

CTF Training Camp for Hackers



Crypto 101: Introduction to Cryptography and Cryptanalysis

Cousin Wu
October 28, 2021



Open Innovation Lab
CUHK



① Introduction

- Background
- Logistics
- Terminologies

② Classical Ciphers

- Caesar Cipher
- Affine Cipher
- Substitution Cipher
- Vigenère Cipher
- Exercises
- Transposition Cipher

③ Introduction to Modern Block Ciphers

XOR

Exercise

Substitution-Permutation

Network

Meet-in-the-Middle Attack

AES

④ Mode of Operations

Electronic Code Book

Cipher Block Chaining

Padding Oracle

Counter Mode

⑤ Appendix

Modular Arithmetic

Multiplicative inverse



Disclaimer

A horizontal bar with a teal segment on the left and an orange segment on the right.

This slides are modified from the speaker's previous training given about introduction to cryptography and cryptanalysis talk in 2019 and 2020, and the mode of operation part is added.

We will not cover everything here since even the speaker can only get to page 67 in two hours (three hours when more time are allocated to do exercises together), and the speaker plan to focus more on the block cipher part.



What is Cryptography?

Cryptography is the study of secure communication.

Example

Encryption is a way to prevent anyone other than the receiver who get hold of the message to get the actual content.

Cryptography emerged thousands of years ago! Julius Caesar used [his cipher](#) to protect the message :) (will talk about this later). Some classical ciphers are far from trivial.

The key thing is: **cryptography focuses on creating secure cryptosystems.**



Are there any cryptography besides encryption?

Of course yes! E.g. Key exchange, digital signature, cryptographic hashes, secret sharing, zero-knowledge proofs...


On the other hand, one almost never uses a **cryptographic primitive** or **cryptosystem** by itself. Many components are combined to create a complex **cryptographic protocol** to achieve certain **security services**.

But we will only cover encryption. Or is it?

Yes. But maybe we can discuss those in the future :)



History (No one cares)

A horizontal bar with a teal segment on the left and an orange segment on the right.

Around the second world war, modern cryptography that made use of mechanical machinery made things more complicated. Enigma machine was used by the Nazi German! The computer science god Alan Turing helped breaking it. AES, one of the most wide used symmetric key encryption scheme, was created in 1997 only! Not much older than you and me!

As you can see, cryptography was, and is still being used, and considered as munitions. (PGP)



Cryptanalysis

Cryptanalysis is the study of **breaking cryptography**. We can either attack the protocol itself, which are usually difficult, or attack implementations of such protocols (e.g. bad random number generation, bad prime number generation. etc.).

Example (Brute Force)

We could try out all the possible ways to decrypt a message, one of the ways would be the correct way!

This is what we will be doing in CTFs. By the way,

Definition

Cryptology is the study of both **cryptography** and **cryptanalysis**.



Never Implement Your Own Crypto

NEVER implement your own crypto!!
NEVER implement your own crypto!!
NEVER implement your own crypto!!

If something is important, you gotta say it three times.



Why?



- ▶ Commonly used ciphers are implemented by others already.
(fast and secure enough)
- ▶ Need to exactly follow the protocol.
- ▶ Side-channel attacks! (Attacks on [AES](#), [RSA](#) by youtuber/ctf player liveoverflow)
- ▶ Most ciphers are tested and attacked by thousands of academics before considered safe enough!



Why Crypto in CTF?



- ▶ Important in information security
- ▶ Any data transfer we do on the internet is protected by crypto!
- ▶ Badly implemented crypto appears in real life!
- ▶ It is fun! :)



Prerequisites?

Do I need a lot of math?

You need some, but not a lot.

What courses have you taken?

1. Discrete Math?
2. Cybersecurity/Cryptography course?
3. Number Theory?
4. Courses training math maturity?
5. Abstract Algebra?

If you have taken (or are taking) those, then great! If not, don't worry! We will develop the necessary knowledge as we needed them. I would say that discrete math will be the most helpful here.



There are some actual requirements though.
You need to use python (with pwntools!). You need to install some scientific computation library and crypto library (**required**):
gmpy2, pycrypto. Install with

```
python3 -m pip install pycryptodome --user  
sudo apt install python-gmpy2
```

[Sage](#), a computer algebra system, would be helpful later down the line. (The size of sage is massive (GB's)! You could use the [online one](#) instead.)



Brief Overview of Crypto Training

A horizontal bar with a teal segment on the left and an orange segment on the right.


We will have one more training on crypto later.

- ▶ Symmetric Key Cryptography (1 key)
- ▶ Public Key Cryptography (2 keys, One for encryption and one for decryption)

There are also other areas that appears in CTFs, like cryptographic hashes, PRNGs, quantum cryptography, post-quantum cryptography, secret sharing,...



Non-Expectation

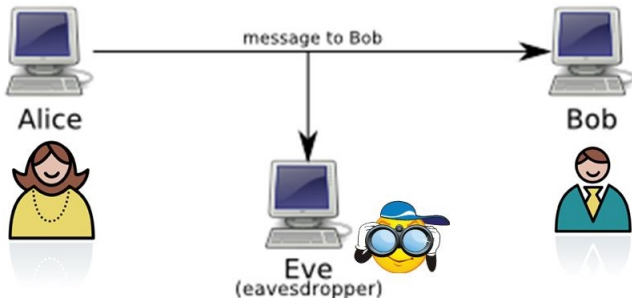
A horizontal bar with a teal left half and an orange right half.

However, we will **NOT** talk about real world cryptosystem and protocols, such as PGP, SSL/TLS, IPsec, VPN, SSH,...

And No, we will **NOT** talk about blockchain in crypto training.



Model



Alice sends messages to Bob through a communication channel, and Eve (us!) can intercept all the messages in the channel. We usually assume/know what encryption/signature/whatever scheme they are using (except for the key of course).



Modern cryptosystems should be secure even if the scheme is known! This is known as the Kerckhoffs's principle. As a corollary, **please do not design a proprietary/secret cryptosystem.** **Security through obscurity** is bad!



Encryption

We have already talked about encryption here. But what is encryption?

Definition (Encryption)

Encryption and decryption are (family of) functions E_k and $D_{k'}$ with key k, k' such that $D_{k'}(E_k(m)) = m$ for all messages m supported by the encryption scheme.

If k' is known, it should be easy to decrypt a message. Conversely, if k' is not known, it should be very difficult (impossible) to decipher. Same goes for encryption and k .

In simpler language, **if you encrypt and then decrypt a message, you should get back the message.**



How to **decipher** the definition of encryption? Lets look at non-examples of encryptions: (treating x as numbers, x can be any real number)

1. $E_k(x) = 0$
2. $E_k(x) = x$
3. $E_k(x) = \sin(x^k)$

Why are these not encryptions?



Definition

A message to be encrypted is known as the **plaintext**, while if a message is encrypted, the result is known as the **ciphertext**.

Definition (Cryptosystem)

(Informally,) A **cryptosystem** consists of a key generation algorithm, encryption functions and decryption functions.



Encryption vs Encoding

Encoding is a conversion between different formats. Base64, ASCII,...

It is very easy to decode if the encoding scheme is known.

For Encryption, even if the encryption scheme is known, if the key is not known, it should be hard to decipher.



Symmetric Key Encryption

A horizontal bar with a teal segment on the left and an orange segment on the right.

An analogy of symmetric key encryption is when we lock things up in a box using a lock. We use the same key to **lock** and **unlock** the box. The catchphrase is that **the key used for encryption and decryption would be the same key**. In other words, $k = k'$.



Public Key Encryption

On the other hand, imagine you have a padlock and a key, and you lock a box by closing the hole on the padlock, and you can unlock it by inserting the (correct) key and turning it. However, you cannot use a padlock to unlock a padlock, nor use a key to lock a box without a padlock.

Back to cryptography terms, **the key for encryption and decryption are different** (though they can share some public parameters).

The idea of a public-key cryptography only arises in the 1970s! So the tools used for constructing these encryptions requires much more mathematical understanding (elementary number theory for RSA, group theory for El gamal and ECC, lattices for NTRU ...) so we will delay the discussion of such ciphers.





In modern cryptography, symmetric key cryptography and public key cryptography serve different purposes, and they are used in conjunction of each other to create complex protocols.

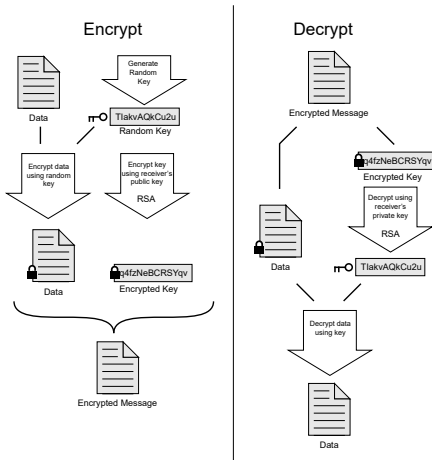


Figure: A simple illustration of how PGP, a cryptographic protocol often used to encrypt emails, works



Modern Symmetric Ciphers

Definition (Block Cipher)

A **Block Cipher** cuts a long message into several blocks of set length, and operate on each block with the same key.

An example of modern block cipher is [AES](#).

Definition (Stream Cipher)

A **Stream Cipher** uses the (short) key to generate a long key to encrypt long messages.

An example of modern stream cipher is [salsa20](#).



Classical Ciphers

Let's go back in time and study classical ciphers.



Classical Ciphers



Caesar Cipher

Caesar cipher is an ancient cipher developed 2000 years ago by Julius Caesar. The encryption just moves every character to the right by 3, wrapping back to A if necessary. So A becomes D, B becomes E, and so on, and Z becomes C.

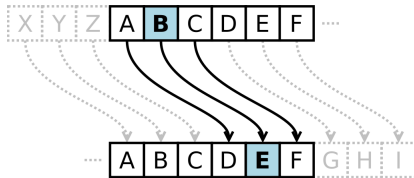


Figure: An illustration of the Caesar Cipher.

Of course it is possible to move character to the right by an offset other than 3. you can use 1, 2, ... or 25 as the **key** of the cipher ($k = 0$ is stupid).



Formally, if we represent (upper case) English characters as numbers, with 0 as A, 1 as B and et cetera, we can define the encryption and decryption function as

$$E_k(x) = x + k \pmod{26}$$

and

$$D_k(x) = x - k \pmod{26}$$

Exercise

Try decrypt the word "FDQBRXUHDGWKLV" with $k = 3$.

Example (ROT13)

ROT13 is an **encoding** with $k = 13$. If we apply ROT13 twice, what will happen?

[Demo](#)



Cryptanalysis (?)

How to break it?

Just try out all the possibilities! There are only 26 possible keys!

dcode

cryptii

Such a brute-force approach is a type of ciphertext only attack.



Definition (Brute-force attack)

A **Brute-force attack** (exhaustive search) is where an attacker tries many (perhaps all) possible keys (or passwords) in hopes of guessing correctly.

Definition (Ciphertext Only Attack)

A **Ciphertext Only Attack** (COA) is where we only have some number of ciphertext encrypted with the scheme, and no corresponding plaintext.

The catchphrase is: **if a cipher has small key space (small possibilities of keys), then it is susceptible to brute-force.**



Encrypting with Caesar Cipher

```
def caesar(m, k):  
    m_number = []  
    m = m.upper()  
    # Converting string to 0 - 25  
    m_number = [ord(char) - ord('A') for char in m]  
    rotated = [(i + k) % 26 for i in m_number]  
    # converting 0 - 25 back to string  
    c_number = [chr(i + ord('A')) for i in rotated]  
    result_string = "".join(c_number)  
    return result_string  
  
C = "CUHKOIL"  
print(caesar(C, 18))
```



Sample Brute Force Code for Caesar Cipher

```
def caesar_brute_force(c):  
    c_number = []  
    c = c.upper()  
    # Converting string to 0 - 25  
    c_number = [ord(char) - ord('A') for char in c]  
    for k in range(26):  
        rotated = [(i + k) % 26 for i in c_number]  
        # converting 0 - 25 back to string  
        m_number = [chr(i + ord('A')) for i in rotated]  
        result_string = "".join(m_number)  
        print("{}:\t{}".format(k, result_string))
```

```
C = "QEBNRFZHYOLTKCLUGRJMPLSBOQEBIXWVALD"  
caesar_brute_force(C)
```




Remark

Of course it is possible to use character sets other than the English alphabet. e.g. alphanumeric characters, ASCII, etc.

Basically you need to convert your characters to numbers first by **encoding**, which here just means assigning each character with an integer between 0 and the size of your alphabet -1 .

The encoding matters! Different encoding gives different encryption result!

Note that caesar cipher can be viewed as a block cipher with block size of 1. Same for the coming substitution ciphers as well.



Affine Cipher

Affine cipher is also an substitution cipher similar to caesar cipher, but the encryption and decryption is a bit more complicated. If we **encode** each character from the character set $\{0, 1, \dots, n-1\}$, and define the key $k = (a, b)$ where a and n do not share common factors other than 1, then

$$E_k(m) = ax + b \pmod{n}.$$

The decryption function is

$$D_k(c) = a^{-1}(c - b) \pmod{n}$$



Keep in mind that the -1 power does not mean $\frac{1}{a}$. Rather it means an integer r such that $a \times r$ dividing n has remainder 1.

Example

Let $n = 13$. Then to find the inverse of 3 modulo 13, we notice that $3 \times 9 = 27 = 13 \times 2 + 1 \equiv 1 \pmod{13}$, so $3^{-1} = 9$ in the case of modulo 13.

If you have installed gmpy2 already, then the **modular inverse** operation can be done using `gmpy2.invert(a,n)` to find the inverse of a modulo n .

You can find out why a needs to not share common factors (besides 1) with n if you read something about modular arithmetic (or read the appendix).



Weakness

If n is large, then the possibilities of a and b are huge! Then it may be infeasible to brute force. However, if n is small, e.g. 26, then there are only $12 \times 26 = 312$ possible keys. So it is still possible to brute force that.

A more deadly weakness of affine cipher is that it is vulnerable to a known-plaintext attack.

Definition (Known-plaintext Attack)

A **known-plaintext attack** (KPA) is an attack model where the attackers have knowledge of some ciphertexts and its decrypted plaintexts.



How to recover the key from plaintext and ciphertext?

Claim

Suppose we know the modulus n , and have the encrypted and plaintext version of **2** characters. Then we can recover the key (a, b) .



KPA for Affine Cipher

Proof

Let $(x_1, c_1), (x_2, c_2)$ be the two plaintext-ciphertext pairs we have. We get

$$c_1 \equiv ax_1 + b \pmod{n} \quad (1)$$

$$c_2 \equiv ax_2 + b \pmod{n} \quad (2)$$

(1) – (2) yields

$$c_1 - c_2 \equiv a(x_1 - x_2) \pmod{n}$$

Then we can solve for a . With a , we can solve b as well. □

Note that this only works if $x_1 - x_2$ do not share any common factors with n other than 1. This requirement should look trivial to those who study any elementary number theory.



Exercise

Suppose the n in affine cipher is unknown, and that $a + b < n$. Show that if you can encrypt 3 plaintexts and get their corresponding ciphertexts, (a, b, n) can be recovered.

Such a model of attacking is known as chosen plaintext attack (CPA).

Definition (Chosen Plaintext Attack)

A **chosen plaintext attack** is where we can choose some specially crafted plaintext, have it encrypted and obtain the ciphertext.



Extra Question

The below problem is too math for the training. But if you have learnt (or are learning) linear algebra, then attempting this would be a good exercise.

Exercise

Suppose the n in affine cipher is unknown.

Proof that if n is a prime, then given 3 plaintext-ciphertext pair (not chosen arbitrarily by us) such that the ciphertexts are coprime, n can still be recovered (disregarding the running time) (and thus cipher can be broken.)

Hint: Let p be a prime, \mathbf{A} be a $n \times n$ matrix with integer entries, and $\vec{x} \in \mathbb{Z}^n$. If $\mathbf{A}\vec{x} \equiv \vec{0} \pmod{p}$ has solutions other than $\vec{x} \equiv \vec{0} \pmod{p}$, then $\det \mathbf{A} \equiv 0 \pmod{p}$.



Substitution and Transposition Ciphers

Substitution and transposition cipher are old techniques used for encryption, and they were (quite easily) done by hand, as there were no computers 2000 years ago. That said, it is still being used today for modern ciphers, but much more difficult to do with pen and paper.

Definition (Permutation)

A **permutation** of n elements is a bijection from the set of n elements to itself.

Example (S_3 acted on $\{1, 2, 3\}$)

Consider the following permutation rule of 3 numbers $\{1, 2, 3\}$: 1 goes to 2, 2 goes to 1, and 3 is fixed in place.



(Monoalphabetic) Substitution Cipher

A Substitution cipher is where characters of the plaintext are substituted with other characters with some rules.

Example (Simple substitution cipher for S_3)

Consider the following permutation rule of 3 numbers 1,2,3 where we exchange 1 and 2. We can define an encryption scheme by applying the permutation to each character in the plaintext. For example, encrypting the word $m = 13231$ gives 23132.



To visualize these substitution, it is often helpful to draw a translation table (like the one for Caesar cipher).

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | B | C | D | E | F | G | H | I | J | K | L | M | N | O |
| F | I | R | E | B | D | K | A | Z | Q | H | J | X | M | S |

and so on. **Monoalphabetic** substitution ciphers make use of just one fixed substitution. **Polyalphabetic** substitution ciphers use different substitutions at different positions in the message.

Example

Caesar cipher and affine cipher are (monoalphabetic) substitution ciphers.



So far we saw substitution rules defined with a linear equation mod n (Caesar cipher and affine cipher). How about arbitrary permutations?

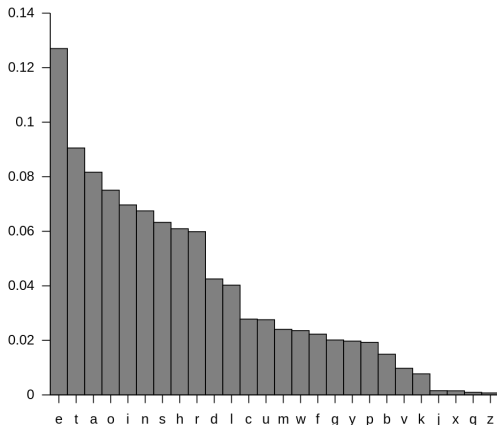
q11 from firebird.ozetta.net

There are $26! \approx 4 \times 10^{26}$ possible permutations of 26 characters!
We cannot just brute force this. What can we do?



Frequency Analysis

The distribution of English characters among common English texts are not uniform nor random.



| | | |
|---|-----|--------|
| R | 40× | 12.35% |
| O | 39× | 12.04% |
| I | 30× | 9.26% |
| E | 26× | 8.02% |
| K | 21× | 6.48% |
| P | 20× | 6.17% |
| D | 17× | 5.25% |
| G | 16× | 4.94% |
| F | 15× | 4.63% |
| W | 14× | 4.32% |
| X | 11× | 3.4% |
| N | 11× | 3.4% |
| V | 10× | 3.09% |
| Z | 9× | 2.78% |
| Q | 8× | 2.47% |
| Y | 7× | 2.16% |
| L | 7× | 2.16% |
| U | 7× | 2.16% |
| H | 6× | 1.85% |
| A | 5× | 1.54% |
| J | 3× | 0.93% |
| B | 2× | 0.62% |



The same goes to digraphs, which is a pair of characters (e.g. th), and trigraph (guess what that means :)). However, doing it by hand is painful. So there is a tool for it :) [quipquip](#)

Note that if the length of ciphertext is short, then this could not be done :(

This shows that the simple substitution cipher is not secure against a ciphertext-only attack.



Remark

Since the affine cipher is also just a special case of the simple substitution cipher, affine cipher is also vulnerable to frequency analysis!

Actually the same goes to Caesar cipher. But why though?



Vigenère¹ Cipher

So far we have seen substitution ciphers with block size 1. Now we will see a non-trivial block cipher, which is a **polyalphabetic** substitution cipher. Now the key becomes a string of characters, e.g. English again, say $k = k_1 \cdots k_6 = \text{"CRYPTO"}$. Here the key length is 6. By converting the characters into numbers, we can define the following encryption:

$$E_k(x_i) = x_i + \bar{k}_i \pmod{n}$$

However, our key length is only 6, and our message can get much longer than that! So if x_i is larger than 6, we need to repeat the key. Thus we have $\bar{k}_i = k_{(i \bmod m)}$, where m is the key length.

¹I won't attempt to pronounce this word.




Example

As an example, let's try to encrypt "CRYPTOISFUN" using the key "CRYPTO".

| | | | | | | | | | | | |
|-----------------------|---|----|----|----|----|----|----|----|----|----|----|
| Plaintext | C | R | Y | P | T | O | I | S | F | U | N |
| base26 | 2 | 17 | 24 | 15 | 19 | 14 | 8 | 18 | 5 | 20 | 13 |
| key | C | R | Y | P | T | O | C | R | Y | P | T |
| Key in base26 | 2 | 17 | 24 | 15 | 19 | 14 | 2 | 17 | 24 | 15 | 19 |
| $x_i + k_i \pmod{26}$ | 4 | 9 | 22 | 4 | 12 | 2 | 10 | 9 | 3 | 9 | 6 |
| Ciphertext | E | J | W | E | M | C | K | J | D | J | G |



How to break this?



Note that if we group every m -th characters together, each group are encrypted using the same character, meaning that the encryption reduces to a Caesar cipher! However, if the key length is not known, then this method is not very helpful. So one possible decryption method is as follows:

1. guess the key length
2. frequency analysis on respective key

Another possible approach is to guess the key. If the key is in English, then it is probably a valid English word! Then we can try to do a dictionary attack. As a corollary, never use an English word as the encryption key for this cipher.



Exercise

Try to implement the Vigenère cipher (both encryption and decryption) in python.

You may find the library function **itertools.cycle** and the built-in **zip** helpful.



Exercise



CryptoCTF 2019: Decode Me! (122 points²)

D: mb xwhvxw mlnX

4X6AhPLAR4eupSRJ6FLt8AgE6JsLdBRxq57L8IeMyBRHp6IGsmgFIB5E
:ztey xam lb lbaH

Hint: Not the whole ciphertext use the same key.

²Some CTFs assigns less points to the challenge the more teams are able to solve it. The score of a team is dynamic and can drop!



Exercise



- ▶ Very Fine
- ▶ Do you know what is very fine? It's af-fine!
- ▶ `nc chal.firebird.sh 35018`
- ▶ https://files.firebird.sh/chal-2021/04/very_fine.py



Transposition Cipher

For substitution cipher, we permute the character set. For **transposition cipher**, we permute the plaintext directly. A simple example would be to first fix the block size, say n , and pick a key k to be an arbitrary permutation of n characters. Then the encryption for each block is just permuting the characters in each block according to the rule.

Example (Transposition cipher for S_3)

Consider the following permutation rule of 3 numbers 1,2,3: 1 goes to 2, 2 goes to 1, and 3 is fixed in place. Using the example encryption scheme given above, encrypting the word $m = 321123$ gives 231213.



Question



Exercise

Suppose you are given a string of English characters as ciphertext. How to tell what type of cipher, substitution cipher or transposition cipher, the string is encrypted with?



Introduction to Modern Block Ciphers



Bitwise operators

Consider a boolean operator, e.g. the **and** operator

AND : $\{0, 1\} \times \{0, 1\} \rightarrow \{0, 1\}$. We can extend the definition of such an operator to binary strings by defining

$$a_1 a_2 \cdots a_n \text{ AND } b_1 b_2 \cdots b_n = (a_1 \text{ AND } b_1) || \cdots || (a_n \text{ AND } b_n)$$

Where a_i and b_i are bits, and $a || b$ and ab means concatenation of a and b .

Simply put, we operate on two binary strings bit by bit using the boolean operator. That give us a bitwise operator. Note that **the two operands need to have the same length**.

Example

1101 and 1011 = 1001.



XOR

To learn modern block ciphers, we need to learn the very important bitwise operation xor, which stands for **exclusive or**. This is often denoted as \oplus . The action of xor are described by the following truth table:

| \oplus | 0 | 1 |
|----------|---|---|
| 0 | 0 | 1 |
| 1 | 1 | 0 |

Alternatively, we can consider the xor operation acting on 2 binary string of equal size as an addition without carrying.

Why is this called the **exclusive** or? Compare this with the bitwise or operation, and note that 1 or 1 evaluates to 1, but $1 \oplus 1 = 0$.



Example

In python, we can do xor by the \wedge operator. For example, we can write $x \wedge y$ to do $x \oplus y$.

Remark

Not to confuse xor \oplus with the direct sum \oplus in abstract algebra, and not to confuse the xor in python \wedge with the power symbol we usually write as \wedge or the wedge product \wedge .

Example

$$100110 \oplus 001100 = 101010.$$

| | | | | | | |
|----------|---|---|---|---|---|---|
| | 1 | 0 | 0 | 1 | 1 | 0 |
| \oplus | 0 | 0 | 1 | 1 | 0 | 0 |
| <hr/> | | | | | | |
| $=$ | 1 | 0 | 1 | 0 | 1 | 0 |



Properties of xor

1. (Commutativity) $a \oplus b = b \oplus a$
2. (Zero) $a \oplus 0 = 0 \oplus a = a$
3. (Self-inverse) $a \oplus a = 0$
4. (Associativity) $a \oplus (b \oplus c) = (a \oplus b) \oplus c$

The third one is especially important for doing calculation on reversing xor operations.

Exercise

Proof all the properties above. (Hint: truth table)

Exercise

Show that $(a \oplus b) \oplus a = b$ for boolean variable a and b .



XOR encryption



Suppose we have a message encoded with binary (or hex). Then one way to encrypt a message is to apply xor with a key. Suppose we have a m bit long message x and a n bit number k as a key, and $m = n$, then we can define the following encryption:

$$E_k(x) = x \oplus k$$

Without the key, we cannot recover the message.



Exercise

Write down the decryption function. Check that if you encrypt then decrypt, you get back the message.

This shows that $E_k(x)$ is a bijection. (This means that E_k permutes all strings of length n .)



Cryptanalysis

Claim

The XOR encryption is vulnerable to a known-plaintext attack.

How?

Let x , k and c be the plaintext, the key and the ciphertext respectively. We have $c = x \oplus k$. Suppose we know x and c . Then

$$\begin{aligned}c \oplus x &= (x \oplus k) \oplus x \\ &= k\end{aligned}$$

As a corollary, we should not reuse keys!



One-time Pad

The idea that we must not reuse keys for the xor encryption give rise to the **one-time pad** (Remark: this is not just for xor encryption).

Each time we encrypt something, we can use a brand new key, where the keys are exchanged to the other person through some side channel (e.g. a code-book).

Suppose the code-book is not known to anyone else, the one-time pad achieves **perfect secrecy**!!! Which means that the ciphertext gives **no** information about the content of plaintext.



The XOR encryption is very simple to do in a computer (Probably all processors have the xor instruction built-in). And so, malwares may use xor to obfuscate their payload. Sometimes it will occur in ctf reverse challenges!



Exercise (Non-scored)




Internetworkache CTF 2016: SPIM (rev 50)

- ▶ My friend keeps telling me, that real hackers speak assembly fluently. Are you a real hacker?
- ▶ Decode this string: IVyN5U3X)ZUMYCs
- ▶ file



Substitution-Permutation Network

A horizontal bar with a teal segment on the left and an orange segment on the right.

It would seem that the substitution and permutation ciphers are simple, and it is. Some of them is easily broken with today's computation power. But what if we combine them together?



A Substitution-Permutation Network (SPN) make use of both substitution cipher (S-box) and permutation cipher (P-box). By applying several rounds of S-box and P-box and combining with some round key generated from a key each round, we get a hard-to-break cipher.

If one bit of the plaintext is changed, then it will change several bits of the output of the S and P-boxes, and so on. What we get in the end looks completely random. At the same time, if we change one bit of the key, then it will change the input of the S and P-boxes as well.

SPN is used in the AES encryption scheme, one of the most widely used modern symmetric key cipher currently.

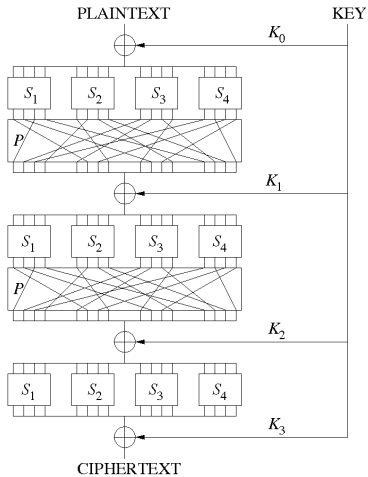


Figure: An example of a SPN



DES



Data Encryption Standard, or DES, is an old block cipher designed in the 1970s. As the key size of DES is 56 bits, it is regarded as insecure against brute force with large enough computer resources nowadays.



3-DES

As such, we may attempt to increase the security of DES by using more than 1 key. Let's use 3 keys k_1, k_2, k_3 , and define the encryption as follows:

$$E'_{(k_1, k_2, k_3)}(m) = E_{k_3}(D_{k_2}(E_{k_1}(m)))$$

Where E_k is the DES encryption function. So we have achieved 3×56 bits of security, right?



Meet-in-the-Middle Attack

The answer is no. Let's use an easier example than 3-DES: suppose the encryption is with 2 keys: $c = E'(m) = E_{k_1}(E_{k_2}(m))$. Note that $D_{k_1}(c) = E_{k_2}(m)$. If we know both the plaintext and the ciphertext, and we compute separately the values of $D_k(c)$ and $E_k(m)$ for all possible key k , then if any of $D_{k_a}(c)$ and $E_{k_b}(m)$ match, we can conclude that $k_1 = k_a$ and $k_2 = k_b$.

This attack is known as the **meet-in-the-middle** attack. (Not to be confused with **MAN**-in-the-middle attack, which is a different thing.)



Note that we only need to compute $2n$ values (where n is the number of possible keys) and find a match in two lists of size n , rather than computing n^2 values if we use a naive brute force. This is known as a meet-in-the-middle attack. In general meet-in-the-middle attacks reduce the search space to the square root of the original.

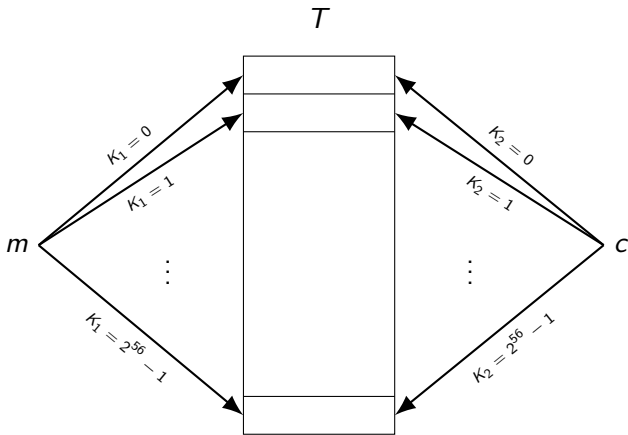



Figure: A pictorial demonstration of the meet-in-the-middle attack.



In the case of 3-DES, we can compute, on one hand, the values of $D_k(c)$, on the other hand, the values of $E_a(D_b(m))$. The former one has 2^{56} values, while the latter has $2^{56 \times 2} = 2^{112}$ values. The meet-in-the-middle attack would still work, and we have computed $2^{56} + 2^{112} \approx 2^{112}$ values. Thus the security of 3-DES would be 112 bits.



AES

A horizontal bar with a teal segment on the left and an orange segment on the right.

Advanced Encryption Standard (AES) is a modern block cipher with block size to be 128 bits (16 bytes) and key size to be either 128, 192 or 256 bits. The exact construction of AES is too complicated and thus is not a topic for today.



Mode of Operations



So far most of the block ciphers we saw (Caesar, Affine) has block size 1 byte, and for block ciphers with higher block size (Vigenère) we just repeat the same encryption if the plaintext size is higher than the key size. This is known as the **Electronic Code Book (ECB) Mode of Operation** of block ciphers.

Definition (Mode of Operation)

Mode of Operation of a cipher allows the use of one key to encrypt multiple blocks of data, at the same time achieving information security properties.

Definition (ECB)

The Electronic Code Book mode of operation is just applying the block cipher to each block using the same key.

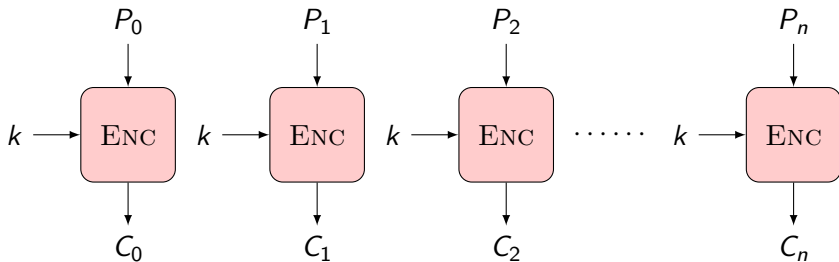


Figure: ECB working principle.



Pitfall of ECB

If the block is the same, then the ciphertext for that block is also the same.

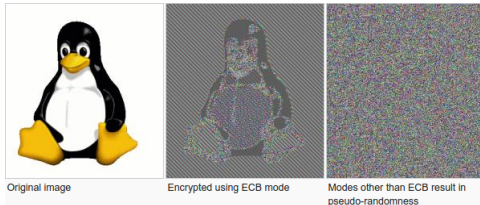


Figure: Using ECB to encrypt the picture of tux.

This is why we need other mode of operations to make these block ciphers more secure.



Cipher Block Chaining

Cipher Block Chaining (CBC) is not blockchain. It is a mode of operation that makes the subsequent ciphertext depend on the previous blocks. In this case, we need a random 16-byte string called the initialization vector (IV) for randomness, and the ciphertext will be different each time if the IVs are different. In mathematical terms,

Definition (CBC)

Let m_i denote the plaintext in blocks (16 bytes in size), and iv be a 16-byte random string. Then the corresponding ciphertext block c_i are defined as:

$$c_1 = E_k(m_1 \oplus iv)$$

$$c_i = E_k(m_i \oplus c_{i-1}) \text{ for } i > 1$$

Then the output is $c = iv || c_1 || c_2 || \cdots || c_n$.

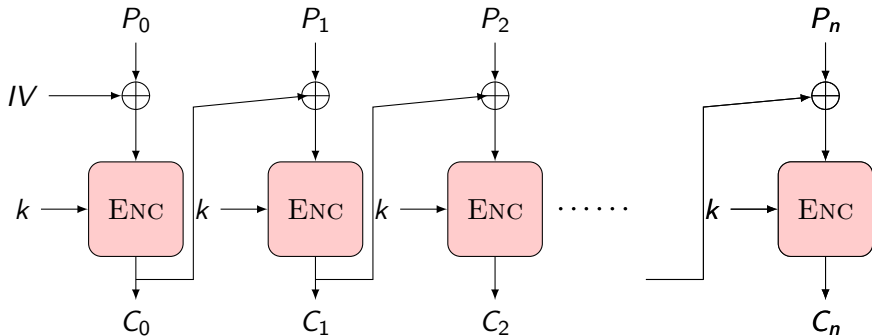
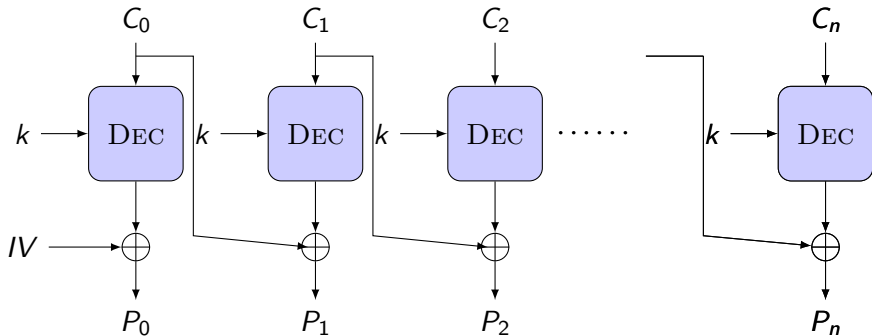


Figure: CBC encryption procedure.



The decryption is just the reverse.





Any Problem?



Bit Flipping

Note that the IV are included in the ciphertext, so we can change it. In fact, changing the IV allows us to control the **first block** of the decrypted text, although the later blocks would also be tempered.


Suppose the original (known) plaintext is m , and we want to change the plaintext to m' given the ciphertext c encrypted in CBC mode. So $c = iv || E_k(m \oplus iv)$. Then by setting $iv' = iv \oplus m \oplus m'$ (and let them decrypt $c' = iv' || E_k(m \oplus iv)$), the decrypted text would be

$$\begin{aligned}(m \oplus iv) \oplus iv' &= (m \oplus iv) \oplus iv \oplus m \oplus m' \\ &= m'\end{aligned}$$

So the plaintext now is m' .



Padding


A horizontal bar with a teal segment on the left and an orange segment on the right.

Remember that in a block cipher, the plaintext should have length exactly equal to the block size. What if it doesn't? Then we should add some padding to the message.

One idea may be to just padding null bytes, or pad random bytes to the message. However in this case we are not able to tell the boundary between the message and the padding.




PKCS# 7 Padding



One of the standards, PKCS# 7, provides a solution for padding. If we have to pad n bytes, we just pad the message with n 's. For example, if we need to pad 10 bytes, we pad the message with 10 $0 \times A$'s in the end, and if we don't need to pad, we would still pad a whole block with 0×10 (in the case of AES, which has a block size of 16 bytes).



Padding Oracle

A horizontal bar with a teal segment on the left and an orange segment on the right.

When we decrypt the message, we shall check if the padding is correct. If the padding is correct, we may proceed to process the message.

If we are able to know whether the padding of the decrypted text is correct (e.g. a webpage returning 502 if padding is correct but content is garbled, returning 500 when the padding is wrong), we are able to recover the plaintext, in the case of AES in CBC mode.

When we are able to use some mechanism get some feedback related to the decryption, we call the mechanism an **oracle**.



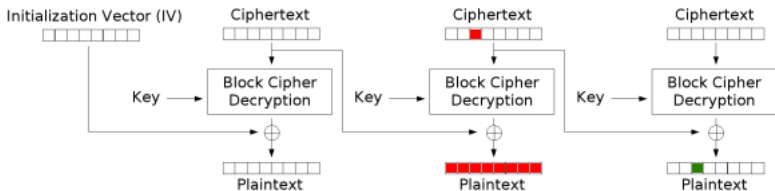
Padding Oracle in AES-CBC

Recall the decryption procedure for CBC mode is

$m_i = D_k(c_i) \oplus c_{i-1}$, thus changing one byte of c_{i-1} also change one byte of m_i (the decrypted text). Let's say we have two blocks, so the ciphertext is $c = iv || c_1 || c_2$. If we alter the last byte of c_1 to get c'_1 and send $iv || c'_1 || c_2$.

If the oracle returns correct padding, since we only changed one byte, the padding must be 0x01, so the last byte of $D_k(c_2) \oplus c'_1$ is 0x01. If the oracle returns false we just try another value. This way we can recover the last byte of c_2 .

Using the same strategy but for the other bytes, we can recover the whole block in $255 \times 16 = 4080$ attempts, rather than 2^{128} .



Cipher Block Chaining (CBC) mode decryption

Figure: Changing one byte of the ciphertext correspond to one byte change of the subsequent block. This is also true for IV. (Note that the blocks after, and that particular block will be garbled)




This padding oracle attack is a kind of chosen ciphertext attack.

Definition (Chosen Ciphertext Attack)

A **chosen ciphertext attack** is where we can choose some specially crafted ciphertext, have it decrypted and obtain the plaintext, or partial information of the plaintext.



Counter Mode



Another mode of operation is the Counter (CTR) mode. We still have a random 16-byte string, this time called a nonce, and this time we introduce a counter (just a function of the current block number i , but a regular counter works well). We encrypt $nonce \oplus ctr(i)$ (where ctr is the counter), and xor the result with the plaintext. This way we are turning block cipher into a **stream cipher**.

The advantage of CTR mode is the fact that all the blocks can be decrypted or encrypted in parallel. Recall for CBC mode, the i -th block encryption or decryption depend on the result of the previous block.

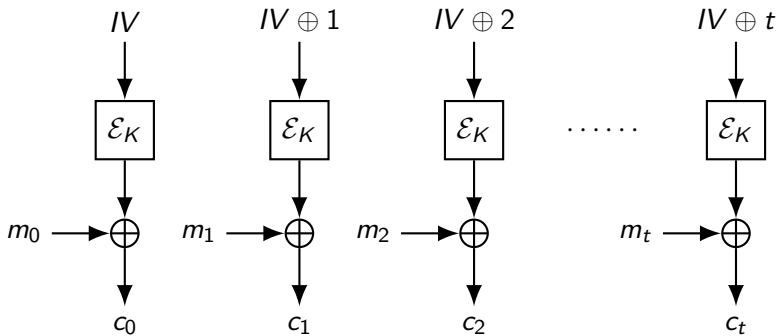


Figure: CTR mode illustration. In this case the counter function is just the regular counter.



Nonce Reuse

What if the nonce is reused?

Write $c_i = m_i \oplus E_k(\text{nonce} \oplus i)$ and $c'_i = m'_i \oplus E_k(\text{nonce} \oplus i)$, then

$$\begin{aligned} c_i \oplus c'_i &= (m_i \oplus E_k(\text{nonce} \oplus i)) \oplus (m'_i \oplus E_k(\text{nonce} \oplus i)) \\ &= m_i \oplus m'_i \end{aligned}$$

Just like xor encryption!



Bit Flipping (Again)

How about bit flipping?

Let's say we want the i -th plaintext m_i (known to us) to become m' . By setting the i -th block of ciphertext (originally c_i) to $c' = c_i \oplus (m_i \oplus m')$, we have

$$\begin{aligned}c' \oplus E_k(\text{nonce} \oplus i) &= (c_i \oplus (m_i \oplus m')) \oplus E_k(\text{nonce} \oplus i) \\&= (m_i \oplus m') \oplus (c_i \oplus E_k(\text{nonce} \oplus i)) \\&= (m_i \oplus m') \oplus m_i \\&= m'\end{aligned}$$

The other blocks are not even garbled!



Summary



1. What is cryptology
 - 1.1 What is encryption
 - 1.2 Basic terminologies
 - 1.3 Symmetric key cryptography vs public key cryptography
2. (Breaking) Classical Ciphers
 - 2.1 Monoalphabetic Substitution Ciphers
 - 2.1.1 Caesar Cipher
 - 2.1.2 Affine Cipher
 - 2.2 Frequency Analysis
 - 2.3 Polyalphabetic Substitution Ciphers
 - 2.3.1 Vigenère Cipher
 - 2.4 Transposition Ciphers
3. XOR Encryption
4. One-time pads
5. Attack Models
 - 5.1 Ciphertext-Only Attack
 - 5.2 Known-Plaintext Attack
 - 5.3 Chosen-Plaintext Attack
 - 5.4 Chosen-Ciphertext Attack



6. Modern Block Cipher

6.1 DES and 3-DES

6.2 Meet-in-the-middle Attacks

6.3 AES

7. Mode of Operations

7.1 Electronic Code Book (ECB)

7.2 Cipher Block Chaining (CBC)

7.2.1 Bit Flipping

7.2.2 Padding Oracle for PKCS# 7

7.3 Counter mode (CTR)



Reference



Interested readers can read the chapter 1 of the following two books if they wish to learn more about classical ciphers.



Appendix



Appendix


The appendix will introduce the concept modular arithmetic. If the reader have taken any course on elementary number theory, then the reader can skip that section.

The proof of theorems stated here will be omitted. Interested readers should look for suitable discrete mathematics, elementary number theory books or lecture notes to learn more. This is vital in actually **understanding** why things work, and not just memorising and working with definitions and theorems without knowing the inner workings.

Some suitable reference maybe chapter 4 of Rosen's *Discrete Mathematics and Its Applications* or Niven's *An Introduction to the Theory of Numbers*.



Modular Arithmetic



Modular arithmetic is an important tool for cryptography. Simply speaking, we want to ask for the remainder of some number dividing other elements. For example, the remainder of 14 dividing 5 is 4. In other words,

$$14 = 2 \times 5 + 4$$

We introduce a notation:

$$14 \bmod 5 = 4,$$

read as 14 mod 5 equals 4.



Alternatively, we have another notation:

Definition (modulo)

If the remainder of two numbers a, b dividing by another non-zero number n are equal, we write

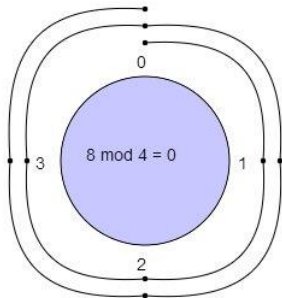
$$a \equiv b \pmod{n}$$

We read this as a is congruent to b mod (or modulo) n .



In python, we can do modular arithmetic by the percent operator %.
For example,

$$14 \% 5 == 4.$$



Example

$$14 \equiv 4 \pmod{5}.$$

$$4 \equiv -1 \pmod{5}.$$

$$k \equiv 1 \pmod{2} \text{ whenever } k \text{ is odd.}$$

15 : 00 and 3 : 00 looks the same on a clock. This is because

$$15 \equiv 3 \pmod{12}.$$



Properties

Let a, b, c be any integers. Then

- ▶ (Reflexivity) $a \equiv a \pmod{n}$
- ▶ (Symmetry) $a \equiv b \pmod{n}$ means $b \equiv a \pmod{n}$ as well.
- ▶ (Transitivity) $a \equiv b \pmod{n}$ and $b \equiv c \pmod{n}$ implies

$$a \equiv c \pmod{n}$$

If $a \equiv b \pmod{n}$, then

- ▶ $a + c \equiv b + c \pmod{n}$
- ▶ $a - c \equiv b - c \pmod{n}$
- ▶ $ac \equiv bc \pmod{n}$



Multiplicative inverse

Definition (GCD)

We say the **greatest common divisor** (gcd) of two numbers a and b to be c , denoted $\gcd(a, b) = c$, if

- ▶ c is a factor of both a and b
- ▶ c is the largest such factor.

Alternatively, readers may have learnt this concept as the **highest common factor** (HCF) of two numbers.



Theorem

If $ac \equiv bc \pmod{n}$ and $\gcd(c, n) = 1$, then $a \equiv b \pmod{n}$.

This theorem is important because it motivates us to define the inverse (kind of division) inside this modulo system (or residue system in number theory terms).



Definition (Multiplicative Inverse)

If $a \times b \equiv 1 \pmod{n}$ where a, b, n are integers, n positive, then we say that a has a multiplicative inverse $b \pmod{n}$, denoted $a^{-1} = b$.

Theorem

Multiplicative inverse of a exists mod n if and only if $\gcd(a, n) = 1$.

Given an equation $ax \equiv b \pmod{n}$, if the multiplicative inverse of a exists, then we can move a to the other side to obtain $x \equiv a^{-1}b \pmod{n}$. The requirement is that $\gcd(a, n) = 1$.

Actually if $\gcd(a, n) \neq 1$, solution may still exist subject to condition on b . This becomes trivial once the reader learn about Euclidean algorithm and Bezout's identity. However, these two results will be delayed until we discuss the RSA encryption scheme.



Example

Solve $2x \equiv 5 \pmod{7}$.

First note that $\gcd(2, 7) = 1$ so solution must exist (uniquely in mod 7). Then we can calculate to see that $2 \times 4 = 8 \equiv 1 \pmod{7}$ so the multiplicative inverse of 2 mod 7 is 4. Finally we get

$$4 \times 2x \equiv 4 \times 5 \pmod{7}$$

$$1 \times x \equiv 20 \pmod{7}$$

$$x \equiv 6 \pmod{7}$$