DD2448 Foundations of Cryptography
Lecture 2

Douglas Wikström
KTH Royal Institute of Technology
dog@kth.se

January 29, 2016

Last Lecture: Simple Ciphers

- Caesar cipher and affine cipher: $m_i \mapsto am_i + b$.
- Substitution cipher: $m_i \mapsto \sigma(m_i)$.
- Vigénère cipher: $m_i \mapsto m_i + k_{i \bmod l}$.
- Hill cipher (linear map):

$$(m_1, \ldots, m_l) \mapsto A(m_1, \ldots, m_l)$$

- Transposition cipher (permutation):

$$(m_1, \ldots, m_l) \mapsto (m_{\pi(1)}, \ldots, m_{\pi(l)})$$

Substitution-Permutation Networks

Good Block Cipher

- For every key a block-cipher with plaintext/ciphertext space $\{0,1\}^n$ gives a permutation of $\{0,1\}^n$.

  What would be an good cipher?

Good Block Cipher

- For every key a block-cipher with plaintext/ciphertext space $\{0,1\}^n$ gives a permutation of $\{0,1\}^n$.

  What would be an good cipher?

- A good cipher is one where each key gives a **randomly chosen permutation** of $\{0,1\}^n$.

  Why is this not possible?

Good Block Cipher

- For every key a block-cipher with plaintext/ciphertext space $\{0,1\}^n$ gives a permutation of $\{0,1\}^n$.

  What would be an good cipher?

- A good cipher is one where each key gives a **randomly chosen permutation** of $\{0,1\}^n$.

  Why is this not possible?

- The representation of a single typical function $\{0,1\}^n \to \{0,1\}^n$ requires roughly $n2^n$ bits ($147 \times 10^{6\cdot 3}$ for $n = 64$)

Good Block Cipher

- For every key a block-cipher with plaintext/ciphertext space $\{0,1\}^n$ gives a permutation of $\{0,1\}^n$.

  What would be an good cipher?

- A good cipher is one where each key gives a **randomly chosen permutation** of $\{0,1\}^n$.

  Why is this not possible?

- The representation of a single typical function $\{0,1\}^n \to \{0,1\}^n$ requires roughly $n2^n$ bits $(147 \times 10^{6 \cdot 3}$ for $n = 64)$

- What should we look for instead?

Something Smaller

Idea. Compose smaller permutations into a large one. Mix the components "thoroughly".

Something Smaller

Idea. Compose smaller permutations into a large one. Mix the components "thoroughly".

Shannon (1948) calls this:

- **Diffusion.** "In the method of diffusion the statistical structure of M which leads to its redundancy is dissipated into long range statistics..."
- **Confusion.** "The method of confusion is to make the relation between the simple statistics of E and the simple description of K a very complex and involved one."

Substitution-Permutation Networks (1/2)

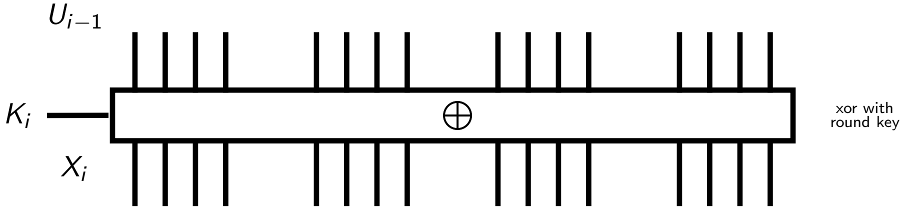
- **Block-size.** We use a block-size of $n = \ell \times m$ bits.
- **Key Schedule.** Round r uses its own round key K_r derived from the key K using a key schedule.
- **Each Round.** In each round we invoke:
  1. **Round Key.** xor with the round key.
  2. **Substitution.** ℓ substitution boxes each acting on one m-bit word (m-bit S-Boxes).
  3. **Permutation.** A permutation π_i acting on $\{1, \ldots, n\}$ to reorder the n bits.

Substitution-Permutation Networks (2/2)

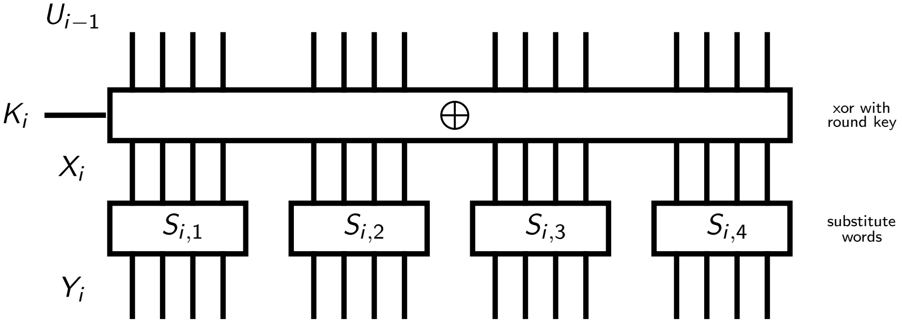
U_{i-1}

K_i

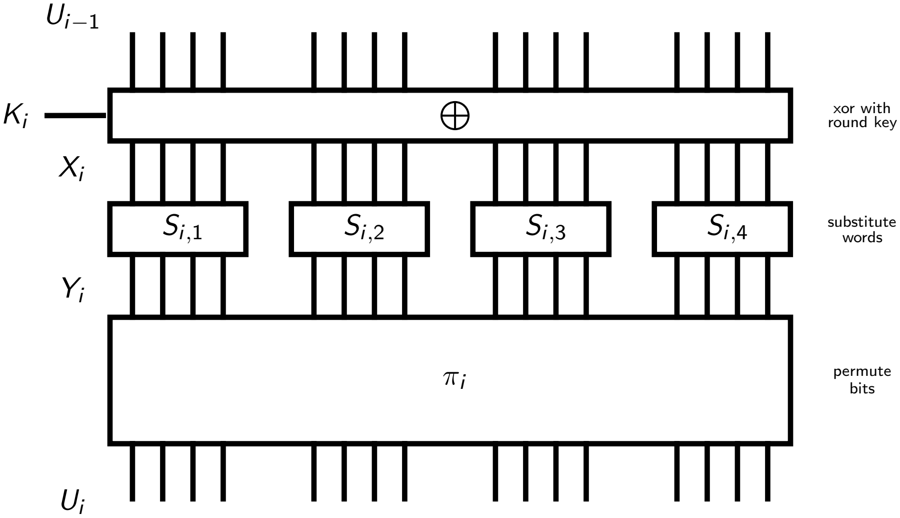
Substitution-Permutation Networks (2/2)



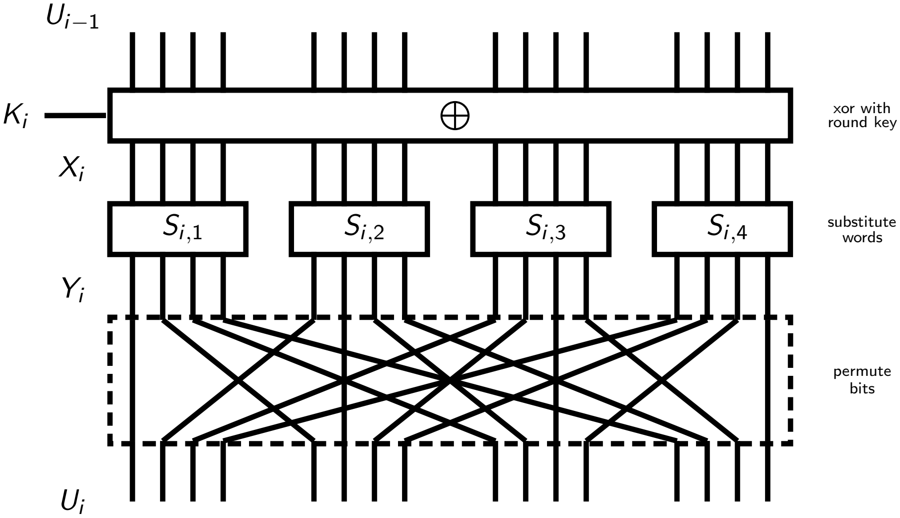
Substitution-Permutation Networks (2/2)



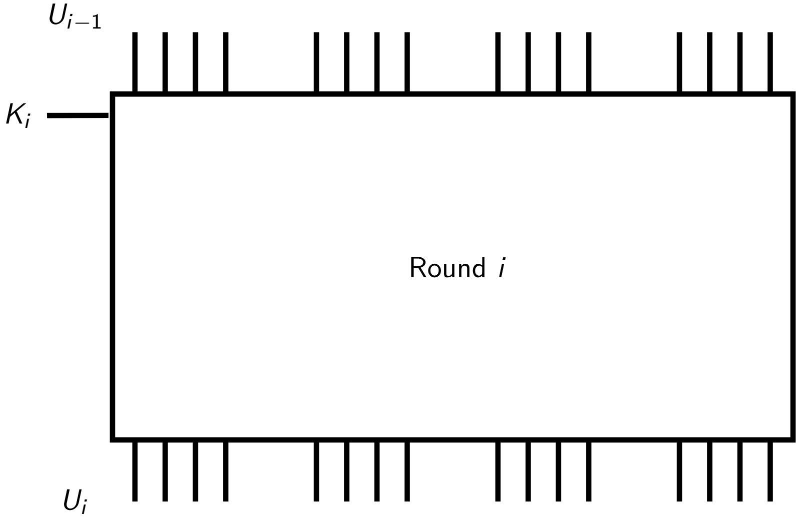
Substitution-Permutation Networks (2/2)



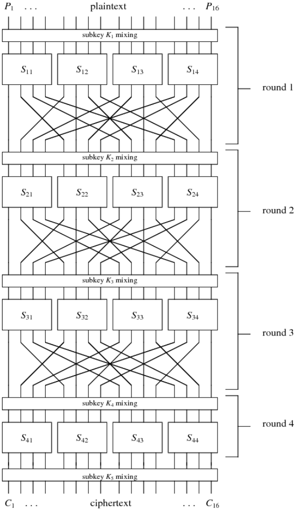
Substitution-Permutation Networks (2/2)



Substitution-Permutation Networks (2/2)



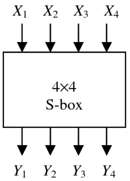
A Simple Block Cipher (1/2)



- $|P| = |C| = 16$
- 4 rounds
- $|K| = 32$
- rth round key K_r consists of the $4r$th to the $(4r+16)$th bits of key K.
- 4-bit S-Boxes

A Simple Block Cipher (2/2)

S-Boxes the same ($S \neq S^{-1}$)

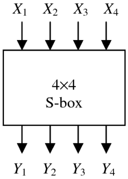


- $Y = S(X)$
- Can be described using 4 boolean functions

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

A Simple Block Cipher (2/2)

S-Boxes the same ($S \neq S^{-1}$)



- $Y = S(X)$
- Can be described using 4 boolean functions

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

16-bit permutation ($\pi = \pi^{-1}$)

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

AES

Advanced Encryption Standard (AES)

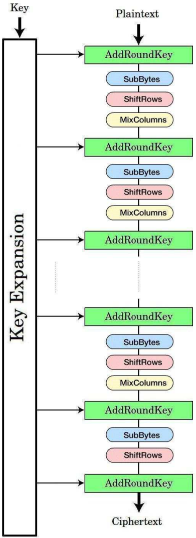
- Chosen in worldwide **public competition** 1997-2000. Probably no backdoors. Increased confidence!
- Winning proposal named "Rijndael", by Rijmen and Daemen
- Family of 128-bit block ciphers:

Key bits	128	192	256
Rounds	10	12	14

- The first key-recovery attacks on full AES due to Bogdanov, Khovratovich, and Rechberger, published **2011**, is faster than brute force by a factor of about **4**.
- ... algebraics of AES make some people uneasy.

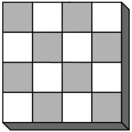
AES

- **AddRoundKey**: xor with round key.
- **SubBytes**: substitution of bytes.
- **ShiftRows**: permutation of bytes.
- **MixColumns**: linear map.



Similar to SPN

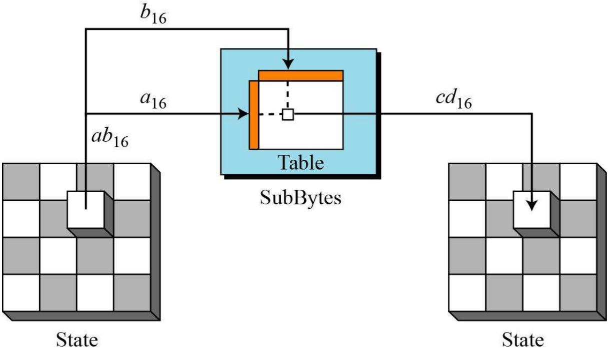
The 128 bit state is interpreted as a 4×4 matrix of bytes.



Something like a mix between substitution, permutation, affine version of Hill cipher. In each round!

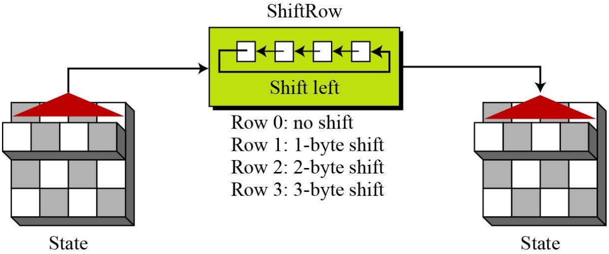
SubBytes

`SubBytes` is field inversion in $\mathbb{F}_{2^8}$ plus affine map in $\mathbb{F}_2^8$.



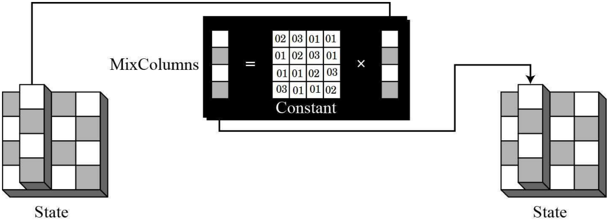
ShiftRows

`ShiftRows` is a cyclic shift of bytes with offsets: 0, 1, 2, and 3.



MixColumns

`MixColumns` is an invertible linear map over $\mathbb{F}_{2^8}$ (with irreducibile polynomial $x^8 + x^4 + x^3 + x + 1$) with good diffusion.



Decryption

Uses the following transforms:

- **AddRoundKey**
- **InvSubBytes**
- **InvShiftRows**
- **InvMixColumns**

Feistel Networks

Feistel Networks

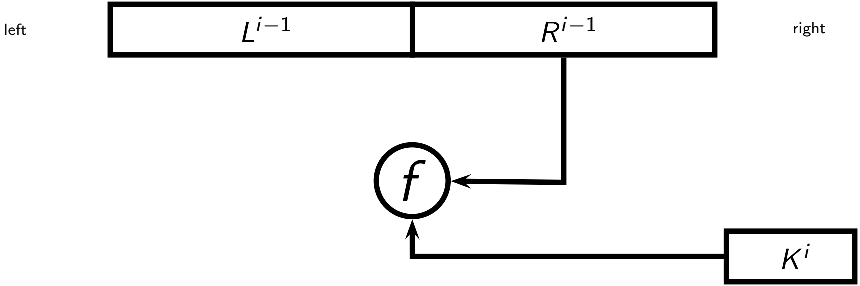
- Identical rounds are iterated, but with different round keys.
- The input to the ith round is divided in a left and right part, denoted L^{i-1} and R^{i-1}.
- f is a function for which it is somewhat hard to find pre-images, but f is typically **not invertible**!
- One round is defined by:

$$
\begin{aligned}
L^i &= R^{i-1} \\
R^i &= L^{i-1} \oplus f(R^{i-1}, K^i)
\end{aligned}
$$

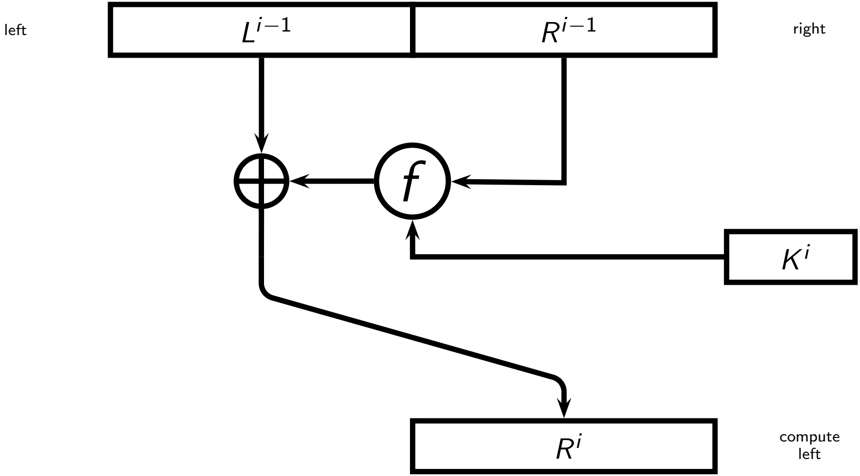
where K^i is the ith round key.

Feistel Round

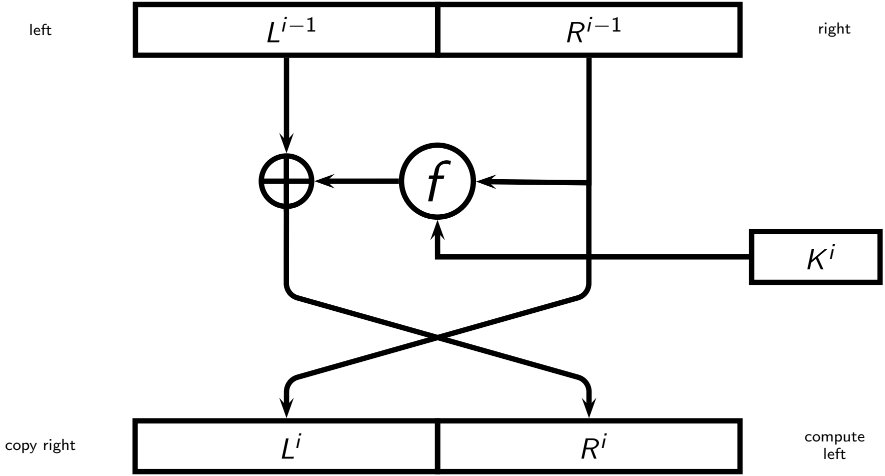
Feistel Round



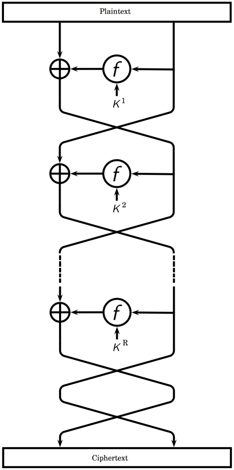
Feistel Round



Feistel Round



Feistel Cipher



Inverse Feistel Round

Feistel Round.

$$L^i = R^{i-1}$$
$$R^i = L^{i-1} \oplus f(R^{i-1}, K^i)$$

Inverse Feistel Round

Feistel Round.

$$L^i = R^{i-1}$$
$$R^i = L^{i-1} \oplus f(R^{i-1}, K^i)$$

Inverse Feistel Round.

$$L^{i-1} = R^i \oplus f(L^i, K^i)$$
$$R^{i-1} = L^i$$

Reverse direction and swap left and right!

DES

Quote

The news here is not that DES is insecure, that hardware algorithm-crackers can be built, or that a 56-bit key length is too short. ... The news is how long the government has been denying that these machines were possible. As recently as 8 June 98, Robert Litt, principal associate deputy attorney general at the Department of Justice, denied that it was possible for the FBI to crack DES. ... My comment was that the FBI is either incompetent or lying, or both.

– Bruce Schneier, 1998

Data Encryption Standard (DES)

- Developed at IBM in 1975, or perhaps...

Data Encryption Standard (DES)

- Developed at IBM in 1975, or perhaps...
- at National Security Agency (NSA). Nobody knows for certain.

Data Encryption Standard (DES)

- Developed at IBM in 1975, or perhaps...
- at National Security Agency (NSA). Nobody knows for certain.
- 16-round Feistel network.

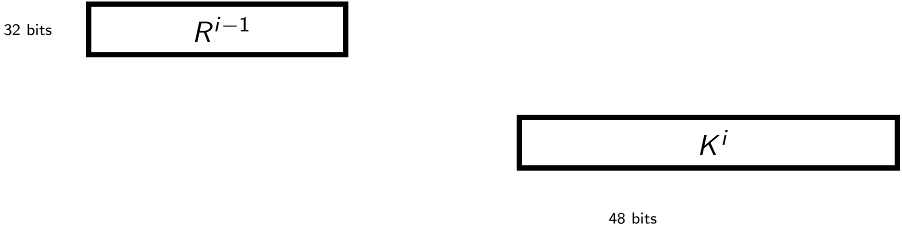
Data Encryption Standard (DES)

- Developed at IBM in 1975, or perhaps...
- at National Security Agency (NSA). Nobody knows for certain.
- 16-round Feistel network.
- Key schedule derives permuted bits for each round key from a 56-bit key. Supposedly not 64-bit due to parity bits.

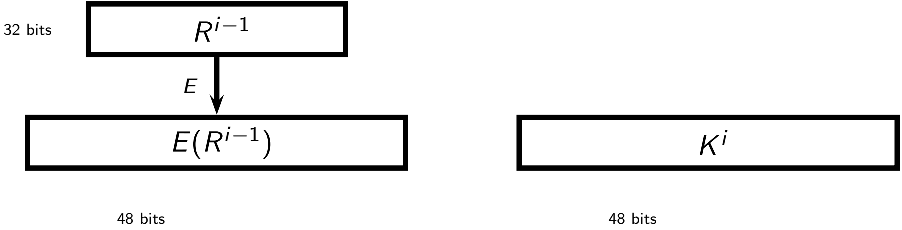
Data Encryption Standard (DES)

- Developed at IBM in 1975, or perhaps...
- at National Security Agency (NSA). Nobody knows for certain.
- 16-round Feistel network.
- Key schedule derives permuted bits for each round key from a 56-bit key. Supposedly not 64-bit due to parity bits.
- Let us look a little at the Feistel-function f.

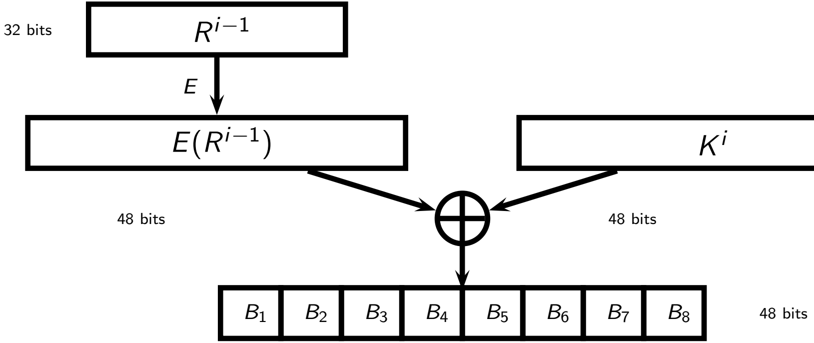
DES's f-Function



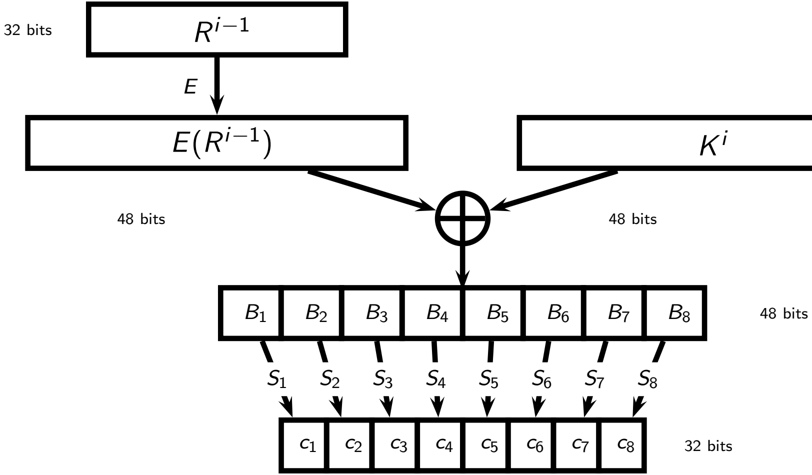
DES's f-Function



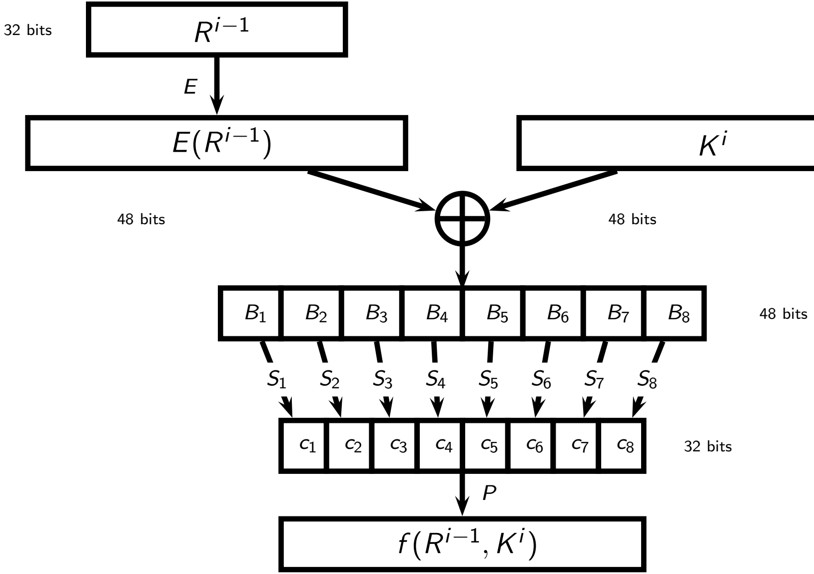
DES's f-Function



DES's f-Function



DES's f-Function



Security of DES

- **Brute Force.** Try all 2^{56} keys. Done in practice with special chip by Electronic Frontier Foundation, 1998. Likely much earlier by NSA and others.
- **Differential Cryptanalysis.** 2^{47} chosen plaintexts, Biham and Shamir, 1991. (approach: late 80'ies). Known earlier by IBM and NSA. DES is surprisingly resistant!
- **Linear Cryptanalysis.** 2^{43} known plaintexts, Matsui, 1993. Probably **not** known by IBM and NSA!

Double DES

We have seen that the key space of DES is too small. One way to increase it is to use DES twice, so called "double DES".

$$\mathrm{2DES}_{k_1,k_2}(x) = \mathrm{DES}_{k_2}(\mathrm{DES}_{k_1}(x))$$

Is this more secure than DES?

This question is valid for any cipher.

Meet-In-the-Middle Attack

- Get hold of a plaintext-ciphertext pair (m, c)
- Compute $X = \{x \mid k_1 \in \mathcal{K}_{\mathrm{DES}} \wedge x = \mathsf{E}_{k_1}(m)\}$.
- For $k_2 \in \mathcal{K}_{\mathrm{DES}}$ check if $\mathsf{E}_{k_2}^{-1}(c) = \mathsf{E}_{k_1}(m)$ for some k_1 using the table X. If so, then (k_1, k_2) is a good candidate.
- Repeat with (m', c'), starting from the set of candidate keys to identify correct key.

Triple DES

What about triple DES?

$$3\mathrm{DES}_{k_1,k_2,k_3}(x) = \mathrm{DES}_{k_3}(\mathrm{DES}_{k_2}(\mathrm{DES}_{k_1}(x)))$$

- Seemingly 112 bit "effective" key size.
- 3 times as slow as DES. DES is slow in software, and this is even worse. One of the motivations of AES.
- Triple DES is still considered to be secure.

Modes of Operation

Modes of Operation

- Electronic codebook mode (ECB mode).
- Cipher feedback mode (CFB mode).
- Cipher block chaining mode (CBC mode).
- Output feedback mode (OFB mode).
- Counter mode (CTR mode).

ECB Mode

Electronic codebook mode

Encrypt each block independently:

$$c_i = \mathsf{E}_k(m_i)$$

ECB Mode

Electronic codebook mode

Encrypt each block independently:

$$c_i = \mathrm{E}_k(m_i)$$

- Identical plaintext blocks give identical ciphertext blocks.

ECB Mode

Electronic codebook mode

Encrypt each block independently:

$$c_i = \mathsf{E}_k(m_i)$$

- Identical plaintext blocks give identical ciphertext blocks.
- How can we avoid this?

CFB Mode

Cipher feedback mode

xor plaintext block with previous ciphertext block **after** encryption:

$$c_0 = \text{initialization vector}$$
$$c_i = m_i \oplus \mathsf{E}_k(c_{i-1})$$

CFB Mode

Cipher feedback mode

xor plaintext block with previous ciphertext block **after** encryption:

$$c_0 = \text{initialization vector}$$
$$c_i = m_i \oplus \mathsf{E}_k(c_{i-1})$$

- Sequential encryption and parallel decryption.

CFB Mode

Cipher feedback mode

xor plaintext block with previous ciphertext block **after** encryption:

$$c_0 = \text{initialization vector}$$
$$c_i = m_i \oplus \mathsf{E}_k(c_{i-1})$$

- Sequential encryption and parallel decryption.
- Self-synchronizing.

CFB Mode

Cipher feedback mode

xor plaintext block with previous ciphertext block **after** encryption:

$$c_0 = \text{initialization vector}$$
$$c_i = m_i \oplus \mathsf{E}_k(c_{i-1})$$

- Sequential encryption and parallel decryption.
- Self-synchronizing.
- How do we pick the initialization vector?

CBC Mode

Cipher block chaining mode

xor plaintext block with previous ciphertext block **before** encryption:

$$\begin{aligned} c_0 &= \text{initialization vector} \\ c_i &= \mathsf{E}_k\left(c_{i-1} \oplus m_i\right) \end{aligned}$$

CBC Mode

Cipher block chaining mode

xor plaintext block with previous ciphertext block **before** encryption:

$$c_0 = \text{initialization vector}$$
$$c_i = \mathsf{E}_k(c_{i-1} \oplus m_i)$$

- Sequential encryption and parallel decryption

CBC Mode

Cipher block chaining mode

xor plaintext block with previous ciphertext block **before** encryption:

$$c_0 = \text{initialization vector}$$
$$c_i = \mathsf{E}_k\left(c_{i-1} \oplus m_i\right)$$

- Sequential encryption and parallel decryption
- Self-synchronizing.

OFB Mode

Output feedback mode

Generate stream, xor plaintexts with stream (emulate "one-time pad"):

$$
\begin{aligned}
s_0 &= \text{initialization vector} \\
s_i &= \mathsf{E}_k(s_{i-1}) \\
c_i &= s_i \oplus m_i
\end{aligned}
$$

OFB Mode

Output feedback mode

Generate stream, xor plaintexts with stream (emulate "one-time pad"):

$$
\begin{aligned}
s_0 &= \text{initialization vector} \\
s_i &= \mathsf{E}_k(s_{i-1}) \\
c_i &= s_i \oplus m_i
\end{aligned}
$$

- Sequential.

OFB Mode

Output feedback mode

Generate stream, xor plaintexts with stream (emulate "one-time pad"):

$$
\begin{aligned}
s_0 &= \text{initialization vector} \\
s_i &= \mathsf{E}_k(s_{i-1}) \\
c_i &= s_i \oplus m_i
\end{aligned}
$$

- Sequential.
- Synchronous.

OFB Mode

Output feedback mode

Generate stream, xor plaintexts with stream (emulate "one-time pad"):

$$
\begin{aligned}
s_0 &= \text{initialization vector} \\
s_i &= \mathsf{E}_k(s_{i-1}) \\
c_i &= s_i \oplus m_i
\end{aligned}
$$

- Sequential.
- Synchronous.
- Allows batch processing.

OFB Mode

Output feedback mode

Generate stream, xor plaintexts with stream (emulate "one-time pad"):

$$\begin{aligned} s_0 &= \text{initialization vector} \\ s_i &= \mathsf{E}_k(s_{i-1}) \\ c_i &= s_i \oplus m_i \end{aligned}$$

- Sequential.
- Synchronous.
- Allows batch processing.
- Malleable!

CTR Mode

Counter mode

Generate stream, xor plaintexts with stream (emulate "one-time pad"):

$$
\begin{aligned}
s_0 &= \text{initialization vector} \\
s_i &= \mathsf{E}_k(s_0 \| i) \\
c_i &= s_i \oplus m_i
\end{aligned}
$$

CTR Mode

Counter mode

Generate stream, xor plaintexts with stream (emulate "one-time pad"):

$$s_0 = \text{initialization vector}$$
$$s_i = \mathsf{E}_k(s_0 \| i)$$
$$c_i = s_i \oplus m_i$$

- Parallel.

CTR Mode

Counter mode

Generate stream, xor plaintexts with stream (emulate “one-time pad”):

$$
\begin{aligned}
s_0 &= \text{initialization vector} \\
s_i &= \mathsf{E}_k(s_0 \| i) \\
c_i &= s_i \oplus m_i
\end{aligned}
$$

- Parallel.
- Synchronous.

CTR Mode

Counter mode

Generate stream, xor plaintexts with stream (emulate "one-time pad"):

$$
\begin{aligned}
s_0 &= \text{initialization vector} \\
s_i &= \mathsf{E}_k(s_0 \| i) \\
c_i &= s_i \oplus m_i
\end{aligned}
$$

- Parallel.
- Synchronous.
- Allows batch processing.

CTR Mode

Counter mode

Generate stream, xor plaintexts with stream (emulate "one-time pad"):

$$\begin{aligned} s_0 &= \text{initialization vector} \\ s_i &= \mathsf{E}_k(s_0 \| i) \\ c_i &= s_i \oplus m_i \end{aligned}$$

- Parallel.
- Synchronous.
- Allows batch processing.
- Malleable!