

Cryptography and Embedded System Security

CRAESS_I

Xiaolu Hou

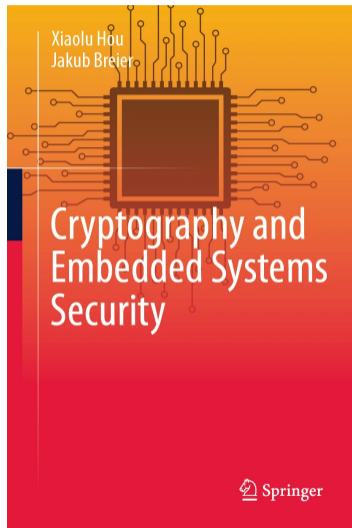
FIIT, STU
xiaolu.hou @ stuba.sk

Course Outline

- Abstract algebra and number theory
- Introduction to cryptography
- Symmetric block ciphers and their implementations
- RSA, RSA signatures, and their implementations
- Probability theory and introduction to SCA
- SPA and non-profiled DPA
- Profiled DPA
- SCA countermeasures
- FA on RSA and countermeasures
- FA on symmetric block ciphers
- FA countermeasures for symmetric block cipher
- Practical aspects of physical attacks
 - Invited speaker: Dr. Jakub Breier, Senior security manager, TTControl GmbH

Recommended reading

- Textbook
 - Sections 3.1, 3.2



Lecture Outline

- Introduction
- DES
- AES
- PRESENT
- Bitsliced implementations
- Implementation of round functions

Symmetric block ciphers and their implementations

- Introduction
- DES
- AES
- PRESENT
- Bitsliced implementations
- Implementation of round functions

Cryptosystem

Definition

A *cryptosystem* is a tuple $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$ with the following properties:

- \mathcal{P} is a finite set of plaintexts, called *plaintext space*.
- \mathcal{C} is a finite set of ciphertexts, called *ciphertext space*.
- \mathcal{K} is a finite set of keys, called *key space*.
- $\mathcal{E} = \{ E_k : k \in \mathcal{K} \}$, where $E_k : \mathcal{P} \rightarrow \mathcal{C}$ is an *encryption function*.
- $\mathcal{D} = \{ D_k : k \in \mathcal{K} \}$, where $D_k : \mathcal{C} \rightarrow \mathcal{P}$ is a *decryption function*.
- For each $e \in \mathcal{K}$, there exists $d \in \mathcal{K}$ such that $D_d(E_e(p)) = p$ for all $p \in \mathcal{P}$.

If $e = d$, the cryptosystem is called a *symmetric key cryptosystem*. Otherwise, it is called a *public-key/asymmetric cryptosystem*.

- In general, symmetric key ciphers are faster, but they require key exchange before communication.

Modern symmetric block cipher

- $\mathcal{P} = \mathcal{C} = \mathbb{F}_2^n$ for a positive integer n , which is called the *block length* of the cipher.
- The key space is also a vector space over \mathbb{F}_2 and its dimension is called the *key length* of the cipher.
- Each key $k \in \mathcal{K}$ is called a *master key*.

Key length

- We have seen that exhaustive key search can be used to break shift cipher and affine cipher
 - key space should be big enough so that the attacker cannot brute force
- Determined by the current computation power
 - The 56-bit secret key of DES was successfully cracked in 1998
 - The U.S. National Institute for Standards and Technology (NIST) issues recommendations for key sizes for government institutions in the USA. According to those recommendations
 - 80-bit keys were “retired” in 2010
 - lesser than 112-bit keys were considered insufficient from 2015 onward
 - National Security Agency (NSA) currently requires AES-256 for Top Secret classification since 2015 due to the emergence of quantum computing

Confusion and diffusion

- For the construction of symmetric block ciphers, two important principles are followed by modern cryptographers – *confusion* and *diffusion*.
- Shannon first introduced them in his famous paper¹
- Confusion obscures the relationship between the ciphertext and the key. To achieve this, each part of the ciphertext should depend on several parts of the key.
- Diffusion obscures the statistical relationship between the plaintext and the ciphertext. Each change in the plaintext is spread over the ciphertext, with the redundancies being dissipated.
 - For example, affine cipher has very low diffusion – the distributions of letters in plaintext correspond directly to those in the ciphertext.
 - That is also why frequency analysis can be applied to break those ciphers.

¹Claude E Shannon. A mathematical theory of cryptography. Mathematical Theory of Cryptography, 1945

Block cipher design specifications

- A symmetric block cipher design specifies a *round function* and a *key schedule*.
- Encryption of a plaintext block consists of a few *rounds* of round functions, possibly with minor differences.
- Each round function takes the cipher's current *state* as an input and outputs the next state.
- The key schedule takes the master key k and outputs the keys for each round, which are called *round keys*.
- In most cases, the key schedule is an invertible function.
 - Given one or more round keys, the master keys can be calculated.


Physical attacks

- By Kerckhoffs' principle
 - round functions and key schedule specifications are public
 - master key (hence also the round keys) are secret
- We will only consider ciphers whose key schedules are invertible functions
- In physical attacks that we will discuss in the later parts of the book, the attacker normally aims to recover some round key(s) and then use the inverse key schedule to find the master key.

Encryption of a block cipher

- F : round function
- N_r : total number of rounds
- K_i : round key for round i
- S_i : the cipher state at the end of round i
- For a plaintext $p \in \mathbb{F}_2^n$, the ciphertext $c \in \mathbb{F}_2^n$ can be computed as follows¹

$$\begin{aligned}S_0 &= p, \\S_1 &= F(S_0, K_1), \\S_2 &= F(S_1, K_2), \\&\vdots \\S_{N_r} &= F(S_{N_r-1}, K_{N_r}), \\c &= S_{N_r}.\end{aligned}$$

¹The round function for the last round might be a bit different, as for the case of AES 

Decryption of a block cipher

- To perform decryption, we require that for any given round key K_i , $F(\mathbf{x}, K_i)$ has an inverse, i.e.

$$F^{-1}(F(\mathbf{x}, K_i), K_i) = \mathbf{x}, \quad \forall \mathbf{x} \in \mathbb{F}_2^n.$$

- In this case, given ciphertext c , plaintext p can be computed as follows:

$$\begin{aligned} S_{N_r} &= c, \\ S_{N_r-1} &= F^{-1}(S_{N_r}, K_{N_r}), \\ &\vdots \\ S_1 &= F^{-1}(S_2, K_2), \\ S_0 &= F^{-1}(S_1, K_1), \\ p &= S_0. \end{aligned}$$

XOR with the round key

- We recall for a vector space over \mathbb{F}_2 , vector addition is given by bitwise XOR, denoted \oplus
- XOR with the round key is a common operation in round functions of symmetric block ciphers.

Example

Take $111, 101 \in \mathbb{F}_2^3$,

$$111 \oplus 101 = 010$$

Sbox

- Another common function is a substitution function called *Sbox*, denoted SB
- SB: $\mathbb{F}_2^{\omega_1} \rightarrow \mathbb{F}_2^{\omega_2}$.
- Normally ω_1 or/and ω_2 is a divisor of the block length n and a few Sboxes are applied in one round function.
- When $\omega_1 = \omega_2$, SB is a permutation on $\mathbb{F}_2^{\omega_1}$ and we say that the Sbox is an ω_1 -*bit Sbox*.

Feistel cipher and SPN cipher

- There are mainly two types of symmetric block ciphers – *Feistel cipher* and *Substitution–permutation network (SPN) cipher*.
- Feistel: DES
- SPN: AES, PRESENT

Symmetric block ciphers and their implementations

- Introduction
- DES
- AES
- PRESENT
- Bitsliced implementations
- Implementation of round functions

Feistel cipher

- The cipher state at the beginning/end of each round is divided into two halves of equal length.
- The cipher state at the end of round i is denoted as L_i and R_i , L – left, R – right.
- The round function F is defined as

$$(L_i, R_i) = F(L_{i-1}, R_{i-1}, K_i), \text{ where } L_i = R_{i-1}, R_i = L_{i-1} \oplus f(R_{i-1}, K_i).$$

- We note that f is a function that does not need to have an inverse since the function F is always invertible

$$L_{i-1} = R_i \oplus f(L_i, K_i), \quad R_{i-1} = L_i.$$

- The ciphertext is normally given by $R_{Nr} || L_{Nr}$ (i.e. swapping the left and right side of the cipher state at the end of the last round).
- In this case, if we let R_i and L_i denote the right and left part of the cipher state at the end of round i in the decryption, then the decryption computation is the same as for encryption, with round keys in reverse order

Feistel cipher

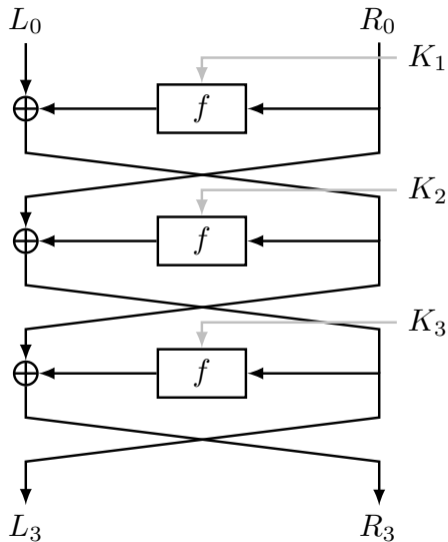


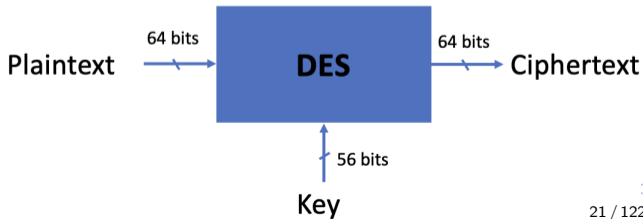
Figure: An illustration of Feistel cipher encryption algorithm.

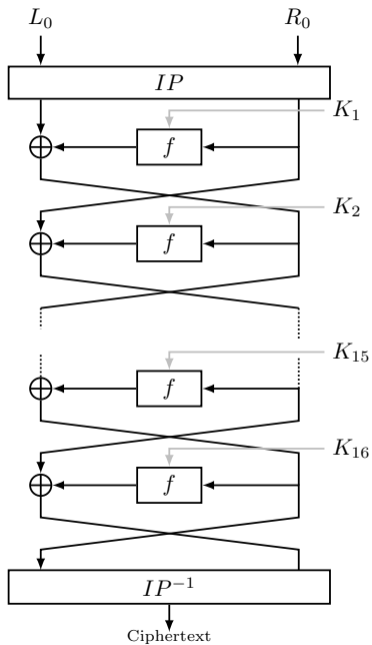
Data Encryption Standard (DES)

- Developed at IBM by a team led by Horst Feistel and the design was based on Lucifer cipher.
- It was used as the NIST standard from 1977 to 2005.
- Even though the key length is too small for the current standard, some variants are still in use today (invited talk)
- It has a significant influence on the development of cipher design.

DES

- The block length of DES is $n = 64$, i.e. $\mathcal{P} = \mathcal{C} = \mathbb{F}_2^{64}$.
- $L_i, R_i \in \mathbb{F}_2^{32}$.
- The master key length is 56, i.e. $\mathcal{K} = \mathbb{F}_2^{56}$.
- The round key length is 48.
- The total number of rounds $N_r = 16$.





DES – Initial permutation

- Before the first round function, the encryption starts with an *initial permutation (IP)*.
- The inverse of IP, called the *final permutation (IP^{-1})* is applied to the cipher state after the last round before outputting the ciphertext.
- Initial and final permutations are included for the ease of loading plaintext/ciphertext.

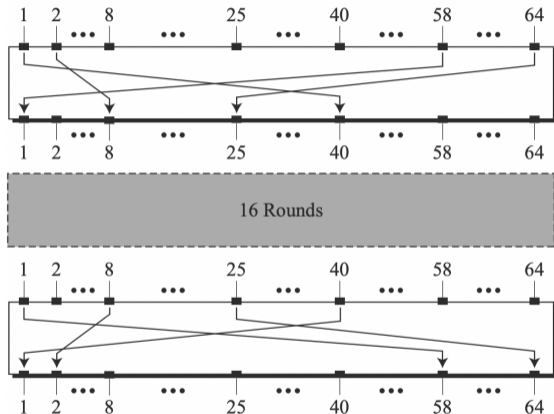
Remark

For DES specification, we consider the 1st bit of a value as the leftmost bit in its binary representation. For example, the 1st bit of $3 = 011_2$ is 0, the 2nd bit is 1 and the last bit is 1.

DES – initial permutation

58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

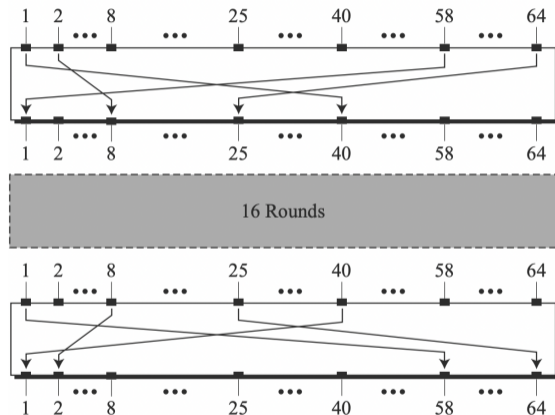
- 1st bit of output is from 58th bit of input
- 2nd bit of output is from 50th bit of input



DES – final permutation

40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

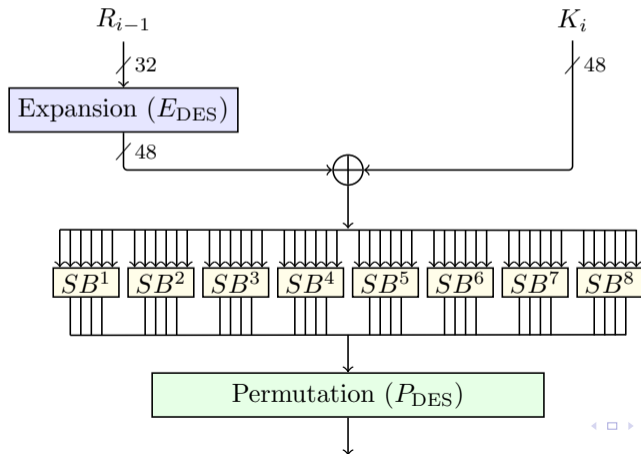
- 1st bit of output is from 40th bit of input
- 2nd bit of output is from 8th bit of input



DES – function f

- At the i th round, the function f in the round function of DES takes input $R_{i-1} \in \mathbb{F}_2^{32}$ and round key $K_i \in \mathbb{F}_2^{48}$, then outputs a 32-bit intermediate value:

$$f(R_{i-1}, K_i) = P_{\text{DES}}(\text{Sboxes}(E_{\text{DES}}(R_{i-1}) \oplus K_i)).$$



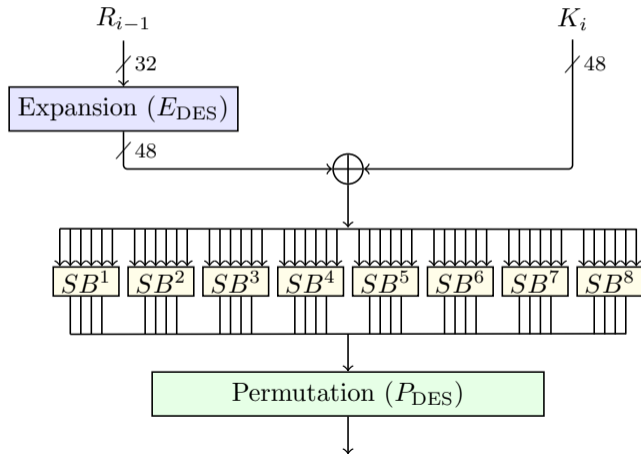
DES – function f – expansion function

- Expansion function: $\mathbb{F}_2^{32} \rightarrow \mathbb{F}_2^{48}$
- 16 bits of the input are repeated and affect two bits of the output, which influence two Sboxes.
- Such a design makes the dependency of the output bits on the input bits spread faster and achieves higher diffusion.
- Recall: construction principle – diffusion obscures the statistical relationship between the plaintext and the ciphertext. Each change in the plaintext is spread over the ciphertext, with the redundancies being dissipated.

32	1	2	3	4	5	4	5
6	7	8	9	8	9	10	11
12	13	12	13	14	15	16	17
16	17	18	19	20	21	20	21
22	23	24	25	24	25	26	27
28	29	28	29	30	31	32	1

DES – function f – Sboxes

- 48-bit value is divided into eight 6-bit subblocks
- 8 distinct Sboxes, $SB_{DES}^j : \mathbb{F}_2^6 \rightarrow \mathbb{F}_2^4$ ($1 \leq j \leq 8$), are applied to each of the 6 bits.



DES – function f – Sboxes

- 6 bit input: $b_1b_2b_3b_4b_5b_6$
 - b_1b_6 , row number
 - $b_2b_3b_4b_5$, column number
- Each row is a permutation of integers $0, 1, \dots, 15$

SB ₁															
14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

Example

- $b_1b_2b_3b_4b_5b_6 = 100110$
- $SB_{DES}^1(100110) = ?$

DES – function f – Sboxes

- 6 bit input: $b_1b_2b_3b_4b_5b_6$
 - b_1b_6 , row number
 - $b_2b_3b_4b_5$, column number
- Each row is a permutation of integers $0, 1, \dots, 15$

SB ₁															
14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

Example

- $b_1b_2b_3b_4b_5b_6 = 100110$
- row number is given by $b_1b_6 = 2$, column number is given by $b_2b_3b_4b_5 = 0011 = 3$
- $SB_{DES}^1(100110) = 8 = 1000$

DES – function f – Sboxes

SB ₂															
15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9

SB ₃															
10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12

Example

- $b_1b_2b_3b_4b_5b_6 = 100110$, $b_1b_6 = 2$, $b_2b_3b_4b_5 = 0011 = 3$
- $SB_{DES}^3(100110) = ?$

DES – function f – Sboxes

SB ₂															
15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9

SB ₃															
10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12

Example

- $b_1b_2b_3b_4b_5b_6 = 100110$, $b_1b_6 = 2$, $b_2b_3b_4b_5 = 0011 = 3$
- $SB_{DES}^3(100110) = 9 = 1001$

DES – function f – permutation

- Permutation: $\mathbb{F}_2^{32} \rightarrow \mathbb{F}_2^{32}$
- 1st bit of output comes from the 16th bit of input

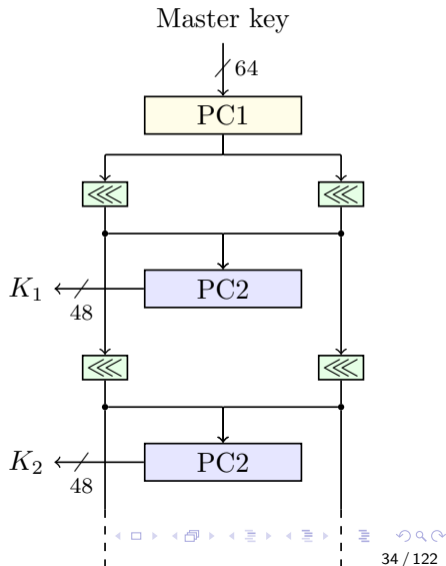
16	7	20	21
29	12	28	17
1	15	23	26
5	18	31	10
2	8	24	14
32	27	3	9
19	13	30	6
22	11	4	25

DES – key schedule

- Input: 64-bit master key. Outputs round keys of length 48.
- PC: *permuted choice*
- Master key: each 8th bit is a parity-check bit – XOR of the previous 7 bits
- PC1: 64 bits are reduced to 56 bits
- 56 bits are divided into two 28-bit halves
- Each half rotates left by one or two bits, depending on the round
- PC2: selects 48 bits out of 56 bits, permutes them

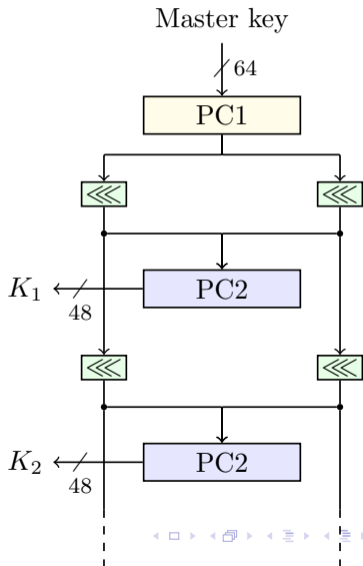
Remark

Knowledge of any round key \rightarrow 48 bits of the master key



DES – key schedule – PC1

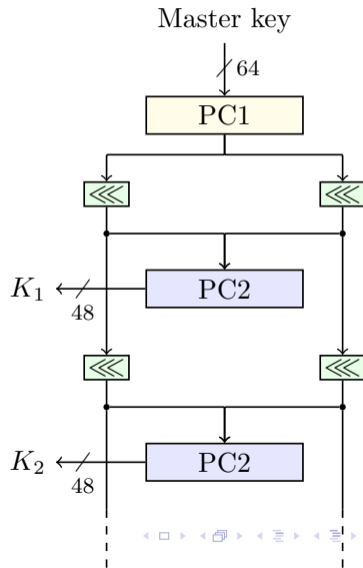
Left						
57	49	41	33	25	17	9
1	58	50	42	34	26	18
10	2	59	51	43	35	27
19	11	3	60	52	44	36
Right						
63	55	47	39	31	23	15
7	62	54	46	38	30	22
14	6	61	53	45	37	29
21	13	5	28	20	12	4



DES – key schedule – left rotate

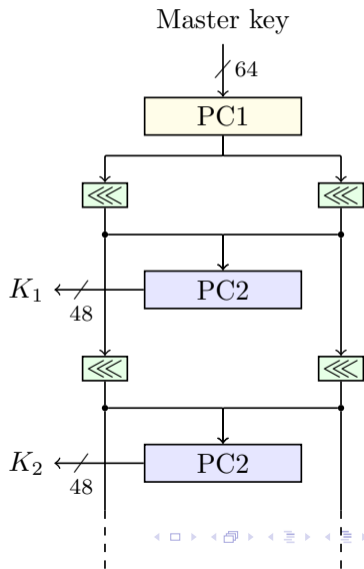
1	2	3	4	5	6	7	8
1	1	2	2	2	2	2	2
9	10	11	12	13	14	15	16
1	2	2	2	2	2	2	1

Number of key bits rotated per round



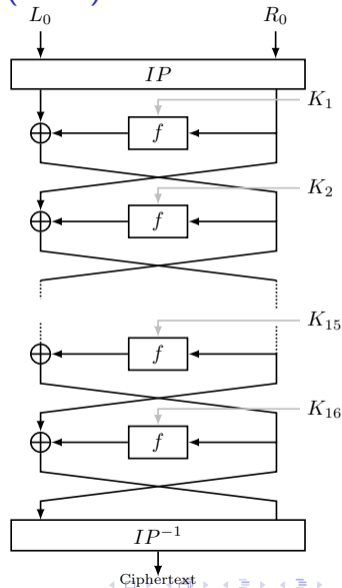
DES – key schedule – PC2

14	17	11	24	1	5	3	28
15	6	21	10	23	19	12	4
26	8	16	7	27	20	13	2
41	52	31	37	47	55	30	40
51	45	33	48	44	49	39	56
34	53	46	42	50	36	29	32



Data Encryption Standard (DES)

- Encryption
- Same algorithm for decryption
 - Round keys in reverse order
- Weak keys: master keys that give the same round keys in more than one round
 - 01010101 01010101
 - FEFEFEFE FEFEFEFE
 - E0E0E0E0 F1F1F1F1
 - 1F1F1F1F 0E0E0E0E

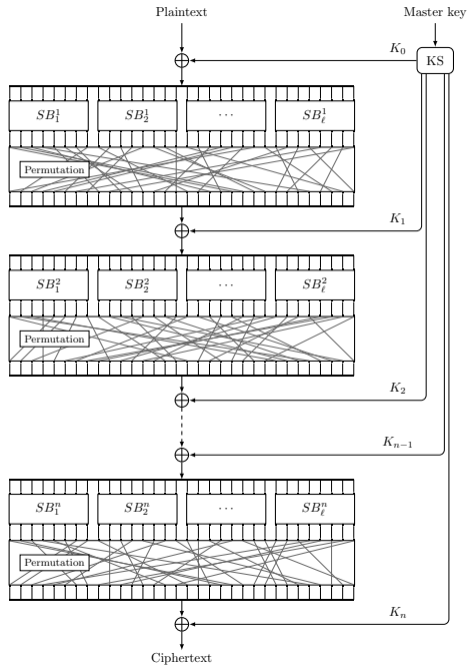


Symmetric block ciphers and their implementations

- Introduction
- DES
- **AES**
- PRESENT
- Bitsliced implementations
- Implementation of round functions

SPN cipher

- Let ω be a divisor of n , the block length, and let $\ell = n/\omega$.
 - In most cases, $\omega = 4, 8$.
- Each round of an SPN cipher normally consists of
 - bitwise XOR with the round key
 - application of ℓ parallel ω -bit Sboxes
 - a permutation on \mathbb{F}_2^n
- The encryption starts with XOR with a round key, also ends with XOR with a round key before outputting the ciphertext, otherwise, the cipher states in the second (or the last) round are all known to the attacker.
 - Those two operations are called *whitening*.
- For decryption, the inverse of Sbox and permutation are computed, and round keys are XOR-ed with the cipher state in reverse order compared to that for encryption.



AES

- NIST, 1997, Call for algorithms, replacement for DES
- Advanced Encryption Standard
- October 2000, Rijndael was selected
- Invented by Belgian cryptographers Joan Daemen and Vincent Rijmen
- Optimized for software efficiency on 8 and 32 bit processors

AES and Rijndael

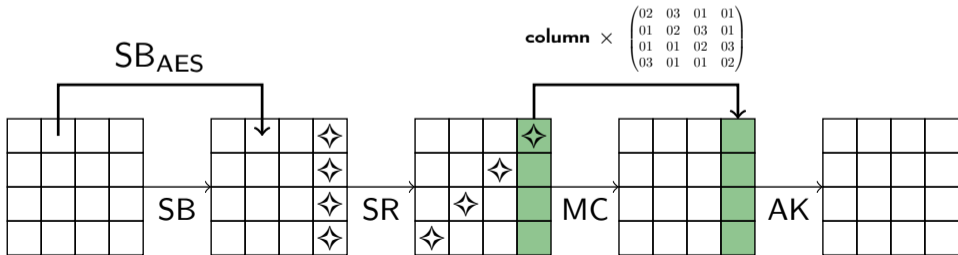
- $n = 128, \omega = 8$
- Number of rounds $N_r = 10, 12, 14$
- Corresponding key length = 128, 192, 256
- The original design of Rijndael also allows for different key lengths and block lengths.

key length	block length				
	128	160	192	224	256
128	10	11	12	13	14
160	11	11	12	13	14
192	12	12	12	13	14
224	13	13	13	13	14
256	14	14	14	14	14

Table: Specifications of Rijndael design, where blue-colored values are adopted by AES.

AES encryption

- An initial AddRoundKey
- Round function for $N_r - 1$ rounds: SubBytes, ShiftRows, MixColumns, AddRoundKey
- Last round, round N_r : SubBytes, ShiftRows, AddRoundKey
- AddRoundKey is bitwise XOR with the round key
- SubBytes is the application of 8-bit Sboxes.
- ShiftRows permutes the bytes
- MixColumns is a function on 32-bit values (four bytes).



AES – decryption

- The inverse of SubBytes, ShiftRows, and MixColumns are denoted as InvSubBytes, InvShiftRows, and InvMixColumns respectively.
- The first round of AES decryption computes AddRoundKey, InvShiftRows, and InvSubBytes.
- Then the round function for the next $N_r - 1$ rounds consists of AddRoundKey, InvMixColumns, InvShiftRows, and InvSubBytes.
- Finally, an additional AddRoundKey is computed
- Round keys for decryption are in reverse order as those for encryption.

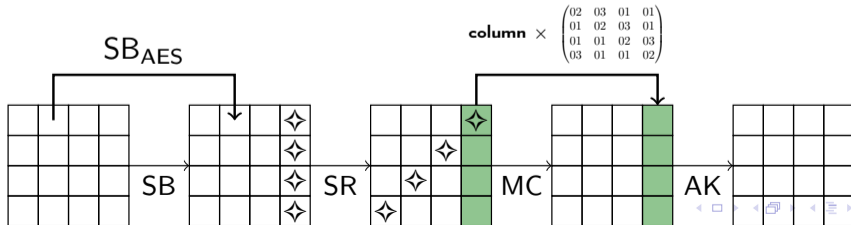
AES – state

- base-16 representation, base-10 representation, base-2 representation

$0_{16} = 0000_2 = 0_{10}, \dots, 9_{16} = 1001 = 9_{10}, A_{16} = 1010_2 = 10_{10}, F_{16} = 1111_2 = 15_{10}$

- One byte is a vector in \mathbb{F}_2^8 and can be represented as a hexadecimal number between 00 and FF e.g. $57_{16} = 01010111_2$
- Cipher state - four by four matrix of bytes

$$\begin{pmatrix} s_{00} & s_{01} & s_{02} & s_{03} \\ s_{10} & s_{11} & s_{12} & s_{13} \\ s_{20} & s_{21} & s_{22} & s_{23} \\ s_{30} & s_{31} & s_{32} & s_{33} \end{pmatrix}$$



AES – ShiftRows

- As the name suggests, the ShiftRows operation shifts the bytes in the rows of the cipher state.

$$\begin{pmatrix} s_{00} & s_{01} & s_{02} & s_{03} \\ s_{10} & s_{11} & s_{12} & s_{13} \\ s_{20} & s_{21} & s_{22} & s_{23} \\ s_{30} & s_{31} & s_{32} & s_{33} \end{pmatrix} \rightarrow \begin{pmatrix} s_{00} & s_{01} & s_{02} & s_{03} \\ s_{11} & s_{12} & s_{13} & s_{10} \\ s_{22} & s_{23} & s_{20} & s_{21} \\ s_{33} & s_{30} & s_{31} & s_{32} \end{pmatrix}.$$

- In another representation, let the output of ShiftRows be a matrix B with entries b_{ij} , then

$$\begin{pmatrix} b_{0j} \\ b_{1j} \\ b_{2j} \\ b_{3j} \end{pmatrix} = \begin{pmatrix} s_{0j} \\ s_{1(j+1 \bmod 4)} \\ s_{2(j+2 \bmod 4)} \\ s_{3(j+3 \bmod 4)} \end{pmatrix}, \quad 0 \leq j < 4.$$

- For decryption, the inverse of ShiftRows, InvShiftRows, can be easily deduced.

AES – SubBytes

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

$$SB_{AES}(12) = C9$$

AES – Sbox

- Different from the eight Sboxes in DES, AES Sbox can also be defined algebraically
- Let us first recall what we have learned about bytes and polynomial rings from the first week
- Will also be useful for understanding MixColumns

Polynomials

- $(F, +, \cdot)$: a field,

$$F[x] := \left\{ \sum_{i=0}^n a_i x^i \mid a_i \in F, n \geq 0 \right\}.$$

- A polynomial $f(x) \in F[x]$ of positive degree is said to be *reducible (over F)* if there exist $g(x), h(x) \in F[x]$ such that

$$\deg(g(x)) < \deg(f(x)), \deg(h(x)) < \deg(f(x)), \text{ and } f(x) = g(x)h(x).$$

Congruence classes modulo $f(x)$

- For any $g(x), h(x) \in F[x]$, if $f(x) \mid (g(x) - h(x))$, we say $h(x)$ is congruent to $g(x)$ modulo $f(x)$, written $g(x) \equiv h(x) \pmod{f(x)}$.
- Congruence class of $g(x)$ modulo $f(x)$ is given by

$$\{ h(x) \mid h(x) \equiv g(x) \pmod{f(x)} \}.$$

- Suppose $f(x)$ has degree n , where $n \geq 1$. Let $F[x]/(f(x))$ denote the set of all congruence classes of $g(x) \in F[x]$ modulo $f(x)$. Then

$$F[x]/(f(x)) = \left\{ \sum_{i=0}^{n-1} a_i x^i \mid a_i \in F \text{ for } 0 \leq i < n \right\}.$$

- For any $g(x), h(x) \in F[x]/(f(x))$, same as in for \mathbb{Z}_n , addition and multiplication in $F[x]/(f(x))$ are computed modulo $f(x)$.

\mathbb{F}_{2^8}

- $f(x) = x^8 + x^4 + x^3 + x + 1 \in \mathbb{F}_2[x]$.
- It can be shown that $f(x)$ is irreducible over \mathbb{F}_2
- Then the set of congruence classes of $g(x) \in \mathbb{F}_2[x]$ modulo $f(x)$ is

$$\mathbb{F}_2[x]/(f(x)) = \left\{ \sum_{i=0}^7 b_i x^i \mid b_i \in \mathbb{F}_2 \forall i \right\},$$

- We have also learned that

$$\mathbb{F}_2[x]/(f(x)) \cong \mathbb{F}_{2^8}.$$

Bytes

- Any

$$b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x + b_0 \in \mathbb{F}_2[x]/(f(x))$$

can be stored as a byte $b_7b_6b_5b_4b_3b_2b_1b_0 \in \mathbb{F}_2^8$

- A byte represents an integer between 0 (00_{16}) and 255 (FF_{16})
- There are 256 different values for a byte, and $|\mathbb{F}_{2^8}| = 2^8 = 256$.
- Define φ :

$$\begin{aligned} \varphi : \mathbb{F}_2[x]/(f(x)) &\rightarrow \mathbb{F}_2^8 \\ b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x + b_0 &\mapsto b_7b_6b_5b_4b_3b_2b_1b_0 \end{aligned}$$

- φ is bijective

Example

- $x^6 + x^4 + x^2 + x + 1 \in \mathbb{F}_2[x]/(f(x))$ corresponds to $01010111_2 = 57_{16}$
- $x^7 + x + 1 \in \mathbb{F}_2[x]/(f(x))$ corresponds to $10000011_2 = 83_{16}$.

Addition and multiplication between bytes

With addition and multiplication modulo $f(x)$ in $\mathbb{F}_2[x]/(f(x))$, we can define the corresponding addition and multiplication between bytes.

Definition

For any two bytes $\mathbf{v} = v_7v_6 \dots v_1v_0$ and $\mathbf{w} = w_7w_6 \dots w_1w_0$, let $g_{\mathbf{v}}(x) = v_7x^7 + v_6x^6 + \dots + v_1x + v_0$ and $g_{\mathbf{w}}(x) = w_7x^7 + w_6x^6 + \dots + w_1x + w_0$ be the corresponding polynomials in $\mathbb{F}_2[x]/(f(x))$. We define

$$\mathbf{v} + \mathbf{w} = g_{\mathbf{v}}(x) + g_{\mathbf{w}}(x) \bmod f(x), \quad \mathbf{v} \times \mathbf{w} = g_{\mathbf{v}}(x)g_{\mathbf{w}}(x) \bmod f(x).$$

Example

$f(x) = x^8 + x^4 + x^3 + x + 1$. Compute the sum and product between

$$x^6 + x^4 + x^2 + x + 1 \in \mathbb{F}_2[x]/(f(x)) \quad \text{i.e.} \quad 01010111_2 = 57_{16}$$

and

$$x^7 + x + 1 \in \mathbb{F}_2[x]/(f(x)) \quad \text{i.e.} \quad 10000011_2 = 83_{16}$$

Addition and multiplication between bytes

Example

$$f(x) = x^8 + x^4 + x^3 + x + 1.$$

$$\begin{aligned} 57_{16} + 83_{16} &= (x^6 + x^4 + x^2 + x + 1) + (x^7 + x + 1) \bmod f(x) \\ &= x^7 + x^6 + x^4 + x^2 \bmod f(x) = 11010100_2 = D4_{16}. \end{aligned}$$

$$(x^6 + x^4 + x^2 + x + 1)(x^7 + x + 1) = x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1,$$

$$x^8 = x^4 + x^3 + x + 1 \bmod f(x)$$

$$x^9 = x^5 + x^4 + x^2 + x \bmod f(x)$$

$$x^{11} = x^7 + x^6 + x^4 + x^3 \bmod f(x)$$

$$x^{13} = x^9 + x^8 + x^6 + x^5 \bmod f(x).$$

$$x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1 = x^{11} + x^4 + x^3 + 1 = x^7 + x^6 + 1 \bmod f(x).$$

$$\text{And } 57_{16} \times 83_{16} = 11000001_2 = C1_{16}.$$

Addition between bytes

For any

$$g(x) = \sum_{i=0}^{n-1} a_i x^i, \quad h(x) = \sum_{i=0}^{n-1} b_i x^i$$

from $\mathbb{F}_2[x]/(f(x))$, we have

$$g(x) + h(x) \bmod f(x) = \sum_{i=0}^{n-1} c_i x^i, \quad \text{where } c_i = a_i + b_i \bmod 2.$$

Recall that a byte is also a vector in \mathbb{F}_2^8 , we have defined vector addition as bitwise XOR, and

$$\mathbf{v} +_{\mathbb{F}_2^8} \mathbf{w} = \mathbf{u} = u_7 u_6 \dots u_1 u_0, \quad \text{where } u_i = v_i \oplus w_i.$$

We note that $a + b \bmod 2 = a \oplus b$ for $a, b \in \mathbb{F}_2$. Thus, our definition of addition between two bytes agrees with the vector addition between two vectors in \mathbb{F}_2^8 .

Multiplication by 02

$$f(x) = x^8 + x^4 + x^3 + x + 1.$$

We will compute the formula for a byte multiplied by $02_{16} = x$. Take any $g(x) = b_7x^7 + b_6x^6 + \dots + b_1x + b_0 \in \mathbb{F}_2[x]/(f(x))$

$$\begin{aligned} & g(x)x \bmod f(x) \\ = & (b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x + b_0)x \bmod f(x) \\ = & b_7x^8 + b_6x^7 + b_5x^6 + b_4x^5 + b_3x^4 + b_2x^3 + b_1x^2 + b_0x \bmod f(x) \\ = & b_6x^7 + b_5x^6 + b_4x^5 + b_3x^4 + b_2x^3 + b_1x^2 + b_0x + b_7x^4 + b_7x^3 + b_7x + b_7 \bmod f(x) \\ = & b_6x^7 + b_5x^6 + b_4x^5 + (b_3 + b_7)x^4 + (b_2 + b_7)x^3 + b_1x^2 + (b_0 + b_7)x + b_7 \bmod f(x). \end{aligned}$$

Thus, for any byte $b_7b_6 \dots b_1b_0$, multiplication by 02_{16} is equivalent to left shift by 1 and XOR with $00011011_2 = 1B_{16}$ if $b_7 = 1$.

Multiplication by 02

For any byte $b_7b_6 \dots b_1b_0$, multiplication by 02_{16} is equivalent to left shift by 1 and XOR with $00011011_2 = 1B_{16}$ if $b_7 = 1$.

Example

- $57_{16} = 01010111_2$, $02_{16} \times 57_{16} = 10101110 = AE_{16}$.
- $83_{16} = 10000011_2$, $02_{16} \times 83_{16} = 00000110_2 \oplus 00011011_2 = 00011101_2 = 1D_{16}$.
- $D4_{16} = 11010100_2$, $02_{16} \times D4_{16} = 10101000_2 \oplus 00011011_2 = 10110011_2 = B3_{16}$.

Multiplication by 03

Let us compute the multiplication of a byte by $03_{16} = x + 1$. Take any $h(x) = b_7x^7 + b_6x^6 + \dots + b_1x + b_0 \in \mathbb{F}_2[x]/(f(x))$, then

$$h(x)(x + 1) \bmod f(x) = h(x)x + h(x) \bmod f(x).$$

Thus, for any byte $b_7b_6 \dots b_1b_0$, multiplication by 03_{16} is equivalent to first multiply by 02_{16} (left shift by 1 and XOR with $00011011_2 = 1B_{16}$ if $b_7 = 1$) then XOR with the byte itself ($b_7b_6 \dots b_1b_0$).

Example

We have computed

$$02_{16} \times 57_{16} = AE_{16}, \quad 02_{16} \times 83_{16} = 1D_{16}, \quad 02_{16} \times D4_{16} = B3_{16}.$$

We have

- $03_{16} \times 57_{16} = AE_{16} \oplus 57_{16} = F9_{16}$.
- $03_{16} \times 83_{16} = 1D_{16} \oplus 83_{16} = 9E_{16}$.
- $03_{16} \times D4_{16} = B3_{16} \oplus D4_{16} = 67_{16}$.

Inverse of a byte as an element in $\mathbb{F}_2[x]/(f(x))$.

$$f(x) = x^8 + x^4 + x^3 + x + 1.$$

As mentioned before, multiplicative inverse of $g(x) \in \mathbb{F}_2[x]/(f(x))$ can be found using the extended Euclidean algorithm

Example

$03_{16} = 00000011_2 = x + 1$. By the Euclidean algorithm,

$$f(x) = (x + 1)(x^7 + x^6 + x^5 + x^4 + x^2 + x) + 1 \implies \gcd(f(x), (x + 1)) = 1.$$

Long division

In primary school, we learned to do long division for calculating the quotient and remainder of dividing one integer by another integer. For example, to compute

$$1346 = 25 \times q + r,$$

we can write

$$\begin{array}{r} 53 \\ 25 \overline{)1346} \\ \underline{125} \\ 96 \\ \underline{75} \\ 21 \end{array}$$

and we get $q = 53$, $r = 21$.

Similarly, let us take two polynomials $f(x), g(x) \in F[x]$, where F is a field. We can also compute $f(x)$ divided by $g(x)$ using long division.

Long division

$$\begin{array}{r} x^7 + x^6 + x^5 + x^4 + x^2 + x + 1 \\ x + 1 \overline{) x^8 + x^4 + x^3 + x + 1} \\ \underline{x^8 + x^7} \\ x^7 + x^4 + x^3 + x + 1 \\ \underline{x^7 + x^6} \\ x^6 + x^4 + x^3 + x + 1 \\ \underline{x^6 + x^5} \\ x^5 + x^4 + x^3 + x + 1 \\ \underline{x^5 + x^4} \\ x^3 + x + 1 \\ \underline{x^3 + x^2} \\ x^2 + x + 1 \\ \underline{x^2 + x} \\ 1 \end{array}$$

$$f(x) = (x+1)(x^7+x^6+x^5+x^4+x^2+x+1)+1.$$

Inverse of a byte as an element in $\mathbb{F}_2[x]/(f(x))$

$$f(x) = x^8 + x^4 + x^3 + x + 1.$$

Multiplicative inverse of $g(x) \in \mathbb{F}_2[x]/(f(x))$ can be found using the extended Euclidean algorithm

Example

$03_{16} = 00000011_2 = x + 1$. By the Euclidean algorithm,

$$f(x) = (x + 1)(x^7 + x^6 + x^5 + x^4 + x^2 + x) + 1 \implies \gcd(f(x), (x + 1)) = 1.$$

By the extended Euclidean algorithm,

$$1 = f(x) - (x + 1)(x^7 + x^6 + x^5 + x^4 + x^2 + x).$$

We have

$$03_{16}^{-1} = (x + 1)^{-1} \bmod f(x) = x^7 + x^6 + x^5 + x^4 + x^2 + x = 11110110_2 = \mathbf{F6}_{16}.$$

Inverse of a byte as an element in $\mathbb{F}_2[x]/(f(x))$

$$f(x) = x^8 + x^4 + x^3 + x + 1.$$

Example

Find inverse of $5B_{16} = 01011011_2$ as an element in $\mathbb{F}_2[x]/(f(x))$.

Inverse of a byte as an element in $\mathbb{F}_2[x]/(f(x))$

$$f(x) = x^8 + x^4 + x^3 + x + 1.$$

Example

$$5B_{16} = 01011011_2 \rightarrow x^6 + x^4 + x^3 + x + 1 \in \mathbb{F}_2[x]/(f(x)).$$

$$f(x) = (x^2 + 1)(x^6 + x^4 + x^3 + x + 1) + (x^5 + x^3 + x^2),$$

$$x^6 + x^4 + x^3 + x + 1 = x(x^5 + x^3 + x^2) + (x + 1),$$

$$x^5 + x^3 + x^2 = (x^4 + x^3 + x + 1)(x + 1) + 1,$$

$$\begin{aligned} 1 &= (x^5 + x^3 + x^2) + (x^4 + x^3 + x + 1)(x + 1) \\ &= (x^5 + x^3 + x^2) + (x^4 + x^3 + x + 1)((x^6 + x^4 + x^3 + x + 1) + x(x^5 + x^3 + x^2)) \\ &= (x^4 + x^3 + x + 1)(x^6 + x^4 + x^3 + x + 1) + (x^5 + x^4 + x^2 + x + 1)(x^5 + x^3 + x^2) \\ &= (x^4 + x^3 + x + 1)(x^6 + x^4 + x^3 + x + 1) \\ &\quad + (x^5 + x^4 + x^2 + x + 1)(f(x) + (x^2 + 1)(x^6 + x^4 + x^3 + x + 1)) \\ &= (x^5 + x^4 + x^2 + x + 1)f(x) + (x^7 + x^6 + x^5 + x^4)(x^6 + x^4 + x^3 + x + 1). \end{aligned}$$

Inverse of a byte as an element in $\mathbb{F}_2[x]/(f(x))$

$$f(x) = x^8 + x^4 + x^3 + x + 1.$$

Example

$$5B_{16} = 01011011_2 \rightarrow x^6 + x^4 + x^3 + x + 1 \in \mathbb{F}_2[x]/(f(x)).$$

$$\begin{aligned} 1 &= (x^5 + x^3 + x^2) + (x^4 + x^3 + x + 1)(x + 1) \\ &= (x^5 + x^3 + x^2) + (x^4 + x^3 + x + 1)((x^6 + x^4 + x^3 + x + 1) + x(x^5 + x^3 + x^2)) \\ &= (x^4 + x^3 + x + 1)(x^6 + x^4 + x^3 + x + 1) + (x^5 + x^4 + x^2 + x + 1)(x^5 + x^3 + x^2) \\ &= (x^4 + x^3 + x + 1)(x^6 + x^4 + x^3 + x + 1) \\ &\quad + (x^5 + x^4 + x^2 + x + 1)(f(x) + (x^2 + 1)(x^6 + x^4 + x^3 + x + 1)) \\ &= (x^5 + x^4 + x^2 + x + 1)f(x) + (x^7 + x^6 + x^5 + x^4)(x^6 + x^4 + x^3 + x + 1). \end{aligned}$$

$$(x^6 + x^4 + x^3 + x + 1)^{-1} \bmod f(x) = x^7 + x^6 + x^5 + x^4 = 11110000_2 = \text{F0}$$

AES – SubBytes

Let

$$A = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}, \quad a = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix},$$

then

$$\text{SB}_{\text{AES}}(z) = \begin{cases} Az^{-1} + a & z \neq 0 \\ a & z = 0 \end{cases}$$

where z^{-1} is the inverse of z as an element in $\mathbb{F}_2[x]/(f(x))$

Example

What is $\text{SB}_{\text{AES}}(03)$?

AES – SubBytes

Example

We have seen that $03^{-1} = 11110110_2$. And we have

$$A \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix} + a = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 1 \\ 1 \end{pmatrix}.$$

So $SB_{AES}(03) = 01111011_2 = 7B$, which agrees with the AES Sbox table.

AES – InvSubBytes

- Recall SubBytes:

$$SB_{\text{AES}}(z) = \begin{cases} Az^{-1} + a & z \neq 0 \\ a & z = 0 \end{cases}$$

- Let g denote the function $g(z) = Az + a$.
- Then InvSubBytes computes

$$SB_{\text{AES}}^{-1}(z) = \begin{cases} (g^{-1}(z))^{-1} & g^{-1}(z) \neq 0 \\ 0 & g^{-1}(z) = 0 \end{cases}.$$

$$g^{-1}(z) = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \end{pmatrix} z + \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \end{pmatrix}$$

AES – InvSubBytes

InvSubBytes computes

$$SB_{\text{AES}}^{-1}(z) = \begin{cases} (g^{-1}(z))^{-1} & g^{-1}(z) \neq 0 \\ 0 & g^{-1}(z) = 0 \end{cases}.$$

$$g^{-1}(z) = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \end{pmatrix} z + \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \end{pmatrix}$$

Example

Let $z = 8C = 10001100_2$. What is $SB_{\text{AES}}^{-1}(8C)$?

AES – InvSubBytes

Example

Let $z = 8C = 10001100_2$. Then

$$g^{-1}(z) = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 1 \end{pmatrix} = 5B_{16}.$$

We have seen that

$$(x^6 + x^4 + x^3 + x + 1)^{-1} \bmod f(x) = x^7 + x^6 + x^5 + x^4 = 11110000_2 = F0,$$

which gives $SB_{AES}^{-1}(8C) = F0$

AES – InvSubBytes

InvSubBytes can also be described using a table

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	52	09	6A	D5	30	36	A5	38	BF	40	A3	9E	81	F3	D7	FB
1	7C	E3	39	82	9B	2F	FF	87	34	8E	43	44	C4	DE	E9	CB
2	54	7B	94	32	A6	C2	23	3D	EE	4C	95	0B	42	FA	C3	4E
3	08	2E	A1	66	28	D9	24	B2	76	5B	A2	49	6D	8B	D1	25
4	72	F8	F6	64	86	68	98	16	D4	A4	5C	CC	5D	65	B6	92
5	6C	70	48	50	FD	ED	B9	DA	5E	15	46	57	A7	8D	9D	84
6	90	D8	AB	00	8C	BC	D3	0A	F7	E4	58	05	B8	B3	45	06
7	D0	2C	1E	8F	CA	3F	0F	02	C1	AF	BD	03	01	13	8A	6B
8	3A	91	11	41	4F	67	DC	EA	97	F2	CF	CE	F0	B4	E6	73
9	96	AC	74	22	E7	AD	35	85	E2	F9	37	E8	1C	75	DF	6E
A	47	F1	1A	71	1D	29	C5	89	6F	B7	62	0E	AA	18	BE	1B
B	FC	56	3E	4B	C6	D2	79	20	9A	DB	C0	FE	78	CD	5A	F4
C	1F	DD	A8	33	88	07	C7	31	B1	12	10	59	27	80	EC	5F
D	60	51	7F	A9	19	B5	4A	0D	2D	E5	7A	9F	93	C9	9C	EF
E	A0	E0	3B	4D	AE	2A	F5	B0	C8	BE	BB	3C	83	53	99	61
F	17	2B	04	7E	BA	77	D6	26	E1	69	14	63	55	21	0C	7D

AES – MixColumns

- Recall AES state

$$\begin{pmatrix} s_{00} & s_{01} & s_{02} & s_{03} \\ s_{10} & s_{11} & s_{12} & s_{13} \\ s_{20} & s_{21} & s_{22} & s_{23} \\ s_{30} & s_{31} & s_{32} & s_{33} \end{pmatrix}$$

- The MixColumns function takes each of the four columns of the cipher state as input
- Multiplies the input column by a matrix:

$$\begin{pmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \begin{pmatrix} s_{0j} \\ s_{1j} \\ s_{2j} \\ s_{3j} \end{pmatrix}, \quad j = 0, 1, 2, 3. \quad (1)$$

AES – MixColumns

Example

Suppose

$$(s_{0j} \ s_{1j} \ s_{2j} \ s_{3j}) = (D4 \ BF \ 5D \ 30).$$

Then

$$\begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \begin{pmatrix} D4 \\ BF \\ 5D \\ 30 \end{pmatrix} = \begin{pmatrix} ? \\ 66 \\ 81 \\ E5 \end{pmatrix}$$

AES – MixColumns

Example

$$\begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \begin{pmatrix} D4 \\ BF \\ 5D \\ 30 \end{pmatrix} = \begin{pmatrix} 04 \\ 66 \\ 81 \\ E5 \end{pmatrix}$$

We have calculated that

$$02 \times D4 = 10110011_2.$$

We can compute

$$03_{16} \times BF_{16} = 01111110_2 \oplus 00011011_2 \oplus 10111111_2 = 11011010_2 = DA_{16}.$$

Then we have

$$B3 + DA + 5D + 30 = 10110011 \oplus 11011010 \oplus 01011101 \oplus 00110000 = 00000100 = 04.$$

AES – InvMixColumns

Multiply each column by the following matrix

$$\begin{pmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{pmatrix} \cdot$$

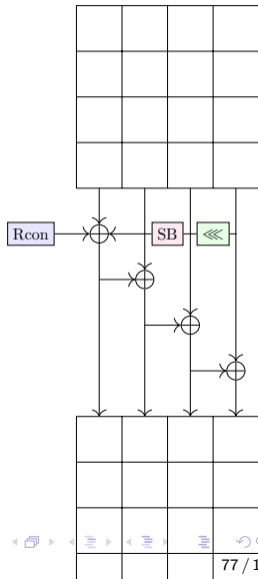
AES-128 – key schedule

- The round keys are represented as four-by-four grids and each box corresponds to one byte
- The rotation \ll rotates the right-most column by one byte

$$\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{pmatrix} \mapsto \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_0 \end{pmatrix}.$$

- Round constants:

$$\text{Rcon} = \{ 01, 02, 04, 08, 10, 20, 40, 80, 1B, 36 \}.$$



Remark

Knowledge of any round key of AES-128 \rightarrow master key

Symmetric block ciphers and their implementations

- Introduction
- DES
- AES
- **PRESENT**
- Bitsliced implementations
- Implementation of round functions

PRESENT

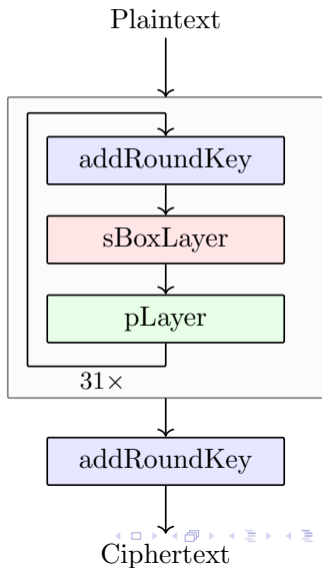
- Proposed in 2007 as a symmetric block cipher optimized for hardware implementation.
- Block length: $n = 64$
- Number of rounds: $N_r = 31$
- Key length: either 80 or 128.
- When the key length is 80, the algorithm is called PRESENT-80.

PRESENT – encryption

- Round function: addRoundKey, sBoxLayer, and pLayer.
- After 31 rounds, addRoundKey is applied again before the ciphertext output

Remark

As opposed to DES specification, for PRESENT specification, we consider the 0th bit of a value as the rightmost bit in its binary representation. For example, the 0th bit of $3 = 011_2$ is 1, the 1st bit is 1 and the 2nd bit is 0.



PRESENT – addRoundKey

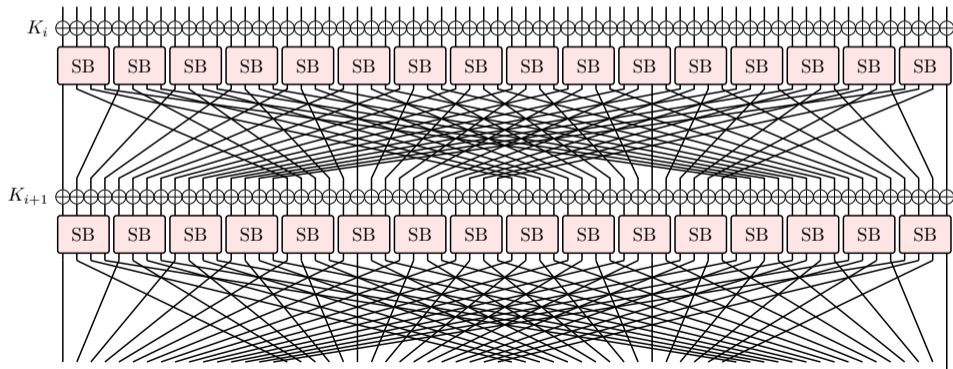
- Round key $K_i = \kappa_{63}^i \dots \kappa_0^i$, ($1 \leq i \leq 32$)
- Current state $b_{63}b_{62} \dots b_0$
- For $0 \leq j \leq 63$

$$b_j = b_j \oplus \kappa_j^i$$

PRESENT – sBoxLayer

- sBoxLayer applies sixteen 4-bit Sboxes to each nibble of the current cipher state.
- For example, if the input is 0, the output is C.

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
C	5	6	B	9	0	A	D	3	E	F	8	4	7	1	2



PRESENT – pLayer

pLayer permutes the 64 bits using the following formula:

$$\text{pLayer}(j) = \left\lfloor \frac{j}{4} \right\rfloor + (j \bmod 4) \times 16,$$

where j denotes the bit position.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
?	?	?	?	1	17	33	49	2	18	34	50	3	19	35	51
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
4	20	36	52	5	21	37	53	6	22	38	54	7	23	39	55
32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
8	24	40	56	9	25	41	57	10	26	42	58	11	27	43	59
48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
12	28	44	60	13	29	45	61	14	30	46	62	15	31	47	63

PRESENT – pLayer

pLayer permutes the 64 bits using the following formula:

$$\text{pLayer}(j) = \left\lfloor \frac{j}{4} \right\rfloor + (j \bmod 4) \times 16,$$

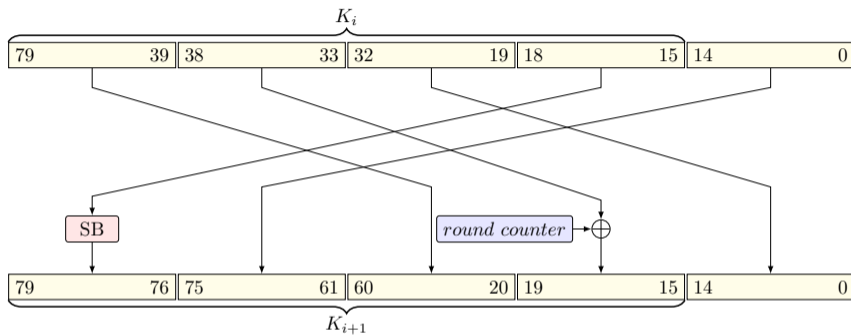
where j denotes the bit position.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	16	32	48	1	17	33	49	2	18	34	50	3	19	35	51
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
4	20	36	52	5	21	37	53	6	22	38	54	7	23	39	55
32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
8	24	40	56	9	25	41	57	10	26	42	58	11	27	43	59
48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
12	28	44	60	13	29	45	61	14	30	46	62	15	31	47	63

PRESENT-80 – key schedule

- $k_{79}k_{78} \dots k_0$: the variable storing the key
- At round i , the round key is given by $K_i = \kappa_{63}^i \kappa_{62}^i \dots \kappa_0^i = k_{79}k_{78} \dots k_{16}$.
- After extracting the round key, the variable $k_{79}k_{78} \dots k_0$ is updated using the following steps:
 - Left rotate of 61 bits, $k_{79}k_{78} \dots k_1k_0 = k_{18}k_{17} \dots k_{20}k_{19}$;
 - $k_{79}k_{78}k_{77}k_{76} = \text{SB}_{\text{PRESENT}}(k_{79}k_{78}k_{77}k_{76})$;
 - $k_{19}k_{18}k_{17}k_{16}k_{15} = k_{19}k_{18}k_{17}k_{16}k_{15} \oplus \text{round_counter}$;
- $\text{round_counter} = 1, 2, \dots, 31$

PRESENT-80 – key schedule



Remark

Knowledge of any round key for PRESENT-80 \rightarrow 64 bits of the master key. The other 16 bits can be recovered by brute force/knowledge of another round key

Other symmetric block ciphers

- GIFT
- PRINCE
- LED
- ...

Symmetric block ciphers and their implementations

- Introduction
- DES
- AES
- PRESENT
- Bitsliced implementations
- Implementation of round functions

Boolean function

Let n be a positive integer.

Definition

A *Boolean function* is a function $\varphi : \mathbb{F}_2^n \rightarrow \mathbb{F}_2$.

- A Boolean function has 2^n possible input values.
- For each input value, there are 2 possible output values.
- Thus, in total, we have 2^{2^n} possible Boolean functions defined for $\mathbb{F}_2^n \rightarrow \mathbb{F}_2$.
- A Boolean function can be specified by giving the output values for all inputs, such a table is called a *truth table*.

Example

The parity-check bit defined for 3 bits is a Boolean function

$$\begin{aligned}\varphi : \mathbb{F}_2^3 &\rightarrow \mathbb{F}_2 \\ x_2x_1x_0 &\mapsto x_0 + x_1 + x_2.\end{aligned}$$

What is the truth table for φ ?

Boolean function

Example

The parity-check bit defined for 3 bits is a Boolean function

$$\begin{aligned}\varphi : \mathbb{F}_2^3 &\rightarrow \mathbb{F}_2 \\ x_2x_1x_0 &\mapsto x_0 + x_1 + x_2.\end{aligned}$$

Its truth table is then given below:

x_2	0	0	0	0	1	1	1	1
x_1	0	0	1	1	0	0	1	1
x_0	0	1	0	1	0	1	0	1
$\varphi(\mathbf{x})$	0	1	1	0	1	0	0	1

Boolean function

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
C	5	6	B	9	0	A	D	3	E	F	8	4	7	1	2

Example

Now let us consider the Boolean function defined as follows:

$$\begin{aligned}\varphi_0 : \mathbb{F}_2^4 &\rightarrow \mathbb{F}_2 \\ \mathbf{x} &\mapsto \text{SB}_{\text{PRESENT}}(\mathbf{x})_0\end{aligned}$$

where $\text{SB}_{\text{PRESENT}}(\mathbf{x})_0$ is the 0th bit of $\text{SB}_{\text{PRESENT}}(\mathbf{x})$, the PRESENT Sbox output corresponding to \mathbf{x} . For example, if the input is 0, the Sbox output is C= 1100. Hence $\varphi_0(0) = 0$. What is the truth table for φ_0 ?

Boolean function

Example

$$\begin{aligned}\varphi_0 : \mathbb{F}_2^4 &\rightarrow \mathbb{F}_2 \\ \mathbf{x} &\mapsto \text{SB}_{\text{PRESENT}}(\mathbf{x})_0\end{aligned}$$

\mathbf{x}	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
x_3	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
x_2	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
x_1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
x_0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
$\text{SB}_{\text{PRESENT}}(\mathbf{x})$	C	5	6	B	9	0	A	D	3	E	F	8	4	7	1	2
$\varphi_0(\mathbf{x})$	0	1	0	1	1	0	0	1	1	0	1	0	0	1	1	0

Algebraic normal form

Theorem

Every Boolean function $\varphi : \mathbb{F}_2^n \rightarrow \mathbb{F}_2$ has a unique algebraic normal form representation

$$\varphi(\mathbf{x}) = \sum_{\mathbf{v} \in \mathbb{F}_2^n} \left(\lambda_{\mathbf{v}} \prod_{i=0}^{n-1} x_i^{v_i} \right).$$

The coefficients $\lambda_{\mathbf{v}} \in \mathbb{F}_2$ are given by

$$\lambda_{\mathbf{v}} = \sum_{\mathbf{w} \leq \mathbf{v}} \varphi(\mathbf{w}),$$

where $\mathbf{w} \leq \mathbf{v}$ means that $w_i \leq v_i$ for all $0 \leq i \leq n - 1$.

Remark

There are 2^{2^n} Boolean functions defined for $\mathbb{F}_2^n \rightarrow \mathbb{F}_2$. There are 2^{2^n} choices for the coefficients $\lambda_{\mathbf{v}}$ ($\lambda_{\mathbf{v}} = 0, 1$ and there are 2^n distinct \mathbf{v}).

Algebraic normal form

$$\varphi(\mathbf{x}) = \sum_{\mathbf{v} \in \mathbb{F}_2^n} \left(\lambda_{\mathbf{v}} \prod_{i=0}^{n-1} x_i^{v_i} \right),$$

$$\lambda_{\mathbf{v}} = \sum_{\mathbf{w} \leq \mathbf{v}} \varphi(\mathbf{w}),$$

where $\mathbf{w} \leq \mathbf{v}$ means that $w_i \leq v_i$ for all $0 \leq i \leq n-1$.

Example

The parity-check bit defined for 3 bits is a Boolean function. Its truth table is given by:

x_2	0	0	0	0	1	1	1	1
x_1	0	0	1	1	0	0	1	1
x_0	0	1	0	1	0	1	0	1
$\varphi(\mathbf{x})$	0	1	1	0	1	0	0	1

What is the algebraic normal form for φ ?

Algebraic normal form

Example

x_2		0	0	0	0	1	1	1	1
x_1		0	0	1	1	0	0	1	1
x_0		0	1	0	1	0	1	0	1
$\varphi(\mathbf{x})$		0	1	1	0	1	0	0	1

$$\lambda_{110} = \varphi(000) + \varphi(100) + \varphi(010) + \varphi(110) = 0 + 1 + 1 + 0 = 0.$$

$$\lambda_{000} = 0, \quad \lambda_{001} = 1, \quad \lambda_{010} = 1, \quad \lambda_{011} = 1 + 1 = 0,$$

$$\lambda_{100} = 1, \quad \lambda_{101} = 0, \quad \lambda_{110} = 0, \quad \lambda_{111} = 0.$$

$$\varphi(\mathbf{x}) = \sum_{\mathbf{v} \in \mathbb{F}_2^n} \left(\lambda_{\mathbf{v}} \prod_{i=0}^{n-1} x_i^{v_i} \right) = \lambda_{001}x_0 + \lambda_{010}x_1 + \lambda_{100}x_2 = x_0 + x_1 + x_2$$

Algebraic normal form

Example

\mathbf{x}	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
x_3	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
x_2	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
x_1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
x_0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
$\text{SB}_{\text{PRESENT}}(\mathbf{x})$	C	5	6	B	9	0	A	D	3	E	F	8	4	7	1	2
$\varphi_0(\mathbf{x})$	0	1	0	1	1	0	0	1	1	0	1	0	0	1	1	0
$\lambda_{\mathbf{x}}$	0	1	0	0	1	0	1	0	1	0	0	0	0	0	0	0

$$\begin{aligned} \varphi_0(\mathbf{x}) &= \sum_{\mathbf{v} \in \mathbb{F}_2^n} \left(\lambda_{\mathbf{v}} \prod_{i=0}^{n-1} x_i^{v_i} \right) = \lambda_{0001}x_0 + \lambda_{0100}x_2 + \lambda_{0110}x_1x_2 + \lambda_{1000}x_3 \\ &= x_0 + x_2 + x_1x_2 + x_3. \end{aligned}$$

Algebraic normal form

Example

- Define $\varphi_i(\mathbf{x}) = \text{SB}_{\text{PRESENT}}(\mathbf{x})_i$ for $i = 1, 2, 3$, where $\text{SB}_{\text{PRESENT}}(\mathbf{x})_i$ is the i th bit of PRESENT Sbox output for \mathbf{x} .
- We can calculate the algebraic normal form for each of φ_i in a similar way

$$\varphi_1(\mathbf{x}) = x_1 + x_3 + x_1x_3 + x_2x_3 + x_0x_1x_2 + x_0x_1x_3 + x_0x_2x_3,$$

$$\varphi_2(\mathbf{x}) = 1 + x_2 + x_3 + x_0x_1 + x_0x_3 + x_1x_3 + x_0x_1x_3 + x_0x_2x_3,$$

$$\varphi_3(\mathbf{x}) = 1 + x_0 + x_1 + x_3 + x_1x_2 + x_0x_1x_2 + x_0x_1x_3 + x_0x_2x_3.$$

Bitsliced implementation

- The goal of a bitsliced implementation is to simulate a hardware implementation in software so that several plaintext blocks can be encrypted in parallel.
- The operations in symmetric block ciphers will be represented as a sequence of logical operations.
- Naturally the implementations should be adjusted based on the specific underlying hardware, more specifically, the word length of the architecture.
- We will see that with word length ω , we can encrypt ω blocks of plaintext in parallel.

Bitsliced format

- First, we discuss how to transform the plaintext blocks into bitsliced format.
- A cipher with block length 3
- 4-bit architecture, which allows us to encrypt 4 blocks of plaintext simultaneously.
- Take 4 plaintext blocks $p_1 = 010, p_2 = 110, p_3 = 001, p_4 = 100$.
- The bitsliced format of p_j s is given by a 3×4 array, denoted S , where each column is given by one block of plaintext.

$$S = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix}.$$

- $S[x]$: x th row of S , then
 - $S[0]$ – 0th bits of p_j .
 - $S[1]$ – 1st bits of p_j .
 - $S[2]$ – 2nd bits of p_j .

Bitsliced implementation of PRESENT

- Encrypt 8 plaintext blocks in parallel with PRESENT assuming an 8-bit architecture.
- Let p_1, p_2, \dots, p_8 be eight plaintext blocks, each of length 64.
- We convert them into bitsliced format and store them in a 64×8 array S_0
 - $S_0[y]$ contains the y th bits of each plaintext block.
- For each round key K_i , we construct a 64×8 array Key_i whose columns are given by K_i , i.e.

$$\text{Key}_i[y][z] = K_i[y] \quad \forall 0 \leq z < 8.$$

Algebraic normal form

$$\begin{aligned}\varphi_0(\mathbf{x}) &= x_0 + x_2 + x_1x_2 + x_3 \\ &= x_0 \oplus x_2 \oplus (x_1 \& x_2) \oplus x_3\end{aligned}$$

$$\begin{aligned}\varphi_1(\mathbf{x}) &= x_1 + x_3 + x_1x_3 + x_2x_3 + x_0x_1x_2 + x_0x_1x_3 + x_0x_2x_3, \\ &= x_1 \oplus x_3 \oplus (x_1 \& x_3) \oplus (x_2 \& x_3) \oplus (x_0 \& x_1 \& x_2) \oplus (x_0 \& x_1 \& x_3) \oplus (x_0 \& x_2 \& x_3)\end{aligned}$$

$$\begin{aligned}\varphi_2(\mathbf{x}) &= 1 + x_2 + x_3 + x_0x_1 + x_0x_3 + x_1x_3 + x_0x_1x_3 + x_0x_2x_3, \\ &= 1 \oplus x_2 \oplus x_3 \oplus (x_0 \& x_1) \oplus (x_0 \& x_3) \oplus (x_1 \& x_3) \oplus (x_0 \& x_1 \& x_3) \oplus (x_0 \& x_2 \& x_3)\end{aligned}$$

$$\begin{aligned}\varphi_3(\mathbf{x}) &= 1 + x_0 + x_1 + x_3 + x_1x_2 + x_0x_1x_2 + x_0x_1x_3 + x_0x_2x_3 \\ &= 1 \oplus x_0 \oplus x_1 \oplus x_3 \oplus (x_1 \& x_2) \oplus (x_0 \& x_1 \& x_2) \oplus (x_0 \& x_1 \& x_3) \oplus (x_0 \& x_2 \& x_3)\end{aligned}$$

Algorithm 1: Bitsliced implementation of round i of PRESENT, $1 \leq i \leq 31$.

Input: S_{i-1} , $\text{Key}i$ // S_{i-1} is the output of round $i-1$. When $i=1$, S_0 contains the plaintext blocks in bitsliced format.

// $\text{Key}i$ is the i th round key K_i in bitsliced format

Output: S_i : output of round i

// addRoundKey-----

1 $S_{i-1} = S_{i-1} \oplus \text{Key}i$ // bitwise XOR

// sBoxLayer-----

2 **for** $b = 0, b < 16, b++$ **do**

 // Bits of Sbox inputs

3 $x_0 = S_{i-1}[4b], x_1 = S_{i-1}[4b + 1], x_2 = S_{i-1}[4b + 2], x_3 = S_{i-1}[4b + 3]$

 // 0th bit of Sbox output

4 $\text{state}[4b] = x_0 \oplus x_2 \oplus (x_1 \& x_2) \oplus x_3$

5 \dots

// pLayer-----

6 $S_i[0] = \text{state}[0]$

7 $S_i[16] = \text{state}[1]$

8 \dots

9 **return** S_i

Remarks

- It is easy to see that with 32-bit (resp. 64-bit) architecture we can encrypt 32 (resp. 64) plaintext blocks in parallel.
- We note that bitsliced implementations are mostly used for bit-oriented ciphers (e.g. DES, PRESENT).
- For byte-oriented ciphers (e.g. AES), table-based implementation will likely give better performance.

Symmetric block ciphers and their implementations

- Introduction
- DES
- AES
- PRESENT
- Bitsliced implementations
- Implementation of round functions

Building blocks

- Building blocks for a symmetric block cipher: bitwise XOR with round key, Sbox, and permutation.
- It is easy in both software and hardware to implement bitwise XOR with a round key.
 - In hardware, there is an XOR gate
 - almost every processor has a dedicated XOR instruction

Implementing Sboxes

- In software, a naïve way to implement Sbox is to use a lookup table.
- The table is stored as an array in random access memory or flash memory.
- The storage space required for an Sbox $SB: \mathbb{F}_2^{\omega_1} \rightarrow \mathbb{F}_2^{\omega_2}$ is at least $\omega_2 \times 2^{\omega_1}$.
- For example, PRESENT has a 4-bit Sbox and the storage required is at least $2^4 \times 4 = 64$ bits, or 8 bytes.
- Actual memory consumption depends on the architecture

AES – T-tables

- Implementation method that combines SubBytes, ShiftRows, and MixColumns for AES round function.
- SB: the AES Sbox.
- Let us denote the input of SubBytes by a matrix S

$$S = \begin{pmatrix} s_{00} & s_{01} & s_{02} & s_{03} \\ s_{10} & s_{11} & s_{12} & s_{13} \\ s_{20} & s_{21} & s_{22} & s_{23} \\ s_{30} & s_{31} & s_{32} & s_{33} \end{pmatrix}$$

- A, B, D : outputs of SubBytes, ShiftRows, MixColumns
- $a_{ij} = \text{SB}(s_{ij}), 0 \leq i, j < 4$.

AES – T-tables

- SB: AES Sbox.
- S : input of SubBytes
- A, B, D : outputs of SubBytes, ShiftRows, MixColumns
- $a_{ij} = \text{SB}(s_{ij}), 0 \leq i, j < 4$.
- For $j = 0, 1, 2, 3$

$$\begin{pmatrix} b_{0j} \\ b_{1j} \\ b_{2j} \\ b_{3j} \end{pmatrix} = \begin{pmatrix} a_{0j} \\ a_{1(j+1 \bmod 4)} \\ a_{2(j+2 \bmod 4)} \\ a_{3(j+3 \bmod 4)} \end{pmatrix}, \quad \begin{pmatrix} d_{0j} \\ d_{1j} \\ d_{2j} \\ d_{3j} \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \begin{pmatrix} b_{0j} \\ b_{1j} \\ b_{2j} \\ b_{3j} \end{pmatrix}$$

AES – T-tables

$$\begin{aligned}
 \begin{pmatrix} d_{0j} \\ d_{1j} \\ d_{2j} \\ d_{3j} \end{pmatrix} &= \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \begin{pmatrix} \text{SB}(s_{0j}) \\ \text{SB}(s_{1(j+1 \bmod 4)}) \\ \text{SB}(s_{2(j+2 \bmod 4)}) \\ \text{SB}(s_{3(j+3 \bmod 4)}) \end{pmatrix} \\
 &= \begin{pmatrix} 02 \\ 01 \\ 01 \\ 03 \end{pmatrix} \text{SB}(s_{0j}) \oplus \begin{pmatrix} 03 \\ 02 \\ 01 \\ 01 \end{pmatrix} \text{SB}(s_{1(j+1 \bmod 4)}) \\
 &\quad \oplus \begin{pmatrix} 01 \\ 03 \\ 02 \\ 01 \end{pmatrix} \text{SB}(s_{2(j+2 \bmod 4)}) \oplus \begin{pmatrix} 01 \\ 01 \\ 03 \\ 02 \end{pmatrix} \text{SB}(s_{3(j+3 \bmod 4)})
 \end{aligned}$$

AES – T-tables

For $a \in \mathbb{F}_2^8$, define

$$\begin{aligned} T_0(a) &:= \begin{pmatrix} 02 \\ 01 \\ 01 \\ 03 \end{pmatrix} \text{SB}(a), & T_1(a) &:= \begin{pmatrix} 03 \\ 02 \\ 01 \\ 01 \end{pmatrix} \text{SB}(a), \\ T_2(a) &:= \begin{pmatrix} 01 \\ 03 \\ 02 \\ 01 \end{pmatrix} \text{SB}(a), & T_3(a) &:= \begin{pmatrix} 01 \\ 01 \\ 03 \\ 02 \end{pmatrix} \text{SB}(a). \end{aligned}$$

Then

$$\begin{pmatrix} d_{0j} \\ d_{1j} \\ d_{2j} \\ d_{3j} \end{pmatrix} = T_0(s_{0j}) \oplus T_1(s_{1(j+1 \bmod 4)}) \oplus T_2(s_{2(j+2 \bmod 4)}) \oplus T_3(s_{3(j+3 \bmod 4)})$$

AES – T-tables

- The four tables T_0, T_1, T_2, T_3 of size 8×32 can be used to implement SubBytes, ShiftRows, and MixColumns
- Those four tables are called *T-tables* for AES.
- To store the T-tables we need processors with a word length of 32 or above.
- They cannot be used for the last round of AES as there is no Mixcolumns operation.

PRESENT Sbox – lookup table

PRESENT Sbox:

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
C	5	6	B	9	0	A	D	3	E	F	8	4	7	1	2

Algorithm 2: A lookup table implementation of PRESENT Sbox in pseudocode.

```
1 integer array [1..16] Sbox = {C, 5, 6, B, 9, 0, A, D, 3, E, F, 8, 4, 7, 1, 2}
2 s = Sbox[s] // Table lookup
```

- As current computer architectures normally use word sizes of at least one byte (generally multiple bytes), it is not efficient to implement Sbox nibble-wise. To optimize the execution time, we can merge two PRESENT Sbox lookup tables

PRESENT Sbox – lookup table

- To optimize the execution time, we can merge two PRESENT Sbox lookup tables

Algorithm 3: A more efficient lookup table implementation of PRESENT Sbox in pseudocode.

```
1 integer array [1..16] Sbox = {C, 5, 6, B, 9, 0, A, D, 3, E, F, 8, 4, 7, 1, 2}
2 integer big_s = Sbox[s & 0x0F] // Lower nibble; &: bitwise AND
3 big_s = big_s ∨ (Sbox[(s≫4) & 0x0F] ≪4) // Upper nibble; ∨: bitwise OR
4 s = big_s // State update
```

PRESENT Sbox – lookup table

- To avoid the bit shifts and boolean operations, it is better to combine two 4×4 Sbox tables into one bigger 8×8 table

SB(0) SB(0)	SB(0) SB(1)	...	SB(0) SB(F)
SB(1) SB(0)	SB(1) SB(1)	...	SB(1) SB(F)
⋮	⋮		⋮
SB(F) SB(0)	SB(F) SB(1)	...	SB(F) SB(F)

Algorithm 4: A lookup table implementation combining two PRESENT Sboxes in parallel in pseudocode.

- integer array** [1..256] Sbox = {CC, C5, ..., C1, C2, 5C, 55, ..., 51, 52, ... 2C, 25, ..., 21, 22}
 - s = Sbox[s] // Table lookup of two nibbles in parallel
-

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
C	5	6	B	9	0	A	D	3	E	F	8	4	7	1	2

Implementing permutations

- The efficiency of the implementation is highly dependent on the design of the permutation.
- For AES ShiftRows, the bytes are permuted, making it easier to implement.
- For PRESENT pLayer, the bit level permutations are “free” in hardware as we just need to reorder the wires, no new gates are required.
- However, in software, extracting each bit and putting it in the right position is time-consuming.

PRESENT – combine sBoxLayer and pLayer

- Construct sixteen 4×64 lookup tables, TB1, TB2, ..., TB16.
- The input of TB $_i$ is given by the i th nibble of the cipher state at the input of sBoxLayer.
- The outputs are 64-bit values with mostly 0s except for 4 bits that correspond to the input nibble.

Remark

For PRESENT specification, we consider the 0th bit of a value as the rightmost bit in its binary representation. For example, the 0th bit of $3 = 011_2$ is 1, the 1st bit is 1 and the 2nd bit is 0.

PRESENT – combine sBoxLayer and pLayer

- TB1: input is the first nibble of the cipher state at the input of sBoxLayer.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	16	32	48	1	17	33	49	2	18	34	50	3	19	35	51

- The Sbox output corresponding to this nibble should go to bits 0, 16, 32 and 48 of the output of pLayer.
- Each entry of TB1 is a 64-bit value with bits in positions 0, 16, 32 and 48 given by the Sbox output, and the other bits are all 0.

Example

If the input is A, the Sbox output should be $F = 1111_2$ and

$$TB1[A] = 0 \dots 010 \dots 010 \dots 010 \dots 1,$$

where the 0th, 16th, 32nd and 48th bits are 1. How about $TB1[B] = ?$

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
C	5	6	B	9	0	A	D	3	E	F	8	4	7	1	2

PRESENT – combine sBoxLayer and pLayer

- TB1: input is the first nibble of the cipher state at the input of sBoxLayer.
- The Sbox output corresponding to this nibble should go to bits 0, 16, 32 and 48 of the output of pLayer.
- Each entry of TB1 is a 64-bit value with bits in positions 0, 16, 32 and 48 given by the Sbox output, and the other bits are all 0.

Example

PRESENT Sbox output for input B is 1000_2 , and

$$TB1[B] = 0 \dots 010 \dots 0,$$

where the 48th bit is 1.

PRESENT – combine sBoxLayer and pLayer

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
C	5	6	B	9	0	A	D	3	E	F	8	4	7	1	2

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	16	32	48	1	17	33	49	2	18	34	50	3	19	35	51

Example

TB2 takes the second nibble of the cipher state as input. The output bits should be positioned at ? And

$$\text{TB2}[B] = ?$$

PRESENT – combine sBoxLayer and pLayer

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	16	32	48	1	17	33	49	2	18	34	50	3	19	35	51

Example

TB2 takes the second nibble of the cipher state as input. The output bits should be positioned at 1, 17, 33 and 49. Thus

$$\text{TB2}[\text{B}] = 0 \dots 010 \dots 0,$$

where only the 49th bit is 1.

PRESENT – combine sBoxLayer and pLayer

- A 4×64 table takes 64×2^4 bits and those sixteen tables take 16384 bits of memory.
- Compared to one Sbox table, which is 64 bits, this is much bigger, but these tables also implement pLayer of PRESENT.
- The speed can be further improved by merging two Sbox computations and constructing eight 8×64 lookup tables.
 - The memory consumption will be the same, 16384 bits.
 - But the speed will be much faster.

Assignment 3

- Implementation should use algebraic normal form
- Bring computer next week
- Be prepared to answer questions