Data Encryption Standard (DES)

- Practical construction of a Block Cipher
- Developed by IBM in the 1970s
- Standardized in 1977

Data Encryption Standard (DES)

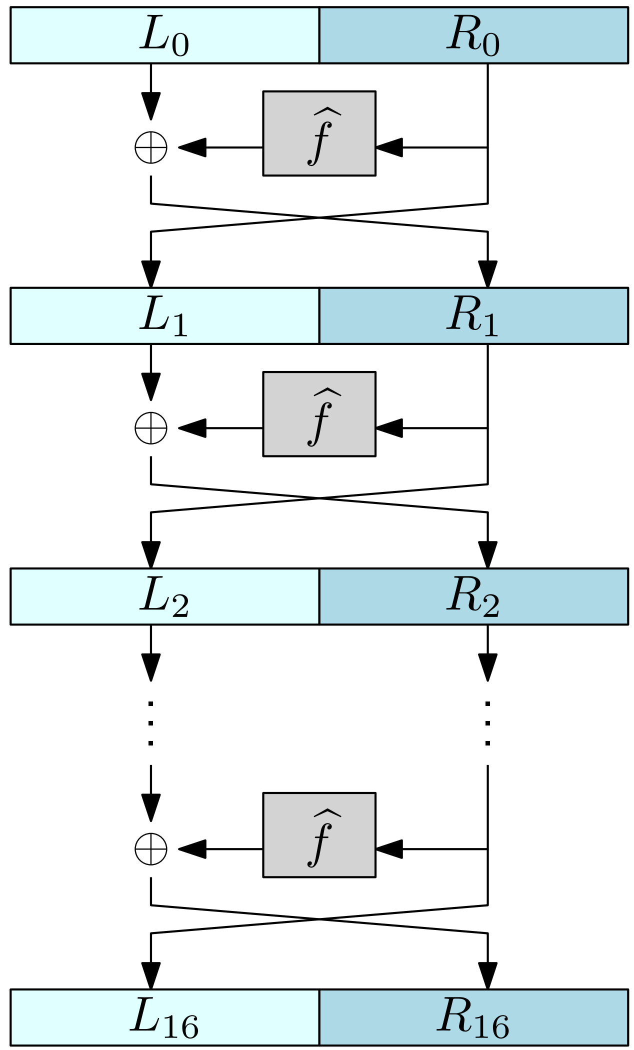
- Practical construction of a Block Cipher
- Developed by IBM in the 1970s
- Standardized in 1977
- 56-bit keys, 64-bit block length
- Extremely well-designed, no structural weaknesses, no practical attack better than bruteforce

Data Encryption Standard (DES)

- Practical construction of a Block Cipher
- Developed by IBM in the 1970s
- Standardized in 1977
- 56-bit keys, 64-bit block length
- Extremely well-designed, no structural weaknesses, no practical attack better than bruteforce
- Still considered *insecure* nowadays

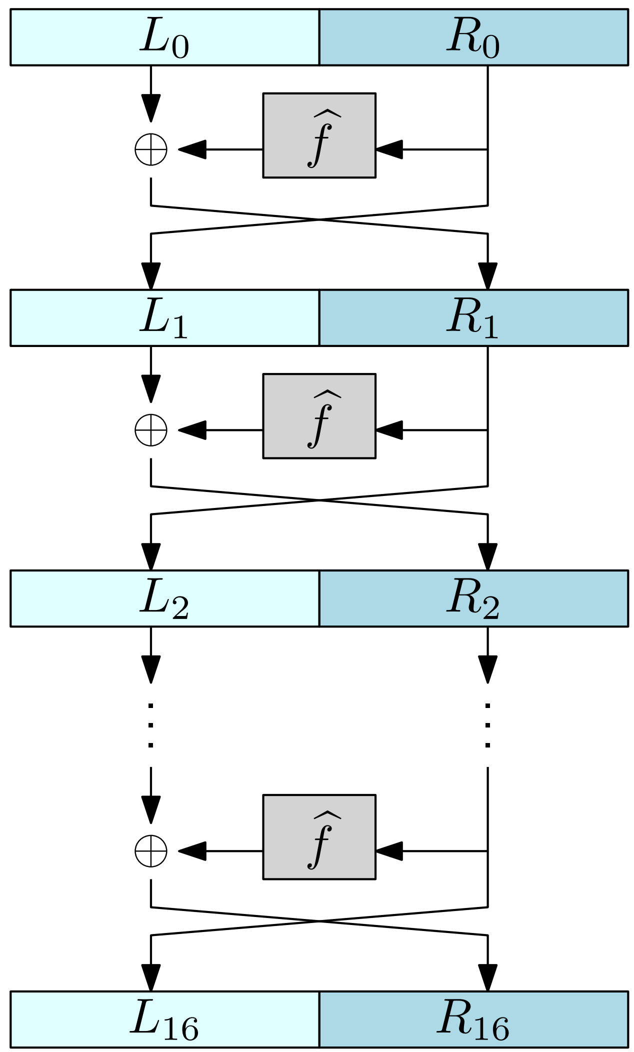
Structure of DES

- From a high-level perspective, DES is a Feistel network
- The number of rounds is 16



Structure of DES

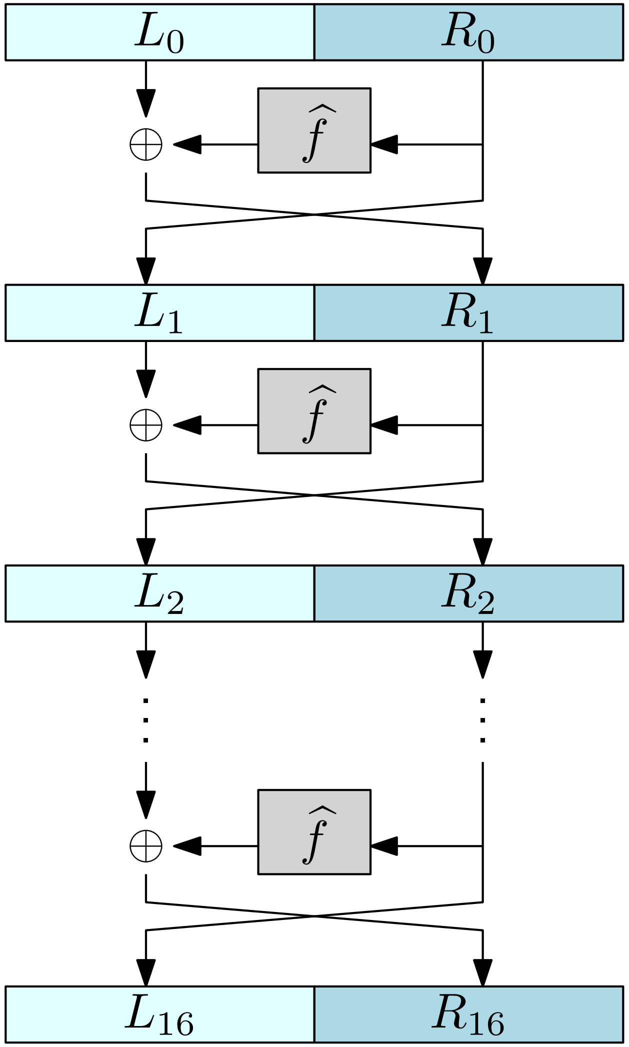
- From a high-level perspective, DES is a Feistel network
- The number of rounds is 16
- The same round function $\widehat{f}$ is used in all rounds (with different keys)



Structure of DES

- From a high-level perspective, DES is a Feistel network
- The number of rounds is 16
- The same round function $\widehat{f}$ is used in all rounds (with different keys)
- The function $\widetilde{f}$ takes a 48-bit sub-key and a 32-bit (half the block length) input

$$\widehat{f} : \{0,1\}^{48} \times \{0,1\}^{32} \to \{0,1\}^{32}$$

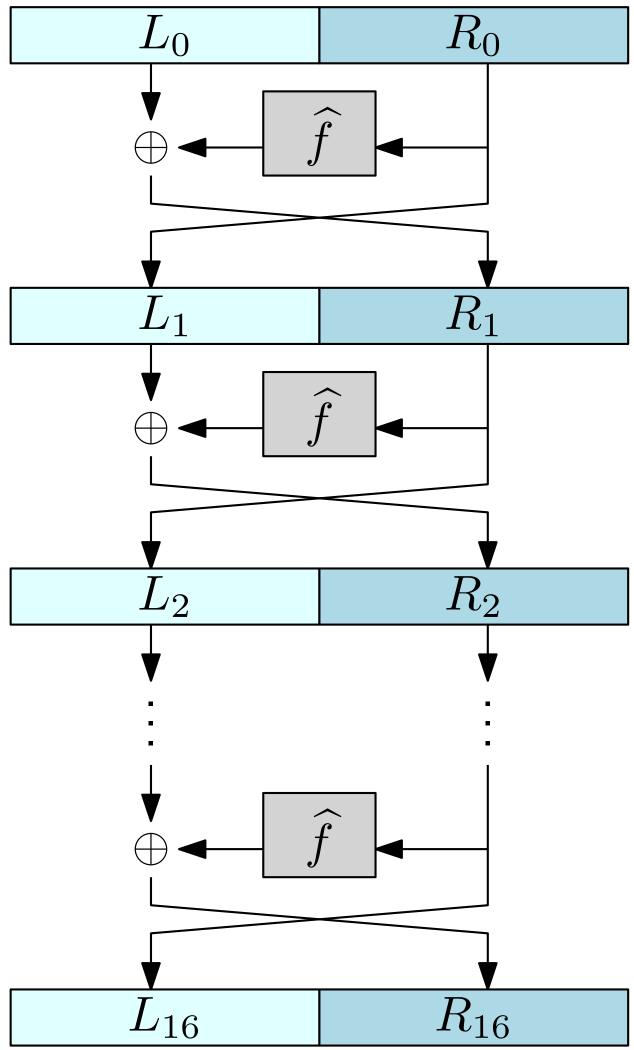


Structure of DES

- From a high-level perspective, DES is a Feistel network
- The number of rounds is 16
- The same round function $\widehat{f}$ is used in all rounds (with different keys)
- The function $\widetilde{f}$ takes a 48-bit sub-key and a 32-bit (half the block length) input

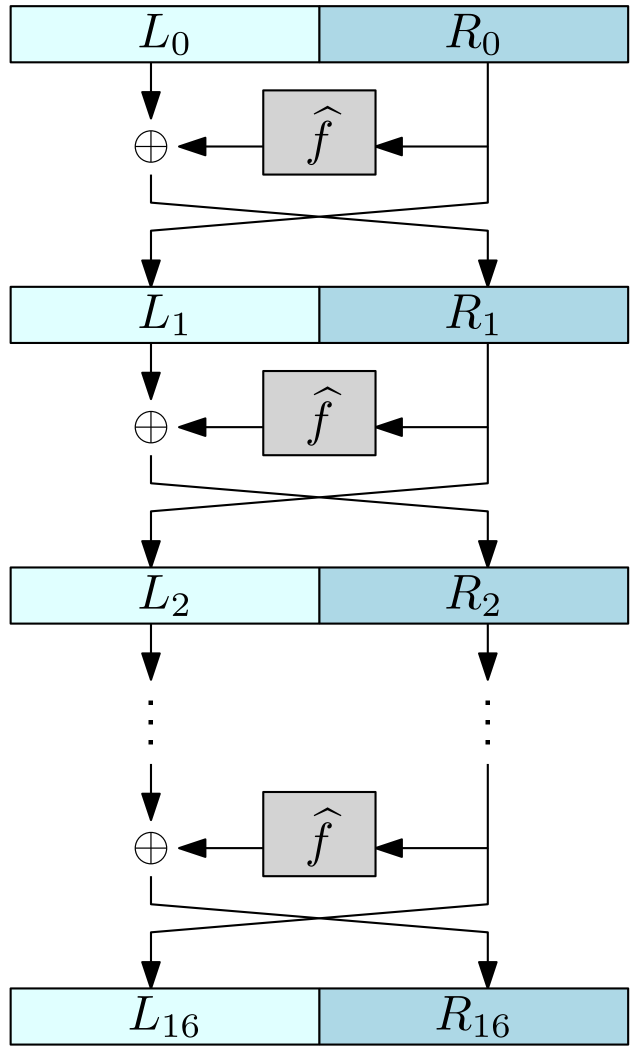
$$\widehat{f} : \{0,1\}^{48} \times \{0,1\}^{32} \rightarrow \{0,1\}^{32}$$

- The sub-keys are are formed by selecting and permuting a subset of 48-bit from the 56-bit master key
- The bit selection rule and the permutations are public, the only secret information is the master key itself



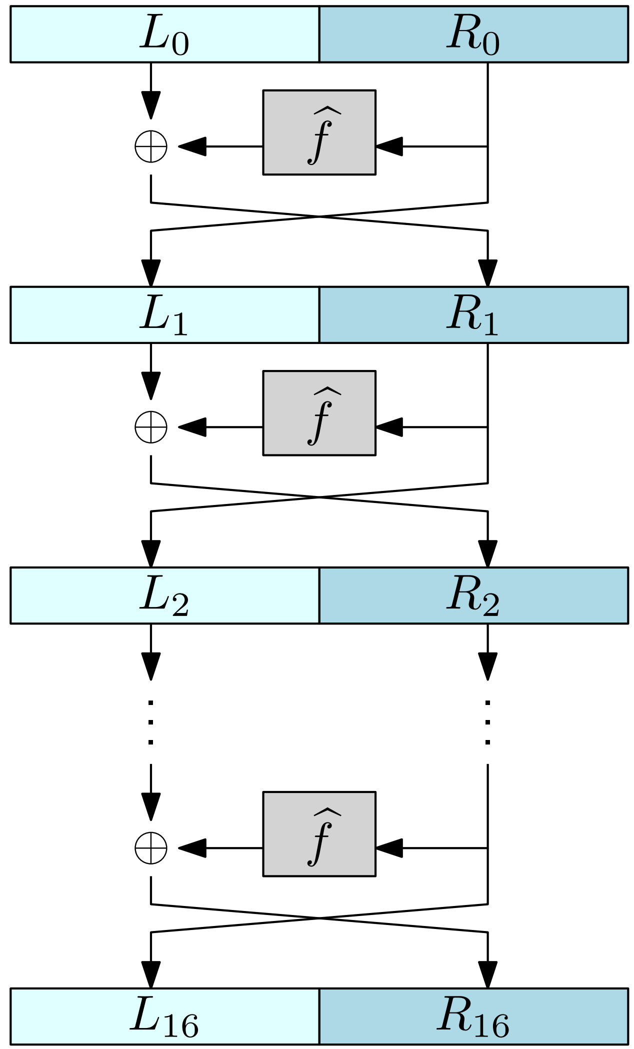
Structure of DES

- The function $\widehat{f}$ is called the **DES mangler function**
- First, the 32-bit input R_i to $\widehat{f}$ is expanded to a 48-bit input by duplicating some of the bits
- We denote the result by $R_i' = E(R_i)$ where E is called the **expansion function**



Structure of DES

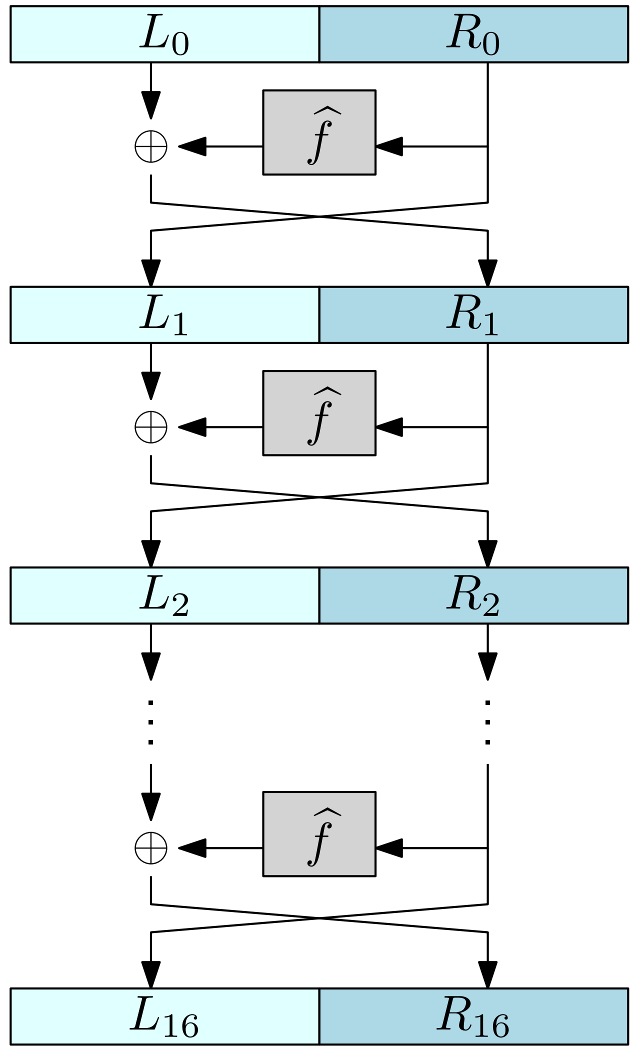
- The function $\widehat{f}$ is called the **DES mangler function**
- First, the 32-bit input R_i to $\widehat{f}$ is expanded to a 48-bit input by duplicating some of the bits
- We denote the result by $R_i' = E(R_i)$ where E is called the **expansion function**
- The rest of the function is just a one-round **substitution permutation network** that operates on the expanded input R'!



Structure of DES

- The function $\widehat{f}$ is called the **DES mangler function**
- First, the 32-bit input R_i to $\widehat{f}$ is expanded to a 48-bit input by duplicating some of the bits
- We denote the result by $R_i' = E(R_i)$ where E is called the **expansion function**
- The rest of the function is just a one-round **substitution permutation network** that operates on the expanded input R'!
- The S-boxes take 6-bit inputs and produce 4-bit outputs

  Note that the S-box is not a permutation

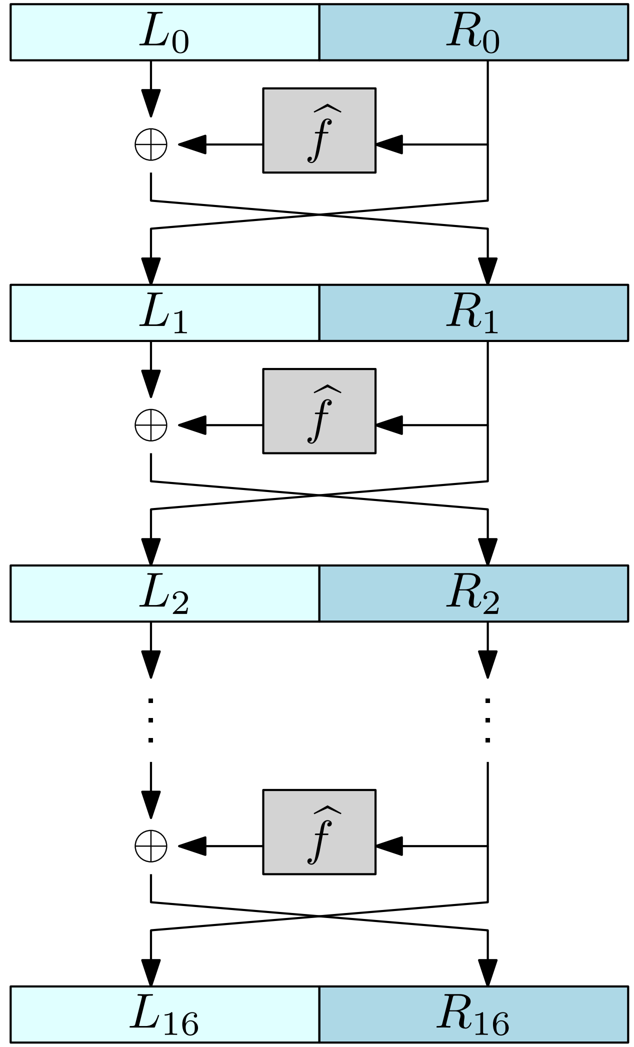


Structure of DES

- The function $\widehat{f}$ is called the **DES mangler function**
- First, the 32-bit input R_i to $\widehat{f}$ is expanded to a 48-bit input by duplicating some of the bits
- We denote the result by $R_i' = E(R_i)$ where E is called the **expansion function**
- The rest of the function is just a one-round **substitution permutation network** that operates on the expanded input R'!
- The S-boxes take 6-bit inputs and produce 4-bit outputs

  Note that the S-box is not a permutation

  $\Longrightarrow$ The function computed by the SPN is not a permutation



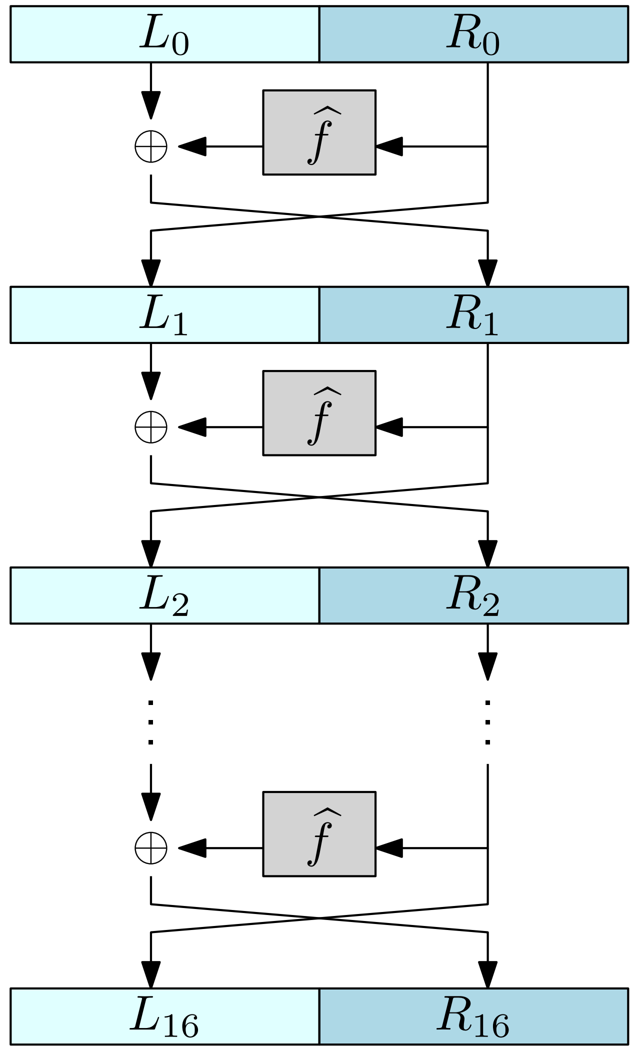
Structure of DES

- The function $\widehat{f}$ is called the **DES mangler function**
- First, the 32-bit input R_i to $\widehat{f}$ is expanded to a 48-bit input by duplicating some of the bits
- We denote the result by $R_i' = E(R_i)$ where E is called the **expansion function**
- The rest of the function is just a one-round **substitution permutation network** that operates on the expanded input R'!
- The S-boxes take 6-bit inputs and produce 4-bit outputs

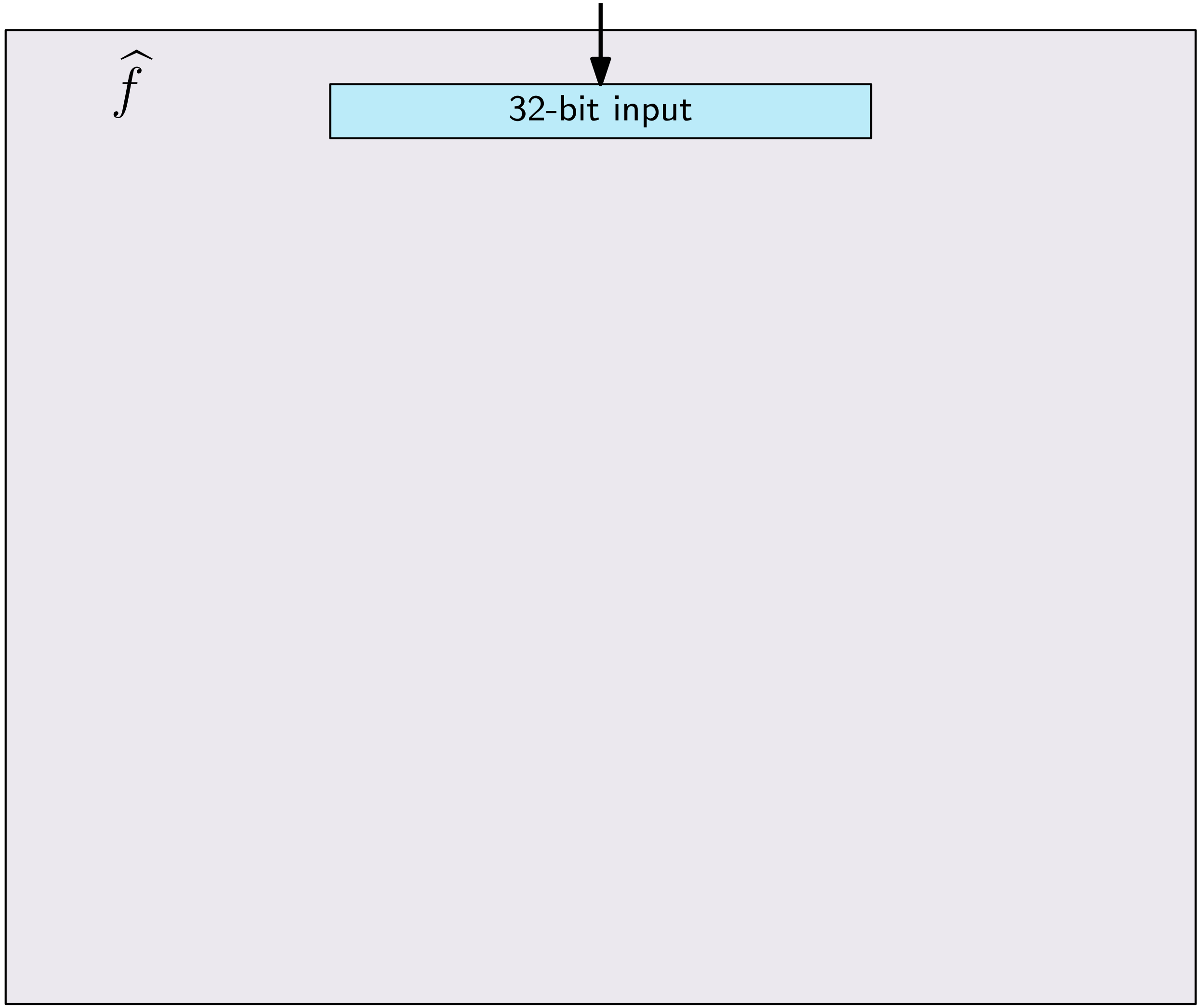
  Note that the S-box is not a permutation

  $\Longrightarrow$ The function computed by the SPN is not a permutation

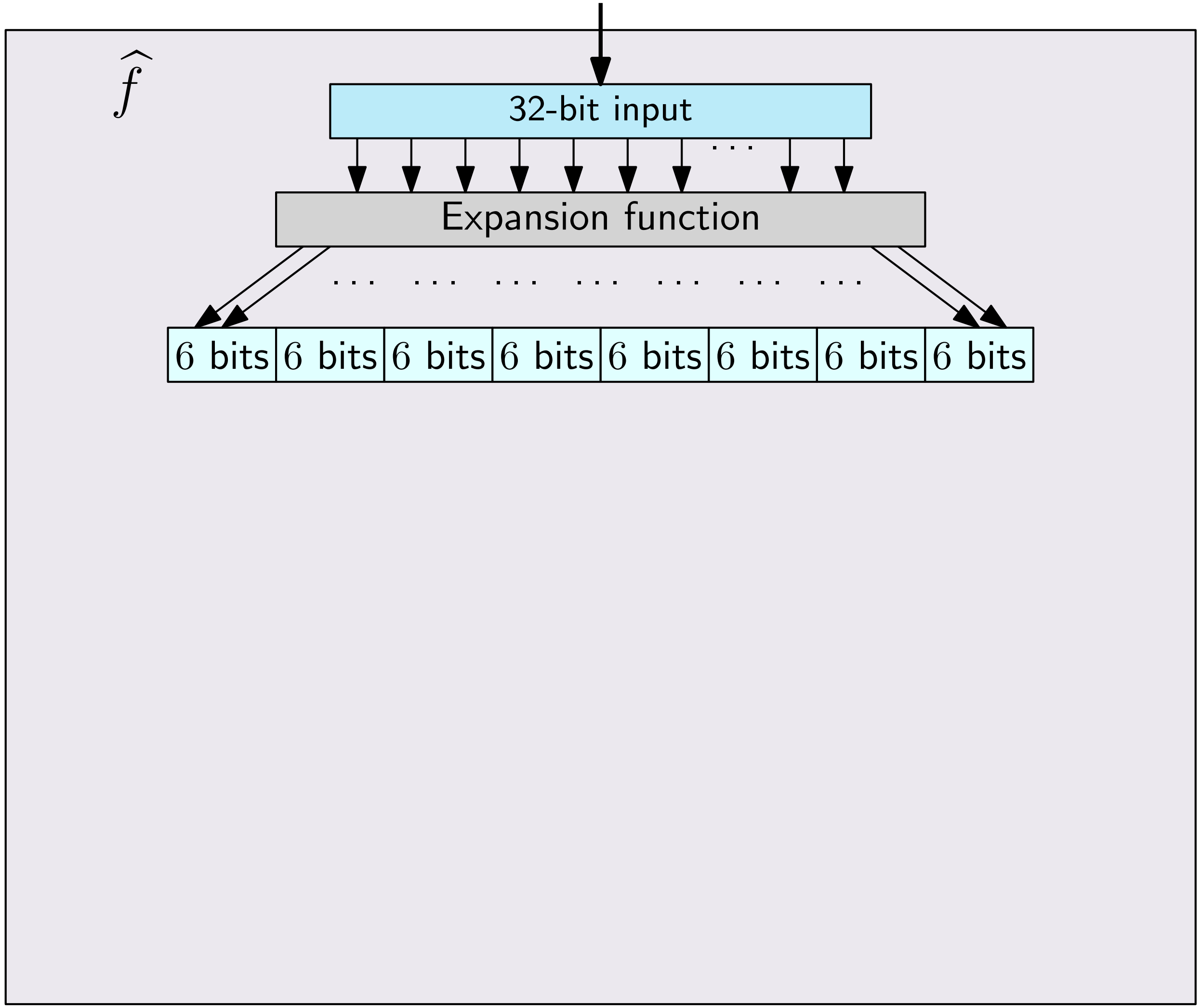
  This is not a problem, since Feistel networks do not require the round function to be PRP



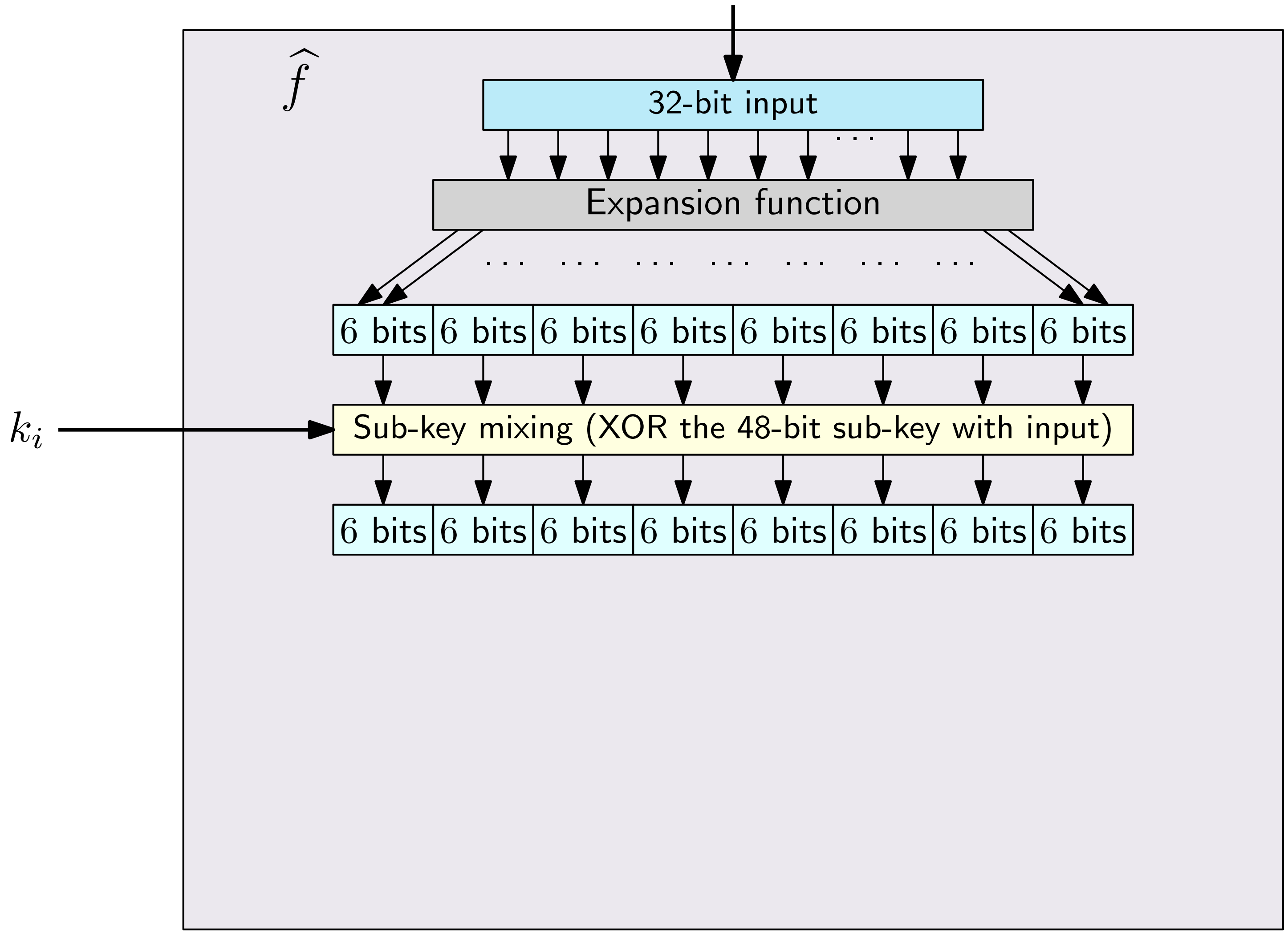
DES mangler function



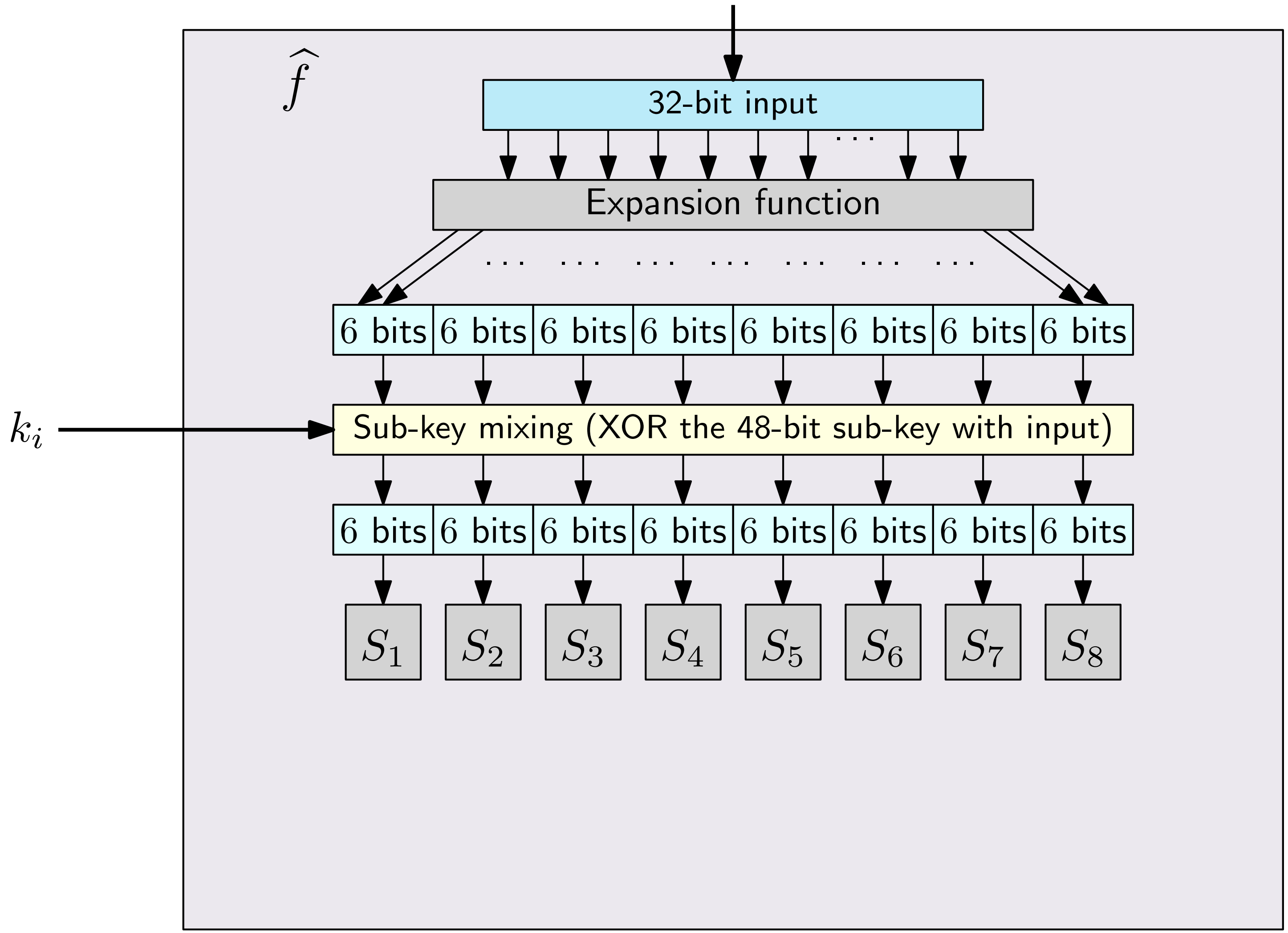
DES mangler function



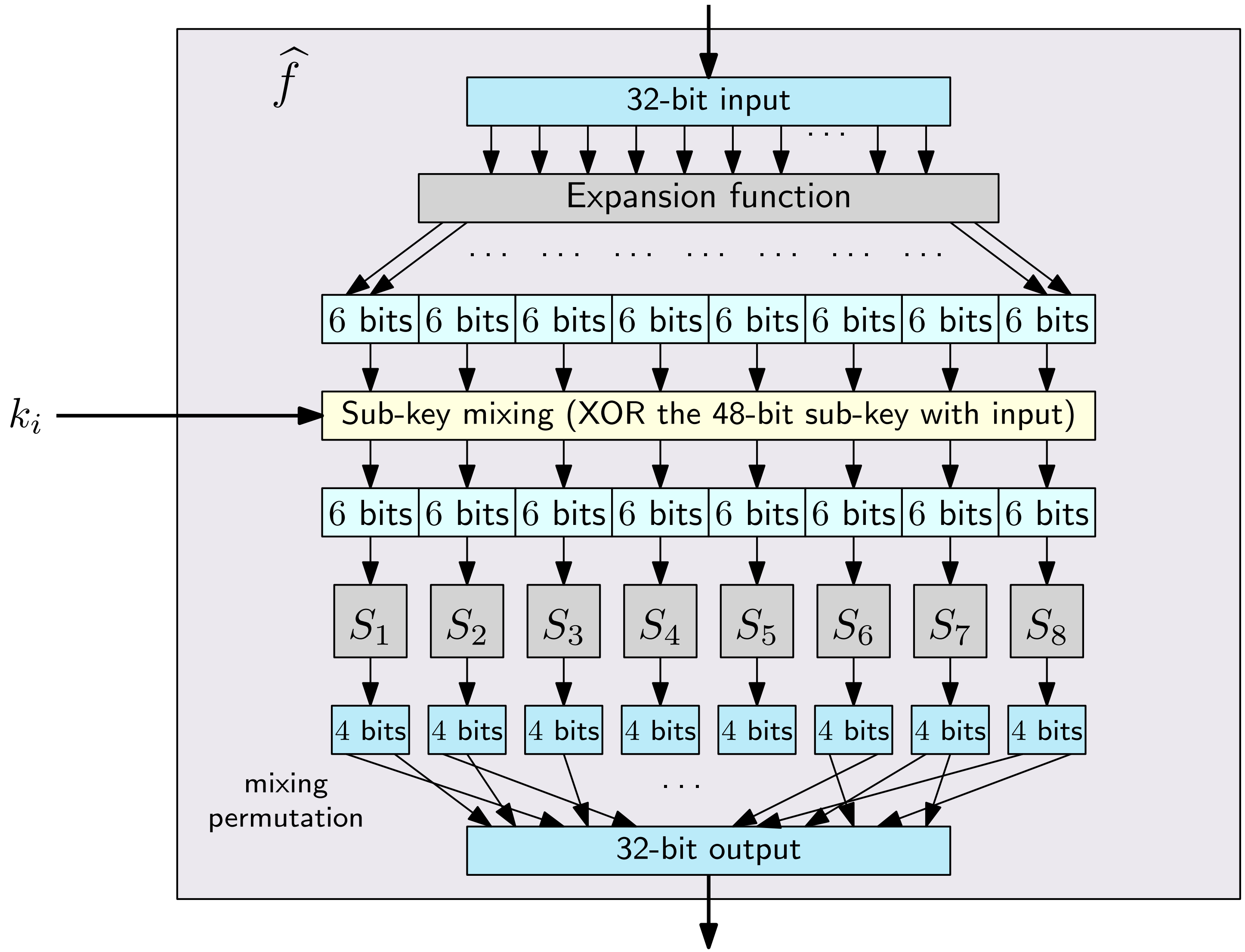
DES mangler function



DES mangler function



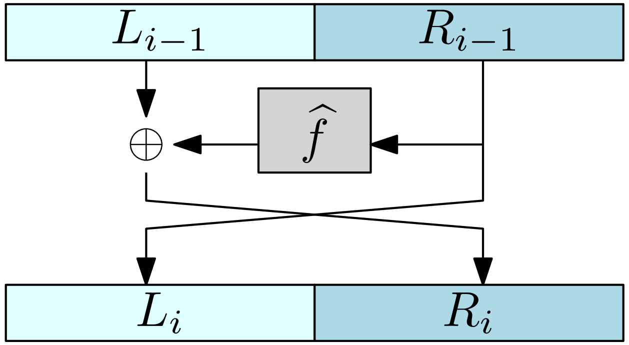
DES mangler function



DES avalanche effect

Confusion:

- The S-boxes map exactly 4 of the $2^6 = 64$ possible inputs to each of the $2^4 = 16$ possible outputs
- Changing 1 bit of input changes at least 2 bits of output



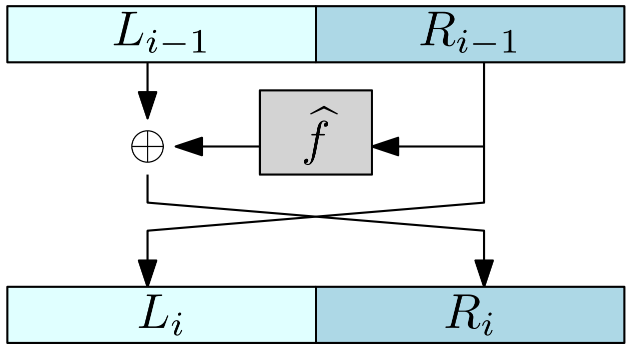
DES avalanche effect

Confusion:

- The S-boxes map exactly 4 of the $2^6 = 64$ possible inputs to each of the $2^4 = 16$ possible outputs
- Changing 1 bit of input changes at least 2 bits of output

Diffusion:

- The mixing permutation ensures that the 4 bits output from any S-box affect the inputs of 6 S-boxes in the next round



DES avalanche effect

Confusion:

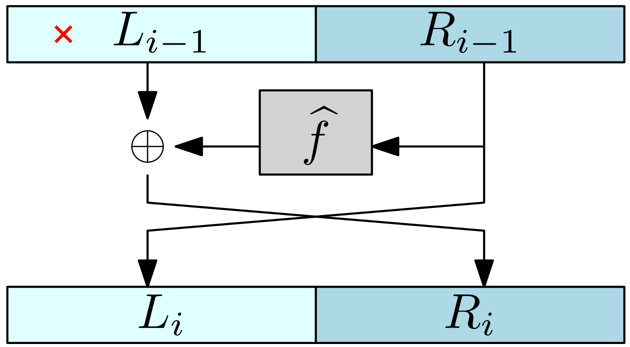
- The S-boxes map exactly 4 of the $2^6 = 64$ possible inputs to each of the $2^4 = 16$ possible outputs
- Changing 1 bit of input changes at least 2 bits of output

Diffusion:

- The mixing permutation ensures that the 4 bits output from any S-box affect the inputs of 6 S-boxes in the next round

Example of the avalanche effect:

- Consider a 1-bit difference in the left half of the input



DES avalanche effect

Confusion:

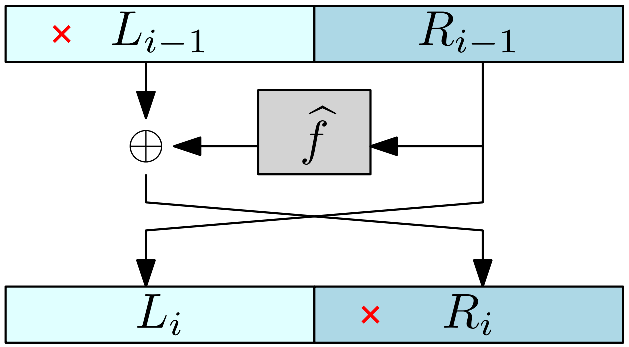
- The S-boxes map exactly 4 of the $2^6 = 64$ possible inputs to each of the $2^4 = 16$ possible outputs
- Changing 1 bit of input changes at least 2 bits of output

Diffusion:

- The mixing permutation ensures that the 4 bits output from any S-box affect the inputs of 6 S-boxes in the next round

Example of the avalanche effect:

- Consider a 1-bit difference in the left half of the input
- After 1 round of the Feistel Network, 1-bit difference in R



DES avalanche effect

Confusion:

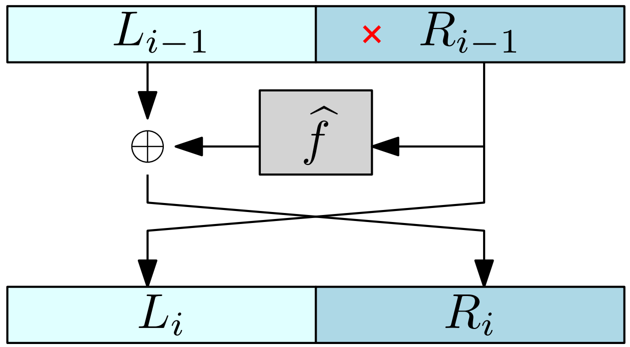
- The S-boxes map exactly 4 of the $2^6 = 64$ possible inputs to each of the $2^4 = 16$ possible outputs
- Changing 1 bit of input changes at least 2 bits of output

Diffusion:

- The mixing permutation ensures that the 4 bits output from any S-box affect the inputs of 6 S-boxes in the next round

Example of the avalanche effect:

- Consider a 1-bit difference in the left half of the input
- After 1 round of the Feistel Network, 1-bit difference in R
- Next round the S-boxes cause a 2-bit difference in R (plus 1 bit difference in L)



DES avalanche effect

Confusion:

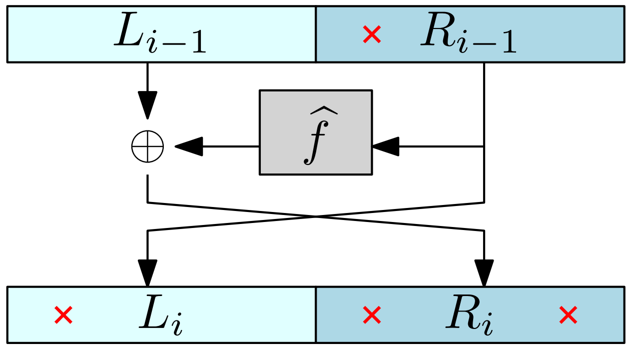
- The S-boxes map exactly 4 of the $2^6 = 64$ possible inputs to each of the $2^4 = 16$ possible outputs
- Changing 1 bit of input changes at least 2 bits of output

Diffusion:

- The mixing permutation ensures that the 4 bits output from any S-box affect the inputs of 6 S-boxes in the next round

Example of the avalanche effect:

- Consider a 1-bit difference in the left half of the input
- After 1 round of the Feistel Network, 1-bit difference in R
- Next round the S-boxes cause a 2-bit difference in R (plus 1 bit difference in L)



DES avalanche effect

Confusion:

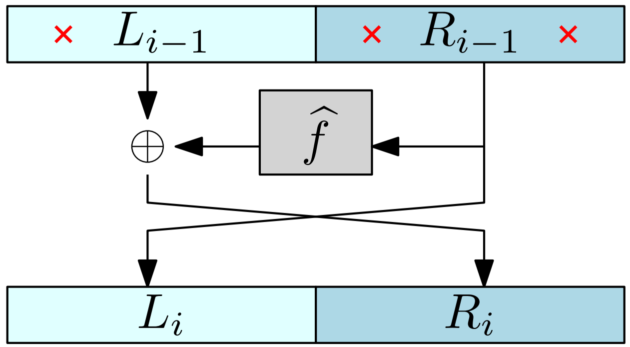
- The S-boxes map exactly 4 of the $2^6 = 64$ possible inputs to each of the $2^4 = 16$ possible outputs
- Changing 1 bit of input changes at least 2 bits of output

Diffusion:

- The mixing permutation ensures that the 4 bits output from any S-box affect the inputs of 6 S-boxes in the next round

Example of the avalanche effect:

- Consider a 1-bit difference in the left half of the input
- After 1 round of the Feistel Network, 1-bit difference in R
- Next round the S-boxes cause a 2-bit difference in R (plus 1 bit difference in L)
- The mixing permutation ensures that the affected bits end up in different S-boxes for next round...



DES avalanche effect

Confusion:

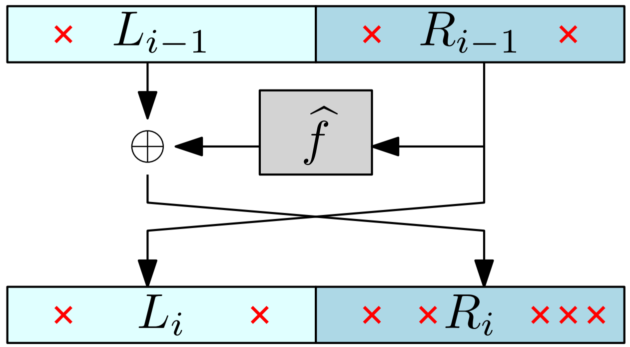
- The S-boxes map exactly 4 of the $2^6 = 64$ possible inputs to each of the $2^4 = 16$ possible outputs
- Changing 1 bit of input changes at least 2 bits of output

Diffusion:

- The mixing permutation ensures that the 4 bits output from any S-box affect the inputs of 6 S-boxes in the next round

Example of the avalanche effect:

- Consider a 1-bit difference in the left half of the input
- After 1 round of the Feistel Network, 1-bit difference in R
- Next round the S-boxes cause a 2-bit difference in R (plus 1 bit difference in L)
- The mixing permutation ensures that the affected bits end up in different S-boxes for next round...



Security of DES

- No practical attacks better than brute-force are known

  There are some other attacks but they require huge amounts of plaintext

Security of DES

- No practical attacks better than brute-force are known

  There are some other attacks but they require huge amounts of plaintext

- The weakness of DES comes from the fact that the **key length is too small**

Security of DES

- No practical attacks better than brute-force are known

  There are some other attacks but they require huge amounts of plaintext

- The weakness of DES comes from the fact that the **key length is too small**

- This concern was raised as soon as DES was released, although the computing power needed to break DES was not (readily) available

Security of DES

- No practical attacks better than brute-force are known

  There are some other attacks but they require huge amounts of plaintext

- The weakness of DES comes from the fact that the **key length is too small**
- This concern was raised as soon as DES was released, although the computing power needed to break DES was not (readily) available

Brute force search over 2^{56} kesys:

- In 1997: 96 days using thousands of computers (DESCHALL project)

Security of DES

- No practical attacks better than brute-force are known

  There are some other attacks but they require huge amounts of plaintext

- The weakness of DES comes from the fact that the **key length is too small**
- This concern was raised as soon as DES was released, although the computing power needed to break DES was not (readily) available

Brute force search over 2^{56} kesys:

- In 1997: 96 days using thousands of computers (DESCHALL project)

  41 days (distributed.net)

Security of DES

- No practical attacks better than brute-force are known

  There are some other attacks but they require huge amounts of plaintext

- The weakness of DES comes from the fact that the **key length is too small**
- This concern was raised as soon as DES was released, although the computing power needed to break DES was not (readily) available

Brute force search over 2^{56} kesys:

- In 1997: 96 days using thousands of computers (DESCHALL project)

  41 days (distributed.net)

- In 1998: 56 hours using a specialized 250 000 $ *Deep Crack* machine built by FSF

Security of DES

- No practical attacks better than brute-force are known

  There are some other attacks but they require huge amounts of plaintext

- The weakness of DES comes from the fact that the **key length is too small**
- This concern was raised as soon as DES was released, although the computing power needed to break DES was not (readily) available

Brute force search over 2^{56} kesys:

- In 1997: 96 days using thousands of computers (DESCHALL project)

  41 days (distributed.net)

- In 1998: 56 hours using a specialized 250 000 $ *Deep Crack* machine built by FSF
- Nowadays: 22 hours using 48 FPGAs (crack.sh), $> 100\,000$ $

Security of DES

Another concern of DES is the fact that the block length ℓ is just 64 bits

Security of DES

Another concern of DES is the fact that the block length ℓ is just 64 bits

- For example, we described CTR mode using $3\ell/4$ bits for the IV and $\ell/4$ bits for the counter
- This means that an IVs are 48 bits long

Security of DES

Another concern of DES is the fact that the block length ℓ is just 64 bits

- For example, we described CTR mode using $3\ell/4$ bits for the IV and $\ell/4$ bits for the counter
- This means that an IVs are 48 bits long
- If IVs are chosen uniformly at random, a repetition occurrs after $2^{24} \approx 16$M IVs with probability $> 60\%$

Security of DES

Another concern of DES is the fact that the block length ℓ is just 64 bits

- For example, we described CTR mode using $3\ell/4$ bits for the IV and $\ell/4$ bits for the counter
- This means that an IVs are 48 bits long
- If IVs are chosen uniformly at random, a repetition occurrs after $2^{24} \approx 16$M IVs with probability $> 60\%$
- Also, messages longer than $2^{\ell/4} = 2^{16} = 65536$ blocks require changing IV

$$2^{16} \cdot 64\text{b} = 2^{22}\text{b} = 2^{19}\text{B} = 0.5\text{MB}$$

Security of DES

Another concern of DES is the fact that the block length ℓ is just 64 bits

- For example, we described CTR mode using $3\ell/4$ bits for the IV and $\ell/4$ bits for the counter
- This means that an IVs are 48 bits long
- If IVs are chosen uniformly at random, a repetition occurrs after $2^{24} \approx 16$M IVs with probability $> 60\%$
- Also, messages longer than $2^{\ell/4} = 2^{16} = 65536$ blocks require changing IV

$$2^{16} \cdot 64\text{b} = 2^{22}\text{b} = 2^{19}\text{B} = 0.5\text{MB}$$

- Probability of collision $> 60\%$ after encrypting 8TB

  (think, e.g., of full-disk encryption)

Increasing the key length?

Two approaches:

- **Option 1:** structural modification to DES to directly increase the key length

Increasing the key length?

Two approaches:

- **Option 1:** structural modification to DES to directly increase the key length

  E.g., leave the key length of the round function the same but use a 128-bit master key and a different key schedule

Increasing the key length?

Two approaches:

- **Option 1:** structural modification to DES to directly increase the key length

  E.g., leave the key length of the round function the same but use a 128-bit master key and a different key schedule

  Alternatively, use a longer mixing sub-key in each round and modify the expansion function E and the S-boxes to account for this

Increasing the key length?

Two approaches:

- **Option 1:** structural modification to DES to directly increase the key length

  E.g., leave the key length of the round function the same but use a 128-bit master key and a different key schedule

  Alternatively, use a longer mixing sub-key in each round and modify the expansion function E and the S-boxes to account for this

  Downside: DES has received extensive attention from the crypto community and withstood all* attempts at attacks

  If we modify DES, we lose all the confidence on its security we gained with the test of time

* There are some theoretical attacks but they are considered infeasible in practice

Increasing the key length?

Two approaches:

- **Option 1:** structural modification to DES to directly increase the key length

  E.g., leave the key length of the round function the same but use a 128-bit master key and a different key schedule

  Alternatively, use a longer mixing sub-key in each round and modify the expansion function E and the S-boxes to account for this

  Downside: DES has received extensive attention from the crypto community and withstood all* attempts at attacks

  If we modify DES, we lose all the confidence on its security we gained with the test of time

- **Option 2:** Build new block-ciphers that use (unmodified) DES as a black-box

* There are some theoretical attacks but they are considered infeasible in practice

Increasing the key length?

Two approaches:

- **Option 1:** structural modification to DES to directly increase the key length

  E.g., leave the key length of the round function the same but use a 128-bit master key and a different key schedule

  Alternatively, use a longer mixing sub-key in each round and modify the expansion function E and the S-boxes to account for this

  Downside: DES has received extensive attention from the crypto community and withstood all* attempts at attacks

  If we modify DES, we lose all the confidence on its security we gained with the test of time

- **Option 2:** Build new block-ciphers that use (unmodified) DES as a black-box

  E.g., double encryption? Triple encryption?

* There are some theoretical attacks but they are considered infeasible in practice

Double Encryption

Let $F : \{0,1\}^n \times \{0,1\}^\ell \to \{0,1\}^\ell$ be a block-cipher (with key length n and block length ℓ)

(for DES $n = 56$, $\ell = 64$)

Double Encryption

Let $F : \{0,1\}^n \times \{0,1\}^\ell \to \{0,1\}^\ell$ be a block-cipher (with key length n and block length ℓ)

(for DES $n = 56$, $\ell = 64$)

We can define $F' : \{0,1\}^{2n} \times \{0,1\}^\ell \to \{0,1\}^\ell$ as:

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

where $k_1, k_2 \in \{0,1\}^n$.

Double Encryption

Let $F : \{0,1\}^n \times \{0,1\}^\ell \to \{0,1\}^\ell$ be a block-cipher (with key length n and block length ℓ)

(for DES $n = 56$, $\ell = 64$)

We can define $F' : \{0,1\}^{2n} \times \{0,1\}^\ell \to \{0,1\}^\ell$ as:

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

where $k_1, k_2 \in \{0,1\}^n$.

Is F' "twice as strong" as F?

If the best attack on F takes time $\approx 2^n$, does the best attack on F' take time $\approx 2^{2n}$?

Meet-in-the-middle attack

There is a weakness that stems from the fact that F' can be "factored" into two independent components

Given a single input output pair (x, y), with $y = F'_{k_1^* \| k_2^*}(x)$, the adversary can:

Meet-in-the-middle attack

There is a weakness that stems from the fact that F' can be "factored" into two independent components

Given a single input output pair (x, y), with $y = F'_{k_1^* \| k_2^*}(x)$, the adversary can:

- Try all possible 2^n choices for k_1
  - For each k_1, compute $z = F_{k_1}(x)$
  - Store z in a dictionary with a fast lookup, keep k_1 as satellite data

    (easy solution: append all pairs (z, k_1) to a list, then sort the list in time $O(2^n \text{poly}(n))$)

Meet-in-the-middle attack

There is a weakness that stems from the fact that F' can be "factored" into two independent components

Given a single input output pair (x, y), with $y = F'_{k_1^* \| k_2^*}(x)$, the adversary can:

- Try all possible 2^n choices for k_1
  - For each k_1, compute $z = F_{k_1}(x)$
  - Store z in a dictionary with a fast lookup, keep k_1 as satellite data

    (easy solution: append all pairs (z, k_1) to a list, then sort the list in time $O(2^n \text{poly}(n))$)
- Try all possible 2^n choices for k_2
  - For each k_2, compute $z = F_{k_2}^{-1}(y)$
  - Check whether z is in the dictionary. If z is found retrieve the satellite data k_1 and output $k_1 \| k_2$ as a candidate key for F'

Meet-in-the-middle attack

Do we always output the real key $k_1^* \| k_2^*$?

Meet-in-the-middle attack

Do we always output the real key $k_1^* \| k_2^*$? **Yes!**

Meet-in-the-middle attack

Do we always output the real key $k_1^* \| k_2^*$? **Yes!**

- When the first loop considers $k_1 = k_1^*$, the computed value z is $z^* = F_{k_1^*}(x)$

Meet-in-the-middle attack

Do we always output the real key $k_1^* \| k_2^*$? **Yes!**

- When the first loop considers $k_1 = k_1^*$, the computed value z is $z^* = F_{k_1^*}(x)$
- When the second loop considers $k_2 = k_2^*$, the computed value z is

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

Meet-in-the-middle attack

Do we always output the real key $k_1^* \| k_2^*$? **Yes!**

- When the first loop considers $k_1 = k_1^*$, the computed value z is $z^* = F_{k_1^*}(x)$
- When the second loop considers $k_2 = k_2^*$, the computed value z is

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

Meet-in-the-middle attack

Do we always output the real key $k_1^* \| k_2^*$? **Yes!**

- When the first loop considers $k_1 = k_1^*$, the computed value z is $z^* = F_{k_1^*}(x)$
- When the second loop considers $k_2 = k_2^*$, the computed value z is

$$F_{k_2^*}^{-1}(y) = F_{k_2^*}^{-1}(\, F'_{k_1^* \| k_2^*}(x)\,) = F_{k_2^*}^{-1}(\, F_{k_2^*}(\, F_{k_1^*}(x)\,)\,) \ = F_{k_1^*}(x) = z^*$$

Meet-in-the-middle attack

Do we always output the real key $k_1^*\|k_2^*$? **Yes!**

- When the first loop considers $k_1 = k_1^*$, the computed value z is $z^* = F_{k_1^*}(x)$
- When the second loop considers $k_2 = k_2^*$, the computed value z is

$$F_{k_2^*}^{-1}(y) = F_{k_2^*}^{-1}(\, F'_{k_1^*\|k_2^*}(x)\,) = F_{k_2^*}^{-1}(\, F_{k_2^*}(\, F_{k_1^*}(x)\,)\,) \; = F_{k_1^*}(x) = z^*$$

- Therefore z^* is found in the dictionary and $k_1^*\|k_2^*$ is output

Meet-in-the-middle attack

Do we always output the real key $k_1^* \| k_2^*$? **Yes!**

- When the first loop considers $k_1 = k_1^*$, the computed value z is $z^* = F_{k_1^*}(x)$
- When the second loop considers $k_2 = k_2^*$, the computed value z is

$$F_{k_2^*}^{-1}(y) = F_{k_2^*}^{-1}(\, F'_{k_1^* \| k_2^*}(x)\,) \; = F_{k_2^*}^{-1}(\, F_{k_2^*}(\, F_{k_1^*}(x)\,)\,) \;\; = F_{k_1^*}(x) = z^*$$

- Therefore z^* is found in the dictionary and $k_1^* \| k_2^*$ is output

This is not enough...

Meet-in-the-middle attack

How many **other** candidate keys are output?

Meet-in-the-middle attack

How many **other** candidate keys are output?

- Let's model each F_{k_1} and $F_{k_2}^{-1}$ as a uniform random function: $\Pr[F_{k_1}(x) = F_{k_2}^{-1}(y)] = 2^{-\ell}$

Meet-in-the-middle attack

How many **other** candidate keys are output?

- Let's model each F_{k_1} and $F_{k_2}^{-1}$ as a uniform random function: $\Pr[F_{k_1}(x) = F_{k_2}^{-1}(y)] = 2^{-\ell}$
- Since there are $2^n \cdot 2^n = 2^{2n}$ pairs (k_1, k_2), the expected number of candidates (other than $k_1^* \| k_2^*$) is at most:

$$2^{2n} \cdot 2^{-\ell} = 2^{2n-\ell}$$

Meet-in-the-middle attack

How many **other** candidate keys are output?

- Let's model each F_{k_1} and $F_{k_2}^{-1}$ as a uniform random function: $\Pr[F_{k_1}(x) = F_{k_2}^{-1}(y)] = 2^{-\ell}$
- Since there are $2^n \cdot 2^n = 2^{2n}$ pairs (k_1, k_2), the expected number of candidates (other than $k_1^* \| k_2^*$) is at most:

$$2^{2n} \cdot 2^{-\ell} = 2^{2n-\ell}$$

For Double-DES: we reduced the possible keys from 2^{112} to $\approx 2^{48}$ in time $\approx 2^{56} = 2^n$

(recall that we were hoping for 2^{2n})

Meet-in-the-middle attack

How many **other** candidate keys are output?

- Let's model each F_{k_1} and $F_{k_2}^{-1}$ as a uniform random function: $\Pr[F_{k_1}(x) = F_{k_2}^{-1}(y)] = 2^{-\ell}$
- Since there are $2^n \cdot 2^n = 2^{2n}$ pairs (k_1, k_2), the expected number of candidates (other than $k_1^* \| k_2^*$) is at most:

$$2^{2n} \cdot 2^{-\ell} = 2^{2n-\ell}$$

For Double-DES: we reduced the possible keys from 2^{112} to $\approx 2^{48}$ in time $\approx 2^{56} = 2^n$
(recall that we were hoping for 2^{2n})

How to narrow the candidates down?

Meet-in-the-middle attack

How many **other** candidate keys are output?

- Let's model each F_{k_1} and $F_{k_2}^{-1}$ as a uniform random function: $\Pr[F_{k_1}(x) = F_{k_2}^{-1}(y)] = 2^{-\ell}$
- Since there are $2^n \cdot 2^n = 2^{2n}$ pairs (k_1, k_2), the expected number of candidates (other than $k_1^* \| k_2^*$) is at most:

$$2^{2n} \cdot 2^{-\ell} = 2^{2n-\ell}$$

For Double-DES: we reduced the possible keys from 2^{112} to $\approx 2^{48}$ in time $\approx 2^{56} = 2^n$
(recall that we were hoping for 2^{2n})

How to narrow the candidates down?

Repeat the attack with another pair (x, y) and look at the intersection of the candidates

What's the probability that a (wrong) pair of keys $k_1 \| k_2$ is a candidate **both** times?

Meet-in-the-middle attack

How many **other** candidate keys are output?

- Let's model each F_{k_1} and $F_{k_2}^{-1}$ as a uniform random function: $\Pr[F_{k_1}(x) = F_{k_2}^{-1}(y)] = 2^{-\ell}$
- Since there are $2^n \cdot 2^n = 2^{2n}$ pairs (k_1, k_2), the expected number of candidates (other than $k_1^* \| k_2^*$) is at most:

$$2^{2n} \cdot 2^{-\ell} = 2^{2n-\ell}$$

For Double-DES: we reduced the possible keys from 2^{112} to $\approx 2^{48}$ in time $\approx 2^{56} = 2^n$

(recall that we were hoping for 2^{2n})

How to narrow the candidates down?

Repeat the attack with another pair (x, y) and look at the intersection of the candidates

What's the probability that a (wrong) pair of keys $k_1 \| k_2$ is a candidate **both** times? $\approx 2^{-2\ell}$

Meet-in-the-middle attack

How many **other** candidate keys are output?

- Let's model each F_{k_1} and $F_{k_2}^{-1}$ as a uniform random function: $\Pr[F_{k_1}(x) = F_{k_2}^{-1}(y)] = 2^{-\ell}$
- Since there are $2^n \cdot 2^n = 2^{2n}$ pairs (k_1, k_2), the expected number of candidates (other than $k_1^* \| k_2^*$) is at most:

$$2^{2n} \cdot 2^{-\ell} = 2^{2n-\ell}$$

For Double-DES: we reduced the possible keys from 2^{112} to $\approx 2^{48}$ in time $\approx 2^{56} = 2^n$
(recall that we were hoping for 2^{2n})

How to narrow the candidates down?

Repeat the attack with another pair (x, y) and look at the intersection of the candidates

What's the probability that a (wrong) pair of keys $k_1 \| k_2$ is a candidate **both** times? $\approx 2^{-2\ell}$

$$2^{2n} \cdot 2^{-2\ell} = 2^{2n-2\ell} \qquad < 1 \text{ for Double-DES}$$

Triple Encryption

Double encryption is not more secure than a single encryption...

What about **triple** encrption?

Triple Encryption

Double encryption is not more secure than a single encryption...

What about **triple** encrption?

Two ways to define triple encryption:

- **Using three keys:** Pick three independent keys $k_1, k_2, k_3 \in \{0,1\}^n$ and let:

$$F''_{k_1\|k_2\|k_3}(x) = F_{k_3}(F_{k_2}(F_{k_1}(x)))$$

Triple Encryption

Double encryption is not more secure than a single encryption...

What about **triple** encrption?

Two ways to define triple encryption:

- **Using three keys:** Pick three independent keys $k_1, k_2, k_3 \in \{0,1\}^n$ and let:

$$F''_{k_1\|k_2\|k_3}(x) = F_{k_3}(F_{k_2}(F_{k_1}(x)))$$

  One would hope for all attacks to take time $\approx 2^{3n}$, but the scheme is still susceptible to a meet-in the middle attack

  How?

Triple Encryption

Double encryption is not more secure than a single encryption...

What about **triple** encrption?

Two ways to define triple encryption:

- **Using three keys:** Pick three independent keys $k_1, k_2, k_3 \in \{0,1\}^n$ and let:

$$F''_{k_1\|k_2\|k_3}(x) = F_{k_3}(F_{k_2}(F_{k_1}(x)))$$

  One would hope for all attacks to take time $\approx 2^{3n}$, but the scheme is still susceptible to a meet-in the middle attack

  How? Compute $F_{k_2}(F_{k_1}(x))$ and $F_{k_3}^{-1}(y)$ separately

Triple Encryption

Double encryption is not more secure than a single encryption...

What about **triple** encrption?

Two ways to define triple encryption:

- **Using three keys:** Pick three independent keys $k_1, k_2, k_3 \in \{0,1\}^n$ and let:

$$F''_{k_1 \| k_2 \| k_3}(x) = F_{k_3}(F_{k_2}(F_{k_1}(x)))$$

  One would hope for all attacks to take time $\approx 2^{3n}$, but the scheme is still susceptible to a meet-in the middle attack

  How? Compute $F_{k_2}(F_{k_1}(x))$ and $F^{-1}_{k_3}(y)$ separately

  Time: 2^{2n} (still an improvement over double encryption)

Triple Encryption

- **Using two keys:** Pick two independent keys $k_1, k_2 \in \{0,1\}^n$ and let:

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

Triple Encryption

- **Using two keys:** Pick two independent keys $k_1, k_2 \in \{0,1\}^n$ and let:

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

Triple Encryption

- **Using two keys:** Pick two independent keys $k_1, k_2 \in \{0,1\}^n$ and let:

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

Backwards compatible with single encryption:

$$F''_{k_1\|k_1}(x) = F_{k_1}(F^{-1}_{k_1}(F_{k_1}(x))) = F_{k_1}(x)$$

Triple Encryption

- **Using two keys:** Pick two independent keys $k_1, k_2 \in \{0,1\}^n$ and let:

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

Backwards compatible with single encryption:

$$F''_{k_1\|k_1}(x) = F_{k_1}(F^{-1}_{k_1}(F_{k_1}(x))) = F_{k_1}(x)$$

The key length is now $2n$.

If relatively few input-output pairs are known, then the best attack takes time 2^{2n}

Triple Encryption

- **Using two keys:** Pick two independent keys $k_1, k_2 \in \{0,1\}^n$ and let:

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

Backwards compatible with single encryption:

$$F''_{k_1 \| k_1}(x) = F_{k_1}(F^{-1}_{k_1}(F_{k_1}(x))) = F_{k_1}(x)$$

The key length is now $2n$.

If relatively few input-output pairs are known, then the best attack takes time 2^{2n}

Best possible given the key length!

Triple Encryption

- **Using two keys:** Pick two independent keys $k_1, k_2 \in \{0,1\}^n$ and let:

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

Backwards compatible with single encryption:

$$F''_{k_1\|k_1}(x) = F_{k_1}(F^{-1}_{k_1}(F_{k_1}(x))) = F_{k_1}(x)$$

The key length is now $2n$.

If relatively few input-output pairs are known, then the best attack takes time 2^{2n}

Best possible given the key length!

There are better attacks when many input-output pairs are known. If 2^t pairs are known then the key can be recovered in time

$$\approx 2^{n+\ell-t}$$

3DES

Triple encryption DES has been standardized in 1999 to try to overcome the small key-length of DES

- Two-key 3DES is no longer recommended (also due to the $\approx 2^{n+\ell-t}$ time known-plaintext attack)
- Three-key 3DES is still used, but it is advised to phase it out due to its small block length and the fact that it is slow to compute

3DES

Triple encryption DES has been standardized in 1999 to try to overcome the small key-length of DES

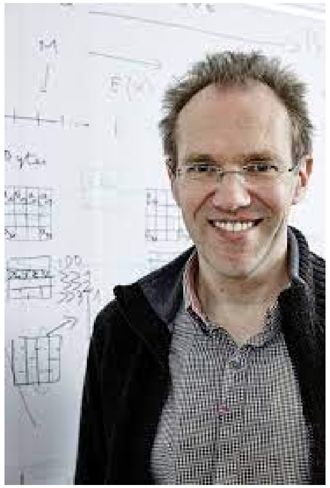
- Two-key 3DES is no longer recommended (also due to the $\approx 2^{n+\ell-t}$ time known-plaintext attack)
- Three-key 3DES is still used, but it is advised to phase it out due to its small block length and the fact that it is slow to compute

DES and 3DES have been superseded by the **Advanced Encryption Standard (AES)**

Advanced Encryption Standard (AES)

- Winner of a public competition by NIST (National Institute of Standards and Technology) in 1997
- The public and each team that submitted a cipher tried to find vulnerabilities in the (other) ciphers
- 5 finalist were selected, any of them would have been an excellent choice for the winner
- AES (whose name was Rijndael) has been selected based in part on properties such as efficiency, performance in hardware, flexibility, etc.

Vincent Rijmen



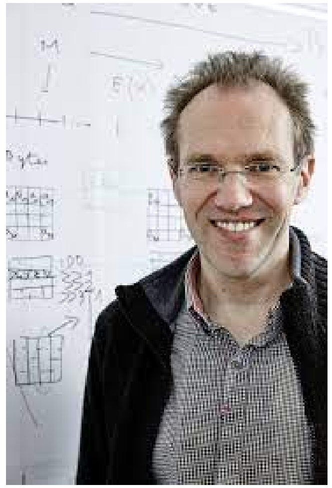
Joan Daemen

Advanced Encryption Standard (AES)

- Winner of a public competition by NIST (National Institute of Standards and Technology) in 1997
- The public and each team that submitted a cipher tried to find vulnerabilities in the (other) ciphers
- 5 finalist were selected, any of them would have been an excellent choice for the winner
- AES (whose name was Rijndael) has been selected based in part on properties such as efficiency, performance in hardware, flexibility, etc.

No significant weaknesses currently known!

Vincent Rijmen



Joan Daemen

Advanced Encryption Standard (AES)

- Block length of 128bits (vs. 64 of DES)
- Key lengths of 128, 192, and 256 (three different variants of AES) (vs. 56 of DES)

Advanced Encryption Standard (AES)

- Block length of 128bits (vs. 64 of DES)

- Key lengths of 128, 192, and 256 (three different variants of AES) (vs. 56 of DES)

- Its structure is a (slightly modified) SPN

- The number of rounds and the key schedule depend on the chosen variant (i.e., on the chosen key length)

Advanced Encryption Standard (AES)

- Block length of 128bits (vs. 64 of DES)
- Key lengths of 128, 192, and 256 (three different variants of AES) (vs. 56 of DES)
- Its structure is a (slightly modified) SPN
- The number of rounds and the key schedule depend on the chosen variant (i.e., on the chosen key length)
- The input is interepreted as a 4×4 matrix of bytes $(4 \cdot 4 \cdot 8 = 128)$, called the **state**

$$x = b_0\, b_1\, b_2\, b_3\, b_4\, b_5\, b_6\, b_7\, b_8\, b_9\, b_{10}\, b_{11}\, b_{12}\, b_{13}\, b_{14}\, b_{15} \qquad b_i \in \{0,1\}^8$$

Advanced Encryption Standard (AES)

- Block length of 128bits (vs. 64 of DES)
- Key lengths of 128, 192, and 256 (three different variants of AES) (vs. 56 of DES)
- Its structure is a (slightly modified) SPN
- The number of rounds and the key schedule depend on the chosen variant (i.e., on the chosen key length)
- The input is interepreted as a 4×4 matrix of bytes ($4 \cdot 4 \cdot 8 = 128$), called the **state**

$$x = b_0\, b_1\, b_2\, b_3\, b_4\, b_5\, b_6\, b_7\, b_8\, b_9\, b_{10}\, b_{11}\, b_{12}\, b_{13}\, b_{14}\, b_{15} \qquad b_i \in \{0,1\}^8$$

b_0	b_4	b_8	b_{12}
b_1	b_5	b_9	b_{13}
b_2	b_6	b_{10}	b_{14}
b_3	b_7	b_{11}	b_{15}

Advanced Encryption Standard (AES)

Each round of the SPN modifies the state by performing the following operations:

1) AddRoundKey: A 128-bit subkey is derived from the master key, viewed as a 4×4 matrix and XOR-ed with the state. This is the only step that depends on the key.

$$\begin{array}{|c|c|c|c|}\hline b_0 & b_4 & b_8 & b_{12} \\ \hline b_1 & b_5 & b_9 & b_{13} \\ \hline b_2 & b_6 & b_{10} & b_{14} \\ \hline b_3 & b_7 & b_{11} & b_{15} \\ \hline \end{array} \quad \leftarrow \quad \begin{array}{|c|c|c|c|}\hline k_0 & k_4 & k_8 & k_{12} \\ \hline k_1 & k_5 & k_9 & k_{13} \\ \hline k_2 & k_6 & k_{10} & k_{14} \\ \hline k_3 & k_7 & k_{11} & k_{15} \\ \hline \end{array} \quad \oplus \quad \begin{array}{|c|c|c|c|}\hline b_0 & b_4 & b_8 & b_{12} \\ \hline b_1 & b_5 & b_9 & b_{13} \\ \hline b_2 & b_6 & b_{10} & b_{14} \\ \hline b_3 & b_7 & b_{11} & b_{15} \\ \hline \end{array}$$

The generic entry b_i is updated to $b_i \oplus k_i$

Advanced Encryption Standard (AES)

Each round of the SPN modifies the state by performing the following operations:

2) SubBytes: Each byte b_i is replaced by another byte $S(b_i)$ where S is a **single, fixed** permutation on $\{0,1\}^8$

$$\begin{array}{|c|c|c|c|}\hline b_0 & b_4 & b_8 & b_{12} \\ \hline b_1 & b_5 & b_9 & b_{13} \\ \hline b_2 & b_6 & b_{10} & b_{14} \\ \hline b_3 & b_7 & b_{11} & b_{15} \\ \hline \end{array} \quad \longleftarrow \quad \begin{array}{|c|c|c|c|}\hline S(b_0) & S(b_4) & S(b_8) & S(b_{12}) \\ \hline S(b_1) & S(b_5) & S(b_9) & S(b_{13}) \\ \hline S(b_2) & S(b_6) & S(b_{10}) & S(b_{14}) \\ \hline S(b_3) & S(b_7) & S(b_{11}) & S(b_{15}) \\ \hline \end{array}$$

Advanced Encryption Standard (AES)

Each round of the SPN modifies the state by performing the following operations:

3) ShiftRows: The bytes in each row in the matrix undergo a cyclic left shift. The i-th row, counting from 0, is shifted by i places (row 0 is unaffected).

b_0	b_4	b_8	b_{12}
b_1	b_5	b_9	b_{13}
b_2	b_6	b_{10}	b_{14}
b_3	b_7	b_{11}	b_{15}

b_0	b_4	b_8	b_{12}
b_5	b_9	b_{13}	b_1
b_{10}	b_{14}	b_2	b_6
b_{15}	b_3	b_7	b_{11}

Advanced Encryption Standard (AES)

Each round of the SPN modifies the state by performing the following operations:

4) MixColumns: An invertible linear transformation is applied to each column. This transformation has the property that if two inputs differ in $b > 0$ bytes, then the resulting outputs differ in at least $5 - b$ bytes.

b_0	b_4	b_8	b_{12}
b_1	b_5	b_9	b_{13}
b_2	b_6	b_{10}	b_{14}
b_3	b_7	b_{11}	b_{15}

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} \leftarrow \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \cdot \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

Multiplication and additions are done over the finite field GF(2^8)

Advanced Encryption Standard (AES)

In the final round, the **MixColumns** step is replaced with **AddRoundKey**

Advanced Encryption Standard (AES)

In the final round, the **MixColumns** step is replaced with **AddRoundKey**

This is because the **SubBytes**, **MixRows**, and **MixColumns** do not depend on the key

Without the final **AddRoundKey** step, an adversary could simply invert the last three steps of the last round

Advanced Encryption Standard (AES)