1 One-Time Pad

Secure communication is the act of conveying information from a sender to a receiver, while simultaneously hiding it from everyone else. Secure communication is the oldest application of cryptography, and remains the centerpiece of cryptography to this day. After all, the word *cryptography* means "hidden writing" in Greek. So, secure communication is a natural place to start our study of cryptography.

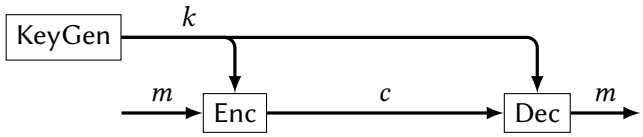
History provides us with roughly 2000 years of attempts to secure sensitive communications in the presence of eavesdroppers. Despite the many brilliant minds that lived during this period of time, almost no useful concepts remain relevant to modern cryptography. In fact, it was not clear how to even *formally define the goal* of secure communication until the 1940s. Today we use security definitions for encryption that were developed only in 1982 and 1990.

The *only* cryptographic method developed before 1900 that has stood the test of time is the **one-time pad**, which appears in some form in essentially every modern encryption scheme. In this chapter, we introduce the one-time pad and discuss its important characteristics. Along the way, we will start to get acclimated to cryptographic security definitions.

1.1 One-Time Pad Definition

For a long time the idea of one-time pad (OTP) was attributed to Gilbert Vernam, a telegraph engineer who patented the scheme in 1919. For that reason, one-time pad is sometimes called "Vernam's cipher." However, an earlier description of one-time pad was recently discovered in an 1882 text on telegraph encryption written by banker Frank Miller.[1]

The idea of one-time pad (and indeed, encryption in general) is for a sender and receiver to initially share a **key** k that was chosen according to a key-generation process that we will call KeyGen. When the sender wishes to securely send a message m (called the **plaintext**) to the receiver, she **encrypts** it using the encryption algorithm Enc and the key k. The result of encryption is a **ciphertext** c, which is sent to the receiver. The receiver can then use the **decryption** algorithm Dec with the same key k to recover the plaintext m.



One-time pad uses keys, plaintexts, and ciphertexts which are all λ-bit strings (*i.e.*, elements of $\{0,1\}^\lambda$). The choice of λ (length of plaintexts, ciphertext, and keys) is a public

[1] Steven M. Bellovin: "Frank Miller: Inventor of the One-Time Pad." *Cryptologia* 35 (3), 2011.

parameter of the scheme, meaning that it does not need to be kept secret. The specific KeyGen, Enc, and Dec algorithms for one-time pad are given below:

Construction 1.1 (One-time pad)

KeyGen:	$\mathsf{Enc}(k, m \in \{0,1\}^\lambda)$:	$\mathsf{Dec}(k, c \in \{0,1\}^\lambda)$:
$k \leftarrow \{0,1\}^\lambda$	return $k \oplus m$	return $k \oplus c$
return k		

Recall that "$k \leftarrow \{0,1\}^\lambda$" means to sample k uniformly from the set of λ-bit strings. The definition of one-time pad mandates that the key should be chosen in exactly this way.

Example *Encrypting the following 20-bit plaintext m under the 20-bit key k using OTP results in the ciphertext c below:*

$$\begin{array}{cll} & 11101111101111100011 & (m) \\ \oplus & 00011001110000111101 & (k) \\ \hline & 11110110011111011110 & (c = \mathsf{Enc}(k,m)) \end{array}$$

Decrypting the following ciphertext c using the key k results in the plaintext m below:

$$\begin{array}{cll} & 00001001011110010000 & (c) \\ \oplus & 10010011101011100010 & (k) \\ \hline & 10011010110101110010 & (m = \mathsf{Dec}(k,c)) \end{array}$$

Note that Enc and Dec are essentially the same algorithm (return the XOR of the two arguments). This results in some small level of convenience and symmetry when implementing one-time pad but you can think of it more as a coincidence than anything fundamental (see Exercises 1.10 & 2.3).

1.2 Properties of One-Time Pad

Correctness

The first property of one-time pad that we would like to verify is that the receiver does indeed recover the intended plaintext when decrypting the ciphertext. Written mathematically:

Claim 1.2 *For all $k, m \in \{0,1\}^\lambda$, it is true that $\mathsf{Dec}(k, \mathsf{Enc}(k,m)) = m$.*

Proof This follows by substituting the definitions of OTP Enc and Dec, along with the properties of XOR listed in Chapter 0.2. For all $k, m \in \{0,1\}^\lambda$, we have:

$$\begin{aligned} \mathsf{Dec}(k, \mathsf{Enc}(k,m)) &= \mathsf{Dec}(k, k \oplus m) \\ &= k \oplus (k \oplus m) \\ &= (k \oplus k) \oplus m \\ &= 0^\lambda \oplus m \\ &= m. \end{aligned}$$

Example *Encrypting the following plaintext m under the key k results in ciphertext c, as follows:*

$$\begin{array}{rll} & \texttt{00110100110110001111} & (m) \\ \oplus & \texttt{11101010011010001101} & (k) \\ \hline & \texttt{11011110101100000010} & (c) \end{array}$$

Decrypting c using the same key k results in the original m:

$$\begin{array}{rll} & \texttt{11011110101100000010} & (c) \\ \oplus & \texttt{11101010011010001101} & (k) \\ \hline & \texttt{00110100110110001111} & (m) \end{array}$$

Security

Suppose you encrypt a plaintext m and an eavesdropper eventually sees the resulting ciphertext. We want to say something like "the eavesdropper doesn't learn about m." We can be quite precise about what exactly the eavesdropper sees in this situation — in fact, the eavesdropper gets an output of the following algorithm:

$\textsc{view}(m \in \{\texttt{0},\texttt{1}\}^\lambda)$:
  $k \leftarrow \{\texttt{0},\texttt{1}\}^\lambda$
  $c := k \oplus m$
  return c

.

This algorithm describes how the sender computes the values using secret values (choosing a key k in a specific way, and using the one-time-pad encryption procedure). It also describes that the eavesdropper sees *only* the ciphertext (but not the key).

This is a *randomized* algorithm, which you can see from the random choice of k. Even after fixing the input m, the output is not fixed. Instead of thinking of $\textsc{view}(m)$ as a fixed value, we will think of it as a *probability distribution*. So more precisely, we can say that an eavesdropper sees a **sample** from the distribution $\textsc{view}(m)$.

Example *Let's take $\lambda = 3$ and work out by hand the distributions $\textsc{view}(\texttt{010})$ and $\textsc{view}(\texttt{111})$. In each case $\textsc{view}$ chooses a value of k uniformly in $\{\texttt{0},\texttt{1}\}^3$ — each of the possible values with probability 1/8. For each possible choice of k, we can compute what the output c will be:*

$\textsc{view}(\texttt{010})$:

Pr	k	$c = k \oplus \texttt{010}$
1/8	000	010
1/8	001	011
1/8	010	000
1/8	011	001
1/8	100	110
1/8	101	111
1/8	110	100
1/8	111	101

$\textsc{view}(\texttt{111})$:

Pr	k	$c = k \oplus \texttt{111}$
1/8	000	111
1/8	001	110
1/8	010	101
1/8	011	100
1/8	100	011
1/8	101	010
1/8	110	001
1/8	111	000

So the distribution $\textsc{view}(\texttt{010})$ assigns probabilty 1/8 to 010, probability 1/8 to 011, and so on.

In this example, notice how every string in $\{0,1\}^3$ appears *exactly once* in the c column of $\text{VIEW}(010)$. This means that VIEW assigns probability $1/8$ to *every* string in $\{0,1\}^3$, which is just another way of saying that the distribution is the *uniform distribution* on $\{0,1\}^3$. The same can be said about the distribution $\text{VIEW}(111)$, too. Both distributions are just the uniform distribution in disguise!

This property of one-time pad generalizes beyond these two specific examples. For any λ and any $m, m' \in \{0,1\}^\lambda$, the distributions $\text{VIEW}(m)$ and $\text{VIEW}(m')$ are identically distributed. Before we prove it, think about this fact from the eavesdropper's point of view. Someone chooses a plaintext m and shows you a sample from the distribution $\text{VIEW}(m)$. But this is a distribution that you can sample from yourself, even if you don't know m. Indeed, you could have chosen an arbitrary m' and run $\text{VIEW}(m')$ yourself, which would have induced the same distribution as $\text{VIEW}(m)$. Truly, the ciphertext that you see (a sample from some distribution) can carry *no information* about m if it is possible to sample from the same ciphertext distribution without even knowing m!

Claim 1.3 *For every $m \in \{0,1\}^\lambda$, the distribution $\text{VIEW}(m)$ is the **uniform distribution** on $\{0,1\}^\lambda$. Hence, for all $m, m' \in \{0,1\}^\lambda$, the distributions $\text{VIEW}(m)$ and $\text{VIEW}(m')$ are identical.*

Proof Arbitrarily fix $m, c \in \{0,1\}^\lambda$. We will calculate the probability that $\text{VIEW}(m)$ produces output c. That event happens only when

$$c = k \oplus m \iff k = m \oplus c.$$

The equivalence follows from the properties of XOR given in Section 0.2. That is,

$$\Pr[\text{VIEW}(m) = c] = \Pr[k = m \oplus c],$$

where the probability is over uniform choice of $k \leftarrow \{0,1\}^\lambda$.

We have fixed specific m and c, so there is *only one* value of k that makes the condition true (encrypts m to c), and that value is exactly $m \oplus c$. Since k is chosen *uniformly* from $\{0,1\}^\lambda$, the probability of choosing the particular value $k = m \oplus c$ is $1/2^\lambda$. ∎

Discussion

- **Isn't there a paradox?** Claim 1.2 says that c can always be decrypted to get m, but Claim 1.3 says that c contains no information about m! The answer to this riddle is that Claim 1.2 talks about what can be done with knowledge of the key k. Claim 1.3 is about the output distribution of the VIEW algorithm, which doesn't include k (see Exercise 1.7). In short, if you know k, then you can decrypt c to obtain m; if you don't know k, then c carries no information about m (in fact, it looks uniformly distributed). This is because m, c, k are all *correlated* in a delicate way.[2]

- **Isn't there another paradox?** Claim 1.3 says that the output of $\text{VIEW}(m)$ doesn't depend on m, but the VIEW algorithm uses its argument m right there in the last line! The answer to this riddle is perhaps best illustrated by the previous examples of $\text{VIEW}(010)$ and $\text{VIEW}(111)$. The two tables of values are indeed different (so the

[2] This correlation is explored further in Chapter 3.

choice of $m \in \{\texttt{010}, \texttt{111}\}$ clearly has some effect), but they represent the *same probability distribution* (since order doesn't matter). Claim 1.3 considers only the resulting probability distribution.

Limitations

The keys in one-time pad are as long as the plaintexts they encrypt. This is more or less unavoidable (see Exercise 2.8) and leads to a kind of chicken-and-egg dilemma: If two users want to securely convey a λ-bit message, they first need to securely agree on a λ-bit string. Additionally, one-time pad keys can be used to securely encrypt only one plaintext (hence, "one-time" pad); see Exercise 1.5. Indeed, we can see that the VIEW subroutine in Claim 1.3 provides no way for a caller to guarantee that two plaintexts are encrypted with the same key, so it is not clear how to use Claim 1.3 to argue about what happens in one-time pad when keys are reused in this way.

Despite these limitations, one-time pad illustrates fundamental ideas that appear in most forms of encryption in this course.

Exercises

1.1. The one-time pad encryption of plaintext `mario` (written in ASCII) under key k is:

$$\texttt{100001000000011101010100000111000001101}.$$

What is the one-time pad encryption of `luigi` under the same key?

1.2. Alice is using one-time pad and notices that when her key is the all-zeroes string $k = \texttt{0}^\lambda$, then $\mathsf{Enc}(k, m) = m$ and her message is sent in the clear! To avoid this problem, she decides to modify KeyGen to exclude the all-zeroes key. She modifies KeyGen to choose a key uniformly from $\{\texttt{0},\texttt{1}\}^\lambda \setminus \{\texttt{0}^\lambda\}$, the set of all λ-bit strings except $\texttt{0}^\lambda$. In this way, she guarantees that her plaintext is never sent in the clear.

Is it still true that the eavesdropper's ciphertext distribution is uniform? Prove or disprove.

1.3. When Alice encrypts the key k itself using one-time pad, the ciphertext will always be the all-zeroes string! So if an eavesdropper sees the all-zeroes ciphertext, she learns that Alice encrypted the key itself. Does this contradict Claim 1.3? Why or why not?

1.4. Describe the flaw in this argument:

> Consider the following attack against one-time pad: upon seeing a ciphertext c, the eavesdropper tries every candidate key $k \in \{\texttt{0},\texttt{1}\}^\lambda$ until she has found the one that was used, at which point she outputs the plaintext m. This contradicts the argument in Section 1.2 that the eavesdropper can obtain no information about m by seeing the ciphertext.

1.5. Suppose Alice encrypts two plaintexts m and m' using one-time pad with the same key k. What information about m and m' is leaked to an eavesdropper by doing this (assume the eavesdropper knows that Alice has reused k)? Be as specific as you can!

1.6. A **known-plaintext attack** refers to a situation where an eavesdropper sees a ciphertext $c = \mathsf{Enc}(k, m)$ and also learns/knows what plaintext m was used to generate c.

(a) Show that a known-plaintext attack on OTP results in the attacker learning the key k.

(b) Can OTP be secure if it allows an attacker to recover the encryption key? Is this a contradiction to the security we showed for OTP? Explain.

1.7. Suppose we modify the subroutine discussed in Claim 1.3 so that it also returns k:

$$\boxed{\begin{array}{l}\underline{\textsc{view}'(m \in \{0,1\}^\lambda):} \\ \quad k \leftarrow \{0,1\}^\lambda \\ \quad c := k \oplus m \\ \quad \text{return } (k, c)\end{array}}.$$

Is it still true that for every m, the output of $\textsc{view}'(m)$ is distributed uniformly in $(\{0,1\}^\lambda)^2$? Or is the output distribution different for different choice of m?

1.8. In this problem we discuss the security of performing one-time pad encryption twice:

(a) Consider the following subroutine that models the result of applying one-time pad encryption with two *independent* keys:

$$\boxed{\begin{array}{l}\underline{\textsc{view}'(m \in \{0,1\}^\lambda):} \\ \quad k_1 \leftarrow \{0,1\}^\lambda \\ \quad k_2 \leftarrow \{0,1\}^\lambda \\ \quad c := k_2 \oplus (k_1 \oplus m) \\ \quad \text{return } c\end{array}}.$$

Show that the output of this subroutine is uniformly distributed in $\{0,1\}^\lambda$.

(b) What security is provided by performing one-time pad encryption twice with *the same key*?

1.9. We mentioned that one-time pad keys can be used to encrypt only one plaintext, and how this was reflected in the VIEW subroutine of Claim 1.3. Is there a similar restriction about re-using *plaintexts* in OTP (without re-using keys)? If an eavesdropper *knows* that the same plaintext is encrypted twice (but doesn't know what the plaintext is), can she learn anything? Does Claim 1.3 have anything to say about a situation where the same plaintext is encrypted more than once?

1.10. In this problem we consider a variant of one-time pad, in which the keys, plaintexts, and ciphertexts are all elements of $\mathbb{Z}_n$ instead of $\{0,1\}^\lambda$.

(a) What is the decryption algorithm that corresponds to the following encryption algorithm?

$$\boxed{\begin{array}{l}\underline{\mathsf{Enc}(k, m \in \mathbb{Z}_n):} \\ \quad \text{return } (k + m) \,\%\, n\end{array}}.$$

(b) Show that the output of the following subroutine is uniformly distributed in $\mathbb{Z}_n$:

$$\boxed{\begin{array}{l}\underline{\textsc{view}'(m \in \mathbb{Z}_n):} \\ \quad k \leftarrow \mathbb{Z}_n \\ \quad c := (k + m) \,\%\, n \\ \quad \text{return } c\end{array}}\ .$$