputs from a box that contains either F_k (for a random $k \in \{0,1\}^n$) or f (for a random function f); the keyed function F is pseudorandom if $\mathcal{A}$ cannot tell which is the case with probability significantly better than random guessing. Formally, F is a (t, ε)-**pseudorandom function** [20] if for any adversary $\mathcal{A}$ running in time t

$$\left| \Pr[\mathcal{A}^{F_k(\cdot)} = 1] - \Pr[\mathcal{A}^{f(\cdot)} = 1] \right| \leq \varepsilon,$$

where in the first case k is chosen uniformly from $\{0,1\}^n$ and in the second case f is a completely random function.

If, for all k, the function F_k is a permutation (i.e., bijection) over $\{0,1\}^\ell$ and moreover the inverse F_k^{-1} can be computed efficiently (given the key k), then F is called a keyed *permutation*. A pseudorandom function that is also a keyed permutation is called a pseudorandom permutation or a **block cipher**. (Technically, in this case F_k should be indistinguishable from a random permutation, but for large enough ℓ this is equivalent to being indistinguishable from a random function.)

A long sequence of theoretical results culminating in [20, 24, 31] shows that pseudorandom functions and block ciphers can be constructed from rather minimal cryptographic assumptions, and thus in particular from the cryptographic assumptions we will introduce in Section 4. Such constructions are rather inefficient. In practice, dedicated and highly efficient block ciphers are used instead; although the security of these block ciphers cannot be cleanly reduced to a concise mathematical assumption, several design principles used in their construction can be given theoretical justification. More importantly, such block ciphers have been subjected to intense scrutiny by the cryptographic community for several years, and thus it is not unreasonable to view the assumption that these block ciphers are secure as being on par with other assumptions used in cryptography.

The most popular block cipher today is the Advanced Encryption Standard (AES) [10], which was standardized by NIST in 2001 after a multi-year, public competition. AES supports 128-, 192-, or 256-bit keys, and has a 128-bit block length. It superseded the Data Encryption Standard (DES) [13], which had been standardized by the US government in 1977. DES is still in wide use, but is considered insecure due to its relatively short key length (56 bits) and block length (64 bits). Further details and additional block ciphers are discussed in [30].

2.5 A CPA-Secure Scheme

Given any pseudorandom function $F : \{0,1\}^n \times \{0,1\}^\ell \to \{0,1\}^\ell$ with ℓ sufficiently long, it is possible to construct a private-key encryption scheme $(\mathsf{Enc}, \mathsf{Dec})$ that is CPA-secure [21]. To encrypt a message $m \in \{0,1\}^\ell$ using a key k, first choose a random string $r \in \{0,1\}^\ell$; then output the ciphertext $\langle r, F_k(r) \oplus m \rangle$. Note that encryption here is randomized, and there are many possible ciphertexts associated with a given key and message. Decryption of a ciphertext $c = \langle c_1, c_2 \rangle$ using key k is performed by computing $m := c_2 \oplus F_k(c_1)$. It can be verified that correctness holds.

Intuitively, the sender and receiver are using $F_k(r)$ as a "one-time pad" to encrypt m. Although $F_k(r)$ is not random, it *is* pseudorandom and one can show that this is enough for security to hold. In a bit more detail: say the sender encrypts multiple messages $m_1, m_2, \ldots, m_q$ using random strings $r_1, r_2, \ldots, r_q$, respectively. The facts that F is a pseudorandom function and k is unknown to the adversary imply that $F_k(r_1), F_k(r_2), \ldots, F_k(r_q)$ are indistinguishable from independent, uniform strings of length ℓ unless $r_i = r_j$ for some $i \neq j$ (in which case $F_k(r_i)$ and $F_k(r_j)$ are, of course, equal). Assuming this does not occur, then, usage of the encryption scheme is equivalent to using the one-time pad encryption scheme with q *independent* keys, and is thus secure. The probability that there exist distinct i, j with $r_i = r_j$ can be bounded by $q^2/2^\ell$, which is small for typical values of q, ℓ. A full proof can be found in [27, Chapter 3].

The scheme described above can be applied to messages of arbitrary length by encrypting in a block-by-block fashion. This results in a ciphertext whose length is twice that of the original plaintext. More efficient **modes of encryption** [14, 27] are used in practice to encrypt long messages. As an example, counter mode (CTR-mode) encryption of a message $m = \langle m_1, \ldots, m_t \rangle$ (with $m_i \in \{0,1\}^\ell$) using a key k is done by choosing a random r as above, and then computing the ciphertext

$$\langle r,\ F_k(r+1) \oplus m_1,\ F_k(r+2) \oplus m_2,\ \ldots,\ F_k(r+t) \oplus m_t \rangle.$$

(A proof of security for this scheme follows along similar lines as above.) The ciphertext is now only a single block longer than the plaintext.

2.6 Stronger Security Notions

Even the notion of CPA-security considers only a passive adversary who eavesdrops on the public channel, but not an active adversary who interferes with the communication between the parties. (Although we will treat active attacks in the next section, there our concern will primarily be integrity rather than secrecy.) Encryption schemes providing security against an active adversary are available; see Section 3.2 and [27, Section 3.7] for further discussion.

3 Message Authentication Codes

The preceding section discussed how to achieve *secrecy*; we now discuss techniques for ensuring *integrity* (sometimes also called *authenticity*). Here, the problem is as follows: parties A and B share a key k in advance, and then communicate over a channel that is under the complete control of an adversary. When B receives a message, he wants to ensure that this message indeed originated from A and was not, e.g., injected by the adversary, or generated by modifying the real message sent by A.

Secrecy and integrity are incomparable goals, and it is possible to achieve either one without the other. In particular, the one-time pad —which achieves perfect secrecy — provides no integrity whatsoever since *any* ciphertext c of the appropriate length decrypts to some valid message. Even worse, if c represents the encryption of some (possibly unknown) message m, then flipping the first bit of c has the predictable effect of flipping the first bit of the resulting decrypted message. This illustrates that *integrity is not implied by secrecy, and if integrity is required then specific techniques to achieve it must be used.*

In the private-key setting, the right tool in this context is a message authentication code (MAC). We first define the syntax. A MAC consists of two algorithms $(\mathsf{Mac}, \mathsf{Vrfy})$. The *tag-generation algorithm* Mac takes as input a key k and a message m, and outputs a tag tag; although it is possible for this algorithm to be randomized there is not much loss of generality in assuming that it is deterministic, and so we write this as $\mathsf{tag} := \mathsf{Mac}_k(m)$. The *verification algorithm* Vrfy takes as input a key, a message, and a tag; it outputs a single bit b with the intention that $b = 1$ denotes “validity” and $b = 0$ indicates “invalidity”. (If $\mathsf{Vrfy}_k(m, \mathsf{tag}) = 1$ then we say that tag is a *valid tag* for message m with respect to key k.) An honestly generated tag on a message m should be accepted as valid, and so correctness requires that for any key k and message m we have $\mathsf{Vrfy}_k(m, \mathsf{Mac}_k(m)) = 1$.

Defining security for message authentication codes is relatively simple, and there is only one widely accepted definition [23, 3]. At a high level, the goal is to prevent an adversary from generating a valid tag on a message that was never previously authenticated by one of the honest parties. This should hold even if the adversary observes valid tags on several other messages of its choice. Formally, consider an adversary $\mathcal{A}$ who is given access to an oracle $\mathsf{Mac}_k(\cdot)$; the adversary can submit messages of its choice to this oracle and obtain the corresponding valid tags. $\mathcal{A}$ succeeds if it can then output (m, tag) such that (1) m was not one of the messages $m_1, \ldots$ that $\mathcal{A}$ had previously submitted to its oracle, and (2) tag is a valid tag for m; i.e., $\mathsf{Vrfy}_k(m, \mathsf{tag}) = 1$. A MAC is (t, ε)-*secure* if for all $\mathcal{A}$ running in time at most t, the probability with which $\mathcal{A}$ succeeds in this experiment (where the probability is taken over random choice of k) is at most ε. This security notion is also called *existential unforgeability under adaptive chosen-message attack*.

One attack not addressed in the above discussion is a *replay attack*, whereby an adversary re-sends an honestly generated message m along with its valid tag. The receiver, in general, has no way of knowing whether the legitimate sender has simply sent m again, or whether this second instance of m was injected by the adversary. Message authentication codes, as defined, are stateless and so are unable to prevent such an attack. This is by choice: since repeated messages may be legitimate in some contexts, any handling of replay attacks must be taken care of at a higher level.

We now show a simple construction of a secure MAC based on any pseudorandom function/block cipher $F : \{0,1\}^n \times \{0,1\}^\ell \to \{0,1\}^\ell$ with ℓ sufficiently long [21]. To generate a tag on a message $m \in \{0,1\}^\ell$ using key k, simply compute $\mathsf{tag} := F_k(m)$. Verification is done in the obvious way: $\mathsf{Vrfy}_k(m, \mathsf{tag})$ outputs 1 if and only if $\mathsf{tag} \overset{?}{=} F_k(m)$. We sketch the proof of security for this construction. Let $m_1, \ldots$ denote those messages for which adversary $\mathcal{A}$ has requested a tag. Since F is a pseudorandom function, $\mathsf{Mac}_k(m) = F_k(m)$ "looks random" for any $m \notin \{m_1, \ldots\}$; the probability with which $\mathcal{A}$ can correctly predict the value of $F_k(m)$, then, is roughly $2^{-\ell}$. For ℓ large enough, the probability of a successful forgery is very small.

3.1 Message Authentication Codes for Long Messages

The construction in the previous section assumed that messages to be authenticated had length ℓ, the block length of the underlying pseudorandom function. Practical block ciphers have relatively short block length (e.g., $\ell \approx 128$ bits), which means that only very short messages can be authenticated by this construction. In this section we explore two approaches for doing better.

The first approach is a specific construction called the *cipher-block chaining MAC* (CBC-MAC), which can be based on any pseudorandom function/block cipher F as before. Here, we assume that the length of the messages to be authenticated is some *fixed* multiple of the block length ℓ. The tag on a message $M = \langle m_1, m_2, \ldots, m_L \rangle$ (with $m_i \in \{0,1\}^\ell$) using key k is computed as follows:

```
$\mathsf{tag}_0 = 0^\ell$
For $i = 1$ to $L$:
  $\mathsf{tag}_i = F_k(m_i \oplus \mathsf{tag}_{i-1})$
Output $\mathsf{tag}_L$
```

Verification of a tag tag on a message $M = \langle m_1, \ldots, m_L \rangle$ is done by re-computing tag_L as above and outputting 1 if and only if $\mathsf{tag} = \mathsf{tag}_L$.

CBC-MAC is known to be secure if F is a pseudorandom function [3]. This is true only as long as fixed-length messages (i.e., when the number of message blocks L is fixed) are authenticated, and there are several known attacks on the basic CBC-MAC presented above when this is not the case. Subsequent work has shown how to extend basic CBC-MAC to allow authentication of arbitrary-length messages [34, 8, 25].

A second approach to authenticating arbitrary-length messages is generic, in that it gives a way to modify *any* MAC for short messages so as to handle longer messages. As we will present it here, however, this approach requires an additional cryptographic primitive called a **collision-resistant hash function**. Although hash functions play an important role in cryptography, our discussion will be brief and informal since they are used sparingly in the remainder of this survey.

A **hash function** H is a function that *compresses* an arbitrary-length input to a short, fixed-length string. Hash functions are widely used in many areas of computer science, but cryptographic hash functions have some special requirements that are typically not needed for other applications. The most important such requirement, and the only one we will discuss, is *collision resistance*. Informally, H is collision resistant if it is infeasible for an efficient adversary to find a *collision* in H, where a collision is a pair of distinct inputs x, x' with $H(x) = H(x')$. If H is collision resistant then the hash of a long message serves as a "secure digest" of that message, in the following sense: for any value y (whether produced by an adversary or not), an adversary can come up with at most one x such that $H(x) = y$. The output length of a hash function fixes an upper bound on the computational difficulty of finding a collision: if H has output length ℓ, then a collision can always be found in $\mathcal{O}(2^{\ell/2})$ steps (see [27]).

As in the case of block ciphers, collision-resistant hash functions can be constructed from number-theoretic assumptions but such constructions are inefficient. (Interestingly, the precise assumptions needed to construct collision-resistant hash functions appear to be stronger than what is necessary to construct block ciphers.) Several dedicated, efficient constructions of hash functions are known; the most popular ones are currently given by the SHA family of hash functions [15], which have output lengths ranging from 160–512 bits. As of the time of this writing, however, NIST is running a public competition to choose a replacement (see `http://www.nist.gov/hash-competition`).

Returning to our discussion of message authentication codes, let $H : \{0,1\}^* \rightarrow \{0,1\}^\ell$ be a collision-resistant hash function, and let $(\mathsf{Mac}, \mathsf{Vrfy})$ be a secure message authentication code for messages of length ℓ. Then we can construct a message authentication code $(\mathsf{Mac}', \mathsf{Vrfy}')$ for arbitrary-length messages as follows. To authenticate m, first hash it to an ℓ-bit digest, and then authenticate the digest using the original MAC; i.e., $\mathsf{Mac}'_k(m) \overset{\text{def}}{=} \mathsf{Mac}_k(H(m))$. Verification is done in the natural way, with $\mathsf{Vrfy}'_k(m, \mathsf{tag}) \overset{\text{def}}{=} \mathsf{Vrfy}_k(H(m), \mathsf{tag})$. It is not difficult to show that this construction is secure. The standardized HMAC message authentication code [2, 17] can be viewed as following the above paradigm.

3.2 Joint Secrecy and Integrity

When communicating over a public channel, it is usually the case that both secrecy and integrity are required. Schemes achieving both these properties are called **authenticated encryption schemes** [4, 28]. The natural way to achieve both these properties is to combine a private-key encryption scheme with a message authentication code; there are a number of subtleties in doing so, and the reader is referred elsewhere for a more in-depth treatment [4, 27]. More efficient constructions of authenticated encryption schemes are also known.

4 The Public-Key Setting and Public-Key Encryption

The private-key setting we have been considering until now requires the honest parties to share a secret key in advance in order to secure their communication. Historically, the private-key setting was the only one considered in cryptography. In the mid-1970s [11, 36, 33, 35], however, the field of cryptography was revolutionized by the development of *public-key cryptography* which can enable secure communication between parties who share no secret information in advance and carry out all their communication over a public channel. The only requirement is that there is a way for one party to reliably send a copy of their public key to the other.

In the setting of public-key cryptography, any party A generates a pair of keys (pk, sk) on its own; the private key sk is held privately by A, while the public key pk must be obtained by any other party B who wishes to communicate with A. The first party can either send a copy of its public key directly to B, if an authenticated (but not necessarily private!) channel is available between A and B; alternately, A can place a copy of its public key in a public directory (or on her webpage) and B can then obtain pk when needed. Regardless of which method is used, when we analyze security of public-key cryptosystems we simply assume that parties are able to obtain authentic copies of each others' public keys. (In practice, certification authorities and a public-key infrastructure are used to reliably distribute public keys; a discussion is outside the scope of this chapter.) We assume the adversary also knows all parties' public keys; this makes sense since parties make no effort to keep their public keys secret.

Public-key cryptography has a number of advantages relative to private-key cryptography. As we have already discussed, the use of public-key cryptography can potentially simplify key distribution since a private channel between the communicating users is not needed (as it is in the private-key setting). Public-key cryptography can also simplify key management in large systems. For example, consider a company with N employees where each employee should be able to communicate securely with any other employee. Using a private-key solution would require each user to share a (unique) key with every other user, requiring each employee to store $N-1$ secret keys. In contrast, with a public-key solution each user would need to know their own private key and the $N-1$ public keys of the other employees; these public keys, however, could be stored in a public directory or other non-private storage. Finally, public-key cryptography is more suitable for "open systems" such as the Internet where the parties who need to communicate securely may have no prior trust relationship, as is the case when a (new) customer wants to encrypt their credit card prior to sending it to an on-line merchant.

The primary disadvantage of public-key cryptography is that it is less efficient than private-key cryptography. An exact comparison depends on many factors, but as a rough estimate, when encrypting "short" messages (say, less than 10 kilobytes) public-key cryptosystems are 500–1000 times slower than comparable private-key cryptosystems and use roughly a factor of 10 more bandwidth. The power consumption needed to use public-key cryptography can also be an issue when cryptography is used in low-power devices (sensors, RFID tags, etc.). Thus, when private-key cryptography is an option it is preferable to use it.

Let us formalize the syntax of public-key encryption. A public-key encryption scheme is composed of three algorithms $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$. The *key-generation algorithm* Gen is a randomized algorithm that outputs a pair of keys (pk, sk) as described earlier. The *encryption algorithm* Enc takes as input a public key pk and a message m, and outputs a ciphertext c. The *decryption algorithm* Dec takes as input a private key sk and a ciphertext c, and outputs a message m. (We highlight here that the decryption algorithm and the encryption algorithm use *different* keys.) Correctness requires that for all (pk, sk) output by Gen, and any message m, we have $\mathsf{Dec}_{sk}(\mathsf{Enc}_{pk}(m)) = m$.

Our definition of security is patterned on the earlier definition given for private-key encryption schemes; in fact, the only difference is that the adversary is given the public key. That is, public-key encryption scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is (t, ε)*-indistinguishable* if for every (probabilistic) eavesdropper $\mathcal{A}$ running in time at most t, and for all (equal-length) messages m_0, m_1, we have

$$\left| \Pr\left[\mathcal{A}(pk, \mathsf{Enc}_{pk}(m_b)) = b\right] - \frac{1}{2} \right| \leq \varepsilon,$$

where the probability is taken over random generation of (pk, sk) by Gen, uniform choice of b, and any randomness used by the encryption algorithm itself.

Interestingly, in the public-key setting the above definition implies security against chosen-plaintext attacks (and hence also security when multiple messages are encrypted) and for this reason we will also use the above as our definition of CPA-security; this is in contrast to the private-key setting where security against chosen-plaintext attacks is stronger than indistinguishability. A bit of reflection shows why: in the public-key setting access to an encryption oracle is superfluous, since an adversary who is given the public key can encrypt any messages it likes by itself. A consequence is that, in order to meet even a minimal notion of security in the public-key setting, encryption must be randomized.

In contrast to the private-key setting, public-key cryptography seems to inherently rely on number-theoretic techniques. The two most commonly used techniques are explored in the following two subsections. These sections assume some basic mathematical background on the part of the reader; the necessary background is covered in [27].

4.1 RSA Encryption

We discuss the general case of **RSA cryptography**, followed by its application to the particular case of public-key encryption.

Rivest, Shamir, and Adleman introduced the concept of RSA-based cryptography in 1978 [36]. Security here is ultimately based on (though not equivalent to) the assumption that factoring large numbers is hard, even though multiplying large numbers is easy. This, in turn, gives rise to problems that are easy if the factorization of some modulus N is known, but that are believed to be hard when the factors of N are unknown. This asymmetry can be exploited to construct public-key cryptosystems.

Specifically, let $N = pq$ be the product of two large primes p and q. (For concreteness, one may take p and q to be 1000-bit integers.) Let $\mathbb{Z}_N^*$ denote the set of integers between 1 and $N-1$ that are invertible modulo N; i.e., $x \in \mathbb{Z}_N^*$ if there exists an integer x^{-1} such that $x \cdot x^{-1} = 1 \bmod N$. It is known that $\mathbb{Z}_N^*$ consists precisely of those integers between 1 and $N-1$ that are *relatively prime* to (i.e., have no factor in common with) N. Using this, one can show that

$$|\mathbb{Z}_N^*| = (p-1) \cdot (q-1).$$

Define $\varphi(N) \stackrel{\text{def}}{=} |\mathbb{Z}_N^*|$. Basic group theory implies that for any integers e, d with $ed = 1 \bmod \varphi(N)$ and any $x \in \mathbb{Z}_N^*$, it holds that

$$(x^e)^d = x \bmod N; \tag{2}$$

in particular, this means that the function $f_{N,e} : \mathbb{Z}_N^* \to \mathbb{Z}_N^*$ defined by

$$f_{N,e}(x) = x^e \bmod N$$

is a bijection. Anyone given N, e, and x can easily compute $f_{N,e}(x)$ using standard algorithms for efficient modular exponentiation. The *RSA problem* is the problem of inverting this function: namely, given N, e, y with $y \in \mathbb{Z}_N^*$, to find a value $x \in \mathbb{Z}_N^*$ such that $x^e = y \bmod N$.

If the factors of N are known, then it is easy to compute $\varphi(N)$ and hence for any e relatively prime to $\varphi(N)$ the value $d \stackrel{\text{def}}{=} e^{-1} \bmod \varphi(N)$ can be efficiently computed. This value d can then be used to invert $f_{N,e}$ using Equation (2), and so in this case the RSA problem is easily solved. On the other hand, there is no known efficient algorithm for inverting $f_{N,e}$ given only N and e

(and, in particular, without the factorization of N). The *RSA assumption* formalizes the apparent computational difficulty of solving the RSA problem. Let RSAGen be a randomized algorithm that outputs (N, e, d) where N is a product of two random, large primes and $ed = 1 \bmod \varphi(N)$. (We do not discuss how such an algorithm can be constructed; suffice it to say that efficient algorithms with the required behavior are known.) We say *the RSA problem is* (t, ε)*-hard for* RSAGen if, for all algorithms $\mathcal{A}$ running in time at most t,

$$\Pr[\mathcal{A}(N, e, y) = x \text{ s.t. } x^e = y \bmod N] \leq \varepsilon,$$

where the probability is taken over the randomness of RSAGen as well as uniform choice of $y \in \mathbb{Z}_N^*$. It is clear that if the RSA problem is hard for RSAGen, then factoring moduli N output by RSAGen must be hard; the converse is not known, but hardness of the RSA problem for moduli generated appropriately (i.e., the RSA assumption) is widely believed to hold.

The above discussion should naturally motivate our first candidate public-key encryption scheme that we call "textbook RSA encryption". For this scheme, the key-generation algorithm runs RSAGen to obtain (N, e, d); the public key consists of (N, e) while the private key contains (N, d). To encrypt a message $m \in \mathbb{Z}_N^*$ using the public key (N, e), the sender computes $c := m^e \bmod N$. Given a ciphertext c thus computed, the message can be recovered by using the corresponding private key to compute

$$c^d \bmod N = (m^e)^d = m \bmod N.$$

It follows directly from the RSA assumption that if m is chosen uniformly from $\mathbb{Z}_N^*$, then an adversary who observes the ciphertext c cannot compute m in its entirety. But this, alone, is not a very satisfying guarantee! For one thing, a real-life message is unlikely to be random; in particular, it may very well correspond to some structured text. Furthermore, even in the case that m is random, the textbook RSA scheme provides no assurance that an adversary cannot deduce some partial information about m (and in fact, recovering some partial information about m is known to be possible). Finally, this scheme does not have randomized encryption; textbook RSA encryption thus cannot possibly be CPA-secure.

Randomized variants of textbook RSA, however, are used extensively in practice. For simplicity, we describe here a variant that roughly corresponds to the RSA PKCS #1 v1.5 standard [37]. (This standard has since been superseded by a later version which should be used instead.) Here, the message is randomly padded during encryption. I.e., if N is a 2000-bit modulus then encryption of an 800-bit message m can be done by choosing a random 200-bit string r and then computing

$$c = (r \| m)^e \bmod N;$$

decryption is done in the natural way. It is conjectured that this scheme is CPA-secure, though no proof based on the RSA assumption is known.

4.2 El Gamal Encryption

A second class of problems that can be used for public-key cryptography is related to the presumed hardness of the *discrete logarithm problem* in certain groups. This idea was introduced, in somewhat different form, by Diffie and Hellman in 1976 [11]. Our treatment will be abstract, though after introducing the basic ideas we will discuss some concrete instantiations.

Let $\mathbb{G}$ be a finite, cyclic group of large prime order q, and let $g \in \mathbb{G}$ be a generator of $\mathbb{G}$. The fact that g is a generator implies that for any element $h \in \mathbb{G}$ there exists an $x \in \{0, \ldots, q-1\} = \mathbb{Z}_q$ such that $g^x = h$. By analogy with logarithms over the real numbers, in this case we write $x = \log_g h$. The *discrete logarithm problem in* $\mathbb{G}$ is to compute $\log_g h$ given h. For many groups (see below), this problem appears to be computationally infeasible. We may formalize this, the *discrete logarithm assumption in* $\mathbb{G}$, as follows: the discrete logarithm problem in $\mathbb{G}$ is (t, ε)-hard if for all algorithms $\mathcal{A}$ running in time at most t it holds that

$$\Pr[\mathcal{A}(h) = x \text{ s.t. } g^x = h] \leq \varepsilon,$$

where the probability is taken over uniform choice of $h \in \mathbb{G}$.

For cryptographic applications a stronger assumption is often needed. The **decisional Diffie-Hellman** problem is to distinguish tuples of the form (g^x, g^y, g^{xy}) (where x, y are chosen uniformly from $\mathbb{Z}_q$) from tuples of the form (g^x, g^y, g^z) (where x, y, z are chosen uniformly from $\mathbb{Z}_q$). We say the *decisional Diffie-Hellman problem is* (t, ε)*-hard in* $\mathbb{G}$ if for all algorithms $\mathcal{A}$ running in time at most t we have

$$|\Pr[\mathcal{A}(g^x, g^y, g^{xy}) = 1] - \Pr[\mathcal{A}(g^x, g^y, g^z) = 1]| \leq \varepsilon,$$

where the probability space in each case is as described earlier. One way to solve the decisional Diffie-Hellman problem given a candidate tuple (g_1, g_2, g_3) is to compute $x = \log_g g_1$ and then check whether $g_3 \overset{?}{=} g_2^x$. We thus see that the decisional Diffie-Hellman problem in some group $\mathbb{G}$ is no harder than the discrete logarithm problem in the same group. The converse is not, in general, true as there are candidate groups in which the discrete logarithm problem is hard but the decisional Diffie-Hellman problem is not. Nevertheless, for the groups discussed below the stronger decisional Diffie-Hellman problem is widely believed to hold.

A classical example of a class of groups for which the above assumptions are believed to hold is given by large prime-order subgroups of $\mathbb{Z}_p^*$ for p prime. As one concrete example, let $p = 2q + 1$ where both p and q are prime. Then the set of *quadratic residues* modulo p (namely, elements of $\mathbb{Z}_p^*$ that can be written as squares of other elements) constitutes a subgroup $\mathbb{G} \subset \mathbb{Z}_p^*$ of order q. Another important example is given by the group of points over certain elliptic curves [43].

In their initial paper [11], Diffie and Hellman suggested a key-exchange protocol based on (what would later come to be called) the decisional Diffie-Hellman problem. This protocol was later adapted [12] to give the *El Gamal encryption scheme* that we describe now. Fix a group $\mathbb{G}$ of prime order q, and generator $g \in \mathbb{G}$ as above. The key-generation algorithm simply chooses a random $x \in \mathbb{Z}_q$ which will be the private key; the public key is $h = g^x$. To encrypt a message $m \in \mathbb{G}$ with respect to the public key h, the sender chooses a random $y \in \mathbb{Z}_q$ and outputs the ciphertext $\langle g^y, h^y \cdot m \rangle$. Decryption of a ciphertext $\langle c_1, c_2 \rangle$ using the private key x is done by computing c_2/c_1^x. This recovers the correct result since

$$\frac{c_2}{c_1^x} = \frac{h^y \cdot m}{(g^y)^x} = \frac{(g^x)^y \cdot m}{g^{yx}} = \frac{g^{xy} \cdot m}{g^{yx}} = m.$$

It can be shown that the El Gamal encryption scheme is CPA-secure if the decisional Diffie-Hellman problem is hard in the underlying group $\mathbb{G}$.

4.3 Hybrid Encryption

In contrast to private-key encryption, which can be based on highly efficient block ciphers, current public-key encryption schemes involve arithmetic operations with "very large" integers. (Appropriate choice of parameters depends on numerous factors; the reader can think of operations on

$\approx$1000-bit numbers though that need not always be the case.) As a consequence, naive public-key encryption is orders of magnitude slower than private-key encryption.

When encrypting very long messages, however, *hybrid encryption* can be used to obtain the functionality of public-key encryption with the (asymptotic) efficiency of private-key encryption. Let Enc denote the encryption algorithm for some public-key scheme, and let Enc' denote the encryption algorithm for a private-key scheme using keys of length n. Hybrid encryption works as follows: to encrypt a (long) message m the sender first chooses a random secret key $k \in \{0,1\}^n$, encrypts k using Enc and the public key pk, and then encrypts m using Enc' and the key k. I.e., the entire ciphertext is $\langle \mathsf{Enc}_{pk}(k), \mathsf{Enc}'_k(m)\rangle$. Decryption is done in the natural way, by recovering k from the first component of the ciphertext and then using k to decrypt the second part of the ciphertext. Note that the public-key scheme Enc is used only to encrypt a short key, while the bulk of the work is done using the more efficient private-key scheme.

One can show that hybrid encryption is CPA-secure if (1) the underlying public-key encryption scheme is CPA-secure and (2) the underlying private-key encryption scheme is indistinguishable. (CPA-security of the private-key encryption scheme is not necessary; intuitively, this is because the key k is generated freshly at random each time encryption is done, so any particular key is used only once.)

4.4 Stronger Security Notions

Our treatment of public-key encryption has only focused on security against passive attacks. As in the case of private-key encryption, however, in many scenarios security against an active adversary is also required. In fact, active attacks are arguably an even greater concern in the public-key setting where a recipient uses the same public key pk to communicate with multiple senders, some of whom may be malicious. (We refer to [27] for examples of some potential attacks.) To defend against this possibility, public-key encryption schemes satisfying stronger notions of security should be used; see [27] for details.

5 Digital Signature Schemes

As public-key encryption is to private-key encryption, so are digital signature schemes to message authentication codes. That is, signature schemes allow a *signer* who has established a public key to "sign" messages in a way which is verifiable to anyone who knows the signer's public key. Furthermore (by analogy with MACs), no adversary can forge a valid signature on any message that was not explicitly authenticated (i.e., signed) by the legitimate signer.

Formally, a signature scheme is comprised of three algorithms $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$. The *key-generation algorithm* Gen is a probabilistic algorithm that outputs a pair of public and private keys (pk, sk) exactly as in the case of public-key encryption. The *signing algorithm* Sign takes as input a private key sk and a message m, and outputs a *signature* σ; we denote this by $\sigma := \mathsf{Sign}_{sk}(m)$. (One may assume this algorithm is deterministic without loss of generality.) Finally, the *verification algorithm* Vrfy takes as input a public key, a message, and a signature and outputs 0 or 1. If $\mathsf{Vrfy}_{pk}(m, \sigma) = 1$ then we say that σ is a *valid signature* on m (with respect to pk). Analogous to the case of MACs, the correctness requirement is that for every (pk, sk) output by Gen, and for any message m, an honestly generated signature $\mathsf{Sign}_{sk}(m)$ is always a valid signature on m.

A signature scheme can be used in the following way. User A locally generates (pk, sk) and widely publicizes her public key pk. She can then authenticate any desired message m by computing $\sigma := \mathsf{Sign}_{sk}(m)$. Any other user B who has obtained a legitimate copy of A's public key can then verify that this message was indeed certified by A by checking that $\mathsf{Vrfy}_{pk}(m, \sigma) \overset{?}{=} 1$. Note that in this case, in contrast to the case of public-key encryption, the owner of the public key acts as the sender rather than as the receiver.

Security for digital signature schemes is defined in a manner completely analogous to that of message authentication codes. As there, we consider an adversary $\mathcal{A}$ who is given access to an oracle $\mathsf{Sign}_{sk}(\cdot)$; the adversary can submit messages of its choice to this oracle and obtain the corresponding valid signatures. In addition, here the adversary is also given the public key pk. Adversary $\mathcal{A}$ succeeds if it can then output (m, σ) such that (1) m was not one of the messages $m_1, \ldots$ that $\mathcal{A}$ had previously submitted to its oracle, and (2) σ is a valid signature on m; i.e., $\mathsf{Vrfy}_{pk}(m, \sigma) = 1$. A signature scheme is (t, ε)-*secure* if for all $\mathcal{A}$ running in time at most t, the probability with which $\mathcal{A}$ succeeds in this experiment (where the probability is taken over generation of (pk, sk)) is at most ε. This security notion is also called *existential unforgeability under adaptive chosen-message attack* [23].

Signature schemes offer several advantages relative to message authentication codes. A basic advantage is that, once a signer's public key is widely distributed, that signer's signatures can be verified by *anyone*. (Contrast this to message authentication codes, where a tag can only be verified by the party who shares the key with the sender.) As a result, signature schemes are suitable when "one-to-many" communication is required, with a canonical example being a software company releasing signed updates of its software. The very fact that signatures are publicly verifiable also implies their usefulness for *non-repudiation*. That is, once a signer has issued a signature on a message, a recipient can prove that fact to someone else (a judge, say) by simply showing them the signature. (This makes digital signatures a potential replacement for handwritten signatures on legally binding documents.) On the other hand, for authenticating "short" messages a signature scheme may be orders of magnitude less efficient than a message authentication code.

In the following sections we briefly discuss two signature schemes in wide use today. Other schemes, some with better theoretical properties that those discussed here, can be found in [26]. Note that it suffices to construct a signature scheme for "short" messages; collision-resistant hashing can then be used, as in Section 3.1, to extend any such scheme for signing arbitrary-length messages.

5.1 RSA Signatures

We can use the RSA assumption introduced in Section 4.1 to construct signature schemes. We begin with a simple scheme that is insecure, and then discuss how to adapt it so as to make it resistant to attack.

At an intuitive level, signing a message must involve some operation that is "easy" for the signer who knows the private key corresponding to its own public key, but "difficult" for anyone who does *not* know the signer's private key. At the same time, the signature thus computed must be easy to verify for anyone knowing only the signer's public key. The RSA problem discussed earlier provides exactly this. Key generation is exactly the same as for RSA public-key encryption; i.e., the algorithm runs RSAGen to obtain (N, e, d), and outputs (N, e) and the public key, and (N, d) as the private key. To sign a message $m \in \mathbb{Z}_N^*$ using its private key, the signer computes $\sigma := m^d \bmod N$. Verification of a purported signature σ on a message m with respect to a public key (N, e) is done by checking whether $\sigma^e \overset{?}{=} m \bmod N$. Correctness holds for an honestly generated signature since

$$\sigma^e = \left(m^d\right)^e = m \bmod N.$$

Hardness of the RSA problem implies that it is infeasible for an adversary who knows only the signer's public key to generate a valid signature on a random message $m \in \mathbb{Z}_N^*$. But this is insufficient for security! In fact, the "textbook RSA" signature scheme just described is not secure. Here are two attacks:

1. An adversary knowing only the public key can easily construct a forgery by choosing an arbitrary $\sigma \in \mathbb{Z}_N^*$ and outputting this as the signature on the message $m = \sigma^e \bmod N$. Although m computed in this way may not be "meaningful," this attack is already enough to violate the security definition. (In fact, the definition of what constitutes a "meaningful" message is context-dependent and so such an attack may be problematic in practice. Moreover, by repeatedly choosing random σ and following the same steps the adversary may end up hitting upon a meaningful message.)

2. Perhaps even worse, an adversary can forge a valid signature on an arbitrary message $\hat{m}$ if it can get the signer to sign a single (different) message of the adversary's choice. This attack begins by first having the adversary compute (m, σ) as above. Assume the adversary can then obtain a signature σ' (from the legitimate signer) on the message $m' = (\hat{m}/m) \bmod N$. The adversary then computes the signature $\hat{\sigma} = \sigma \cdot \sigma' \bmod N$ on $\hat{m}$. Note that $\hat{\sigma}$ is a valid signature on $\hat{m}$ since

$$\hat{\sigma}^e = \left(\sigma \cdot \sigma'\right)^e = \sigma^e \cdot (\sigma')^e = m \cdot m' = \hat{m} \bmod N.$$

The above show that textbook RSA signatures are not secure. Fortunately, a simple modification prevents the above attacks: *hash* the message before signing it. That is, the signature on a message $m \in \{0,1\}^*$ is now

$$\sigma = H(m)^d \bmod N,$$

where H is a hash function mapping arbitrary-length inputs to $\mathbb{Z}_N^*$. Verification is done in the natural way. It can be shown [6] that this scheme is secure if the RSA problem is hard and if, in addition, H "acts like a random function" in a sense we do not define here; see [5, 27] for further discussion.

5.2 The Digital Signature Standard

The Digital Signature Algorithm (DSA), included as part of the Digital Signature Standard (DSS) [1, 16], is another widely used signature scheme. Its security is related (though not known to be equivalent) to the hardness of computing discrete logarithms in a specific group $\mathbb{G}$. Specifically, $\mathbb{G}$ is taken to be a subgroup of $\mathbb{Z}_p^*$ of order q where both p and q are large primes. Let g be a generator of $\mathbb{G}$, and let H be a cryptographic hash function that, for simplicity, we view as mapping arbitrary-length inputs to $\mathbb{Z}_q$. (As will be clear, for security to hold H must at least be collision resistant.) Key generation works by choosing a random $x \in \mathbb{Z}_q$ and setting $y = g^x \bmod p$; the public key is y and the private key is x. To sign a message $m \in \{0,1\}^*$, the signer generates a random $k \in \mathbb{Z}_q$ and computes

$$\begin{aligned} r &= (g^k \bmod p) \bmod q \\ s &= (H(m) + xr) \cdot k^{-1} \bmod q\,; \end{aligned}$$

the signature is (r, s). Verification of signature (r, s) on message m with respect to public key y is done by checking that $r, s \in \mathbb{Z}_q^*$ and

$$r \overset{?}{=} (g^{H(m)s^{-1}} \cdot y^{rs^{-1}} \bmod p) \bmod q.$$

No proof of security for DSA is known; we refer the reader to a survey article by Vaudenay [42] for further discussion. See [26] and references therein for other signature schemes whose security can be proved equivalent to the discrete logarithm or Diffie-Hellman problems.

6 Further Information

The textbook by this author and Yehuda Lindell [27] covers the material of this survey, and more, in extensive detail. The on-line notes of Bellare and Rogaway [7] are also quite useful. The books by Smart [39] and Stinson [41] give slightly different perspectives on the material. Readers looking for a more advanced treatment may consult the texts by Goldreich [18, 19]. More applied aspects of cryptography can be found in the book by Stallings [40] or the comprehensive (though now somewhat outdated) *Handbook of Applied Cryptography* [32].

The International Association for Cryptologic Research (IACR) sponsors the *Journal of Cryptology* along with a number of conferences, with Crypto and Eurocrypt being the best known. Proceedings of these conferences (dating, in some cases, to the early 1980s) are published as part of Springer-Verlag's *Lecture Notes in Computer Science.* Research in theoretical cryptography often appears at the ACM Symposium on Theory of Computing, the IEEE Symposium on Foundations of Computer Science, and elsewhere; more applied aspects of cryptography are frequently published in security conferences including the ACM Conference on Computer and Communications Security. Articles on cryptography also appear in the *Journal of Computer and System Sciences*, the *Journal of the ACM*, and the *SIAM Journal on Computing.*

Defining Terms

Block cipher: An efficient instantiation of a pseudorandom function that has the additional property of being efficiently invertible.
Chosen-plaintext attack. An attack on an encryption scheme in which an adversary causes the sender to encrypt messages of the adversary's choice.
Ciphertext: The result of encrypting a message.
Collision-resistant hash function: A hash function for which it is infeasible to find two different inputs mapping to the same output.
Computational security: Provides security against computationally bounded adversaries, assuming the computational hardness of some problem.
Data integrity: Ensuring that modifications to a communicated message are detected.
Data secrecy: Hiding the contents of a communicated message.
Decrypt: To recover the original message from the ciphertext.
Digital signature scheme: A method for providing data integrity in the public-key setting.
Encrypt: To apply an encryption scheme to a plaintext message.
Hash function: A cryptographic function mapping arbitrary-length inputs to fixed-length outputs.

Information-theoretic security: Provides security even against computationally unbounded adversaries.
Key: The secret shared by parties in the private-key setting. Also used for the public and private values generated by a party in the public-key setting.
Message authentication code: A method for providing data integrity in the private-key setting.
Mode of encryption: A method for using a block cipher to encrypt arbitrary-length messages.
One-time pad: A private-key encryption scheme achieving perfect secrecy.
Plaintext: The communicated data, or message.
Private-key encryption: Technique for ensuring data secrecy in the private-key setting.
Private-key setting: Setting in which communicating parties share a secret in advance of their communication.
Pseudorandom function: A keyed function which is indistinguishable from a truly random function.
Public-key encryption: Technique for ensuring data secrecy in the public-key setting.
Public-key setting: Setting in which parties generate public/private keys and widely disseminate their public key.
RSA cryptography: Cryptography based on the hardness of computing eth roots modulo a number N that is the product of two large primes.

References

[1] ANSI X9.30. 1997. Public key cryptography for the financial services industry, part 1: The digital signature algorithm (DSA). American National Standards Institute. American Bankers Association.

[2] Bellare, M., Canetti, R., and Krawczyk, H. 1996. Keying hash functions for message authentication. *Advances in Cryptology — Crypto '96*, Lecture Notes in Computer Science, vol. 1109, N. Koblitz, Ed., Springer-Verlag, pp. 1–15.

[3] Bellare, M., Kilian, J., and Rogaway, P. 2000. The security of the cipher block chaining message authentication code. *J. Computer and System Sciences* 61(3): 362–399.

[4] Bellare, M. and Namprempre, C. 2000. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. *Advances in Cryptology — Asiacrypt 2000*, Lecture Notes in Computer Science, vol. 1976, T. Okamoto, Ed., Springer-Verlag, pp. 531–545.

[5] Bellare, M. and Rogaway, P. 1993. Random oracles are practical: a paradigm for designing efficient protocols. *First ACM Conference on Computer and Communications Security*, ACM, pp. 62–73.

[6] Bellare, M. and Rogaway, P. 1996. The exact security of digital signatures: How to sign with RSA and Rabin. *Advances in Cryptology — Eurocrypt '96*, Lecture Notes in Computer Science, vol. 1070, U. Maurer, Ed., Springer-Verlag, pp. 399–416.

[7] Bellare, M. and Rogaway, P. 2005. Introduction to modern cryptography (course notes). Available at `http://www.cs.ucsd.edu/users/mihir/cse207/classnotes.html`.

[8] Black, J. and Rogaway, P. 2005. CBC MACs for arbitrary-length messages: The three-key constructions. *J. Cryptology* 18(2): 111-131.

[9] Blum, M., and Micali, S. 1984. How to generate cryptographically strong sequences of pseudorandom bits. *SIAM J. Computing* 13(4): 850–864.

[10] Daemen, J. and Rijmen, V. 2002. *The Design of Rijndael: AES — the Advanced Encryption Standard*. Springer-Verlag.

[11] Diffie, W. and Hellman, M. 1976. New directions in cryptography. *IEEE Transactions on Information Theory* 22(6): 644–654.

[12] El Gamal, T. 1985. A public-key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory* 31(4): 469–472.

[13] *Federal Information Processing Standards* publication #46-3. 1999. Data encryption standard (DES). U.S. Department of Commerce/National Institute of Standards and Technology.

[14] *Federal Information Processing Standards* publication #81. 1980. DES modes of operation. U.S. Department of Commerce/National Bureau of Standards.

[15] *Federal Information Processing Standards* publication #180-2. 2002. Secure hash standard. U.S. Department of Commerce/National Institute of Standards and Technology.

[16] *Federal Information Processing Standards* publication #186-2. 2000. Digital signature standard (DSS). U.S. Department of Commerce/National Institute of Standards and Technology.

[17] *Federal Information Processing Standards* publication #198. 2002. The Keyed-Hash Message Authentication Code (HMAC). U.S. Department of Commerce/National Institute of Standards and Technology.

[18] Goldreich, O. 2001. *Foundations of Cryptography, vol. 1: Basic Tools*. Cambridge University Press.

[19] Goldreich, O. 2004. *Foundations of Cryptography, vol. 2: Basic Applications*. Cambridge University Press.

[20] Goldreich, O., Goldwasser, S., and Micali, S. 1986. How to construct random functions. *J. ACM* 33(4): 792–807.

[21] Goldreich, O., Goldwasser, S., and Micali, S. 1985. On the cryptographic applications of random functions. *Advances in Cryptology — Crypto '84*, Lecture Notes in Computer Science, vol. 196, G.R. Blakley and D. Chaum, Eds., Springer-Verlag, pp. 276–288.

[22] Goldwasser, S. and Micali, S. 1984. Probabilistic encryption. *J. Computer and System Sciences* 28(2): 270–299.

[23] Goldwasser, S., Micali, S., and Rivest, R. 1988. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Computing* 17(2): 281–308.

[24] Håstad, J., Impagliazzo, R., Levin, L., and Luby, M. 1999. A pseudorandom generator from any one-way function. *SIAM J. Computing* 28(4): 1364–1396.

[25] Iwata, T. and Kurosawa, K. 2003. OMAC: One-Key CBC MAC. *Fast Software Encryption — FSE 2003*, Lecture Notes in Computer Science, vol. 2887, T. Johansson, Ed., Springer-Verlag, pp. 129-153.

[26] Katz, J. 2010. *Digital Signatures.* Springer.

[27] Katz, J. and Lindell, Y. 2007. *Introduction to Modern Cryptography.* Chapman and Hall/CRC Press.

[28] Katz, J. and Yung, M. 2000. Unforgeable encryption and chosen ciphertext secure modes of operation. *Fast Software Encryption — FSE 2000*, Lecture Notes in Computer Science, vol. 1978, B. Schneier, Ed., Springer-Verlag, pp. 284–299.

[29] Kerckhoffs, A. 1883. La cryptographie militaire. *J. des Sciences Militaires*, 9th Series, pp. 161–191.

[30] Knudsen, L.R. and Robshaw, M.J.B. 2011. *The Block Cipher Companion.* Springer.

[31] Luby, M. and Rackoff, C. 1988. How to construct pseudorandom permutations from pseudorandom functions. *SIAM J. Computing* 17(2): 412–426.

[32] Menezes, A.J., van Oorschot, P.C., and Vanstone, S.A. 2001. *Handbook of Applied Cryptography.* CRC Press.

[33] Merkle, R. and Hellman, M. 1978. Hiding information and signatures in trapdoor knapsacks. *IEEE Transactions on Information Theory* 24: 525–530.

[34] Petrank, E. and Rackoff, C. 2000. CBC MAC for real-time data sources. *J. Cryptology* 13(3): 315–338.

[35] Rabin, M.O. 1979. Digitalized signatures and public key functions as intractable as factoring. MIT/LCS/TR-212, MIT Laboratory for Computer Science.

[36] Rivest, R., Shamir, A., and Adleman, L.M. 1978. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM* 21(2): 120–126.

[37] RSA Laboratories. 1993. RSA laboratories public-key cryptography standard #1, version 1.5. Available from `http://www.rsa.com/rsalabs`.

[38] Shannon, C.E. 1949. Communication theory of secrecy systems. *Bell System Technical Journal* 28(4): 656–715.

[39] Smart, N. 2004. *Cryptography: An Introduction.* McGraw-Hill.

[40] Stallings, W. 2010. *Cryptography and Network Security: Principles and Practice, 5th edition.* Prentice Hall.

[41] Stinson, D.R. 2005. *Cryptography: Theory and Practice (third edition).* Chapman & Hall/CRC Press.

[42] Vaudenay, S. 2003. The security of DSA and ECDSA. *Public-Key Cryptography — PKC 2003*, Lecture Notes in Computer Science, vol. 2567, Y. Desmedt, Ed., Springer-Verlag, pp. 309–323.

[43] Washington, L. 2008. *Elliptic Curves: Number Theory and Cryptography*. Chapman and Hall/CRC Press.

[44] Yao, A.C. 1982. Theory and application of trapdoor functions. *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*, IEEE, pp. 80–91.