# *Introduction to Encryption*

*6.1600 Course Staff: Henry Corrigan-Gibbs, Yael Kalai, Ben Kettle (TA), Nickolai Zeldovich*

*Fall 2022*

**Disclaimer:** This set of notes is a work in progress. It may have errors and be missing citations. It is certainly incomplete. Please let the staff know of any errors that you find.

So far, we have explored methods for authenticating data—verifying that it has not been modified in transit. However, the integrity-protection mechanisms we have discussed provide no *confidentiality*: a network eavesdropper can still view everything that we send over the network.

In this chapter, we will discuss *encryption*, which allow two parties to exchange messages over an insecure network while hiding the contents of their communications.

We will cover encryption in a sequence of steps:

- First, we will construct an encryption scheme with a *weak form of security* for *fixed-length messages* for settings in which the sender and recipient have a *shared secret key*.
- Next, we will show how to extend this scheme to support *variable-length messages*.
- Then, we will show how to improve the scheme to have a *strong form of security*.
- Finally, we will show how to implement encryption in settings in which the sender and recipient *have no shared secrets*.

At the conclusion of this part, we will discuss how deployed systems use encryption, and we will think about some problems that encryption does not solve.

## 1 *Background*

*The need for encryption* The internet is a massive network of wifi access points, routers, switches, undersea cables, DNS servers, and much more. There are many, many devices for a potential adversary to compromise and many vantage points from which an attacker can observe network traffic. Every single hop your packets take is a potential point of compromise.

To make matters worse, most standard network protocols provide *no authentication or encryption*: in Ethernet, IP, DNS, email, HTTP, and others, an adversary is free to modify and read the traffic we send and receive. Protecting confidentiality typically requires augmenting these standard network protocols with some form of authentication and encryption.

*Systems using encryption* Encryption shows up in a large number of deployed systems. A few examples are:

- **Messaging apps**, such as WhatsApp, Signal, and iMessage, encrypt traffic between app users so that the server cannot easily read it.
- **Network protocols**, such as SSH and HTTPS use encryption to protect traffic between a service's clients and servers.
- **File-storage systems** use encryption to protect data at rest. So if a thief steals your laptop, they will not easily be able to read the encrypted files on your hard disk.

## 2 *Encryption Scheme Syntax*

Encryption schemes are defined with respect to a key space $\mathcal{K}$, a message space $\mathcal{M}$, and a ciphertext space $\mathcal{C}$. For now, think of $\mathcal{K} = \mathcal{M} = \{0,1\}^n$ and $\mathcal{C} = \{0,1\}^{2n}$, for a security parameter $n$. An encryption scheme then consists of two algorithms:

Remember that in practice, we will often take the size of the keyspace $|\mathcal{K}|$ to be at least $2^{128}$ to prevent brute-force key-guessing attacks.

- $\mathsf{Enc} : \mathcal{K} \times \mathcal{M} \to \mathcal{C}$
- $\mathsf{Dec} : \mathcal{K} \times \mathcal{C} \to \mathcal{M}$

As we will see later on, the decryption algorithm Dec can sometimes output "FAIL" or $\perp$.

The definition of correctness for an encryption scheme just states that if you encrypt a message $m$ with a key $k$, and then you decrypt the resulting ciphertext with the same key $k$, you end up with the same message $m$ that you started with:

**Definition 2.1** (Encryption Scheme, Correctness)**.** An encryption scheme is correct if, for all keys $k \in \mathcal{K}$ and all messages $m \in \mathcal{M}$, $\mathsf{Dec}(k, \mathsf{Enc}(k, m)) = m$.

Defining security for encryption scheme is tricky business. Many of the most obvious security definitions are insufficient:

- **Bad definition:** *"An encryption scheme is secure if it is infeasible for an attacker to recover the plaintext message given only a ciphertext.* This definition admits encryption schemes in which the ciphertext leaks half of the plaintext bits.
- **Bad definition:** *"An encryption scheme is secure if it is infeasible for an attacker to recover any bit of the plaintext message given only a ciphertext.* This definition admits encryption schemes in which the ciphertext leaks the parity of the plaintext bits.

The starting-point (weak) security definition we will use is called *indistinguishability under adaptive chosen plaintext attack* (IND-CPA).

Intuitively, a scheme is CPA-secure if an attacker cannot tell which of two chosen messages are encrypted, even after seeing encryptions of many attacker-chosen messages.

**Definition 2.2** (Encryption Scheme, CPA Security (weak))**.** Formally, an encryption scheme $(\mathsf{Enc}, \mathsf{Dec})$ over message space $\mathcal{M}$ and key space $\mathcal{K}$ is CPA-secure if all efficient adversaries win the following game with probability at most $\frac{1}{2}$ + "negligible:"

- The challenger samples $b \xleftarrow{R} \{0,1\}$ and $k \xleftarrow{R} \mathcal{K}$.
- Polynomially many times: *// Chosen-plaintext queries*
  - The adversary sends the challenger a message $m_i \in \mathcal{M}$
  - The challenger replies with $c_i \leftarrow \mathsf{Enc}(k, m_i)$.
- The adversary then sends two messages $m_0^*, m_1^* \in \mathcal{M}$ to the challenger. (We require $|m_0^*| = |m_1^*|$.)
- The challenger replies with $c^* \leftarrow \mathsf{Enc}(k, m_b^*)$.
- The adversary outputs a value $b' \in \{0,1\}$. The adversary wins if $b = b'$.

Standard encryption systems do not hide the length of the message being encrypted. So, if the message space $\mathcal{M}$ contains messages of different lengths, our security definition requires the adversary to distinguish the encryption of two messages of the *same* length.

One potentially surprising consequence of the CPA-security definition for encryption schemes is:

*For an encryption scheme to be secure in any meaningful sense, the encryption algorithm* ***must*** *be randomized.*

In contrast, secure MACs can be—and typically are—deterministic!

If the encryption algorithm is deterministic (i.e., not randomized), an attacker can win the CPA security game in Definition 2.2 with probability 1. To do so:

- The attacker first asks for encryption of a message $m_0$ and receives a ciphertext $c_0$.
- Then, the attacker attacker choose a message $m_1 \neq m_0$ and sends $(m_0^*, m_1^*) = (m_0, m_1)$ to the challenger and receives the challenge ciphertext $c^*$.
- If $c^* = c_0$, the attacker outputs 0. Otherwise, the attacker outputs 1.

Deterministic encryption schemes are not only broken in theory, they are also broken in practice. For example, if you encrypt the pixels of an image using a deterministic encryption scheme, the encrypted image essentially reveals the plaintext image.

*Why CPA security is a "weak" form of security* There are two reasons why CPA security is a weak or insufficient definition of security for an encryption scheme:

- First, the CPA security definition guarantees nothing about message *integrity*: an attacker can modify the ciphertext and potentially change the meaning of the encrypted message in a meaningful way (even if the attacker does not know the encrypted message!).
- Second, the CPA security definition guarantees nothing in the event that the adversary can obtain *decryptions* of ciphertexts of its choosing. In practice, attackers can often obtain decryptions of chosen ciphertexts. for example, in a system where a client sends encrypted messages to a server and the server does something in response, an attacker can send encrypted queries to the server and observe its behavior to learn some function of the decrypted contents of the message. Later on, we will expand our definition to include these *chosen ciphertext attacks*.

## 3 *One-time Pad*

The one-time pad is perhaps the simplest encryption scheme. Its keyspace, message space, and ciphertext space are all the set of $n$-bit strings. The algorithms are then:

- $\mathsf{Enc}(k, m) \to c$. Compute $c \leftarrow (k \oplus m)$.
- $\mathsf{Dec}(k, c) \to m$. Compute $m' \leftarrow (k \oplus c)$.

The encryption scheme is correct since $\mathsf{Dec}(k, \mathsf{Enc}(k, m)) = k \oplus (k \oplus m) = m$. A less obvious fact is that it is also *one-time secure*: if an attacker sees only one message with encrypted a one-time-pad key $k$, it learns nothing about the underlying plaintext.

The one-time pad is actually one-time secure in a very strong sense: it protects confidentiality even against a computationally unbounded attacker—one that can perform arbitrary amounts of computation.

*The "two-time pad" attack* The one-time pad is insecure if the same key $k$ is every used to encrypt two messages. In particular, if two ciphertexts are ever computed using the same key, we have:

$$\begin{aligned} c_1 &= k \oplus m_1 \\ c_2 &= k \oplus m_2 \\ c_1 \oplus c_2 &= (k \oplus m_1) \oplus (k \oplus m_2) = m_1 \oplus m_2. \end{aligned}$$

The attacker then learns the XOR of the two encrypted messages, which is often enough to leak all sorts of sensitive information about the plaintext. For example, if the attacker knows some bits of $m_1$, it can learn some bits of $m_2$.

*Why the one-time pad is useful* The one-time pad encryption scheme feels in some sense useless: if a secure channel exists through which

Alice and Bob can exchange an $n$-bit secret key, they may as well use that channel to exchange the message itself! There is some merit still in the one-time pad: it is possible to exchange the key ahead of time, and then send encrypted messages later on. Diplomats indeed used the one-time pad in this way throughout the 20th century. They would exchange huge books of keying material (random strings) and then use these keys to communicate securely over long distances.

In practice, it is much more convenient to be able to exchange a *short* key and then use it to encrypt many *long* messages.

## 4 A Weak Encryption Scheme

What we effectively need is a way to generate many bits of random-looking keys (i.e., keys for the one-time pad encryption scheme) from a single short random string.

To generate this, we can use a pseudorandom function! In particular, we would like a pseudorandom function of the form $F\colon \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$.

If you forget what a pseudorandom function is, refer back to **??**.

In practice, we will use AES or another block cipher as a pseudorandom function, so we will have $n = 128$ or so.

Using such a pseudorandom function, we can construct a new encryption scheme that replaces the truly random keys with pseudorandom keys generated from the pseudorandom function. The keyspace and message space for this encryption space are $\{0,1\}^n$ and the ciphertext space is $\{0,1\}^{2n}$. The algorithms are:

- $\mathsf{Enc}(k, m) \to c$:
  - Sample a random value $r \xleftarrow{\text{R}} \{0,1\}^n$. We call this the "nonce."
  - Compute a one-time key $k \leftarrow F(k, r)$ using the pseudorandom function.
  - Use this key to compute the ciphertext using the one-time pad: output $c \leftarrow (r, k \oplus m) \in \{0,1\}^{2n}$.
- $\mathsf{Dec}(k, c)$:
  - Parse the ciphertext $c$ into $(r, c')$.
  - Compute $m \leftarrow c' \oplus F(k, r)$.

Correctness holds by construction. The security argument goes in three steps:

While this encryption scheme is CPA-secure, it provides no message integrity. For any string $\Delta \in \{0,1\}^n$, an attacker can modify a ciphetext $(r, c')$ to $(r, c' \oplus \Delta)$. This ciphertexts now decrypts to the original message XORd with the attacker-chosen string $\Delta$.

By the Birthday Paradox, after encrypting $T$ messages, the probability of a repeated $r$ value is $O(T^2/2^n)$, which is negligible in the key length $n$.

- First, we argue that if $n$ is large enough, the probability that the encryption algorithm ever chooses the same $r$ value twice is negligible.
- Second, we argue that as long as the encryption algorithm never chooses the same $r$ value twice, we can replace the values $F(k, r)$

with truly random strings. By the security of the pseudorandom function, the adversary will not notice this change.

- At this point, we can appeal to the security of the one-time pad scheme to argue that the adversary has no chance of winning the security game (Definition 2.2).

For security to hold, it is crucial that the probability that the encryption algorithm uses the same encryption nonce $r$ twice be negligibly small. If the encryption algorithm ever selects the same random nonce $r$ twice, the pad $F(k, r)$ will be identical in two ciphertexts. An attacker can then apply the two-time-pad attack to recover the XOR of the two messages.

By the Birthday Paradox, if we sample the nonce from a space of size $2^{128}$, we can expect a collision in 128-bit random values once around $2^{64}$ have been generated. Therefore, any single encryption-key must be used for $\ll 2^{64}$ messages. Cryptographic standards typically limit the number of bytes that users can encrypt with the same key to prevent these sorts of problems.

## 5 *Encrypting longer messages: Counter mode*

The CPA-secure encryption scheme of Section 4 only allowed encrypting messages of a fixed length. We now show how to use *counter-mode encryption* to extend this scheme to support messages of arbitrary length. That is, we will construct an encryption scheme $\mathsf{Enc}\colon \mathcal{K} \times \{0,1\}^* \to \{0,1\}^*$ for messages of any length.

In practice, encryption schemes place some (large) bound on the length of encrypted messages. For example, the AES-GCM cipher has a maximum message limit of just under 64 GiB.

Counter-mode encryption works much as the CPA-secure encryption scheme we have already seen in Section 4, except that we split the message into blocks and encrypt each block separately.

We will use the function $\mathsf{ToString}\colon \{0, \ldots, 2^n - 1\} \to \{0,1\}^n$, which converts an integer in $\{0, \ldots, 2^n - 1\}$ to an $n$-bit string in the natural way.

The encryption scheme uses a pseudorandom function $F\colon \mathcal{K} \times \{0,1\}^n \to \{0,1\}^n$. The secret encryption key is a key $k \in \mathcal{K}$ for the pseudorandom function. To encrypt a message, we choose a fresh random value $r \xleftarrow{R} \{0, \ldots, 2^n - 1\}$ and XOR the $i$th block of the message with the value $F(k, \mathsf{ToString}(r + i \bmod 2^n))$.

- $\mathsf{Enc}(k, m)$ :
  - Split the message $m$ into blocks of $n$ bits: $(m_1, m_2, m_3, \ldots, m_\ell)$. The last message block $m_\ell$ may be shorter than $n$ bits.
  - Sample a random nonce $r \xleftarrow{R} \{0, \ldots, 2^n - 1\}$.
  - For $i = 1, \ldots, \ell$: Compute $c_i \leftarrow m_i \oplus F(k, \mathsf{ToString}(r + i \bmod 2^n))$.

The nonce is sometimes called an "initialization vector" or "IV."

(If the last message block $m_\ell$ is less than $n$ bits long, truncate the last ciphertext block $c_\ell$ to the length of $m_\ell$.)

- Output the ciphertext $c = (r, c_1, \ldots, c_\ell)$.

- $\mathsf{Dec}(k, c)$ :
  - Parse the ciphertext $c$ as $(r, c_1, \ldots, c_\ell)$, where all values but the last are exactly $n$ bits long.
  - For $i = 1, \ldots, \ell$: Compute $m_i \leftarrow c_i \oplus F(k, \mathsf{ToString}(r + i \bmod 2^n))$. (Truncate $m_\ell$ to the length of $c_\ell$.)
  - Output the message $m = (m_1 \| \ldots \| m_\ell)$.

As long as the space of random values $r$ is large enough to ensure that the encryption routine never evaluates the pseudorandom function $F(k, \cdot)$ on the same input twice, this scheme will be CPA secure.

You may notice that this encryption scheme reveals the length of the encrypted message to the attacker! This indeed is a potential risk, but in some sense it is required: if we were to hide the length of the message, we would need to set some maximum message length and pad it up to this length. If we did this, encrypting a single word would necessarily result in a ciphertext equal in length to the ciphertext of encrypting a movie! This would greatly decrease the practicality of our encryption scheme. For applications where hiding the length is especially important, the messages can be padded to ensure they are all the same length before they are passed to the encryption scheme.

## *References*