CPSC 467: Cryptography and Security

Michael J. Fischer

Lecture 4
September 10, 2020

Symmetric Cryptography

Caesar Cipher

One-time pad

Symmetric Cryptography

Secret message transmission problem

Alice wants to send Bob a private message m over the internet.

Eve is an *eavesdropper* who listens in and wants to learn m.

Alice and Bob want m to remain private and unknown to Eve.

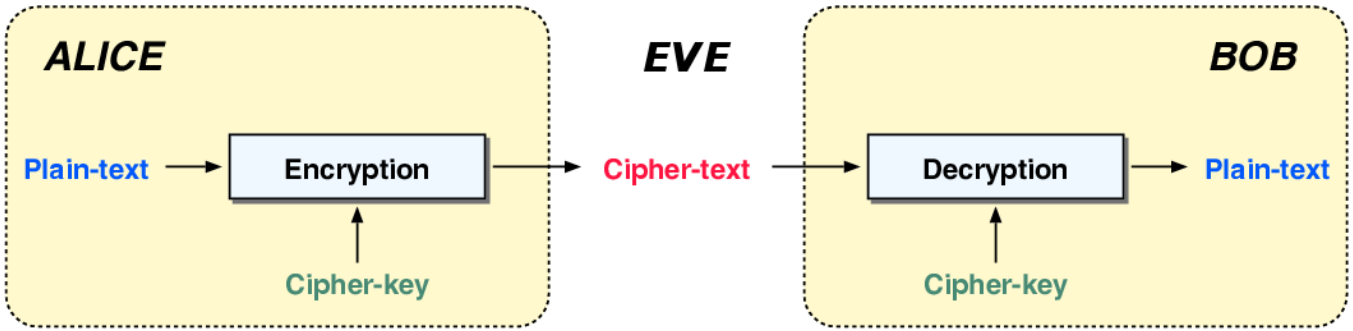


Image credit: Derived from image by Frank Kagan Gürkaynak,
`http://www.iis.ee.ethz.ch/~kgf/acacia/fig/alice_bob.png`

The protocol

Protocol:

1. Alice and Bob share a common *secret key* k.
2. Alice encrypts *plaintext* message m by computing *ciphertext* $c = E(k, m)$.
3. She sends c to Bob.
4. Bob receives c', computes $m' = D(k, c')$, and takes m' to be Alice's message m.

Needed assumptions

For this protocol to work and maintain the privacy that Alice and Bob desire, several assumptions are needed:

- Eve is a *passive eavesdropper*. She can read but not modify c.
- Eve learns nothing else during the protocol except for c.
- The channel is perfect, so $c' = c$.[1]
- $D(c, k) = m$ whenever $c = E(m, k)$. This allows Bob to read Alice's message.

[1] The real world is not so perfect, so we must ask what happens if $c' \neq c$?

Symmetric cryptosystem

A *symmetric cryptosystem* (sometimes called a *private-key* or *one-key* system) consists of

- a set $\mathcal{M}$ of *plaintext messages*,
- a set $\mathcal{C}$ of *ciphertexts*,
- a set $\mathcal{K}$ of *keys*,
- an *encryption* function $E : \mathcal{K} \times \mathcal{M} \to \mathcal{C}$
- a *decryption* function $D : \mathcal{K} \times \mathcal{C} \to \mathcal{M}$.

Notation: We often write $E_k(m)$ to mean $E(k, m)$ and $D_k(c)$ to mean $D(k, c)$.

Basic requirements

Encryption $E_k(m)$ *encrypts plaintext message* m using key k to produce a *ciphertext* c.

Decryption $D_k(c)$ *decrypts* ciphertext c using k to produce a message m.

Efficiency Both E and D must be efficiently computable.

Correctness $D_k(E_k(m)) = m$ for all keys k and all messages m. Thus, D_k is the *inverse* of E_k.

Security Given $c = E_k(m)$, it is hard to find m without knowing k.

Security assumptions

Given a cryptosystem, what assumptions are needed when it is used?

- Eve does not know k.
- k is chosen randomly from a large key space $\mathcal{K}$, so Eve can not guess k with better than a negligible success probability.
- Alice and Bob successfully keep k secret, e.g., their computers have not been compromised, Eve can't find k on their computers even if she is a legitimate user, etc.)
- Eve can't obtain k in other ways, e.g., by social engineering, using binoculars to watch Alice or Bob's keyboard, etc.

What's missing from these definitions?

These definitions are vague about three important questions.

1. What is a "feasible" amount of time and storage for computing E and D?
2. What does it mean to be "hard" to find m without knowning k?
3. What does it mean to "know" k?

Practical considerations

These questions are all critical in practice.

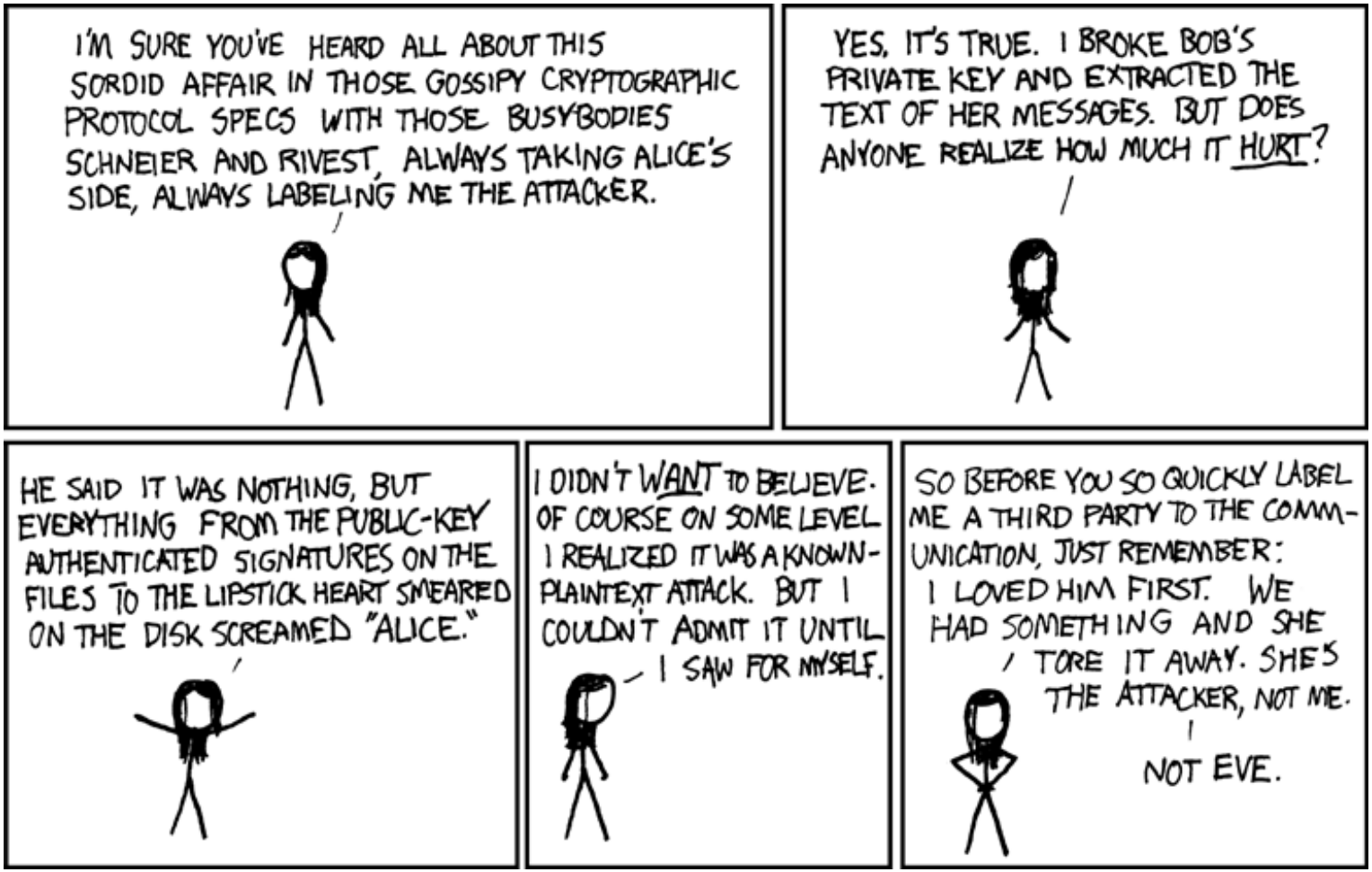
1. E and D must be practically computable by Alice and Bob or the cryptosystem can't be used. For most applications, this means computable in milliseconds, not minutes or days.
2. The confidentiality of m must withstand a sustained attack by Eve, possibly for years, after she discovers c. How long is long enough?
3. The only way to be certain that Eve does not know k is to choose k at random from an independent random source to which Eve has no access. This is easy to get wrong.

Key Management

Managing keys presents several difficulties:

1. Who chooses the key? Alice? Bob? Jointly? How do they both arrive at the same key?
2. How do Alice and Bob keep their copies of the key secret?
3. If Eve ever discovers the key, then she can decrypt both past messages and future ones.
4. Once Alice and Bob share the key, then Bob can successfully impersonate Alice by presenting the shared key as hers. This requires a level of trust between Alice and Bob that might not always be present.

Eve's side of the story



Cartoon by Randall Munroe, `https://www.xkcd.com/177/`

Caesar Cipher

Caesar cipher

The Caesar Cipher is said to go back to Roman times.

The 26 letters of the Roman alphabet $A, B, \ldots, Z$ are represented by successive numbers $A = 0$, $B = 1$, $\ldots$, $Z = 25$.

The *Basic Caesar Cipher* is formally defined only for 1-letter messages.

$$\mathcal{M} = \mathcal{C} = \mathcal{K} = \{0, \ldots, 25\}.$$

$$E_k(m) = (m + k) \bmod 26.$$

$$D_k(c) = (c - k) \bmod 26.$$

Encoding longer messages

To encrypt longer messages, the Basic Caesar Cipher is applied to each letter of the message in turn, *using the same key* for each.

Key	6											
Plaintext	a	t	t	a	c	k	a	t	d	a	w	n
Encoding	0	19	19	0	2	10	0	19	3	0	22	13
Encryption	6	25	25	6	8	16	6	25	9	6	2	19
Ciphertext	G	Z	Z	G	I	Q	G	Z	J	G	C	T
Decoding	0	19	19	0	2	10	0	19	3	0	22	13
Message	a	t	t	a	c	k	a	t	d	a	w	n

Note that it is customary to ignore spaces in both plaintext and ciphertext. Also, plaintext and ciphertext are often represented by their corresponding letters, either upper case or lower case, rather than by numbers.

The Full Caesar Cipher

Using the Basic Caesar Cipher repeatedly for each letter of a message gives a derived cipher that operates on sequences of letters, called the *Full Caesar Cipher*.

The message and ciphertext spaces $\mathcal{M}'$ and $\mathcal{C}'$ are each the set of all sequences of numbers in $\mathcal{M} = \mathcal{C}$. The key space $\mathcal{K}'$ is the same as for the Basic Caesar Cipher, $\mathcal{K}$.

Let $\bar{m} = m_1 \dots m_r \in \mathcal{M}'$ be a message. Define

$$E'_k(\bar{m}) = E_k(m_1) \dots E_k(m_r)$$

Similarly, let $\bar{c} = c_1 \dots c_r \in \mathcal{C}'$ be a ciphertext. Define.

$$D'_k(\bar{c}) = D_k(c_1) \dots D_k(c_r).$$

A brute force attack on the Caesar cipher

Ciphertext	`HWWXE UXWH`
Decryption key	`Plaintext`
$k = 0$	`hwwxe uxwh`
$k = 1$	`gvvwd twvg`
$k = 2$	`fuuvc svuf`
$k = 3$	`ettub rute`
$k = 4$	`dssta qtsd`
$k = 5$	`crrsz psrc`
...	...

Which is the correct key?

Recognizing the correct key

Caesar's last words, "Et tu, Brute?"

[From William Shakespeare's play, *Julius Caesar*, Act 3, Scene 1.]

Ciphertext	`HWWXE UXWH`
Decryption key	`Plaintext`
$k = 0$	`hwwxe uxwh`
$k = 1$	`gvvwd twvg`
$k = 2$	`fuuvc svuf`
$k = 3$	`ettub rute`
$k = 4$	`dssta qtsd`
$k = 5$	`crrsz psrc`
...	...

How do you know when you've found the correct key?

You don't always know!

Suppose you intercept the ciphertext JXQ.
You quickly discover that $E_3(\text{GUN}) = \text{JXQ}$.
But is $k = 3$ and is GUN the correct decryption?

You then discover that $E_{23}(\text{MAT}) = \text{JXQ}$.
Now you are in a quandary. Which decryption is correct?

Have you broken the system or haven't you?

You haven't found the plaintext for sure, but you've reduced the possibilities down to a small set.

Terminology

A *shift cipher* uses a letter substitution defined by a rotation of the alphabet.

Any cipher that uses a substitution to replace a plaintext letter by a ciphertext letter is called a *substitution cipher*. A shift cipher is a special case of a substitution cipher where the substitution is defined by a rotation of the alphabet.

Any cipher that encrypts a message by applying the same substitution to each letter of the message is called a *monoalphabetic* cipher.

One-time pad

Vernam cipher

The *Vernam cipher* (one-time pad) is an *information-theoretically secure* cryptosystem.

This means that Eve, knowing only the ciphertext, can extract absolutely no information about the plaintext other than its length.

Exclusive-or on bits

The Vernam cipher is based on *exclusive-or* (XOR), which we write as $\oplus$.

$x \oplus y$ is true when exactly one of x and y is true.

$x \oplus y$ is false when either x and y are both true or are both false.

Exclusive-or is just sum modulo two if 1 represents true and 0 represents false.

$$x \oplus y = (x + y) \bmod 2.$$

XOR is associative and commutative. 0 is the identity element.

$$k \oplus 0 = 0 \oplus k = k$$

XOR is its own inverse.

$$k \oplus k = 0$$

Informal description

The one-time pad encrypts a message m by XORing it with the key k, which must be as long as m.

Assume both m and k are represented by strings of bits. Then ciphertext bit $c_i = m_i \oplus k_i$.

Note that $c_i = m_i$ if $k_i = 0$, and $c_i = \neg m_i$ if $k_i = 1$.

Decryption is the same, i.e., $m_i = c_i \oplus k_i$.

The one-time pad cryptosystem formally defined

$\mathcal{M} = \mathcal{C} = \mathcal{K} = \{0,1\}^r$ for some length r.

$E_k(m) = D_k(m) = k \oplus m$, where $\oplus$ is applied to corresponding bits of k and m.

It works because

$$D_k(E_k(m)) = k \oplus (k \oplus m) = (k \oplus k) \oplus m = 0 \oplus m = m.$$

Security

Like the Basic Caesar cipher, for given m and c, there is *exactly one* key k such that $E_k(m) = c$ (namely, $k = m \oplus c$).

For fixed c, m varies over all possible messages as k ranges over all possible keys, so c gives no information about m.

The one-time pad is said to be information-theoretically secure.

What more is there to prove?

Importance of the Vernam cipher

It is important because

- it is sometimes used in practice;
- it is the basis for many *stream ciphers*, where the truly random key is replaced by a pseudo-random bit string.

Attraction of one-time pad

The one-time pad would seem to be the perfect cryptosystem.

- It works for messages of any length (by choosing a key of the same length).
- It is easy to encrypt and decrypt.
- It is information-theoretically secure.

In fact, it is sometimes used for highly sensitive data.

Drawbacks of one-time pad

It has two major drawbacks:

1. The key k must be as long as the message to be encrypted.
2. The same key must never be used more than once. (Hence the term "one-time".)

Together, these make the problem of key distribution and key management very difficult.

Why the key cannot be reused

If Eve knows just one plaintext-ciphertext pair (m_1, c_1), then she can recover the key $k = m_1 \oplus c_1$.
This allows her to decrypt all future messages sent with that key.

Even in a ciphertext-only situation, if Eve has two ciphertexts c_1 and c_2 encrypted by the same key k, she can gain significant partial information about the corresponding messages m_1 and m_2.

In particular, she can compute $m_1 \oplus m_2$ without knowing either m_1 or m_2 since

$$m_1 \oplus m_2 = (c_1 \oplus k) \oplus (c_2 \oplus k) = c_1 \oplus c_2.$$

How knowing $m_1 \oplus m_2$ might help an attacker

Fact (important property of $\oplus$)

For bits b_1 and b_2, $b_1 \oplus b_2 = 0$ if and only if $b_1 = b_2$.

Hence, blocks of 0's in $m_1 \oplus m_2$ indicate regions where the two messages m_1 and m_2 are identical.

That information, together with other information Eve might have about the likely content of the messages, may be enough for her to seriously compromise the secrecy of the data.