Key Concepts & Basic Mechanisms of Principal Protection Mechanisms

To manage security, we need to have a policy that specifies *who* (which subjects) can access *what* (which objects) and *how* (by which means).

Authentication

- If only Adam can access something, and someone else can impersonate Adam to do the same thing, then security fails
- Thus, we need ways to determine beyond a reasonable doubt that a subject's identity is accurate
- This property of accurate identification is called authentication

Access Control

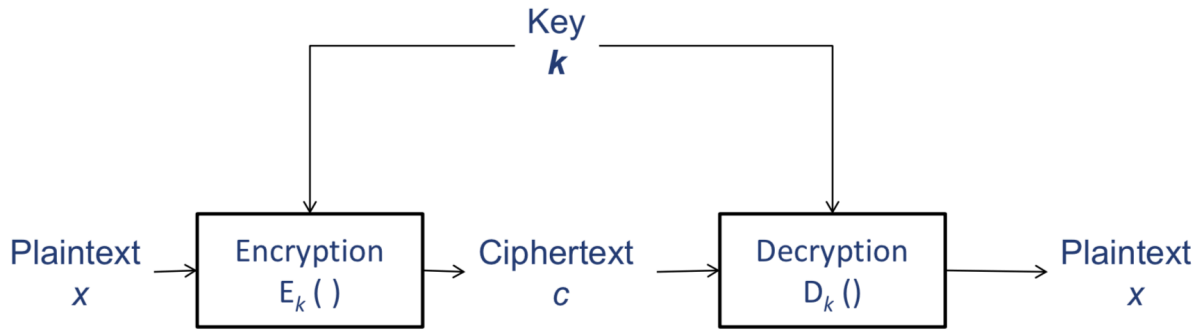
- The converse of the above must also be true: unless authorized, all other subjects must not be allowed to access the object
- After all, there's no meaning to saying that only Adam can access it if everyone else can anyways
- Thus, there needs to be a way to restrict access to only those subjects authorized
- This is thus called access control

Encryption

- Lastly, if the objects to be accessed can only be viewed and deduced by the subjects, then the chances of there being security issues are significantly lower
- The way to ensure non-intended receivers are unable to deduce the concealed objects is called encryption.

Cryptography & Encryption

An encryption scheme (also known as cipher) consists of two algorithms: encryption and decryption.



The ciphertext is transferred via insecure channel

There are two requirements:

- Correctness
  - For any plaintext x and key k, $D_k(E_k(x)) = x$
- Security
  - Given the ciphertext, it should be computationally difficult to derive useful information about the key k and plaintext x. The ciphertext should resemble a random sequence of bytes.

There is also a performance requirement: the encryption and decryption processes need to be able to be efficiently computed (Not really tested).

Example

I can encrypt a file using Winzip and a password, and send the encrypted file to someone via an insecure channel. I must, however, send the password via a secure channel. The file then can be decrypted using the password.

NOTE: Winzip is not an encryption scheme, but merely employs standard encryptions schemes such as AES.

Cryptography?

Cryptography is the study of techniques in securing communication in the presence of **adversaries** who have access to the communication.

Although cryptography is commonly associated with encryption, encryption is merely one of the primitives of cryptography.

Characters in Cryptography:

- Alice
- Bob
- Eve
- Mallory

Refer to the appendix for more information.

Classical Ciphers

Substitution Cipher

In substitution cipher, every symbol/character is substituted by another symbol/character. Let us define some terms:

- Plaintext and Ciphertext
  - A string containing elements of a set of symbols U
  - For example: U = {'a', 'b', 'c', …, 'z', '_'}
  - The plaintext may then be "hello_world"
- Key
  - A substitution table S that represents a 1-1 mapping function from U to U

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	_
g	v	w	b	n	e	f	h	d	a	t	l	u	c	q	m	z	i	r	s	j	x	o	y	k	_	p

$S(a) = g,\ S(b) = v,\ \ldots$

The inverse of S:
$S^{-1}(g)=a,\ S^{-1}(v) = b$

- Key Space
  - The set of all possible keys
  - Substitution Cipher: it is a set of all permutations of ABC/U
- Key Space Size
  - Total number of possible keys
  - Substitution Cipher: for U with size 27, it will be 27!
  - If size of U is 50, then the key space size will be 50!
- Key Size or Key Length
  - Number of bits to represent a particular key
  - Substitution Cipher: Key size = $\log_2(27!) = 94$ bits
  - Why? We can think of the key size as the number of possible "states" for the key, i.e. it's a specific combination of n bits of 1s and 0s to form this key out of all possible keys. As such, it's not 27, which is the number of elements, but is rather based on the key space. This will be more relevant when it comes to more modern ciphers that work more with bits.

Attacking Substitution Cipher

The goal for attackers is to figure out the key. If the key can be found, then the plaintext can be obtained. The converse is also true. If the attackers can get information on the plaintext or even the complete plaintext, they can also easily obtain the key.

There are thus two levels of access to information:

- **ciphertext only** – a large number of ciphertexts all encrypted using the same key
- **known plaintext** – pairs of ciphertext and the corresponding plaintext

Known Plaintext Attack: Substitution Cipher

This attack occurs when the attacker has access to pairs of ciphertexts and their corresponding plaintexts. The attacker can figure out the entries in the key that are used in the ciphertext and plaintext. With a sufficiently long ciphertext, the entire key can be determined.

If the adversary can derive the key, we call the scheme **"insecure under known plaintext attack"** or **"broken under known plaintext attack"**. Hence, substitution cipher is not secure under known plaintext attack.

First Few Bytes
It is not unreasonable for the attacker to obtain at least one pair of ciphertext and plaintext, as only a small number of bytes is required. In fact, only the first few bytes may be required of the plaintext. The rest can be deduced.

These first few bytes can sometime be guessed:

- Email data: Certain words in the headers are fixed, such as “From”, “Subject” etc.
- Many network protocols have fixed headers, or only a few choices in their first few bytes of data packets
- During WWII, cryptologists exploited commonly-used words like “weather” and “nothing to report” as the known plaintext to guess the secret keys

Ciphertext Only Attack: Substitution Cipher
Method #1: Exhaustive Search
One way to find the key is to use exhaustive search, or also known as the brute force search. The idea is to simply examine all possible keys one by one until a logical plaintext is obtained. For most schemes, brute force attacks are not feasible.

However, for some modern ciphers, it may actually be possible to use exhaustive search. More sophisticated attacks exploit characteristics of the encryption scheme to speed up the process.

Consider a table size of 27 for the key. In the worst case, the exhaustive search needs to carry out $27! \approx 2^{94}$ loops, and on average $\approx 2^{93}$ loops. This is infeasible using current computing power.

Method #2: Frequency Analysis
Suppose the plaintexts are English sentences. The **letter frequency distribution** in English text is not uniform, for example, “e” is more commonly used than “z”.

If the adversary is aware that the plaintexts are English sentences, given a sufficiently long ciphertext (say around 50 characters), then an adversary may be able to guess the plaintext by:

- Mapping the frequently-occurring letters in the ciphertext to the frequently-occurring letters of English.
- Frequency analysis can be successfully carried out!

Hence, substitution cipher is **not secure under ciphertext-only attack** either, when the plaintexts are English sentences. In fact, the attack is effective on any language.

Heuristics for Frequency Analysis
The most frequently occurring character in English is either e or a space. There are also single letter words such as I and a, digraphs such as to, it, is, do, on, in, at etc., and trigraphs such as can, and, get, not, for, the, has, one, man, men etc.

Caesar Cipher (Type of Substitution Cipher)
It's a type of substitution cipher where every letter in the table is "shifted" down the alphabet/symbols by a certain number of spaces. For example, with a shift of 1, A will be represented by B in the ciphertext, B will be replaced by C, so on until Z is replaced by _.

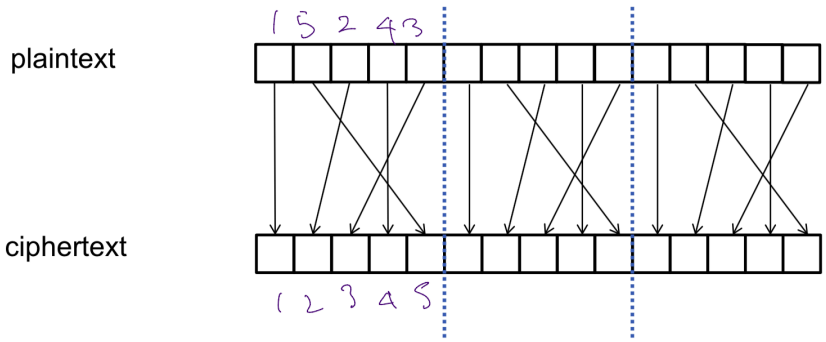
For Caesar Cipher:

- Key Space
  - The set of all possible keys
  - Caesar Cipher: Set of all possible shifts until the original table is obtained.
- Key Space Size
  - Total number of possible keys
  - Caesar Cipher: 26 or 27, depending on whether there's a delimiter character
- Key Size or Key Length
  - Number of bits to represent a particular key
  - Caesar Cipher: Key size = $\log_2(27) = 5$ bits

Permutation Cipher/Transposition Cipher
This cipher first groups the plaintext into blocks of *t* characters, then shuffles the character within each individual block to obtain a fixed and secret permutation. It is a one-to-one function that basically maps characters within a block to another position in the resultant block.

We can express the permutation p as a sequence: $p = (p_1, p_2, p_3, ..., p_t)$. The block size *t* can also be kept secret and be part of the key, as this reduces the information attackers have access to.

Given the plaintext and the key ***t***=5, ***p***=(1,5,2,4,3):



Known Plaintext Attack: Permutation Cipher
Permutation cipher fails miserably under known-plaintext attack, as it is very easy to determine the key given a plaintext and ciphertext.

Ciphertext Only Attack: Permutation Cipher
Permutation cipher can also be easily broken under ciphertext only attack if the plaintext is an English text.

One-Time-Pad

Given a n-bit plaintext ($x_1x_2\ldots x_n$) and a n-bit key ($k_1k_2\ldots k_n$), we can output the ciphertext, C:

$$C = (x_1 \oplus k_1)(x_2 \oplus k_2)(x_3 \oplus k_3)\ldots(x_n \oplus k_n)$$

The condition is that the key and the plaintext must be of the same length.

We can decrypt the ciphertext by XORing the ciphertext with the key once more to get the plaintext, X:

$$X = (c_1 \oplus k_1)(c_2 \oplus k_2)(c_3 \oplus k_3)\ldots(c_n \oplus k_n)$$

For this to work, we need to be able to transfer the key securely. However, if we can securely transfer the key, we might as well securely transfer the plaintext!

What is $\oplus$ or XOR?

The XOR or $\oplus$ operation between A and B is basically: (A + B) mod 2. Basically, if both A and B are the same bit i.e. both 1s or both 0s, then the result from the XOR operation will be 0. If they are different, the result will be 1.

Some interesting properties of XOR:

- Commutative: $A \oplus B = B \oplus A$
- Associative: $A \oplus (B \oplus C) = (A \oplus B) \oplus C$
- Identity element: $A \oplus 0 = A$, where 0 is the identity element
- Self-inverse: $A \oplus A = 0$

For One-Time-Pad:

- Key Space
  - One-Time-Pad: Set of all possible keys of the same length as the plaintext
- Key Space Size
  - Total number of possible keys
  - One-Time-Pad: 2^n
- Key Size or Key Length
  - Number of bits to represent a particular key
  - Caesar Cipher: Key size = $\log_2(2^n)$ = n bits

Security of One-Time-Pad

It is easy to derive the key from knowing the ciphertext and plaintext, but this key will be useless, as the key will not be used again. It is also very difficult to perform exhaustive search on it.

Furthermore, even if you know part of the key, you cannot figure out the rest i.e. it **leaks no information** of the plaintext, even if the attacker has an arbitrary running time. Hence, the one-time-pad is also called “unbreakable”, provided that a good “random” key is used.

Modern Ciphers

Designs of modern ciphers take into consideration of known-plaintext attack, frequency analysis and other known attacks. Some examples of modern ciphers include:

- DES (Data Encryption Standard, 1977)
  - Key length is too short
- RC4 (Rivest's Cipher 4, 1987)
  - Broken in some adoptions
- A5/1 (used in GSM, 1987)
  - Vulnerable
- A5/3 or KASUMI (3G) or SNOW 3E (LTE)
- AES (Advanced Encryption Standard, 2001)
  - Believed to be secure, classified as Type 1

Exhaustive Search, Key Length and Work Factor

If the key length is 32 bits, there are 2^{32} possible keys. As such, the exhaustive search will take:

- 2^{32} searches in the worst case
- 2^{31} searches on average (i.e. half of the possible cases)

We can thus quantify the security of an encryption scheme by the length of the key. For example, comparing scheme A with 64-bit keys and scheme B with 54-bit keys, scheme A is more secure with respect to exhaustive search.

Of course, there are attacks that are more efficient than exhaustive search. In those cases, we still quantify the security by the **equivalent exhaustive search**.

- For example, the best known attack on a 2048-bit RSA requires roughly 2^{112} searches
- Hence, we treat its security as equivalent to 112 bits, and we say that it has a key strength of 112 bits.

Work Factor is the amount of effort needed to break an encryption or mount a successful attack. For example, a 25 character message with 26 possible symbols has 26^{25} possible decipherments. If your computer could perform on the order of 10^{10} operations per second, finding this decipherment would require 10^{25} seconds, or roughly 10^{17} years.

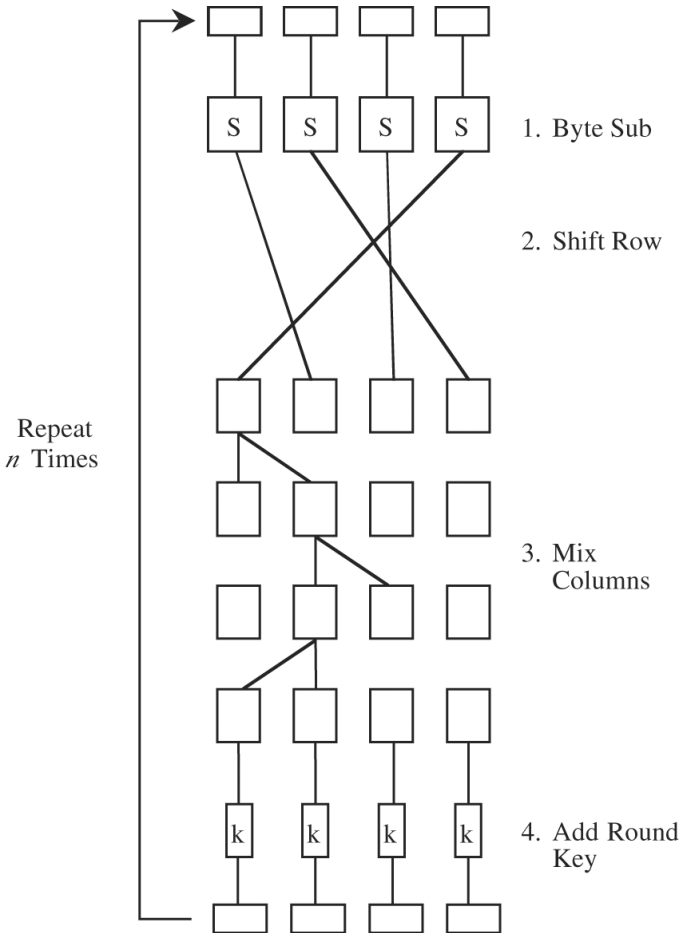
Data Encryption Standard

The key length of DES is 56 bits. While using exhaustive search to crack it did not seem feasible in the 70s, it soon became possible using distributed computing or a specialised chip.

Advanced Encryption Standard

AES is symmetric block cipher developed in 1999 by independent Dutch cryptographers. It involves:

- Byte Sub
- Shift Row
- Mix Columns
- Add Round Key
- The above repeated n times



Comparison between AES and DES

	DES	AES
Date designed	1976	1999
Block size	64 bits	128 bits
Key length	56 bits (effective length); up to 112 bits with multiple keys	128, 192, 256 (and possibly more) bits
Operations	16 rounds	10, 12, 14 (depending on key length); can be increased
Encryption primitives	Substitution, permutation	Substitution, shift, bit mixing
Cryptographic primitives	Confusion, diffusion	Confusion, diffusion
Design	Open	Open
Design rationale	Closed	Open
Selection process	Secret	Secret, but open public comments and criticisms invited
Source	IBM, enhanced by NSA	Independent Dutch Cryptographers

Properties of Ciphers

Confusion

An attacker should not be able to predict what will happen to the ciphertext when one character in the plaintext changes. In other words, the input (plaintext and key pair) undergoes complex transformations during encryption.

A cipher with good confusion has a complex functional relationship between the plaintext/key pair and the ciphertext.

Diffusion

A change in the plaintext will affect many parts of the ciphertext. In other words, information from the plaintext is spread over the entire ciphertext. The transformations depend equally on all bits of the input.

A cipher with good diffusion requires an attack to access much of the ciphertext in order to infer the encryption algorithm.

Block Ciphers vs Stream Ciphers

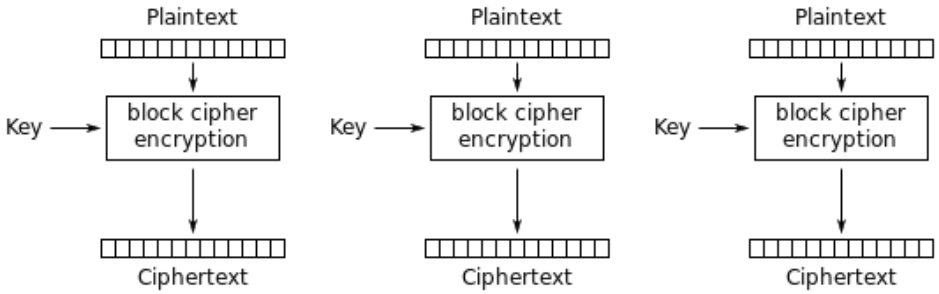
Block Cipher

The concept behind block cipher is to basically break up the plaintext into multiple blocks of some fixed-length, then encrypt each block separately. This encryption process between blocks may be independent or related, depending on the mode used.

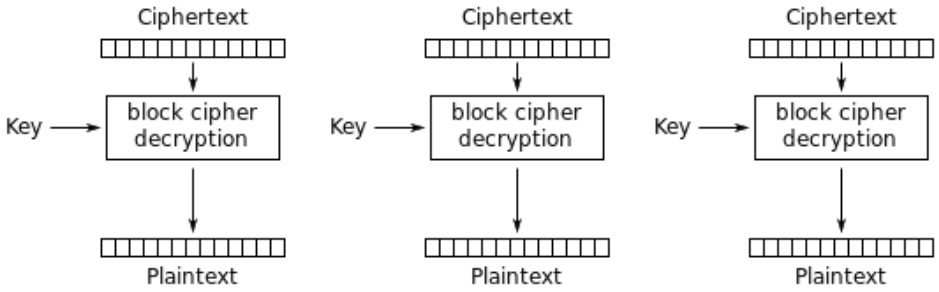
Most modes require a separate initialisation vector, which is a block of bits to randomise the encryption and produce distinct ciphertexts for the same plaintext under the same key.

Electronic Codebook (ECB)

The simplest of the encryption modes is the Electronic Codebook (ECB) mode. The message is divided into blocks, and each block is encrypted separately.



Electronic Codebook (ECB) mode encryption



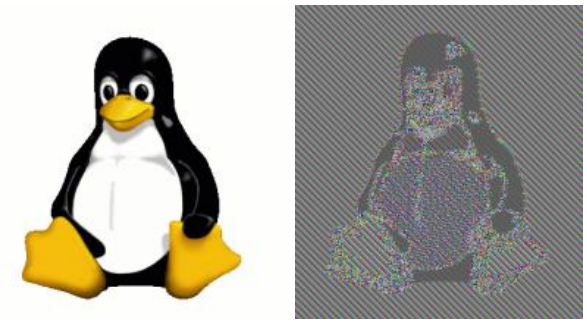
Electronic Codebook (ECB) mode decryption

The disadvantage of this method is the lack of diffusion. Because ECB encrypts identical plaintext blocks into identical ciphertext blocks, it does not hide data patterns well. In some senses, it doesn't provide serious message confidentiality, and it is not recommended for use in cryptographic protocols at all.

A striking example of the degree to which ECB can leave plaintext data patterns in the ciphertext can be seen when ECB mode is used to encrypt a bitmap image which uses large areas of uniform colour. While the colour of each individual pixel is encrypted, the overall image may still be discerned, as the pattern of identically coloured pixels in the original remains in the encrypted version.

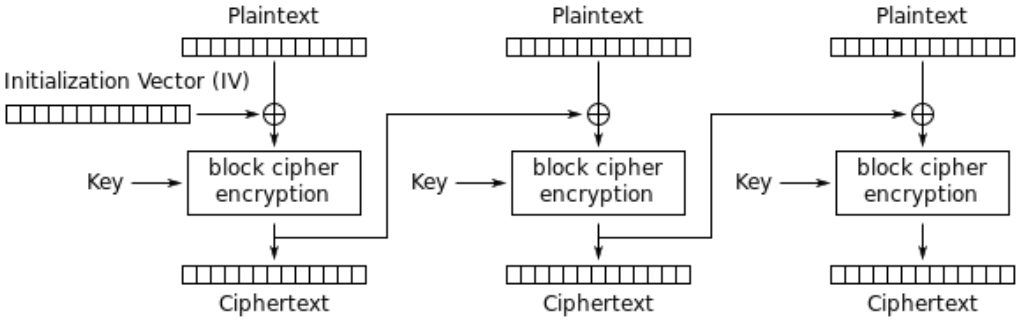
An encryption scheme is “deterministic” in that it always produces the same output when given the same input. In contrast, a “probabilistic” encryption scheme produces different ciphertexts even with the same input. AES is deterministic, hence we cannot employ AES with randomly-chosen IVs for each block.

Thus, we have a mode-of-operation that describes how the blocks are to be “linked” such that different blocks at different locations would give different ciphertexts, even if they have the same content. We unfortunately cannot just choose a random IV for each block and store it somewhere, as it will significantly increase the size of the final ciphertext.

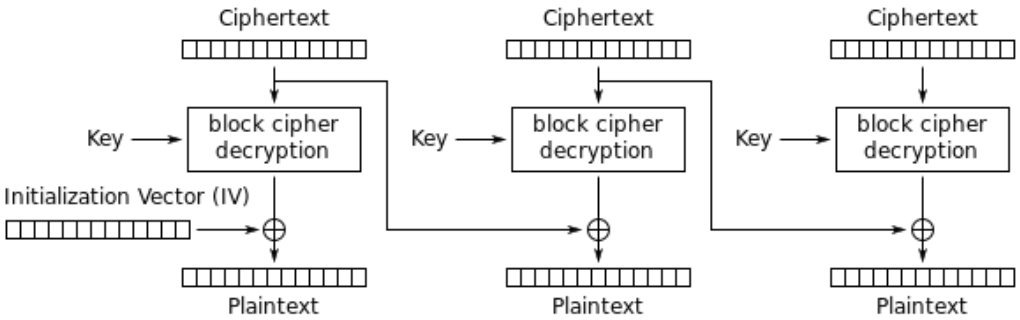


ECB mode can also make protocols without integrity protection even more susceptible to replay attacks, since each block gets decrypted in exactly the same way.

Cipher Block Chaining (CBC)
Ehrsam, Meyer, Smith and Tuchman invented the Cipher Block Chaining (CBC) mode of operation in 1976. In CBC mode, each block of plaintext is XORed with the previous ciphertext block before being encrypted. This way, each ciphertext block depends on all plaintext blocks processed up to that point. To make each message unique, an initialization vector must be used in the first block.



Cipher Block Chaining (CBC) mode encryption



Cipher Block Chaining (CBC) mode decryption

If the first block has index 0, the mathematical formula for CBC encryption is

$$C_0 = E_K(P_0 \oplus IV)$$
$$C_i = E_K(P_i \oplus C_{i-1})$$

And the mathematical formula for CBC decryption is

$$P_0 = D_K(C_0) \oplus IV$$
$$P_i = D_K(C_i) \oplus C_{i-1}$$

Block Cipher thus can ensure both confusion and diffusion.

Why must we use IV for CBC (specifically)?
If no IV is used in AES-CBC, while two identical blocks within one single plaintext will be encrypted into two different ciphertext blocks, two separate but identical plaintexts will always been encrypted into the same ciphertext. This may not be desirable.

Furthermore, this makes the encryption susceptible to chosen-plaintext attacks.

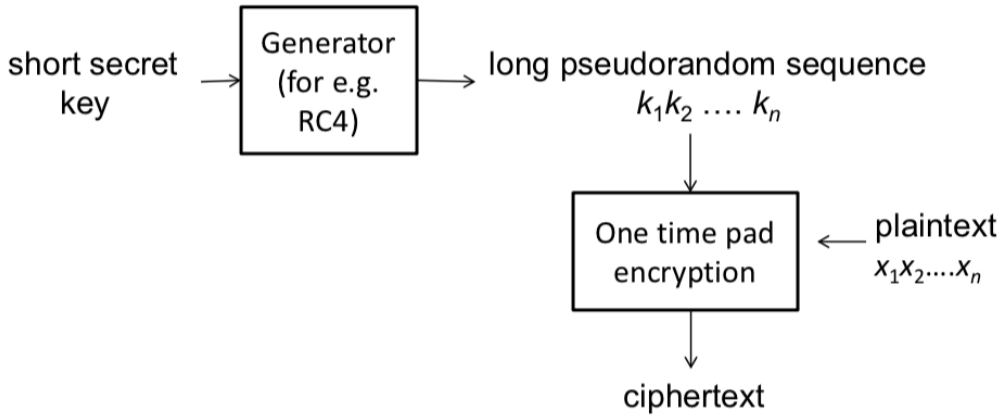
Localised Reordering Attacks
Integrity can be compromised with CBC without the receiver noticing. This can be done via reordering or modifying ciphertext blocks. Notice that in CBC decryption, an altered or corrupted ciphertext block affects the corresponding plaintext block and the following one, but the rest of the blocks remain intact. As such, the effect of a block-ordering attack will be localised.

If the plaintext has some embedded extra information about the block order, or if the out-of-order blocks can be observed "semantically" by the receiver, then this block-reordering attack can be easily detected. Yet, the reordering attack can go undetected if the plaintext represents an image or video, where a small localized modification may not be sufficiently noticeable by the receiver.

A MAC may be used on the whole file to ensure its authenticity and integrity. However, if MAC is applied on a block by block basis, then it is also possible to reorder the MAC along with the blocks to remove any initial suspicion of compromise of authenticity and integrity. Basically, don't MAC block by block.

<u>Stream Cipher</u>
Stream cipher is inspired by the one-time-pad. Suppose the plaintext is 2^{20} bits long, but the secret key is only 256 bits. Stream cipher will generate a 2^{20}-bit sequence from the key, and takes the generated sequence as the "secret key" in one-time-pad.

This generator will need to be carefully designed so as to give a cryptographically-secure pseudorandom sequence.

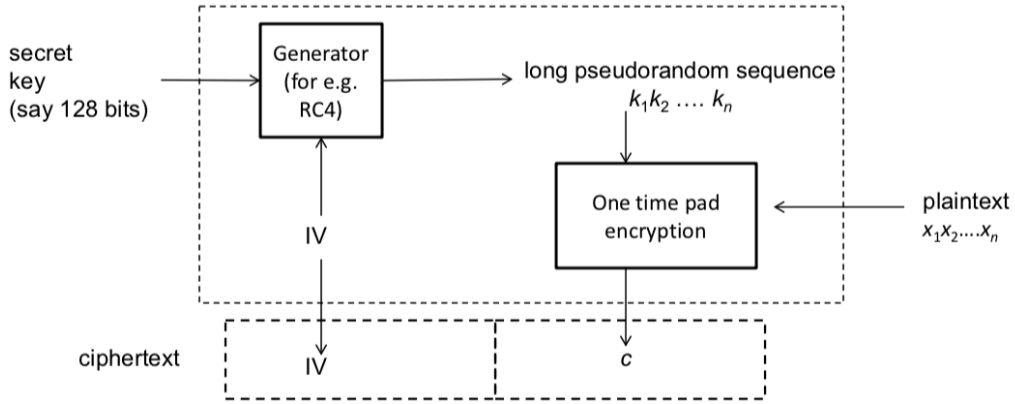


The stream cipher is thus a symmetric cipher, where the same key (or rather, same secret key) is used for both encryption and decryption.

Initial Value / Initialisation Vector (IV)
Stream cipher also has an initial value or initialisation vector, much like block cipher. The IV can be randomly chosen or obtained from a counter.

The IV is then used with the secret key (usually via some process of XOR) to generate a long pseudorandom sequence that is ultimately used to encrypt the plaintext. The IV is then appended to the front of the ciphertext in clear (i.e. non-encrypted).

For decryption, the IV is then extracted from the ciphertext and used with the key to obtain the pseudorandom sequence to decrypt the ciphertext.



Stream Cipher ensures only confusion and not diffusion.

Why must we use IV for Stream Cipher?
Without the IV, if an attacker can obtain two ciphertexts, they can simply XOR the two ciphertexts together.

Let's say we have ciphertext U and V, which are obtained from plaintexts X and Y respectively without using IV.

$$U \oplus V = (X \oplus K) \oplus (Y \oplus K) = (X \oplus Y) \oplus (K \oplus K) = X \oplus Y$$

The above is obtained after doing some manipulation as a result of the associative and commutative properties of XOR. We can get $X \oplus Y$.

So what if we have $X \oplus Y$?
It doesn't sound like it's anything big but let us take X and Y as black and white images, where every pixel corresponds to a bit.

Image X

Image Y

Image X ⊕ Image Y (without IV)

Image X ⊕ Image Y ⊕ K_1 ⊕ K_2

Why is it so different? This is because as K_1 and K_2 are completely different due to being produced with completely different IVs, they cannot cancel the effect of the pseudorandom sequences out. Any attempts to XOR with obtained ciphertexts would not return anything useful.

Attacks on Cryptosystem Implementations

Reusing IV

Some applications overlook the importance of the IV generation, and thus can end up reusing the same IV for different plaintexts. For example, one might derive the IV from a filename to encrypt a file F. However, it is not uncommon for different files to have the same filename.

The below is the RC4 flaw from Microsoft:
In this report, we point out a serious security flaw in Microsoft Word and Excel. The stream cipher RC4 with key length up to 128 bits is used in Microsoft Word and Excel to protect the documents. But when an encrypted document gets modified and saved, the initialization vector remains the same and thus the same keystream generated from RC4 is applied to encrypt the different versions of that document. The consequence is disastrous since a lot of information of the document could be recovered easily.

For the Cipher Block Chaining (CBC) mode of AES, the IV needs to be unpredictable to prevent a certain type of attack. The Browser Exploit Against SSL/TLS Attack (BEAST) exploits this, as when encrypting multiple packets, the IV of a packet would be the last ciphertext block of the previous packet, which is visible to anyone.

Reusing One-Time-Pad Key

Similar to reusing IV, if the one-time-pad key is reused and the attacker notices it, it is possible to easily decrypt entire messages. For example, the Venona project saw the decryption of encrypted messages from the Soviet Union by the US.

Predictable Secret Key Generation

When coding, we may use packages such as java.util.Random despite the fact that it is not actually secure. java.security.SecureRandom is preferred. This is because:

1. **Size:** A Random class has only 48 bits whereas SecureRandom can have up to 128 bits. So the chances of repeating in SecureRandom are smaller.

2. **Seed Generation:** Random uses the system clock as the seed/or to generate the seed. So they can be reproduced easily if the attacker knows the time at which the seed was generated. But SecureRandom takes Random Data from your OS (they can be interval between keystrokes etc – most OS collect these data and store them in files – /dev/random and /dev/urandom in case of linux/solaris) and use that as the seed.

3. **Breaking the code:** In case of random, just 2^48 attempts are required, with today's advanced cpu's it is possible to break it in practical time. But for SecureRandom 2^128 attempts will be required, which will take years and years to break even with today's advanced machines.

4. **Generating Function:** The standard Oracle JDK 7 implementation uses what's called a Linear Congruential Generator to produce random values in java.util.Random. Whereas Secure Random implements SHA1PRNG algorithm, which uses SHA1 to generate pseudo-random numbers. The algorithm computes the SHA-1 hash over a true random number(uses an entropy source) and then concatenates it with a 64-bit counter which increments by 1 on each operation.

5. **Security:** Consequently, the java.util.random class must not be used either for security-critical applications or for protecting sensitive data.

Designing Your Own Cipher

You're asking for trouble unless you're really an expert in this field.

Kerckhoff's Principle
"A system should be secure even if everything about the system, *except the secret key*, is public knowledge"

vs

Security through Obscurity
To hide the design of the system in order to achieve security.

Which one is better?
Against Obscurity
RC4

- Was introduced in 1987 and its algorithm was a trade secret
- In 1994, a description of its algorithm was anonymously posted in a mailing group.

MIFARE Classic

- A contactless smartcard widely used in Europe employed a set of proprietary protocols/algorithms
- However, they were **reverse-engineered** in 2007
- It turned out that the encryption algorithms were already known to be weak (using only 48bits) and breakable

For Obscurity
Usernames

- They are not secrets
- However, it is not advisable to publish all the usernames

Computer Network Structure & Settings

- E.g. location of firewall and the firewall rules
- These are not secrets, and many users within the organization may already know the settings
- Still, it is not advisable to reveal them

The actual program used in a smart-card

- It is not advisable to publish it
- If the program is published, an adversary may be able to identity vulnerabilities that were previously unknown, or carry out side-channel attacks
- A sophisticated advisory may be able to reverse-engineer the code nevertheless

So should we use obscurity?
In general, obscurity can be used as one layer in a "defence in depth" strategy. It could deter or discourage novice attackers, but is ineffective against attackers with high skill and motivation. The system must remain secure even if everything about it, except its secret key, becomes known.

Historical Facts

Cryptography

- Cryptography is closely related to warfare and can be traced back to ancient Greece
- Its role became significant when information is sent over the air
- WWII: Famous encryption machines include the Enigma, and the Bombe (that helped to break Enigma)

Modern Ciphers

DES (Data Encryption Standard):

- 1977: DES, 56 bits
- During cold war, cryptography, in particular DES was considered as “munition”, and subjected to export control. Currently, export of certain cryptography products is still controlled by US.
- 1998: A DES key broken in 56 hours
- Triple DES (112 bits) is still in used

AES (Advanced Encryption Standard):

- 2001: NIST. 128, 192, 256 bits

RC4

- 1987: Designed by Ron Rivest (RSA Security), initially a trade secret
- 1994: Algorithm leaked
- 1999: Used in widely popular WEP (for WiFi); WEP implementation has 40 or 104-bit key
- 2001: A weakness in how WEP adopts RC4 is published by Fluhrer, Mantin, Shamir
- 2005: A group from FBI demonstrated the attack
- Afterward: Industry switched to WPA2 (with WPA as an intermediate solution)

Tutorial Learnings

Clock Cycles

If 512 clock cycles are needed to test whether a 64-bit cryptographic key is correct, when given a 64-bit plaintext and its corresponding ciphertext, and a 4GHz single-core processor has 2^{32} cycles per second, then it takes 2^{41} seconds to check all 2^{64} keys, which is approximately 2^{16} years, which is approximately 64K years, where $1K = 2^{10}$.

If we have 1024 servers, each with a quad-core processor, we have 2^{12} processors, and the time needed is now 16 years.

Time-Space Trade-off

If the question asks about a certain ciphertext that is transmitted with a lot of 0s at the front and 1s at the back, and the key used is short, we can actually construct a table to allow for quick lookup, in exchange for large amount of storage space used.

In the tutorial question, we construct a table of $Enc_k(000…000)$ (64 bits of 0s) for all possible 2^{32} values of (k XOR IV), which is 32-bit long. Thus the derived table will take 2^{32} x 64 bits = 32GB. If (k XOR IV) is also stored, then another 16GB is needed. Upon receiving the first 64 bits, we look it up and determine the employed (k XOR IV).

We don't even need to know the original k, as we just need the (k XOR IV) to decipher the ciphertext.

Compression

Compressing an encrypted file will yield very little or no compression gain. This is since the encrypted file will resemble a random sequence, which is a requirement of a good encryption scheme. A compression algorithm, which takes advantages of repeating patterns, therefore will not work well on an encrypted file.

Password Length

Even if we use a 256-bit AES key, if the key is generated from hashing a password of known length, known keyspace and with a known hash function, then an attacker can easily reverse engineer the password to decrypt the file, without needing to try all 2^{256} possible keys. For example, if the password is made of all digits and is 6 digit long, then there are only 10^6 possible passwords.

Ciphertext XOR-ing Attack

If two plaintexts were encrypted using stream cipher with the same secret key and IV, by XOR-ing the two ciphertexts, we actually get the same result as when we XOR the two plaintexts, as the secret key and IV XORs itself and cancels out.