Cryptography III: Symmetric Ciphers

Computer Security Lecture 7

Mike Just[1]

School of Informatics
University of Edinburgh

28th January 2010

[1] Based on original lecture notes by David Aspinall

Outline

Stream ciphers and block ciphers

Symmetric-key encryption schemes are often characterised as *stream ciphers* or *block ciphers*, but the distinction can be fuzzy.

- A **stream cipher** is an encryption scheme which treats the plaintext symbol-by-symbol (e.g., by bit or byte);
- security in a stream cipher lies in a changing **keystream** rather than the encryption function, which may be simple.
- A **block cipher** is an encryption scheme which breaks up the plaintext message into *blocks* of a fixed length (e.g., 128 bits), and encrypts one block at a time;
- the block encryption function is a complex function parameterised on a fixed size **key**.

Outline

Stream ciphers

Typically, $\mathcal{M} = \mathcal{C} = \mathcal{A}$ and a stream of symbols

$$m_1 \ m_2 \ m_3 \ \cdots$$

is encrypted using a keystream

$$e_1 \ e_2 \ e_3 \ \cdots$$

to generate

$$E_{e_1}(m_1) \ E_{e_2}(m_2) \ E_{e_3}(m_3) \ \cdots$$

Stream ciphers may be

- **synchronous** (keystream generated independently of the plaintext and ciphertext), or
- **self-synchronizing** (the keystream is generated as a function of the key and a fixed amount of previous ciphertext).

Vernam cipher and one-time pad

- The **Vernam cipher** is a stream cipher defined on the alphabet $\mathcal{A} = \{0, 1\}$, with a key stream also of binary digits. Each symbol m_i in the message is encoded using the corresponding symbol k_i of the key stream, using exclusive-or:

  $$c_i = m_i \oplus k_i.$$

  Because $(a \oplus b) \oplus b = a$, the decryption operation is identical:

  $$m_i = c_i \oplus k_i.$$

Vernam cipher and one-time pad

- The **Vernam cipher** is a stream cipher defined on the alphabet $\mathcal{A} = \{0, 1\}$, with a key stream also of binary digits. Each symbol m_i in the message is encoded using the corresponding symbol k_i of the key stream, using exclusive-or:

  $$c_i = m_i \oplus k_i.$$

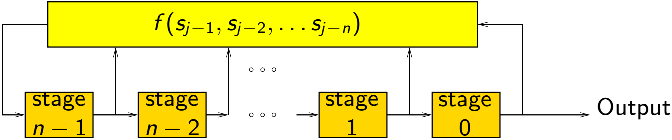
  Because $(a \oplus b) \oplus b = a$, the decryption operation is identical:

  $$m_i = c_i \oplus k_i.$$

If the key string is randomly chosen, and never reused, then this cipher is called a **one-time pad**. Claude Shannon proved that this cipher is unconditionally secure. Unfortunately, to guarantee this, it requires a true random source for key bits (hard to come by), and a key stream as long as the message. This makes it impractical for most applications. It used to be used for high security communications between Washington and Moscow.

Feedback Shift Registers

- More practical than the one-time pad would be to use a pseudorandom keystream, which is *seeded* with a much shorter key. **Feedback shift registers** (FSRs) are the basic component of many keystream generators, used to produce pseudorandom bit streams.
- An FSR of length n consists of n 1-bit register stages connected together, whose contents $\vec{s}$ are inputs to a boolean function f. At each tick, the contents are shifted right, and f calculates the feedback digit.



- If the initial state is $[s_{n-1}, \ldots, s_0]$, then the output sequence $s_0, s_1, \ldots$ is determined by the equation: $s_j = f(s_{j-1}, s_{j-2}, \ldots s_{j-n})$ for $j \geq n$.

Linear Feedback Shift Registers

- In a n-length LFSR, the feedback function f is set by a n-degree *connection polynomial* C with binary coefficients c_i

$$C(X) = 1 + c_1 X + c_2 X^2 \ldots + c_n X^n$$

this determines the feedback function, as:

$$s_j = (c_1 s_{j-1} + c_2 s_{j-2} + \cdots + c_n s_{j-n}) \bmod 2 \qquad \text{for } j \geq n.$$

Linear Feedback Shift Registers

- In a n-length LFSR, the feedback function f is set by a n-degree *connection polynomial* C with binary coefficients c_i

$$C(X) = 1 + c_1 X + c_2 X^2 \ldots + c_n X^n$$

  this determines the feedback function, as:

$$s_j = (c_1 s_{j-1} + c_2 s_{j-2} + \cdots + c_n s_{j-n}) \bmod 2 \qquad \text{for } j \geq n.$$

- LFSRs have an elegant algebraic theory and can be constructed to produce sequences with good properties: a large *period* (the maximum, $2^n - 1$), good *statistical randomness* properties, and a large *linear complexity* (a statistical "effectiveness" measure).

Linear Feedback Shift Registers

- In a n-length LFSR, the feedback function f is set by a n-degree *connection polynomial* C with binary coefficients c_i
$$C(X) = 1 + c_1X + c_2X^2 \ldots + c_nX^n$$
this determines the feedback function, as:
$$s_j = (c_1 s_{j-1} + c_2 s_{j-2} + \cdots + c_n s_{j-n}) \bmod 2 \qquad \text{for } j \geq n.$$

- LFSRs have an elegant algebraic theory and can be constructed to produce sequences with good properties: a large *period* (the maximum, $2^n - 1$), good *statistical randomness* properties, and a large *linear complexity* (a statistical "effectiveness" measure).
- However, LFSRs are **insecure**. To break, find the length n, and then use a known-plaintext attack of length $2n$.

Linear Feedback Shift Registers

- In a n-length LFSR, the feedback function f is set by a n-degree *connection polynomial* C with binary coefficients c_i
$$C(X) = 1 + c_1 X + c_2 X^2 \ldots + c_n X^n$$
this determines the feedback function, as:
$$s_j = (c_1 s_{j-1} + c_2 s_{j-2} + \cdots + c_n s_{j-n}) \bmod 2 \qquad \text{for } j \geq n.$$
- LFSRs have an elegant algebraic theory and can be constructed to produce sequences with good properties: a large *period* (the maximum, $2^n - 1$), good *statistical randomness* properties, and a large *linear complexity* (a statistical "effectiveness" measure).
- However, LFSRs are **insecure**. To break, find the length n, and then use a known-plaintext attack of length $2n$.
- In practice, some controlled **non-linearity** is added by either non-linear filtering or composition of LFSRs, or LFSR-controlled clocking.

Outline

Simple substitution ciphers

A *simple substitution cipher* is a block cipher for arbitrary block length t. It swaps each letter for another letter, using a permutation of the alphabet.

- Let $\mathcal{A}$ be an alphabet, $\mathcal{M}$ be the set of strings over $\mathcal{A}$ of length t, and $\mathcal{K}$ be the set of all permutations on $\mathcal{A}$.

Simple substitution ciphers

A *simple substitution cipher* is a block cipher for arbitrary block length t. It swaps each letter for another letter, using a permutation of the alphabet.

- Let $\mathcal{A}$ be an alphabet, $\mathcal{M}$ be the set of strings over $\mathcal{A}$ of length t, and $\mathcal{K}$ be the set of all permutations on $\mathcal{A}$.
- For each $e \in \mathcal{K}$ define E_e by applying the permutation e to each letter in the plaintext block:

$$E_e(m) = e(m_1)e(m_2)\cdots e(m_t) = c$$

  where $m \in \mathcal{M}$ and $m = m_1 m_2 \cdots m_t$.

Simple substitution ciphers

A *simple substitution cipher* is a block cipher for arbitrary block length t. It swaps each letter for another letter, using a permutation of the alphabet.

- Let $\mathcal{A}$ be an alphabet, $\mathcal{M}$ be the set of strings over $\mathcal{A}$ of length t, and $\mathcal{K}$ be the set of all permutations on $\mathcal{A}$.
- For each $e \in \mathcal{K}$ define E_e by applying the permutation e to each letter in the plaintext block:

$$E_e(m) = e(m_1)e(m_2)\cdots e(m_t) = c$$

where $m \in \mathcal{M}$ and $m = m_1 m_2 \cdots m_t$.

Simple substitution ciphers

A *simple substitution cipher* is a block cipher for arbitrary block length t. It swaps each letter for another letter, using a permutation of the alphabet.

- Let $\mathcal{A}$ be an alphabet, $\mathcal{M}$ be the set of strings over $\mathcal{A}$ of length t, and $\mathcal{K}$ be the set of all permutations on $\mathcal{A}$.
- For each $e \in \mathcal{K}$ define E_e by applying the permutation e to each letter in the plaintext block:

  $$E_e(m) = e(m_1)e(m_2)\cdots e(m_t) = c$$

  where $m \in \mathcal{M}$ and $m = m_1 m_2 \cdots m_t$.
  For each $d \in \mathcal{K}$ we define E_d in exactly the same way,

  $$D_d(c) = d(c_1)d(c_2)\cdots d(c_t).$$

- Key pairs are permutations and their inverses, so $d = e^{-1}$, and

  $$D_d(c) = e^{-1}(c_1)e^{-1}(c_2)\cdots e^{-1}(c_t) = m_1 m_2 \cdots m_t = m.$$

Simple substitution ciphers, cont'd

- The **Caesar cipher** is a simple substitution cipher which replaces $\mathtt{A} \longrightarrow \mathtt{D}$, $\mathtt{B} \longrightarrow \mathtt{E}$, $\mathtt{C} \longrightarrow \mathtt{F}$, $\ldots$, $\mathtt{X} \longrightarrow \mathtt{A}$, $\mathtt{Y} \longrightarrow \mathtt{B}$, $\mathtt{Z} \longrightarrow \mathtt{C}$.

Simple substitution ciphers, cont'd

- The **Caesar cipher** is a simple substitution cipher which replaces $\mathtt{A} \to \mathtt{D}$, $\mathtt{B} \to \mathtt{E}$, $\mathtt{C} \to \mathtt{F}$, ..., $\mathtt{X} \to \mathtt{A}$, $\mathtt{Y} \to \mathtt{B}$, $\mathtt{Z} \to \mathtt{C}$.
- The **ROT-13** transformation provided in Usenet news readers is similar, but replaces each letter character c with the character $(c + 13) \bmod 26$. This permutation is its own inverse.

Simple substitution ciphers, cont'd

- The **Caesar cipher** is a simple substitution cipher which replaces $\mathtt{A} \longrightarrow \mathtt{D}$, $\mathtt{B} \longrightarrow \mathtt{E}$, $\mathtt{C} \longrightarrow \mathtt{F}$, ..., $\mathtt{X} \longrightarrow \mathtt{A}$, $\mathtt{Y} \longrightarrow \mathtt{B}$, $\mathtt{Z} \longrightarrow \mathtt{C}$.
- The **ROT-13** transformation provided in Usenet news readers is similar, but replaces each letter character c with the character $(c + 13) \bmod 26$. This permutation is its own inverse.

Simple substitution ciphers, cont'd

- The **Caesar cipher** is a simple substitution cipher which replaces $\mathtt{A} \rightarrow \mathtt{D}$, $\mathtt{B} \rightarrow \mathtt{E}$, $\mathtt{C} \rightarrow \mathtt{F}$, ..., $\mathtt{X} \rightarrow \mathtt{A}$, $\mathtt{Y} \rightarrow \mathtt{B}$, $\mathtt{Z} \rightarrow \mathtt{C}$.
- The **ROT-13** transformation provided in Usenet news readers is similar, but replaces each letter character c with the character $(c + 13) \bmod 26$. This permutation is its own inverse.

Simple substitution ciphers are **insecure**, even when the key space is large. The reason is that the distribution of letter frequencies is preserved in the ciphertext, which allows easy cryptanalysis with a fairly small amount of ciphertext and known properties of plain text (e.g., the relative frequencies of letters in English text).

Simple substitution ciphers, cont'd

- The **Caesar cipher** is a simple substitution cipher which replaces $\mathtt{A} \to \mathtt{D}$, $\mathtt{B} \to \mathtt{E}$, $\mathtt{C} \to \mathtt{F}$, ..., $\mathtt{X} \to \mathtt{A}$, $\mathtt{Y} \to \mathtt{B}$, $\mathtt{Z} \to \mathtt{C}$.
- The **ROT-13** transformation provided in Usenet news readers is similar, but replaces each letter character c with the character $(c + 13) \bmod 26$. This permutation is its own inverse.

Simple substitution ciphers are **insecure**, even when the key space is large. The reason is that the distribution of letter frequencies is preserved in the ciphertext, which allows easy cryptanalysis with a fairly small amount of ciphertext and known properties of plain text (e.g., the relative frequencies of letters in English text). This emphasises what should be an obvious point: a **large keyspace does not guarantee a strong cipher**. For the alphabet A-Z, the size of the key space for this cipher is $26! \approx 2^{88}$, large enough to prevent brute force attacks by today's standards. But the cipher is easy to break.

Polyalphabetic substitution ciphers

A *polyalphabetic substitution cipher* is a block cipher with block length t. Instead of a single permutation, it uses a set of t permutations, and substitutes each letter using a permutation corresponding to its position in the block.

Polyalphabetic substitution ciphers

A *polyalphabetic substitution cipher* is a block cipher with block length t. Instead of a single permutation, it uses a set of t permutations, and substitutes each letter using a permutation corresponding to its position in the block.

- Let $\mathcal{A}$ and $\mathcal{M}$ be as before. Let $\mathcal{K}$ be the set of all t-tuples $(p_1, \ldots, p_t)$ where each p_i is a permutation on $\mathcal{A}$.

Polyalphabetic substitution ciphers

A *polyalphabetic substitution cipher* is a block cipher with block length t. Instead of a single permutation, it uses a set of t permutations, and substitutes each letter using a permutation corresponding to its position in the block.

- Let $\mathcal{A}$ and $\mathcal{M}$ be as before. Let $\mathcal{K}$ be the set of all t-tuples $(p_1, \ldots, p_t)$ where each p_i is a permutation on $\mathcal{A}$.
- For each $e = (p_1, \ldots, p_n) \in \mathcal{K}$ define E_e by applying the permutation p_i to the ith letter in the plaintext block:

$$E_e(m) = p_1(m_1)p_2(m_2)\cdots p_t(m_t) = c$$

where $m = m_1 m_2 \cdots m_t$.

Polyalphabetic substitution ciphers

A *polyalphabetic substitution cipher* is a block cipher with block length t. Instead of a single permutation, it uses a set of t permutations, and substitutes each letter using a permutation corresponding to its position in the block.

- Let $\mathcal{A}$ and $\mathcal{M}$ be as before. Let $\mathcal{K}$ be the set of all t-tuples $(p_1, \ldots, p_t)$ where each p_i is a permutation on $\mathcal{A}$.
- For each $e = (p_1, \ldots, p_n) \in \mathcal{K}$ define E_e by applying the permutation p_i to the ith letter in the plaintext block:

$$E_e(m) = p_1(m_1)p_2(m_2)\cdots p_t(m_t) = c$$

  where $m = m_1 m_2 \cdots m_t$.
- The corresponding decryption key is $d = (p_1^{-1}, \ldots, p_t^{-1})$.

Polyalphabetic substitution ciphers, cont'd

- The **Vigenère cipher** has a block-length of 3, and uses the permutations $e = (p_1, p_2, p_3)$ where p_1 rotates each letter of the alphabet three places to the right, p_2 rotates seven positions, and p_3 ten positions
  (e may be represented as the word `DHK`). For example:

Polyalphabetic substitution ciphers, cont'd

- The **Vigenère cipher** has a block-length of 3, and uses the permutations $e = (p_1, p_2, p_3)$ where p_1 rotates each letter of the alphabet three places to the right, p_2 rotates seven positions, and p_3 ten positions
  (e may be represented as the word `DHK`). For example:

$$\begin{aligned} m &= \mathit{COM\ EON\ EVE\ RYB\ ODY} \\ E_e(m) &= \mathit{FVW\ HVX\ HCO\ UFL\ RKI} \end{aligned}$$

Polyalphabetic substitution ciphers, cont'd

- The **Vigenère cipher** has a block-length of 3, and uses the permutations $e = (p_1, p_2, p_3)$ where p_1 rotates each letter of the alphabet three places to the right, p_2 rotates seven positions, and p_3 ten positions
  (e may be represented as the word `DHK`). For example:

$$\begin{array}{lcl} m & = & \textit{COM EON EVE RYB ODY} \\ E_e(m) & = & \textit{FVW HVX HCO UFL RKI} \end{array}$$

Polyalphabetic substitution ciphers, cont'd

- The **Vigenère cipher** has a block-length of 3, and uses the permutations $e = (p_1, p_2, p_3)$ where p_1 rotates each letter of the alphabet three places to the right, p_2 rotates seven positions, and p_3 ten positions
  (e may be represented as the word `DHK`). For example:

$$\begin{array}{lcl} m & = & \mathtt{COM\ EON\ EVE\ RYB\ ODY} \\ E_e(m) & = & \mathtt{FVW\ HVX\ HCO\ UFL\ RKI} \end{array}$$

Polyalphabetic substitution ciphers have the advantage over simple substitution ciphers that symbol frequencies are not preserved: a single letter may be encrypted to several different letters, in different positions.

Polyalphabetic substitution ciphers, cont'd

- The **Vigenère cipher** has a block-length of 3, and uses the permutations $e = (p_1, p_2, p_3)$ where p_1 rotates each letter of the alphabet three places to the right, p_2 rotates seven positions, and p_3 ten positions
  (e may be represented as the word `DHK`). For example:

$$
\begin{array}{lcl}
m & = & \mathit{COM\ EON\ EVE\ RYB\ ODY} \\
E_e(m) & = & \mathit{FVW\ HVX\ HCO\ UFL\ RKI}
\end{array}
$$

Polyalphabetic substitution ciphers have the advantage over simple substitution ciphers that symbol frequencies are not preserved: a single letter may be encrypted to several different letters, in different positions. However, cryptanalysis is still straightforward, by first determining the block size, and then applying frequency analysis by splitting the letters into groups which are encrypted with the same permutation. So polyalphabetic substitutions are certainly **not secure**.

Simple transposition ciphers

The *simple transposition cipher* is a block cipher with block-length t. It simply permutes the symbols in the block.

Simple transposition ciphers

The *simple transposition cipher* is a block cipher with block-length t. It simply permutes the symbols in the block.

- Let $\mathcal{K}$ be the set of all permutations on the set $\{1, 2, \ldots, t\}$. For each $e \in \mathcal{K}$, the encryption function is defined by

$$E_e(m) = (m_{e(1)}, m_{e(2)}, \ldots, m_{e(t)}).$$

The corresponding decryption key is the inverse permutation.

Simple transposition ciphers

The *simple transposition cipher* is a block cipher with block-length t. It simply permutes the symbols in the block.

- Let $\mathcal{K}$ be the set of all permutations on the set $\{1, 2, \ldots, t\}$. For each $e \in \mathcal{K}$, the encryption function is defined by

$$E_e(m) = (m_{e(1)}, m_{e(2)}, \ldots, m_{e(t)}).$$

The corresponding decryption key is the inverse permutation.

This cipher again preserves letter frequencies, which allows easy cryptanalysis. So it is **not secure**.

Simple transposition ciphers

The *simple transposition cipher* is a block cipher with block-length t. It simply permutes the symbols in the block.

- Let $\mathcal{K}$ be the set of all permutations on the set $\{1, 2, \ldots, t\}$. For each $e \in \mathcal{K}$, the encryption function is defined by

$$E_e(m) = (m_{e(1)}, m_{e(2)}, \ldots, m_{e(t)}).$$

  The corresponding decryption key is the inverse permutation.

This cipher again preserves letter frequencies, which allows easy cryptanalysis. So it is **not secure**. These block ciphers so far are not useful by themselves, but get interesting when combined. A good cipher should add both **confusion** by substitution transformations and **diffusion** by transpositions. Confusion obscures the relationship between the key and the ciphertext. Diffusion spreads out redundancy in the plaintext across the ciphertext. Modern block ciphers apply *rounds* consisting of substitution and transposition steps.

Product ciphers

It's easy to combine encryption functions using composition, because the composition of two bijections is again a bijection.

Product ciphers

It's easy to combine encryption functions using composition, because the composition of two bijections is again a bijection.

- A *product cipher* is defined as the composition of N encryption transformations, $E1_e, E2_e, \ldots, EN_e$, for $n \geq 0$. (We can consider a single key space wlog: each transformation may depend on a different part of the key e, or may be independent of the key.)

Product ciphers

It's easy to combine encryption functions using composition, because the composition of two bijections is again a bijection.

- A *product cipher* is defined as the composition of N encryption transformations, $E1_e, E2_e, \ldots, EN_e$, for $n \geq 0$. (We can consider a single key space wlog: each transformation may depend on a different part of the key e, or may be independent of the key.)
- The overall encryption function composes the parts:

$$E_e = E1_e;\ E2_e;\ \ldots\ ;\ EN_e$$

  where ; denotes function composition in the diagramatic order.

Product ciphers

It's easy to combine encryption functions using composition, because the composition of two bijections is again a bijection.

- A *product cipher* is defined as the composition of N encryption transformations, $E1_e, E2_e, \ldots, EN_e$, for $n \geq 0$. (We can consider a single key space wlog: each transformation may depend on a different part of the key e, or may be independent of the key.)
- The overall encryption function composes the parts:

  $$E_e = E1_e;\ E2_e;\ \ldots\ ;\ EN_e$$

  where ; denotes function composition in the diagramatic order.
- The overal decryption function composes the decryptions:

  $$D_d = DN_d;\ \ldots\ D2_d;\ D1_d$$

Product ciphers

It's easy to combine encryption functions using composition, because the composition of two bijections is again a bijection.

- A *product cipher* is defined as the composition of N encryption transformations, $E1_e, E2_e, \ldots, EN_e$, for $n \geq 0$. (We can consider a single key space wlog: each transformation may depend on a different part of the key e, or may be independent of the key.)
- The overall encryption function composes the parts:

$$E_e = E1_e;\ E2_e;\ \ldots\ ;\ EN_e$$

  where ; denotes function composition in the diagramatic order.
- The overal decryption function composes the decryptions:

$$D_d = DN_d;\ \ldots\ D2_d;\ D1_d$$

- Involutions (functions that are their own inverse) are particularly useful in constructing product ciphers. The favourite is XOR: $f(x) = x \oplus c$.

Constructing block ciphers

- Classical block ciphers are constructed from circuits of **S-boxes** and **P-boxes**.

Constructing block ciphers

- Classical block ciphers are constructed from circuits of **S-boxes** and **P-boxes**.
- The **Feistel principle** gives a way of constructing a cipher so that the same circuit is used for both encryption and decryption. A *round* in a Feistel cipher treats the input block in two halfs, L_i and R_i. It uses the right-hand half to modify the left, and then swaps:

  $$L_{i+1} = R_i \qquad R_{i+1} = L_i \oplus f(K_i, R_i).$$

  The inverse operation is:

  $$R_i = L_{i+1} \qquad L_i = R_{i+1} \oplus f(K_i, L_{i+1}).$$

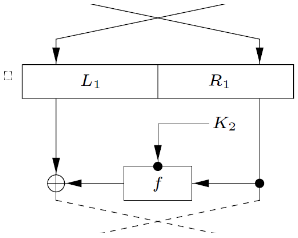
Constructing block ciphers

- Classical block ciphers are constructed from circuits of **S-boxes** and **P-boxes**.
- The **Feistel principle** gives a way of constructing a cipher so that the same circuit is used for both encryption and decryption. A *round* in a Feistel cipher treats the input block in two halfs, L_i and R_i. It uses the right-hand half to modify the left, and then swaps:

$$L_{i+1} = R_i \qquad R_{i+1} = L_i \oplus f(K_i, R_i).$$

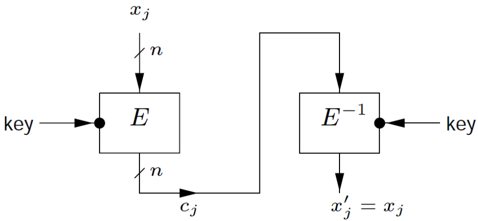
The inverse operation is:

$$R_i = L_{i+1} \qquad L_i = R_{i+1} \oplus f(K_i, L_{i+1}).$$



Modes for block ciphers: ECB

- Block ciphers can be used in various **modes**. **Important reading exercise**: compare the *security*, *efficiency*, inbuilt *data integrity*, and *error recovery* of these different modes.
- **ECB**: *electronic codebook mode*. Each block of plaintext x_j is enciphered independently.



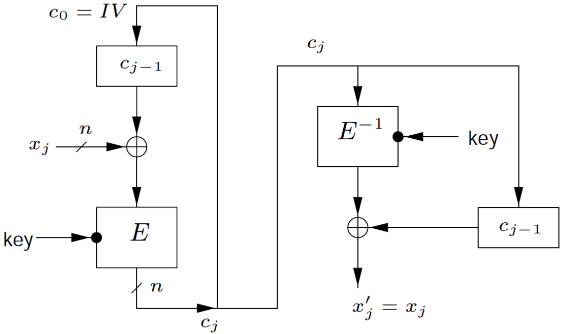
(i) encipherment (ii) decipherment

- This is the simplest mode, but it has obvious failings.

Modes for block ciphers: CBC

- **CBC**: *cipherblock chaining mode*. Each plaintext block x_j is XORed with the previous ciphertext c_{j-1} block before encryption. An *initialization vector* (IV) (optionally secret, fresh for each message) is used for c_0.

$$c_j = E_k(x_j \oplus c_{j-1}) \qquad x_j = c_{j-1} \oplus E_k^{-1}(c_j)$$

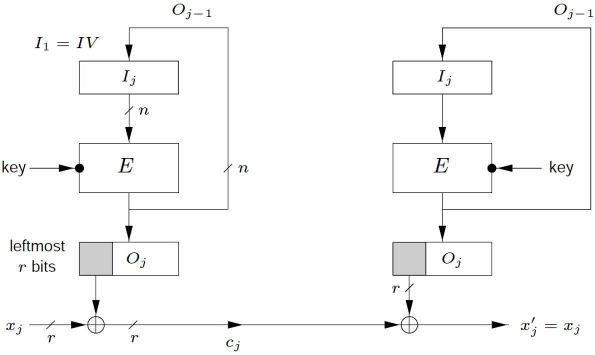


(i) encipherment (ii) decipherment

Modes for block ciphers: OFB

- **OFB**: *output-feedback mode*. Block cipher encryption function used as synchronous stream cipher (*internal feedback*).

$$c_j = x_j \oplus s_j; \; s_j = E_k(s_{j-1}) \qquad x_j = c_j \oplus s_j; \; s_j = E_k(s_{j-1})$$

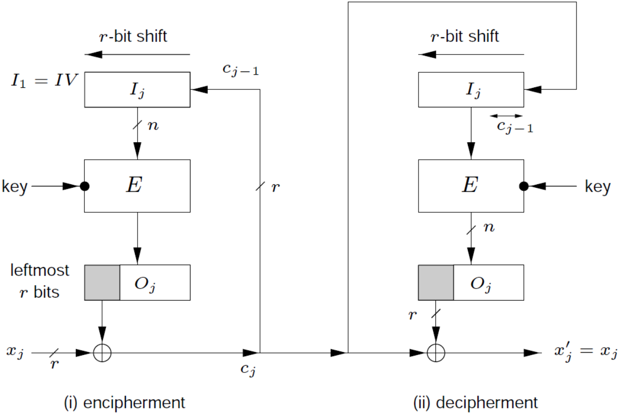


(i) encipherment (ii) decipherment

Modes for block ciphers: CFB

- **CFB** *cipher-feedback mode*. Encryption function of block cipher used as self-synchronizing stream cipher for symbols of size up to block size.

$$c_j = x_j \oplus E_k(c_{j-1}) \qquad x_j = c_j \oplus E_k(c_{j-1})$$



Outline

DES

- **DES** is a block cipher based on Feistel's principle. Block-size is 64 bits, key-size 56 bits (+8 parity bits). Invented by IBM in 1970s, tweaked by NSA. Still widely used, esp. in financial sector. Much analysed.

DES

- **DES** is a block cipher based on Feistel's principle. Block-size is 64 bits, key-size 56 bits (+8 parity bits). Invented by IBM in 1970s, tweaked by NSA. Still widely used, esp. in financial sector. Much analysed.
- Main threat isn't cryptanalytic, but (slightly optimised) **exhaustive search** in small key-space. Remedied by **3DES** (triple DES), 3 keys:

$$C = E_{k_3}(D_{k_2}(E_{k_1}(P))) \qquad P = D_{k_1}(E_{k_2}(D_{k_3}(C))).$$

DES

- **DES** is a block cipher based on Feistel's principle. Block-size is 64 bits, key-size 56 bits (+8 parity bits). Invented by IBM in 1970s, tweaked by NSA. Still widely used, esp. in financial sector. Much analysed.
- Main threat isn't cryptanalytic, but (slightly optimised) **exhaustive search** in small key-space. Remedied by **3DES** (triple DES), 3 keys:

  $$C = E_{k_3}(D_{k_2}(E_{k_1}(P))) \qquad P = D_{k_1}(E_{k_2}(D_{k_3}(C))).$$

  Security of 3DES is not obvious: repeated encryption may not gain security (one-step DES is not closed, so it in fact does), and new attacks may be possible (**meet-in-the-middle attack**). With 3 independently chosen keys, security is roughly the same as expected with 2 keys.

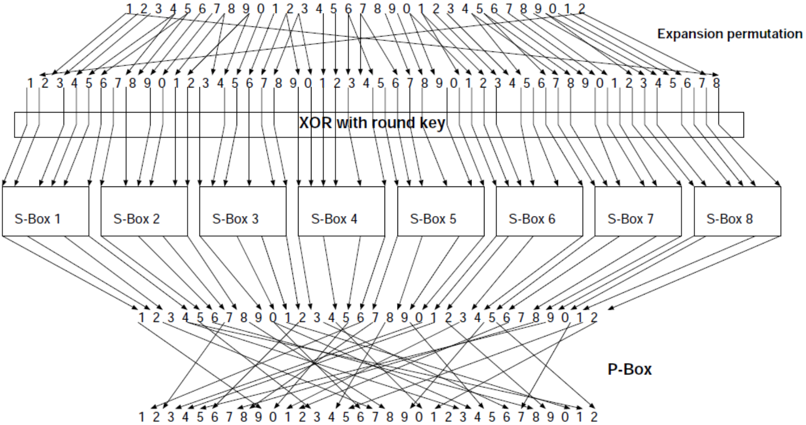
DES

- **DES** is a block cipher based on Feistel's principle. Block-size is 64 bits, key-size 56 bits (+8 parity bits). Invented by IBM in 1970s, tweaked by NSA. Still widely used, esp. in financial sector. Much analysed.
- Main threat isn't cryptanalytic, but (slightly optimised) **exhaustive search** in small key-space. Remedied by **3DES** (triple DES), 3 keys:

  $$C = E_{k_3}(D_{k_2}(E_{k_1}(P))) \qquad P = D_{k_1}(E_{k_2}(D_{k_3}(C))).$$

  Security of 3DES is not obvious: repeated encryption may not gain security (one-step DES is not closed, so it in fact does), and new attacks may be possible (**meet-in-the-middle attack**). With 3 independently chosen keys, security is roughly the same as expected with 2 keys.
- Several other DES variants, including **DESX**, using *whitenening* keys k_1, k_2 as $C = E_k(P \oplus k_1) \oplus k_2$. (Used in Win2K encrypting FS).

Overview of DES internals [FIPS 46-3]



The Advanced Encryption Standard

- In October 2000, the US NIST selected **Rijndael** as the new AES, to replace the aging DES. Rijndael was designed by two Belgian cryptographers, Vincent Rijmen and Joan Daemen. The algorithm was selected as a result of a 3 year worldwide review process. No proof of security, but a high level of confidence amongst cryptographers.

The Advanced Encryption Standard

- In October 2000, the US NIST selected **Rijndael** as the new AES, to replace the aging DES. Rijndael was designed by two Belgian cryptographers, Vincent Rijmen and Joan Daemen. The algorithm was selected as a result of a 3 year worldwide review process. No proof of security, but a high level of confidence amongst cryptographers.
- Rijndael satisfied a number of requisite criteria for the AES:
  - **Security**: mathematical, cryptanalytic resistance; randomness;
  - **Efficiency**: time/space, hardware and software;
  - **Flexibility**: block sizes 128 bits, key sizes 128/192/256 bits.
  - **Intellectual property**: unclassified, published, royalty-free.

The Advanced Encryption Standard

- In October 2000, the US NIST selected **Rijndael** as the new AES, to replace the aging DES. Rijndael was designed by two Belgian cryptographers, Vincent Rijmen and Joan Daemen. The algorithm was selected as a result of a 3 year worldwide review process. No proof of security, but a high level of confidence amongst cryptographers.
- Rijndael satisfied a number of requisite criteria for the AES:
  - **Security**: mathematical, cryptanalytic resistance; randomness;
  - **Efficiency**: time/space, hardware and software;
  - **Flexibility**: block sizes 128 bits, key sizes 128/192/256 bits.
  - **Intellectual property**: unclassified, published, royalty-free.

  The US Federal Information Processing standard FIPS 197 for AES was published in November 2001.

The Advanced Encryption Standard

- In October 2000, the US NIST selected **Rijndael** as the new AES, to replace the aging DES. Rijndael was designed by two Belgian cryptographers, Vincent Rijmen and Joan Daemen. The algorithm was selected as a result of a 3 year worldwide review process. No proof of security, but a high level of confidence amongst cryptographers.
- Rijndael satisfied a number of requisite criteria for the AES:
  - **Security**: mathematical, cryptanalytic resistance; randomness;
  - **Efficiency**: time/space, hardware and software;
  - **Flexibility**: block sizes 128 bits, key sizes 128/192/256 bits.
  - **Intellectual property**: unclassified, published, royalty-free.

  The US Federal Information Processing standard FIPS 197 for AES was published in November 2001.
- Rijndael is built as a network of linear transformations and substitutions, with 10, 12 or 14 rounds, depending on key size.

Outline

Stream ciphers

Block ciphers

DES and Rijndael

Summary

Current symmetric crypto algorithms

Stream ciphers

Current symmetric crypto algorithms

Stream ciphers

- **A5**, encrypts GSM digital cellular traffic. Originally secret, but leaked and reverse-engineered. Based on three LFSRs. Very feasible attack.

Current symmetric crypto algorithms

Stream ciphers

- **A5**, encrypts GSM digital cellular traffic. Originally secret, but leaked and reverse-engineered. Based on three LFSRs. Very feasible attack.
- **PKZIP** has a byte-wide stream cipher. Easily broken with a small amount of plain text.

Current symmetric crypto algorithms

Stream ciphers

- **A5**, encrypts GSM digital cellular traffic. Originally secret, but leaked and reverse-engineered. Based on three LFSRs. Very feasible attack.
- **PKZIP** has a byte-wide stream cipher. Easily broken with a small amount of plain text.
- **RC4/ARCFOUR**. RSADSI trade secret; code posted anonymously in 1994. Variable key-size, byte-wide, OFB with 8×8 S-box. Very fast & simple, widely licensed (Lotus Notes, Oracle SQL), less widely studied.

Current symmetric crypto algorithms

Stream ciphers

- **A5**, encrypts GSM digital cellular traffic. Originally secret, but leaked and reverse-engineered. Based on three LFSRs. Very feasible attack.
- **PKZIP** has a byte-wide stream cipher. Easily broken with a small amount of plain text.
- **RC4/ARCFOUR**. RSADSI trade secret; code posted anonymously in 1994. Variable key-size, byte-wide, OFB with 8×8 S-box. Very fast & simple, widely licensed (Lotus Notes, Oracle SQL), less widely studied.

Current symmetric crypto algorithms

Stream ciphers

- **A5**, encrypts GSM digital cellular traffic. Originally secret, but leaked and reverse-engineered. Based on three LFSRs. Very feasible attack.
- **PKZIP** has a byte-wide stream cipher. Easily broken with a small amount of plain text.
- **RC4/ARCFOUR**. RSADSI trade secret; code posted anonymously in 1994. Variable key-size, byte-wide, OFB with 8×8 S-box. Very fast & simple, widely licensed (Lotus Notes, Oracle SQL), less widely studied.

Block ciphers

- **DES**, **3DES**, **Rijndael** outlined previously.
- **IDEA**, 64-bit blocks, 128-bit key. Efficient: uses XOR, addition and multiplication operations. Patented for commercial use. Used in PGP.

Current symmetric crypto algorithms

Stream ciphers

- **A5**, encrypts GSM digital cellular traffic. Originally secret, but leaked and reverse-engineered. Based on three LFSRs. Very feasible attack.
- **PKZIP** has a byte-wide stream cipher. Easily broken with a small amount of plain text.
- **RC4/ARCFOUR**. RSADSI trade secret; code posted anonymously in 1994. Variable key-size, byte-wide, OFB with 8×8 S-box. Very fast & simple, widely licensed (Lotus Notes, Oracle SQL), less widely studied.

Block ciphers

- **DES**, **3DES**, **Rijndael** outlined previously.
- **IDEA**, 64-bit blocks, 128-bit key. Efficient: uses XOR, addition and multiplication operations. Patented for commercial use. Used in PGP.
- **Skipjack**. NSA designed, once classified (key escrow and LEAF issue) and patented under a secrecy order; now public domain. Block size 64 bits, 80-bit key. Used in tamperproof **Clipper** and **Capstone** chips.

References

The DES diagram is from Smart, Chapter 8 and the block cipher diagrams are from Figure 7.1 in the HAC.

- Alfred J. Menezes, Paul C. Van Oorschot, and Scott A. Vanstone, editors. *Handbook of Applied Cryptography*. CRC Press Series on Discrete Mathematics and Its Applications. CRC Press, 1997. Online version at http://www.cacr.math.uwaterloo.ca/hac.
- Bruce Schneier. *Applied Cryptography*. John Wiley & Sons, second edition, 1996.
- Nigel Smart. *Cryptography: An Introduction*. 3rd edition, 2008, at http://www.cs.bris.ac.uk/~nigel/Crypto_Book/.

Recommended Reading

Chapters 7 and 8 of Smart.