CPSC 367: Cryptography and Security

Michael J. Fischer

Lecture 3
January 22, 2019

- Secret Message Transmission
- Symmetric Cryptography
- Caesar cipher
- One-time pad
- Appendix
- Cryptanalysis
    - Breaking the Caesar cipher
    - Brute force attack
    - Letter frequencies
    - Key length
    - Manual attacks
- References

Reading assignment

Read Chapter 1 of Christof Paar and Jan Pelzl,*Understanding Cryptography* [PP10].

Secret Message Transmission

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.

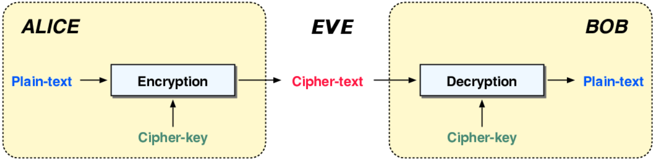


Image credit: Derived from image by Frank Kagan Gürkaynak,
`http://www.iis.ee.ethz.ch/~kgf/acacia/fig/alice_bob.png`

Solution using encryption

A *symmetric cryptosystem* (sometimes called a *private-key* or *one-key* system) is a pair of efficiently-computable functions E and D such that

- $E(k, m)$ *encrypts plaintext message* m using key k to produce a *ciphertext* c.
- $D(k, c)$ *decrypts* ciphertext c using k to produce a message m.

Requirements:

Correctness $D(k, E(k, m)) = m$ for all keys k and all messages m.

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

The protocol

Protocol:

1. Alice and Bob share a common secret key k.
2. Alice computes $c = E(k, m)$ and sends c to Bob.
3. Bob receives c', computes $m' = D(k, c')$, and assumes m' to be Alice's message.

Assumptions:

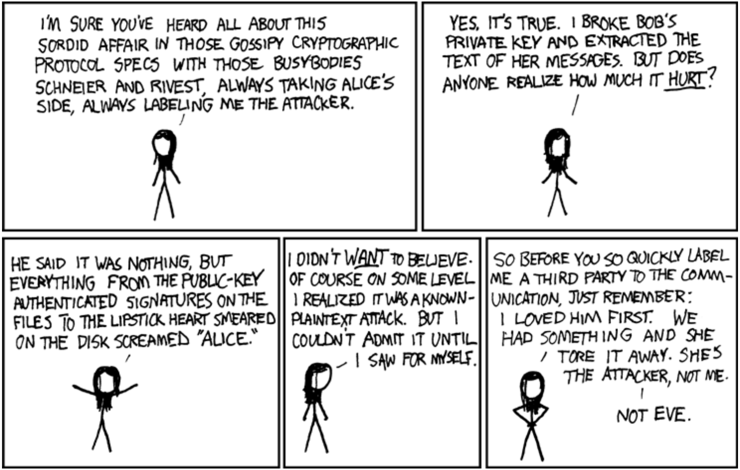
- Eve learns nothing except for c during the protocol.
- The channel is perfect, so $c' = c$. The real world is not so perfect, so we must ask what happens if $c' \neq c$?
- Eve is a *passive eavesdropper* who can read c but not modify it.

Requirements

What do we require of E, D, and the computing environment?

- Given c, it is hard to find m without also knowing k.
- k is not initially known to Eve.
- Eve can guess k with at most negligible success probability. (k must be chosen randomly from a large key space.)
- Alice and Bob successfully keep k secret. (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.

Eve's side of the story


I'M SURE YOU'VE HEARD ALL ABOUT THIS SORDID AFFAIR IN THOSE GOSSIPY CRYPTOGRAPHIC PROTOCOL SPECS WITH THOSE BUSYBODIES SCHNEIER AND RIVEST, ALWAYS TAKING ALICE'S SIDE, ALWAYS LABELING ME THE ATTACKER.
YES, IT'S TRUE. I BROKE BOB'S PRIVATE KEY AND EXTRACTED THE TEXT OF HER MESSAGES. BUT DOES ANYONE REALIZE HOW MUCH IT HURT?
HE SAID IT WAS NOTHING, BUT EVERYTHING FROM THE PUBLIC-KEY AUTHENTICATED SIGNATURES ON THE FILES TO THE LIPSTICK HEART SMEARED ON THE DISK SCREAMED "ALICE."
I DIDN'T WANT TO BELIEVE. OF COURSE ON SOME LEVEL I REALIZED IT WAS A KNOWN-PLAINTEXT ATTACK. BUT I COULDN'T ADMIT IT UNTIL I SAW FOR MYSELF.
SO BEFORE YOU SO QUICKLY LABEL ME A THIRD PARTY TO THE COMMUNICATION, JUST REMEMBER: I LOVED HIM FIRST. WE HAD SOMETHING AND SHE TORE IT AWAY. SHE'S THE ATTACKER, NOT ME.
NOT EVE.

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

Symmetric Cryptography

Formalizing what a cryptosystem is

A *symmetric cryptosystem* 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 $D : \mathcal{K} \times \mathcal{C} \to \mathcal{M}$.

We often write $E_k(m) = E(k, m)$ and $D_k(c) = D(k, c)$.

Desired properties

Decipherability $\forall m \in \mathcal{M}, \forall k \in \mathcal{K}, D_k(E_k(m)) = m$. In other words, D_k is the left inverse of E_k.

Feasibility E and D, regarded as functions of two arguments, should be computable using a feasible amount of time and storage.

Security (weak) It should be difficult to find m given $c = E_k(m)$ without knowing k.

What's wrong with this definition?

This definition leaves three important questions unanswered?

1. What is a "feasible" amount of time and storage?
2. What does it mean to be "difficult" to find m?
3. What does it mean to not "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 be preserved, possibly for years, after Eve 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 a 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.

Caesar cipher

Encoding single letters

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

It encodes the 26 letters of the Roman alphabet $A, B, \ldots, Z$.

Assume the letters are represented as $A = 0$, $B = 1$, . . . , $Z = 25$.

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

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

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

Formally, we have a cryptosystem for 1-letter messages.

Encoding longer messages

The Caesar cipher encrypts longer messages by encrypting each letter separately.

How do we formalize this?

- What is the message space now?
- What is the ciphertext space?
- What is the key space?
- What is the encryption function?
- What is the decryption function?

Caesar cipher formally defined

For arbitrary strings, we have

$$\mathcal{M}' = \mathcal{C}' = \mathcal{M}^*$$

where $\mathcal{M}^*$ is the transitive closure of $\mathcal{M}$.

That is, $\mathcal{M}^*$ consists of all sequences of 0 or more letters from $\mathcal{M}$.

The encryption and decryption for length-r sequences are

$$E'_k(m_1 \ldots m_r) = E_k(m_1) \ldots E_k(m_r)$$

$$D'_k(c_1 \ldots c_r) = D_k(c_1) \ldots D_k(c_r).$$

A brute force attack on the Casear 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 Casear*, 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.

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 1-letter 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.

Appendix

Playfair cipher

The *Playfair* cipher, invented by Charles Wheatstone in 1854 but popularized by Lord Lyon Playfair, is another example of a polygraphic cipher [MvOV96, chapter 7, pp. 239-240] and [Wik].

Here, the key is a passphrase from which one constructs a 5×5 matrix of letters. Pairs of plaintext letters are then located in the matrix and used to produce a corresponding pair of ciphertext letters.

How Playfair works

Construct the matrix from the passphrase.

- Construct the matrix by writing the passphrase into the matrix cells from left to right and top to bottom.
- Omit any letters that have previously been used.
- Fill remaining cells with the letters of the alphabet that do not occur in the passphrase, in alphabetical order.
- In carrying out this process, “I” and “J” are identified, so we are effectively working over a 25-character alphabet.

Thus, each letter of the 25-character alphabet occurs exactly once in the resulting matrix.

Example Playfair matrix

Let the passphrase be

> *“CRYPTOGRAPHY REQUIRES STRONG KEYS”.*

The resulting matrix is

C	R	Y	P	T
O	G	A	H	E
Q	U	I/J	S	N
K	B	D	F	L
M	V	W	X	Z

First occurrence of each letter in the passphrase shown in orange:

> *“CRYPTOGRAPHY REQUIRES STRONG KEYS”.*

Letters not occurring in the passphrase: BDFLMVWXZ.

Encrypting in Playfair: preparing the message

To encrypt a message using Playfair:

- Construct the matrix.
- Remove spaces and pad the message with a trailing 'X', if necessary, to make the length even.
- Break up the message into pairs of letters.
- In case a pair of identical letters is about to be produced, insert an "X" to prevent that.

Examples:

- "MEET ME AT THE SUBWAY" becomes "ME" "ET" "ME" "AT" "TH" "ES" "UB" "WA" "YX".
- "A GOOD BOOK" becomes "AG", "OX", "OD" "BO", "OK".

Encrypting in Playfair: substituting the pairs

To encrypt pair *ab*, look at rectangle with *a* and *b* at its corners.

1. If *a* and *b* appear in different rows and different columns, replace each by the letter at the opposite end of the corresponding row. Example: replace “AT” by “EY”:

Y	P	**T**
A	H	E

2. If *a* and *b* appear in the same row, then replace *a* by the next letter circularly to its right in the row, and similarly for *b*. For example, the encryption of “LK” is “KB”.
3. If *a* and *b* appear in the same column, then replace *a* by the next letter circularly down in the column, and similarly for *b*.

Example: “MEET ME AT THE SUBWAY” encrypts as “ZONEZOEYPEHNBVYIPW”.

Decrypting in Playfair

Decryption is by a similar procedure.

In decrypting, one must manually remove the spurious occurrences of "X" and resolve the "I/J" ambiguities.

See Trappe and Washington [TW06] or Wikipedia [Wik] for a discussion of how the system was successfully attacked by French cryptanalyst Georges Painvin and the Bureau du Chiffre.

Cryptanalysis

Breaking the Caesar: A brute force attack

We saw last time an example of breaking the Caesar cipher using a brute force attack.

Brute force attack means trying every possible key to see which one "works".

Determining which is the correct key is the problem.

For our Caesar cipher example, there were only 26 possible keys; hence only 26 possible decryptions of the given ciphertext `HWWXE UXWH`, only one of which "makes sense".

Breaking the Caesar cipher: Extending these ideas

The longer the correct message, the more likely that only one key results in a sensible decryption.

For example, suppose the ciphertext were “EXB JXQ”.
We saw two possible keys for “JXQ” — 3 and 23.
Trying them both we get:

$$k = 3\text{:}\ \ D_3(\text{EXB JXQ}) = \text{BUY GUN.}$$

$$k = 23\text{:}\ \ D_{23}(\text{EXB JXQ}) = \text{HAE MAT.}$$

Latter is nonsense, so we know $k = 3$ and the message is “BUY GUN”.

Breaking the Caesar cipher: Conclusion

Let n be the message length.

$n = 1$: The Caesar cipher is information-theoretically secure!

$n > 1$: The Caesar cipher is only partially secure or completely breakable, depending on message length and redundancy present in the message.

How long is long enough for a brute force attack to succeed?

There is a whole theory of redundancy of natural language that allows one to calculate a number called the “unicity distance” for a given cryptosystem. If a message is longer than the unicity distance, there is a high probability that it is the only meaningful message with a given ciphertext and hence can be recovered uniquely, as we were able to recover “BUY GUN” from the ciphertext “EXB JXW” in the example. See [Sti06, section 2.6] for more information on this interesting topic.

Trying all keys

A *brute force attack* can be attempted against any cryptosystem.

It tries all possible keys k. It works against the Caesar cipher because the key space is so small.

For each k, Eve computes $m_k = D_k(c)$ and tests if m_k is meaningful. If exactly one meaningful m_k is found, she knows that $m = m_k$.

Given long enough messages, the Caesar cipher is easily broken by brute force—one simply tries all 26 possible keys to see which leads to a sensible plaintext.

What is long enough?

Automating brute force attacks

With modern computers, it is quite feasible for an attacker to try millions ($\sim 2^{20}$) or billions ($\sim 2^{30}$) of keys.

The attacker also needs an automated test to determine when she has a likely candidate for the real key.

How does one write a program to distinguish valid English sentences from gibberish?

One could imagine applying all sorts of complicated natural language processing techniques to this task. However, much simpler techniques can be nearly as effective.

Random English-like messages

Consider random messages whose letter frequencies are similar to that of valid English sentences.

For each letter b, let p_b be the probability (relative frequency) of that letter in normal English text.

A message $m = m_1 m_2 \dots m_r$ has probability $p_{m_1} \cdot p_{m_2} \cdots p_{m_r}$.

This is the probability of m being generated by the simple process that chooses r letters one at a time according to the probability distribution p.

Determining likely keys

Assume Eve knows that $c = E_k(m)$, where m was chosen randomly as described above and k is uniformly distributed.

Eve easily computes the 26 possible plaintext messages $D_0(c), \ldots, D_{25}(c)$, one of which is correct.

To choose which, she computes the conditional probability of each message given c, then picks the message with the greatest probability.

This guess will not always be correct, but for letter distributions that are not too close to uniform (including English text) and sufficiently long messages, it works correctly with very high probability.

How long should the keys be?

The DES (Data Encryption Standard) cryptosystem (which we will talk about next week) has 56-bit keys for a key space of size 2^{56}.

A special DES Key Search Machine was built as a collaborative project by Cryptography Research, Advanced Wireless Technologies, and EFF. (Click here for details.)

This machine was capable of searching 90 billion keys/second and discovered the RSA DES Challenge key on July 15, 1998, after searching for 56 hours. The entire project cost was under $250,000.

Now, 15+ years later, the same task could likely be done on a commercial cluster computer such as Amazon's Elastic Compute Cloud (EC2) at modest cost.

What is safe today and into the future?

DES with its 56-bit keys offers little security today.

80-bit keys were considered acceptable in the past decade, but in 2005, NIST proposed that they be used only until 2010.

Triple DES (with 112-bit keys) and AES (with 128-bit keys) will probably always be safe from brute-force attacks (but not necessarily from other kinds of attacks).

Quantum computers, if they become a reality, would cut the effective key length in half (see Wikipedia "key size"), so some people recommend 256-bit keys (which AES supports).

Cryptography before computers

Large-scale brute force attacks were not feasible before computers.

While Caesar is easily broken by hand, clever systems have been devised that can be used by hand but are surprisingly secure.

Attacks on any monoalphabetic ciphers

The Caesar cipher uses only the 26 rotations out of the 26! permutations on the alphabet. The *monoalphabetic cipher* uses them all. A key k is an arbitrary permutation of the alphabet. $E_k(m)$ replaces each letter a of m by $k(a)$ to yield c. To decrypt, $D_k(c)$ replaces each letter b of c by $k^{-1}(b)$.

The size of the key space is $|\mathcal{K}| = 26! > 2^{74}$, large enough to be moderately resistant to a brute force attack.

Nevertheless, monoalphabetic ciphers can be readily broken using letter frequency analysis, given a long enough message.

This is because *monoalphabetic ciphers preserve letter frequencies*.

How to break monoalphabetic ciphers

Each occurrence of a in m is replaced by $k(a)$ to get c.
Hence, if a is the most frequent letter in m, $k(a)$ will be the most frequent letter in c.

Eve now guesses that a is one of the most frequently-occurring letters in English, i.e., ‘e’ or ‘t’.
She then repeats on successively less frequent ciphertext letters.

Of course, not all of these guesses will be correct, but in this way the search space is vastly reduced.

Moreover, many wrong guesses can be quickly discarded even without constructing the entire trial key because they lead to unlikely letter combinations.

Why can't one break the one-time pad?

For the one-time pad on n-bit messages and keys, there are 2^n possible keys.

For any fixed ciphertext c, every n-bit message is a possible decryption of c.

This completely masks all letter frequency information from the ciphertext.

References

Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone.
Handbook of Applied Cryptography.
CRC Press, 1996.

Christof Paar and Jan Pelzl.
Understanding Cryptography.
Springer-Verlag, Berlin Heidelberg, 2010.

Douglas R. Stinson.
Cryptography: Theory and Practice.
Chapman & Hall/CRC, third edition, 2006.
ISBN-10: 1-58488-508-4; ISBN-13: 978-58488-508-5.

References (cont.)

Wade Trappe and Lawrence C. Washington.
Introduction to Cryptography with Coding Theory.
Prentice Hall, second edition, 2006.
ISBN 0-13-186239-1.

Wikipedia.
Playfair cipher.
URL http://en.wikipedia.org/wiki/Playfair_cipher.