Cryptography

Lecture 16

Arkady Yerukhimovich

October 23, 2024

Outline

Lecture 15 Review

- Review of exam and research project
- Blockciphers
- Confusion-Diffusion paradigm and the avalanche effect
- Substitution-Permutation Paradigm
- AES

Outline

AES / Rijndael History

Advanced Encryption Standard (AES)

Advanced Encryption Standard (AES)

- Developed by Vincent Rijmen and Joan Daemen

AES / Rijndael History

Advanced Encryption Standard (AES)

- Developed by Vincent Rijmen and Joan Daemen
- Winner of NIST blockcipher competition in 2000

AES / Rijndael History

Advanced Encryption Standard (AES)

- Developed by Vincent Rijmen and Joan Daemen
- Winner of NIST blockcipher competition in 2000
- Now, standardized and very widely used

AES / Rijndael History

Advanced Encryption Standard (AES)

- Developed by Vincent Rijmen and Joan Daemen
- Winner of NIST blockcipher competition in 2000
- Now, standardized and very widely used
- Hardware support (e.g., AES-NI) makes this very fast: 10^8 per second

AES / Rijndael History

Advanced Encryption Standard (AES)

- Developed by Vincent Rijmen and Joan Daemen
- Winner of NIST blockcipher competition in 2000
- Now, standardized and very widely used
- Hardware support (e.g., AES-NI) makes this very fast: 10^8 per second
- This is the right block-cipher to use for most applications

AES

Available Versions: $\ell = 128(16 \text{ Bytes}), n = 128, 192, 256$

AES

Available Versions: $\ell = 128$(16 Bytes), $n = 128, 192, 256$

- Change key schedule for different key lengths

AES

Available Versions: $\ell = 128$(16 Bytes), $n = 128, 192, 256$

- Change key schedule for different key lengths
- 10 rounds for 128-bit key, 12 rounds for 192-bit key, 14 rounds for 256-bit key

AES

Available Versions: $\ell = 128$(16 Bytes), $n = 128, 192, 256$

- Change key schedule for different key lengths
- 10 rounds for 128-bit key, 12 rounds for 192-bit key, 14 rounds for 256-bit key

State: 4×4 array of Bytes

$$\begin{pmatrix} B_{11} & B_{12} & B_{13} & B_{14} \\ B_{21} & B_{22} & B_{23} & B_{24} \\ B_{31} & B_{32} & B_{33} & B_{34} \\ B_{41} & B_{42} & B_{43} & B_{44} \end{pmatrix}$$

AES

1. Add Round Key: View $k_i = k_{11}||k_{12}||\cdots||k_{44}$ with $|k_{ij}| = 1$ Byte

$$\begin{pmatrix} B_{11} \oplus k_{11} & B_{12} \oplus k_{12} & B_{13} \oplus k_{13} & B_{14} \oplus k_{14} \\ B_{21} \oplus k_{21} & B_{22} \oplus k_{22} & B_{23} \oplus k_{23} & B_{24} \oplus k_{24} \\ B_{31} \oplus k_{31} & B_{32} \oplus k_{32} & B_{33} \oplus k_{33} & B_{34} \oplus k_{34} \\ B_{41} \oplus k_{41} & B_{42} \oplus k_{42} & B_{43} \oplus k_{43} & B_{44} \oplus k_{44} \end{pmatrix}$$

AES

1. Add Round Key: View $k_i = k_{11}||k_{12}||\cdots||k_{44}$ with $|k_{ij}| = 1$ Byte

$$\begin{pmatrix} B_{11} \oplus k_{11} & B_{12} \oplus k_{12} & B_{13} \oplus k_{13} & B_{14} \oplus k_{14} \\ B_{21} \oplus k_{21} & B_{22} \oplus k_{22} & B_{23} \oplus k_{23} & B_{24} \oplus k_{24} \\ B_{31} \oplus k_{31} & B_{32} \oplus k_{32} & B_{33} \oplus k_{33} & B_{34} \oplus k_{34} \\ B_{41} \oplus k_{41} & B_{42} \oplus k_{42} & B_{43} \oplus k_{43} & B_{44} \oplus k_{44} \end{pmatrix}$$

2. SubBytes: Uses a single 8-bit S-box

$$\begin{pmatrix} S(B_{11}) & S(B_{12}) & S(B_{13}) & S(B_{14}) \\ S(B_{21}) & S(B_{22}) & S(B_{23}) & S(B_{24}) \\ S(B_{31}) & S(B_{32}) & S(B_{33}) & S(B_{34}) \\ S(B_{41}) & S(B_{42}) & S(B_{43}) & S(B_{44}) \end{pmatrix}$$

3. Shift Rows: Shift each row to the left by varying amounts

$$\begin{pmatrix} B_{11} & B_{12} & B_{13} & B_{14} & \text{(0 shift)} \\ B_{22} & B_{23} & B_{24} & B_{21} & \text{(1 shift)} \\ B_{33} & B_{34} & B_{31} & B_{32} & \text{(2 shift)} \\ B_{44} & B_{41} & B_{42} & B_{43} & \text{(3 shift)} \end{pmatrix}$$

AES

3. Shift Rows: Shift each row to the left by varying amounts

$$\begin{pmatrix} B_{11} & B_{12} & B_{13} & B_{14} & \text{(0 shift)} \\ B_{22} & B_{23} & B_{24} & B_{21} & \text{(1 shift)} \\ B_{33} & B_{34} & B_{31} & B_{32} & \text{(2 shift)} \\ B_{44} & B_{41} & B_{42} & B_{43} & \text{(3 shift)} \end{pmatrix}$$

4. MixColumns: Apply invertible transformation to Bytes in each column

AES

3. Shift Rows: Shift each row to the left by varying amounts

$$\begin{pmatrix} B_{11} & B_{12} & B_{13} & B_{14} & (0 \text{ shift}) \\ B_{22} & B_{23} & B_{24} & B_{21} & (1 \text{ shift}) \\ B_{33} & B_{34} & B_{31} & B_{32} & (2 \text{ shift}) \\ B_{44} & B_{41} & B_{42} & B_{43} & (3 \text{ shift}) \end{pmatrix}$$

4. MixColumns: Apply invertible transformation to Bytes in each column

Observations:

AES

3. Shift Rows: Shift each row to the left by varying amounts

$$\begin{pmatrix} B_{11} & B_{12} & B_{13} & B_{14} & \text{(0 shift)} \\ B_{22} & B_{23} & B_{24} & B_{21} & \text{(1 shift)} \\ B_{33} & B_{34} & B_{31} & B_{32} & \text{(2 shift)} \\ B_{44} & B_{41} & B_{42} & B_{43} & \text{(3 shift)} \end{pmatrix}$$

4. MixColumns: Apply invertible transformation to Bytes in each column

Observations:

- Steps 3 and 4 correspond to permutation step

AES

3. Shift Rows: Shift each row to the left by varying amounts

$$\begin{pmatrix} B_{11} & B_{12} & B_{13} & B_{14} & \text{(0 shift)} \\ B_{22} & B_{23} & B_{24} & B_{21} & \text{(1 shift)} \\ B_{33} & B_{34} & B_{31} & B_{32} & \text{(2 shift)} \\ B_{44} & B_{41} & B_{42} & B_{43} & \text{(3 shift)} \end{pmatrix}$$

4. MixColumns: Apply invertible transformation to Bytes in each column

Observations:

- Steps 3 and 4 correspond to permutation step
- Even this very structured permutation seems enough

Outline

Feistel Networks

A different approach to building block ciphers:

Feistel Networks

A different approach to building block ciphers:

- SPN used small (1-Byte) permutations (S-boxes)

Feistel Networks

A different approach to building block ciphers:

- SPN used small (1-Byte) permutations (S-boxes)
- Feistel instead uses functions $f_i : \{0,1\}^{\ell/2} \rightarrow \{0,1\}^{\ell/2}$

Feistel Networks

A different approach to building block ciphers:

- SPN used small (1-Byte) permutations (S-boxes)
- Feistel instead uses functions $f_i : \{0,1\}^{\ell/2} \to \{0,1\}^{\ell/2}$
  - These f_i do not need to be invertible (not permutations)

Feistel Networks

A different approach to building block ciphers:

- SPN used small (1-Byte) permutations (S-boxes)
- Feistel instead uses functions $f_i : \{0,1\}^{\ell/2} \to \{0,1\}^{\ell/2}$
    - These f_i do not need to be invertible (not permutations)
    - Allows the f_i to have "less structure"

Feistel Networks

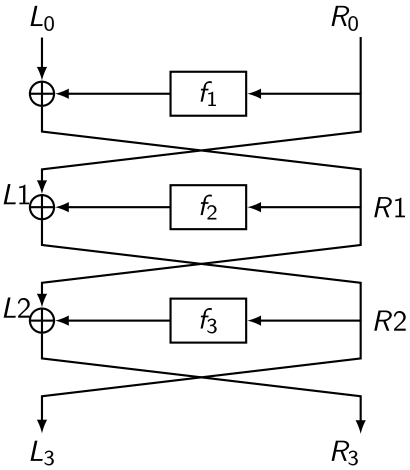
A different approach to building block ciphers:

- SPN used small (1-Byte) permutations (S-boxes)
- Feistel instead uses functions $f_i : \{0,1\}^{\ell/2} \rightarrow \{0,1\}^{\ell/2}$
    - These f_i do not need to be invertible (not permutations)
    - Allows the f_i to have "less structure"

The Challenge

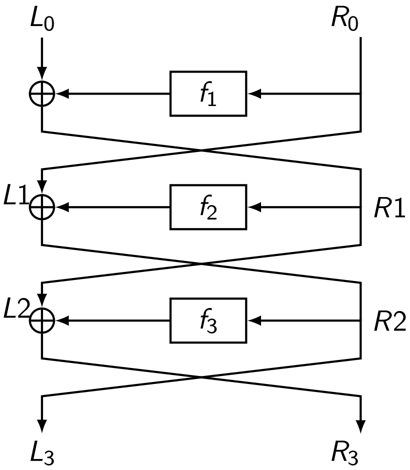
How do we use such f_i to build an invertible function?

Feistel Networks – Making f Invertible



Feistel Networks – Making f Invertible

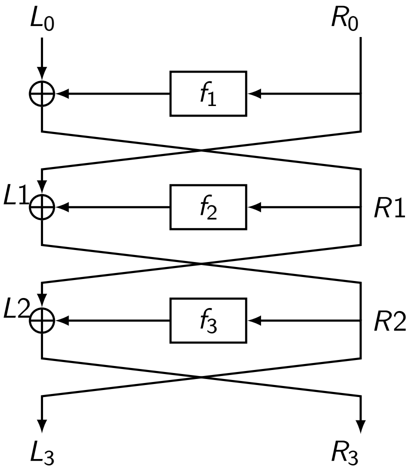
Evaluating round i:



Feistel Networks – Making f Invertible

Evaluating round i:

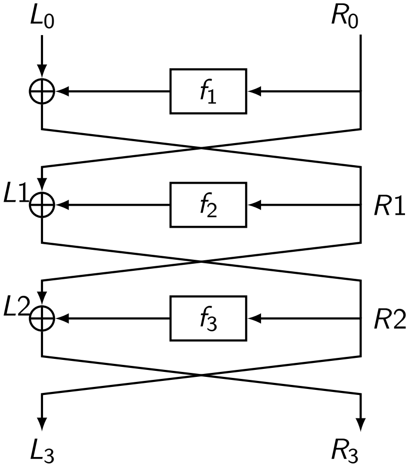
- Split input in half: $L_{i-1}||R_{i-1}$



Feistel Networks – Making f Invertible

Evaluating round i:

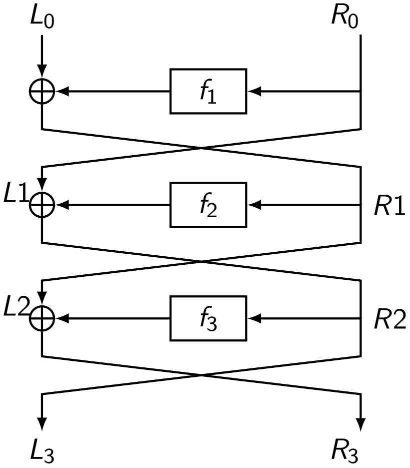
- Split input in half: $L_{i-1}||R_{i-1}$
- $L_i = R_{i-1}$



Feistel Networks – Making f Invertible

Evaluating round i:

- Split input in half: $L_{i-1}||R_{i-1}$
- $L_i = R_{i-1}$
- $R_i = L_{i-1} \oplus f_i(R_{i-1})$



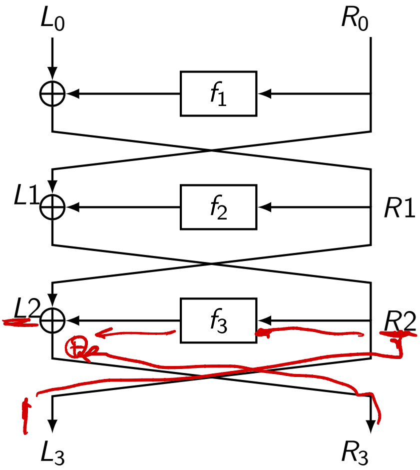
Feistel Networks – Making f Invertible

Evaluating round i:

- Split input in half: $L_{i-1}||R_{i-1}$
- $L_i = R_{i-1}$
- $R_i = L_{i-1} \oplus f_i(R_{i-1})$

Inverting round i:

$R_2 = L_3$



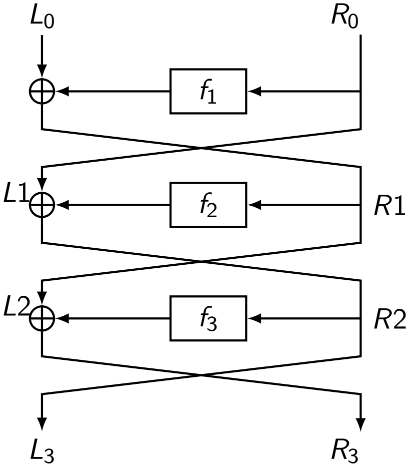
Feistel Networks – Making f Invertible

Evaluating round i:

- Split input in half: $L_{i-1}||R_{i-1}$
- $L_i = R_{i-1}$
- $R_i = L_{i-1} \oplus f_i(R_{i-1})$

Inverting round i:

- Set $R_{i-1} = L_i$



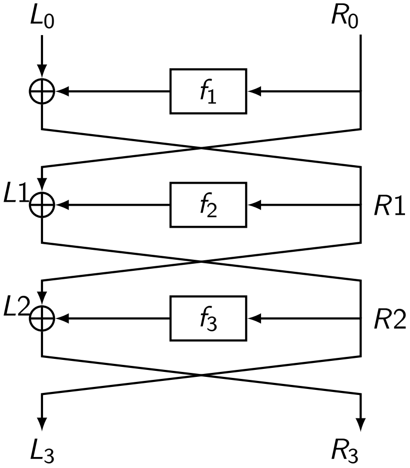
Feistel Networks – Making f Invertible

Evaluating round i:

- Split input in half: $L_{i-1}||R_{i-1}$
- $L_i = R_{i-1}$
- $R_i = L_{i-1} \oplus f_i(R_{i-1})$

Inverting round i:

- Set $R_{i-1} = L_i$
- Set $L_{i-1} = R_i \oplus f_i(R_{i-1})$



Feistel Networks – Making f Invertible

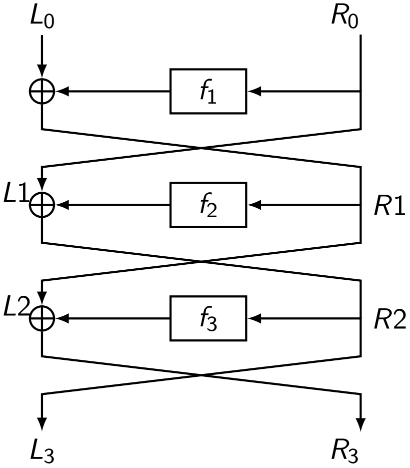
Evaluating round i:

- Split input in half: $L_{i-1}||R_{i-1}$
- $L_i = R_{i-1}$
- $R_i = L_{i-1} \oplus f_i(R_{i-1})$

Inverting round i:

- Set $R_{i-1} = L_i$
- Set $L_{i-1} = R_i \oplus f_i(R_{i-1})$

Round Functions:



Feistel Networks – Making f Invertible

Evaluating round i:

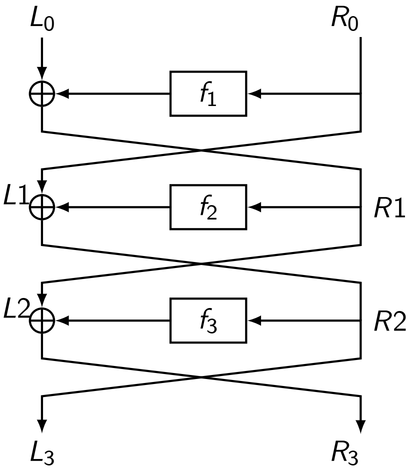
- Split input in half: $L_{i-1}||R_{i-1}$
- $L_i = R_{i-1}$
- $R_i = L_{i-1} \oplus f_i(R_{i-1})$

Inverting round i:

- Set $R_{i-1} = L_i$
- Set $L_{i-1} = R_i \oplus f_i(R_{i-1})$

Round Functions:

- Each round has function f_i



Feistel Networks – Making f Invertible

Evaluating round i:

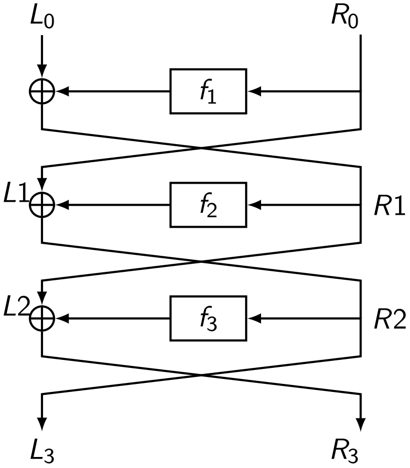
- Split input in half: $L_{i-1}||R_{i-1}$
- $L_i = R_{i-1}$
- $R_i = L_{i-1} \oplus f_i(R_{i-1})$

Inverting round i:

- Set $R_{i-1} = L_i$
- Set $L_{i-1} = R_i \oplus f_i(R_{i-1})$

Round Functions:

- Each round has function f_i
- f_i does not have to be invertible



Feistel Networks – Making f Invertible

Evaluating round i:

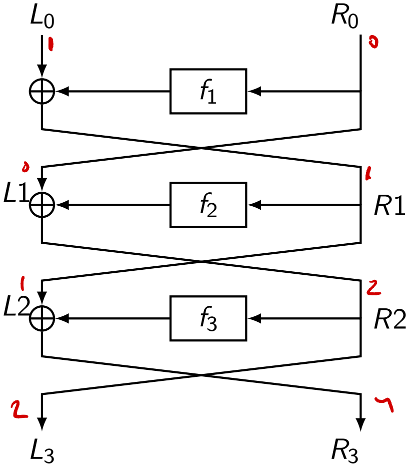
- Split input in half: $L_{i-1}||R_{i-1}$
- $L_i = R_{i-1}$
- $R_i = L_{i-1} \oplus f_i(R_{i-1})$

Inverting round i:

- Set $R_{i-1} = L_i$
- Set $L_{i-1} = R_i \oplus f_i(R_{i-1})$

Round Functions:

- Each round has function f_i
- f_i does not have to be invertible
- Usually f_i derived from global public f and round key k_i



Data Encryption Standard (DES) History

- Developed in 1970s by IBM + NSA
    - Used techniques not publicly known at the time

Data Encryption Standard (DES) History

- Developed in 1970s by IBM + NSA
  - Used techniques not publicly known at the time
- 16-round Feistel network with $\ell = 64$ (block length) and $n = 56$ (key length)

Data Encryption Standard (DES) History

- Developed in 1970s by IBM + NSA
  - Used techniques not publicly known at the time
- 16-round Feistel network with $\ell = 64$ (block length) and $n = 56$ (key length)
- There are no known successful cryptanalytic attacks on DES

Data Encryption Standard (DES) History

- Developed in 1970s by IBM + NSA
  - Used techniques not publicly known at the time
- 16-round Feistel network with $\ell = 64$ (block length) and $n = 56$ (key length)
- There are no known successful cryptanalytic attacks on DES
- But, $n = 56$ is too short, can be brute-forced on today's computers

Data Encryption Standard (DES) History

- Developed in 1970s by IBM + NSA
  - Used techniques not publicly known at the time
- 16-round Feistel network with $\ell = 64$ (block length) and $n = 56$ (key length)
- There are no known successful cryptanalytic attacks on DES
- But, $n = 56$ is too short, can be brute-forced on today's computers
- We will talk about how to extend the key later

DES Key Schedule

DES Key Schedule: Choosing round keys k_i from master key k

DES Key Schedule

DES Key Schedule: Choosing round keys k_i from master key k

- $k \in \{0,1\}^{56}$, $k_i \in \{0,1\}^{48}$

DES Key Schedule

DES Key Schedule: Choosing round keys k_i from master key k

- $k \in \{0,1\}^{56}$, $k_i \in \{0,1\}^{48}$
- Each k_i is a subset of bits of k

$$\underbrace{\overbrace{0110\ldots1}^{28\text{ bits}}}_{\text{pick }24\text{ bits}} \,||\, \underbrace{\overbrace{1001\ldots0}^{28\text{ bits}}}_{\text{pick }24\text{ bits}}$$

DES Key Schedule

DES Key Schedule: Choosing round keys k_i from master key k

- $k \in \{0,1\}^{56}$, $k_i \in \{0,1\}^{48}$
- Each k_i is a subset of bits of k

$$\underbrace{\overbrace{0110\ldots1}^{28\text{ bits}}}_{\text{pick }24\text{ bits}} \;||\; \underbrace{\overbrace{1001\ldots0}^{28\text{ bits}}}_{\text{pick }24\text{ bits}}$$

- Which bits chosen in each round is public

DES Mangler Function

DES round function f_i:

DES Mangler Function

DES round function f_i:

- $f_i(R) = \hat{f}(k_i, R)$, $k_i \in \{0,1\}^{48}, R \in \{0,1\}^{32}$

DES Mangler Function

DES round function f_i:

- $f_i(R) = \hat{f}(k_i, R)$, $k_i \in \{0,1\}^{48}, R \in \{0,1\}^{32}$
- Same $\hat{f}$ used in all 16 rounds

DES Mangler Function

DES round function f_i:

- $f_i(R) = \hat{f}(k_i, R)$, $k_i \in \{0,1\}^{48}, R \in \{0,1\}^{32}$
- Same $\hat{f}$ used in all 16 rounds
- $\hat{f}$ is a substitution-permutation network (SPN)

DES Mangler Function

DES round function f_i:

- $f_i(R) = \hat{f}(k_i, R)$, $k_i \in \{0,1\}^{48}, R \in \{0,1\}^{32}$
- Same $\hat{f}$ used in all 16 rounds
- $\hat{f}$ is a substitution-permutation network (SPN)

Constructing f_i:

DES Mangler Function

DES round function f_i:

- $f_i(R) = \hat{f}(k_i, R)$, $k_i \in \{0,1\}^{48}, R \in \{0,1\}^{32}$
- Same $\hat{f}$ used in all 16 rounds
- $\hat{f}$ is a substitution-permutation network (SPN)

Constructing f_i:

1. Extend R to 48 bits by duplicating half of the bits of R: $R' = E(R)$

DES Mangler Function

DES round function f_i:

- $f_i(R) = \hat{f}(k_i, R)$, $k_i \in \{0,1\}^{48}, R \in \{0,1\}^{32}$
- Same $\hat{f}$ used in all 16 rounds
- $\hat{f}$ is a substitution-permutation network (SPN)

Constructing f_i:

1. Extend R to 48 bits by duplicating half of the bits of R: $R' = E(R)$
2. Key mixing – $R'' = R' \oplus k_i$

DES Mangler Function

DES round function f_i:

- $f_i(R) = \hat{f}(k_i, R)$, $k_i \in \{0,1\}^{48}, R \in \{0,1\}^{32}$
- Same $\hat{f}$ used in all 16 rounds
- $\hat{f}$ is a substitution-permutation network (SPN)

Constructing f_i:

1. Extend R to 48 bits by duplicating half of the bits of R: $R' = E(R)$
2. Key mixing – $R'' = R' \oplus k_i$
3. Substitution
   - Break R'' into 8 blocks of 6 bits
   - Pass each block through different S-box (6-bit to 4-bit) (not invertible)

DES Mangler Function

DES round function f_i:

- $f_i(R) = \hat{f}(k_i, R)$, $k_i \in \{0,1\}^{48}, R \in \{0,1\}^{32}$
- Same $\hat{f}$ used in all 16 rounds
- $\hat{f}$ is a substitution-permutation network (SPN)

Constructing f_i:

1. Extend R to 48 bits by duplicating half of the bits of R: $R' = E(R)$
2. Key mixing – $R'' = R' \oplus k_i$
3. Substitution
   - Break R'' into 8 blocks of 6 bits
   - Pass each block through different S-box (6-bit to 4-bit) (not invertible)
4. Permutation
   - Apply mixing permutation to 32-bit output

DES Mangler Function

DES round function f_i:

- $f_i(R) = \hat{f}(k_i, R)$, $k_i \in \{0,1\}^{48}, R \in \{0,1\}^{32}$
- Same $\hat{f}$ used in all 16 rounds
- $\hat{f}$ is a substitution-permutation network (SPN)

Constructing f_i:

1. Extend R to 48 bits by duplicating half of the bits of R: $R' = E(R)$
2. Key mixing – $R'' = R' \oplus k_i$
3. Substitution
   - Break R'' into 8 blocks of 6 bits
   - Pass each block through different S-box (6-bit to 4-bit) (not invertible)
4. Permutation
   - Apply mixing permutation to 32-bit output

Other DES Components

DES S-Boxes:

Other DES Components

DES S-Boxes:

- DES has 8 S-Boxes, each taking 6-bits to 4-bits

Other DES Components

DES S-Boxes:

- DES has 8 S-Boxes, each taking 6-bits to 4-bits
- These S-Boxes are very carefully designed to resist cryptanalysis

Other DES Components

DES S-Boxes:

- DES has 8 S-Boxes, each taking 6-bits to 4-bits
- These S-Boxes are very carefully designed to resist cryptanalysis
- Each S-Box is 4-to-1

Other DES Components

DES S-Boxes:

- DES has 8 S-Boxes, each taking 6-bits to 4-bits
- These S-Boxes are very carefully designed to resist cryptanalysis
- Each S-Box is 4-to-1
- Changing 1 bit of input changes ≥ 2 bits of output

Other DES Components

DES S-Boxes:

- DES has 8 S-Boxes, each taking 6-bits to 4-bits
- These S-Boxes are very carefully designed to resist cryptanalysis
- Each S-Box is 4-to-1
- Changing 1 bit of input changes ≥ 2 bits of output

Mixing Permutation:

Other DES Components

DES S-Boxes:

- DES has 8 S-Boxes, each taking 6-bits to 4-bits
- These S-Boxes are very carefully designed to resist cryptanalysis
- Each S-Box is 4-to-1
- Changing 1 bit of input changes ≥ 2 bits of output

Mixing Permutation:

- Output from any S-Box, affects input to 6 S-Boxes in next round

DES Avalanche Effect

$$
\begin{array}{lccccccccc}
 & (L_0, R_0) & & (L_1, R_1) & & (L_2, R_2) & & (L_3, R_3) & \\
 & (L'_0, R'_0) & \rightarrow & (L'_1, R'_1) & \rightarrow & (L'_2, R'_2) & \rightarrow & (L'_3, R'_3) & \cdots \\
\#\ \text{Bits different} & 1, 0 & & 0, 1 & & 1, 2 & & 2, 4 &
\end{array}
$$

- After 7 rounds all 32 bits in R affected
- After 8 rounds all 32 bits in L affected

Increasing Key Length

- Recall that DES had a key that was too short ($n = 56$)

Increasing Key Length

- Recall that DES had a key that was too short ($n = 56$)
- It is possible to brute-force DES by trying all possible keys

Increasing Key Length

- Recall that DES had a key that was too short ($n = 56$)
- It is possible to brute-force DES by trying all possible keys

Question

How can we increase the key length of a block cipher?

$DES_K(m)$

Double Encryption

Try 1: Double Encryption (2DES) – double the key size

$$F'_{k_1,k_2}(x) = F_{k_2}(F_{k_1}(x))$$

Double Encryption

Try 1: Double Encryption (2DES) – double the key size

$$F'_{k_1,k_2}(x) = F_{k_2}(F_{k_1}(x))$$

Meet-in-the-middle Attack: Recovers key in time 2^n

Double Encryption

Try 1: Double Encryption (2DES) – double the key size

$$F'_{k_1,k_2}(x) = F_{k_2}(F_{k_1}(x))$$

Meet-in-the-middle Attack: Recovers key in time 2^n

$\mathcal{A}$ gets (x, y) such that $y = F'_{k_1^*,k_2^*}(x) = F_{k_2^*}(F_{k_1^*}(x))$

Double Encryption

Try 1: Double Encryption (2DES) – double the key size

$$F'_{k_1,k_2}(x) = F_{k_2}(F_{k_1}(x))$$

Meet-in-the-middle Attack: Recovers key in time 2^n

$\mathcal{A}$ gets (x, y) such that $y = F'_{k_1^*,k_2^*}(x) = F_{k_2^*}(F_{k_1^*}(x))$

1. For each $k_1 \in \{0,1\}^n$, compute $z = F_{k_1}(x)$, store (z, k_1) in table L

Double Encryption

Try 1: Double Encryption (2DES) – double the key size

$$F'_{k_1,k_2}(x) = F_{k_2}(F_{k_1}(x))$$

Meet-in-the-middle Attack: Recovers key in time 2^n

$\mathcal{A}$ gets (x, y) such that $y = F'_{k_1^*,k_2^*}(x) = F_{k_2^*}(F_{k_1^*}(x))$

1. For each $k_1 \in \{0,1\}^n$, compute $z = F_{k_1}(x)$, store (z, k_1) in table L
2. For each $k_2 \in \{0,1\}^n$, compute $z = F^{-1}_{k_2}(y)$, store (z, k_2) in table L'

(x,y) $y = F_{k_2^*}(F_{k_1^*}(x))$

k_1	$F_{k_1}(x)$
$\vdots$	$\vdots$

$F_{k_1^*}(x)$

k_2	$F^{-1}_{k_2}(y)$
$\vdots$	$\vdots$

$F^{-1}_{k_2^*}(y)$

Double Encryption

Try 1: Double Encryption (2DES) – double the key size

$$F'_{k_1,k_2}(x) = F_{k_2}(F_{k_1}(x))$$

Meet-in-the-middle Attack: Recovers key in time 2^n
$\mathcal{A}$ gets (x, y) such that $y = F'_{k_1^*,k_2^*}(x) = F_{k_2^*}(F_{k_1^*}(x))$

1. For each $k_1 \in \{0,1\}^n$, compute $z = F_{k_1}(x)$, store (z, k_1) in table L
2. For each $k_2 \in \{0,1\}^n$, compute $z = F^{-1}_{k_2}(y)$, store (z, k_2) in table L'
3. Find all intersections s.t. $z_1 \in L = z_2 \in L'$ add (k_1, k_2) to table S

Double Encryption

Try 1: Double Encryption (2DES) – double the key size

$$F'_{k_1,k_2}(x) = F_{k_2}(F_{k_1}(x))$$

Meet-in-the-middle Attack: Recovers key in time 2^n
$\mathcal{A}$ gets (x, y) such that $y = F'_{k_1^*,k_2^*}(x) = F_{k_2^*}(F_{k_1^*}(x))$

1. For each $k_1 \in \{0,1\}^n$, compute $z = F_{k_1}(x)$, store (z, k_1) in table L
2. For each $k_2 \in \{0,1\}^n$, compute $z = F^{-1}_{k_2}(y)$, store (z, k_2) in table L'
3. Find all intersections s.t. $z_1 \in L = z_2 \in L'$ add (k_1, k_2) to table S

Observations:

- For any such k_1, k_2, we know that $F_{k_1}(x) = F^{-1}_{k_2}(y)$, so k_1, k_2 is a possible key for (x, y)

Double Encryption

Try 1: Double Encryption (2DES) – double the key size

$$F'_{k_1,k_2}(x) = F_{k_2}(F_{k_1}(x))$$

Meet-in-the-middle Attack: Recovers key in time 2^n
$\mathcal{A}$ gets (x, y) such that $y = F'_{k_1^*,k_2^*}(x) = F_{k_2^*}(F_{k_1^*}(x))$

1. For each $k_1 \in \{0,1\}^n$, compute $z = F_{k_1}(x)$, store (z, k_1) in table L
2. For each $k_2 \in \{0,1\}^n$, compute $z = F_{k_2}^{-1}(y)$, store (z, k_2) in table L'
3. Find all intersections s.t. $z_1 \in L = z_2 \in L'$ add (k_1, k_2) to table S

Observations:

- For any such k_1, k_2, we know that $F_{k_1}(x) = F_{k_2}^{-1}(y)$, so k_1, k_2 is a possible key for (x, y)
- Using a few more (x, y) pairs can uniquely identify k_1^*, k_2^* whp.

Double Encryption

Try 1: Double Encryption (2DES) – double the key size

$$F'_{k_1,k_2}(x) = F_{k_2}(F_{k_1}(x))$$

Meet-in-the-middle Attack: Recovers key in time 2^n

$\mathcal{A}$ gets (x, y) such that $y = F'_{k_1^*,k_2^*}(x) = F_{k_2^*}(F_{k_1^*}(x))$

1. For each $k_1 \in \{0,1\}^n$, compute $z = F_{k_1}(x)$, store (z, k_1) in table L
2. For each $k_2 \in \{0,1\}^n$, compute $z = F^{-1}_{k_2}(y)$, store (z, k_2) in table L'
3. Find all intersections s.t. $z_1 \in L = z_2 \in L'$ add (k_1, k_2) to table S

Observations:

- For any such k_1, k_2, we know that $F_{k_1}(x) = F^{-1}_{k_2}(y)$, so k_1, k_2 is a possible key for (x, y)
- Using a few more (x, y) pairs can uniquely identify k_1^*, k_2^* whp.
- This attack takes $O(2^n)$ time, so we are no better off than we were with just single encryption.

Triple Encryption (3DES)

Triple Encryption (3DES)

1. Triple encryption with 3 keys

$$F''_{k_1,k_2,k_3}(x) = F_{k_3}(F^{-1}_{k_2}(F_{k_1}(x)))$$

Triple Encryption (3DES)

1. Triple encryption with 3 keys

$$F''_{k_1,k_2,k_3}(x) = F_{k_3}(F^{-1}_{k_2}(F_{k_1}(x)))$$

2. Triple encryption with 2 keys

$$F''_{k_1,k_2}(x) = F_{k_1}(F^{-1}_{k_2}(F_{k_1}(x)))$$

Triple Encryption (3DES)

1. Triple encryption with 3 keys

$$F''_{k_1,k_2,k_3}(x) = F_{k_3}(F^{-1}_{k_2}(F_{k_1}(x)))$$

2. Triple encryption with 2 keys

$$F''_{k_1,k_2}(x) = F_{k_1}(F^{-1}_{k_2}(F_{k_1}(x)))$$

Observations:

- Second call to F_k is reversed for backward compatibility: $F''_{k_1,k_1,k_1} = F_{k_1}$

Triple Encryption (3DES)

1. Triple encryption with 3 keys

$$F''_{k_1,k_2,k_3}(x) = F_{k_3}(F^{-1}_{k_2}(F_{k_1}(x)))$$

2. Triple encryption with 2 keys

$$F''_{k_1,k_2}(x) = F_{k_1}(F^{-1}_{k_2}(F_{k_1}(x)))$$

Observations:

- Second call to F_k is reversed for backward compatibility: $F''_{k_1,k_1,k_1} = F_{k_1}$
- Both variants have 2^{2n} security (this is good enough, 112-bits)

Triple Encryption (3DES)

1. Triple encryption with 3 keys

$$F''_{k_1,k_2,k_3}(x) = F_{k_3}(F^{-1}_{k_2}(F_{k_1}(x)))$$

2. Triple encryption with 2 keys

$$F''_{k_1,k_2}(x) = F_{k_1}(F^{-1}_{k_2}(F_{k_1}(x)))$$

Observations:

- Second call to F_k is reversed for backward compatibility: $F''_{k_1,k_1,k_1} = F_{k_1}$
- Both variants have 2^{2n} security (this is good enough, 112-bits)
- In 3-key version, can still do meet-in-the-middle, but takes $O(2^{2n})$ keys to do it

Triple Encryption (3DES)

1. Triple encryption with 3 keys

$$F''_{k_1,k_2,k_3}(x) = F_{k_3}(F^{-1}_{k_2}(F_{k_1}(x)))$$

2. Triple encryption with 2 keys

$$F''_{k_1,k_2}(x) = F_{k_1}(F^{-1}_{k_2}(F_{k_1}(x)))$$

Observations:

- Second call to F_k is reversed for backward compatibility: $F''_{k_1,k_1,k_1} = F_{k_1}$
- Both variants have 2^{2n} security (this is good enough, 112-bits)
- In 3-key version, can still do meet-in-the-middle, but takes $O(2^{2n})$ keys to do it
- In 2-key version, meet-in-the-middle still takes $O(2^{2n})$ time, but can also do enumeration of keys in $O(2^{2n})$ time

Outline

Davies-Meyer

Collision-resistant hash function from an (ideal) block-cipher:

Davies-Meyer

Collision-resistant hash function from an (ideal) block-cipher:

- We build a compression function: $h : \{0,1\}^{n+\ell} \to \{0,1\}^{\ell}$ from block cipher with n-bit key and ℓ-bit block

Davies-Meyer

Collision-resistant hash function from an (ideal) block-cipher:

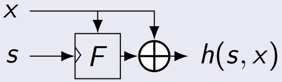
- We build a compression function: $h : \{0,1\}^{n+\ell} \to \{0,1\}^{\ell}$ from block cipher with n-bit key and ℓ-bit block
- Can turn this into arbitrary length hash (Merkle-Damgård)

Davies-Meyer

Collision-resistant hash function from an (ideal) block-cipher:

- We build a compression function: $h : \{0,1\}^{n+\ell} \to \{0,1\}^{\ell}$ from block cipher with n-bit key and ℓ-bit block
- Can turn this into arbitrary length hash (Merkle-Damgård)

Davies-Meyer Construction



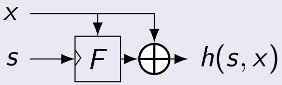
$$h(s,x) = F_s(x) \oplus x$$

Davies-Meyer

Collision-resistant hash function from an (ideal) block-cipher:

- We build a compression function: $h : \{0,1\}^{n+\ell} \to \{0,1\}^{\ell}$ from block cipher with n-bit key and ℓ-bit block
- Can turn this into arbitrary length hash (Merkle-Damgård)

Davies-Meyer Construction



$$h(s,x) = F_s(x) \oplus x$$

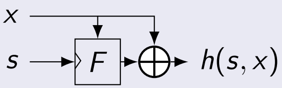
Properties:

Davies-Meyer

Collision-resistant hash function from an (ideal) block-cipher:

- We build a compression function: $h : \{0,1\}^{n+\ell} \to \{0,1\}^{\ell}$ from block cipher with n-bit key and ℓ-bit block
- Can turn this into arbitrary length hash (Merkle-Damgård)

Davies-Meyer Construction



$$h(s,x) = F_s(x) \oplus x$$

Properties:

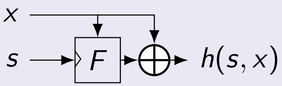
- Not known how to prove this secure based on F being a strong PRP

Davies-Meyer

Collision-resistant hash function from an (ideal) block-cipher:

- We build a compression function: $h : \{0,1\}^{n+\ell} \to \{0,1\}^{\ell}$ from block cipher with n-bit key and ℓ-bit block
- Can turn this into arbitrary length hash (Merkle-Damgård)

Davies-Meyer Construction



$$h(s,x) = F_s(x) \oplus x$$

Properties:

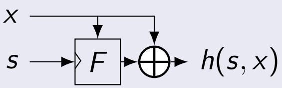
- Not known how to prove this secure based on F being a strong PRP
- Can prove secure if F is modeled as an ideal cipher
  - Randomly chosen cipher guaranteeing decryption correctness

Davies-Meyer

Collision-resistant hash function from an (ideal) block-cipher:

- We build a compression function: $h : \{0,1\}^{n+\ell} \rightarrow \{0,1\}^{\ell}$ from block cipher with n-bit key and ℓ-bit block
- Can turn this into arbitrary length hash (Merkle-Damgård)

Davies-Meyer Construction



$$h(s, x) = F_s(x) \oplus x$$

Properties:

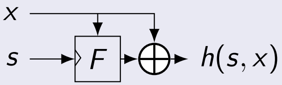
- Not known how to prove this secure based on F being a strong PRP
- Can prove secure if F is modeled as an ideal cipher
  - Randomly chosen cipher guaranteeing decryption correctness
  - Such a cipher has very strong security properties

Davies-Meyer

Collision-resistant hash function from an (ideal) block-cipher:

- We build a compression function: $h : \{0,1\}^{n+\ell} \to \{0,1\}^{\ell}$ from block cipher with n-bit key and ℓ-bit block
- Can turn this into arbitrary length hash (Merkle-Damgård)

Davies-Meyer Construction



$$h(s,x) = F_s(x) \oplus x$$

Properties:

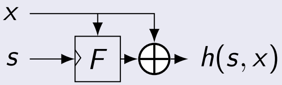
- Not known how to prove this secure based on F being a strong PRP
- Can prove secure if F is modeled as an ideal cipher
  - Randomly chosen cipher guaranteeing decryption correctness
  - Such a cipher has very strong security properties
- Insecure if instantiated with DES

Davies-Meyer

Collision-resistant hash function from an (ideal) block-cipher:

- We build a compression function: $h : \{0,1\}^{n+\ell} \to \{0,1\}^{\ell}$ from block cipher with n-bit key and ℓ-bit block
- Can turn this into arbitrary length hash (Merkle-Damgård)

Davies-Meyer Construction

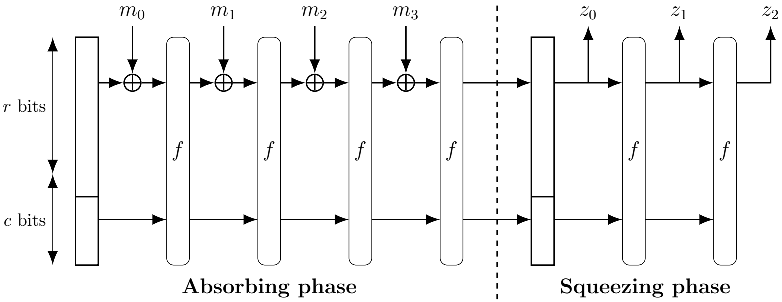


$$h(s,x) = F_s(x) \oplus x$$

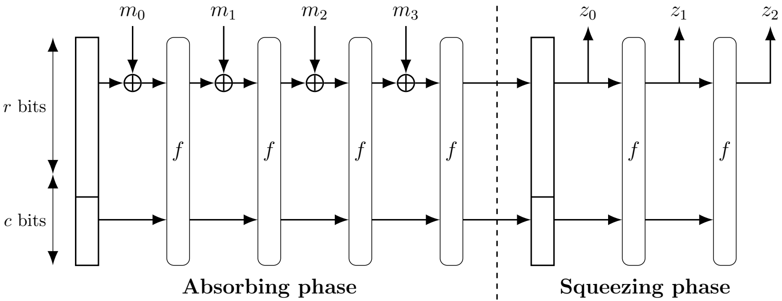
Properties:

- Not known how to prove this secure based on F being a strong PRP
- Can prove secure if F is modeled as an ideal cipher
    - Randomly chosen cipher guaranteeing decryption correctness
    - Such a cipher has very strong security properties
- Insecure if instantiated with DES
- But, there are block-ciphers for which this is believed to be secure

Sponge Function (Keccak/SHA-3)

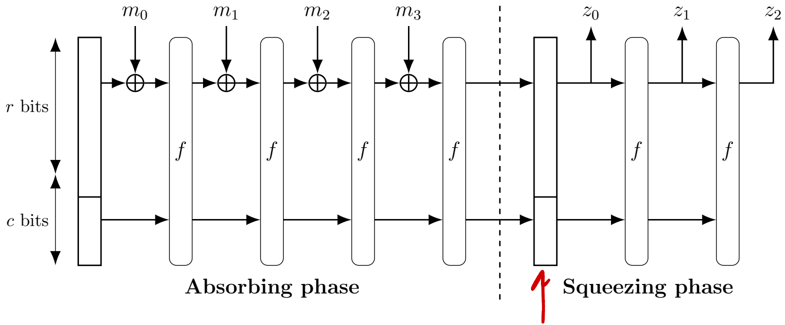


Sponge Function (Keccak/SHA-3)



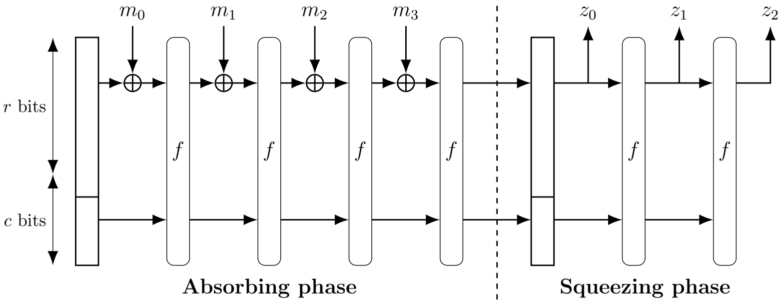
- More general than a hash function
  - Can have arbitrary length output
  - Can be used to build block-cipher, MAC, PRG, etc.

Sponge Function (Keccak/SHA-3)



- More general than a hash function
  - Can have arbitrary length output
  - Can be used to build block-cipher, MAC, PRG, etc.
- Uses a b-bit permutation f with $b = r + c$
  - r – rate
  - c – security parameter

Sponge Function (Keccak/SHA-3)



- More general than a hash function
  - Can have arbitrary length output
  - Can be used to build block-cipher, MAC, PRG, etc.
- Uses a b-bit permutation f with $b = r + c$
  - r – rate
  - c – security parameter
- Looks like a random function if f is random permutation