Cryptography

3. Symmetric Encryption Schemes

Contact : `kevin.carrier@cyu.fr`

Plan

1. Strategy 1: Stream Cipher
2. Strategy 2: Block Cipher
3. Block Cipher modes

Introduction

Problem with One-Time-Pad:

- KPA, CPA or CCA attacker models: the attacker can recover the key with only one pair plaintext/ciphertext.
- But we want to reuse the key so as not to have to renew the key for each exchange.

Strategy 1: Stream Cipher

Strategy 1: Stream Cipher

- An algorithm generates the bits of a string cipher one by one
- The encryption/decryption algorithm consists in XORing the message with the string cipher

For instance, there is lots of way to design a Pseudo Random Generator (PRG) function. We use one of them to generate the string cipher and so, the key will be the seed of the PRG function.

Known Plaintext Attacks

If the string cipher is independent on the plaintext, then there are trivial attacks when the attacker knows some pairs plain-text/ciphertext.

Strategy 1: Stream Cipher

- An algorithm generates the bits of a string cipher one by one
- The encryption/decryption algorithm consists in XORing the message with the string cipher

For instance, there is lots of way to design a Pseudo Random Generator (PRG) function. We use one of them to generate the string cipher and so, the key will be the seed of the PRG function.

Known Plaintext Attacks

If the string cipher is independent on the plaintext, then there are trivial attacks when the attacker knows some pairs plain-text/ciphertext.

The string cipher must be dependent on the plaintext

Stream Cipher: Examples

- **A5/1**: Published in 1994 and broken few years later. It is used in the mobile phone network of type GSM, in particular for the radio communication between the phone and the relay antenna. Even if A5/1 has been completely broken, it is still very used in Europe and Africa. It uses 3 LFSR. The cipher key is of size 64 bits (but only 54 bits are non-zero in GSM).
- **RC4**: Designed by Ronald Rivest (very famous cryptologist) in 1987. RC4 is the most used stream cipher to date. It is in particular used in the WEP, WPA, TLS... The size of the cipher key can be between 8 and 2048 bits.
- **Py**: More secure and efficient than RC4. The size of the cipher key is at most 256 bits.
- **E0**: It is in particular used in the Bluetooth protocol. It uses 4 LFSR. The size of the cipher key is 128 bits (but it is actually arbitrary).

Stream Cipher: LFSR

Mathematical recall. The **Finite field** (or **Galois Field**) $\mathbb{F}_2$ (or $GF(2)$) is the set $\{0, 1\}$ equipped with the following operation:

- $+$: the addition modulo 2
- $\times$: the multiplication modulo 2

Stream Cipher: LFSR (definition)

Definition (Linear Feedback Shift Register)

An **LFSR** of length n is made of:

- an **initial state** : $\overrightarrow{\mathbf{r}_0} := (r_{n-1}, r_{n-2}, \cdots, r_1, r_0) \in \mathbb{F}_2^n$
- a **feedback polynomial** : $P(X) := 1 + c_1X + c_2X^2 + \cdots + c_nX^n \in \mathbb{F}_2[X]_{\leq n}$

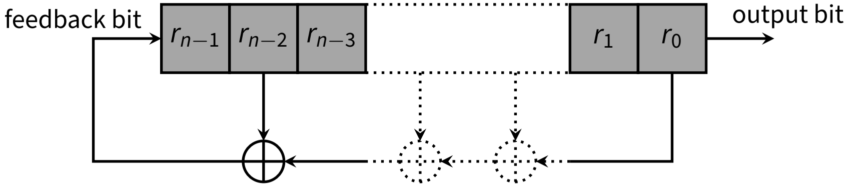
The sequence of registers $\left(\overrightarrow{\mathbf{r}_t}\right)_{t\geq 0}$ is computed recursively: for each instant $t \geq 0$, $\overrightarrow{\mathbf{r}_{t+1}} := (r_{t+n}, r_{t+n-1}, \cdots, r_{t+2}, r_{t+1}) \in \mathbb{F}_2^n$ where

$$r_{t+n} := \sum_{i=1}^{n} c_i \times r_{t+n-i}$$

The cipher string is $(r_t)_{t\geq 0} := (r_0, r_1, \cdots, r_t, \cdots)$

Stream Cipher: LFSR (graphical representation)

The LFSR are very easy to implement, in particular thanks to a representation of them which uses **logical circuit**:



Stream Cipher: LFSR (periodicity)

We want to avoid to repeat the same pattern in the cipher string.

Can the cipher string generated by an LFSR be of arbitrary length before to be repeated?

Stream Cipher: LFSR (periodicity)

We want to avoid to repeat the same pattern in the cipher string.

Can the cipher string generated by an LFSR be of arbitrary length before to be repeated? **No!**

Stream Cipher: LFSR (periodicity)

We want to avoid to repeat the same pattern in the cipher string.

Can the cipher string generated by an LFSR be of arbitrary length before to be repeated? **No!**

The value of the register of an LFSR is in $\mathbb{F}_2^n$ which is of cardinality 2^n.

So the number of value that can take the register is **finite**...

Theorem (period of an LFSR)

The maximal period of an LFSR is $2^n - 1$.

Moreover, if the feedback polynomial is of degree n and irreducible, then the LFSR achieves the maximal period $2^n - 1$ *for any non-zero initial register.*

Proof. It is the maximal number of different registers we can have. The second part of the theorem is admit.

Stream Cipher: LFSR (attack)

Berlekamp-Massey

The **Berlekamp-Massey algorithm** can find the feedback polynomial by only knowing $2n$ bits of the cipher string.

Proof. It essentially consists in solving a linear system of n equations and n unknowns (see exercise sheet).

Consequences. Attacks in the KPA, CPA, CCA models or even any attacker models where one pair plaintext/ciphertext is known.

Stream Cipher: LFSR (A5/1)

A5/1 is a standard encryption of the **GSM**. Even if it is **broken** today, it is still used in particular in Europe and Africa.

A5/1 combines the output of 3 LFSRs $\mathcal{L}_1$, $\mathcal{L}_2$ and $\mathcal{L}_3$ having respectively the following feedback polynomials:

$$\begin{aligned} P_1 &= 1 + X^{14} + X^{17} + X^{18} + X^{19} \\ P_2 &= 1 + X^{21} + X^{22} \\ P_3 &= 1 + X^{8} + X^{21} + X^{22} + X^{23} \end{aligned}$$

Stream Cipher: LFSR (A5/1)

The LFSRs are not incremented synchronously... One bit per register is used to know if a LFSR has to be incremented or not at an instant t.

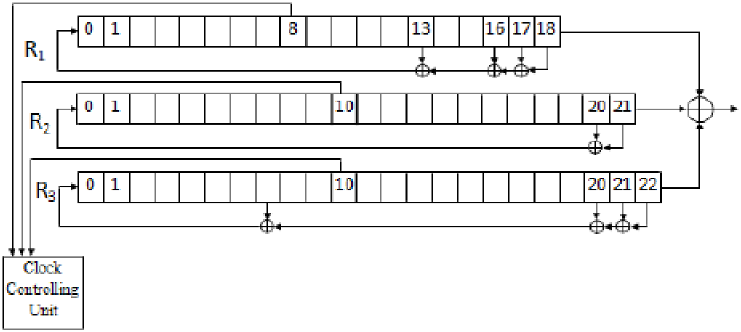
Those bits are called *clock bits* and are:

- h_1: the 9th bit of the register of $\mathcal{L}_1$;
- h_2: the 11th bit of the register of $\mathcal{L}_2$;
- h_3: the 11th bit of the register of $\mathcal{L}_3$;

$$majority(h_1, h_2, h_3) = \begin{cases} 1 & \text{if } h_1 + h_2 + h_3 \geq 2 \\ 0 & \text{if } h_1 + h_2 + h_3 < 2 \end{cases}$$

If $h_i = majority(h_1, h_2, h_3)$, then the register of $\mathcal{L}_i$ will be updated at the next instant. Otherwise, it is unchanged.

Stream Cipher: LFSR (A5/1)



Exercise

Exercise sheet on stream cipher

Strategy 2: Block Cipher

Strategy 2: Block Cipher

- A message of **fixed size** is encrypted.
- If the message to encrypt is longer than the size of the encryption scheme, then the encryption is **repeated as often as necessary**.
- If the size of the message is not a multiple of the size of the encryption scheme, then a **padding** is added at the end of the message. It is usually a random string or a zero string.

Advantages.

- Blocks can be addressed in parallel and in no particular order.
- It is easier to make the ciphertext dependent on the plaintext.

Block Cipher: Highly non-linear functions

Let an encryption scheme with blocksize n bits.

In a very abstract way, we have:

$$\mathbf{c} = S_{\mathbf{k}}(\mathbf{m})$$

where

- $\mathbf{m} \in \mathbb{F}_2^n$ is the plaintext
- $\mathbf{c} \in \mathbb{F}_2^n$ is the ciphertext
- $S_{\mathbf{k}} : \mathbb{F}_2^n \to \mathbb{F}_2^n$ is a function which is parameterized by $\mathbf{k} \in \mathbb{F}_2^t$

The whole question is how to design the function $S_{\mathbf{k}}$ with as little (mathematical) structure as possible?

Block Cipher: Highly non-linear functions

Recall.

Definition (linear function)

The function $S_{\mathbf{k}} : \mathbb{F}_2^n \to \mathbb{F}_2^n$ is linear if and only if for all messages $\mathbf{m}_1, \mathbf{m}_2 \in \mathbb{F}_2^n$ and any scalar $\lambda \in \mathbb{F}_2$:

$$\begin{aligned} S_{\mathbf{k}}(\mathbf{m}_1 + \mathbf{m}_2) &= S_{\mathbf{k}}(\mathbf{m}_1) + S_{\mathbf{k}}(\mathbf{m}_2) \\ S_{\mathbf{k}}(\lambda \mathbf{m}_1) &= \lambda S_{\mathbf{k}}(\mathbf{m}_1) \end{aligned}$$

Block Cipher: Highly non-linear functions

Definition (non-linear function)

A function is non-linear if and only if it is not linear.

A function $S_{\mathbf{k}} : \mathbb{F}_2^n \to \mathbb{F}_2^n$ is **Highly non-linear** if for *most* messages $\mathbf{m}_1, \mathbf{m}_2 \in \mathbb{F}_2^n$ and *most* scalars $\lambda \in \mathbb{F}_2$:

$$\begin{aligned} S_{\mathbf{k}}(\mathbf{m}_1 + \mathbf{m}_2) &\neq S_{\mathbf{k}}(\mathbf{m}_1) + S_{\mathbf{k}}(\mathbf{m}_2) \\ S_{\mathbf{k}}(\lambda \mathbf{m}_1) &\neq \lambda S_{\mathbf{k}}(\mathbf{m}_1) \end{aligned}$$

Block Cipher: Highly non-linear functions

A Highly non-linear function must verify the two following principles:

- **Confusion**: The output must be very different from the input.
  More formally, $M + S_{\mathbf{k}}(M)$ must be distributed uniformly at random in $\mathbb{F}_2^n$ if M is the uniform distribution over $\mathbb{F}_2^n$.
- **Diffusion**: The output must depend on every bits of the input.
  That is, $S_{\mathbf{k}}(M) + S_{\mathbf{k}}(M + U)$ must be distributed uniformly at random in $\mathbb{F}_2^n$ if M is the uniform distribution over $\mathbb{F}_2^n$ and U is the uniform distribution on the sphere $\{\mathbf{x} \in \mathbb{F}_2 \ : \ |\mathbf{x}| = 1\}$.

Block Cipher: Highly non-linear functions

It is quite easy to design a highly non-linear function $S_{\mathbf{k}} : \mathbb{F}_2^n \to \mathbb{F}_2^n$. One only has to generate a random bijection ; a such bijection is simply a shuffle of the table containing (binary representation of) integers $\{0, \cdots, 2^n - 1\}$.

Block Cipher: Highly non-linear functions

It is quite easy to design a highly non-linear function $S_{\mathbf{k}} : \mathbb{F}_2^n \to \mathbb{F}_2^n$. One only has to generate a random bijection ; a such bijection is simply a shuffle of the table containing (binary representation of) integers $\{0, \cdots, 2^n - 1\}$.

- The key **k** must describe $S_{\mathbf{k}}$; so if we do not use a pseudo-random generator to make the shuffle, then **the key is actually a whole description of the shuffled table**. Thus, the key is of size $n \times 2^n$

Block Cipher: Highly non-linear functions

It is quite easy to design a highly non-linear function $S_{\mathbf{k}} : \mathbb{F}_2^n \to \mathbb{F}_2^n$. One only has to generate a random bijection ; a such bijection is simply a shuffle of the table containing (binary representation of) integers $\{0, \cdots, 2^n - 1\}$.

- The key **k** must describe $S_{\mathbf{k}}$; so if we do not use a pseudo-random generator to make the shuffle, then **the key is actually a whole description of the shuffled table**. Thus, the key is of size $n \times 2^n$
- **Brute force attack**: To recover the shuffled table, we need to know 2^n pairs plaintext/ciphertext. To achieve a security of 80 bits, we need $n = 80$ and so the key would be a table of 2^{80} entries...

Block Cipher: Highly non-linear functions

It is quite easy to design a highly non-linear function $S_{\mathbf{k}} : \mathbb{F}_2^n \to \mathbb{F}_2^n$. One only has to generate a random bijection ; a such bijection is simply a shuffle of the table containing (binary representation of) integers $\{0, \cdots, 2^n - 1\}$.

- The key **k** must describe $S_{\mathbf{k}}$; so if we do not use a pseudo-random generator to make the shuffle, then **the key is actually a whole description of the shuffled table**. Thus, the key is of size $n \times 2^n$
- **Brute force attack**: To recover the shuffled table, we need to know 2^n pairs plaintext/ciphertext. To achieve a security of 80 bits, we need $n = 80$ and so the key would be a table of 2^{80} entries...
- Change the key is tedious... We would like to be able to change the key easily. Here, we need to generate a shuffled table of 2^n entries each time we want to change the key.

Block Cipher: Highly non-linear functions

How to design a highly non-linear function in practical?

1. Using a key $\mathbf{k} \in \mathbb{F}_2^n$, we build a highly non-linear function:

$$S_{\mathbf{k}}(\mathbf{m}) := S(\mathbf{m} + \mathbf{k})$$

   where S is a fixed highly non-linear function which doesn't depend on $\mathbf{k}$. S is called **Substitution Box**.

2. We combine several small substitution boxes using **Permutations** (non-linear functions).
3. We iterate 1. and 2. several times. One iteration is called **Round**.

Remark

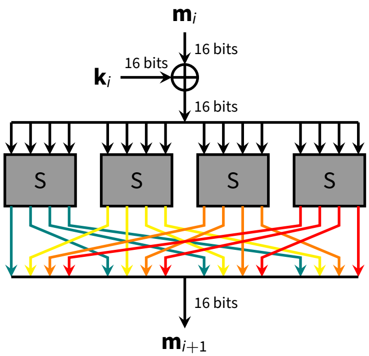
$$\text{1. and 3.} \implies \text{Confusion}$$
$$\text{2. and 3.} \implies \text{Diffusion}$$

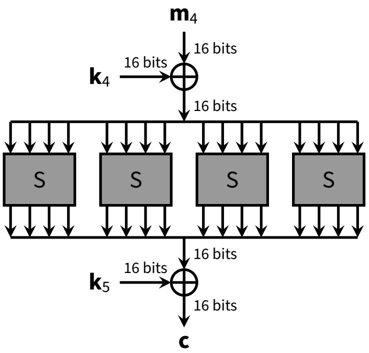
Block Cipher: Highly non-linear functions (Exercise)

Let S be a substitution box in $\mathbb{F}_2^4$:

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S(x)$	1	3	5	7	9	B	D	F	0	2	4	6	8	A	C	E

$\forall i \in \{1, \cdots, 3\}$ and with $\mathbf{m}_1 := \mathbf{m}$:





Exercise

Exercise sheet on linear and differential cryptanalysis

Block Cipher: Feistel Scheme

To decipher, we need to be able to invert “easily” $S_{\mathbf{k}}$. To do that, we use the Feistel Scheme.

Block Cipher: Feistel Scheme

To decipher, we need to be able to invert “easily” $S_{\mathbf{k}}$. To do that, we use the Feistel Scheme.

Let $F(\mathbf{k}, \cdot)$ be a non-linear operator parameterized by a key $\mathbf{k}$.

We consider here a plaintext $\mathbf{m} := \mathbf{m}_1 || \mathbf{m}_2$ of length $2n$ bits ($len(\mathbf{m}_1) = len(\mathbf{m}_2) = n$).

We compute the ciphertext $\mathbf{c} := \mathbf{c}_1 || \mathbf{c}_2$ such that:

$$\begin{aligned} \mathbf{c}_1 &= \mathbf{m}_2 \\ \mathbf{c}_2 &= F(\mathbf{k}, \mathbf{m}_2) + \mathbf{m}_1 \end{aligned}$$

Then we repeat the operation for several rounds.

Block Cipher: Feistel Scheme (more formally)

Let $F(\mathbf{k}, \cdot)$ be a non-linear operator parameterized by a key $\mathbf{k}$.

We consider here a plaintext $\mathbf{m} := \mathbf{m}_1 || \mathbf{m}_2$ of length $2n$ bits ($len(\mathbf{m}_1) = len(\mathbf{m}_2) = n$).

We compute recursively the series $\left(\mathbf{c}^{(i)}\right)_{i=0..N}$ where N is the number of rounds:

- **Initialization.** $\mathbf{c}^{(0)} := \mathbf{c}_1^{(0)} || \mathbf{c}_2^{(0)} := \mathbf{m}_1 || \mathbf{m}_2$
- **Induction.** $\forall i \in \{1, \cdots, N-1\}, \mathbf{c}^{(i+1)} := \mathbf{c}_1^{(i+1)} || \mathbf{c}_2^{(i+1)}$ where

$$\begin{aligned} \mathbf{c}_1^{(i+1)} &= \mathbf{c}_2^{(i)} \\ \mathbf{c}_2^{(i+1)} &= F(\mathbf{k}_i, \mathbf{c}_2^{(i)}) + \mathbf{c}_1^{(i)} \end{aligned}$$

The ciphertext is then $\mathbf{c}^{(N)}$.

Block Cipher: Feistel Scheme

Remark

The various partial keys $\mathbf{k}_i$ are generated by deriving a main key $\mathbf{k}$ using a Pseudo Random Generator.

Theorem (Decryption with a Feistel Scheme)

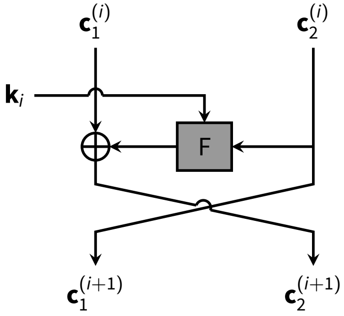
To decipher a ciphertext that has been encrypted with a Feistel Scheme, one only has to apply the same Feistel Scheme by inverting the order of using the partial keys:

$$\mathbf{k}'_i = \mathbf{k}_{N-i}$$

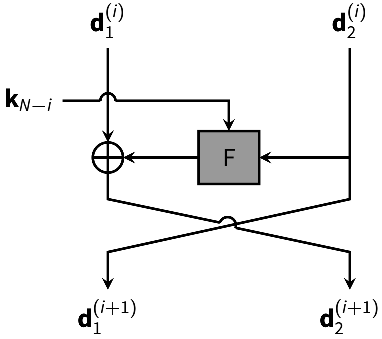
where the $\mathbf{k}'_i$ are the partial decipher keys.

Block Cipher: Feistel Scheme (more graphically)

Encryption:



Decryption:



where $\mathbf{c}_1^{(0)}||\mathbf{c}_2^{(0)} := \mathbf{d}_1^{(N)}||\mathbf{d}_2^{(N)} := \mathbf{m}$

and $\mathbf{c}_1^{(N)}||\mathbf{c}_2^{(N)} := \mathbf{d}_1^{(0)}||\mathbf{d}_2^{(0)} := \mathbf{c}$

DES: Data Encryption Standard

See the document **DES.pdf**

AES: Advanced Encryption Standard

- It encrypts a **128-bits plaintext** into a **128-bits ciphertext**
- The size of the **main key** can be **128 bits, 192 bits or 256 bits**

Size of the main key & number of rounds

The size of the key is relied to the number of rounds to achieve:

$$128 \text{ bits} \longleftrightarrow 11 \text{ rounds}$$
$$192 \text{ bits} \longleftrightarrow 13 \text{ rounds}$$
$$256 \text{ bits} \longleftrightarrow 15 \text{ rounds}$$

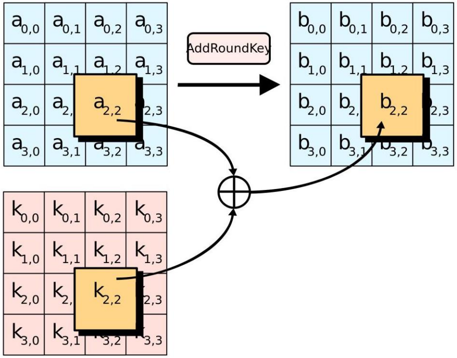
AES: Advanced Encryption Standard

High level description of the encryption algorithm:

```
1.    KeyExpansion
2.    AddRoundKey
3.    repeat 9, 11 or 13 times:
4.        SubBytes
5.        ShiftRows
6.        MixColumns
7.        AddRoundKey
8.    SubBytes
9.    ShiftRows
10.   AddRoundKey
```

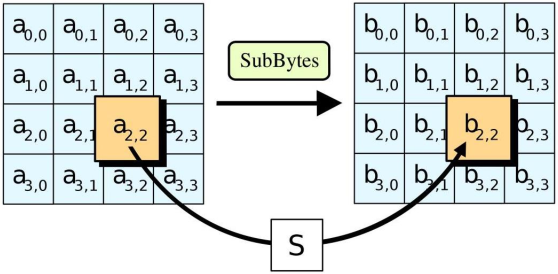
AES: AddRoundKey

In the AddRoundKey step, each byte of the state is combined with a byte of the round subkey using the XOR operation.


AddRoundKey
$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,0}$ $a_{1,1}$ $a_{1,2}$ $a_{1,3}$
$a_{2,0}$ $a_{2,1}$ $a_{2,2}$ $a_{2,3}$
$a_{3,0}$ $a_{3,1}$ $a_{3,2}$ $a_{3,3}$
$b_{0,0}$ $b_{0,1}$ $b_{0,2}$ $b_{0,3}$
$b_{1,0}$ $b_{1,1}$ $b_{1,2}$ $b_{1,3}$
$b_{2,0}$ $b_{2,1}$ $b_{2,2}$ $b_{2,3}$
$b_{3,0}$ $b_{3,1}$ $b_{3,2}$ $b_{3,3}$
$k_{0,0}$ $k_{0,1}$ $k_{0,2}$ $k_{0,3}$
$k_{1,0}$ $k_{1,1}$ $k_{1,2}$ $k_{1,3}$
$k_{2,0}$ $k_{2,1}$ $k_{2,2}$ $k_{2,3}$
$k_{3,0}$ $k_{3,1}$ $k_{3,2}$ $k_{3,3}$
$\oplus$

AES: SubBytes

In the SubBytes step, each byte in the state is replaced with its entry in a fixed 8-bit lookup table S; $b_{i,j} = S(a_{i,j})$.

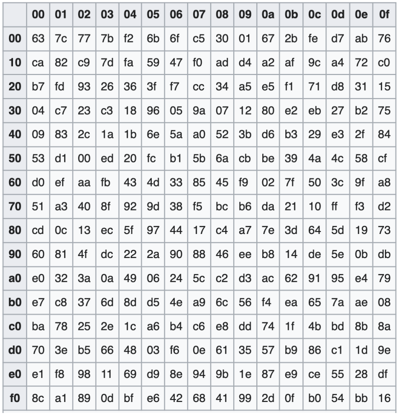


$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,0}$ $a_{1,1}$ $a_{1,2}$ $a_{1,3}$
$a_{2,0}$ $a_{2,1}$ $a_{2,2}$ $a_{2,3}$
$a_{3,0}$ $a_{3,1}$ $a_{3,2}$ $a_{3,3}$
SubBytes
$b_{0,0}$ $b_{0,1}$ $b_{0,2}$ $b_{0,3}$
$b_{1,0}$ $b_{1,1}$ $b_{1,2}$ $b_{1,3}$
$b_{2,0}$ $b_{2,1}$ $b_{2,2}$ $b_{2,3}$
$b_{3,0}$ $b_{3,1}$ $b_{3,2}$ $b_{3,3}$
S

source: wikipedia

AES: SubBytes (AES-S-Box)

The *S* function is computed thanks to the following table. The column is determined by the least significant nibble, and the row by the most significant nibble. For example, the value 0*x*9*A* is converted into 0*xB*8.

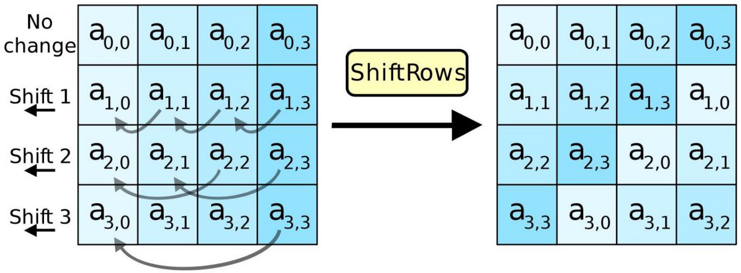


	00	**01**	**02**	**03**	**04**	**05**	**06**	**07**	**08**	**09**	**0a**	**0b**	**0c**	**0d**	**0e**	**0f**
00	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
10	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
20	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
30	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
40	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
50	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
60	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
70	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
80	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
90	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a0	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b0	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c0	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d0	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e0	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f0	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

source: wikipedia

AES: ShiftRows

In the ShiftRows step, bytes in each row of the state are shifted cyclically to the left. The number of places each byte is shifted differs incrementally for each row.

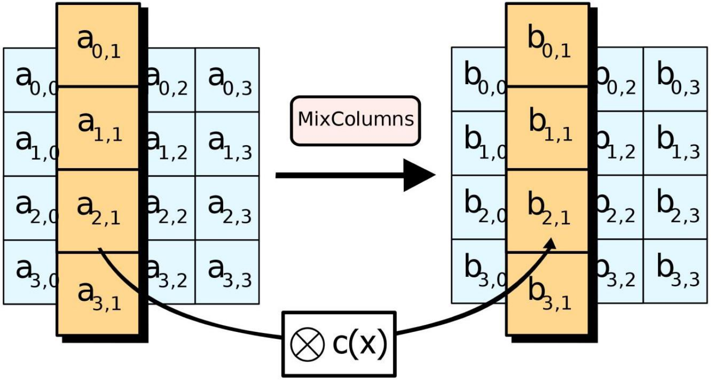


No change
Shift 1
Shift 2
Shift 3
$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,0}$ $a_{1,1}$ $a_{1,2}$ $a_{1,3}$
$a_{2,0}$ $a_{2,1}$ $a_{2,2}$ $a_{2,3}$
$a_{3,0}$ $a_{3,1}$ $a_{3,2}$ $a_{3,3}$
ShiftRows
$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,1}$ $a_{1,2}$ $a_{1,3}$ $a_{1,0}$
$a_{2,2}$ $a_{2,3}$ $a_{2,0}$ $a_{2,1}$
$a_{3,3}$ $a_{3,0}$ $a_{3,1}$ $a_{3,2}$

source: wikipedia

AES: MixColumns

In the MixColumns step, each column of the state is multiplied with a fixed polynomial $c(x)$.



$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,0}$ $a_{1,1}$ $a_{1,2}$ $a_{1,3}$
$a_{2,0}$ $a_{2,1}$ $a_{2,2}$ $a_{2,3}$
$a_{3,0}$ $a_{3,1}$ $a_{3,2}$ $a_{3,3}$
MixColumns
$\otimes c(x)$
$b_{0,0}$ $b_{0,1}$ $b_{0,2}$ $b_{0,3}$
$b_{1,0}$ $b_{1,1}$ $b_{1,2}$ $b_{1,3}$
$b_{2,0}$ $b_{2,1}$ $b_{2,2}$ $b_{2,3}$
$b_{3,0}$ $b_{3,1}$ $b_{3,2}$ $b_{3,3}$

source: wikipedia

AES: MixColumns (more details)

Here, operations are made in $\mathbb{F}_{2^8}[Z]$

where $\mathbb{F}_{2^8}$ is viewed as $\mathbb{F}_2[X]/(X^8 + X^4 + X^3 + X + 1)$

Then a bytes has several representations:

polynomial	binary	hexadecimal	integer
$X^6 + X^5 + X^3 + X^1 + 1$	$0b01101011$	$0x6B$	107

AES: MixColumns (more details)

Here, operations are made in $\mathbb{F}_{2^8}[Z]$

where $\mathbb{F}_{2^8}$ is viewed as $\mathbb{F}_2[X]/(X^8 + X^4 + X^3 + X + 1)$

Then a bytes has several representations:

polynomial	binary	hexadecimal	integer
$X^6 + X^5 + X^3 + X^1 + 1$	$0b01101011$	$0x6B$	107

$$\begin{aligned} & b_{0,j} + b_{1,j}Z + b_{2,j}Z^2 + b_{3,j}Z^3 \\ & \qquad := \left(a_{0,j} + a_{1,j}Z + a_{2,j}Z^2 + a_{3,j}Z^3\right) \times c(Z) \mod \left(Z^4 + 1\right) \end{aligned}$$

where

$$c(Z) := 3Z^3 + Z^2 + Z + 2$$

AES: KeyExpansion

- N: the length of the key in 32-bit words: 4 words for AES-128, 6 words for AES-192, and 8 words for AES-256
- R: the number of round keys needed: 11 round keys for AES-128, 13 keys for AES-192, and 15 keys for AES-256
- $K_0, K_1, \cdots, K_{N-1}$: the 32-bit words of the original key
- $W_0, W_1, \cdots, W_{4R-1}$: the 32-bit words of the expanded key

AES: KeyExpansion

- *RotWord*: a one-byte left circular shift

$$RotWord([b_0, b_1, b_2, b_3]) = [b_1, b_2, b_3, b_0]$$

- *SubWord*: an application of the AES S-box to each of the four bytes of the word

$$SubWord([b_0, b_1, b_2, b_3]) = [S(b_0), S(b_1), S(b_2), S(b_3)]$$

- $rcon_i$: for each $i \in \{1, \cdots, 10\}$, it is a 32-bits constant word defined as

i	1	2	3	4	5	6	7	8	9	10
$rcon_i$	$0x01000000$	$0x02000000$	$0x04000000$	$0x08000000$	$0x10000000$	$0x20000000$	$0x40000000$	$0x80000000$	$0x1B000000$	$0x36000000$

AES: KeyExpansion

For all $i \in \{0, \cdots, 4R - 1\}$:

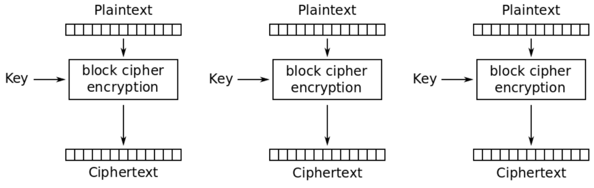
$$W_i := \begin{cases} K_i & \text{if } i < N \\ W_{i-N} \oplus \mathit{SubWord}(\mathit{RotWord}(W_{i-1})) \oplus \mathit{rcon}_i & \text{if } i \geq N \text{ and } i \equiv 0 \bmod N \\ W_{i-N} \oplus \mathit{SubWord}(W_{i-1}) & \text{if } i \geq N > 6 \text{ and } i \equiv 4 \bmod N \\ W_{i-N} \oplus W_{i-1} & \text{otherwise} \end{cases}$$

AES (Exercise)

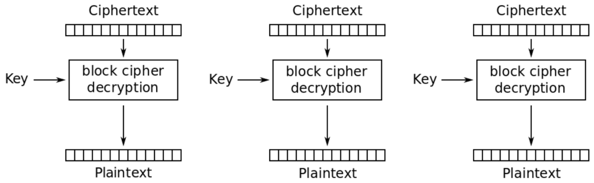
1. Is AES a Feistel's network? Describe the decryption algorithm.
2. In AddRoundKey, what are exactly the $k_{i,j}$ bytes? Why do we need to expand the key into $4R - 1$ words of 32 bits ?
3. What guarantees confusion principle? Justify.
4. What guarantees diffusion principle? Justify.
5. What is a cost of a brute force attack on AES-128? on AES-256?

Block Cipher modes

Block Cipher modes: ECB



Electronic Codebook (ECB) mode encryption



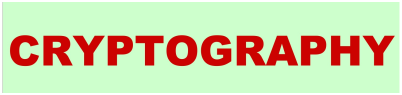
Electronic Codebook (ECB) mode decryption

source: wikipedia

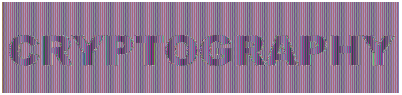
Block Cipher modes: ECB

- The plaintext is divided into blocks that are encrypted separately.
- **Lack of diffusion**.
- **two same blocks** are encrypted **identically**.
- **not recommended**.

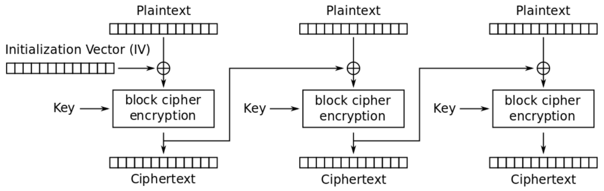
Original image:



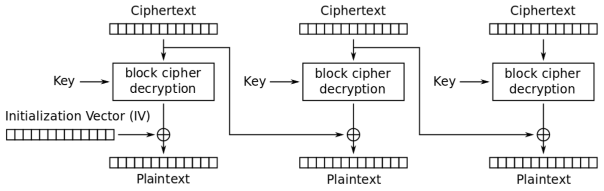
AES-128 (ECB mode):



Block Cipher modes: CBC



Cipher Block Chaining (CBC) mode encryption



Cipher Block Chaining (CBC) mode decryption

source: wikipedia

Block Cipher modes: CBC

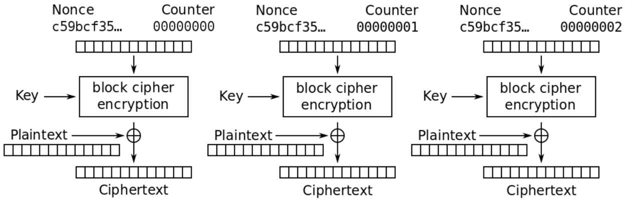
- Fixes the lack of diffusion problem.
- Encryption cannot be parallelized
- Decryption can be parallelized

Attack on the IV

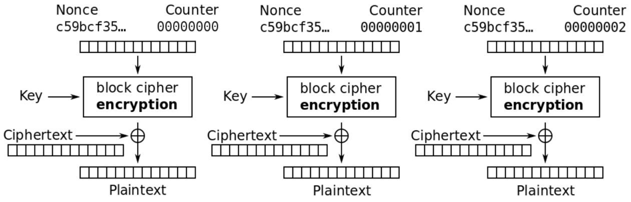
The **Initialization Vector** must be **random**, **secret** and it **must not be reused** with the same key.

Exercise. Build the attack on the IV.

Block Cipher modes: CTR



Counter (CTR) mode encryption



Counter (CTR) mode decryption

Block Cipher modes: comparison

Type	Algorithm	Key size (bits)	Speed (without materiel)	Speed (with materiel)	Advantages	Disadvantages
Block	AES-GCM	128-192-256	~ 100 Mo/s (integrity included)	~ 2 000 Mo/s (integrity included)	• Fast • Parallelizable • Authentication encryption (AEAD) • No padding	• Do not reuse a counter
Block	AES-CBC	128-192-256	~ 120 Mo/s	~ 2 700 Mo/s	• Parallelizable Decryption • Errors Propagation (for Message Authentication Code)	• Encryption cannot be parallelized • unique and secret IV • Padding
Block	AES-CTR	128-192-256	~ 150 Mo/s	~ 3 000 Mo/s	• Fast • Parallelizable • No padding	• No integrity • Do not reuse a counter
Stream	Salsa20 or ChaCha	128-256	~ 700 Mo/s	~ 700 Mo/s	• Very fast without material support • No padding	• Do not reuse a nonce • Malleable

Exercise

Exercise sheet on block cipher