ECS655U: Security Engineering

Week 2: Perfect Secrecy, Practical Security, Symmetric Key Encryption

Dr Arman Khouzani

January 18, 2018

EECS, QMUL

Table of contents

Learning Outcomes

- Define the notion of perfect secrecy.
- Understand how it can be achieved in theory, simply!
- Appreciate why it of limited practical use.
- Appreciate (and get comfortable with the fact) that all practical cryptosystems are (theoretically) insecure.
- Recognise the concept of “practical” security.
- Identify the basic design features of AES, as the current standard for symmetric key encryption;
- Being able to compare several different block cipher “modes of operation” and their properties.

Perfect Secrecy

Perfect Secrecy: definition

Definition

A cryptosystem has ***perfect secrecy*** if, after seeing the ciphertext, an interceptor gets no *extra* information about the plaintext other than what was known before the ciphertext was observed.

Perfect Secrecy: Implication of the definition

- ▷ The definition of *perfect secrecy* does not refer to any computational restriction. So if the adversaries have all the computational power and all the time in the universe at their disposal, they still cannot learn anything about the plaintext beyond what they already knew before observing the ciphertext.
- ▷ So this means, even exhaustive key search (brute-force) cannot break “Perfect Secrecy”!
- ▶ but is this ever achievable?

Achieving Perfect Secrecy!

Surprisingly, perfect secrecy is achievable simply!

- ▷ To see the idea, consider a very small setting, where there are only two possible plaintext messages: `ATTACK`, `RETREAT`.
- ▷ Now, consider the following encryption/decryption table:

	`ATTACK`	`RETREAT`
Key K_1	$E_{K_1}(\texttt{ATTACK}) = 0$	$E_{K_2}(\texttt{RETREAT}) = 1$
Key K_2	$E_{K_2}(\texttt{ATTACK}) = 1$	$E_{K_2}(\texttt{RETREAT}) = 0$

- ▷ The (symmetric) "key" is "which row" the sender uses to encode its message. The sender chooses one of the rows randomly with probability 0.5.

Achieving Perfect Secrecy!

	ATTACK	RETREAT
Key K_1	$E_{K_1}(\texttt{ATTACK}) = 0$	$E_{K_2}(\texttt{RETREAT}) = 1$
Key K_2	$E_{K_2}(\texttt{ATTACK}) = 1$	$E_{K_2}(\texttt{RETREAT}) = 0$

- Suppose the adversary sees the ciphertext is 0. What can they learn about the plaintext?
- Suppose the adversary knows “in advance” (beforehand) that with probability of 0.9, the sender is actually going to send `ATTACK`. What would be the answer the previous question?

Achieving Perfect Secrecy

Let us represent our two plaintexts in binary numbers:

`ATTACK`:0　　　`RETREAT`:1

Similarly, let us represent the key (which row) in binary:

K_1(first row):0　　　K_2(second row):1

Then the table will become:

	0	1
$(k_1 = 0)$	$E_0(0) = 0$	$E_0(1) = 1$
$(k_2 = 1)$	$E_1(0) = 1$	$E_1(1) = 0$

In words: if the key bit is 0, the plaintext bit is as-is (not changed), and if the key bit is 1, the plaintext bit is flipped.

Achieving Perfect Secrecy!

	0	1
$(k_1 = 0)$	$E_0(0) = 0$	$E_0(1) = 1$
$(k_2 = 1)$	$E_1(0) = 1$	$E_1(1) = 0$

- Does this remind you of a binary operation?

plaintext	key	ciphertext
0	0	0
1	0	1
0	1	1
1	1	0

- ▷ This is indeed the good-ol' `XOR` operation!

$$c = \text{Enc}_k(m) := m \oplus k$$

Achieving Perfect Secrecy!

But what about decryption?

plaintext	key	ciphertext
0	0	0
1	0	1
0	1	1
1	1	0

▷ It should be straightforward to see that the decryption function is the same as encryption:

$$m = \mathrm{Dec}_k(c) := c \oplus k$$

Achieving Perfect Secrecy!

- In words: if the plaintext bit is “as-is” (so the key=0), we also leave it “as-is”, if it has been “flipped” (so the key=1), we “flip it back”. This is exactly describing XORing with the ciphertext again.
- Or formally:

$$(m \oplus k) \oplus k = m \oplus (k \oplus k) = m \oplus 0 = m$$

Achieving Perfect Secrecy!

Now, suppose we had four possible plaintext messages: `ATTACK FROM NORTH (N)`, `ATTACK FROM SOUTH (S)`, `HOLD POSITION (H)`, and `RETREAT (R)`.

	N	S	H	R
K_1:	$E_{K_1}(\mathtt{N}) = 00$	$E_{K_1}(\mathtt{S}) = 01$	$E_{K_1}(\mathtt{H}) = 10$	$E_{K_1}(\mathtt{R}) = 11$
K_2:	$E_{K_2}(\mathtt{N}) = 01$	$E_{K_2}(\mathtt{S}) = 00$	$E_{K_2}(\mathtt{H}) = 11$	$E_{K_2}(\mathtt{R}) = 10$
K_3:	$E_{K_3}(\mathtt{N}) = 10$	$E_{K_3}(\mathtt{S}) = 11$	$E_{K_3}(\mathtt{H}) = 00$	$E_{K_3}(\mathtt{R}) = 01$
K_4:	$E_{K_4}(\mathtt{N}) = 11$	$E_{K_4}(\mathtt{S}) = 10$	$E_{K_4}(\mathtt{H}) = 01$	$E_{K_4}(\mathtt{R}) = 00$

- What happens if one of the rows is missing?
- What if the same ciphertext is repeated in a row?
- What if the same ciphertext is repeated in a column?

“Latin square”

The above tables are examples of what are known as *Latin Squares*, i.e., tables with the following properties:

1. *Every row contains every table entry exactly once.*
2. *Every column contains every table entry exactly once.*

A Latin square can work as a recipe to achieve “perfect secrecy”. For encryption of a plaintext message:

- ▷ select one of the rows (the “key”) uniformly randomly;
- ▷ look up the ciphertext for the plaintext in that row.

The decryption is as follows:

- ▷ knowing which row is used, the recipient looks up which column in that row the ciphertext appears.

“One-Time Pad”

What happens if the key is not (uniformly) randomly picked for every message?

- *Hint:* consider a scenario where the key is “renewed” after every other plaintext message.
  - At time 1, the message is `H` and the key is K_1, so the attackers see 10.
  - At time 2, the message is `N`, and the key still K_1. So the attackers see 00. What do they learn?

	`N`	`S`	`H`	`R`
K_1:	$E_{K_1}(\texttt{N}) = 00$	$E_{K_1}(\texttt{S}) = 01$	$E_{K_1}(\texttt{H}) = 10$	$E_{K_1}(\texttt{R}) = 11$
K_2:	$E_{K_2}(\texttt{N}) = 01$	$E_{K_2}(\texttt{S}) = 00$	$E_{K_2}(\texttt{H}) = 11$	$E_{K_2}(\texttt{R}) = 10$
K_3:	$E_{K_3}(\texttt{N}) = 10$	$E_{K_3}(\texttt{S}) = 11$	$E_{K_3}(\texttt{H}) = 00$	$E_{K_3}(\texttt{R}) = 01$
K_4:	$E_{K_4}(\texttt{N}) = 11$	$E_{K_4}(\texttt{S}) = 10$	$E_{K_4}(\texttt{H}) = 01$	$E_{K_4}(\texttt{R}) = 00$

"One-Time Pad"

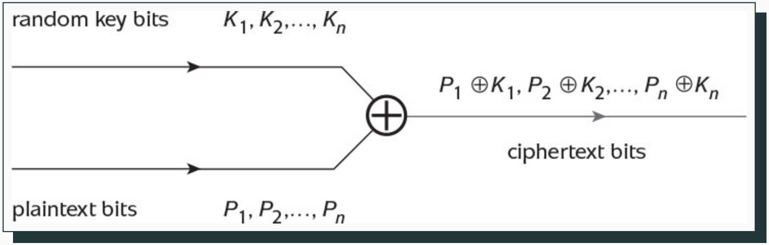
- ▷ As before, we can turn our Latin Square cipher from a looking-up process to an efficient binary operation:

	N=00	S=01	H=10	R=11
00:	$E_{00}(00)=00$	$E_{00}(01)=01$	$E_{00}(10)=10$	$E_{00}(11)=11$
01:	$E_{01}(00)=01$	$E_{01}(01)=00$	$E_{01}(10)=11$	$E_{01}(11)=10$
10:	$E_{10}(00)=10$	$E_{10}(01)=11$	$E_{10}(10)=00$	$E_{10}(11)=01$
11:	$E_{11}(00)=11$	$E_{11}(01)=10$	$E_{11}(10)=01$	$E_{11}(11)=00$

- ▷ Hence, as before, the encryption is `XOR`ing with the key (like a "pad") for both encryption and decryption.
  - Since the key should be uniformly randomly picked for each message, hence the name "one-time pad".

Perfect Secrecy: Vernam Cipher

This can be generalized to as many messages as one likes, in what is known as Vernam Cipher:



- ▷ So, we have a very simple cipher that can be implemented in the simplest hardware (very fast and low power) and achieve “perfect secrecy”! Where is the catch?

Practical Security

Practical Problems with Perfect Secrecy

- ▷ Recall that the Latin Square needs as many rows as there are possible plaintext messages. That is: we need as many keys as there are messages.
- ▷ E.g., for 256 possible messages, i.e., an 8-bit message, one needs 256 keys, i.e., an 8-bit key.
- ▷ To see why this is a problem, suppose you were going to watch a movie (about 1 Gigabyte) online securely. Then you would need to have established a key of size 1 Gigabyte securely somehow! (compare that to only 128 bits that AES typically uses)
- ▷ To make matters worse, once we use that $1GB$ of key, it is now useless!

Theoretical vs. Practical Security

- ▷ In practice, we use only short keys (e.g. 128 bits) for potentially gigabytes of traffic! So we definitely violate “key length being the same length as the message length” and “uniformly selecting a new key per each message”, in other words, we are not using one-time-pad.
- ▷ What does it imply about the “security” of our encryption schemes?
- ▶ **Every practical encryption scheme that we use is (theoretically) insecure!**

Theoretical vs. Practical Security

Unlike perfect secrecy, *practical security* of a cryptosystem is measured in terms of *the difficulty of executing known attacks against it*.

- Note that *known* attacks include *but is not limited to* brute-force (exhaustive key search).
- But how do we measure the *difficulty*? Answer: by comparing *how long* it takes to conduct an attack, given the computational power of an attacker and the *cover time* (the length of time for which a plaintext must be kept secret).

Theoretical vs. Practical Security

- To compute how long it takes to conduct a known attack on a cryptosystem, we need:
    - what computational processes are involved in the attack; and
    - how much time it takes to conduct these processes.
- A well-designed cryptosystem is usually built around a computational problem that is proven or at least widely perceived to be hard to solve.
- This is where the notion of algorithmic “complexity” becomes useful.

Theoretical vs. Practical Security

- The *complexity* of an algorithm (or a process) refers the relation of the number of simple (1-time-slot) machine operations that need to be done to finish the process with respect to the length of the input (usually, its representation in binary).
  - examples of simple (1-time-slot) machine operations: logical operations (AND, OR, XOR, NOT, etc), addition, comparison, etc.

Theoretical vs. Practical Security

Operation	Complexity	Explanation
Addition of two n-bit numbers	n	Ignoring carrying, one addition for every bit.
Multipl. of two n-bit numbers	n^2	n additions, one for every bit of the input numbers.
Raising a number to an n-bit power (using repeated squaring trick)	n^3	n operations, each is either a squaring or a multiplication, both of which take n^2 operations, hence n^3 in total.
Exhaustive search for an n-bit key	2^n	trying out every possible n-bit key, of which there are 2^n.

Theoretical vs. Practical Security

For this module, you just need to appreciate that there two main classes of complexities: *polynomial* vs. *exponential*:

- **polynomial**: if the time taken to execute the process for an input of size n is no greater than n^r, for some r
  - Informally, these are ‘quick’ on all inputs of ‘reasonable’ size. E.g.: multiplication of two n-bit numbers (complexity n^2), raising a number to an n-bit power (n^3), so all ‘easy’ processes.
- **exponential time** if the time taken to execute the process for an input of size n is $\sim a^n$, for some a.
  - Informally, these are ‘too slow’ on all inputs of ‘reasonable’ size, as it becomes *practically impossible* to carry out. E.g. exhaustive search for an n-bit key.

Theoretical vs. Practical Security

$$\text{Est. Attack Time (s)} = \frac{\text{complexity of the process}}{\text{computer speed (\# opeartions/second)}}$$

Complexity	n=10	n=30	n=50
n	0.00001s	0.00003s	0.00005s
n^3	0.001s	0.027s	0.125s
2^n	0.001s	17.9 minutes	37.7 years
3^n	0.059s	6.5 years	200 mil. centuries!

Table 1: Execution time on a (single) processor capable of one million operations/sec. There is a significant difference between processes with polynomial vs. with exponential complexity.

Theoretical vs. Practical Security

So to summarise, for a measure of practical security: We need to make sure that <u>all</u> known attacks will take impractically long to finish. This implies:

- Making sure that the the computational complexity of <u>all</u> known attacks are at least exponential.
- Then choose large enough parameters (e.g. key length) such that (given a ‘reasonable’ estimate about the computational power of attackers), the attack time is ‘sufficiently’ longer than the cover time of the plaintext.

Semantic Security

(side note)

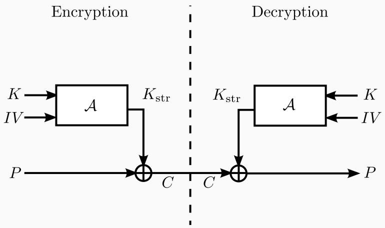
The notion of practical security can still be formalized:

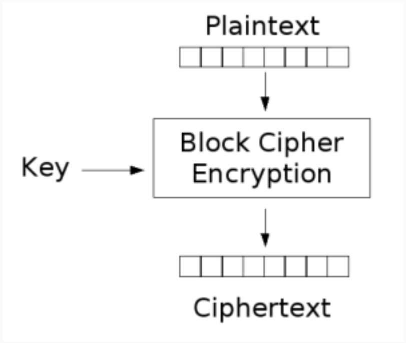
Semantic Security : "a cryptosystem is semantically secure if any probabilistic, polynomial-time algorithm (PPTA) that is given the ciphertext of a certain message `m` (taken from any distribution of messages), and the message's length, cannot determine any partial information on the message with probability non-negligibly higher than all other PPTA's that only have access to the message length (and not the ciphertext)".

Questions?

Symmetric-Key Encryption

Stream Cipher vs Block Cipher





- ▷ **Stream Cipher:** Process one plaintext bit at a time
- ▷ **Block Cipher:** Process a block of plaintext bits at a time.

Stream cipher

A *keystream* (a stream of pseudo-random bits) is generated from a secret *key* and the plaintext bits are XOR'ed with it.

- the keystream generator is a **CSPRNG**.
- tries to mimic the Vernam cipher (Q: what are the similarities and what are the differences?)

Properties of stream ciphers

We say that **error propagation** has occurred if a number of errors in the ciphertext leads to a greater number of errors in the resulting plaintext.

- Q: Where may such errors come from?
- Q: If a steam-cipher is used, a 1 bit of error in the cipher-text results in how many error bits in the deciphered plaintext?
- Q: Based on this property, what kind of application scenario it is good for? (high-error environment/low error environment?)

Properties of stream ciphers

- Speed:
  - XOR is very fast to operate
- On-the-fly encryption
  - large chunks of plaintext do not sit around in registers before being encrypted.
- Implementation efficiency
  - Some stream cipher designs can be implemented in hardware extremely efficiently

For all these reasons, it is used in applications like mobile communications (3G/4G).

- *Disadvantage:* critical need for synchronisation (what does it mean? why?)

Properties of block ciphers

Advantages:

- *Versatility*: not just used for encryption
- *Compatibility*: widely implemented and used in encryption algorithms
- *Adaptability*: can be implemented in different modes of operation

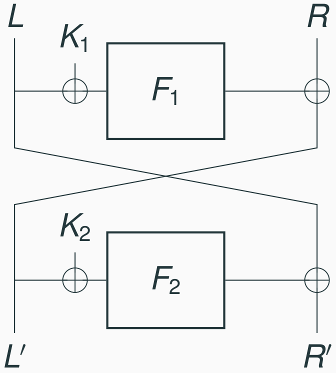
Properties of block ciphers

Disadvantages:

- ▷ *Error propagation:* 1-bit transmission error of a ciphertext block will result in a plaintext block with, on average, half of its bits incorrect (why?)
- ▷ *Need for padding:* Block ciphers operate on fixed block sizes (such as 128 bits) but the length of most plaintexts is not a multiple of the block size, e.g., for a 400-bit plaintext, how many blocks will be needed?

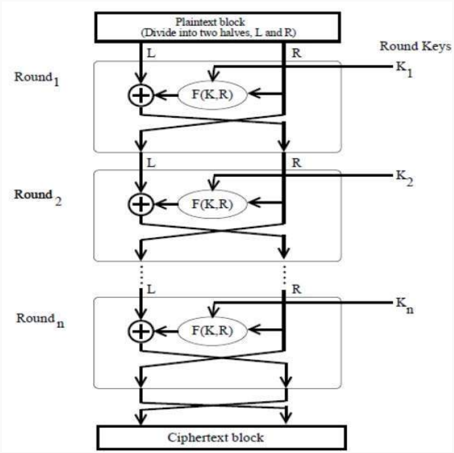
DES

Data Encryption Standard (DES) was the most well-known, well-studied, and well-used block cipher, constructed based on the “Feistel” design model:



Feistel cipher

Feistel encryption:



Q: what is Feistel decryption?

DES: attributes

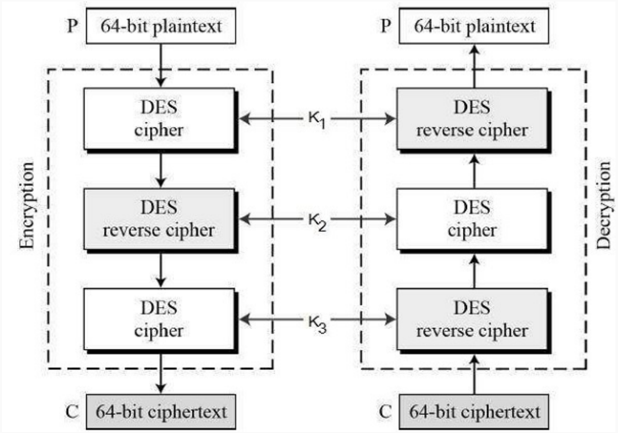
DES was a block cipher with:

- ▷ Block size : 64 bits
- ▷ Key size : 56 bits
- ▷ Number of rounds: 16

Main issue: Inadequate key length. Modern dedicated hardware can search a 56 bits DES key in less than a day!

TDES/3DES

(side-note)
Triple-DES was introduced as an interim solution without replacing DES completely:



AES

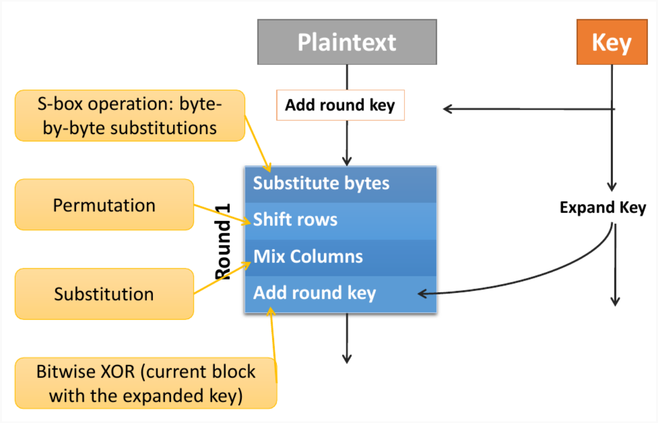
Advanced Encryption Standard (AES) – security standard for symmetric encryption most likely you encounter

- ▷ In 1998, NIST issued a call for proposals for a new block cipher standard to be referred to as the AES.
- ▷ The selection process was by an open public 'competition' and the chosen algorithm and design details made freely available.
- ▷ The **Rijndael**[1] algorithm developed by Joan Daemen and Vincent Rijmen (Belgium) was selected.

[1] pronounced like rain-doll!

AES

AES is an iterated block cipher, based on a “substitution–permutation” network design principle. AES encryption:



AES attributes

- ► Key length: variable: (128-,192-,256- bit).
- ► Block size: 128-bit (fixed)
- ▷ AES performs all its computations on bytes rather than bits. Hence, AES first interprets the 128 bits of a plaintext block as 16 bytes.
- ▷ AES computes a number of rounds. Each of these rounds uses a different 128-bit round key, which is calculated from the original AES key.
- ► The encryption and decryption algorithms of AES have to be separately implemented, although they are very closely related.

AES attributes

- Key length: variable: (128-,192-,256- bit).
- Block size: 128-bit (fixed)
- ▷ AES performs all its computations on bytes rather than bits. Hence, AES first interprets the 128 bits of a plaintext block as 16 bytes.
- ▷ AES computes a number of rounds. Each of these rounds uses a different 128-bit round key, which is calculated from the original AES key.
- The encryption and decryption algorithms of AES have to be separately implemented, although they are very closely related.

AES today

AES is now widely adopted and supported in both hardware and software, including for low-cost environments such as RFID.

- ▷ So far no practical cryptanalytic attacks despite great deal of scrutiny and analysis;
- ▷ has built-in flexibility of key length, which allows a degree of “future-proofing” against exhaustive key searches;
- ▷ However, as for any cryptographic algorithm, the use of AES only guarantees security if it is correctly implemented and good key management is applied.

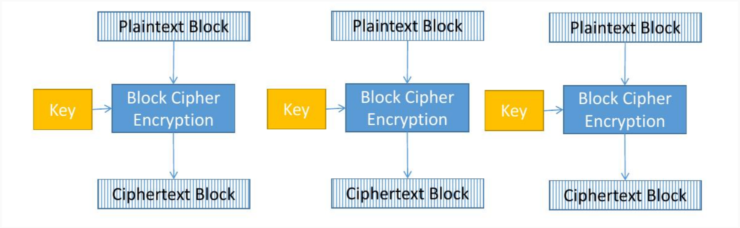
Block-ciphers: Modes of Operation

Modes of operation: a generic block cipher can be used in different “modes” with distinct properties when applied to plaintexts longer than one block.

- ▷ They enable improving the encryption of block ciphers using the same key: A block cipher processes one block of data at a time, using the same key.
- ▷ Without these modes of operations, for a given key, this would produce the same ciphertext for the same plaintexts. **THIS WOULD BE REALLY BAD!** (Q: Why?)

Electronic Code Book (ECB) mode

A block cipher processes one block of data at a time, using the same key:

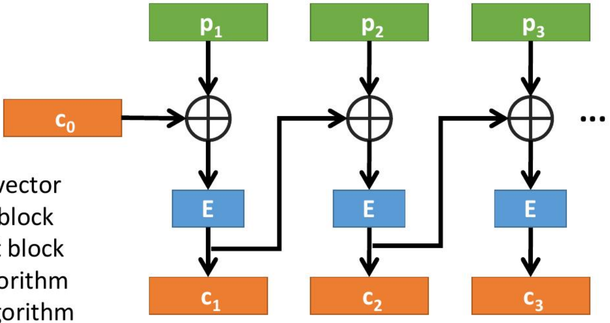


This mode should be avoided as much as possible, esp., when the plaintext is any longer than a single block (why?)

Cipher Block Chaining (CBC) mode

The encryption depends on the encryption's history:

- **Encryption**



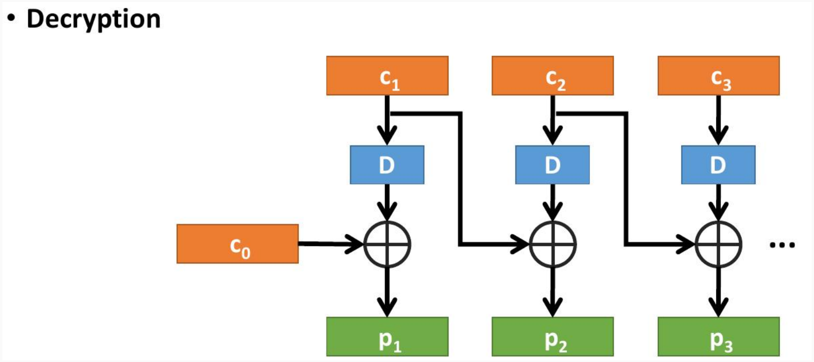
c_0 : Initialisation vector
p_i : *i*-th plaintext block
c_i : *i*-th ciphertext block
E: Encryption algorithm
D: Decryption algorithm

Note: encr. key is the same across encryption blocks.

Q: Does this resolved the problem with ECB mode?

Cipher Block Chaining (CBC) mode

Decryption algorithm can be easily derived from the encryption, and vice versa (how?):



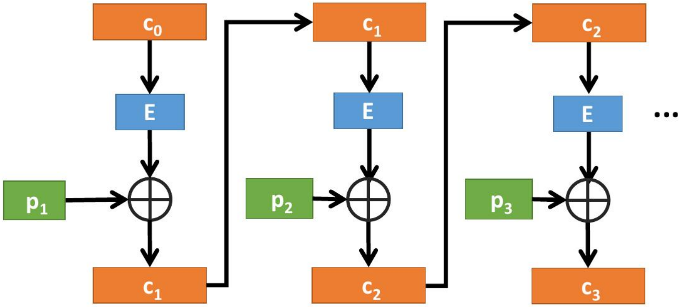
Q: What's the error propagation for CBC mode if 1-bit error occurs during transmission of ciphertext?

Cipher Block Chaining (CBC) mode: Properties

- ▷ **Positional dependency**
- ▷ **Limited error propagation**
- ▷ **No synchronisation required**: in the sense that the receiver could miss the beginning of the ciphertext and still succeed in decrypting from the point the ciphertext is received (why?)
- ▷ **Efficiency**: only slightly less efficient than ECB mode due to the extremely fast XOR operation.
- ▷ **Need for Padding**: Plaintext must be padded before encryption (why?)
- ▷ Implementation advantage: CBC mode forms the basis for a well-known data origin authentication mechanism (CMAC), to be discussed in week 4.

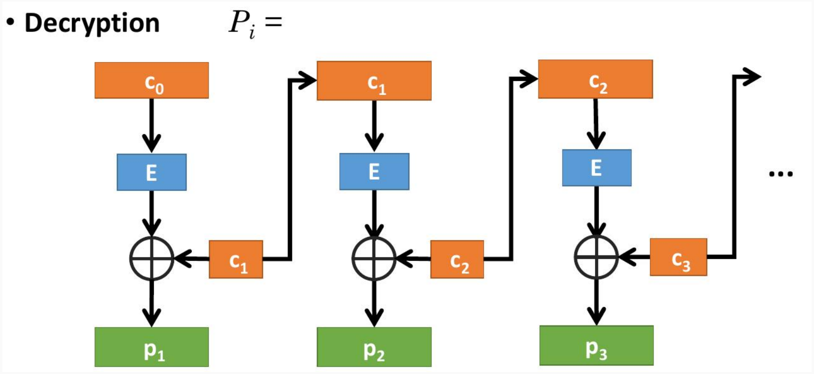
Cipher FeedBack (CFB) mode

- **Encryption** $C_i = P_i \oplus E_K(C_{i-1})$



Cipher FeedBack (CFB) mode

Again, decryption algorithm can be derived from the encryption algorithm and vice-versa (how?)

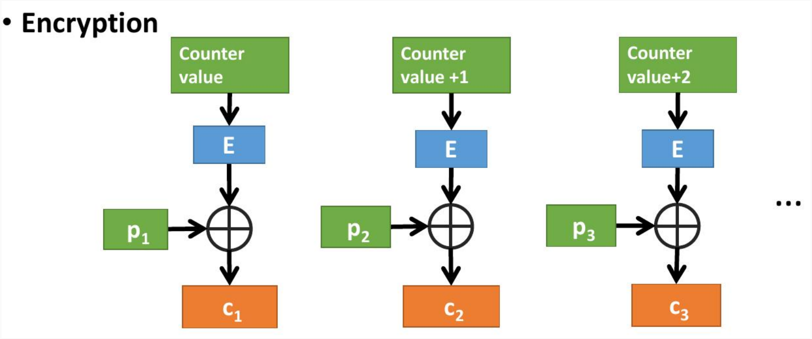


Q: What's the error propagation for CFB mode if 1-bit error occurs during transmission of ciphertext?

Cipher FeedBack (CFB) mode: Properties

- **Positional dependency**
- **Limited error propagation**
- **No synchronisation required**
- **Efficiency**
- **Need for Padding**: no padding is required! (why?)
- **Implementation advantage**: there is no need to implement block cipher decryption, which may save some costs

Counter mode (CTR) mode



- ▷ Both the sender and receiver have access to a reliable counter, which computes a new shared value each time a ciphertext block is exchanged.
- ▷ This shared counter is not necessarily a secret value, but both sides must keep the counters synchronised.

Counter (CTR) mode: Properties

CTR mode preserves most of the other advantages of CFB mode, including the lack of need for padding.

- ▷ Need a synchronous counter
- ▷ Error propagation. Like a dedicated stream cipher, CTR mode has no error propagation (why?)
- ▷ Parallelisation. it can exploit parallel computations, since the slow block cipher encryption computations can be computed independently (in advance) of the actual “encryption” of the plaintext through the fast XOR operation.

Summary

- Perfect secrecy
- Practical secrecy
- Symmetric key standard: AES
- Modes of operations of block-ciphers and their properties.

Next week: Introduction to Public-Key encryption

Questions?