

Introduction to cryptography

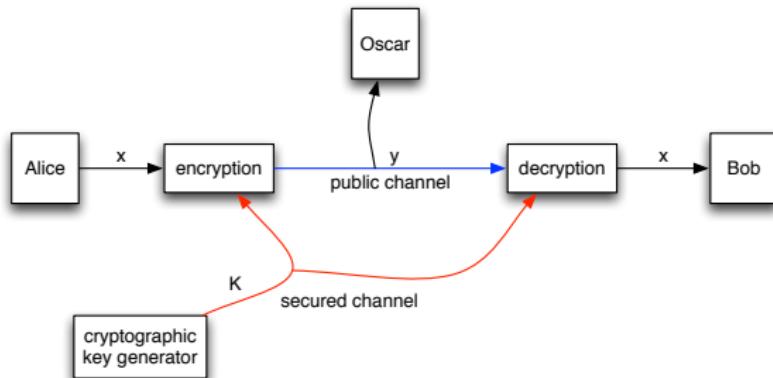
2. Secret-key techniques

Gilles VAN ASSCHE
Olivier MARKOWITCH

INFO-F-405
Université Libre de Bruxelles
2021-2022

© 2019-2021 Gilles Van Assche and Olivier Markowitch. All rights reserved.

Setting



The secret key comes from, e.g.,

- hand-to-hand distribution
- smartcard
- a key establishment scheme

A layered approach

- Primitives
 - Keystream generators: e.g., RC4, Trivium, Salsa20
 - Block ciphers: e.g., DES, AES, Skinny, Saturnin
 - Permutations: e.g., KECCAK- f , Ascon, Gimli, Xoodoo
- Modes of operations (and constructions) to build
 - an encryption scheme: e.g., stream cipher, CBC, CTR, sponge
 - an authentication scheme: e.g., CBC-MAC, sponge
 - an auth. enc. scheme: e.g., GCM, SIV, SpongeWrap
 - a hash function: e.g., Davies-Meyer + Merkle-Damgård, sponge
(see next chapter)
 - one primitive from another: e.g.,
 - CTR turns a block cipher into a keystream generator
 - Even-Mansour turns a permutation into a block cipher

A layered approach

- Primitives
 - Require peer review and cryptanalysis
- Modes of operations (and constructions)
 - Can have generic attacks, i.e., attacks that work regardless of the underlying primitive
 - Are analyzed assuming the primitive is ideal (random)
 - to determine the generic attacks
 - to prove that no other generic attack exists

At the end of the day, this gives us secure schemes

- up to the generic attacks
- until someone finds a flaw in the primitive

A layered approach

- Primitives
 - Require peer review and cryptanalysis
- Modes of operations (and constructions)
 - Can have generic attacks, i.e., attacks that work regardless of the underlying primitive
 - Are analyzed assuming the primitive is ideal (random)
 - to determine the generic attacks
 - to prove that no other generic attack exists

At the end of the day, this gives us secure schemes

- up to the generic attacks
- until someone finds a flaw in the primitive

What is a stream cipher?

A stream cipher is an encryption scheme that uses a keystream generator to output bits gradually and on demand. The output bits are called the *keystream*.

- To encrypt, the keystream is XORed bit by bit with the plaintext.
- To decrypt, the keystream is XORed bit by bit with the ciphertext.

What is a stream cipher?

Keystream generator: $G : K \times \mathbb{N} \rightarrow \mathbb{Z}_2^\infty$

- inputs a **secret key** $k \in K$ and a **diversifier** $d \in \mathbb{N}$
- outputs a long (potentially infinite) keystream $(s_i) \in \mathbb{Z}_2^\infty$

To encrypt plaintext $(m_0, \dots, m_{|m|-1})$:

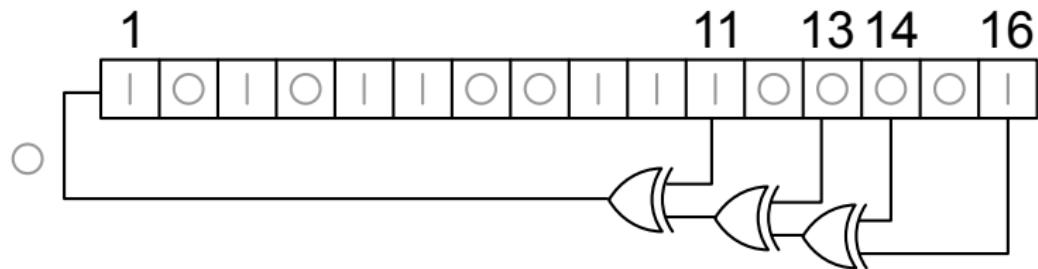
$$c_i = m_i + s_i \quad (\text{in } \mathbb{Z}_2)$$

To decrypt ciphertext $(c_0, \dots, c_{|c|-1})$:

$$m_i = c_i + s_i \quad (\text{in } \mathbb{Z}_2)$$

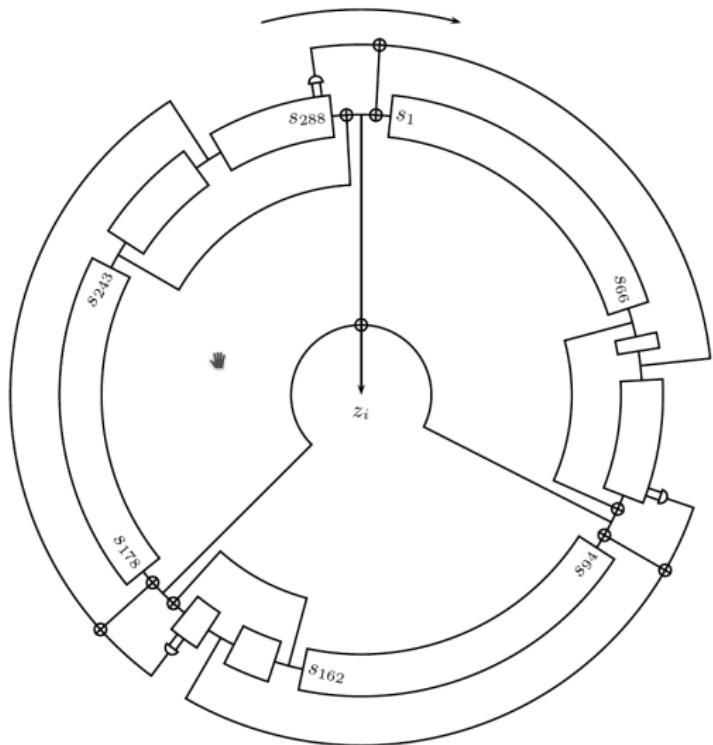
The diversifier d is **public** and must be a **nonce**, i.e., unique per encryption for a given key.

Linear feedback shift register (LFSR)



Feedback polynomial: $1 + D^{11} + D^{13} + D^{14} + D^{16}$

Trivium



[Christophe De Cannière, Bart Preneel, eSTREAM, 2005]

RC4

$S \in \mathbb{Z}_{256}^{256}$, an array of 256 bytes

Initialization($k \in \mathbb{Z}_{256}^*$: **key** and **diversifier**)

- for i from 0 to 255:
 - $S[i] \leftarrow i$
 - $j \leftarrow 0$
- for i from 0 to 255:
 - $j \leftarrow (j + S[i] + k[i \bmod |k|]) \bmod 256$
 - swap values of $S[i]$ and $S[j]$

Keystream generation

...

RC4

$S \in \mathbb{Z}_{256}^{256}$, an array of 256 bytes

Initialization($k \in \mathbb{Z}_{256}^*$: **key** and **diversifier**) ...

Keystream generation

- $i \leftarrow 0, j \leftarrow 0$
- loop as long as necessary:
 - $i \leftarrow (i + 1) \bmod 256$
 - $j \leftarrow (j + S[i]) \bmod 256$
 - swap values of $S[i]$ and $S[j]$
 - $s \leftarrow S[(S[i] + S[j]) \bmod 256]$
 - output s

[presumably Ron Rivest, 1987; secret then anonymously published in 1994]

Block cipher

A **block cipher** is a mapping $E : \mathbb{Z}_2^n \times \mathbb{Z}_2^m \rightarrow \mathbb{Z}_2^n$

- from a **secret key** $k \in \mathbb{Z}_2^m$ and an **input block** $x \in \mathbb{Z}_2^n$
- to an **output block** $y = E_k(x) \in \mathbb{Z}_2^n$

For each key k , it has an inverse: $x = E_k^{-1}(y)$.

Note: Despite its name, a block cipher is not an encryption scheme, except for the very restricted plaintext space \mathbb{Z}_2^n but then it's not even IND-CPA.

Security notions (informally):

- **Pseudo-random permutation (PRP):** without knowing the secret key, it should be infeasible for an adversary having access to $E_k(\cdot)$ to distinguish it from a permutation randomly drawn from the set of all permutations on \mathbb{Z}_2^n
- **Strong PRP (SPRP):** same, but the adversary also gets access to $E_k^{-1}(\cdot)$

Block cipher

A **block cipher** is a mapping $E : \mathbb{Z}_2^n \times \mathbb{Z}_2^m \rightarrow \mathbb{Z}_2^n$

- from a **secret key** $k \in \mathbb{Z}_2^m$ and an **input block** $x \in \mathbb{Z}_2^n$
- to an **output block** $y = E_k(x) \in \mathbb{Z}_2^n$

For each key k , it has an inverse: $x = E_k^{-1}(y)$.

Note: Despite its name, a block cipher is not an encryption scheme, except for the very restricted plaintext space \mathbb{Z}_2^n but then it's not even IND-CPA.

Security notions (informally):

- **Pseudo-random permutation (PRP):** without knowing the secret key, it should be infeasible for an adversary having access to $E_k(\cdot)$ to distinguish it from a permutation randomly drawn from the set of all permutations on \mathbb{Z}_2^n
- **Strong PRP (SPRP):** same, but the adversary also gets access to $E_k^{-1}(\cdot)$

Block cipher

A **block cipher** is a mapping $E : \mathbb{Z}_2^n \times \mathbb{Z}_2^m \rightarrow \mathbb{Z}_2^n$

- from a **secret key** $k \in \mathbb{Z}_2^m$ and an **input block** $x \in \mathbb{Z}_2^n$
- to an **output block** $y = E_k(x) \in \mathbb{Z}_2^n$

For each key k , it has an inverse: $x = E_k^{-1}(y)$.

Note: Despite its name, a block cipher is not an encryption scheme, except for the very restricted plaintext space \mathbb{Z}_2^n but then it's not even IND-CPA.

Security notions (informally):

- **Pseudo-random permutation (PRP):** without knowing the secret key, it should be infeasible for an adversary having access to $E_k(\cdot)$ to distinguish it from a permutation randomly drawn from the set of all permutations on \mathbb{Z}_2^n
- **Strong PRP (SPRP):** same, but the adversary also gets access to $E_k^{-1}(\cdot)$

DES



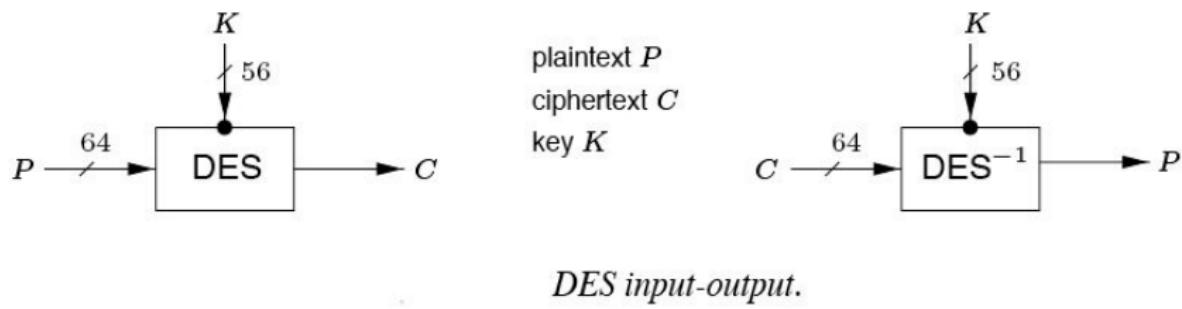
Horst Feistel

Don Coppersmith

DES

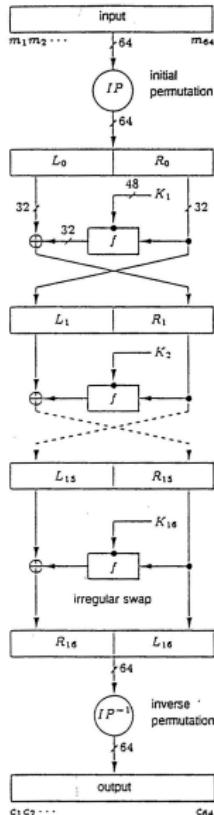
$$\text{DES} : \mathbb{Z}_2^{64} \times \mathbb{Z}_2^{56} \rightarrow \mathbb{Z}_2^{64}$$

Lucifer in 1973 - DES is a standard in 1976



Sorry, this is a bad figure; the input is often *not* plaintext, and the output often *not* ciphertext.

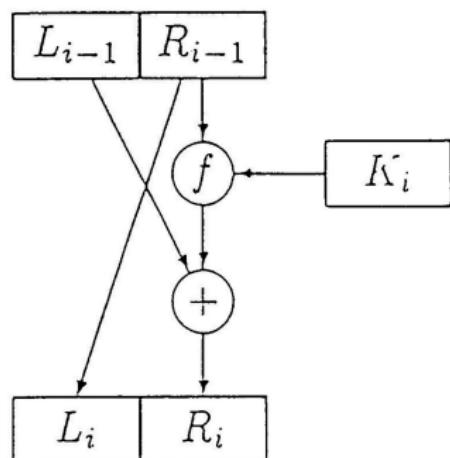
DES is made of rounds



- An initial bit transposition (IP) is applied to the input: $L_0 \parallel R_0 = \text{IP}(x)$, with $L_0, R_0 \in \mathbb{Z}_2^{32}$.
 - Then 16 iterations of the **round function**:
- $$L_i = R_{i-1}$$
- $$R_i = L_{i-1} \oplus f(R_{i-1}, k_i),$$
- where $k_1, \dots, k_{16} \in \mathbb{Z}_2^{48}$ are the 16 sub-keys.
- The inverse of IP is applied before returning the output: $y = \text{IP}^{-1}(R_{16} \parallel L_{16})$.

DES is a Feistel network

One round forward:



$$\begin{aligned}L_i &= R_{i-1} \\R_i &= L_{i-1} \oplus f(R_{i-1}, k_i)\end{aligned}$$

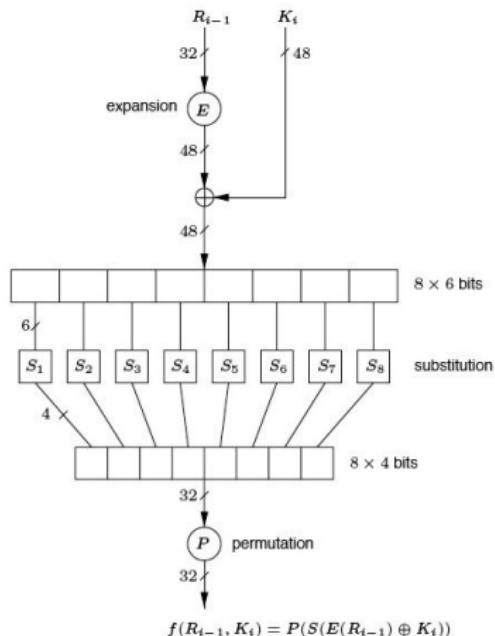
One round backward:

$$\begin{aligned}R_{i-1} &= L_i \\L_{i-1} &= R_i \oplus f(L_i, k_i)\end{aligned}$$

Advantages:

- E and E^{-1} share the same structure, subkeys in reverse order;
- f does not need to be invertible.

Inside DES' f function



$$f : \mathbb{Z}_2^{32} \times \mathbb{Z}_2^{48} \rightarrow \mathbb{Z}_2^{32}$$

- 1 Expand R through E from 32 to $48 = 16 + 2 \times 16$ bits
- 2 Compute $E(R) \oplus K_i$
- 3 Cut in 6-bit blocs B_j
- 4 $C_j = S_j(B_j)$, 8 different S-Boxes
 $S_j : \mathbb{Z}_2^6 \rightarrow \mathbb{Z}_2^4$
- 5 Bit transposition
 $P : \mathbb{Z}_2^{32} \rightarrow \mathbb{Z}_2^{32}$

Figure 7.10: DES inner function f .

DES: Tables for bit transpositions and expansion E

IP							
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

IP ⁻¹							
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

Table 7.2: DES initial permutation and inverse (IP and IP⁻¹).

E					
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

P			
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

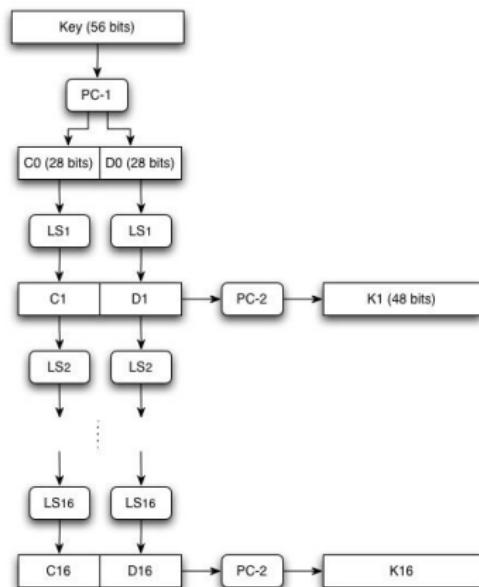
Table 7.3: DES per-round functions: expansion E and permutation P.

DES: Tables of the S-boxes

row	column number															
	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]
S_1																
[0]	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
[1]	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
[2]	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
[3]	15	12	8	2	4	9	1	.7	5	11	3	14	10	0	6	13
S_2																
[0]	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
[1]	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
[2]	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
[3]	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9
S_3																
[0]	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
[1]	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
[2]	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
[3]	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12
S_4																
[0]	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
[1]	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
[2]	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
[3]	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14
S_5																
[0]	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
[1]	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
[2]	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
[3]	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3
S_6																
[0]	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
[1]	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
[2]	9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
[3]	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13
S_7																
[0]	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
[1]	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
[2]	1	4	11	13	12'	3	7	14	10	15	6	8	0	5	9	2
[3]	6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12
S_8																
[0]	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
[1]	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
[2]	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
[3]	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

DES' key schedule

From the secret key $k \in \mathbb{Z}_2^{56}$ to round keys $K_i \in \mathbb{Z}_2^{48}$



- 1 Bit transposition $C_0 \parallel D_0 = \text{PC1}(k)$
 $\text{PC1} : \mathbb{Z}_2^{56} \rightarrow \mathbb{Z}_2^{28} \times \mathbb{Z}_2^{28}$
- 2 For $i = 1$ to 16 :
 - $C_i = \text{LS}_i(C_{i-1})$ and $D_i = \text{LS}_i(D_{i-1})$, where LS_i is a circular shift to the left by one position when $i = 1, 2, 9, 16$ and by two positions otherwise
 - $K_i = \text{PC2}(C_i \parallel D_i)$, with
 $\text{PC2} : \mathbb{Z}_2^{56} \rightarrow \mathbb{Z}_2^{48}$

DES: Tables for PC1 and PC2

PC1						
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
above for C_i ; below for D_i						
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

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

Table 7.4: DES key schedule bit selections (PC1 and PC2).

DES non-ideal properties

- 4 weak keys k : E_k is an involution, i.e., $\forall x E_k(E_k(x)) = x$
- 6 pairs of semi-weak keys (k_1, k_2) such that $\forall x E_{k_1}(E_{k_2}(x)) = x$
- Complementarity property: $E_k(x) = y \Leftrightarrow E_{\bar{k}}(\bar{x}) = \bar{y}$
⇒ reduces the exhaustive search by one bit

DES security

- Exhaustive key search in 2^{56}
feasible with dedicated hardware or FPGA
- Susceptible to **differential cryptanalysis** [Biham and Shamir, 1993]
- Susceptible to **linear cryptanalysis** [Matsui, 1993]

attack method	data complexity		storage complexity	processing complexity
	known	chosen		
exhaustive precomputation	—	1	2^{56}	1 (table lookup)
exhaustive search	1	—	negligible	2^{55}
linear cryptanalysis	2^{43} (85%)	—	for texts	2^{43}
	2^{38} (10%)	—	for texts	2^{50}
differential cryptanalysis	—	2^{47}	for texts	2^{47}
	2^{55}	—	for texts	2^{55}

Table 7.7: DES strength against various attacks.

Differential cryptanalysis

Analyzes the propagation of differences through the rounds

$$f(x) = y$$

$$f(\delta \oplus x) = y + \Delta$$

(x varies, δ constant)



Eli Biham and Adi Shamir

Linear cryptanalysis

Analyzes the correlation between input and output parities

$$\left| \Pr[v^T x = u^T f(x)] - \frac{1}{2} \right|$$

(x varies, u and v constant)



Mitsuru Matsui

Triple-DES

Triple-key triple-DES (168-bit keys)

$$3\text{DES}_{k_1 \parallel k_2 \parallel k_3} = \text{DES}_{k_3} \circ \text{DES}_{k_2}^{-1} \circ \text{DES}_{k_1}$$

Double-key triple-DES (112-bit keys)

$$3\text{DES}_{k_1 \parallel k_2} = \text{DES}_{k_1} \circ \text{DES}_{k_2}^{-1} \circ \text{DES}_{k_1}$$

Retro-compatibility with single DES

$$3\text{DES}_{k \parallel k \parallel k} = \text{DES}_k \circ \text{DES}_k^{-1} \circ \text{DES}_k = \text{DES}_k$$

Triple-DES

Triple-key triple-DES (168-bit keys)

$$3\text{DES}_{k_1 \parallel k_2 \parallel k_3} = \text{DES}_{k_3} \circ \text{DES}_{k_2}^{-1} \circ \text{DES}_{k_1}$$

Double-key triple-DES (112-bit keys)

$$3\text{DES}_{k_1 \parallel k_2} = \text{DES}_{k_1} \circ \text{DES}_{k_2}^{-1} \circ \text{DES}_{k_1}$$

Retro-compatibility with single DES

$$3\text{DES}_{k \parallel k \parallel k} = \text{DES}_k \circ \text{DES}_k^{-1} \circ \text{DES}_k = \text{DES}_k$$

Triple-DES

Triple-key triple-DES (168-bit keys)

$$3\text{DES}_{k_1 \parallel k_2 \parallel k_3} = \text{DES}_{k_3} \circ \text{DES}_{k_2}^{-1} \circ \text{DES}_{k_1}$$

Double-key triple-DES (112-bit keys)

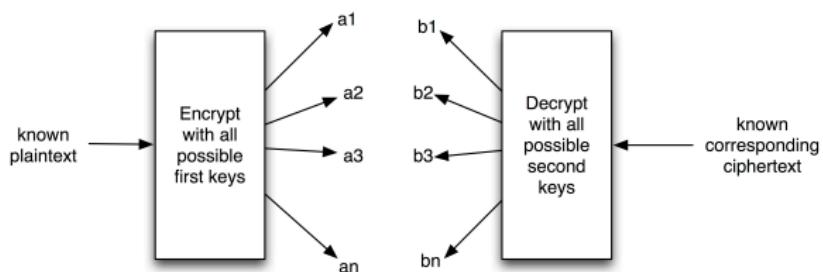
$$3\text{DES}_{k_1 \parallel k_2} = \text{DES}_{k_1} \circ \text{DES}_{k_2}^{-1} \circ \text{DES}_{k_1}$$

Retro-compatibility with single DES

$$3\text{DES}_{k \parallel k \parallel k} = \text{DES}_k \circ \text{DES}_k^{-1} \circ \text{DES}_k = \text{DES}_k$$

Why not Double-DES?

Why not $2\text{DES}_{k_1 \parallel k_2} = \text{DES}_{k_2} \circ \text{DES}_{k_1}$?



The adversary looks for matches between the a_i and the b_j

Attack in time 2^{57} with 2^{56} memory.

Competition for a new standard

In September 1997, USA's National Institute of Standards and Technology (NIST) called for a new block cipher family, with

- block size $n = 128$ bits
- key sizes $m = 128, 192$ and 256 bits

Organized as an open competition, the winner would become the Advanced Encryption Standard (AES).

- August 1998: 15 candidates
- August 1999: 5 finalists
 - MARS [IBM, including Don Coppersmith]
 - RC6 [Ron Rivest, Matt Robshaw, Ray Sidney and Yiqun Lisa Yin]
 - Rijndael [Joan Daemen and Vincent Rijmen]
 - Serpent [Ross Anderson, Eli Biham and Lars Knudsen]
 - Twofish [Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall and Niels Ferguson]
- October 2000: Rijndael announced as the winner
- November 2001: AES standard [FIPS 197]

Competition for a new standard

In September 1997, USA's National Institute of Standards and Technology (NIST) called for a new block cipher family, with

- block size $n = 128$ bits
- key sizes $m = 128, 192$ and 256 bits

Organized as an open competition, the winner would become the Advanced Encryption Standard (AES).

- August 1998: 15 candidates
- August 1999: 5 finalists
 - MARS [IBM, including Don Coppersmith]
 - RC6 [Ron Rivest, Matt Robshaw, Ray Sidney and Yiqun Lisa Yin]
 - Rijndael [Joan Daemen and Vincent Rijmen]
 - Serpent [Ross Anderson, Eli Biham and Lars Knudsen]
 - Twofish [Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall and Niels Ferguson]
- October 2000: Rijndael announced as the winner
- November 2001: AES standard [FIPS 197]

Rijndael



Joan Daemen and Vincent Rijmen

Rijndael – AES

NIST standardized 3 instances of Rijndael with block size 128 bits and key sizes $\in \{128, 192, 256\}$:

AES-128 : $\mathbb{Z}_2^{128} \times \mathbb{Z}_2^{128} \rightarrow \mathbb{Z}_2^{128}$	10 rounds
AES-192 : $\mathbb{Z}_2^{128} \times \mathbb{Z}_2^{192} \rightarrow \mathbb{Z}_2^{128}$	12 rounds
AES-256 : $\mathbb{Z}_2^{128} \times \mathbb{Z}_2^{256} \rightarrow \mathbb{Z}_2^{128}$	14 rounds

Rijndael - $\mathbb{Z}_2^{128} \leftrightarrow \text{GF}(2^8)^{4 \times 4}$

Input $x \in \mathbb{Z}_2^{128}$ is mapped to an array of 4×4 bytes:

$$\begin{pmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{pmatrix} \leftarrow \begin{pmatrix} x_0 & x_4 & x_8 & x_{12} \\ x_1 & x_5 & x_9 & x_{13} \\ x_2 & x_6 & x_{10} & x_{14} \\ x_3 & x_7 & x_{11} & x_{15} \end{pmatrix}$$

where x_i is the i^{th} byte of x . And vice-versa for the output.

Each byte $s_{i,j}$ represents an element of the finite field $\text{GF}(2^8)$.

Rijndael - $\mathbb{Z}_2^{128} \leftrightarrow \text{GF}(2^8)^{4 \times 4}$

Input $x \in \mathbb{Z}_2^{128}$ is mapped to an array of 4×4 bytes:

$$\begin{pmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{pmatrix} \leftarrow \begin{pmatrix} x_0 & x_4 & x_8 & x_{12} \\ x_1 & x_5 & x_9 & x_{13} \\ x_2 & x_6 & x_{10} & x_{14} \\ x_3 & x_7 & x_{11} & x_{15} \end{pmatrix}$$

where x_i is the i^{th} byte of x . And vice-versa for the output.

Each byte $s_{i,j}$ represents an element of the finite field $\text{GF}(2^8)$.

GF(2⁸) in Rijndael

- GF(2⁸) is the finite field (Galois Field) of size 2⁸ = 256.
- Rijndael uses the representation GF(2)[x]/(x⁸ + x⁴ + x³ + x + 1).
- A byte with arithmetic value $s = \sum_{n=0}^7 s_i 2^i$ represents the polynomial $s = \sum_{n=0}^7 s_i x^n$ in GF(2)[x].
E.g., 0x83 represents $x^7 + x + 1$.
- Addition is like a bitwise XOR operations
E.g., $(x^7 + x + 1) + (x^6 + x) = (x^7 + x^6 + 1) \Leftrightarrow 0x83 \oplus 0x42 = 0xC1$
- Multiplication is done modulo $x^8 + x^4 + x^3 + x + 1$
E.g., $x(x^7 + x + 1) = x^8 + x^2 + x = (x^8 + x^2 + x) + (x^8 + x^4 + x^3 + x + 1) = x^4 + x^3 + x^2 + 1$

GF(2⁸) in Rijndael

- GF(2⁸) is the finite field (Galois Field) of size 2⁸ = 256.
- Rijndael uses the representation GF(2)[x]/(x⁸ + x⁴ + x³ + x + 1).
- A byte with arithmetic value $s = \sum_{n=0}^7 s_i 2^i$ represents the polynomial $s = \sum_{n=0}^7 s_i x^i$ in GF(2)[x].
E.g., 0x83 represents $x^7 + x + 1$.
- Addition is like a bitwise XOR operations
E.g., $(x^7 + x + 1) + (x^6 + x) = (x^7 + x^6 + 1) \Leftrightarrow 0x83 \oplus 0x42 = 0xC1$
- Multiplication is done modulo $x^8 + x^4 + x^3 + x + 1$
E.g., $x(x^7 + x + 1) = x^8 + x^2 + x = (x^8 + x^2 + x) + (x^8 + x^4 + x^3 + x + 1) = x^4 + x^3 + x^2 + 1$

GF(2⁸) in Rijndael

- GF(2⁸) is the finite field (Galois Field) of size 2⁸ = 256.
- Rijndael uses the representation $GF(2)[x]/(x^8 + x^4 + x^3 + x + 1)$.
- A byte with arithmetic value $s = \sum_{n=0}^7 s_i 2^i$ represents the polynomial $s = \sum_{n=0}^7 s_i x^i$ in $GF(2)[x]$.
E.g., 0x83 represents $x^7 + x + 1$.
- Addition is like a bitwise XOR operations
E.g., $(x^7 + x + 1) + (x^6 + x) = (x^7 + x^6 + 1) \Leftrightarrow 0x83 \oplus 0x42 = 0xC1$
- Multiplication is done modulo $x^8 + x^4 + x^3 + x + 1$
E.g., $x(x^7 + x + 1) = x^8 + x^2 + x = (x^8 + x^2 + x) + (x^8 + x^4 + x^3 + x + 1) = x^4 + x^3 + x^2 + 1$

GF(2⁸) in Rijndael

- GF(2⁸) is the finite field (Galois Field) of size 2⁸ = 256.
- Rijndael uses the representation $GF(2)[x]/(x^8 + x^4 + x^3 + x + 1)$.
- A byte with arithmetic value $s = \sum_{n=0}^7 s_i 2^i$ represents the polynomial $s = \sum_{n=0}^7 s_i x^i$ in $GF(2)[x]$.
E.g., 0x83 represents $x^7 + x + 1$.
- Addition is like a bitwise XOR operations
E.g., $(x^7 + x + 1) + (x^6 + x) = (x^7 + x^6 + 1) \Leftrightarrow 0x83 \oplus 0x42 = 0xC1$
- Multiplication is done modulo $x^8 + x^4 + x^3 + x + 1$
E.g., $x(x^7 + x + 1) = x^8 + x^2 + x = (x^8 + x^2 + x) + (x^8 + x^4 + x^3 + x + 1) = x^4 + x^3 + x^2 + 1$

GF(2⁸) in Rijndael

- GF(2⁸) is the finite field (Galois Field) of size 2⁸ = 256.
- Rijndael uses the representation $GF(2)[x]/(x^8 + x^4 + x^3 + x + 1)$.
- A byte with arithmetic value $s = \sum_{n=0}^7 s_i 2^i$ represents the polynomial $s = \sum_{n=0}^7 s_i x^i$ in $GF(2)[x]$.
E.g., 0x83 represents $x^7 + x + 1$.
- Addition is like a bitwise XOR operations
E.g., $(x^7 + x + 1) + (x^6 + x) = (x^7 + x^6 + 1) \Leftrightarrow 0x83 \oplus 0x42 = 0xC1$
- Multiplication is done modulo $x^8 + x^4 + x^3 + x + 1$
E.g., $x(x^7 + x + 1) = x^8 + x^2 + x = (x^8 + x^2 + x) + (x^8 + x^4 + x^3 + x + 1) = x^4 + x^3 + x^2 + 1$

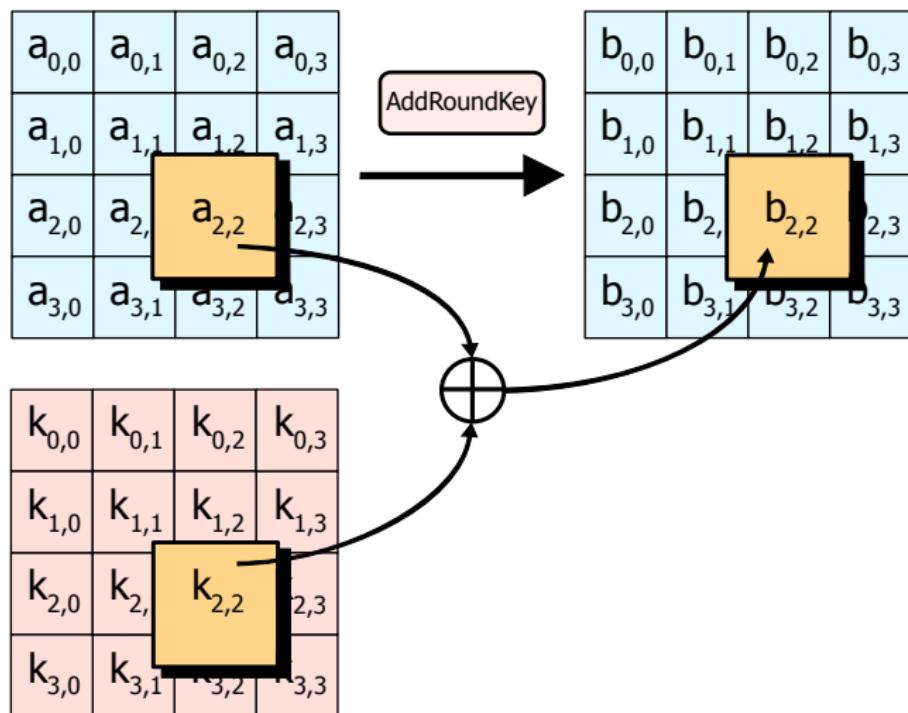
Rijndael data path

Input $x \in \mathbb{Z}_2^{128}$ and $k \in \mathbb{Z}_2^{128}$ (or 192 or 256)

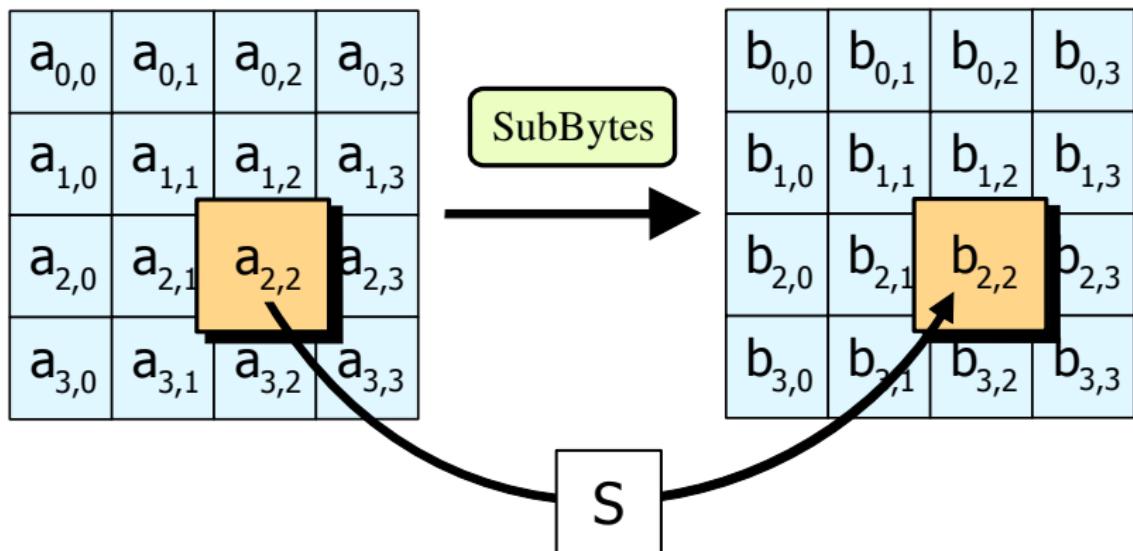
- Key schedule: $(K_i) \leftarrow \text{KeyExpansion}(k)$
- $\text{state} \leftarrow x$ (but represented as $\text{GF}(2^8)^{4 \times 4}$)
- $\text{AddRoundKey}(\text{state}, K_0)$
- For each round $i = 1$ to 9 (or 11 or 13):
 - $\text{SubBytes}(\text{state})$
 - $\text{ShiftRows}(\text{state})$
 - $\text{MixColumns}(\text{state})$
 - $\text{AddRoundKey}(\text{state}, K_i)$
- And for the last round:
 - $\text{SubBytes}(\text{state})$
 - $\text{ShiftRows}(\text{state})$
 - $\text{AddRoundKey}(\text{state}, K_{10} \text{ (or } 12 \text{ or } 14\text{)})$

Output $y \leftarrow \text{state back in } \mathbb{Z}_2^{128}$

Rijndael – AddRoundKey



Rijndael – SubBytes



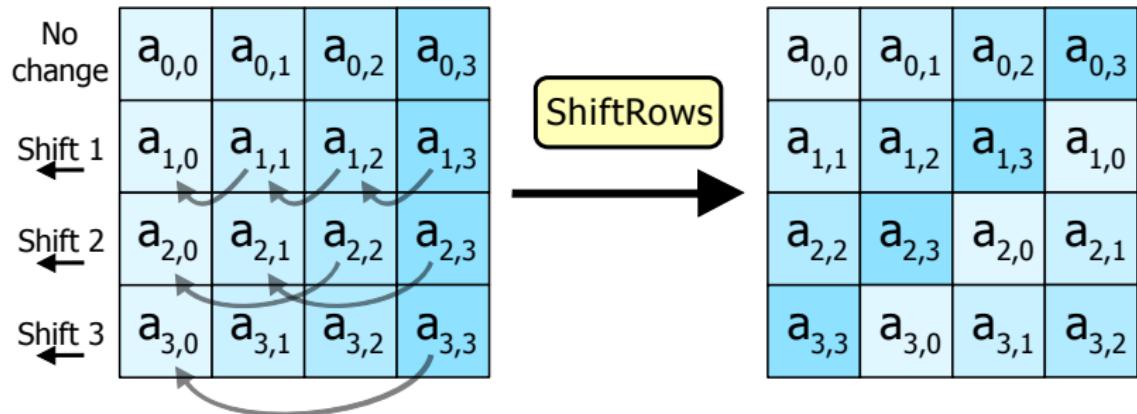
Rijndael – SubBytes

		y																
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	
x		0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
0	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0		
1	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15		
2	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75		
3	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84		
4	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf		
5	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8		
6	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2		
7	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73		
8	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		

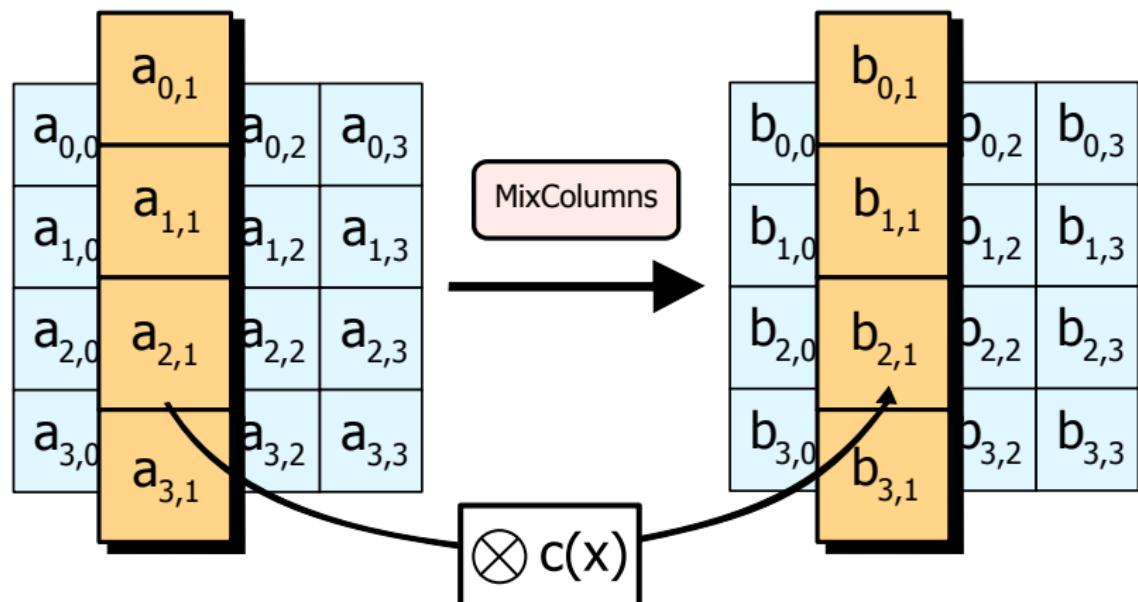
Figure 7. S-box: substitution values for the byte xy (in hexadecimal format).

The Rijndael S-box is based on the multiplicative inverse in $\text{GF}(2^8)$.

Rijndael – ShiftRows



Rijndael – MixColumns



Rijndael – MixColumns

Each column undergoes the following matrix multiplication:

$$\begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix} = \begin{pmatrix} x & x+1 & 1 & 1 \\ 1 & x & x+1 & 1 \\ 1 & 1 & x & x+1 \\ x+1 & 1 & 1 & x \end{pmatrix} \begin{pmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{pmatrix}$$

The inverse of MixColumns uses the inverse matrix:

$$\begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix} = \begin{pmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{pmatrix} \begin{pmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{pmatrix}$$

Rijndael – MixColumns

Each column undergoes the following matrix multiplication:

$$\begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \begin{pmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{pmatrix}$$

The inverse of MixColumns uses the inverse matrix:

$$\begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix} = \begin{pmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{pmatrix} \begin{pmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{pmatrix}$$

Rijndael – MixColumns

Each column undergoes the following matrix multiplication:

$$\begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \begin{pmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{pmatrix}$$

The inverse of MixColumns uses the inverse matrix:

$$\begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix} = \begin{pmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{pmatrix} \begin{pmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{pmatrix}$$

Rijndael inverse data path

Input $y \in \mathbb{Z}_2^{128}$ and $k \in \mathbb{Z}_2^{128}$ (or 192 or 256)

- Equivalent key schedule: $(K'_i) \leftarrow \text{EqKeyExpansion}(k)$
- state $\leftarrow y$ (but represented as $\text{GF}(2^8)^{4 \times 4}$)
- AddRoundKey(state, K'_{10} (or 12 or 14))
- For each round $i = 9$ (or 11 or 13) down to 1:
 - InvSubBytes(state)
 - InvShiftRows(state)
 - InvMixColumns(state)
 - AddRoundKey(state, K'_i)
- And for the last round:
 - InvSubBytes(state)
 - InvShiftRows(state)
 - AddRoundKey(state, K'_0)

Output $x \leftarrow$ state back in \mathbb{Z}_2^{128}

What is a mode of operation?

Going from a block cipher to

- an encryption scheme that works for plaintexts of any lengths
- an authentication scheme that works for messages of any lengths

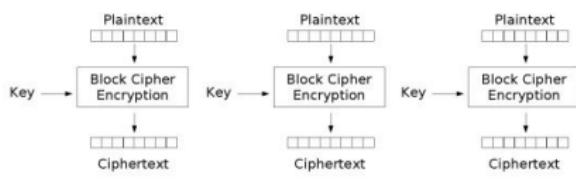
A bad example: Electronic codebook (ECB)

Input: **secret key** $k \in \mathbb{Z}_2^m$,
plaintext $p \in \mathbb{Z}_2^*$, output:
ciphertext $c \in (\mathbb{Z}_2^n)^*$

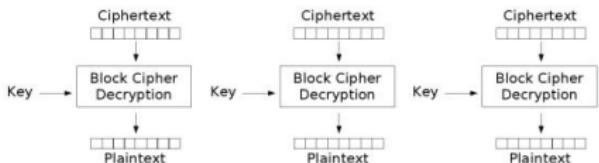
- Pad p with 10^* to reach a multiple of the block size n
- Cut $p \parallel 10^*$ into blocks p_i of size n
- Process each block independently through E_k

$$c_i = E_k(p_i)$$

And similarly with E_k^{-1} for decryption.

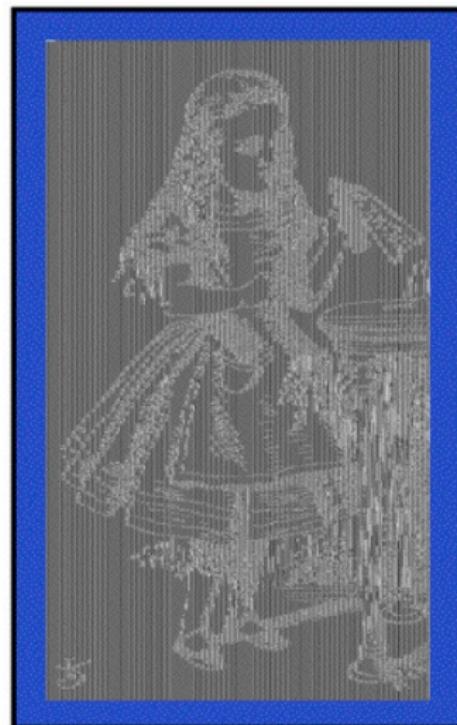


Electronic Codebook (ECB) mode encryption

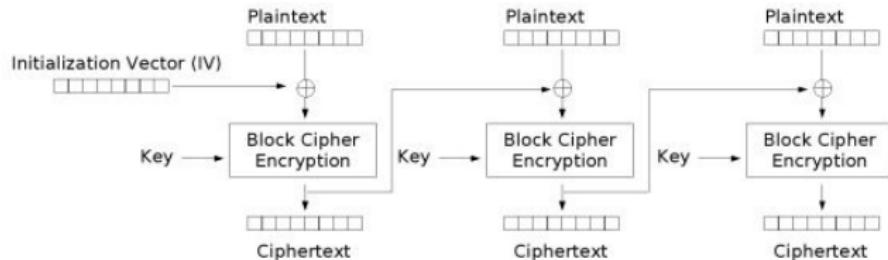


Electronic Codebook (ECB) mode decryption

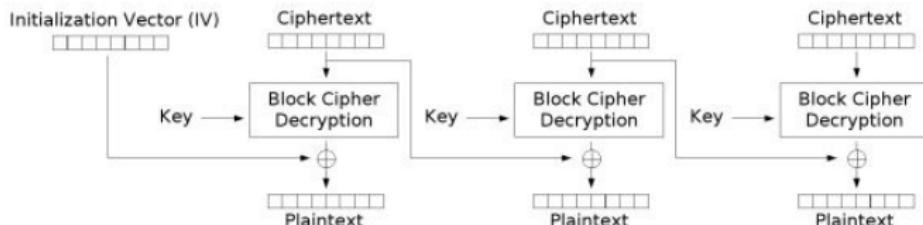
A bad example: Electronic codebook (ECB)



Ciphertext block chaining (CBC)



Cipher Block Chaining (CBC) mode encryption



Cipher Block Chaining (CBC) mode decryption

Ciphertext block chaining (CBC)

Input: **secret key** $k \in \mathbb{Z}_2^m$, **plaintext** $p \in \mathbb{Z}_2^*$ and **diversifier** $d \in \mathbb{N}$

Output: ciphertext $c \in (\mathbb{Z}_2^n)^*$

- Pad p with 10^* and cut into blocks (p_1, p_2, \dots)
- Define $c_0 = E_k(d)$ and then for $i \geq 1$

$$c_i = E_k(p_i \oplus c_{i-1})$$

To decrypt:

$$p_i = E_k^{-1}(c_i) \oplus c_{i-1}$$

Here again, the diversifier d must be a **nonce**.

The limits of CBC

What happens if $c_i = c'_j$ for some ciphertext blocks encrypted under the same key?

$$c_i = c'_j$$

$$E_k(p_i \oplus c_{i-1}) = E_k(p'_j \oplus c'_{j-1})$$

$$p_i \oplus c_{i-1} = p'_j \oplus c'_{j-1}$$

$$p_i \oplus p'_j = c_{i-1} \oplus c'_{j-1}$$

Information is revealed on the plaintext!

How likely is $c_i = c'_j$?

The limits of CBC

What happens if $c_i = c'_j$ for some ciphertext blocks encrypted under the same key?

$$c_i = c'_j$$

$$E_k(p_i \oplus c_{i-1}) = E_k(p'_j \oplus c'_{j-1})$$

$$p_i \oplus c_{i-1} = p'_j \oplus c'_{j-1}$$

$$p_i \oplus p'_j = c_{i-1} \oplus c'_{j-1}$$

Information is revealed on the plaintext!

How likely is $c_i = c'_j$?

The limits of CBC

What happens if $c_i = c'_j$ for some ciphertext blocks encrypted under the same key?

$$c_i = c'_j$$

$$E_k(p_i \oplus c_{i-1}) = E_k(p'_j \oplus c'_{j-1})$$

$$p_i \oplus c_{i-1} = p'_j \oplus c'_{j-1}$$

$$p_i \oplus p'_j = c_{i-1} \oplus c'_{j-1}$$

Information is revealed on the plaintext!

How likely is $c_i = c'_j$?

Intermezzo: the birthday paradox

Say we have 2^n objects and we make L draws with replacement. How much does L need to be to have a good chance to get a collision?

Sketch:

- After L draws, there are $\binom{L}{2} = \frac{L(L-1)}{2} \approx \frac{L^2}{2}$ pairs
- For each pair, the probability of getting the same object is 2^{-n}
- The probability of collision is therefore $\approx \frac{L^2}{2^{n+1}}$
- When $L = \sqrt{2^n} = 2^{n/2}$, this probability $\approx \frac{1}{2}$

Answer: $L \approx 2^{n/2}$.

Intermezzo: the birthday paradox

Say we have 2^n objects and we make L draws with replacement. How much does L need to be to have a good chance to get a collision?

Sketch:

- After L draws, there are $\binom{L}{2} = \frac{L(L-1)}{2} \approx \frac{L^2}{2}$ pairs
- For each pair, the probability of getting the same object is 2^{-n}
- The probability of collision is therefore $\approx \frac{L^2}{2^{n+1}}$
- When $L = \sqrt{2^n} = 2^{n/2}$, this probability $\approx \frac{1}{2}$

Answer: $L \approx 2^{n/2}$.

Intermezzo: the birthday paradox

Say we have 2^n objects and we make L draws with replacement. How much does L need to be to have a good chance to get a collision?

Sketch:

- After L draws, there are $\binom{L}{2} = \frac{L(L-1)}{2} \approx \frac{L^2}{2}$ pairs
- For each pair, the probability of getting the same object is 2^{-n}
- The probability of collision is therefore $\approx \frac{L^2}{2^{n+1}}$
- When $L = \sqrt{2^n} = 2^{n/2}$, this probability $\approx \frac{1}{2}$

Answer: $L \approx 2^{n/2}$.

Intermezzo: the birthday paradox

Say we have 2^n objects and we make L draws with replacement. How much does L need to be to have a good chance to get a collision?

Sketch:

- After L draws, there are $\binom{L}{2} = \frac{L(L-1)}{2} \approx \frac{L^2}{2}$ pairs
- For each pair, the probability of getting the same object is 2^{-n}
- The probability of collision is therefore $\approx \frac{L^2}{2^{n+1}}$
- When $L = \sqrt{2^n} = 2^{n/2}$, this probability $\approx \frac{1}{2}$

Answer: $L \approx 2^{n/2}$.

Intermezzo: the birthday paradox

Say we have 2^n objects and we make L draws with replacement. How much does L need to be to have a good chance to get a collision?

Sketch:

- After L draws, there are $\binom{L}{2} = \frac{L(L-1)}{2} \approx \frac{L^2}{2}$ pairs
- For each pair, the probability of getting the same object is 2^{-n}
- The probability of collision is therefore $\approx \frac{L^2}{2^{n+1}}$
- When $L = \sqrt{2^n} = 2^{n/2}$, this probability $\approx \frac{1}{2}$

Answer: $L \approx 2^{n/2}$.

Intermezzo: the birthday paradox

Say we have 2^n objects and we make L draws with replacement. How much does L need to be to have a good chance to get a collision?

Sketch:

- After L draws, there are $\binom{L}{2} = \frac{L(L-1)}{2} \approx \frac{L^2}{2}$ pairs
- For each pair, the probability of getting the same object is 2^{-n}
- The probability of collision is therefore $\approx \frac{L^2}{2^{n+1}}$
- When $L = \sqrt{2^n} = 2^{n/2}$, this probability $\approx \frac{1}{2}$

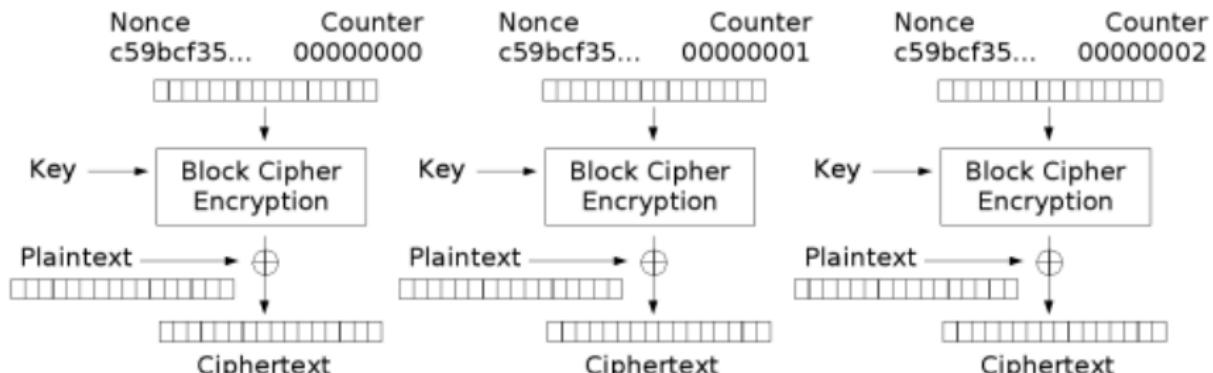
Answer: $L \approx 2^{n/2}$.

The limits of CBC

How likely is $c_i = c'_j$?

- This becomes very likely as the number of blocks encrypted under the same key reaches the order of $2^{n/2}$, with n the block size.
- This is more of a problem for DES (2^{32} blocks) than for AES (2^{64} blocks).

Counter (CTR)



Counter (CTR) mode encryption

Counter (CTR)

Input: **secret key** $k \in \mathbb{Z}_2^m$, **plaintext** $p \in \mathbb{Z}_2^*$ and **diversifier** $d \in \mathbb{N}$

Output: ciphertext $c \in \mathbb{Z}_2^*$

- Cut into blocks of n bits (p_1, p_2, \dots, p_x), except for the last one that can be shorter
- Generate the keystream $k_i = E_k(d \parallel i)$

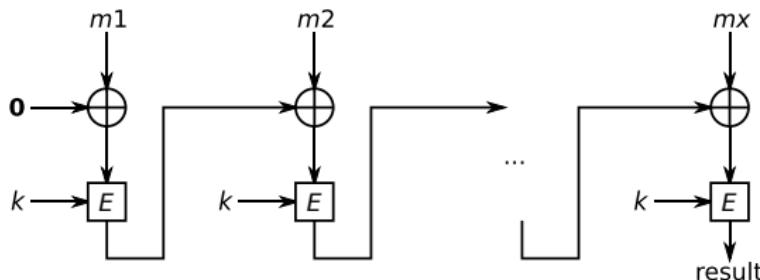
$$c_i = k_i \oplus p_i$$

Note: For the last block, the keystream block k_x is truncated to the size of p_x .

To decrypt, generate the same keystream and

$$p_i = k_i \oplus c_i$$

CBC-MAC



Input: **secret key** $k \in \mathbb{Z}_2^m$, **message** $m \in \mathbb{Z}_2^*$

Output: $\text{MAC} \in \mathbb{Z}_2^n$

- Prepend m with its length
- Pad $\text{len}(m) \| m$ with 10^* and cut into blocks (m_1, m_2, \dots, m_x)
- Define $z_0 = 0^n$ and compute

$$z_i = E_{\color{red}k}(m_i \oplus z_{i-1})$$

- Output $\text{MAC} = z_x$

Authenticated encryption

Best generic method:

- Encrypt (with k_1), then MAC the ciphertext (with k_2)

Specific modes (using only one key k):

- CCM combines CTR and CBC-MAC
- GCM combines CTR and a polynomial MAC in $\text{GF}(2^{128})$
- ... and many many more ...

Permutation

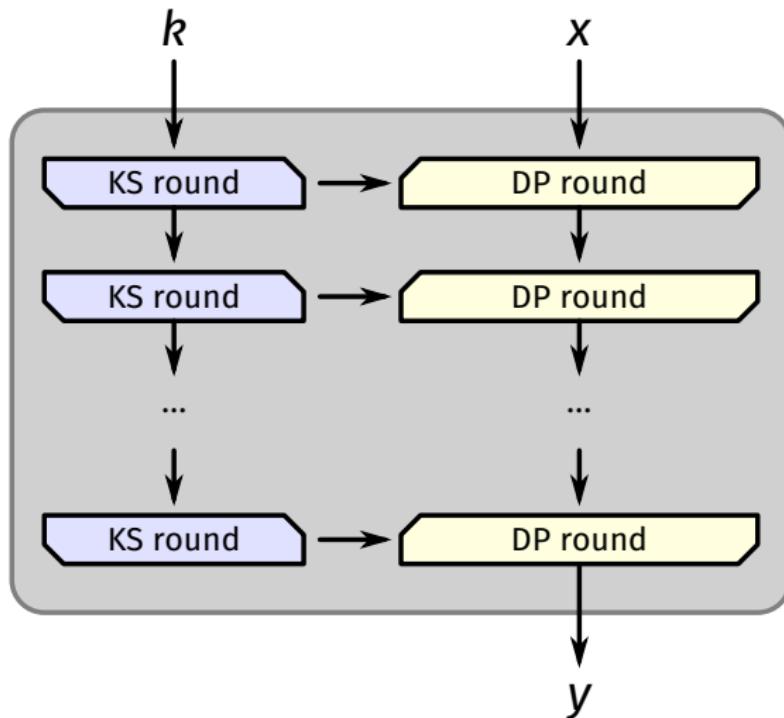
A cryptographic **permutation** is a bijective mapping

$$f : \mathbb{Z}_2^b \rightarrow \mathbb{Z}_2^b$$

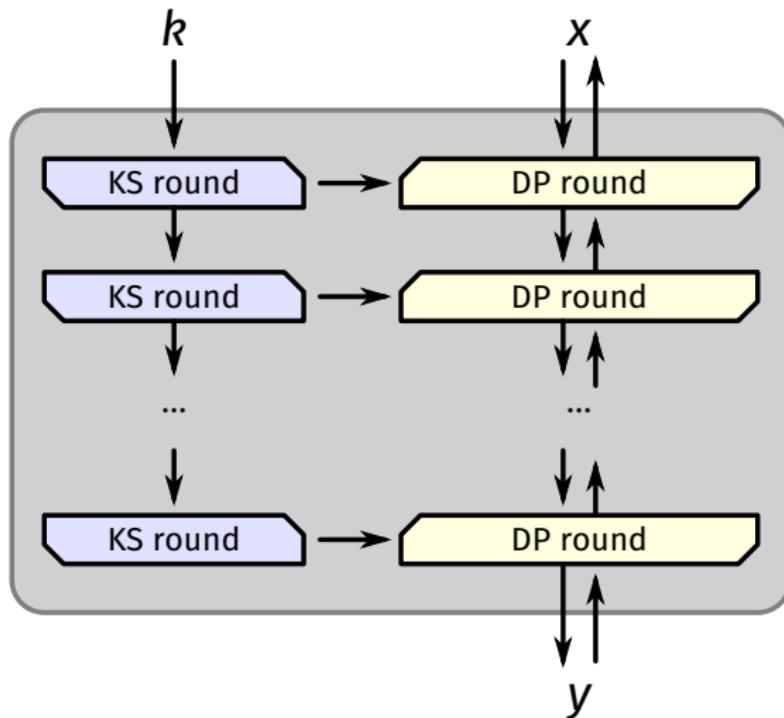
- from an input block $x \in \mathbb{Z}_2^b$
- to an output block $y = f(x) \in \mathbb{Z}_2^b$

It has an inverse: $x = f^{-1}(y)$.

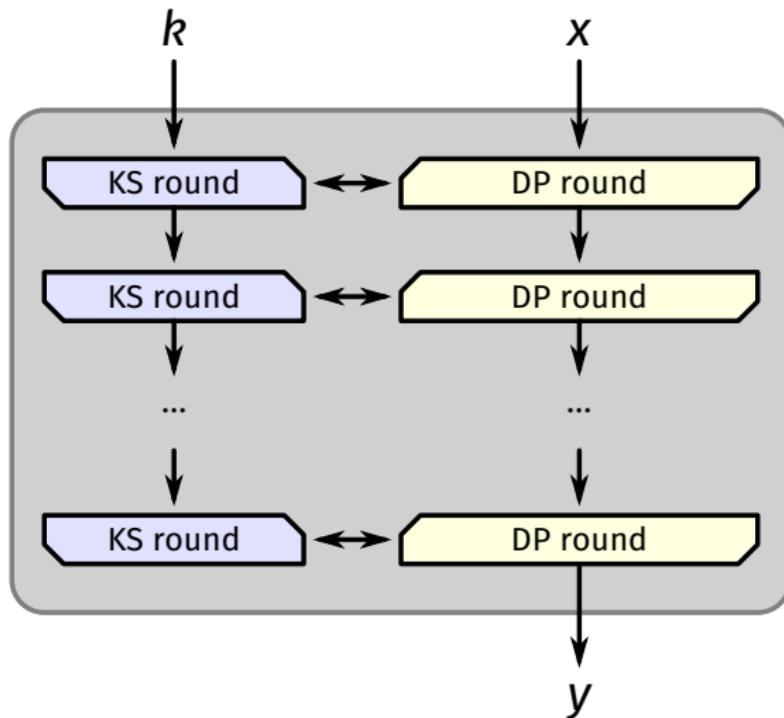
From block ciphers to permutations



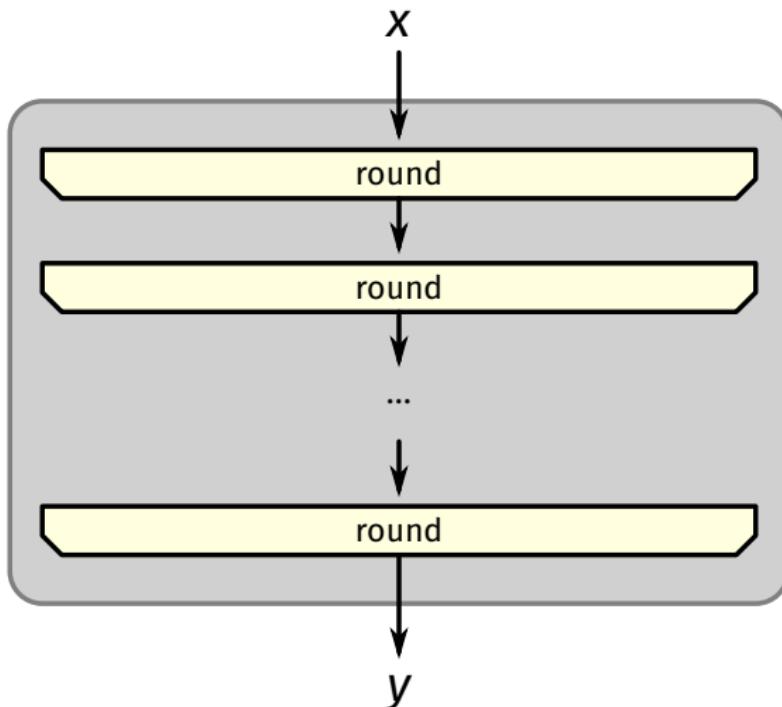
From block ciphers to permutations



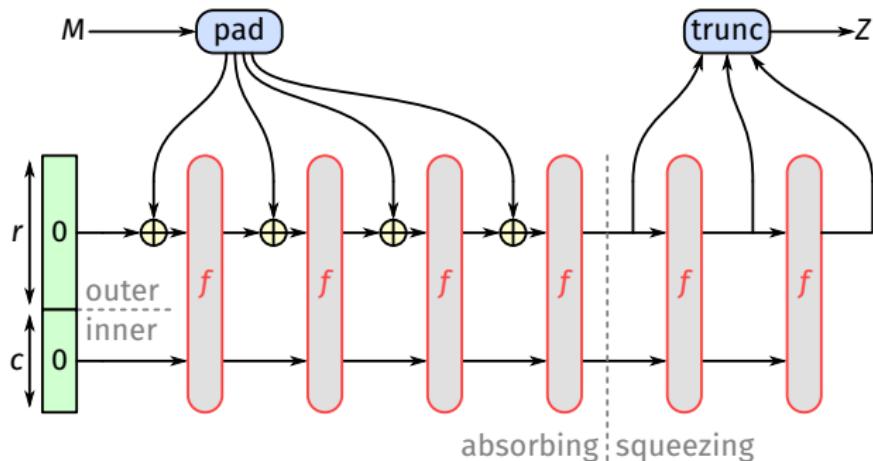
From block ciphers to permutations



From block ciphers to permutations



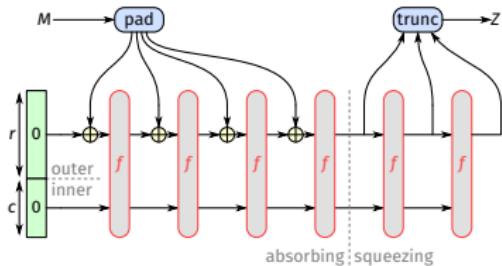
The sponge construction



- Calls a b -bit permutation f , with $b = r + c$
 - r bits of *rate*
 - c bits of *capacity* (security parameter)
- **Sponge function:** concrete instance with a given f, r, c

[Guido Bertoni, Joan Daemen, Michaël Peeters and GVA, 2007]

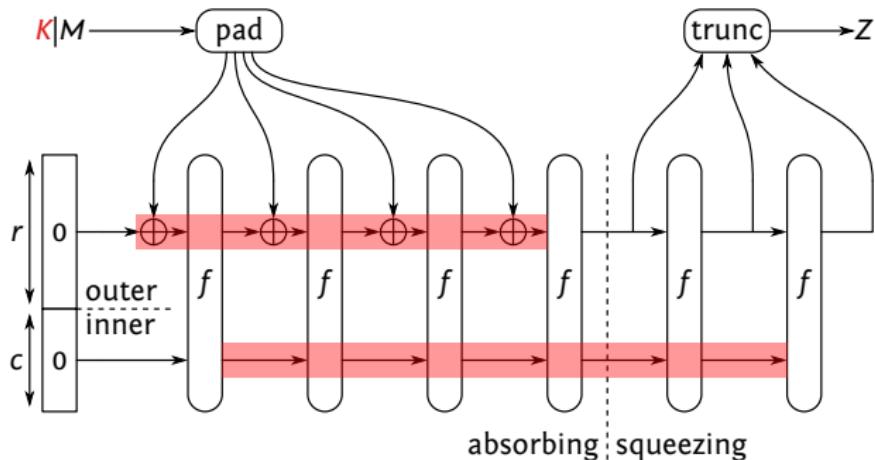
The sponge construction



A sponge function implements a mapping from \mathbb{Z}_2^* to \mathbb{Z}_2^∞ (truncated at an arbitrary length)

- $s \leftarrow 0^b$
- $M \parallel 10^*1$ is cut into r -bit blocks
- For each M_i do (absorbing phase)
 - $s \leftarrow s \oplus (M_i \parallel 0^c)$
 - $s \leftarrow f(s)$
- As long as output is needed do (squeezing phase)
 - Output the first r bits of s
 - $s \leftarrow f(s)$

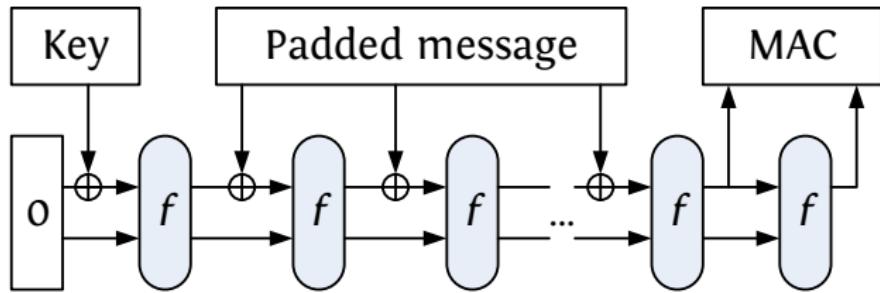
Keyed sponge



$$Z \leftarrow \text{sponge}_K(M) = \text{sponge}(K \| M)$$

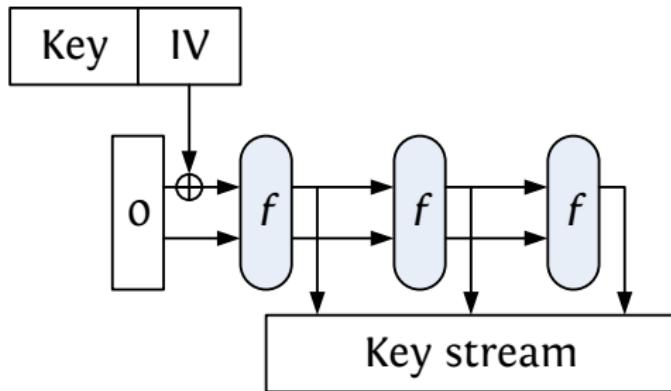
A keyed sponge function implements a mapping from $\mathbb{Z}_2^m \times \mathbb{Z}_2^*$ to \mathbb{Z}_2^∞ (truncated). The input is prefixed with the secret key K .

Keyed sponge for authentication



$\text{sponge}_K(M)$ gives a MAC on M under key K

Keyed sponge as a stream cipher

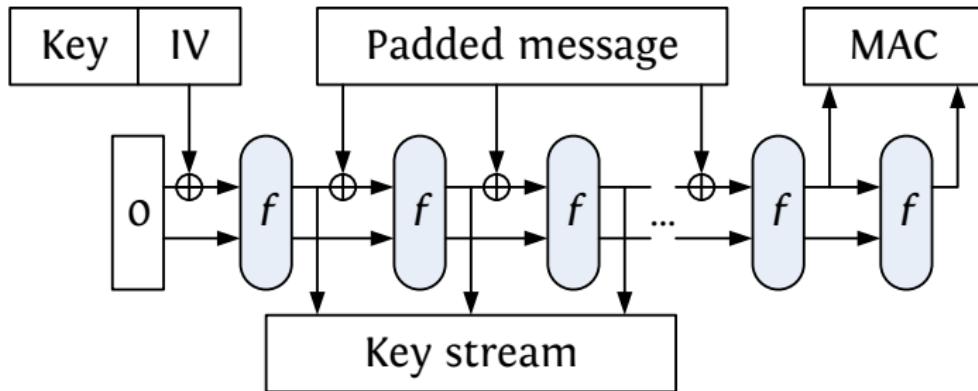


$s_i \leftarrow \text{sponge}_K(d)$ produces keystream from key K and diversifier d

$$c_i = m_i + s_i \quad (\text{in } \mathbb{Z}_2)$$

Note: $\text{IV} = d$

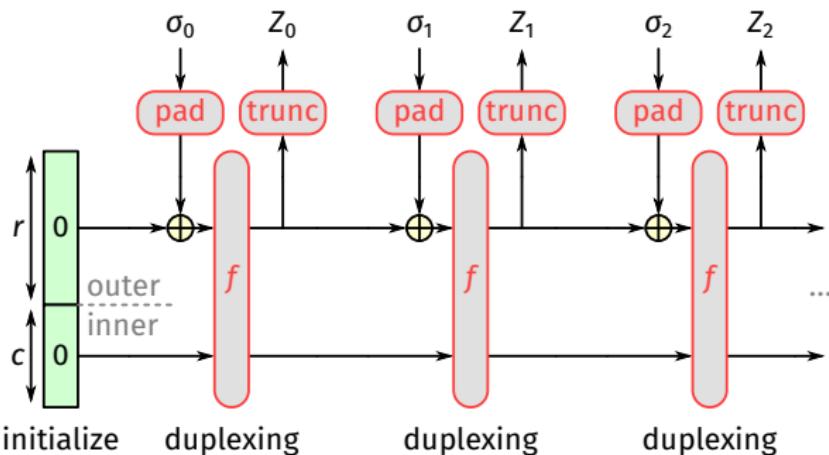
Authenticated encryption: spongeWrap



Variant of the sponge construction:

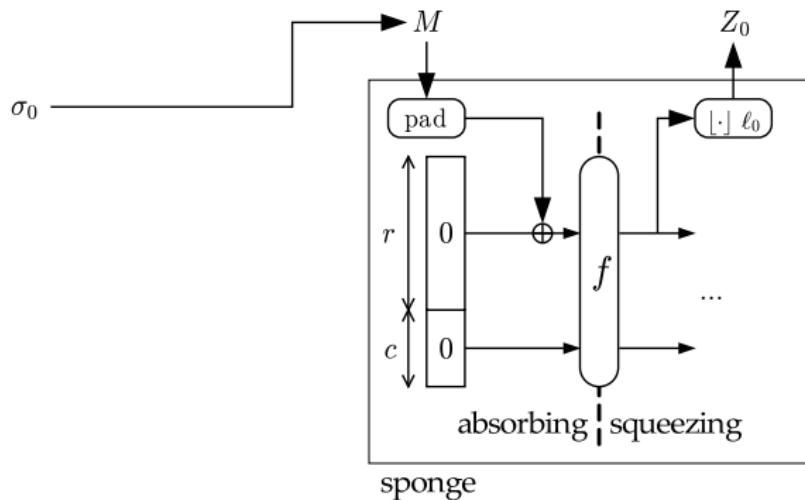
- Keystream: $s_i \leftarrow \text{sponge}_K(d \parallel \text{previous plaintext blocks})$
- MAC: $\text{sponge}_K(d \parallel \text{plaintext})$

The duplex construction



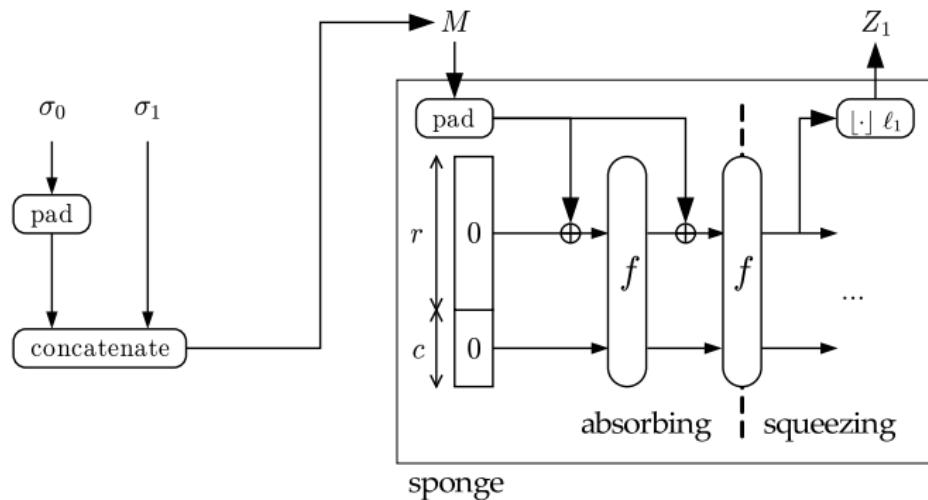
- Object: $D = \text{DUPLEX}[f, \text{pad}, r]$
- Requesting ℓ -bit output $Z = D.\text{duplexing}(\sigma, \ell)$
 - input σ and output Z limited in length
 - Z depends on all previous inputs

Generating duplex responses with a sponge



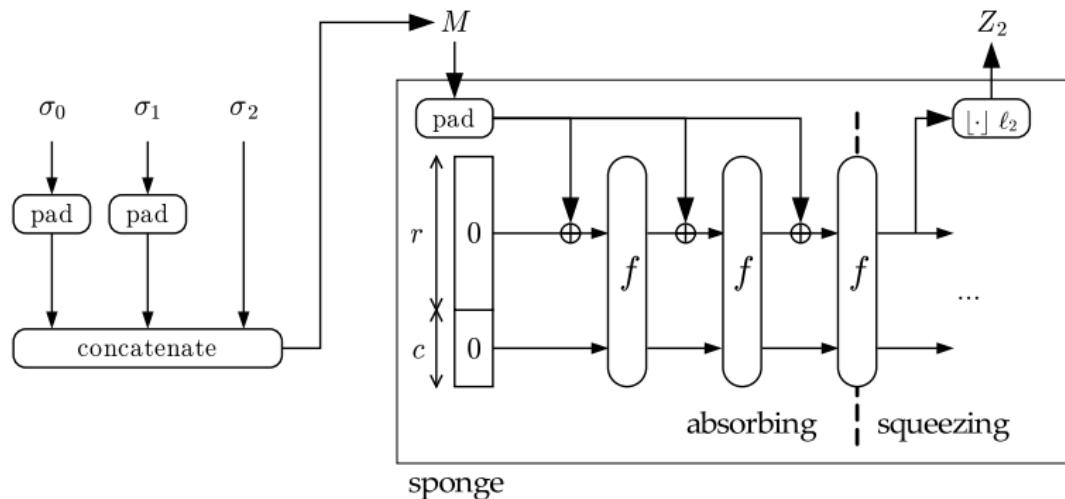
$$Z_0 = \text{sponge}(\sigma_0, \ell_0)$$

Generating duplex responses with a sponge



$$Z_1 = \text{sponge}(\text{pad}(\sigma_0) || \sigma_1, \ell_1)$$

Generating duplex responses with a sponge



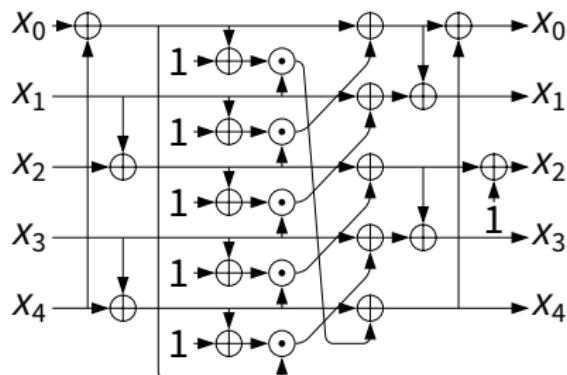
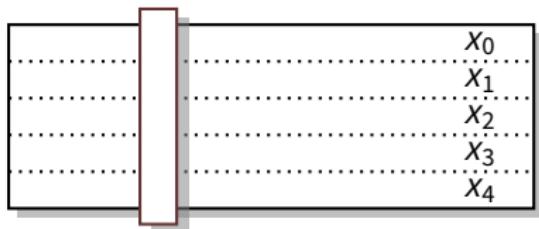
$$Z_2 = \text{sponge}(\text{pad}(\sigma_0) || \text{pad}(\sigma_1) || \sigma_2, \ell_2)$$

Ascon

- Ascon: permutation-based authenticated encryption scheme
- Designed by Christoph Dobraunig, Maria Eichlseder, Florian Mendel and Martin Schläffer in 2014
- One of the algorithms in the CAESAR portfolio

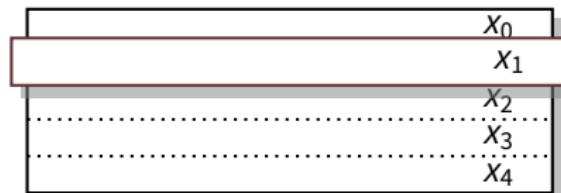
Permutation width: $b = 320 = 5 \times 64$ bits

Ascon: non-linear step



- S-box applied to all 64 columns
- Central: $x_i \leftarrow x_i + (x_{i+1} + 1)x_{i+2}$
 (=KECCAK S-box)
- Linear mixing before and after

Ascon: linear step



$$x_0 \leftarrow x_0 \oplus (x_0 \ggg 19) \oplus (x_0 \ggg 28)$$

$$x_1 \leftarrow x_1 \oplus (x_1 \ggg 61) \oplus (x_1 \ggg 39)$$

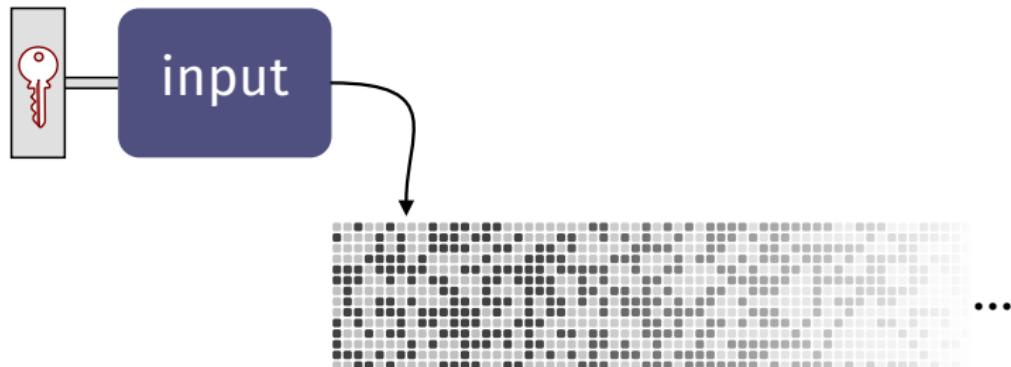
$$x_2 \leftarrow x_2 \oplus (x_2 \ggg 1) \oplus (x_2 \ggg 6)$$

$$x_3 \leftarrow x_3 \oplus (x_3 \ggg 10) \oplus (x_3 \ggg 17)$$

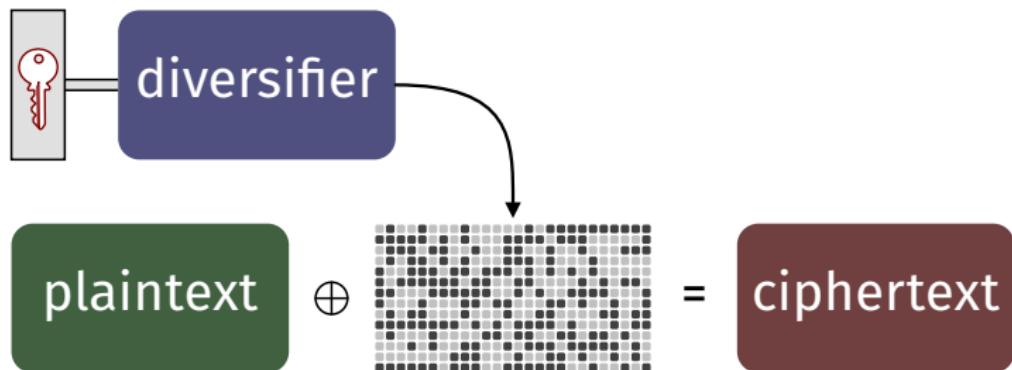
$$x_4 \leftarrow x_4 \oplus (x_4 \ggg 7) \oplus (x_4 \ggg 41)$$

- Intra-word linear diffusion
- Applied on each 64-bit word independently

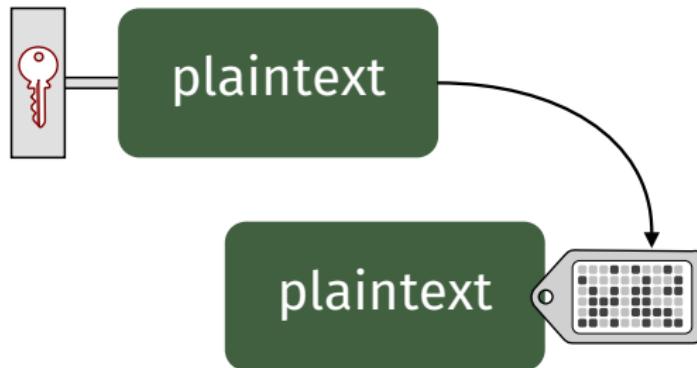
Pseudo-random function (PRF)



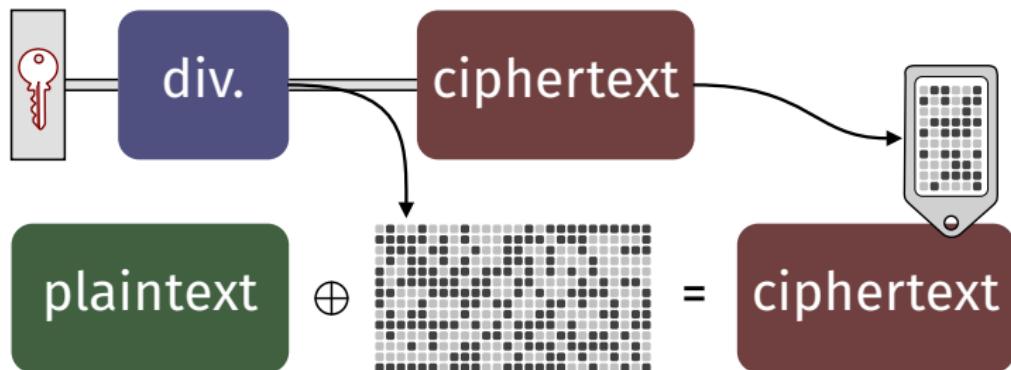
Stream cipher



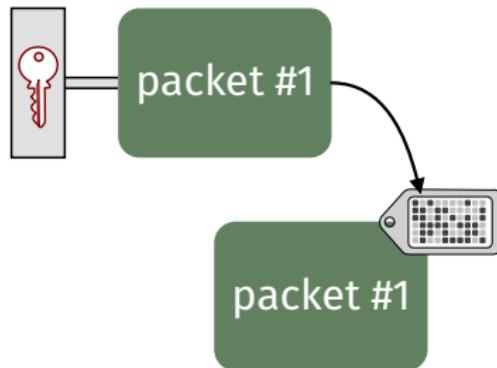
Message authentication code (MAC)



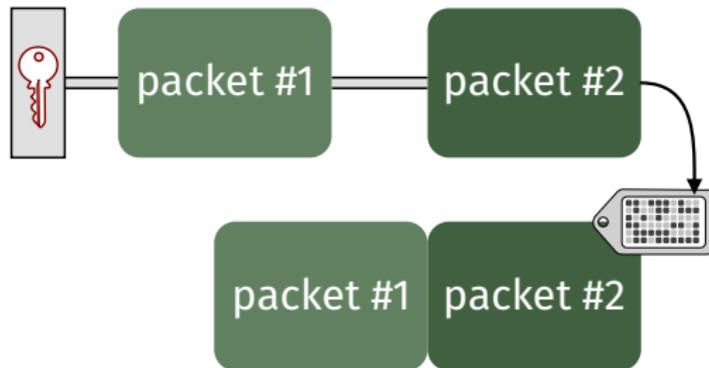
Authenticated encryption



Incremental MACs



Incremental MACs



Incremental MACs

