Cryptology

Jorge L. Villar

FME, UPC, Fall 2024

Outline

1 Stream Ciphers

2 Block Ciphers

3 Block Chaining Modes

4 Key Management

5 Message Authentication Codes

Construction Strategies

- **Stream ciphers:** For arbitrarily long messages (e.g., data streams).
  A key $k \in \mathcal{K}$ and a nonce r are expanded to a **pseudorandom** stream, $g(r, k)$, used as a one-time pad.

$$\text{Enc}(k, m) = (r, m \oplus g(r, k))$$

Construction Strategies

- **Stream ciphers:** For arbitrarily long messages (e.g., data streams).
  A key $k \in \mathcal{K}$ and a nonce r are expanded to a **pseudorandom** stream, $g(r, k)$, used as a one-time pad.

$$\mathrm{Enc}(k, m) = (r, m \oplus g(r, k))$$

- **Block ciphers:** For messages with a fixed length.
  A **pseudorandom** permutation $f_k : \mathcal{M}_\ell \to \mathcal{C}_\ell$, used as a secret key, is applied to the message.

$$\mathrm{Enc}(k, m) = f_k(m)$$

Construction Strategies

- **Stream ciphers:** For arbitrarily long messages (e.g., data streams).
  A key $k \in \mathcal{K}$ and a nonce r are expanded to a **pseudorandom** stream, $g(r, k)$, used as a one-time pad.

$$\mathrm{Enc}(k, m) = (r, m \oplus g(r, k))$$

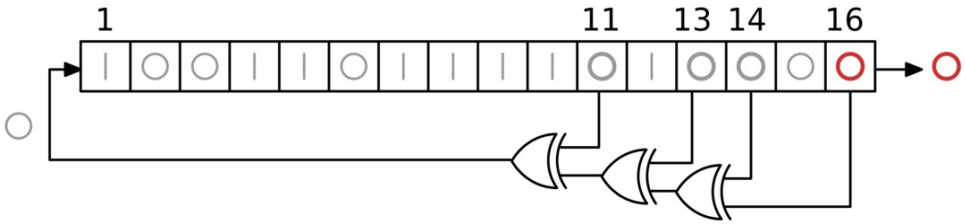
- **Block ciphers:** For messages with a fixed length.
  A **pseudorandom** permutation $f_k : \mathcal{M}_\ell \to \mathcal{C}_\ell$, used as a secret key, is applied to the message.

$$\mathrm{Enc}(k, m) = f_k(m)$$

with a **chaining mode** can encrypt arbitrarily long messages

Examples of stream ciphers

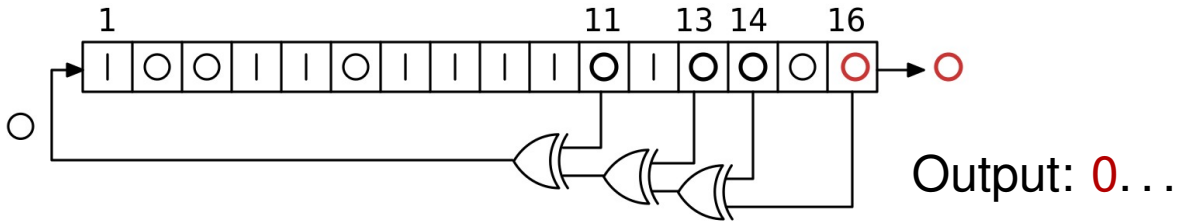
- **Linear Feedback Shift Registers.** Ultra fast but insecure.



(image based on https://commons.wikimedia.org/w/index.php?curid=60225898)

- **Blum-Blum-Shub generator.** Very slow. Provably secure.
- **Non-Linear Feedback Shift Registers.** Very fast. Widely used (GSM, Bluetooth, . . .). In general, they lead to weak symmetric encryption schemes.
- **Other designs:** RC4 (obsolete), chacha, ...

Linear Feedback Shift Registers (LFSR)



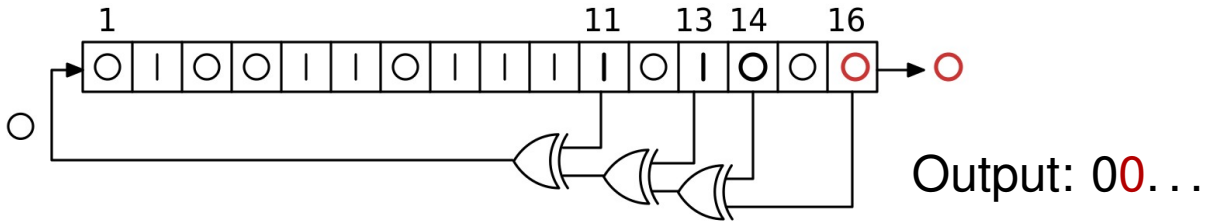
State: m binary cells E.g. $\boldsymbol{x} = (x_1, \ldots, x_{16})$

Initial state: 1001101111010000

Transition function:

- one position right shift
- discarded rightmost cell is the LFSR output
- new value leftmost cell from linear feedback function
  E.g. $f(\boldsymbol{x}) = x_{11} \oplus x_{13} \oplus x_{14} \oplus x_{16}$

Linear Feedback Shift Registers (LFSR)



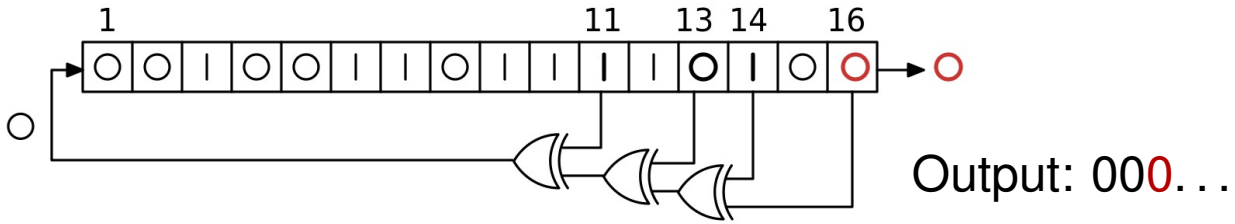
State: m binary cells E.g. $\boldsymbol{x} = (x_1, \ldots, x_{16})$

Initial state: 1001101111010000

Transition function:

- one position right shift
- discarded rightmost cell is the LFSR output
- new value leftmost cell from linear feedback function
  E.g. $f(\boldsymbol{x}) = x_{11} \oplus x_{13} \oplus x_{14} \oplus x_{16}$

Linear Feedback Shift Registers (LFSR)



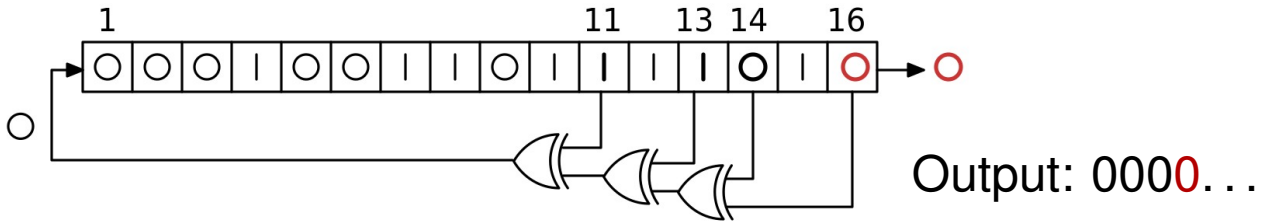
State: m binary cells E.g. $\boldsymbol{x} = (x_1, \ldots, x_{16})$

Initial state: 1001101111010000

Transition function:

- one position right shift
- discarded rightmost cell is the LFSR output
- new value leftmost cell from linear feedback function
  E.g. $f(\boldsymbol{x}) = x_{11} \oplus x_{13} \oplus x_{14} \oplus x_{16}$

Linear Feedback Shift Registers (LFSR)



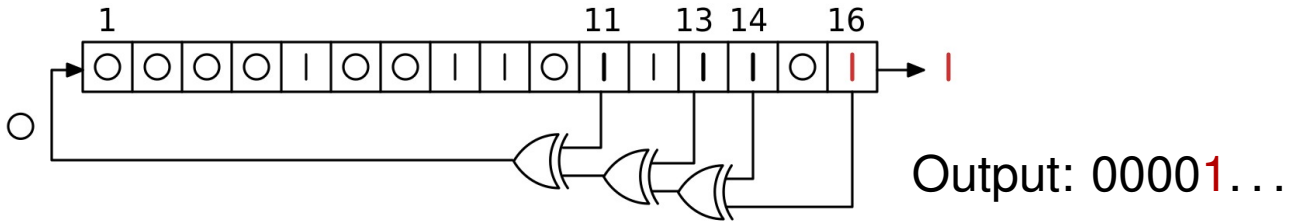
State: m binary cells E.g. $\boldsymbol{x} = (x_1, \ldots, x_{16})$

Initial state: 1001101111010000

Transition function:

- one position right shift
- discarded rightmost cell is the LFSR output
- new value leftmost cell from linear feedback function
  E.g. $f(\boldsymbol{x}) = x_{11} \oplus x_{13} \oplus x_{14} \oplus x_{16}$

Linear Feedback Shift Registers (LFSR)



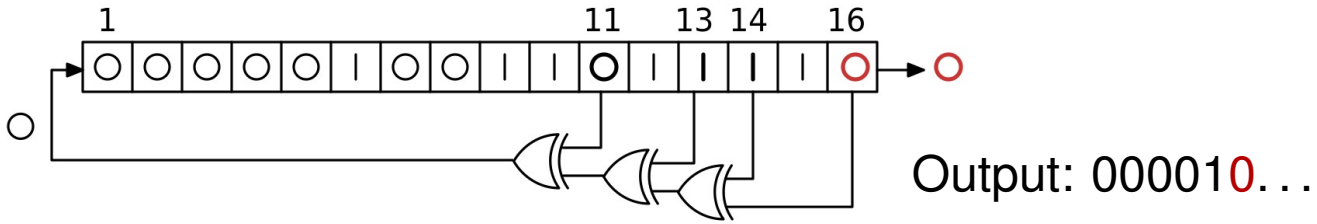
State: m binary cells E.g. $\boldsymbol{x} = (x_1, \ldots, x_{16})$

Initial state: 1001101111010000

Transition function:

- one position right shift
- discarded rightmost cell is the LFSR output
- new value leftmost cell from linear feedback function
  E.g. $f(\boldsymbol{x}) = x_{11} \oplus x_{13} \oplus x_{14} \oplus x_{16}$

Linear Feedback Shift Registers (LFSR)



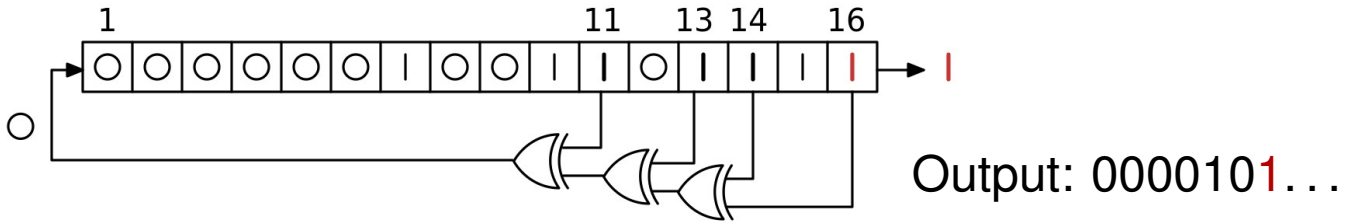
State: m binary cells E.g. $\boldsymbol{x} = (x_1, \ldots, x_{16})$

Initial state: 1001101111010000

Transition function:

- one position right shift
- discarded rightmost cell is the LFSR output
- new value leftmost cell from linear feedback function
  E.g. $f(\boldsymbol{x}) = x_{11} \oplus x_{13} \oplus x_{14} \oplus x_{16}$

Linear Feedback Shift Registers (LFSR)



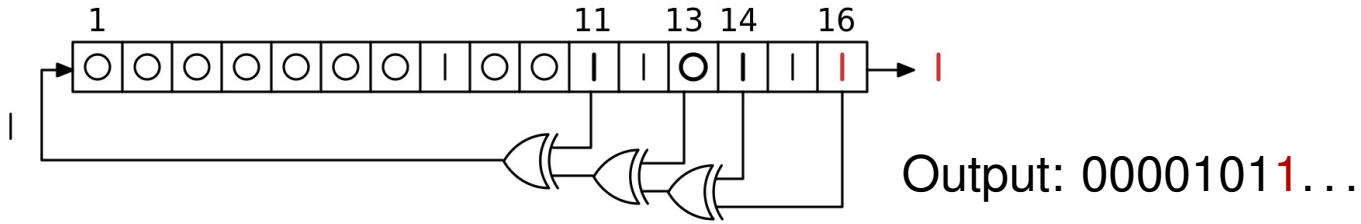
State: m binary cells E.g. $\boldsymbol{x} = (x_1, \ldots, x_{16})$

Initial state: 1001101111010000

Transition function:

- one position right shift
- discarded rightmost cell is the LFSR output
- new value leftmost cell from linear feedback function
  E.g. $f(\boldsymbol{x}) = x_{11} \oplus x_{13} \oplus x_{14} \oplus x_{16}$

Linear Feedback Shift Registers (LFSR)



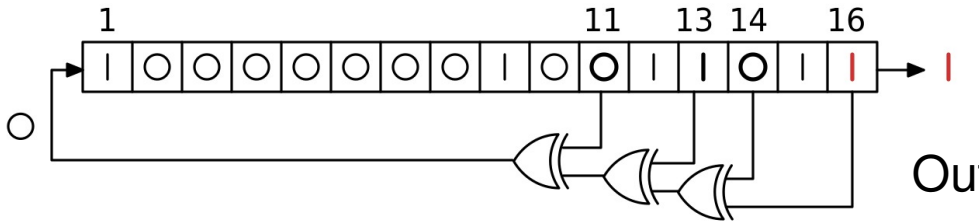
State: m binary cells E.g. $\boldsymbol{x} = (x_1, \ldots, x_{16})$

Initial state: 1001101111010000

Transition function:

- one position right shift
- discarded rightmost cell is the LFSR output
- new value leftmost cell from linear feedback function
  E.g. $f(\boldsymbol{x}) = x_{11} \oplus x_{13} \oplus x_{14} \oplus x_{16}$

Linear Feedback Shift Registers (LFSR)



Output: 000010111...

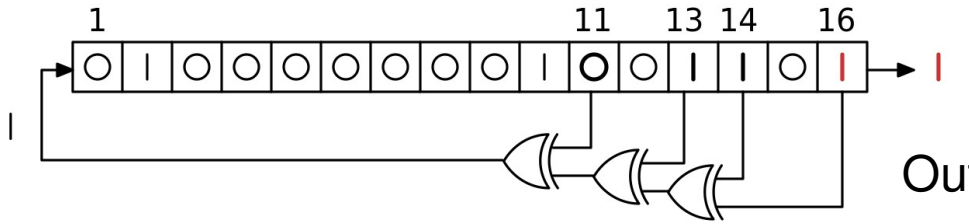
State: m binary cells E.g. $\boldsymbol{x} = (x_1, \ldots, x_{16})$

Initial state: 1001101111010000

Transition function:

- one position right shift
- discarded rightmost cell is the LFSR output
- new value leftmost cell from linear feedback function
  E.g. $f(\boldsymbol{x}) = x_{11} \oplus x_{13} \oplus x_{14} \oplus x_{16}$

Linear Feedback Shift Registers (LFSR)



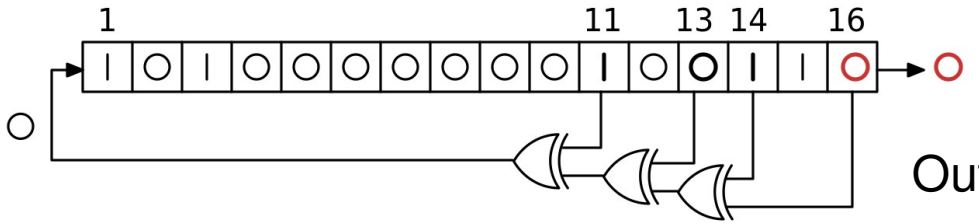
State: m binary cells E.g. $\boldsymbol{x} = (x_1, \ldots, x_{16})$

Initial state: 1001101111010000

Transition function:

- one position right shift
- discarded rightmost cell is the LFSR output
- new value leftmost cell from linear feedback function
  E.g. $f(\boldsymbol{x}) = x_{11} \oplus x_{13} \oplus x_{14} \oplus x_{16}$

Linear Feedback Shift Registers (LFSR)



Output: 000010111110...

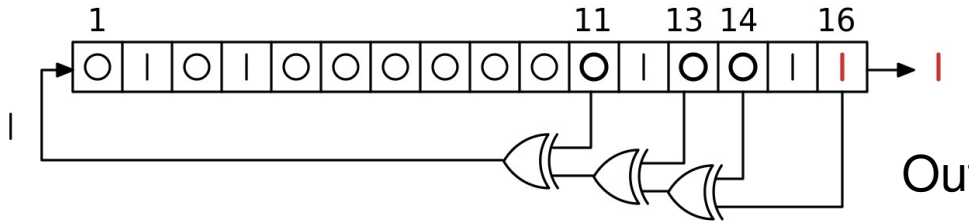
State: m binary cells E.g. $\boldsymbol{x} = (x_1, \ldots, x_{16})$

Initial state: 1001101111010000

Transition function:

- one position right shift
- discarded rightmost cell is the LFSR output
- new value leftmost cell from linear feedback function
  E.g. $f(\boldsymbol{x}) = x_{11} \oplus x_{13} \oplus x_{14} \oplus x_{16}$

Linear Feedback Shift Registers (LFSR)



Output: 000010111101...

State: m binary cells E.g. $\boldsymbol{x} = (x_1, \ldots, x_{16})$

Initial state: 1001101111010000

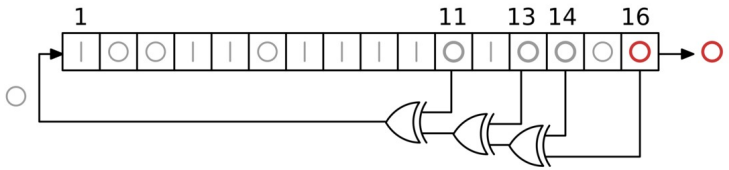
Transition function:

- one position right shift
- discarded rightmost cell is the LFSR output
- new value leftmost cell from linear feedback function
  E.g. $f(\boldsymbol{x}) = x_{11} \oplus x_{13} \oplus x_{14} \oplus x_{16}$

Output equation: $y_{n-16} \oplus y_{n-14} \oplus y_{n-13} \oplus y_{n-11} \oplus y_n = 0$

LFSR (II)

$$y_0 = x_{16}$$
$$y_1 = x_{15}$$
$$\vdots$$
$$y_{15} = x_1$$



$$y_{16} = y_0 \oplus y_2 \oplus y_3 \oplus y_5 \quad \Rightarrow \quad y_0 \oplus y_2 \oplus y_3 \oplus y_5 \oplus y_{16} = 0$$
$$y_{17} = y_1 \oplus y_3 \oplus y_4 \oplus y_6 \quad \Rightarrow \quad y_1 \oplus y_3 \oplus y_4 \oplus y_6 \oplus y_{17} = 0$$
$$y_{18} = y_2 \oplus y_4 \oplus y_5 \oplus y_7 \quad \Rightarrow \quad y_2 \oplus y_4 \oplus y_5 \oplus y_7 \oplus y_{18} = 0$$
$$\vdots$$

Characteristic polynomial: $P(z) = \underbrace{z^{16} + z^{14} + z^{13} + z^{11}}_{\text{LFSR wires}} + 1$

In general, in $\mathbb{F}_2[z]$:

$$\deg(P(z)(\underbrace{y_0 + y_1 z + y_2 z^2 + \ldots}_{Y(z)})) < \deg(P(z)) = m$$

LFSR (II)

Proposition

The output sequence of an LFSR is ultimately periodic.

LFSR (II)

Proposition

The output sequence of an LFSR is ultimately periodic.

Proposition

If $P(z)$ is a primitive polynomial then for any non-zero initial state the output sequence has the maximal period $2^m - 1$.

LFSR (II)

Proposition

The output sequence of an LFSR is ultimately periodic.

Proposition

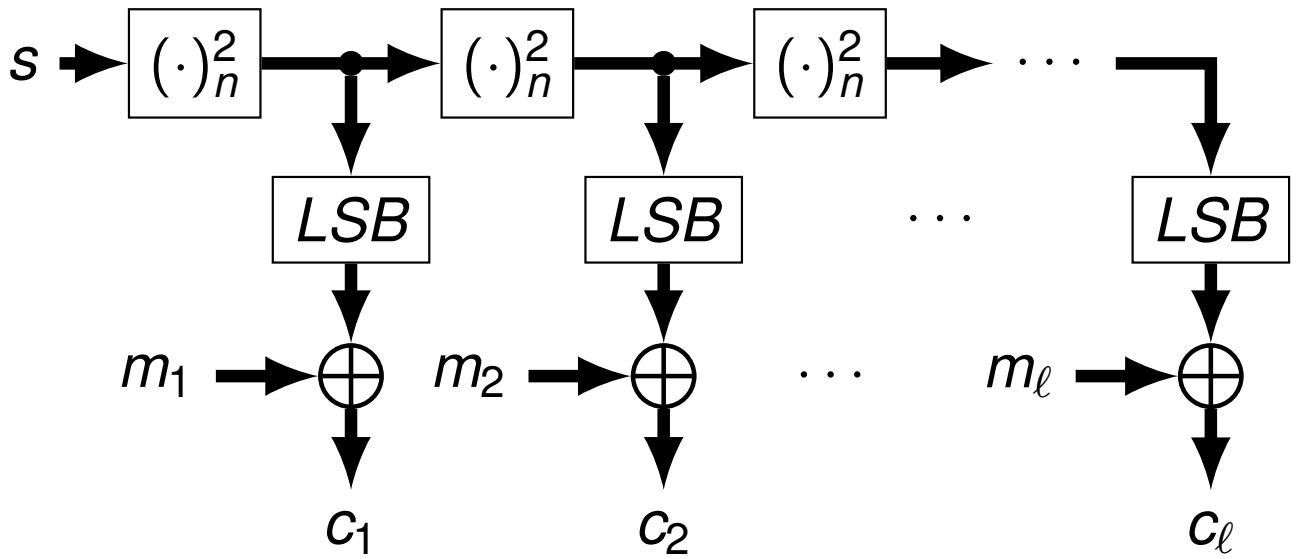
If $P(z)$ is a primitive polynomial then for any non-zero initial state the output sequence has the maximal period $2^m - 1$.

Proposition

From any output subsequence of length $2m$ the structure of a LFSR of size $\leq m$ producing the sequence can be efficiently determined.

LFSR do not yield secure stream ciphers!

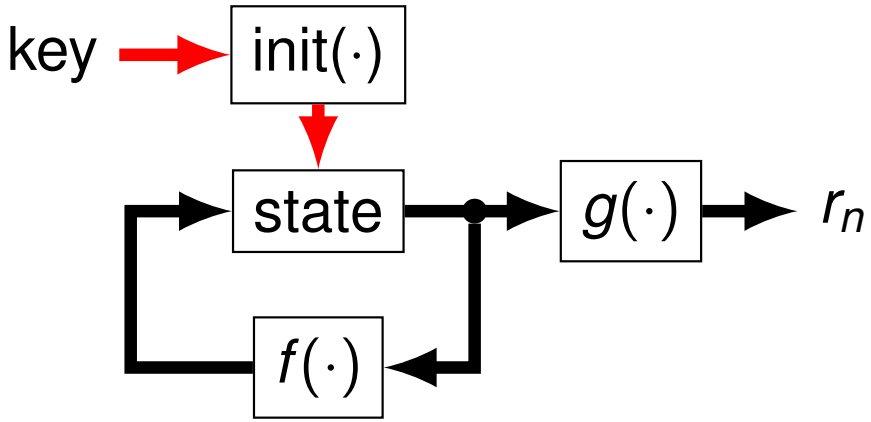
Stream Cipher from Blum-Blum-Shub Generator



$n = pq$, $p = 2p' + 1$, $q = 2q' + 1$
p, q, p', q' primes (512 to 1024 bits).

RC4 Stream Cipher

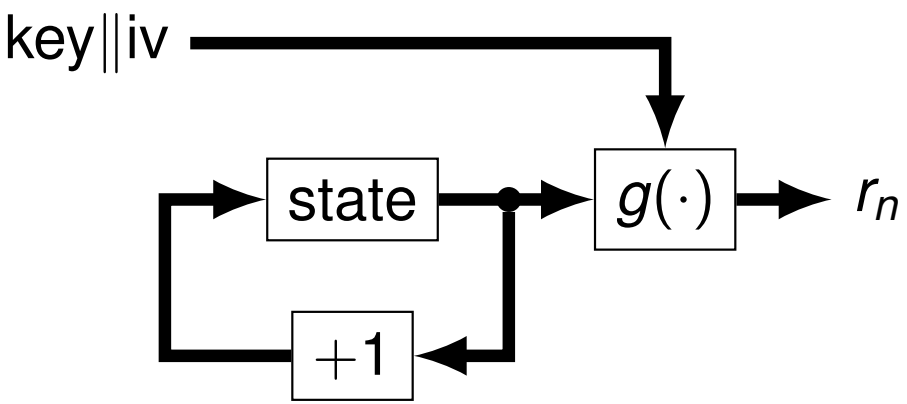
- Internal state: a permutation s of $\{0, \ldots, 255\}$ and two counters i, j.



- initial state: computed from the key.
- Every message byte m_n is XORed with the output byte r_n.
- The key is too short (from 40 bits to 128 bits).
- No specified way to use an initialization value (iv).
- Some real systems using RC4 have been attacked.

Chacha Stream Cipher

- Internal state: just a counter i.
- It works with 512 bit words.



- initial state: $i = 0$.
- Every message word m_n is XORed with the output word r_n.

- key has 256 bits and the iv has 64 bits.
- fully parallelizable (random access to words)
- No successful attack is currently known.

Outline

1 Stream Ciphers

2 Block Ciphers

3 Block Chaining Modes

4 Key Management

5 Message Authentication Codes

Practical Construction of Block Ciphers

Heuristic constructions based on:

- **Confusion:** every bit in the ciphertext must depend on several bits of the key.
- **Diffusion:** flipping a single plaintext bit must change half of the ciphertext bits.
- **Iteration:** the encryption procedure consists of a given number of rounds.

Structure:

- **Key expansion:** a set of round keys is generated from the original key
- **Mixing** of permutation (shuffling, shifting. . .), substitution (S-boxes) and XORing the subkeys in each round.

Example: DES

Key size: 56 bits

Message and ciphertext block size: 64 bits

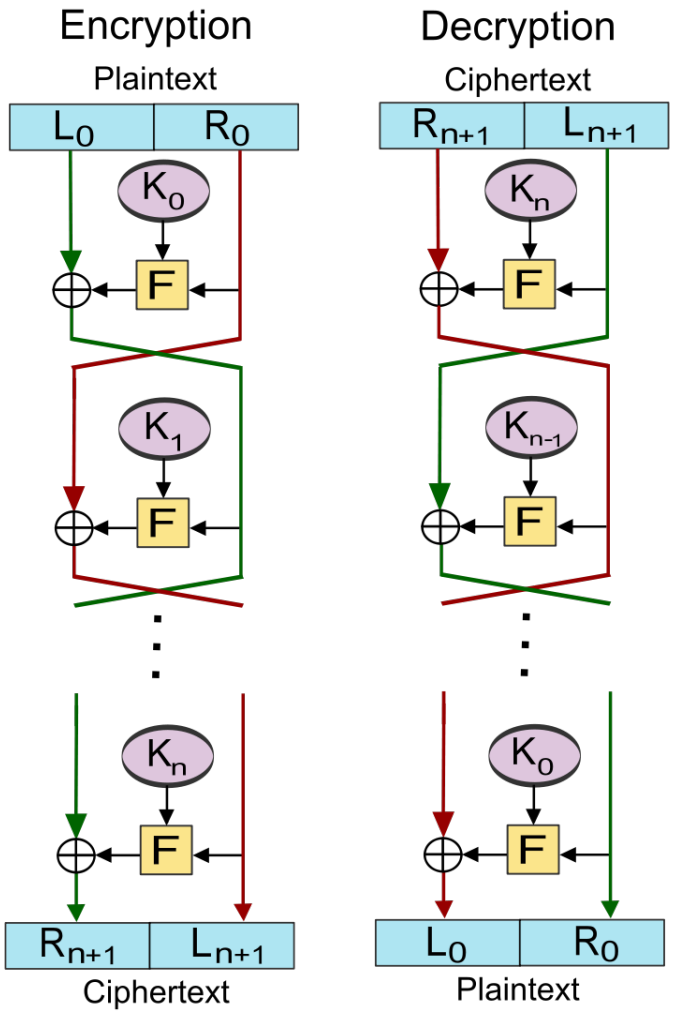
Number of encryption rounds: 16

Key expansion: from 56 bits to 768 bits (16 subkeys, each one of 48 bits)

Encryption rounds: Feistel Network

- Start with the 64 bit message block
- In each iteration, divide the intermediate block into two halves (L_i, R_i)
- Apply a transformation to compute next block $(L_{i+1}, R_{i+1}) = (R_i, L_i \oplus F(k_i, R_i))$
- The resulting ciphertext block is (R_{16}, L_{16}).

Feistel Network



Example: DES (II)

The function F combines:

- Expands the 32 bit half-block into 48 bits, and XORs the round subkey.
- Divides the 48 bits into 8 6-bit words.
- Transforms each word into a 4 bit word with a table (S-box).
- Glues the 4 bit words to obtain a 32 bit half-block.
- Applies a fixed prermutation to the 32 bits.

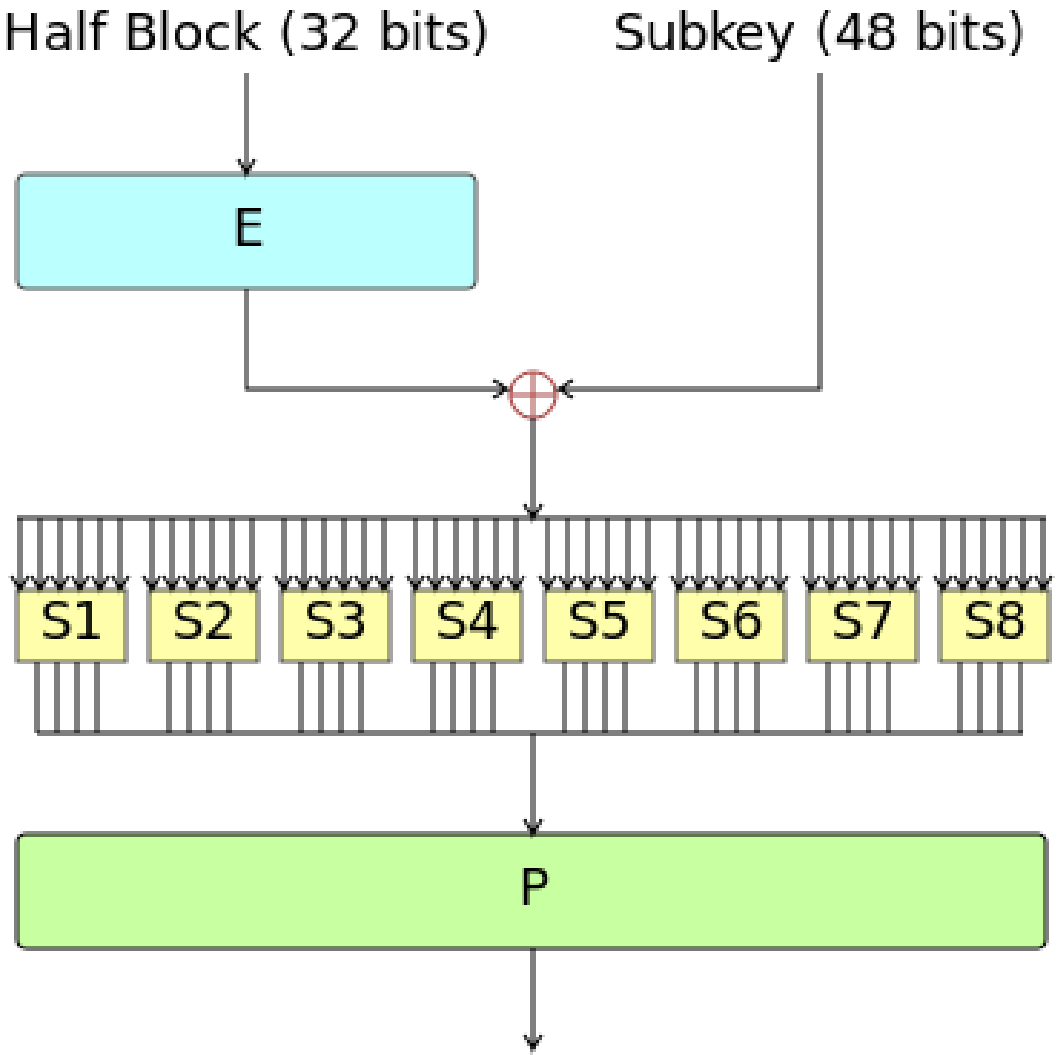
3DES combines three independent keys (168 bits in total) and it is still considered secure.

$3DES_{k_1,k_2,k_3}(m) = DES_{k_3} \circ DES_{k_2}^{-1} \circ DES_{k_1}(m)$

More details in

http://en.wikipedia.org/wiki/Data_Encryption_Standard

Example: DES Round Function



Example: AES256

Key size: 256 bits

Message and ciphertext block size: 128 bits (4×4 byte matrix)

Number of encryption rounds: 14

Key expansion: from 256 bits to 1920 bits (15 subkeys, each one a 4×4 matrix of bytes) in an iterative procedure (it adds 4 new bytes in each iteration)

Encryption round:

- Apply a substitution box to each byte
- Perform rotation operations to rows
- Perform linear transformation to columns
- XOR with the round key

More details in
`http://en.wikipedia.org/wiki/Advanced_Encryption_Standard`

Outline

1 Stream Ciphers

2 Block Ciphers

3 Block Chaining Modes

4 Key Management

5 Message Authentication Codes

Block Chaining Modes

A strategy to encrypt messages larger than one block:

Block Chaining Modes

A strategy to encrypt messages larger than one block: Partition $m \in \{0,1\}^{n\ell}$ into n blocks $m_1, \ldots, m_n \in \{0,1\}^{\ell}$ and encrypt each block separately.

Block Chaining Modes

A strategy to encrypt messages larger than one block: Partition $m \in \{0,1\}^{n\ell}$ into n blocks $m_1, \ldots, m_n \in \{0,1\}^{\ell}$ and encrypt each block separately.

ECB (Electronic Codebook):

$c_i = \mathrm{Enc}(k, m_i)$

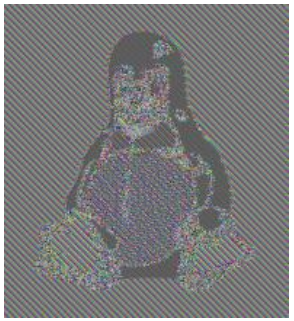
(from Wikipedia)

Block Chaining Modes

A strategy to encrypt messages larger than one block: Partition $m \in \{0,1\}^{n\ell}$ into n blocks $m_1, \ldots, m_n \in \{0,1\}^{\ell}$ and encrypt each block separately.

ECB (Electronic Codebook):

$c_i = \mathrm{Enc}(k, m_i)$



(from Wikipedia)

Equal message blocks result in equal ciphertext blocks!

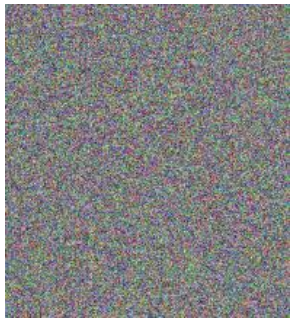
Block Chaining Modes

A strategy to encrypt messages larger than one block: Partition $m \in \{0,1\}^{n\ell}$ into n blocks $m_1, \ldots, m_n \in \{0,1\}^{\ell}$ and encrypt each block separately.

CBC (Cipher Block Chaining):

$c_0 = iv$

$c_i = \mathrm{Enc}(k, m_i \oplus c_{i-1})$



(from Wikipedia)

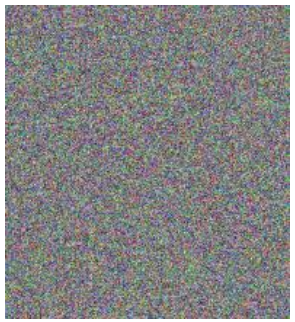
Block Chaining Modes

A strategy to encrypt messages larger than one block: Partition $m \in \{0,1\}^{n\ell}$ into n blocks $m_1, \ldots, m_n \in \{0,1\}^{\ell}$ and encrypt each block separately.

CBC (Cipher Block Chaining):

$c_0 = iv$

$c_i = \mathrm{Enc}(k, m_i \oplus c_{i-1})$

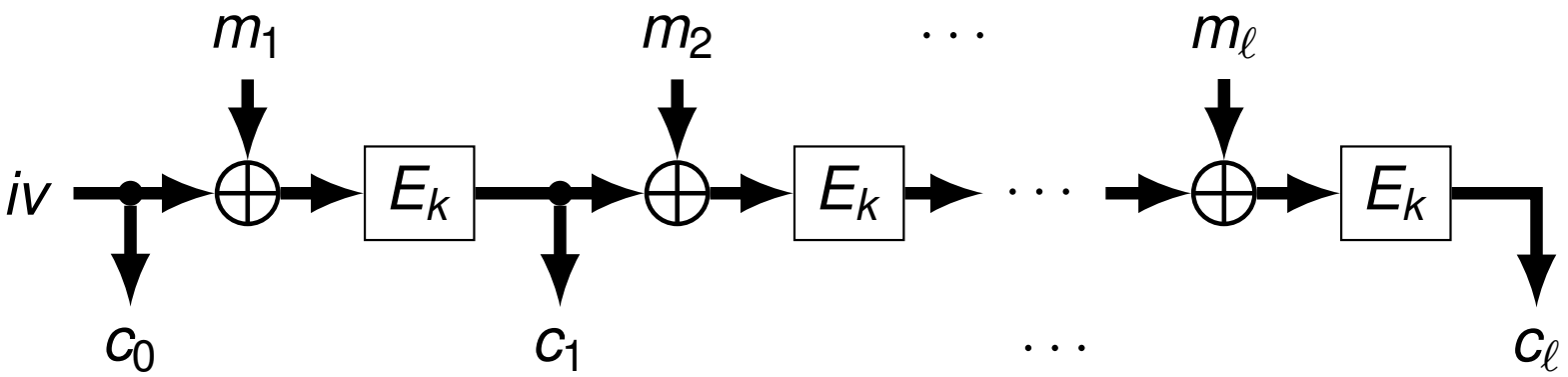


(from Wikipedia)

The ciphertext has an extra random block c_0, the "initialization vector"

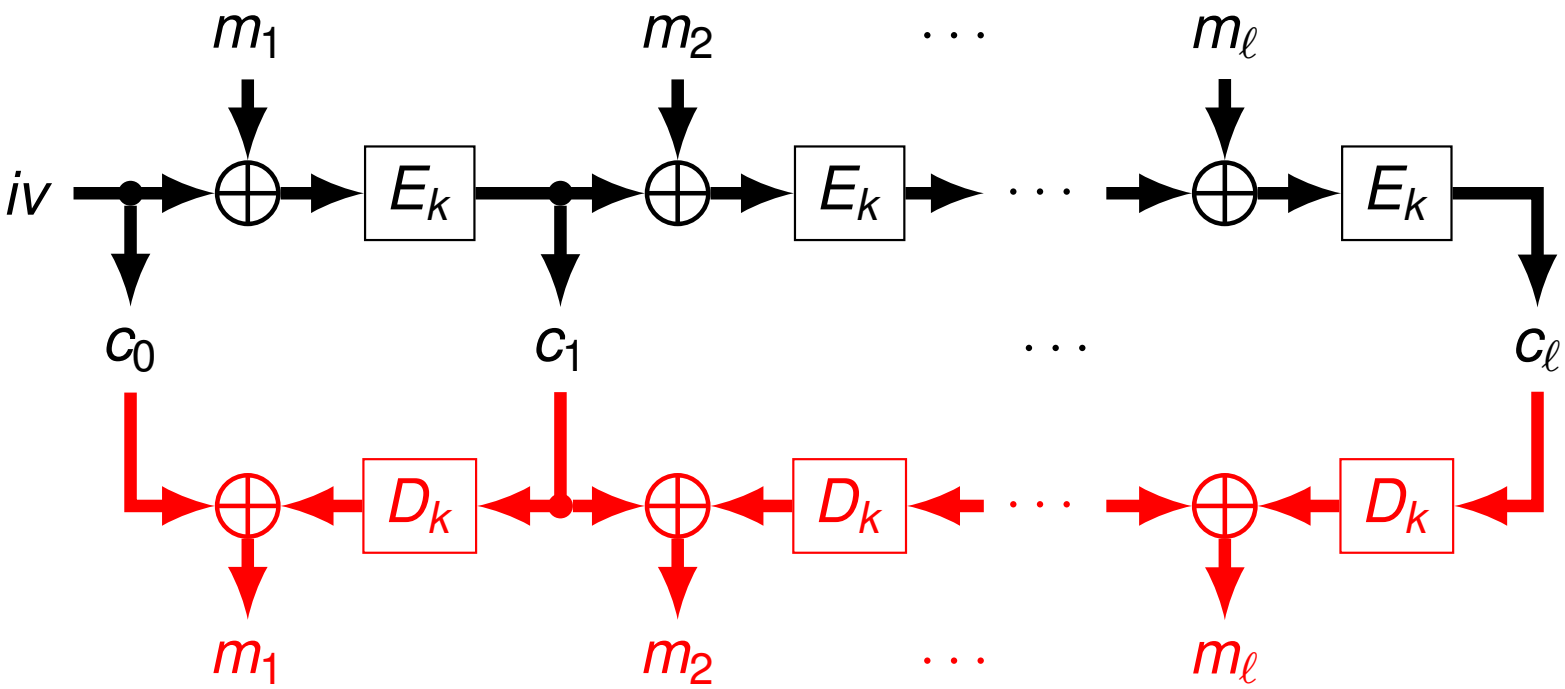
CBC Mode

Comp. . .



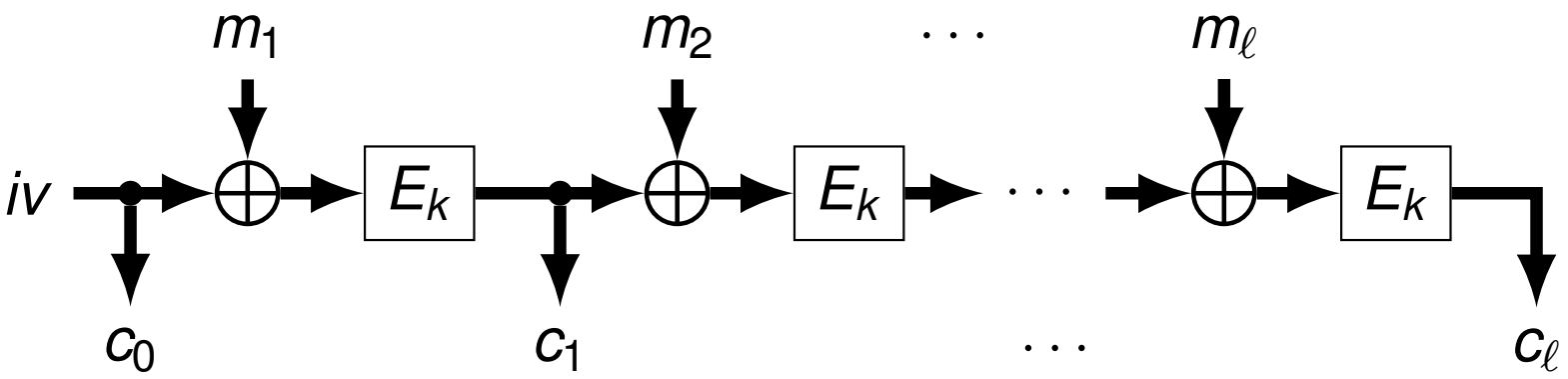
CBC Mode

Comp. . .



CBC Mode

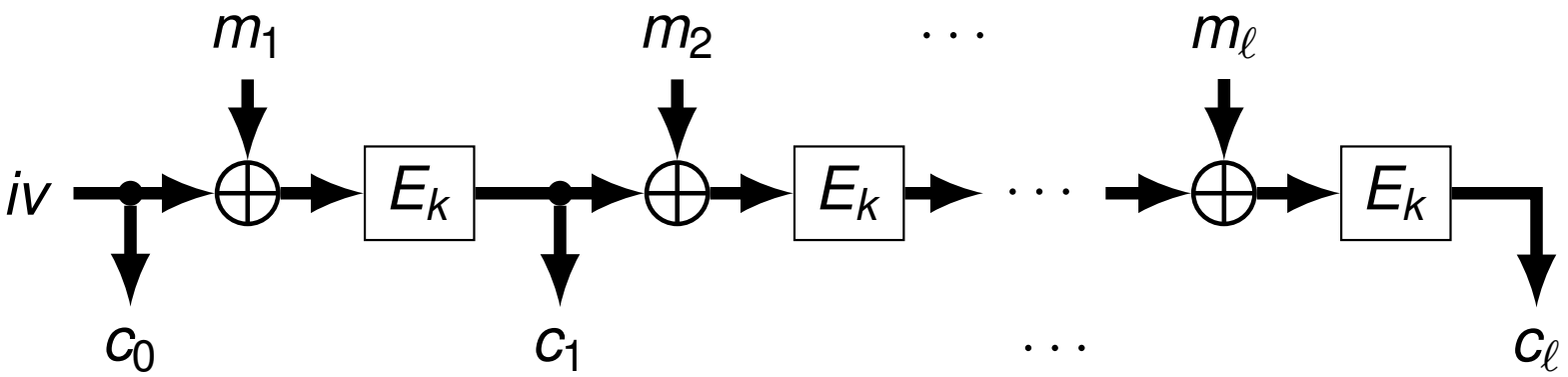
Comp...



$$c_i = c_j \quad \Rightarrow \quad m_i \oplus c_{i-1} = m_j \oplus c_{j-1} \quad \Rightarrow \quad m_i \oplus m_j \text{ revealed!}$$

CBC Mode

Comp...



$c_i = c_j \quad \Rightarrow \quad m_i \oplus c_{i-1} = m_j \oplus c_{j-1} \quad \Rightarrow \quad m_i \oplus m_j$ revealed!

Collision probability must be small

Other Block Chaining Modes

- CFB (Cipher Feedback) operation mode:

  $c_0 = iv$

  $c_i = \text{Enc}(k, c_{i-1}) \oplus m_i$

- OFB (Output Feedback) operation mode:

  $r_0 = iv$

  $r_i = \text{Enc}(k, r_{i-1})$

  $c_i = m_i \oplus r_i$

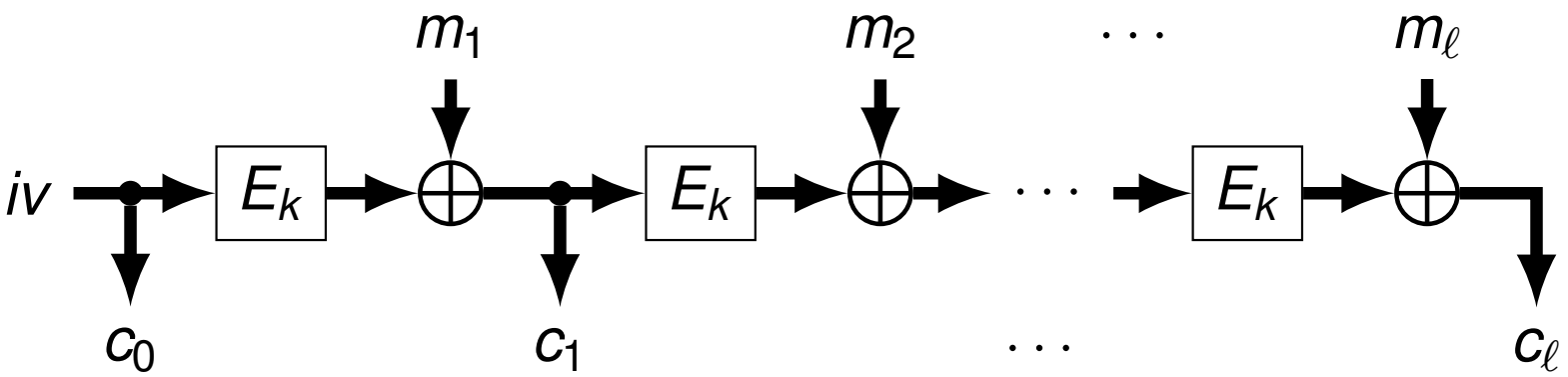
- CTR (Counter) operation mode:

  $r_i = \text{Enc}(k, iv + i)$

  $c_i = m_i \oplus r_i$

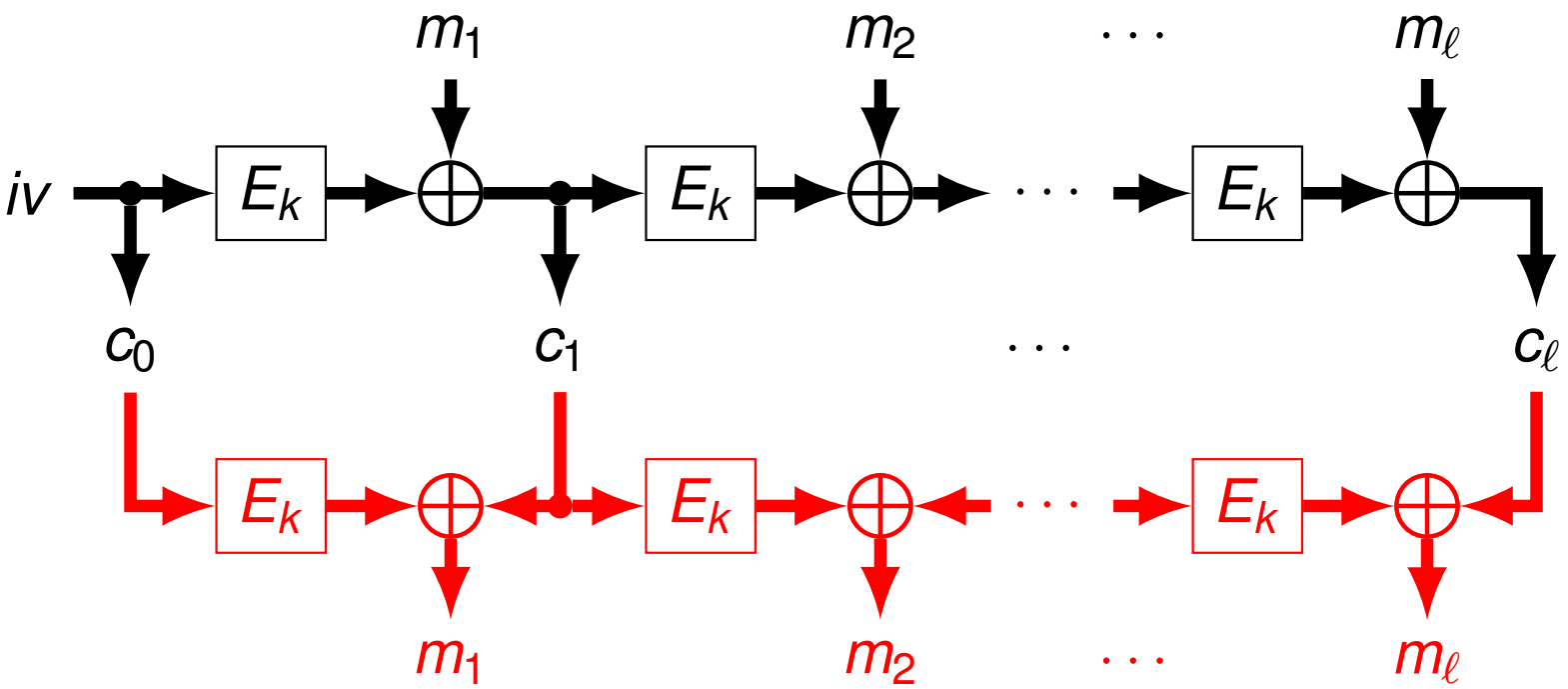
CFB Mode

Comp...



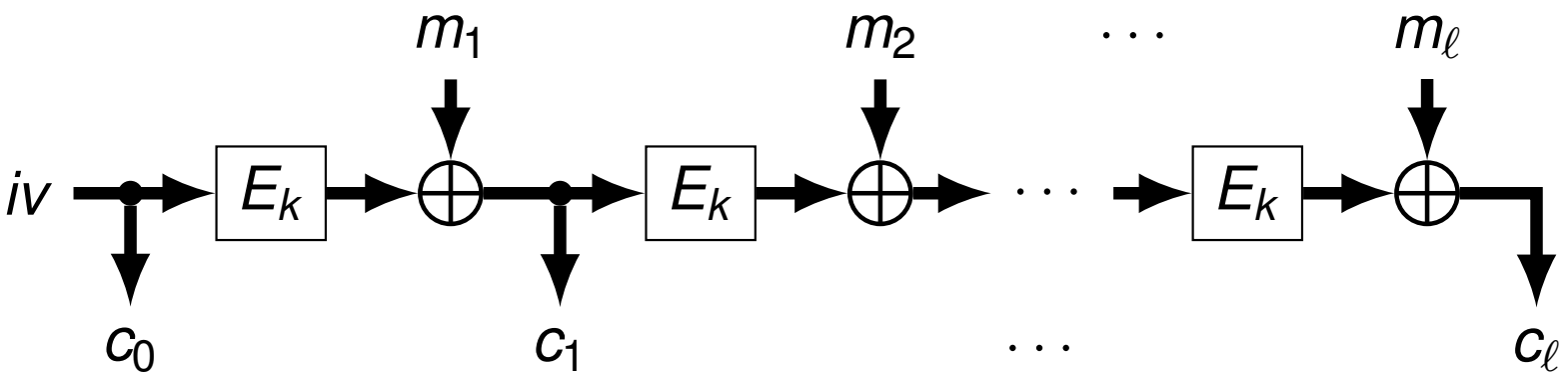
CFB Mode

Comp. . .



CFB Mode

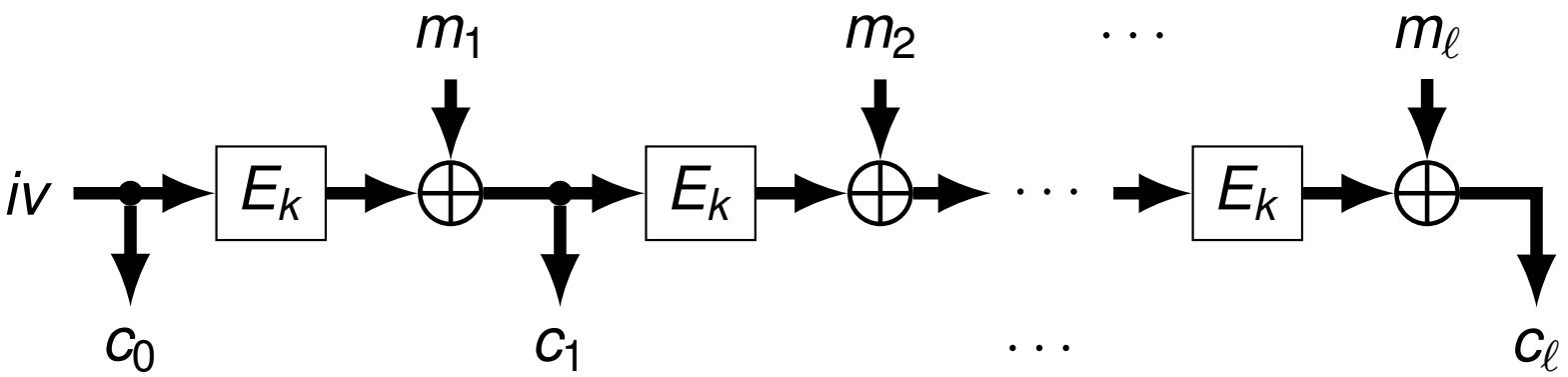
Comp...



$$c_{i-1} = c_{j-1} \quad \Rightarrow \quad m_i \oplus c_i = m_j \oplus c_j \quad \Rightarrow \quad m_i \oplus m_j \text{ revealed!}$$

CFB Mode

Comp...

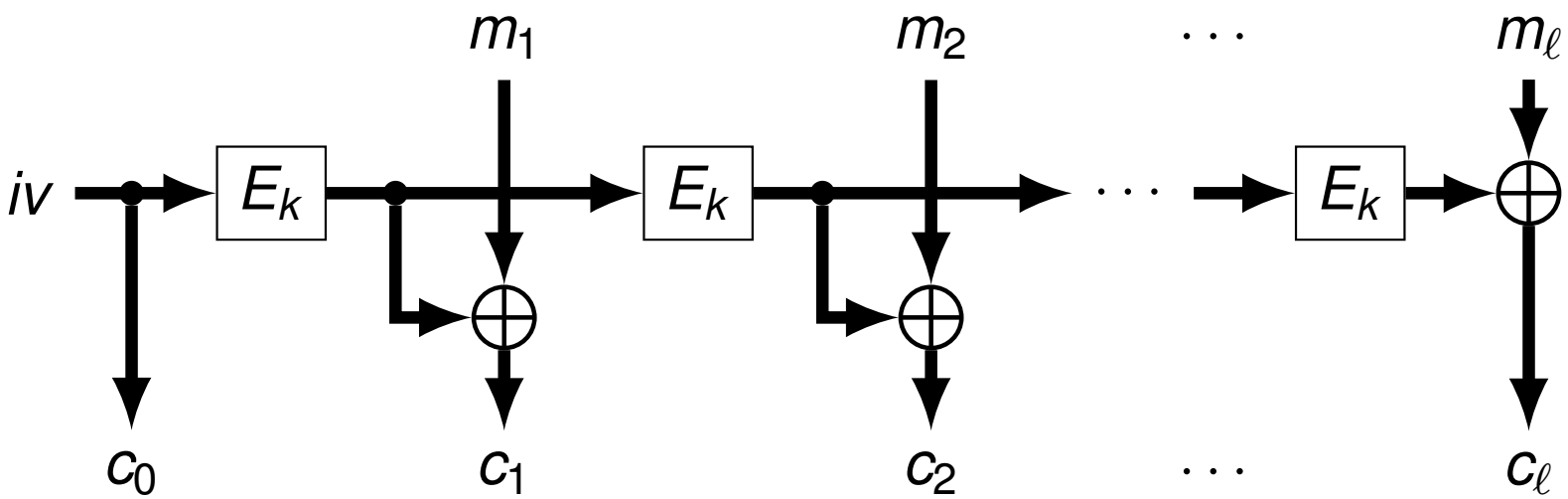


$c_{i-1} = c_{j-1} \quad \Rightarrow \quad m_i \oplus c_i = m_j \oplus c_j \quad \Rightarrow \quad m_i \oplus m_j$ revealed!

Collision probability must be small

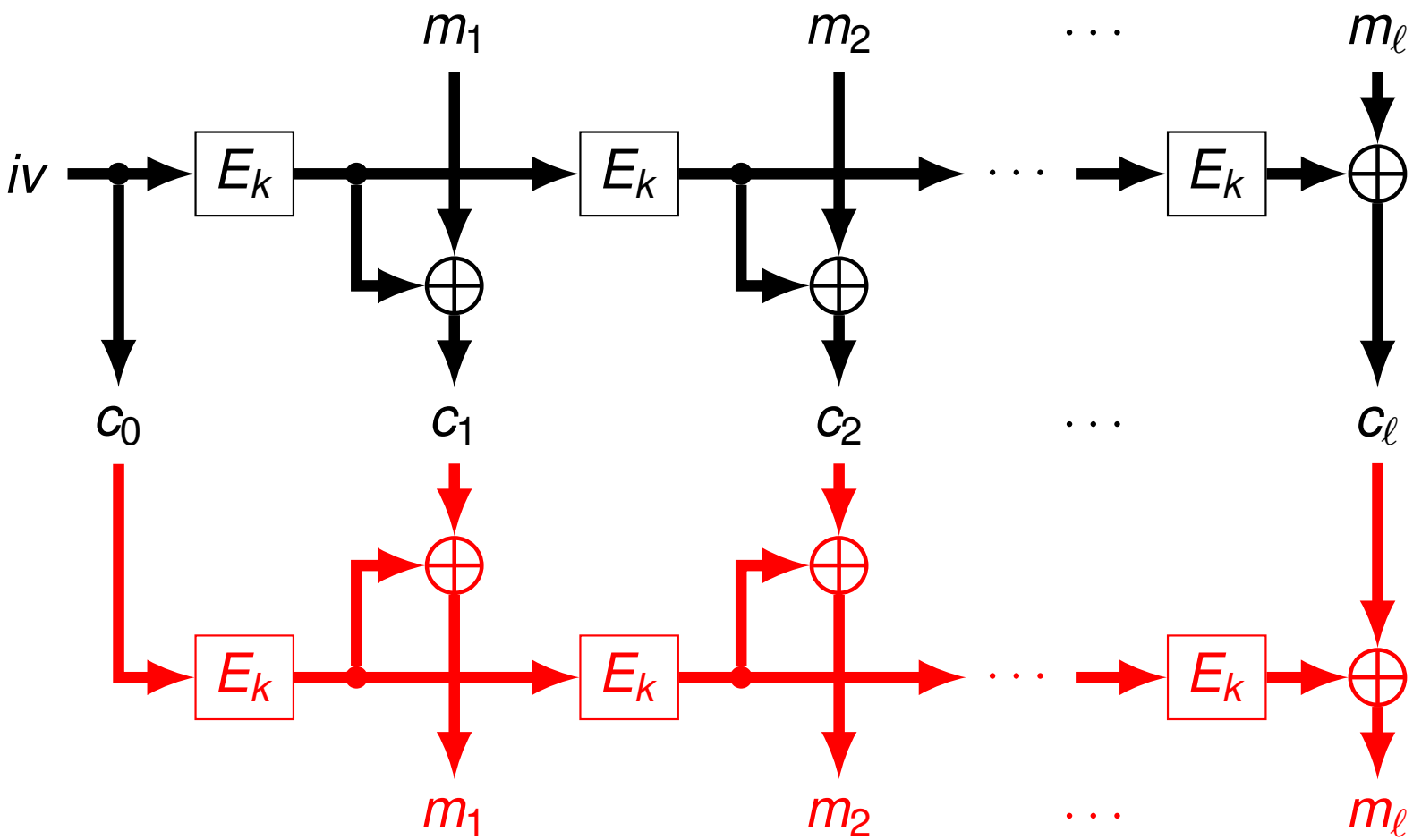
OFB Mode

Comp...



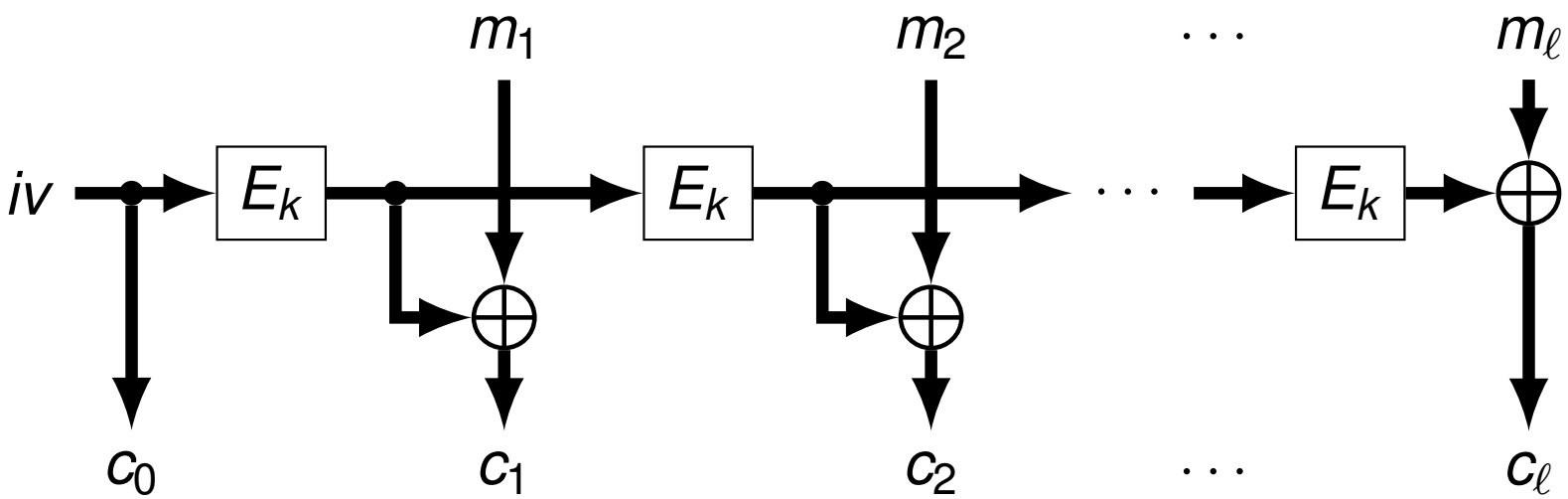
OFB Mode

Comp...



OFB Mode

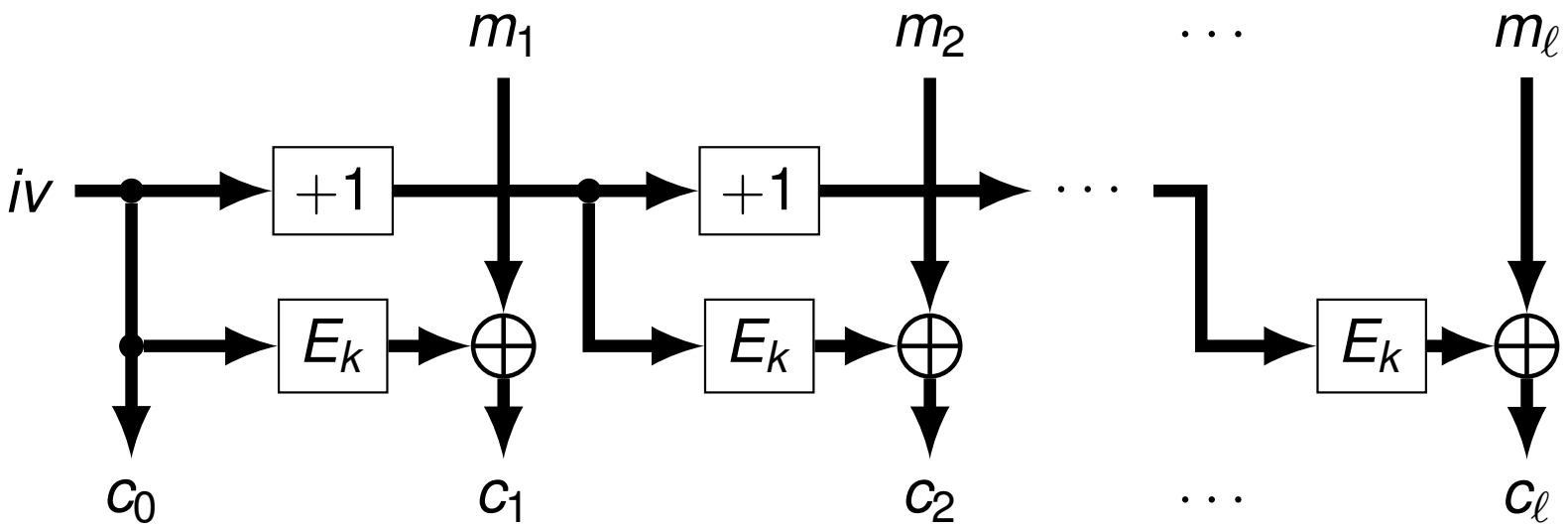
Comp...



A short cycle of E_k of length r reveals $m_i \oplus m_{i+r}$ for all i

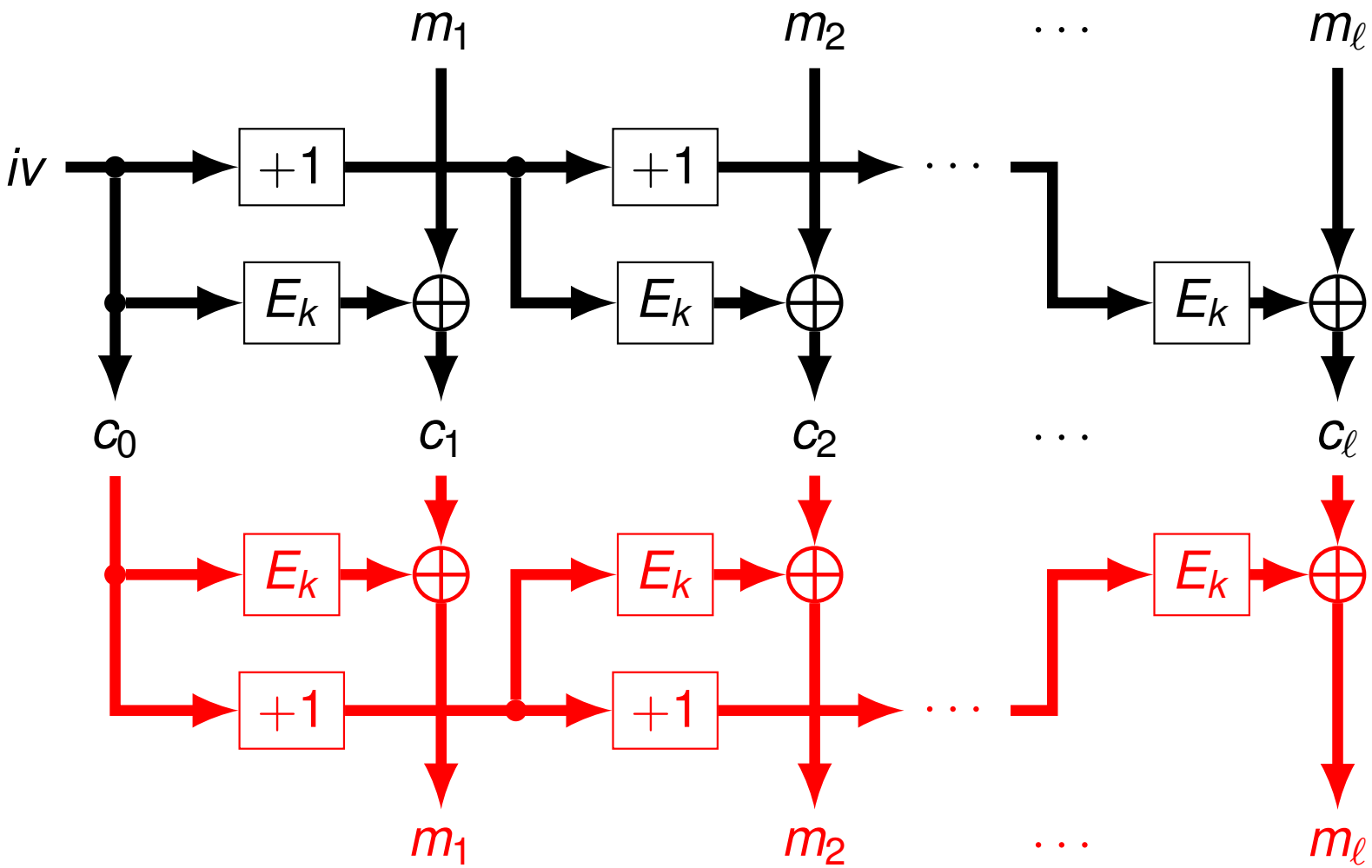
CTR Mode

Comp. . .



CTR Mode

Comp...



Comparison of Block Chaining Modes

- CFB, OFB and CTR require only the encryption box (even for decryption)
- CBC and CFB are “self-synchronizable”
- CTR can use random access for encryption and decryption (i.e., it is parallelizable)
- CBC and CFB decryption are also parallelizable (random access)
- A corrupted *iv* in OFB and CTR makes all blocks undecryptable
- Only ECB and CBC require padding of the last incomplete message block

CBC... CFB... OFB... CTR...

Plaintext Padding

If the last message block is not complete, a bit string is appended such that:

- It has the minimum possible length.
- It has to not introduce any ambiguity in decryption.

Plaintext Padding

If the last message block is not complete, a bit string is appended such that:

- It has the minimum possible length.
- It has to not introduce any ambiguity in decryption.

Example:

- Append a string: “1” followed by zero or more “0”.
- The empty string is not a valid padding. A last complete block also needs to be padded!
- In decryption, the last “1” and all trailing zeros (if any) are discarded.

Outline

1 Stream Ciphers

2 Block Ciphers

3 Block Chaining Modes

4 Key Management

5 Message Authentication Codes

Key Generation and Storage Issues

Key generation needs truly random bits (hard to extract from nature)

Key Generation and Storage Issues

Key generation needs truly random bits (hard to extract from nature)

- Low entropy sources (e.g., passwords) are not enough.
- Smoothing and entropy accumulation (heuristic techniques: diffusion, iteration), or dedicated hardware.
- Known attacks exploiting the lack of true randomness.

Key Generation and Storage Issues

Key generation needs truly random bits (hard to extract from nature)

- Low entropy sources (e.g., passwords) are not enough.
- Smoothing and entropy accumulation (heuristic techniques: diffusion, iteration), or dedicated hardware.
- Known attacks exploiting the lack of true randomness.

Keys have to be stored securely (Then, new keys required)

Key Generation and Storage Issues

Key generation needs truly random bits (hard to extract from nature)

- Low entropy sources (e.g., passwords) are not enough.
- Smoothing and entropy accumulation (heuristic techniques: diffusion, iteration), or dedicated hardware.
- Known attacks exploiting the lack of true randomness.

Keys have to be stored securely (Then, new keys required)

- Use hierarchical encryption: encrypt some keys under a common dedicated key.
- Can incur in **circularity**: plaintext depending on the key!
- Use a **Key Derivation Function** applied to a passphrase and a (public) nonce.

Outline

1 Stream Ciphers

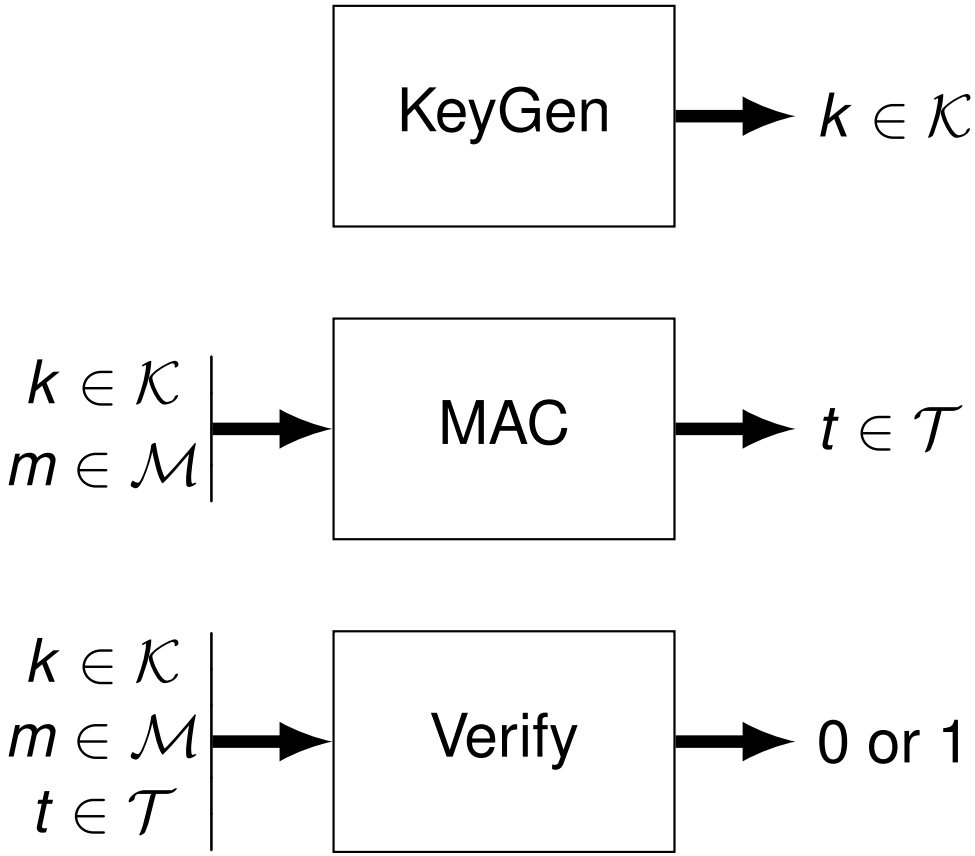
2 Block Ciphers

3 Block Chaining Modes

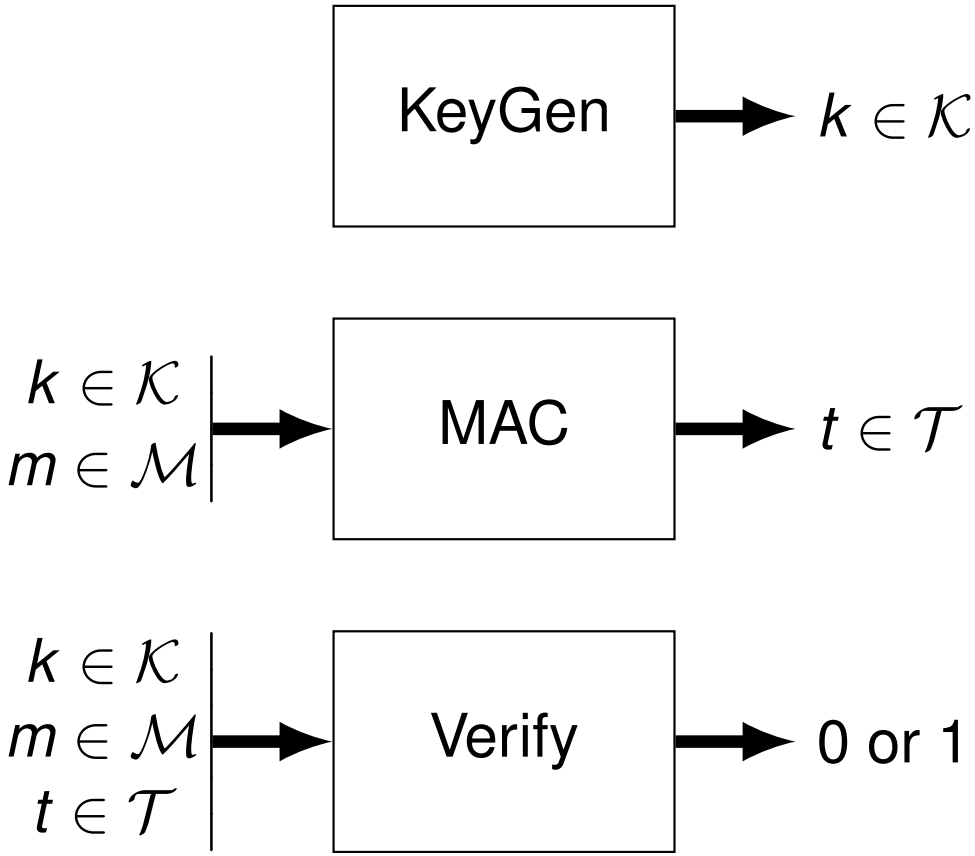
4 Key Management

5 Message Authentication Codes

Message Authentication Codes: Syntax

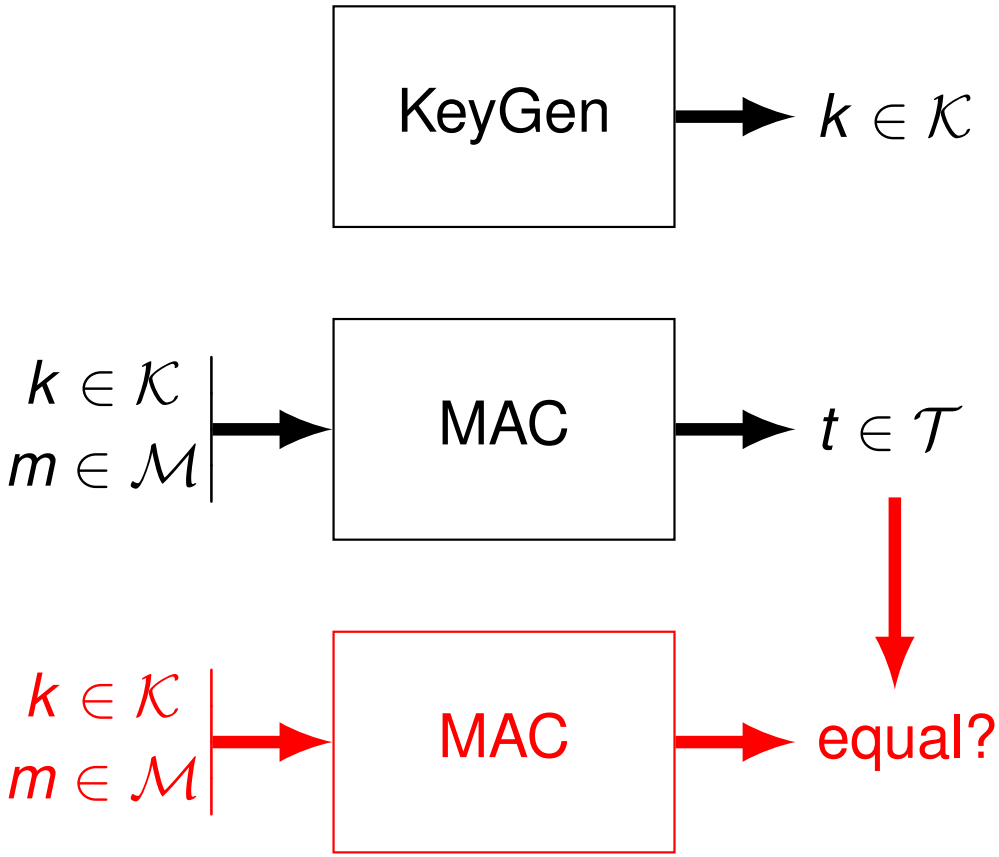


Message Authentication Codes: Correctness



$$\forall m \in \mathcal{M},\ \forall k \in \mathcal{K}, \quad \text{Verify}(k, m, \text{MAC}(k, m)) = 1$$

Typical Verification Procedure



$$\text{Verify}(k, m, t) = 1 \Leftrightarrow t = \text{MAC}(k, m)$$

MAC: Perfect Unforgeability

Informal definition:

"Impossible to find a new pair (m', t') from (m, t) without k"

MAC: Perfect Unforgeability

Informal definition:
"Impossible to find a new pair (m', t') from (m, t) without k"

Definition (Perfect One-Time Unforgeability)

For a uniformly distributed $K \in \mathcal{K}$ and for any different $m, m' \in \mathcal{M}$, **the tags** $\mathrm{MAC}(K, m)$ **and** $\mathrm{MAC}(K, m')$ **are independent random variables.**

MAC: Perfect Unforgeability

Informal definition:
"Impossible to find a new pair (m', t') from (m, t) without k"

Definition (Perfect One-Time Unforgeability)

For a uniformly distributed $K \in \mathcal{K}$ and for any different $m, m' \in \mathcal{M}$, **the tags** $\mathrm{MAC}(K, m)$ **and** $\mathrm{MAC}(K, m')$ **are independent random variables.**

Definition (Perfect n-Times Unforgeability)

For a uniformly distributed $K \in \mathcal{K}$ and for any different $m_1, m_2, \ldots, m_n, m' \in \mathcal{M}$, **the** $n+1$ **tags** $T_1 = \mathrm{MAC}(K, m_1), \ldots, T_n = \mathrm{MAC}(K, m_n)$ **and** $T' = \mathrm{MAC}(K, m')$ **are independent random variables.**

One-Time Perfect MAC

q is a ℓ-bit long prime, $\mathcal{M} = \mathcal{T} = \mathbb{Z}_q = \{0, 1, \ldots, q-1\}$ and $\mathcal{K} = \mathbb{Z}_q \times \mathbb{Z}_q$.

Construction

Public Parameters(ℓ) :
Choose a ℓ-bit long prime q;
output $pp = (q)$;

KeyGen(pp) :
Parse $pp = (q)$;
Choose random $a, b \leftarrow \mathbb{Z}_q$;
output $k = (a, b)$;

MAC(k, m) :
Parse $k = (a, b)$;
output $t = am + b \mod q$;

One-Time Perfect MAC

q is a ℓ-bit long prime, $\mathcal{M} = \mathcal{T} = \mathbb{Z}_q = \{0, 1, \ldots, q-1\}$ and $\mathcal{K} = \mathbb{Z}_q \times \mathbb{Z}_q$.

Construction

Public Parameters(ℓ) :
Choose a ℓ-bit long prime q;
output $pp = (q)$;

KeyGen(pp) :
Parse $pp = (q)$;
Choose random $a, b \leftarrow \mathbb{Z}_q$;
output $k = (a, b)$;

MAC(k, m) :
Parse $k = (a, b)$;
output $t = am + b \mod q$;

Unforgeability (one time): Even knowing a valid pair $(m, t = am + b)$, still $t' = am' + b$ remains random for any $m' \neq m$.

n-Times Perfect MAC

Now $\mathcal{K} = \mathbb{Z}_q^{n+1} = \underbrace{\mathbb{Z}_q \times \cdots \times \mathbb{Z}_q}_{n+1 \text{ copies}}$.

Construction

Public Parameters(ℓ) : (as before)

KeyGen(pp) :
Choose random $a_1, \ldots, a_n, b \leftarrow \mathbb{Z}_q$;
output $k = (a_1, \ldots, a_n, b)$;

MAC(k, m) :
output $t = P(m) = a_n m^n + \ldots + a_1 m + b \mod q$;

n-Times Perfect MAC

Now $\mathcal{K} = \mathbb{Z}_q^{n+1} = \underbrace{\mathbb{Z}_q \times \cdots \times \mathbb{Z}_q}_{n+1 \text{ copies}}$.

Construction

Public Parameters(ℓ) : (as before)

KeyGen(pp) :
Choose random $a_1, \ldots, a_n, b \leftarrow \mathbb{Z}_q$;
output $k = (a_1, \ldots, a_n, b)$;

MAC(k, m) :
output $t = P(m) = a_n m^n + \ldots + a_1 m + b \mod q$;

Unforgeability (n times): Even knowing n valid pairs $(m_1, t_1 = P(m_1)), \ldots, (m_n, t_n = P(m_n))$, still $t' = P(m')$ remains random for any $m' \notin \{m_1, \ldots, m_n\}$. (By Lagrange polynomial interpolation.)

Practical MAC Constructions

Perfect MACs are not practical:

- The key is larger than the message.
- Even in the n-times secure MAC, the key is larger than the size of the n messages together.
- The tag has the same size as the message.

Known efficient (non-perfect) constructions are based on

- hash functions (see next topic)
- block ciphers
- ...

Block Cipher Based MAC

CBC-MAC is a MAC based on any block cipher in CBC mode.

- A short long-term key is used to compute many tags.
- The tag length is independent of the message length.
- The computational complexity is similar to encrypting the message.

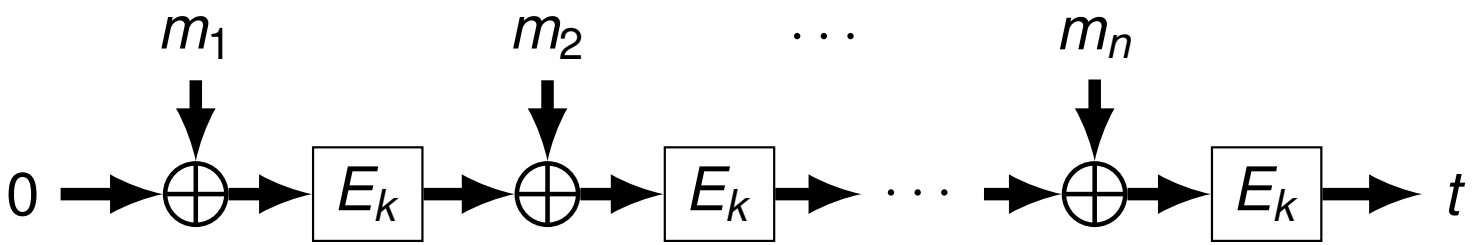
Construction

Public Parameters(ℓ) :
Choose a block cipher with message block size ℓ and a padding mechanism;

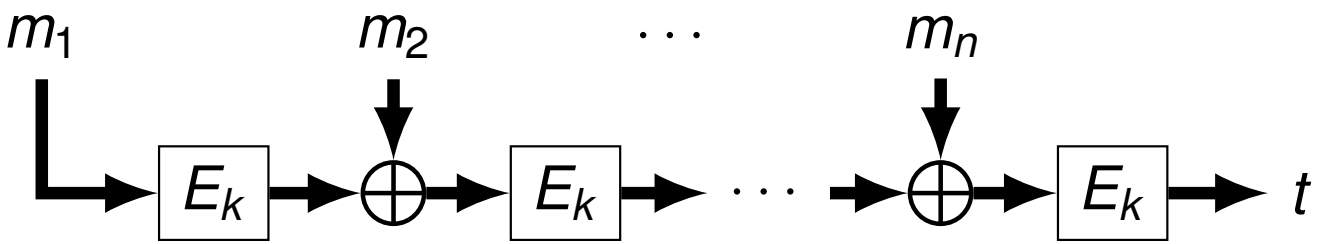
KeyGen(pp) :
Generate a random key k for the block cipher;
output k;

MAC(k, m) :
Encrypt (padded) m with key k using CBC mode and taking $iv = 0$;
output the last block of the encryption;

CBC-MAC Diagram



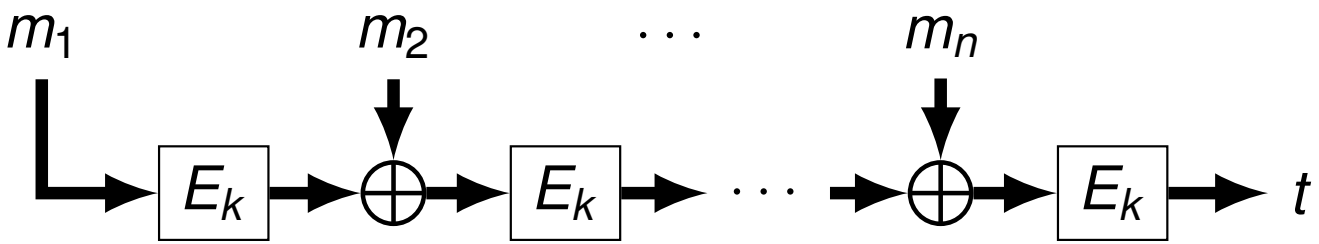
CBC-MAC Diagram



Unforgeability:

- CBC-MAC for messages of a fixed length (exact multiple of the block size) is **proven** to be as secure as the block cipher in CBC mode.
  Any attack against CBC-MAC implies a similar attack against the encryption scheme.

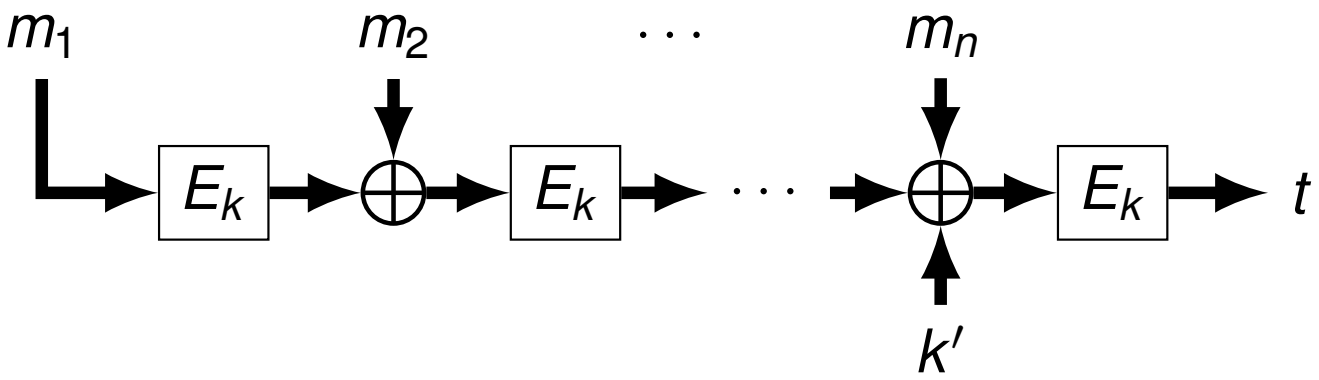
CBC-MAC Diagram



Unforgeability:

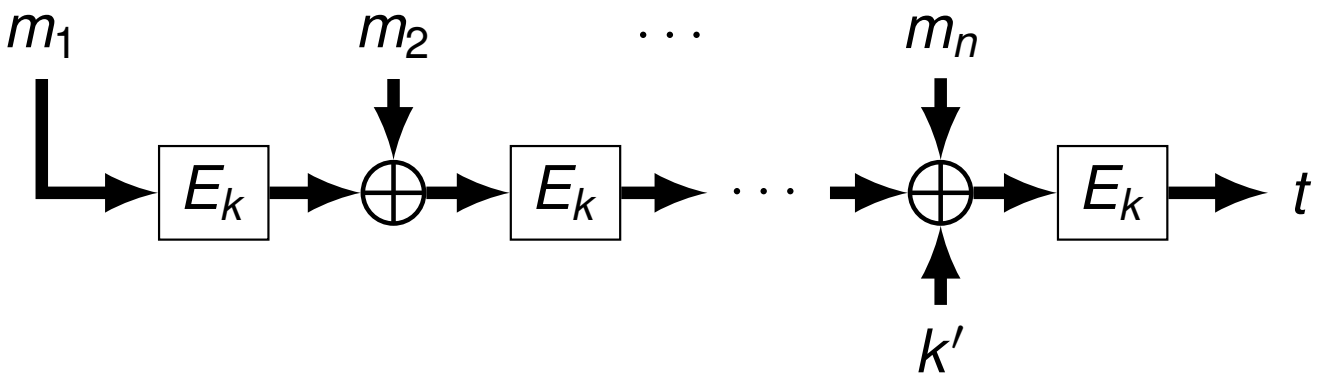
- CBC-MAC for messages of a fixed length (exact multiple of the block size) is **proven** to be as secure as the block cipher in CBC mode.
  Any attack against CBC-MAC implies a similar attack against the encryption scheme.
- A known (concatenation) attack exists against CBC-MAC for any block cipher.
  $t = \text{MAC}(k, m_1, \ldots, m_n) = \text{MAC}(k, m_1, \ldots, m_n, m_1 \oplus t, m_2 \ldots, m_n)$

One-Key MAC



CMAC solves the variable length security problem of CBC-MAC without increasing the size of the key.

One-Key MAC



CMAC solves the variable length security problem of CBC-MAC without increasing the size of the key.

- The last message block is “tweaked” with a key k' derived from k.
- Two different values of k' are generated: one is used when the last block requires to be padded, and the other when no padding is necessary.

Cryptology

Jorge L. Villar

FME, UPC, Fall 2024

END