Public-key Cryptography
Theory and Practice

Abhijit Das

Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur

Appendix A: Symmetric Techniques

Block Ciphers

Block Ciphers

- A block cipher f of **block-size** n and **key-size** r is a function

$$f : \mathbb{Z}_2^n \times \mathbb{Z}_2^r \rightarrow \mathbb{Z}_2^n$$

that maps (M, K) to $C = f(M, K)$.

Block Ciphers

- A block cipher f of **block-size** n and **key-size** r is a function

$$f : \mathbb{Z}_2^n \times \mathbb{Z}_2^r \to \mathbb{Z}_2^n$$

  that maps (M, K) to $C = f(M, K)$.
- For each key K, the map

$$f_K : \mathbb{Z}_2^n \to \mathbb{Z}_2^n$$

  taking a plaintext message M to the ciphertext message $C = f_K(M) = f(M, K)$ should be bijective (invertible).

Block Ciphers

- A block cipher f of **block-size** n and **key-size** r is a function

$$f : \mathbb{Z}_2^n \times \mathbb{Z}_2^r \to \mathbb{Z}_2^n$$

  that maps (M, K) to $C = f(M, K)$.
- For each key K, the map

$$f_K : \mathbb{Z}_2^n \to \mathbb{Z}_2^n$$

  taking a plaintext message M to the ciphertext message $C = f_K(M) = f(M, K)$ should be bijective (invertible).
- n and r should be large enough to preclude successful exhaustive search.

Block Ciphers

- A block cipher f of **block-size** n and **key-size** r is a function

$$f : \mathbb{Z}_2^n \times \mathbb{Z}_2^r \to \mathbb{Z}_2^n$$

  that maps (M, K) to $C = f(M, K)$.
- For each key K, the map

$$f_K : \mathbb{Z}_2^n \to \mathbb{Z}_2^n$$

  taking a plaintext message M to the ciphertext message $C = f_K(M) = f(M, K)$ should be bijective (invertible).
- n and r should be large enough to preclude successful exhaustive search.
- Each f_K should be a sufficiently random permutation.

Block Ciphers: Examples

Name	n, r
DES (Data Encryption Standard)	64, 56
FEAL (Fast Data Encipherment Algorithm)	64, 64
SAFER (Secure And Fast Encryption Routine)	64, 64
IDEA (International Data Encryption Algorithm)	64, 128
Blowfish	64, $\leqslant$ 448
The AES Finalists	
Rijndael (Rijmen and Daemen)	128, 128/192/256
Serpent (Anderson, Biham and Knudsen)	128, 128/192/256
Twofish (Schneier and others)	128, $\leqslant$ 256
RC6 (Rivest and others)	128, 128/192/256
MARS (Coppersmith and others)	128, 128–448 (multiple of 32)

Block Ciphers: Examples

Name	n, r
DES (Data Encryption Standard)	64, 56
FEAL (Fast Data Encipherment Algorithm)	64, 64
SAFER (Secure And Fast Encryption Routine)	64, 64
IDEA (International Data Encryption Algorithm)	64, 128
Blowfish	64, $\leqslant$ 448
The AES Finalists	
Rijndael (Rijmen and Daemen)	128, 128/192/256
Serpent (Anderson, Biham and Knudsen)	128, 128/192/256
Twofish (Schneier and others)	128, $\leqslant$ 256
RC6 (Rivest and others)	128, 128/192/256
MARS (Coppersmith and others)	128, 128–448 (multiple of 32)

Old standard: DES

Block Ciphers: Examples

Name	n, r
DES (Data Encryption Standard)	64, 56
FEAL (Fast Data Encipherment Algorithm)	64, 64
SAFER (Secure And Fast Encryption Routine)	64, 64
IDEA (International Data Encryption Algorithm)	64, 128
Blowfish	64, $\leqslant$ 448
The AES Finalists	
Rijndael (Rijmen and Daemen)	128, 128/192/256
Serpent (Anderson, Biham and Knudsen)	128, 128/192/256
Twofish (Schneier and others)	128, $\leqslant$ 256
RC6 (Rivest and others)	128, 128/192/256
MARS (Coppersmith and others)	128, 128–448 (multiple of 32)

Old standard: DES

New standard: AES (adaptation of the Rijndael cipher)

Block Ciphers: Security Requirements

Block Ciphers: Security Requirements

- Introduced by Shannon in 1949.

Block Ciphers: Security Requirements

- Introduced by Shannon in 1949.
- **Confusion**

Block Ciphers: Security Requirements

- Introduced by Shannon in 1949.
- **Confusion**
  - The relation between key and ciphertext must be very complex.

Block Ciphers: Security Requirements

- Introduced by Shannon in 1949.
- **Confusion**
  - The relation between key and ciphertext must be very complex.
  - Changing a single key bit should affect every ciphertext bit pseudorandomly.

Block Ciphers: Security Requirements

- Introduced by Shannon in 1949.
- **Confusion**
  - The relation between key and ciphertext must be very complex.
  - Changing a single key bit should affect every ciphertext bit pseudorandomly.
  - Ideally, for a change in each key bit, each ciphertext bit should change with probability $1/2$.

Block Ciphers: Security Requirements

- Introduced by Shannon in 1949.
- **Confusion**
  - The relation between key and ciphertext must be very complex.
  - Changing a single key bit should affect every ciphertext bit pseudorandomly.
  - Ideally, for a change in each key bit, each ciphertext bit should change with probability $1/2$.
  - Confusion is meant to make the guess of the key difficult.

Block Ciphers: Security Requirements

- Introduced by Shannon in 1949.
- **Confusion**
  - The relation between key and ciphertext must be very complex.
  - Changing a single key bit should affect every ciphertext bit pseudorandomly.
  - Ideally, for a change in each key bit, each ciphertext bit should change with probability $1/2$.
  - Confusion is meant to make the guess of the key difficult.
- **Diffusion**

Block Ciphers: Security Requirements

- Introduced by Shannon in 1949.
- **Confusion**
  - The relation between key and ciphertext must be very complex.
  - Changing a single key bit should affect every ciphertext bit pseudorandomly.
  - Ideally, for a change in each key bit, each ciphertext bit should change with probability $1/2$.
  - Confusion is meant to make the guess of the key difficult.
- **Diffusion**
  - The relation between plaintext and ciphertext must be very complex.

Block Ciphers: Security Requirements

- Introduced by Shannon in 1949.
- **Confusion**
  - The relation between key and ciphertext must be very complex.
  - Changing a single key bit should affect every ciphertext bit pseudorandomly.
  - Ideally, for a change in each key bit, each ciphertext bit should change with probability $1/2$.
  - Confusion is meant to make the guess of the key difficult.
- **Diffusion**
  - The relation between plaintext and ciphertext must be very complex.
  - Changing a single plaintext bit should affect every ciphertext bit pseudorandomly.

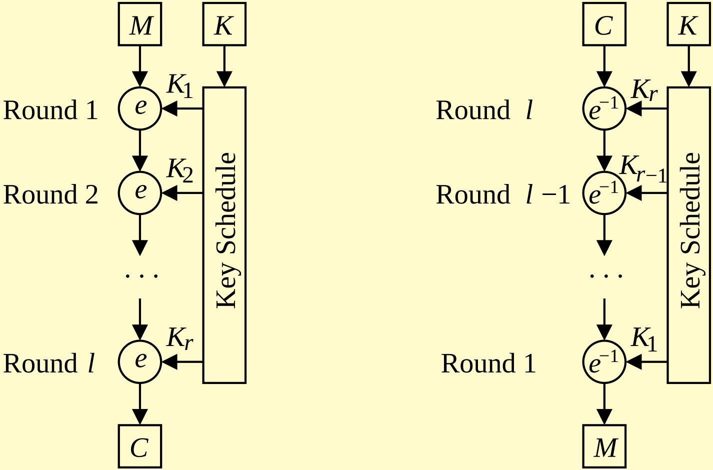
Block Ciphers: Security Requirements

- Introduced by Shannon in 1949.
- **Confusion**
  - The relation between key and ciphertext must be very complex.
  - Changing a single key bit should affect every ciphertext bit pseudorandomly.
  - Ideally, for a change in each key bit, each ciphertext bit should change with probability $1/2$.
  - Confusion is meant to make the guess of the key difficult.
- **Diffusion**
  - The relation between plaintext and ciphertext must be very complex.
  - Changing a single plaintext bit should affect every ciphertext bit pseudorandomly.
  - Ideally, for a change in each plaintext bit, each ciphertext bit should change with probability $1/2$.

Block Ciphers: Security Requirements

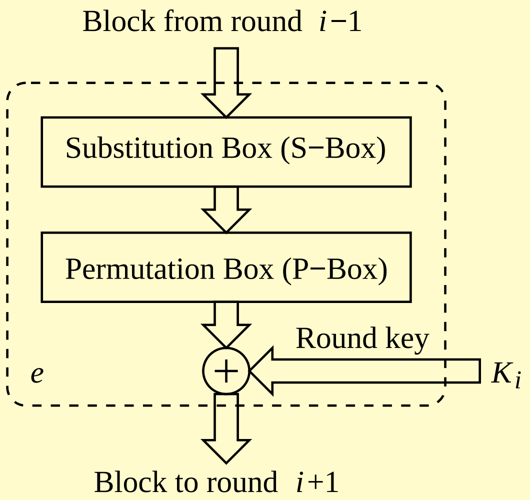
- Introduced by Shannon in 1949.
- **Confusion**
  - The relation between key and ciphertext must be very complex.
  - Changing a single key bit should affect every ciphertext bit pseudorandomly.
  - Ideally, for a change in each key bit, each ciphertext bit should change with probability $1/2$.
  - Confusion is meant to make the guess of the key difficult.
- **Diffusion**
  - The relation between plaintext and ciphertext must be very complex.
  - Changing a single plaintext bit should affect every ciphertext bit pseudorandomly.
  - Ideally, for a change in each plaintext bit, each ciphertext bit should change with probability $1/2$.
  - Diffusion is meant to dissipate plaintext redundancy.

Iterated Block Cipher

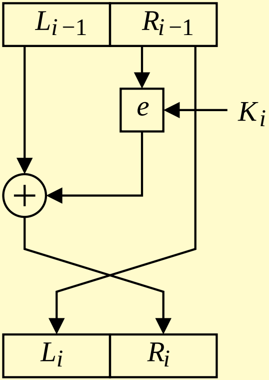


(a) Encryption (b) Decryption

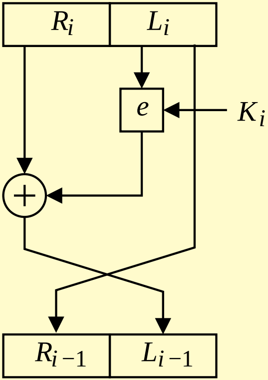
Substitution-Permutation Network (SPN)



Feistel Cipher



(a) Encryption



(b) Decryption

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

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

DES (Data Encryption Standard)

DES (Data Encryption Standard)

- Proposed as a US standard in 1975.

DES (Data Encryption Standard)

- Proposed as a US standard in 1975.
- DES supports blocks of length $n = 64$ bits.

DES (Data Encryption Standard)

- Proposed as a US standard in 1975.
- DES supports blocks of length $n = 64$ bits.
- DES supports keys $K = k_1 k_2 \ldots k_{64}$ of length $r = 64$ bits, but the bits $k_8, k_{16}, \ldots, k_{64}$ are used as parity-check bits. So the effective key size is 56 bits.

DES (Data Encryption Standard)

- Proposed as a US standard in 1975.
- DES supports blocks of length $n = 64$ bits.
- DES supports keys $K = k_1 k_2 \ldots k_{64}$ of length $r = 64$ bits, but the bits $k_8, k_{16}, \ldots, k_{64}$ are used as parity-check bits. So the effective key size is 56 bits.
- DES is a Feistel cipher.

DES (Data Encryption Standard)

- Proposed as a US standard in 1975.
- DES supports blocks of length $n = 64$ bits.
- DES supports keys $K = k_1 k_2 \ldots k_{64}$ of length $r = 64$ bits, but the bits $k_8, k_{16}, \ldots, k_{64}$ are used as parity-check bits. So the effective key size is 56 bits.
- DES is a Feistel cipher.
- The number of rounds in DES is $l = 16$.

DES Key Schedule

Input: A DES key $K = k_1 k_2 \ldots k_{64}$.
Output: Sixteen 48-bit round keys $K_1, K_2, \ldots, K_{16}$.

DES Key Schedule

Input: A DES key $K = k_1 k_2 \ldots k_{64}$.
Output: Sixteen 48-bit round keys $K_1, K_2, \ldots, K_{16}$.

- Generate 56-bit permuted key $U_0 = \mathrm{PC1}(K) = k_{57} k_{49} k_{41} \ldots k_{12} k_4$.

DES Key Schedule

Input: A DES key $K = k_1k_2 \ldots k_{64}$.
Output: Sixteen 48-bit round keys $K_1, K_2, \ldots, K_{16}$.

- Generate 56-bit permuted key $U_0 = \text{PC1}(K) = k_{57}k_{49}k_{41} \ldots k_{12}k_4$.
- Break U_0 in two 28-bit parts: $U_0 = C_0 \,||\, D_0$.

DES Key Schedule

Input: A DES key $K = k_1 k_2 \ldots k_{64}$.
Output: Sixteen 48-bit round keys $K_1, K_2, \ldots, K_{16}$.

- Generate 56-bit permuted key $U_0 = \text{PC1}(K) = k_{57} k_{49} k_{41} \ldots k_{12} k_4$.
- Break U_0 in two 28-bit parts: $U_0 = C_0 \mid\mid D_0$.
- for $i = 1, 2, \ldots, 16$, repeat the following steps:

DES Key Schedule

Input: A DES key $K = k_1 k_2 \ldots k_{64}$.
Output: Sixteen 48-bit round keys $K_1, K_2, \ldots, K_{16}$.

- Generate 56-bit permuted key $U_0 = \mathrm{PC1}(K) = k_{57}k_{49}k_{41}\ldots k_{12}k_4$.
- Break U_0 in two 28-bit parts: $U_0 = C_0 \,||\, D_0$.
- for $i = 1, 2, \ldots, 16$, repeat the following steps:
  - Take $s := \begin{cases} 1 & \text{if } i = 1, 2, 9, 16, \\ 2 & \text{otherwise.} \end{cases}$

DES Key Schedule

Input: A DES key $K = k_1 k_2 \ldots k_{64}$.
Output: Sixteen 48-bit round keys $K_1, K_2, \ldots, K_{16}$.

- Generate 56-bit permuted key $U_0 = \text{PC1}(K) = k_{57} k_{49} k_{41} \ldots k_{12} k_4$.
- Break U_0 in two 28-bit parts: $U_0 = C_0 \,||\, D_0$.
- for $i = 1, 2, \ldots, 16$, repeat the following steps:
  - Take $s := \begin{cases} 1 & \text{if } i = 1, 2, 9, 16, \\ 2 & \text{otherwise.} \end{cases}$
  - Cyclically left shift C_{i-1} by s bits to get C_i.

DES Key Schedule

Input: A DES key $K = k_1 k_2 \ldots k_{64}$.
Output: Sixteen 48-bit round keys $K_1, K_2, \ldots, K_{16}$.

- Generate 56-bit permuted key $U_0 = \mathrm{PC1}(K) = k_{57} k_{49} k_{41} \ldots k_{12} k_4$.
- Break U_0 in two 28-bit parts: $U_0 = C_0 \,||\, D_0$.
- for $i = 1, 2, \ldots, 16$, repeat the following steps:
  - Take $s := \begin{cases} 1 & \text{if } i = 1, 2, 9, 16, \\ 2 & \text{otherwise.} \end{cases}$
  - Cyclically left shift C_{i-1} by s bits to get C_i.
  - Cyclically left shift D_{i-1} by s bits to get D_i.

DES Key Schedule

Input: A DES key $K = k_1 k_2 \ldots k_{64}$.
Output: Sixteen 48-bit round keys $K_1, K_2, \ldots, K_{16}$.

- Generate 56-bit permuted key $U_0 = \text{PC1}(K) = k_{57} k_{49} k_{41} \ldots k_{12} k_4$.
- Break U_0 in two 28-bit parts: $U_0 = C_0 \mid\mid D_0$.
- for $i = 1, 2, \ldots, 16$, repeat the following steps:
  - Take $s := \begin{cases} 1 & \text{if } i = 1, 2, 9, 16, \\ 2 & \text{otherwise.} \end{cases}$
  - Cyclically left shift C_{i-1} by s bits to get C_i.
  - Cyclically left shift D_{i-1} by s bits to get D_i.
  - Let $U_i := C_i \mid\mid D_i = u_1 u_2 \ldots u_{56}$.

DES Key Schedule

Input: A DES key $K = k_1 k_2 \ldots k_{64}$.
Output: Sixteen 48-bit round keys $K_1, K_2, \ldots, K_{16}$.

- Generate 56-bit permuted key $U_0 = \mathrm{PC1}(K) = k_{57} k_{49} k_{41} \ldots k_{12} k_4$.
- Break U_0 in two 28-bit parts: $U_0 = C_0 \,||\, D_0$.
- for $i = 1, 2, \ldots, 16$, repeat the following steps:
  - Take $s := \begin{cases} 1 & \text{if } i = 1, 2, 9, 16, \\ 2 & \text{otherwise.} \end{cases}$
  - Cyclically left shift C_{i-1} by s bits to get C_i.
  - Cyclically left shift D_{i-1} by s bits to get D_i.
  - Let $U_i := C_i \,||\, D_i = u_1 u_2 \ldots u_{56}$.
  - Compute 48-bit round key $K_i = \mathrm{PC2}(U_i) = u_{14} u_{17} u_{11} \ldots u_{29} u_{32}$.

DES Key Schedule (contd)

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
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

DES Encryption

Input: Plaintext block $M = m_1 m_2 \ldots m_{64}$ and round keys $K_1, K_2, \ldots, K_{16}$.

Output: The ciphertext block C.

DES Encryption

Input: Plaintext block $M = m_1 m_2 \ldots m_{64}$ and round keys $K_1, K_2, \ldots, K_{16}$.

Output: The ciphertext block C.

- Apply initial permutation: $V = \mathrm{IP}(M) = m_{58} m_{50} m_{42} \ldots m_{15} m_7$.

DES Encryption

Input: Plaintext block $M = m_1 m_2 \ldots m_{64}$ and round keys $K_1, K_2, \ldots, K_{16}$.

Output: The ciphertext block C.

- Apply initial permutation: $V = \mathrm{IP}(M) = m_{58} m_{50} m_{42} \ldots m_{15} m_7$.
- Break V in two 32-bit parts: $V = L_0 \,||\, R_0$.

DES Encryption

Input: Plaintext block $M = m_1 m_2 \ldots m_{64}$ and round keys $K_1, K_2, \ldots, K_{16}$.

Output: The ciphertext block C.

- Apply initial permutation: $V = \mathrm{IP}(M) = m_{58} m_{50} m_{42} \ldots m_{15} m_7$.
- Break V in two 32-bit parts: $V = L_0 \,||\, R_0$.
- For $i = 1, 2, \ldots, 16$, repeat the following steps:

DES Encryption

Input: Plaintext block $M = m_1 m_2 \ldots m_{64}$ and round keys $K_1, K_2, \ldots, K_{16}$.

Output: The ciphertext block C.

- Apply initial permutation: $V = \mathrm{IP}(M) = m_{58} m_{50} m_{42} \ldots m_{15} m_7$.
- Break V in two 32-bit parts: $V = L_0 \,||\, R_0$.
- For $i = 1, 2, \ldots, 16$, repeat the following steps:
  - $L_i := R_{i-1}$.

DES Encryption

Input: Plaintext block $M = m_1 m_2 \ldots m_{64}$ and round keys $K_1, K_2, \ldots, K_{16}$.

Output: The ciphertext block C.

- Apply initial permutation: $V = \mathrm{IP}(M) = m_{58} m_{50} m_{42} \ldots m_{15} m_7$.
- Break V in two 32-bit parts: $V = L_0 \,||\, R_0$.
- For $i = 1, 2, \ldots, 16$, repeat the following steps:
  - $L_i := R_{i-1}$.
  - $R_i := L_{i-1} \oplus e(R_{i-1}, K_i)$.

DES Encryption

Input: Plaintext block $M = m_1 m_2 \ldots m_{64}$ and round keys $K_1, K_2, \ldots, K_{16}$.

Output: The ciphertext block C.

- Apply initial permutation: $V = \mathrm{IP}(M) = m_{58} m_{50} m_{42} \ldots m_{15} m_7$.
- Break V in two 32-bit parts: $V = L_0 \mid\mid R_0$.
- For $i = 1, 2, \ldots, 16$, repeat the following steps:
  - $L_i := R_{i-1}$.
  - $R_i := L_{i-1} \oplus e(R_{i-1}, K_i)$.
- Let $W = R_{16} \mid\mid L_{16} = w_1 w_2 \ldots w_{64}$.

DES Encryption

Input: Plaintext block $M = m_1 m_2 \ldots m_{64}$ and round keys $K_1, K_2, \ldots, K_{16}$.

Output: The ciphertext block C.

- Apply initial permutation: $V = \text{IP}(M) = m_{58} m_{50} m_{42} \ldots m_{15} m_7$.
- Break V in two 32-bit parts: $V = L_0 \,||\, R_0$.
- For $i = 1, 2, \ldots, 16$, repeat the following steps:
  - $L_i := R_{i-1}$.
  - $R_i := L_{i-1} \oplus e(R_{i-1}, K_i)$.
- Let $W = R_{16} \,||\, L_{16} = w_1 w_2 \ldots w_{64}$.
- Apply inverse of IP: $C = \text{IP}^{-1}(W) = w_{40} w_8 w_{48} \ldots w_{57} w_{25}$.

DES Encryption (contd)

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

DES Encryption (contd)

Encryption primitive

$$e(X, J) = \mathrm{P}(S(\mathrm{E}(X) \oplus J)),$$

where X is a 32-bit message block, and J is a 48-bit round key.

DES Encryption (contd)

Encryption primitive

$$e(X, J) = \mathrm{P}(S(\mathrm{E}(X) \oplus J)),$$

where X is a 32-bit message block, and J is a 48-bit round key.

- Apply 32-to-48 bit expansion: $X' = \mathrm{E}(X) = x_{32}x_1x_2 \ldots x_{32}x_1$.

DES Encryption (contd)

Encryption primitive

$$e(X, J) = \mathrm{P}(S(\mathrm{E}(X) \oplus J)),$$

where X is a 32-bit message block, and J is a 48-bit round key.

- Apply 32-to-48 bit expansion: $X' = \mathrm{E}(X) = x_{32}x_1x_2 \ldots x_{32}x_1$.
- XOR with the round key: $Y = X' \oplus J$.

DES Encryption (contd)

Encryption primitive

$$e(X, J) = \mathrm{P}(S(\mathrm{E}(X) \oplus J)),$$

where X is a 32-bit message block, and J is a 48-bit round key.

- Apply 32-to-48 bit expansion: $X' = \mathrm{E}(X) = x_{32}x_1x_2 \ldots x_{32}x_1$.
- XOR with the round key: $Y = X' \oplus J$.
- Break Y in eight 6-bit parts: $Y = Y_1 \mid\mid Y_2 \mid\mid \cdots \mid\mid Y_8$.

DES Encryption (contd)

Encryption primitive

$$e(X, J) = \mathrm{P}(S(\mathrm{E}(X) \oplus J)),$$

where X is a 32-bit message block, and J is a 48-bit round key.

- Apply 32-to-48 bit expansion: $X' = \mathrm{E}(X) = x_{32}x_1x_2 \ldots x_{32}x_1$.
- XOR with the round key: $Y = X' \oplus J$.
- Break Y in eight 6-bit parts: $Y = Y_1 \mid\mid Y_2 \mid\mid \cdots \mid\mid Y_8$.
- For $j = 1, 2, \ldots, 8$, do the following:

DES Encryption (contd)

Encryption primitive

$$e(X, J) = \mathrm{P}(S(\mathrm{E}(X) \oplus J)),$$

where X is a 32-bit message block, and J is a 48-bit round key.

- Apply 32-to-48 bit expansion: $X' = \mathrm{E}(X) = x_{32}x_1x_2 \ldots x_{32}x_1$.
- XOR with the round key: $Y = X' \oplus J$.
- Break Y in eight 6-bit parts: $Y = Y_1 \mid\mid Y_2 \mid\mid \cdots \mid\mid Y_8$.
- For $j = 1, 2, \ldots, 8$, do the following:
  - Write $Y_j = y_1y_2y_3y_4y_5y_6$.

DES Encryption (contd)

Encryption primitive

$$e(X, J) = \mathrm{P}(S(\mathrm{E}(X) \oplus J)),$$

where X is a 32-bit message block, and J is a 48-bit round key.

- Apply 32-to-48 bit expansion: $X' = \mathrm{E}(X) = x_{32}x_1x_2 \ldots x_{32}x_1$.
- XOR with the round key: $Y = X' \oplus J$.
- Break Y in eight 6-bit parts: $Y = Y_1 \mid\mid Y_2 \mid\mid \cdots \mid\mid Y_8$.
- For $j = 1, 2, \ldots, 8$, do the following:
  - Write $Y_j = y_1y_2y_3y_4y_5y_6$.
  - Consider the integers $\mu = (y_1y_6)_2$ and $\nu = (y_2y_3y_4y_5)_2$.

DES Encryption (contd)

Encryption primitive

$$e(X, J) = \mathrm{P}(S(\mathrm{E}(X) \oplus J)),$$

where X is a 32-bit message block, and J is a 48-bit round key.

- Apply 32-to-48 bit expansion: $X' = \mathrm{E}(X) = x_{32}x_1x_2 \ldots x_{32}x_1$.
- XOR with the round key: $Y = X' \oplus J$.
- Break Y in eight 6-bit parts: $Y = Y_1 \,||\, Y_2 \,||\, \cdots \,||\, Y_8$.
- For $j = 1, 2, \ldots, 8$, do the following:
  - Write $Y_j = y_1y_2y_3y_4y_5y_6$.
  - Consider the integers $\mu = (y_1y_6)_2$ and $\nu = (y_2y_3y_4y_5)_2$.
  - Let $Z_j = (z_1z_2z_3z_4)_2 = S_j(\mu, \nu)$.

DES Encryption (contd)

Encryption primitive

$$e(X, J) = \mathrm{P}(S(\mathrm{E}(X) \oplus J)),$$

where X is a 32-bit message block, and J is a 48-bit round key.

- Apply 32-to-48 bit expansion: $X' = \mathrm{E}(X) = x_{32}x_1x_2 \ldots x_{32}x_1$.
- XOR with the round key: $Y = X' \oplus J$.
- Break Y in eight 6-bit parts: $Y = Y_1 \mid\mid Y_2 \mid\mid \cdots \mid\mid Y_8$.
- For $j = 1, 2, \ldots, 8$, do the following:
  - Write $Y_j = y_1y_2y_3y_4y_5y_6$.
  - Consider the integers $\mu = (y_1y_6)_2$ and $\nu = (y_2y_3y_4y_5)_2$.
  - Let $Z_j = (z_1z_2z_3z_4)_2 = S_j(\mu, \nu)$.
- Concatenate the Z_j's to the 32-bit value:
  $Z = Z_1 \mid\mid Z_2 \mid\mid \cdots \mid\mid Z_8 = z_1z_2 \ldots z_{32}$.

DES Encryption (contd)

Encryption primitive

$$e(X, J) = \mathrm{P}(S(\mathrm{E}(X) \oplus J)),$$

where X is a 32-bit message block, and J is a 48-bit round key.

- Apply 32-to-48 bit expansion: $X' = \mathrm{E}(X) = x_{32}x_1x_2 \ldots x_{32}x_1$.
- XOR with the round key: $Y = X' \oplus J$.
- Break Y in eight 6-bit parts: $Y = Y_1 \mid\mid Y_2 \mid\mid \cdots \mid\mid Y_8$.
- For $j = 1, 2, \ldots, 8$, do the following:
  - Write $Y_j = y_1y_2y_3y_4y_5y_6$.
  - Consider the integers $\mu = (y_1y_6)_2$ and $\nu = (y_2y_3y_4y_5)_2$.
  - Let $Z_j = (z_1z_2z_3z_4)_2 = S_j(\mu, \nu)$.
- Concatenate the Z_j's to the 32-bit value:
  $Z = Z_1 \mid\mid Z_2 \mid\mid \cdots \mid\mid Z_8 = z_1z_2 \ldots z_{32}$.
- Apply permutation function: $e(X, J) = \mathrm{P}(Z) = z_{16}z_7z_{20} \ldots z_4z_{25}$.

DES Encryption (contd)

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

DES Encryption: S-Boxes

S_1

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

S_2

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

S_3

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

DES Encryption: S-Boxes (contd)

S_4

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

S_5

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

S_6

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

DES Encryption: S-Boxes (contd)

S_7

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

S_8

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

DES Decryption

Input: Ciphertext block $C = c_1 c_2 \ldots c_{64}$ and round keys $K_1, K_2, \ldots, K_{16}$.

Output: The plaintext block M.

DES Decryption

Input: Ciphertext block $C = c_1 c_2 \ldots c_{64}$ and round keys $K_1, K_2, \ldots, K_{16}$.
Output: The plaintext block M.

- Apply initial permutation: $V = \mathrm{IP}(C) = c_{58} c_{50} c_{42} \ldots c_{15} c_7$.

DES Decryption

Input: Ciphertext block $C = c_1 c_2 \ldots c_{64}$ and round keys $K_1, K_2, \ldots, K_{16}$.

Output: The plaintext block M.

- Apply initial permutation: $V = \mathrm{IP}(C) = c_{58} c_{50} c_{42} \ldots c_{15} c_7$.
- Break V in two 32-bit parts: $V = R_{16} \mid\mid L_{16}$.

DES Decryption

Input: Ciphertext block $C = c_1 c_2 \ldots c_{64}$ and round keys $K_1, K_2, \ldots, K_{16}$.

Output: The plaintext block M.

- Apply initial permutation: $V = \mathrm{IP}(C) = c_{58} c_{50} c_{42} \ldots c_{15} c_7$.
- Break V in two 32-bit parts: $V = R_{16} \,||\, L_{16}$.
- For $i = 16, 15, \ldots, 1$, repeat the following steps:

DES Decryption

Input: Ciphertext block $C = c_1 c_2 \ldots c_{64}$ and round keys $K_1, K_2, \ldots, K_{16}$.

Output: The plaintext block M.

- Apply initial permutation: $V = \mathrm{IP}(C) = c_{58} c_{50} c_{42} \ldots c_{15} c_7$.
- Break V in two 32-bit parts: $V = R_{16} \,||\, L_{16}$.
- For $i = 16, 15, \ldots, 1$, repeat the following steps:
    - $R_{i-1} = L_i$.

DES Decryption

Input: Ciphertext block $C = c_1 c_2 \ldots c_{64}$ and round keys $K_1, K_2, \ldots, K_{16}$.

Output: The plaintext block M.

- Apply initial permutation: $V = \mathrm{IP}(C) = c_{58} c_{50} c_{42} \ldots c_{15} c_7$.
- Break V in two 32-bit parts: $V = R_{16} \,||\, L_{16}$.
- For $i = 16, 15, \ldots, 1$, repeat the following steps:
  - $R_{i-1} = L_i$.
  - $L_{i-1} = R_i \oplus e(L_i, K_i)$.

DES Decryption

Input: Ciphertext block $C = c_1 c_2 \ldots c_{64}$ and round keys $K_1, K_2, \ldots, K_{16}$.
Output: The plaintext block M.

- Apply initial permutation: $V = \mathrm{IP}(C) = c_{58} c_{50} c_{42} \ldots c_{15} c_7$.
- Break V in two 32-bit parts: $V = R_{16} \mid\mid L_{16}$.
- For $i = 16, 15, \ldots, 1$, repeat the following steps:
  - $R_{i-1} = L_i$.
  - $L_{i-1} = R_i \oplus e(L_i, K_i)$.
- Let $W = L_0 \mid\mid R_0 = w_1 w_2 \ldots w_{64}$.

DES Decryption

Input: Ciphertext block $C = c_1 c_2 \ldots c_{64}$ and round keys $K_1, K_2, \ldots, K_{16}$.
Output: The plaintext block M.

- Apply initial permutation: $V = \mathrm{IP}(C) = c_{58} c_{50} c_{42} \ldots c_{15} c_7$.
- Break V in two 32-bit parts: $V = R_{16} \,||\, L_{16}$.
- For $i = 16, 15, \ldots, 1$, repeat the following steps:
  - $R_{i-1} = L_i$.
  - $L_{i-1} = R_i \oplus e(L_i, K_i)$.
- Let $W = L_0 \,||\, R_0 = w_1 w_2 \ldots w_{64}$.
- Apply inverse of IP: $M = \mathrm{IP}^{-1}(W) = w_{40} w_8 w_{48} \ldots w_{57} w_{25}$.

DES Decryption

Input: Ciphertext block $C = c_1 c_2 \ldots c_{64}$ and round keys $K_1, K_2, \ldots, K_{16}$.

Output: The plaintext block M.

- Apply initial permutation: $V = \mathrm{IP}(C) = c_{58} c_{50} c_{42} \ldots c_{15} c_7$.
- Break V in two 32-bit parts: $V = R_{16} \,||\, L_{16}$.
- For $i = 16, 15, \ldots, 1$, repeat the following steps:
  - $R_{i-1} = L_i$.
  - $L_{i-1} = R_i \oplus e(L_i, K_i)$.
- Let $W = L_0 \,||\, R_0 = w_1 w_2 \ldots w_{64}$.
- Apply inverse of IP: $M = \mathrm{IP}^{-1}(W) = w_{40} w_8 w_{48} \ldots w_{57} w_{25}$.

Note: DES decryption is the same as DES encryption, with the key schedule reversed.

AES (Advanced Encryption Standard)

AES (Advanced Encryption Standard)

- AES is an adaptation of the Rijndael cipher designed by J. Daemen and V. Rijmen.

AES (Advanced Encryption Standard)

- AES is an adaptation of the Rijndael cipher designed by J. Daemen and V. Rijmen.
- Since DES supports short keys (56 bits) vulnerable even to brute-force search, the new standard AES is adopted in 2000.

AES (Advanced Encryption Standard)

- AES is an adaptation of the Rijndael cipher designed by J. Daemen and V. Rijmen.
- Since DES supports short keys (56 bits) vulnerable even to brute-force search, the new standard AES is adopted in 2000.
- AES is a substitution-permutation cipher.

AES (Advanced Encryption Standard)

- AES is an adaptation of the Rijndael cipher designed by J. Daemen and V. Rijmen.
- Since DES supports short keys (56 bits) vulnerable even to brute-force search, the new standard AES is adopted in 2000.
- AES is a substitution-permutation cipher.
- AES is *not* a Feistel cipher.

AES (Advanced Encryption Standard)

- AES is an adaptation of the Rijndael cipher designed by J. Daemen and V. Rijmen.
- Since DES supports short keys (56 bits) vulnerable even to brute-force search, the new standard AES is adopted in 2000.
- AES is a substitution-permutation cipher.
- AES is *not* a Feistel cipher.
- The block size for AES is $n = 128$ bits.

AES (Advanced Encryption Standard)

- AES is an adaptation of the Rijndael cipher designed by J. Daemen and V. Rijmen.
- Since DES supports short keys (56 bits) vulnerable even to brute-force search, the new standard AES is adopted in 2000.
- AES is a substitution-permutation cipher.
- AES is *not* a Feistel cipher.
- The block size for AES is $n = 128$ bits.
- Number of **rounds** for AES is $l = 10, 12,$ or 14 for key sizes $r = 128, 192,$ or 256 bits.

AES (Advanced Encryption Standard)

- AES is an adaptation of the Rijndael cipher designed by J. Daemen and V. Rijmen.
- Since DES supports short keys (56 bits) vulnerable even to brute-force search, the new standard AES is adopted in 2000.
- AES is a substitution-permutation cipher.
- AES is *not* a Feistel cipher.
- The block size for AES is $n = 128$ bits.
- Number of **rounds** for AES is $l = 10, 12$, or 14 for key sizes $r = 128, 192$, or 256 bits.
- AES **key schedule**: From K, generate 32-bit round keys $K_0, K_1, \ldots, K_{4l+3}$. Four round keys are used in a round.

AES (contd)

AES (contd)

- **State:** AES represents a 128-bit message block as a 4×4 array of octets:

$$\mu_0\mu_1\cdots\mu_{15} \equiv \begin{array}{|c|c|c|c|}\hline \mu_0 & \mu_4 & \mu_8 & \mu_{12} \\ \hline \mu_1 & \mu_5 & \mu_9 & \mu_{13} \\ \hline \mu_2 & \mu_6 & \mu_{10} & \mu_{14} \\ \hline \mu_3 & \mu_7 & \mu_{11} & \mu_{15} \\ \hline \end{array}$$

AES (contd)

- **State:** AES represents a 128-bit message block as a 4×4 array of octets:

$$\mu_0\mu_1\cdots\mu_{15} \equiv \begin{array}{|c|c|c|c|}\hline \mu_0 & \mu_4 & \mu_8 & \mu_{12} \\ \hline \mu_1 & \mu_5 & \mu_9 & \mu_{13} \\ \hline \mu_2 & \mu_6 & \mu_{10} & \mu_{14} \\ \hline \mu_3 & \mu_7 & \mu_{11} & \mu_{15} \\ \hline \end{array}$$

- Each octet $A = a_7a_6\ldots a_1a_0$ in the state is identified with the element $a_7\alpha^7 + a_6\alpha^6 + \cdots + a_1\alpha + a_0$ of $\mathbb{F}_{2^8} = \mathbb{F}_2(\alpha)$, where $\alpha^8 + \alpha^4 + \alpha^3 + \alpha + 1 = 0$.

AES (contd)

- **State:** AES represents a 128-bit message block as a 4×4 array of octets:

$$\mu_0\mu_1\ldots\mu_{15} \equiv \begin{array}{|c|c|c|c|}\hline \mu_0 & \mu_4 & \mu_8 & \mu_{12} \\ \hline \mu_1 & \mu_5 & \mu_9 & \mu_{13} \\ \hline \mu_2 & \mu_6 & \mu_{10} & \mu_{14} \\ \hline \mu_3 & \mu_7 & \mu_{11} & \mu_{15} \\ \hline \end{array}$$

- Each octet $A = a_7a_6\ldots a_1a_0$ in the state is identified with the element $a_7\alpha^7 + a_6\alpha^6 + \cdots + a_1\alpha + a_0$ of $\mathbb{F}_{2^8} = \mathbb{F}_2(\alpha)$, where $\alpha^8 + \alpha^4 + \alpha^3 + \alpha + 1 = 0$.
- Each column $A_3A_2A_1A_0$ in the state is identified with the element $A_3y^3 + A_2y^2 + A_1y + A_0$ of $\mathbb{F}_{2^8}[y]$ modulo the (reducible polynomial) $y^4 + 1$.

AES Encryption

AES Encryption

- Generate key schedule $K_0, K_1, \ldots, K_{4l+3}$ from the key K.

AES Encryption

- Generate key schedule $K_0, K_1, \ldots, K_{4l+3}$ from the key K.
- Convert the plaintext block M to a state S.

AES Encryption

- Generate key schedule $K_0, K_1, \ldots, K_{4l+3}$ from the key K.
- Convert the plaintext block M to a state S.
- $S = \text{AddKey}(S, K_0, K_1, K_2, K_3)$. [bitwise XOR]

AES Encryption

- Generate key schedule $K_0, K_1, \ldots, K_{4l+3}$ from the key K.
- Convert the plaintext block M to a state S.
- $S = \text{AddKey}(S, K_0, K_1, K_2, K_3)$. [bitwise XOR]
- for $i = 1, 2, \ldots, l$ do the following:
  $S = \text{SubState}(S)$. [non-linear, involves inverses in $\mathbb{F}_{2^8}$]
  $S = \text{ShiftRows}(S)$. [cyclic shift of octets in each row]
  If $i \neq l$, $S = \text{MixCols}(S)$. [operation in $\mathbb{F}_{2^8}[y] \bmod y^4 + 1$]
  $S = \text{AddKey}(S, K_{4i}, K_{4i+1}, K_{4i+2}, K_{4i+3})$. [bitwise XOR]

AES Encryption

- Generate key schedule $K_0, K_1, \ldots, K_{4l+3}$ from the key K.
- Convert the plaintext block M to a state S.
- $S = \text{AddKey}(S, K_0, K_1, K_2, K_3)$. [bitwise XOR]
- for $i = 1, 2, \ldots, l$ do the following:
  - $S = \text{SubState}(S)$. [non-linear, involves inverses in $\mathbb{F}_{2^8}$]
  - $S = \text{ShiftRows}(S)$. [cyclic shift of octets in each row]
  - If $i \neq l$, $S = \text{MixCols}(S)$. [operation in $\mathbb{F}_{2^8}[y] \bmod y^4 + 1$]
  - $S = \text{AddKey}(S, K_{4i}, K_{4i+1}, K_{4i+2}, K_{4i+3})$. [bitwise XOR]
- Convert the state S to the ciphertext block C.

AES Decryption

AES Decryption

- Generate key schedule $K_0, K_1, \ldots, K_{4l+3}$ from the key K.

AES Decryption

- Generate key schedule $K_0, K_1, \ldots, K_{4l+3}$ from the key K.
- Convert the ciphertext block C to a state S.

AES Decryption

- Generate key schedule $K_0, K_1, \ldots, K_{4l+3}$ from the key K.
- Convert the ciphertext block C to a state S.
- $S = \text{AddKey}(S, K_{4l}, K_{4l+1}, K_{4l+2}, K_{4l+3})$.

AES Decryption

- Generate key schedule $K_0, K_1, \ldots, K_{4l+3}$ from the key K.
- Convert the ciphertext block C to a state S.
- $S = \text{AddKey}(S, K_{4l}, K_{4l+1}, K_{4l+2}, K_{4l+3})$.
- for $i = l-1, l-2, \ldots, 1, 0$ do the following:
  $S = \text{ShiftRows}^{-1}(S)$.
  $S = \text{SubState}^{-1}(S)$.
  $S = \text{AddKey}(S, K_{4i}, K_{4i+1}, K_{4i+2}, K_{4i+3})$.
  If $i \neq 0$, $S = \text{MixCols}^{-1}(S)$.

AES Decryption

- Generate key schedule $K_0, K_1, \ldots, K_{4l+3}$ from the key K.
- Convert the ciphertext block C to a state S.
- $S = \text{AddKey}(S, K_{4l}, K_{4l+1}, K_{4l+2}, K_{4l+3})$.
- for $i = l-1, l-2, \ldots, 1, 0$ do the following:
  $S = \text{ShiftRows}^{-1}(S)$.
  $S = \text{SubState}^{-1}(S)$.
  $S = \text{AddKey}(S, K_{4i}, K_{4i+1}, K_{4i+2}, K_{4i+3})$.
  If $i \neq 0$, $S = \text{MixCols}^{-1}(S)$.
- Convert the state S to the plaintext block M.

AES: The AddKey Primitive

AES: The AddKey Primitive

- Let $S = (\sigma_{uv}) = \begin{array}{|c|c|c|c|}\hline \sigma_{00} & \sigma_{01} & \sigma_{02} & \sigma_{03} \\ \hline \sigma_{10} & \sigma_{11} & \sigma_{12} & \sigma_{13} \\ \hline \sigma_{20} & \sigma_{21} & \sigma_{22} & \sigma_{23} \\ \hline \sigma_{30} & \sigma_{31} & \sigma_{32} & \sigma_{33} \\ \hline \end{array}$ be a state of AES.

AES: The AddKey Primitive

- Let $S = (\sigma_{uv}) = \begin{array}{|c|c|c|c|} \hline \sigma_{00} & \sigma_{01} & \sigma_{02} & \sigma_{03} \\ \hline \sigma_{10} & \sigma_{11} & \sigma_{12} & \sigma_{13} \\ \hline \sigma_{20} & \sigma_{21} & \sigma_{22} & \sigma_{23} \\ \hline \sigma_{30} & \sigma_{31} & \sigma_{32} & \sigma_{33} \\ \hline \end{array}$ be a state of AES.
- Let the four 32-bit round keys be L_0, L_1, L_2, L_3 with the octet representation $L_u = \lambda_{u0}\lambda_{u1}\lambda_{u2}\lambda_{u3}$.

AES: The AddKey Primitive

- Let $S = (\sigma_{uv}) = \begin{array}{|c|c|c|c|} \hline \sigma_{00} & \sigma_{01} & \sigma_{02} & \sigma_{03} \\ \hline \sigma_{10} & \sigma_{11} & \sigma_{12} & \sigma_{13} \\ \hline \sigma_{20} & \sigma_{21} & \sigma_{22} & \sigma_{23} \\ \hline \sigma_{30} & \sigma_{31} & \sigma_{32} & \sigma_{33} \\ \hline \end{array}$ be a state of AES.
- Let the four 32-bit round keys be L_0, L_1, L_2, L_3 with the octet representation $L_u = \lambda_{u0}\lambda_{u1}\lambda_{u2}\lambda_{u3}$.
- The u-th key L_u is XORed with the u-th column of the state S.

AES: The AddKey Primitive

- Let $S = (\sigma_{uv}) = \begin{array}{|c|c|c|c|}\hline \sigma_{00} & \sigma_{01} & \sigma_{02} & \sigma_{03} \\ \hline \sigma_{10} & \sigma_{11} & \sigma_{12} & \sigma_{13} \\ \hline \sigma_{20} & \sigma_{21} & \sigma_{22} & \sigma_{23} \\ \hline \sigma_{30} & \sigma_{31} & \sigma_{32} & \sigma_{33} \\ \hline \end{array}$ be a state of AES.
- Let the four 32-bit round keys be L_0, L_1, L_2, L_3 with the octet representation $L_u = \lambda_{u0}\lambda_{u1}\lambda_{u2}\lambda_{u3}$.
- The u-th key L_u is XORed with the u-th column of the state S.
- S maps to $\text{AddKey}(S, L_0, L_1, L_2, L_3) =$

$$\begin{array}{|c|c|c|c|}\hline \sigma_{00} \oplus \lambda_{00} & \sigma_{01} \oplus \lambda_{10} & \sigma_{02} \oplus \lambda_{20} & \sigma_{03} \oplus \lambda_{30} \\ \hline \sigma_{10} \oplus \lambda_{01} & \sigma_{11} \oplus \lambda_{11} & \sigma_{12} \oplus \lambda_{21} & \sigma_{13} \oplus \lambda_{31} \\ \hline \sigma_{20} \oplus \lambda_{02} & \sigma_{21} \oplus \lambda_{12} & \sigma_{22} \oplus \lambda_{22} & \sigma_{23} \oplus \lambda_{32} \\ \hline \sigma_{30} \oplus \lambda_{03} & \sigma_{31} \oplus \lambda_{13} & \sigma_{32} \oplus \lambda_{23} & \sigma_{33} \oplus \lambda_{33} \\ \hline \end{array}.$$

AES: The SubState Primitive

AES: The SubState Primitive

- Let $A = a_0 a_1 \ldots a_6 a_7$ be an octet (an element of $\mathbb{F}_{2^8}$).

AES: The SubState Primitive

- Let $A = a_0 a_1 \ldots a_6 a_7$ be an octet (an element of $\mathbb{F}_{2^8}$).
- Let $B = b_0 b_1 \ldots b_6 b_7 = A^{-1}$ in $\mathbb{F}_{2^8}$ (with $0^{-1} = 0$).

AES: The SubState Primitive

- Let $A = a_0 a_1 \ldots a_6 a_7$ be an octet (an element of $\mathbb{F}_{2^8}$).
- Let $B = b_0 b_1 \ldots b_6 b_7 = A^{-1}$ in $\mathbb{F}_{2^8}$ (with $0^{-1} = 0$).
- Let $D = d_0 d_1 \ldots d_6 d_7 = 63 = 01100011$.

AES: The SubState Primitive

- Let $A = a_0a_1\ldots a_6a_7$ be an octet (an element of $\mathbb{F}_{2^8}$).
- Let $B = b_0b_1\ldots b_6b_7 = A^{-1}$ in $\mathbb{F}_{2^8}$ (with $0^{-1} = 0$).
- Let $D = d_0d_1\ldots d_6d_7 = 63 = 01100011$.
- SubOctet$(A) = C = c_0c_1\ldots c_6c_7$, where
  $c_i = b_i \oplus b_{(i+1)\mathrm{rem}8} \oplus b_{(i+2)\mathrm{rem}8} \oplus b_{(i+3)\mathrm{rem}8} \oplus b_{(i+4)\mathrm{rem}8} \oplus d_i$.

AES: The SubState Primitive

- Let $A = a_0a_1 \ldots a_6a_7$ be an octet (an element of $\mathbb{F}_{2^8}$).
- Let $B = b_0b_1 \ldots b_6b_7 = A^{-1}$ in $\mathbb{F}_{2^8}$ (with $0^{-1} = 0$).
- Let $D = d_0d_1 \ldots d_6d_7 = 63 = 01100011$.
- SubOctet$(A) = C = c_0c_1 \ldots c_6c_7$, where
  $c_i = b_i \oplus b_{(i+1)\mathrm{rem}8} \oplus b_{(i+2)\mathrm{rem}8} \oplus b_{(i+3)\mathrm{rem}8} \oplus b_{(i+4)\mathrm{rem}8} \oplus d_i$.
- Let $S = \begin{array}{|c|c|c|c|}\hline \sigma_{00} & \sigma_{01} & \sigma_{02} & \sigma_{03} \\ \hline \sigma_{10} & \sigma_{11} & \sigma_{12} & \sigma_{13} \\ \hline \sigma_{20} & \sigma_{21} & \sigma_{22} & \sigma_{23} \\ \hline \sigma_{30} & \sigma_{31} & \sigma_{32} & \sigma_{33} \\ \hline \end{array}$ be a state.

AES: The SubState Primitive

- Let $A = a_0 a_1 \ldots a_6 a_7$ be an octet (an element of $\mathbb{F}_{2^8}$).
- Let $B = b_0 b_1 \ldots b_6 b_7 = A^{-1}$ in $\mathbb{F}_{2^8}$ (with $0^{-1} = 0$).
- Let $D = d_0 d_1 \ldots d_6 d_7 = 63 = 01100011$.
- SubOctet$(A) = C = c_0 c_1 \ldots c_6 c_7$, where
  $c_i = b_i \oplus b_{(i+1)\,\mathrm{rem}\,8} \oplus b_{(i+2)\,\mathrm{rem}\,8} \oplus b_{(i+3)\,\mathrm{rem}\,8} \oplus b_{(i+4)\,\mathrm{rem}\,8} \oplus d_i$.
- Let $S = \begin{array}{|c|c|c|c|}\hline \sigma_{00} & \sigma_{01} & \sigma_{02} & \sigma_{03} \\ \hline \sigma_{10} & \sigma_{11} & \sigma_{12} & \sigma_{13} \\ \hline \sigma_{20} & \sigma_{21} & \sigma_{22} & \sigma_{23} \\ \hline \sigma_{30} & \sigma_{31} & \sigma_{32} & \sigma_{33} \\ \hline \end{array}$ be a state.
- SubState$(S) = \begin{array}{|c|c|c|c|}\hline \sigma'_{00} & \sigma'_{01} & \sigma'_{02} & \sigma'_{03} \\ \hline \sigma'_{10} & \sigma'_{11} & \sigma'_{12} & \sigma'_{13} \\ \hline \sigma'_{20} & \sigma'_{21} & \sigma'_{22} & \sigma'_{23} \\ \hline \sigma'_{30} & \sigma'_{31} & \sigma'_{32} & \sigma'_{33} \\ \hline \end{array}$, where $\sigma'_{uv} = $ SubOctet(σ_{uv}).

AES: The ShiftRows Primitive

Cyclically left rotate the r-th row by r bytes:

$$\begin{array}{|c|c|c|c|}\hline \sigma_{00} & \sigma_{01} & \sigma_{02} & \sigma_{03} \\ \hline \sigma_{10} & \sigma_{11} & \sigma_{12} & \sigma_{13} \\ \hline \sigma_{20} & \sigma_{21} & \sigma_{22} & \sigma_{23} \\ \hline \sigma_{30} & \sigma_{31} & \sigma_{32} & \sigma_{33} \\ \hline \end{array} \text{ maps to } \begin{array}{|c|c|c|c|}\hline \sigma_{00} & \sigma_{01} & \sigma_{02} & \sigma_{03} \\ \hline \sigma_{11} & \sigma_{12} & \sigma_{13} & \sigma_{10} \\ \hline \sigma_{22} & \sigma_{23} & \sigma_{20} & \sigma_{21} \\ \hline \sigma_{33} & \sigma_{30} & \sigma_{31} & \sigma_{32} \\ \hline \end{array}.$$

AES: The MixCols Primitive

AES: The MixCols Primitive

- Let $S = \begin{array}{|c|c|c|c|}\hline \sigma_{00} & \sigma_{01} & \sigma_{02} & \sigma_{03} \\ \hline \sigma_{10} & \sigma_{11} & \sigma_{12} & \sigma_{13} \\ \hline \sigma_{20} & \sigma_{21} & \sigma_{22} & \sigma_{23} \\ \hline \sigma_{30} & \sigma_{31} & \sigma_{32} & \sigma_{33} \\ \hline \end{array}$ be a state.

AES: The MixCols Primitive

- Let $S = \begin{array}{|c|c|c|c|}\hline \sigma_{00} & \sigma_{01} & \sigma_{02} & \sigma_{03} \\ \hline \sigma_{10} & \sigma_{11} & \sigma_{12} & \sigma_{13} \\ \hline \sigma_{20} & \sigma_{21} & \sigma_{22} & \sigma_{23} \\ \hline \sigma_{30} & \sigma_{31} & \sigma_{32} & \sigma_{33} \\ \hline \end{array}$ be a state.
- Each column of S is identified with an element of $\mathbb{F}_{2^8}[y]$, and is multiplied by the constant polynomial $[03]y^3 + [01]y^2 + [01]y + [02]$ modulo $y^4 + 1$.

AES: The MixCols Primitive

- Let $S = \begin{array}{|c|c|c|c|}\hline \sigma_{00} & \sigma_{01} & \sigma_{02} & \sigma_{03} \\ \hline \sigma_{10} & \sigma_{11} & \sigma_{12} & \sigma_{13} \\ \hline \sigma_{20} & \sigma_{21} & \sigma_{22} & \sigma_{23} \\ \hline \sigma_{30} & \sigma_{31} & \sigma_{32} & \sigma_{33} \\ \hline \end{array}$ be a state.
- Each column of S is identified with an element of $\mathbb{F}_{2^8}[y]$, and is multiplied by the constant polynomial $[03]y^3 + [01]y^2 + [01]y + [02]$ modulo $y^4 + 1$.
- The v-th column

$$\begin{pmatrix} \sigma_{0v} \\ \sigma_{1v} \\ \sigma_{2v} \\ \sigma_{3v} \end{pmatrix} \text{ maps to } \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \odot \begin{pmatrix} \sigma_{0v} \\ \sigma_{1v} \\ \sigma_{2v} \\ \sigma_{3v} \end{pmatrix},$$

where $\odot$ is the multiplication of $\mathbb{F}_{2^8}$.

Inverses of AES Encryption Primitives

Inverses of AES Encryption Primitives

- AddKey is the inverse of itself.

Inverses of AES Encryption Primitives

- AddKey is the inverse of itself.
- SubState^{-1} is the octet-by-octet inverse of SubOctet.

Inverses of AES Encryption Primitives

- AddKey is the inverse of itself.
- SubState^{-1} is the octet-by-octet inverse of SubOctet.
- SubOctet^{-1} involves an affine transformation followed by taking inverse in $\mathbb{F}_{2^8}$.

Inverses of AES Encryption Primitives

- AddKey is the inverse of itself.
- SubState^{-1} is the octet-by-octet inverse of SubOctet.
- SubOctet^{-1} involves an affine transformation followed by taking inverse in $\mathbb{F}_{2^8}$.
- ShiftRows^{-1} cyclically right rotates the r-th row by r bytes.

Inverses of AES Encryption Primitives

- AddKey is the inverse of itself.
- SubState^{-1} is the octet-by-octet inverse of SubOctet.
- SubOctet^{-1} involves an affine transformation followed by taking inverse in $\mathbb{F}_{2^8}$.
- ShiftRows^{-1} cyclically right rotates the r-th row by r bytes.
- MixCols^{-1} multiplies each column by the polynomial $[0b]y^3 + [0d]y^2 + [09]y + [0e]$ modulo $y^4 + 1$, with the coefficient arithmetic being that of $\mathbb{F}_{2^8}$.

AES Key Schedule

AES Key Schedule

- Let t be the key size in words ($t = 4, 6, 8$ for $r = 128, 192, 256$).

AES Key Schedule

- Let t be the key size in words ($t = 4, 6, 8$ for $r = 128, 192, 256$).
- The respective numbers of rounds are $l = 10, 12, 14$.

AES Key Schedule

- Let t be the key size in words ($t = 4, 6, 8$ for $r = 128, 192, 256$).
- The respective numbers of rounds are $l = 10, 12, 14$.
- AES key schedule generates $4(l + 1)$ 32-bit keys $K_0, K_1, \ldots, K_{4N_r+3}$ from the secret key K.

AES Key Schedule

- Let t be the key size in words ($t = 4, 6, 8$ for $r = 128, 192, 256$).
- The respective numbers of rounds are $l = 10, 12, 14$.
- AES key schedule generates $4(l + 1)$ 32-bit keys $K_0, K_1, \ldots, K_{4N_r+3}$ from the secret key K.
- Initially, $K = K_0 K_1 \ldots K_{t-1}$.

AES Key Schedule

- Let t be the key size in words ($t = 4, 6, 8$ for $r = 128, 192, 256$).
- The respective numbers of rounds are $l = 10, 12, 14$.
- AES key schedule generates $4(l + 1)$ 32-bit keys $K_0, K_1, \ldots, K_{4N_r+3}$ from the secret key K.
- Initially, $K = K_0 K_1 \ldots K_{t-1}$.
- For $i = t, t + 1, \ldots, 4l + 3$, generate K_i as follows:

AES Key Schedule

- Let t be the key size in words ($t = 4, 6, 8$ for $r = 128, 192, 256$).
- The respective numbers of rounds are $l = 10, 12, 14$.
- AES key schedule generates $4(l + 1)$ 32-bit keys $K_0, K_1, \ldots, K_{4N_r+3}$ from the secret key K.
- Initially, $K = K_0 K_1 \ldots K_{t-1}$.
- For $i = t, t + 1, \ldots, 4l + 3$, generate K_i as follows:
    - Let $K_{i-1} = \tau_0 \tau_1 \tau_2 \tau_3$ (each τ_j an octet).

AES Key Schedule

- Let t be the key size in words ($t = 4, 6, 8$ for $r = 128, 192, 256$).
- The respective numbers of rounds are $l = 10, 12, 14$.
- AES key schedule generates $4(l + 1)$ 32-bit keys $K_0, K_1, \ldots, K_{4N_r+3}$ from the secret key K.
- Initially, $K = K_0 K_1 \ldots K_{t-1}$.
- For $i = t, t + 1, \ldots, 4l + 3$, generate K_i as follows:
  - Let $K_{i-1} = \tau_0 \tau_1 \tau_2 \tau_3$ (each τ_j an octet).
  - Let $\tau_j' = \text{SubOctet}(\tau_j)$.

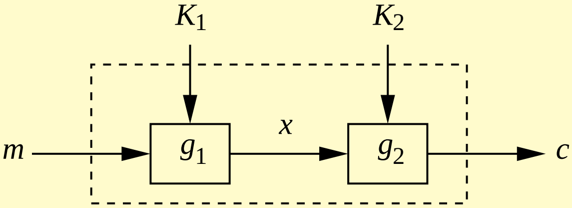
AES Key Schedule

- Let t be the key size in words ($t = 4, 6, 8$ for $r = 128, 192, 256$).
- The respective numbers of rounds are $l = 10, 12, 14$.
- AES key schedule generates $4(l + 1)$ 32-bit keys $K_0, K_1, \ldots, K_{4N_r+3}$ from the secret key K.
- Initially, $K = K_0 K_1 \ldots K_{t-1}$.
- For $i = t, t + 1, \ldots, 4l + 3$, generate K_i as follows:
  - Let $K_{i-1} = \tau_0 \tau_1 \tau_2 \tau_3$ (each τ_j an octet).
  - Let $\tau'_j = \text{SubOctet}(\tau_j)$.
  - If ($i \equiv 0 \pmod t$), then
    set $T = (\tau'_1 \tau'_2 \tau'_3 \tau'_0) \oplus [\alpha^{(i/t)-1} \,||\, 000000]$,
    else if ($t > 6$) and ($i \equiv 4 \pmod t$), then
    set $T = \tau'_0 \tau'_1 \tau'_2 \tau'_3$.

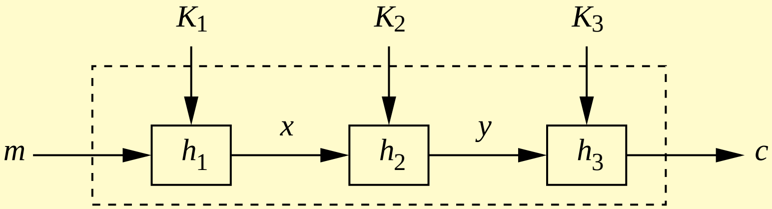
AES Key Schedule

- Let t be the key size in words ($t = 4, 6, 8$ for $r = 128, 192, 256$).
- The respective numbers of rounds are $l = 10, 12, 14$.
- AES key schedule generates $4(l+1)$ 32-bit keys $K_0, K_1, \ldots, K_{4N_r+3}$ from the secret key K.
- Initially, $K = K_0 K_1 \ldots K_{t-1}$.
- For $i = t, t+1, \ldots, 4l+3$, generate K_i as follows:
  - Let $K_{i-1} = \tau_0 \tau_1 \tau_2 \tau_3$ (each τ_j an octet).
  - Let $\tau'_j = \text{SubOctet}(\tau_j)$.
  - If ($i \equiv 0 \pmod{t}$), then
    set $T = (\tau'_1 \tau'_2 \tau'_3 \tau'_0) \oplus [\alpha^{(i/t)-1} \,||\, 000000]$,
    else if ($t > 6$) and ($i \equiv 4 \pmod{t}$), then
    set $T = \tau'_0 \tau'_1 \tau'_2 \tau'_3$.
  - Generate the 32-bit key $K_i = K_{i-t} \oplus T$.

Multiple Encryption



(a) Double encryption



(b) Triple encryption

Modes of Operation: Encryption

Modes of Operation: Encryption

- Break the message $M = M_1 M_2 \ldots M_l$ into blocks of bit length $n' \leqslant n$.

Modes of Operation: Encryption

- Break the message $M = M_1 M_2 \ldots M_l$ into blocks of bit length $n' \leqslant n$.
- To generate the ciphertext $C = C_1 C_2 \ldots C_l$.

Modes of Operation: Encryption

- Break the message $M = M_1M_2 \ldots M_l$ into blocks of bit length $n' \leqslant n$.
- To generate the ciphertext $C = C_1C_2 \ldots C_l$.
- **ECB (Electronic Code-Book) mode:** Here $n' = n$.
  $C_i = f_K(M_i)$.

Modes of Operation: Encryption

- Break the message $M = M_1 M_2 \ldots M_l$ into blocks of bit length $n' \leqslant n$.
- To generate the ciphertext $C = C_1 C_2 \ldots C_l$.
- **ECB (Electronic Code-Book) mode:** Here $n' = n$.
  $C_i = f_K(M_i)$.
- **CBC (Cipher-Block Chaining) mode:** Here $n' = n$. Set $C_0 = \text{IV}$.
  $C_i = f_K(M_i \oplus C_{i-1})$.

Modes of Operation: Encryption

- Break the message $M = M_1 M_2 \ldots M_l$ into blocks of bit length $n' \leqslant n$.
- To generate the ciphertext $C = C_1 C_2 \ldots C_l$.
- **ECB (Electronic Code-Book) mode:** Here $n' = n$.
  $C_i = f_K(M_i)$.
- **CBC (Cipher-Block Chaining) mode:** Here $n' = n$. Set $C_0 = \text{IV}$.
  $C_i = f_K(M_i \oplus C_{i-1})$.
- **CFB (Cipher FeedBack) Mode:** Here $n' \leqslant n$. Set $k_0 = \text{IV}$.
  $C_i = M_i \oplus \text{msb}_{n'}(f_K(k_{i-1}))$. [Mask key and plaintext]
  $k_i = \text{lsb}_{n-n'}(k_{i-1}) \,||\, C_i$. [Generate next key]

Modes of Operation: Encryption

- Break the message $M = M_1 M_2 \ldots M_l$ into blocks of bit length $n' \leqslant n$.
- To generate the ciphertext $C = C_1 C_2 \ldots C_l$.
- **ECB (Electronic Code-Book) mode:** Here $n' = n$.
  $C_i = f_K(M_i)$.
- **CBC (Cipher-Block Chaining) mode:** Here $n' = n$. Set $C_0 = \text{IV}$.
  $C_i = f_K(M_i \oplus C_{i-1})$.
- **CFB (Cipher FeedBack) Mode:** Here $n' \leqslant n$. Set $k_0 = \text{IV}$.
  $C_i = M_i \oplus \text{msb}_{n'}(f_K(k_{i-1}))$. [Mask key and plaintext]
  $k_i = \text{lsb}_{n-n'}(k_{i-1}) \mid\mid C_i$. [Generate next key]
- **OFB (Output FeedBack) Mode:** Here $n' \leqslant n$. Set $k_0 = \text{IV}$.
  $k_i = f_K(k_{i-1})$. [Generate next key]
  $C_i = M_i \oplus \text{msb}_{n'}(k_i)$. [Mask plaintext block]

Modes of Operation: Encryption

- Break the message $M = M_1 M_2 \ldots M_l$ into blocks of bit length $n' \leqslant n$.
- To generate the ciphertext $C = C_1 C_2 \ldots C_l$.
- **ECB (Electronic Code-Book) mode:** Here $n' = n$.
  $C_i = f_K(M_i)$.
- **CBC (Cipher-Block Chaining) mode:** Here $n' = n$. Set $C_0 = \text{IV}$.
  $C_i = f_K(M_i \oplus C_{i-1})$.
- **CFB (Cipher FeedBack) Mode:** Here $n' \leqslant n$. Set $k_0 = \text{IV}$.
  $C_i = M_i \oplus \text{msb}_{n'}(f_K(k_{i-1}))$. [Mask key and plaintext]
  $k_i = \text{lsb}_{n-n'}(k_{i-1}) \,||\, C_i$. [Generate next key]
- **OFB (Output FeedBack) Mode:** Here $n' \leqslant n$. Set $k_0 = \text{IV}$.
  $k_i = f_K(k_{i-1})$. [Generate next key]
  $C_i = M_i \oplus \text{msb}_{n'}(k_i)$. [Mask plaintext block]
- CFB and OFB modes act like stream ciphers

Modes of Operation: Decryption

Modes of Operation: Decryption

- **ECB (Electronic Code-Book) mode:**
  $M_i = f_K^{-1}(C_i)$.

Modes of Operation: Decryption

- **ECB (Electronic Code-Book) mode:**
  $M_i = f_K^{-1}(C_i)$.
- **CBC (Cipher-Block Chaining) mode:** Set $C_0 = \text{IV}$.
  $M_i = f_K^{-1}(C_i) \oplus C_{i-1}$.

Modes of Operation: Decryption

- **ECB (Electronic Code-Book) mode:**
  $M_i = f_K^{-1}(C_i)$.
- **CBC (Cipher-Block Chaining) mode:** Set $C_0 = \text{IV}$.
  $M_i = f_K^{-1}(C_i) \oplus C_{i-1}$.
- **CFB (Cipher FeedBack) Mode:** Set $k_0 = \text{IV}$.
  $M_i = C_i \oplus \text{msb}_{n'}(f_K(k_{i-1}))$. [Remove mask from ciphertext]
  $k_i = \text{lsb}_{n-n'}(k_{i-1}) \,||\, C_i$. [Generate next key]

Modes of Operation: Decryption

- **ECB (Electronic Code-Book) mode:**
  $M_i = f_K^{-1}(C_i)$.
- **CBC (Cipher-Block Chaining) mode:** Set $C_0 = \text{IV}$.
  $M_i = f_K^{-1}(C_i) \oplus C_{i-1}$.
- **CFB (Cipher FeedBack) Mode:** Set $k_0 = \text{IV}$.
  $M_i = C_i \oplus \text{msb}_{n'}(f_K(k_{i-1}))$. [Remove mask from ciphertext]
  $k_i = \text{lsb}_{n-n'}(k_{i-1}) \,||\, C_i$. [Generate next key]
- **OFB (Output FeedBack) Mode:** Set $k_0 = \text{IV}$.
  $k_i = f_K(k_{i-1})$. [Generate next key]
  $M_i = C_i \oplus \text{msb}_{n'}(k_i)$. [Remove mask from ciphertext]

Attacks on Block Ciphers

Attacks on Block Ciphers

- **Exhaustive key search**

Attacks on Block Ciphers

- **Exhaustive key search**
  - If the key space is small, all possibilities for an unknown key can be matched against known plaintext-ciphertext pairs.

Attacks on Block Ciphers

- **Exhaustive key search**
  - If the key space is small, all possibilities for an unknown key can be matched against known plaintext-ciphertext pairs.
  - Many DES challenges are cracked by exhaustive key search. DES has a small key-size (56 bits).

Attacks on Block Ciphers

- **Exhaustive key search**
  - If the key space is small, all possibilities for an unknown key can be matched against known plaintext-ciphertext pairs.
  - Many DES challenges are cracked by exhaustive key search. DES has a small key-size (56 bits).
  - Only two plaintext-ciphertext pairs usually suffice to determine a DES key uniquely.

Attacks on Block Ciphers

- **Exhaustive key search**
    - If the key space is small, all possibilities for an unknown key can be matched against known plaintext-ciphertext pairs.
    - Many DES challenges are cracked by exhaustive key search. DES has a small key-size (56 bits).
    - Only two plaintext-ciphertext pairs usually suffice to determine a DES key uniquely.
    - Exhaustive key search on block ciphers (like AES) with key sizes $\geqslant 128$ is infeasible.

Attacks on Block Ciphers

- **Exhaustive key search**
  - If the key space is small, all possibilities for an unknown key can be matched against known plaintext-ciphertext pairs.
  - Many DES challenges are cracked by exhaustive key search. DES has a small key-size (56 bits).
  - Only two plaintext-ciphertext pairs usually suffice to determine a DES key uniquely.
  - Exhaustive key search on block ciphers (like AES) with key sizes $\geqslant 128$ is infeasible.
- **Linear and differential cryptanalysis**

Attacks on Block Ciphers

- **Exhaustive key search**
  - If the key space is small, all possibilities for an unknown key can be matched against known plaintext-ciphertext pairs.
  - Many DES challenges are cracked by exhaustive key search. DES has a small key-size (56 bits).
  - Only two plaintext-ciphertext pairs usually suffice to determine a DES key uniquely.
  - Exhaustive key search on block ciphers (like AES) with key sizes $\geqslant 128$ is infeasible.
- **Linear and differential cryptanalysis**
  - By far the most sophisticated attacks on block ciphers.

Attacks on Block Ciphers

- **Exhaustive key search**
  - If the key space is small, all possibilities for an unknown key can be matched against known plaintext-ciphertext pairs.
  - Many DES challenges are cracked by exhaustive key search. DES has a small key-size (56 bits).
  - Only two plaintext-ciphertext pairs usually suffice to determine a DES key uniquely.
  - Exhaustive key search on block ciphers (like AES) with key sizes $\geqslant 128$ is infeasible.
- **Linear and differential cryptanalysis**
  - By far the most sophisticated attacks on block ciphers.
  - Impractical if sufficiently many rounds are used.

Attacks on Block Ciphers

- **Exhaustive key search**
  - If the key space is small, all possibilities for an unknown key can be matched against known plaintext-ciphertext pairs.
  - Many DES challenges are cracked by exhaustive key search. DES has a small key-size (56 bits).
  - Only two plaintext-ciphertext pairs usually suffice to determine a DES key uniquely.
  - Exhaustive key search on block ciphers (like AES) with key sizes $\geqslant 128$ is infeasible.
- **Linear and differential cryptanalysis**
  - By far the most sophisticated attacks on block ciphers.
  - Impractical if sufficiently many rounds are used.
  - AES is robust against these attacks.

Attacks on Block Ciphers (contd)

Attacks on Block Ciphers (contd)

- **Specific attacks on AES**
  - Square attack
  - Collision attack
  - Algebraic attacks (like XSL)

Attacks on Block Ciphers (contd)

- **Specific attacks on AES**
  - Square attack
  - Collision attack
  - Algebraic attacks (like XSL)
- **Meet-in-the-middle attack**

Attacks on Block Ciphers (contd)

- **Specific attacks on AES**
  - Square attack
  - Collision attack
  - Algebraic attacks (like XSL)
- **Meet-in-the-middle attack**
  - Applies to multiple encryption schemes.

Attacks on Block Ciphers (contd)

- **Specific attacks on AES**
  Square attack
  Collision attack
  Algebraic attacks (like XSL)
- **Meet-in-the-middle attack**
  - Applies to multiple encryption schemes.
  - For m stages, we get security of $\lceil m/2 \rceil$ keys only.

Stream Ciphers

Stream Ciphers

- Stream ciphers encrypt bit-by-bit.

Stream Ciphers

- Stream ciphers encrypt bit-by-bit.
- Plaintext stream: $M = m_1 m_2 \ldots m_l$.
  Key stream: $K = k_1 k_2 \ldots k_l$.
  Ciphertext stream: $C = c_1 c_2 \ldots c_l$.

Stream Ciphers

- Stream ciphers encrypt bit-by-bit.
- Plaintext stream: $M = m_1 m_2 \ldots m_l$.
  Key stream: $K = k_1 k_2 \ldots k_l$.
  Ciphertext stream: $C = c_1 c_2 \ldots c_l$.
- **Encryption:** $c_i = m_i \oplus k_i$.

Stream Ciphers

- Stream ciphers encrypt bit-by-bit.
- Plaintext stream: $M = m_1 m_2 \ldots m_l$.
  Key stream: $K = k_1 k_2 \ldots k_l$.
  Ciphertext stream: $C = c_1 c_2 \ldots c_l$.
- **Encryption:** $c_i = m_i \oplus k_i$.
- **Decryption:** $m_i = c_i \oplus k_i$.

Stream Ciphers

- Stream ciphers encrypt bit-by-bit.
- Plaintext stream: $M = m_1 m_2 \ldots m_l$.
  Key stream: $K = k_1 k_2 \ldots k_l$.
  Ciphertext stream: $C = c_1 c_2 \ldots c_l$.
- **Encryption:** $c_i = m_i \oplus k_i$.
- **Decryption:** $m_i = c_i \oplus k_i$.
- Source of security: unpredictability in the key-stream.

Stream Ciphers

- Stream ciphers encrypt bit-by-bit.
- Plaintext stream: $M = m_1 m_2 \ldots m_l$.
  Key stream: $K = k_1 k_2 \ldots k_l$.
  Ciphertext stream: $C = c_1 c_2 \ldots c_l$.
- **Encryption:** $c_i = m_i \oplus k_i$.
- **Decryption:** $m_i = c_i \oplus k_i$.
- Source of security: unpredictability in the key-stream.
- **Vernam's one-time pad:** For a truly random key stream,

  $$\Pr(c_i = 0) = \Pr(c_i = 1) = \frac{1}{2}$$

  for each i, irrespective of the probabilities of the values assumed by m_i. This leads to **unconditional security**, that is, the knowledge of any number of plaintext-ciphertext bit pairs, does not help in decrypting a new ciphertext bit.

Stream Ciphers: Drawbacks

Stream Ciphers: Drawbacks

- Key stream should be as long as the message stream. Management of long key streams is difficult.

Stream Ciphers: Drawbacks

- Key stream should be as long as the message stream. Management of long key streams is difficult.
- It is difficult to generate truly random (and reproducible) key streams.

Stream Ciphers: Drawbacks

- Key stream should be as long as the message stream. Management of long key streams is difficult.
- It is difficult to generate truly random (and reproducible) key streams.
- Pseudorandom bit streams provide practical solution, but do not guarantee unconditional security.

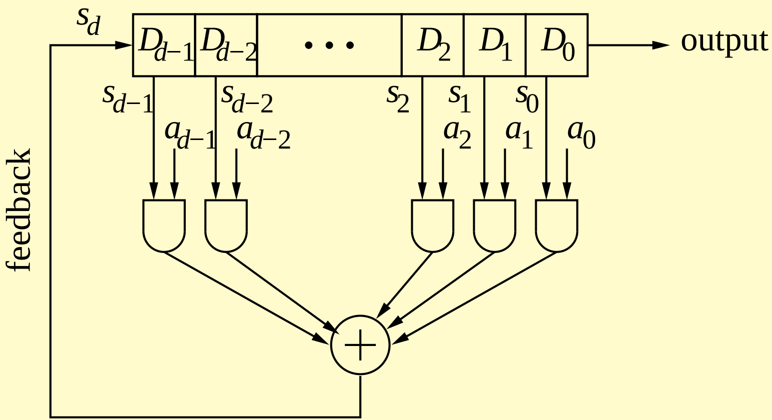
Stream Ciphers: Drawbacks

- Key stream should be as long as the message stream. Management of long key streams is difficult.
- It is difficult to generate truly random (and reproducible) key streams.
- Pseudorandom bit streams provide practical solution, but do not guarantee unconditional security.
- Pseudorandom bit generators are vulnerable to compromise of seeds.

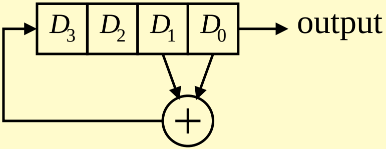
Stream Ciphers: Drawbacks

- Key stream should be as long as the message stream. Management of long key streams is difficult.
- It is difficult to generate truly random (and reproducible) key streams.
- Pseudorandom bit streams provide practical solution, but do not guarantee unconditional security.
- Pseudorandom bit generators are vulnerable to compromise of seeds.
- Repeated use of the same key stream degrades security.

Linear Feedback Shift Register (LFSR)

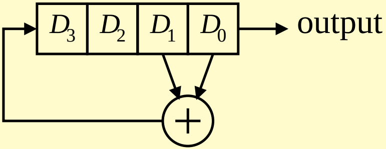


LFSR: Example



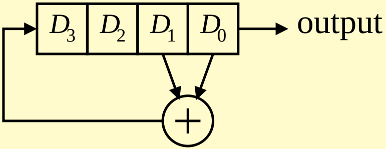
LFSR: Example

Time	D_3	D_2	D_1	D_0
0	1	1	0	1



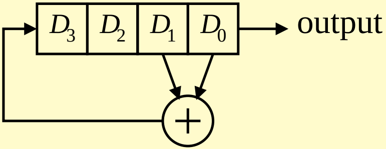
LFSR: Example

Time	D_3	D_2	D_1	D_0
0	1	1	0	1
1	1	1	1	0



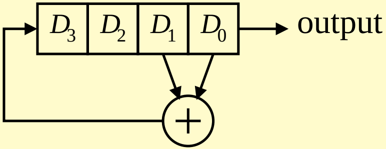
LFSR: Example

Time	D_3	D_2	D_1	D_0
0	1	1	0	1
1	1	1	1	0
2	1	1	1	1

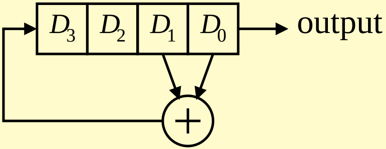


LFSR: Example

Time	D_3	D_2	D_1	D_0
0	1	1	0	1
1	1	1	1	0
2	1	1	1	1
3	0	1	1	1

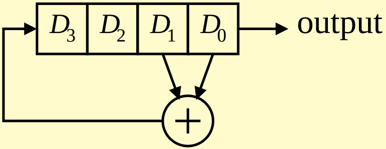


LFSR: Example



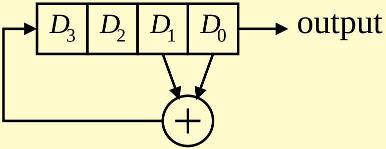
Time	D_3	D_2	D_1	D_0
0	1	1	0	1
1	1	1	1	0
2	1	1	1	1
3	0	1	1	1
4	0	0	1	1

LFSR: Example



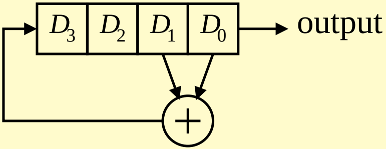
Time	D_3	D_2	D_1	D_0
0	1	1	0	1
1	1	1	1	0
2	1	1	1	1
3	0	1	1	1
4	0	0	1	1
5	0	0	0	1

LFSR: Example



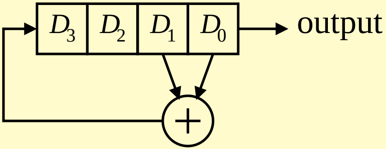
Time	D_3	D_2	D_1	D_0
0	1	1	0	1
1	1	1	1	0
2	1	1	1	1
3	0	1	1	1
4	0	0	1	1
5	0	0	0	1
6	1	0	0	0

LFSR: Example



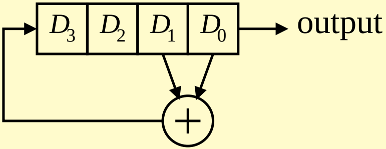
Time	D_3	D_2	D_1	D_0
0	1	1	0	1
1	1	1	1	0
2	1	1	1	1
3	0	1	1	1
4	0	0	1	1
5	0	0	0	1
6	1	0	0	0
7	0	1	0	0

LFSR: Example



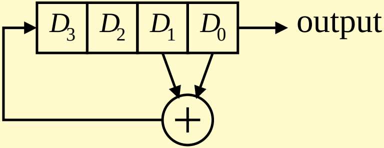
Time	D_3	D_2	D_1	D_0
0	1	1	0	1
1	1	1	1	0
2	1	1	1	1
3	0	1	1	1
4	0	0	1	1
5	0	0	0	1
6	1	0	0	0
7	0	1	0	0
8	0	0	1	0

LFSR: Example



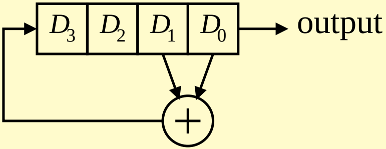
Time	D_3	D_2	D_1	D_0
0	1	1	0	1
1	1	1	1	0
2	1	1	1	1
3	0	1	1	1
4	0	0	1	1
5	0	0	0	1
6	1	0	0	0
7	0	1	0	0
8	0	0	1	0
9	1	0	0	1

LFSR: Example



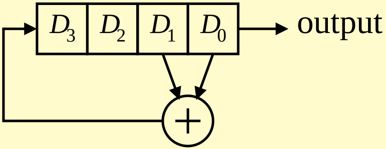
Time	D_3	D_2	D_1	D_0
0	1	1	0	1
1	1	1	1	0
2	1	1	1	1
3	0	1	1	1
4	0	0	1	1
5	0	0	0	1
6	1	0	0	0
7	0	1	0	0
8	0	0	1	0
9	1	0	0	1
10	1	1	0	0

LFSR: Example



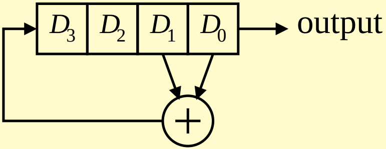
Time	D_3	D_2	D_1	D_0
0	1	1	0	1
1	1	1	1	0
2	1	1	1	1
3	0	1	1	1
4	0	0	1	1
5	0	0	0	1
6	1	0	0	0
7	0	1	0	0
8	0	0	1	0
9	1	0	0	1
10	1	1	0	0
11	0	1	1	0

LFSR: Example



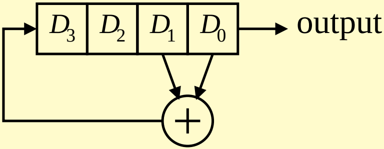
Time	D_3	D_2	D_1	D_0
0	1	1	0	1
1	1	1	1	0
2	1	1	1	1
3	0	1	1	1
4	0	0	1	1
5	0	0	0	1
6	1	0	0	0
7	0	1	0	0
8	0	0	1	0
9	1	0	0	1
10	1	1	0	0
11	0	1	1	0
12	1	0	1	1

LFSR: Example



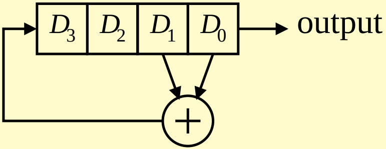
Time	D_3	D_2	D_1	D_0
0	1	1	0	1
1	1	1	1	0
2	1	1	1	1
3	0	1	1	1
4	0	0	1	1
5	0	0	0	1
6	1	0	0	0
7	0	1	0	0
8	0	0	1	0
9	1	0	0	1
10	1	1	0	0
11	0	1	1	0
12	1	0	1	1
13	0	1	0	1

LFSR: Example



Time	D_3	D_2	D_1	D_0
0	1	1	0	1
1	1	1	1	0
2	1	1	1	1
3	0	1	1	1
4	0	0	1	1
5	0	0	0	1
6	1	0	0	0
7	0	1	0	0
8	0	0	1	0
9	1	0	0	1
10	1	1	0	0
11	0	1	1	0
12	1	0	1	1
13	0	1	0	1
14	1	0	1	0

LFSR: Example



Time	D_3	D_2	D_1	D_0
0	1	1	0	1
1	1	1	1	0
2	1	1	1	1
3	0	1	1	1
4	0	0	1	1
5	0	0	0	1
6	1	0	0	0
7	0	1	0	0
8	0	0	1	0
9	1	0	0	1
10	1	1	0	0
11	0	1	1	0
12	1	0	1	1
13	0	1	0	1
14	1	0	1	0
15	1	1	0	1

LFSR: State Transition

LFSR: State Transition

- **Control bits:** $a_0, a_1, \ldots, a_{d-1}$.

LFSR: State Transition

- **Control bits:** $a_0, a_1, \ldots, a_{d-1}$.
- **State:** $\mathbf{s} = (s_0, s_1, \ldots, s_{d-1})$.

LFSR: State Transition

- **Control bits:** $a_0, a_1, \ldots, a_{d-1}$.
- **State:** $\mathbf{s} = (s_0, s_1, \ldots, s_{d-1})$.
- Each clock pulse changes the state as follows:

$$
\begin{aligned}
t_0 &= s_1 \\
t_1 &= s_2 \\
&\vdots \\
t_{d-2} &= s_{d-1} \\
t_{d-1} &\equiv a_0 s_0 + a_1 s_1 + a_2 s_2 + \cdots + a_{d-1} s_{d-1} \pmod 2.
\end{aligned}
$$

LFSR: State Transition (contd)

- In the matrix notation $\mathbf{t} \equiv \Delta_L \mathbf{s} \pmod 2$, where the **transition matrix** is

$$\Delta_L = \begin{pmatrix} 0 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \cdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & 1 \\ a_0 & a_1 & a_2 & \cdots & a_{d-2} & a_{d-1} \end{pmatrix}.$$

LFSR (contd)

LFSR (contd)

- The output bit-stream behaves like a pseudorandom sequence.

LFSR (contd)

- The output bit-stream behaves like a pseudorandom sequence.
- The output stream must be periodic. The period should be large.

LFSR (contd)

- The output bit-stream behaves like a pseudorandom sequence.
- The output stream must be periodic. The period should be large.
- Maximum period of a non-zero bit-stream = $2^d - 1$.

LFSR (contd)

- The output bit-stream behaves like a pseudorandom sequence.
- The output stream must be periodic. The period should be large.
- Maximum period of a non-zero bit-stream = $2^d - 1$.
- **Maximum-length LFSR** has the maximum period.

LFSR (contd)

- The output bit-stream behaves like a pseudorandom sequence.
- The output stream must be periodic. The period should be large.
- Maximum period of a non-zero bit-stream = $2^d - 1$.
- **Maximum-length LFSR** has the maximum period.
- **Connection polynomial**

$$C_L(x) = 1 + a_{d-1}x + a_{d-2}x^2 + \cdots + a_1x^{d-1} + a_0x^d \in \mathbb{F}_2[X].$$

LFSR (contd)

- The output bit-stream behaves like a pseudorandom sequence.
- The output stream must be periodic. The period should be large.
- Maximum period of a non-zero bit-stream = $2^d - 1$.
- **Maximum-length LFSR** has the maximum period.
- **Connection polynomial**

$$C_L(x) = 1 + a_{d-1}x + a_{d-2}x^2 + \cdots + a_1x^{d-1} + a_0x^d \in \mathbb{F}_2[X].$$

- L is a maximum-length LFSR if and only if $C_L(x)$ is a primitive polynomial of $\mathbb{F}_2[x]$.

An Attack on LFSR

An Attack on LFSR

- The linear relation of the feedback bit as a function of the current state in LFSRs invites attacks.

An Attack on LFSR

- The linear relation of the feedback bit as a function of the current state in LFSRs invites attacks.
- **Berlekamp-Massey attack**
  Suppose that the bits m_i and c_i for $2d$ consecutive values of i (say, $1, 2, \ldots, 2d$) are known to an attacker. Then $k_i = m_i \oplus c_i$ are also known for these values of i. Define the states $S_i = (k_i, k_{i+1}, \ldots, k_{i+d-1})$ of the LFSR. Then,

$$S_{i+1} \equiv \Delta_L S_i \pmod 2$$

  for $i = 1, 2, \ldots, d$. Treat each S_i as a column vector. Then,

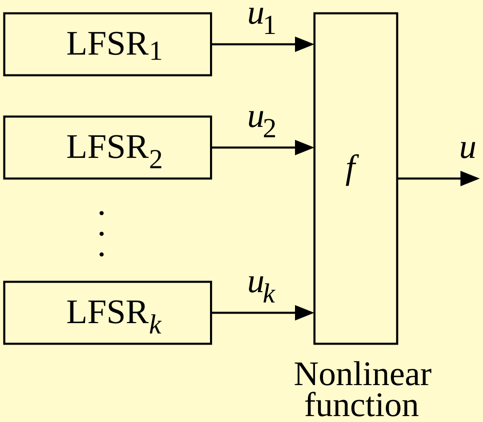
$$\begin{pmatrix} S_2 & S_3 & \cdots & S_{d+1} \end{pmatrix} \equiv \Delta_L \begin{pmatrix} S_1 & S_2 & \cdots & S_d \end{pmatrix} \pmod 2$$

  This reveals Δ_L, that is, the secret $a_0, a_1, \ldots, a_{d-1}$.

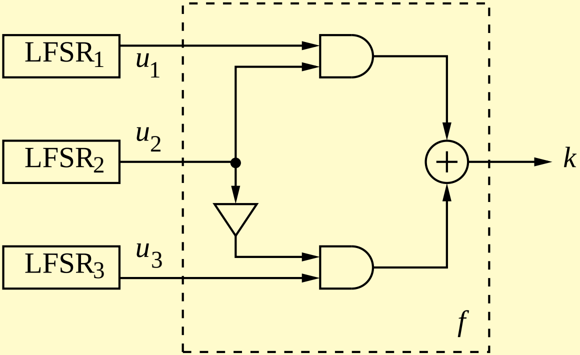
An Attack on LFSR

- The linear relation of the feedback bit as a function of the current state in LFSRs invites attacks.
- **Berlekamp-Massey attack**
  Suppose that the bits m_i and c_i for $2d$ consecutive values of i (say, $1, 2, \ldots, 2d$) are known to an attacker. Then $k_i = m_i \oplus c_i$ are also known for these values of i. Define the states $S_i = (k_i, k_{i+1}, \ldots, k_{i+d-1})$ of the LFSR. Then,
  $$S_{i+1} \equiv \Delta_L S_i \pmod 2$$
  for $i = 1, 2, \ldots, d$. Treat each S_i as a column vector. Then,
  $$\begin{pmatrix} S_2 & S_3 & \cdots & S_{d+1} \end{pmatrix} \equiv \Delta_L \begin{pmatrix} S_1 & S_2 & \cdots & S_d \end{pmatrix} \pmod 2$$
  This reveals Δ_L, that is, the secret $a_0, a_1, \ldots, a_{d-1}$.
- **Remedy:** Introduce non-linearity to the LFSR output.

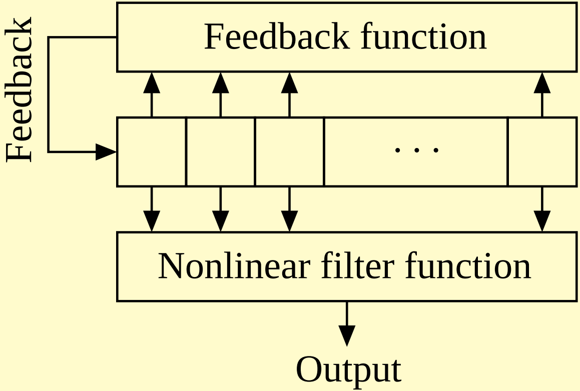
Nonlinear Combination Generator



The Geffe Generator



Nonlinear Filter Generator



Hash Functions

Hash Functions

- Used to convert strings of any length to strings of a fixed length.

Hash Functions

- Used to convert strings of any length to strings of a fixed length.
- Used for the generation of (short) representatives of messages.

Hash Functions

- Used to convert strings of any length to strings of a fixed length.
- Used for the generation of (short) representatives of messages.
- Unkeyed hash functions ensure data integrity.

Hash Functions

- Used to convert strings of any length to strings of a fixed length.
- Used for the generation of (short) representatives of messages.
- Unkeyed hash functions ensure data integrity.
- Keyed hash functions authenticate source of messages.

Hash Functions

- Used to convert strings of any length to strings of a fixed length.
- Used for the generation of (short) representatives of messages.
- Unkeyed hash functions ensure data integrity.
- Keyed hash functions authenticate source of messages.
- Symmetric techniques are typically used for designing hash functions.

Hash Functions

- Used to convert strings of any length to strings of a fixed length.
- Used for the generation of (short) representatives of messages.
- Unkeyed hash functions ensure data integrity.
- Keyed hash functions authenticate source of messages.
- Symmetric techniques are typically used for designing hash functions.

- A **collision** for a hash function H is a pair of two distinct strings x, y with $H(x) = H(y)$.

Hash Functions

- Used to convert strings of any length to strings of a fixed length.
- Used for the generation of (short) representatives of messages.
- Unkeyed hash functions ensure data integrity.
- Keyed hash functions authenticate source of messages.
- Symmetric techniques are typically used for designing hash functions.

- A **collision** for a hash function H is a pair of two distinct strings x, y with $H(x) = H(y)$.
- Since hash functions map an infinite domain to finite sets, collisions must exist for any hash function.

Hash Functions: Desirable Properties

Hash Functions: Desirable Properties

- **Easy to compute**

Hash Functions: Desirable Properties

- **Easy to compute**
- **First pre-image resistance (Difficult to invert):** For most hash values y, it should be difficult to find a string x with $H(x) = y$.

Hash Functions: Desirable Properties

- **Easy to compute**
- **First pre-image resistance (Difficult to invert):** For most hash values y, it should be difficult to find a string x with $H(x) = y$.
- **Second pre-image resistance:** Given a string x, it should be difficult to find a different string x' with $H(x') = H(x)$.

Hash Functions: Desirable Properties

- **Easy to compute**
- **First pre-image resistance (Difficult to invert):** For most hash values y, it should be difficult to find a string x with $H(x) = y$.
- **Second pre-image resistance:** Given a string x, it should be difficult to find a different string x' with $H(x') = H(x)$.
- **Collision resistance:** It should be difficult to find two distinct strings x, x' with $H(x) = H(x')$.

Hash Functions: Properties (contd)

Hash Functions: Properties (contd)

- **Collision resistance implies second pre-image resistance.**

Hash Functions: Properties (contd)

- **Collision resistance implies second pre-image resistance.**
- **Second pre-image resistance does not imply collision resistance:** Let S be a finite set of size $\geqslant 2$ and H a cryptographic hash function. Then

$$H'(x) = \begin{cases} 0^{n+1} & \text{if } x \in S, \\ 1 \,||\, H(x) & \text{otherwise,} \end{cases}$$

is second pre-image resistant but not collision resistant.

Hash Functions: Properties (contd)

Hash Functions: Properties (contd)

- **Collision resistance does not imply first pre-image resistance:** Let H be an n-bit cryptographic hash function. Then

$$H''(x) = \begin{cases} 0 \,||\, x & \text{if } |x| = n, \\ 1 \,||\, H(x) & \text{otherwise.} \end{cases}$$

is collision resistant (so second pre-image resistant), but not first pre-image resistant.

Hash Functions: Properties (contd)

- **Collision resistance does not imply first pre-image resistance:** Let H be an n-bit cryptographic hash function. Then

$$H''(x) = \begin{cases} 0 \,||\, x & \text{if } |x| = n, \\ 1 \,||\, H(x) & \text{otherwise.} \end{cases}$$

  is collision resistant (so second pre-image resistant), but not first pre-image resistant.

- **First pre-image resistance does not imply second pre-image resistance:** Let m be a product of two unknown big primes. Define $H'''(x) = (1 \,||\, x)^2 \pmod{m}$. H''' is first pre-image resistant, but not second pre-image resistant.

Hash Functions: Construction

Hash Functions: Construction

- **Compression function:** A function $F : \mathbb{Z}_2^m \to \mathbb{Z}_2^n$, where $m = n + r$.

Hash Functions: Construction

- **Compression function:** A function $F : \mathbb{Z}_2^m \rightarrow \mathbb{Z}_2^n$, where $m = n + r$.
- **Merkle-Damgård's meta method**

Hash Functions: Construction

- **Compression function:** A function $F : \mathbb{Z}_2^m \rightarrow \mathbb{Z}_2^n$, where $m = n + r$.
- **Merkle-Damgård's meta method**
  - Break the input $x = x_1 x_2 \ldots x_l$ to blocks each of bit-length r.

Hash Functions: Construction

- **Compression function:** A function $F : \mathbb{Z}_2^m \rightarrow \mathbb{Z}_2^n$, where $m = n + r$.
- **Merkle-Damgård's meta method**
  - Break the input $x = x_1 x_2 \ldots x_l$ to blocks each of bit-length r.
  - Initialize $h_0 = 0^r$.

Hash Functions: Construction

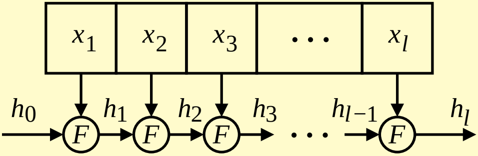
- **Compression function:** A function $F : \mathbb{Z}_2^m \to \mathbb{Z}_2^n$, where $m = n + r$.
- **Merkle-Damgård's meta method**
  - Break the input $x = x_1 x_2 \ldots x_l$ to blocks each of bit-length r.
  - Initialize $h_0 = 0^r$.
  - For $i = 1, 2, \ldots, l$ use compression $h_i = F(h_{i-1} \,||\, x_i)$.

Hash Functions: Construction

- **Compression function:** A function $F : \mathbb{Z}_2^m \rightarrow \mathbb{Z}_2^n$, where $m = n + r$.
- **Merkle-Damgård's meta method**
  - Break the input $x = x_1 x_2 \ldots x_l$ to blocks each of bit-length r.
  - Initialize $h_0 = 0^r$.
  - For $i = 1, 2, \ldots, l$ use compression $h_i = F(h_{i-1} \,||\, x_i)$.
  - Output $H(x) = h_l$ as the hash value.

Hash Functions: Construction

- **Compression function:** A function $F : \mathbb{Z}_2^m \rightarrow \mathbb{Z}_2^n$, where $m = n + r$.
- **Merkle-Damgård's meta method**
  - Break the input $x = x_1 x_2 \ldots x_l$ to blocks each of bit-length r.
  - Initialize $h_0 = 0^r$.
  - For $i = 1, 2, \ldots, l$ use compression $h_i = F(h_{i-1} \,||\, x_i)$.
  - Output $H(x) = h_l$ as the hash value.



Hash Functions: Construction (contd)

Hash Functions: Construction (contd)

- **Properties**

Hash Functions: Construction (contd)

- **Properties**
  - If F is first pre-image resistant, then H is also first pre-image resistant.

Hash Functions: Construction (contd)

- **Properties**
  - If F is first pre-image resistant, then H is also first pre-image resistant.
  - If F is collision resistant, then H is also collision resistant.

Hash Functions: Construction (contd)

- **Properties**
  - If F is first pre-image resistant, then H is also first pre-image resistant.
  - If F is collision resistant, then H is also collision resistant.
- **A concrete realization**

  Let f is a block cipher of block-size n and key-size r. Take:

$$F(M \,||\, K) = f_K(M).$$

Hash Functions: Construction (contd)

- **Properties**
  - If F is first pre-image resistant, then H is also first pre-image resistant.
  - If F is collision resistant, then H is also collision resistant.
- **A concrete realization**
  Let f is a block cipher of block-size n and key-size r. Take:
  $$F(M \,||\, K) = f_K(M).$$
- **Keyed hash function**
  $\text{HMAC}(M) = H(K \,||\, P \,||\, H(K \,||\, Q \,||\, M))$, where H is an unkeyed hash function, K is a key and P, Q are short padding strings.

Custom-Designed Hash Functions

Custom-Designed Hash Functions

- **The SHA (Secure Hash Algorithm) family:**
  SHA-1 (160-bit), SHA-256 (256-bit),
  SHA-384 (384-bit), SHA-512 (512-bit).

Custom-Designed Hash Functions

- **The SHA (Secure Hash Algorithm) family:**
  SHA-1 (160-bit), SHA-256 (256-bit),
  SHA-384 (384-bit), SHA-512 (512-bit).
- **The MD family:**
  MD2 (128-bit), MD5 (128-bit).

Custom-Designed Hash Functions

- **The SHA (Secure Hash Algorithm) family:**
  SHA-1 (160-bit), SHA-256 (256-bit),
  SHA-384 (384-bit), SHA-512 (512-bit).
- **The MD family:**
  MD2 (128-bit), MD5 (128-bit).
- **The RIPEMD family:**
  RIPEMD-128 (128-bit), RIPEMD-160 (160-bit).

SHA-1: Message Padding

SHA-1: Message Padding

- To compute SHA-1(M) for a message M of bit-length λ.

SHA-1: Message Padding

- To compute SHA-1(M) for a message M of bit-length λ.
- Pad M to generate $M' = M \mid\mid 1 \mid\mid 0^k \mid\mid \Lambda$, where

SHA-1: Message Padding

- To compute SHA-1(M) for a message M of bit-length λ.
- Pad M to generate $M' = M \,||\, 1 \,||\, 0^k \,||\, \Lambda$, where
  - Λ is the 64-bit representation of λ, and

SHA-1: Message Padding

- To compute SHA-1(M) for a message M of bit-length λ.
- Pad M to generate $M' = M \,||\, 1 \,||\, 0^k \,||\, \Lambda$, where
    - Λ is the 64-bit representation of λ, and
    - k is the smallest integer $\geqslant 0$ for which $|M'| = \lambda + 1 + k + 64$ is a multiple of 512.

SHA-1: Message Padding

- To compute SHA-1(M) for a message M of bit-length λ.
- Pad M to generate $M' = M \,||\, 1 \,||\, 0^k \,||\, \Lambda$, where
  - Λ is the 64-bit representation of λ, and
  - k is the smallest integer $\geqslant 0$ for which $|M'| = \lambda + 1 + k + 64$ is a multiple of 512.
- Break M' into 512-bit blocks $M^{(1)}, M^{(2)}, \ldots, M^{(l)}$.

SHA-1: Message Padding

- To compute SHA-1(M) for a message M of bit-length λ.
- Pad M to generate $M' = M \mid\mid 1 \mid\mid 0^k \mid\mid \Lambda$, where
  - Λ is the 64-bit representation of λ, and
  - k is the smallest integer $\geqslant 0$ for which $|M'| = \lambda + 1 + k + 64$ is a multiple of 512.
- Break M' into 512-bit blocks $M^{(1)}, M^{(2)}, \ldots, M^{(l)}$.
- Break each $M^{(i)} = M_0^{(i)} \mid\mid M_1^{(i)} \mid\mid \cdots \mid\mid M_{15}^{(i)}$ into sixteen 32-bit words $M_j^{(i)}$.

SHA-1: Iterated Hash Construction

SHA-1: Iterated Hash Construction

- The idea is similar to the Merkle-Damgård construction.

SHA-1: Iterated Hash Construction

- The idea is similar to the Merkle-Damgård construction.
- Start with the initial hash value $H^{(0)} =$ `0x67452301 efcdab89 98badcfe 10325476 c3d2e1f0`.

SHA-1: Iterated Hash Construction

- The idea is similar to the Merkle-Damgård construction.
- Start with the initial hash value $H^{(0)} =$ `0x67452301 efcdab89 98badcfe 10325476 c3d2e1f0`.
- For $i = 1, 2, \ldots, l$, consume the message block $M^{(i)}$ to convert $H^{(i-1)}$ to $H^{(i)}$.

SHA-1: Iterated Hash Construction

- The idea is similar to the Merkle-Damgård construction.

- Start with the initial hash value $H^{(0)} =$ `0x67452301 efcdab89 98badcfe 10325476 c3d2e1f0`.
- For $i = 1, 2, \ldots, l$, consume the message block $M^{(i)}$ to convert $H^{(i-1)}$ to $H^{(i)}$.
- Return $H^{(l)}$ as SHA-1(M).

SHA-1: Iterated Hash Construction

- The idea is similar to the Merkle-Damgård construction.
- Start with the initial hash value $H^{(0)} =$ `0x67452301 efcdab89 98badcfe 10325476 c3d2e1f0`.
- For $i = 1, 2, \ldots, l$, consume the message block $M^{(i)}$ to convert $H^{(i-1)}$ to $H^{(i)}$.
- Return $H^{(l)}$ as SHA-1(M).
- Each $H^{(i)}$ is a 160-bit value.

SHA-1: Iterated Hash Construction

- The idea is similar to the Merkle-Damgård construction.

- Start with the initial hash value $H^{(0)} =$ `0x67452301 efcdab89 98badcfe 10325476 c3d2e1f0`.
- For $i = 1, 2, \ldots, l$, consume the message block $M^{(i)}$ to convert $H^{(i-1)}$ to $H^{(i)}$.
- Return $H^{(l)}$ as SHA-1(M).

- Each $H^{(i)}$ is a 160-bit value.
- Write $H^{(i)} = H_0^{(i)} \,||\, H_1^{(i)} \,||\, H_2^{(i)} \,||\, H_3^{(i)} \,||\, H_4^{(i)}$, where each $H_j^{(i)}$ is a 32-bit word.

SHA-1: Compression Function

SHA-1: Compression Function

- Compute the message schedule W_j, $0 \leqslant j \leqslant 79$:

SHA-1: Compression Function

- Compute the message schedule W_j, $0 \leqslant j \leqslant 79$:
  - For $j = 0, 1, \ldots, 15$, set $W_j := M_j^{(i)}$.

SHA-1: Compression Function

- Compute the message schedule W_j, $0 \leqslant j \leqslant 79$:
  - For $j = 0, 1, \ldots, 15$, set $W_j := M_j^{(i)}$.
  - For $j = 16, 17, \ldots, 79$, set $W_j := \mathrm{LR}^1(W_{j-3} \oplus W_{j-8} \oplus W_{j-14} \oplus W_{j-16})$.

SHA-1: Compression Function

- Compute the message schedule W_j, $0 \leqslant j \leqslant 79$:
    - For $j = 0, 1, \ldots, 15$, set $W_j := M_j^{(i)}$.
    - For $j = 16, 17, \ldots, 79$, set $W_j := \mathrm{LR}^1(W_{j-3} \oplus W_{j-8} \oplus W_{j-14} \oplus W_{j-16})$.
- For $j = 0, 1, 2, 3, 4$, store $H_j^{(i-1)}$ in t_j.

SHA-1: Compression Function

- Compute the message schedule W_j, $0 \leqslant j \leqslant 79$:
  - For $j = 0, 1, \ldots, 15$, set $W_j := M_j^{(i)}$.
  - For $j = 16, 17, \ldots, 79$, set $W_j := \mathrm{LR}^1(W_{j-3} \oplus W_{j-8} \oplus W_{j-14} \oplus W_{j-16})$.
- For $j = 0, 1, 2, 3, 4$, store $H_j^{(i-1)}$ in t_j.
- For $j = 0, 1, \ldots, 79$, do the following:

SHA-1: Compression Function

- Compute the message schedule W_j, $0 \leqslant j \leqslant 79$:
  - For $j = 0, 1, \ldots, 15$, set $W_j := M_j^{(i)}$.
  - For $j = 16, 17, \ldots, 79$, set $W_j := \mathrm{LR}^1(W_{j-3} \oplus W_{j-8} \oplus W_{j-14} \oplus W_{j-16})$.
- For $j = 0, 1, 2, 3, 4$, store $H_j^{(i-1)}$ in t_j.
- For $j = 0, 1, \ldots, 79$, do the following:
  - Set $T = \left(\mathrm{LR}^5(t_0) + f_j(t_1, t_2, t_3) + t_4 + K_j + W_j\right) \text{ rem } 2^{32}$.

SHA-1: Compression Function

- Compute the message schedule W_j, $0 \leqslant j \leqslant 79$:
  - For $j = 0, 1, \ldots, 15$, set $W_j := M_j^{(i)}$.
  - For $j = 16, 17, \ldots, 79$, set $W_j := \mathrm{LR}^1(W_{j-3} \oplus W_{j-8} \oplus W_{j-14} \oplus W_{j-16})$.
- For $j = 0, 1, 2, 3, 4$, store $H_j^{(i-1)}$ in t_j.
- For $j = 0, 1, \ldots, 79$, do the following:
  - Set $T = \left(\mathrm{LR}^5(t_0) + f_j(t_1, t_2, t_3) + t_4 + K_j + W_j\right) \operatorname{rem} 2^{32}$.
  - $t_4 = t_3,\ t_3 = t_2,\ t_2 = \mathrm{RR}^2(t_1),\ t_1 = t_0,\ t_0 = T$.

SHA-1: Compression Function

- Compute the message schedule W_j, $0 \leqslant j \leqslant 79$:
    - For $j = 0, 1, \ldots, 15$, set $W_j := M_j^{(i)}$.
    - For $j = 16, 17, \ldots, 79$, set $W_j := \mathrm{LR}^1(W_{j-3} \oplus W_{j-8} \oplus W_{j-14} \oplus W_{j-16})$.
- For $j = 0, 1, 2, 3, 4$, store $H_j^{(i-1)}$ in t_j.
- For $j = 0, 1, \ldots, 79$, do the following:
    - Set $T = \left(\mathrm{LR}^5(t_0) + f_j(t_1, t_2, t_3) + t_4 + K_j + W_j\right) \operatorname{rem} 2^{32}$.
    - $t_4 = t_3,\ t_3 = t_2,\ t_2 = \mathrm{RR}^2(t_1),\ t_1 = t_0,\ t_0 = T$.
- For $j = 0, 1, 2, 3, 4$, update $H_j^{(i)} := \left(t_j + H_j^{(i-1)}\right) \operatorname{rem} 2^{32}$.

SHA-1: Compression Function (contd)

SHA-1: Compression Function (contd)

- $$f_j(x,y,z) = \begin{cases} xy \oplus \overline{x}z & \text{if } 0 \leqslant j \leqslant 19 \\ x \oplus y \oplus z & \text{if } 20 \leqslant j \leqslant 39 \\ xy \oplus xz \oplus yz & \text{if } 40 \leqslant j \leqslant 59 \\ x \oplus y \oplus z & \text{if } 60 \leqslant j \leqslant 79 \end{cases}$$

SHA-1: Compression Function (contd)

- $$f_j(x,y,z) = \begin{cases} xy \oplus \bar{x}z & \text{if } 0 \leqslant j \leqslant 19 \\ x \oplus y \oplus z & \text{if } 20 \leqslant j \leqslant 39 \\ xy \oplus xz \oplus yz & \text{if } 40 \leqslant j \leqslant 59 \\ x \oplus y \oplus z & \text{if } 60 \leqslant j \leqslant 79 \end{cases}$$

- $$K_j = \begin{cases} \texttt{0x5a827999} & \text{if } 0 \leqslant j \leqslant 19 \\ \texttt{0x6ed9eba1} & \text{if } 20 \leqslant j \leqslant 39 \\ \texttt{0x8f1bbcdc} & \text{if } 40 \leqslant j \leqslant 59 \\ \texttt{0xca62c1d6} & \text{if } 60 \leqslant j \leqslant 79 \end{cases}$$

SHA-1: Compression Function (contd)

- $$f_j(x,y,z) = \begin{cases} xy \oplus \bar{x}z & \text{if } 0 \leqslant j \leqslant 19 \\ x \oplus y \oplus z & \text{if } 20 \leqslant j \leqslant 39 \\ xy \oplus xz \oplus yz & \text{if } 40 \leqslant j \leqslant 59 \\ x \oplus y \oplus z & \text{if } 60 \leqslant j \leqslant 79 \end{cases}$$
- $$K_j = \begin{cases} \texttt{0x5a827999} & \text{if } 0 \leqslant j \leqslant 19 \\ \texttt{0x6ed9eba1} & \text{if } 20 \leqslant j \leqslant 39 \\ \texttt{0x8f1bbcdc} & \text{if } 40 \leqslant j \leqslant 59 \\ \texttt{0xca62c1d6} & \text{if } 60 \leqslant j \leqslant 79 \end{cases}$$
- LR^k and RR^k mean left and right rotate by k bits.

Attacks on Hash Functions

Attacks on Hash Functions

- The **birthday attack** is based on the birthday paradox. For an n-bit hash function, one needs to compute on an average $2^{n/2}$ hash values in order to detect (with high probability) a collision for the hash function.

Attacks on Hash Functions

- The **birthday attack** is based on the birthday paradox. For an n-bit hash function, one needs to compute on an average $2^{n/2}$ hash values in order to detect (with high probability) a collision for the hash function.
- For cryptographic applications one requires $n \geqslant 128$ ($n \geqslant 160$ is preferable).

Attacks on Hash Functions

- The **birthday attack** is based on the birthday paradox. For an n-bit hash function, one needs to compute on an average $2^{n/2}$ hash values in order to detect (with high probability) a collision for the hash function.
- For cryptographic applications one requires $n \geqslant 128$ ($n \geqslant 160$ is preferable).
- **Algebraic attacks** may make hash functions vulnerable.

Attacks on Hash Functions

- The **birthday attack** is based on the birthday paradox. For an n-bit hash function, one needs to compute on an average $2^{n/2}$ hash values in order to detect (with high probability) a collision for the hash function.
- For cryptographic applications one requires $n \geqslant 128$ ($n \geqslant 160$ is preferable).
- **Algebraic attacks** may make hash functions vulnerable.
- Some other attacks:

Attacks on Hash Functions

- The **birthday attack** is based on the birthday paradox. For an n-bit hash function, one needs to compute on an average $2^{n/2}$ hash values in order to detect (with high probability) a collision for the hash function.
- For cryptographic applications one requires $n \geqslant 128$ ($n \geqslant 160$ is preferable).
- **Algebraic attacks** may make hash functions vulnerable.
- Some other attacks:
  - Pseudo-collision attacks

Attacks on Hash Functions

- The **birthday attack** is based on the birthday paradox. For an n-bit hash function, one needs to compute on an average $2^{n/2}$ hash values in order to detect (with high probability) a collision for the hash function.
- For cryptographic applications one requires $n \geqslant 128$ ($n \geqslant 160$ is preferable).
- **Algebraic attacks** may make hash functions vulnerable.
- Some other attacks:
  - Pseudo-collision attacks
  - Chaining attacks

Attacks on Hash Functions

- The **birthday attack** is based on the birthday paradox. For an n-bit hash function, one needs to compute on an average $2^{n/2}$ hash values in order to detect (with high probability) a collision for the hash function.
- For cryptographic applications one requires $n \geqslant 128$ ($n \geqslant 160$ is preferable).
- **Algebraic attacks** may make hash functions vulnerable.
- Some other attacks:
  - Pseudo-collision attacks
  - Chaining attacks
  - Attacks on the underlying cipher

Attacks on Hash Functions

- The **birthday attack** is based on the birthday paradox. For an n-bit hash function, one needs to compute on an average $2^{n/2}$ hash values in order to detect (with high probability) a collision for the hash function.
- For cryptographic applications one requires $n \geqslant 128$ ($n \geqslant 160$ is preferable).
- **Algebraic attacks** may make hash functions vulnerable.
- Some other attacks:
  - Pseudo-collision attacks
  - Chaining attacks
  - Attacks on the underlying cipher
  - Exhaustive key search for keyed hash functions

Attacks on Hash Functions

- The **birthday attack** is based on the birthday paradox. For an n-bit hash function, one needs to compute on an average $2^{n/2}$ hash values in order to detect (with high probability) a collision for the hash function.
- For cryptographic applications one requires $n \geqslant 128$ ($n \geqslant 160$ is preferable).
- **Algebraic attacks** may make hash functions vulnerable.
- Some other attacks:
  - Pseudo-collision attacks
  - Chaining attacks
  - Attacks on the underlying cipher
  - Exhaustive key search for keyed hash functions
  - Long message attacks

The Birthday Paradox

Let S be a set finite size N.

The Birthday Paradox

Let S be a set finite size N.

- k elements are drawn at random from S (with replacement).

The Birthday Paradox

Let S be a set finite size N.

- k elements are drawn at random from S (with replacement).
- The probability that all these k elements are distinct is

$$p_k = \frac{N(N-1)\cdots(N-k+1)}{N^k} = \prod_{i=1}^{k-1}\left(1 - \frac{i}{N}\right) \leqslant e^{\frac{-k(k-1)}{2N}}.$$

The Birthday Paradox

Let S be a set finite size N.

- k elements are drawn at random from S (with replacement).
- The probability that all these k elements are distinct is

$$p_k = \frac{N(N-1)\cdots(N-k+1)}{N^k} = \prod_{i=1}^{k-1}\left(1 - \frac{i}{N}\right) \leqslant e^{\frac{-k(k-1)}{2N}}.$$

- $p_k \leqslant 1/2$ for $k \geqslant \frac{1}{2}\sqrt{1+8N\ln 2} \approx 1.18\sqrt{N}$.

The Birthday Paradox

Let S be a set finite size N.

- k elements are drawn at random from S (with replacement).
- The probability that all these k elements are distinct is

$$p_k = \frac{N(N-1)\cdots(N-k+1)}{N^k} = \prod_{i=1}^{k-1}\left(1 - \frac{i}{N}\right) \leqslant e^{\frac{-k(k-1)}{2N}}.$$

- $p_k \leqslant 1/2$ for $k \geqslant \frac{1}{2}\sqrt{1+8N\ln 2} \approx 1.18\sqrt{N}$.
- $p_k \leqslant 0.136$ for $k \geqslant 2\sqrt{N}$.

The Birthday Paradox

Let S be a set finite size N.

- k elements are drawn at random from S (with replacement).
- The probability that all these k elements are distinct is

$$p_k = \frac{N(N-1)\cdots(N-k+1)}{N^k} = \prod_{i=1}^{k-1}\left(1 - \frac{i}{N}\right) \leqslant e^{\frac{-k(k-1)}{2N}}.$$

- $p_k \leqslant 1/2$ for $k \geqslant \frac{1}{2}\sqrt{1+8N\ln 2} \approx 1.18\sqrt{N}$.
- $p_k \leqslant 0.136$ for $k \geqslant 2\sqrt{N}$.

Examples

The Birthday Paradox

Let S be a set finite size N.

- k elements are drawn at random from S (with replacement).
- The probability that all these k elements are distinct is

$$p_k = \frac{N(N-1)\cdots(N-k+1)}{N^k} = \prod_{i=1}^{k-1}\left(1 - \frac{i}{N}\right) \leqslant e^{\frac{-k(k-1)}{2N}}.$$

- $p_k \leqslant 1/2$ for $k \geqslant \frac{1}{2}\sqrt{1+8N\ln 2} \approx 1.18\sqrt{N}$.
- $p_k \leqslant 0.136$ for $k \geqslant 2\sqrt{N}$.

Examples

- There is a chance of $\geqslant 50\%$ that at least two of $\geqslant 23$ (randomly chosen) persons have the same birthday.

The Birthday Paradox

Let S be a set finite size N.

- k elements are drawn at random from S (with replacement).
- The probability that all these k elements are distinct is

$$p_k = \frac{N(N-1)\cdots(N-k+1)}{N^k} = \prod_{i=1}^{k-1}\left(1 - \frac{i}{N}\right) \leqslant e^{\frac{-k(k-1)}{2N}}.$$

- $p_k \leqslant 1/2$ for $k \geqslant \frac{1}{2}\sqrt{1+8N\ln 2} \approx 1.18\sqrt{N}$.
- $p_k \leqslant 0.136$ for $k \geqslant 2\sqrt{N}$.

Examples

- There is a chance of $\geqslant 50\%$ that at least two of $\geqslant 23$ (randomly chosen) persons have the same birthday.
- A collision of an n-bit hash function can be found with high probability from $O(2^{n/2})$ random hash calculations.