Introduction to Applied Cryptography, Part 1*

Prof. Susanne Wetzel
Stevens Institute of Technolgy
Department of Computer Science
Hoboken, New Jersey, USA

Dr. Torsten Schütze
Robert Bosch GmbH
Corporate Sector Research
and Advance Engineering
Software (CR/AEA)

September 21–23, 2009

*The material has been presented at previous Summer Schools for Studienstiftung des Deutschen Volkes while the second author was at Siemens AG, CT IC 3.

Resources

- Douglas R. Stinson: *Cryptography Theory and Practice* (Third Edition), 2005, Chapman & Hall, CRC, ISBN 1-58488-508-4
  `http://www.cacr.math.uwaterloo.ca/~dstinson/CTAP3/CTAP3.html`
  Errata: `http://www.cacr.math.uwaterloo.ca/~dstinson/CTAP3/errata3.html`

- Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone: *Handbook of Applied Cryptography*, 1996,
  CRC Press, ISBN 0-8493-8523-8
  `http://www.cacr.math.uwaterloo.ca/hac/`

- Ross Anderson: *Security Engineering: A Guide to Building Dependable Distributed Systems* (Second Edition), 2008,
  Wiley Publishing, Inc., ISBN 0-4700-6852-3
  `http://www.cl.cam.ac.uk/~rja14/book.html`

Outline of Part 1

1. **Classical Cryptography**
2. **Probability Theory and Perfect Secrecy**
3. **Block Ciphers**
4. **Cryptographic Hash Functions**
5. **Message Authentication Codes**

Part 1: Outline — Section 1: Classical Cryptography

- 1. Introduction to cryptography, basic definitions and principles
- 2. Math refresher: Modular arithmetic, groups, rings, fields
- 3. Introduction and cryptanalysis of simple ciphers
  - Caesar cipher
  - Shift cipher
  - Substitution cipher
  - Permutation cipher
  - Vigenère cipher
  - Stream ciphers
- 4. Attack models
- 5. Notions of security

1. What is Cryptography?

Definition 1.1.

Cryptography is the study of mathematical techniques relating to aspects of information security such as confidentiality, data integrity, identity authentication, and data origin authentication.

Definition 1.2.

Cryptanalysis is the study of mathematical techniques attempting to defeat cryptographic techniques.

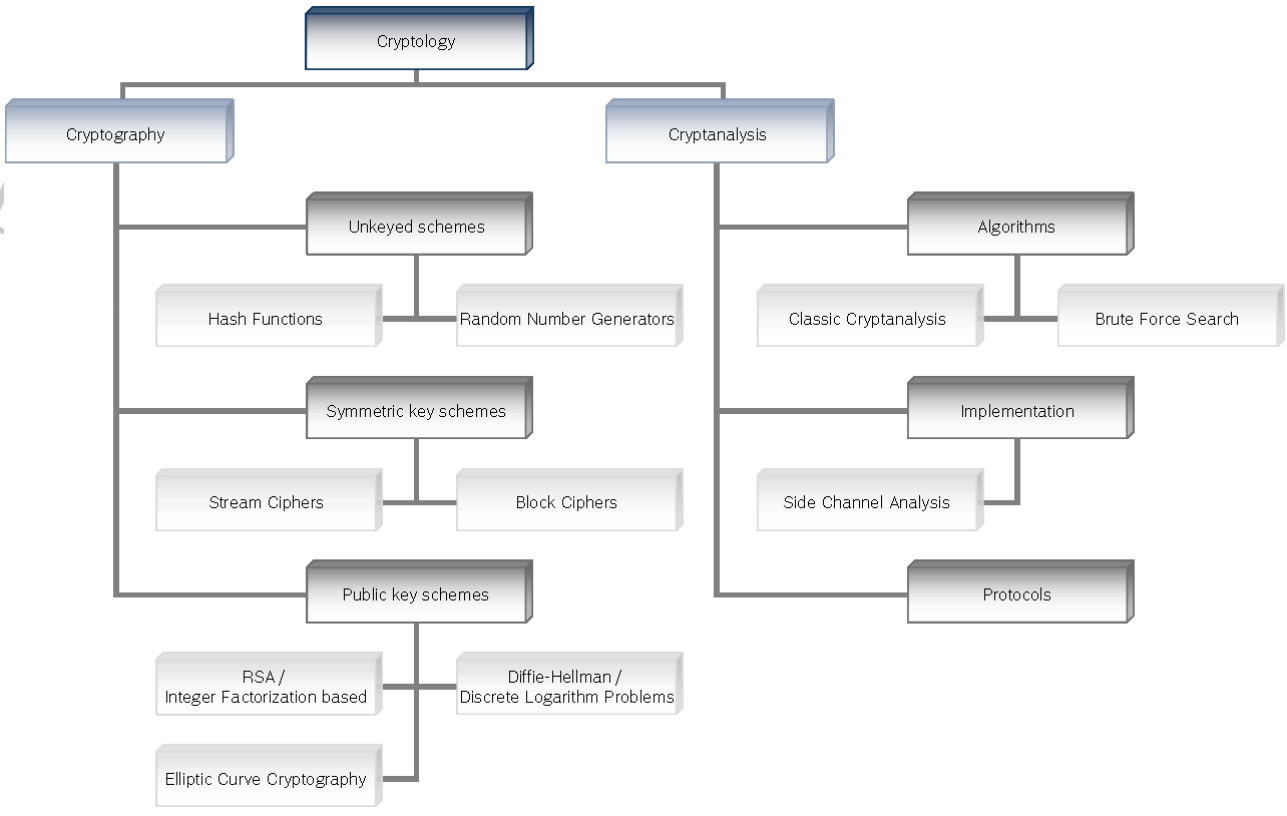
Definition 1.3.

Cryptology = cryptography + cryptanalysis

Remark 1.4.

In order to do good cryptography, it is important to understand cryptanalysis, i.e., to understand potential weaknesses of particular cryptosystems.

Taxonomy of Cryptology



Why do we need Security/Cybersecurity or Information Assurance?

- Threats and attacks:
  - Manipulation of information (stored or in transit)
  - Unauthorized use of services
  - Disclosing identity
  - Eavesdropping
  - Denial of Service
  - . . .
- Potential attackers (maybe with limited resources, e.g., computing power, money, technical sophistication):
  - Active attacker
  - Passive eavesdropper

  Examples: Hackers, unauthorized users, organized crime, big brother, . . .

Security Objectives (1)

- **Authentication**: Assurance of the identity
- **Access Control/Authorisation**: Unauthorised users are kept out
- **Confidentiality** (privacy, better reserved for anonymity issues): Assurance to keep information from all but those authorized to have it
- **Availability**: Legitimate users have access when they need it
- **Integrity**: Assurance to prevent unauthorized alterations of data
- **Non-repudiation**: Assurance to prevent denial of previous commitments of actions (Originator of communications can't deny it later)

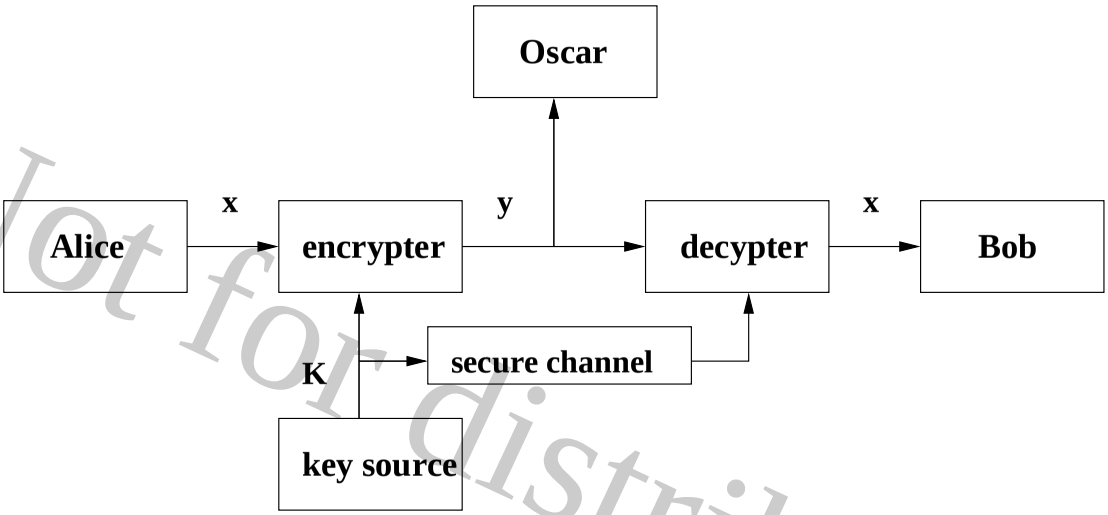
Security Objectives (2)

- **Anonymity/Unlinkability**: Assurance of non-disclosure of identities, non-traceability
- **Time Integrity**: Ensuring that the indicated time of creation of a piece of information is correct
- **Accountability**: Ensuring that entities can be accounted for their actions
- **Auditability**: Ensuring that previous system states can later be reconstructed
- **Remark 1.5.**
  Encryption techniques constitute only a minor part of a Cryptographer's life, mainly he is concerned with authentication, integrity protection, etc.
- **Remark 1.6.**
  Above security objectives cannot be achieved with cryptographic solutions alone.

Cryptographic Primitives

encryption techniques	→	confidentiality
message authentication and data integrity techniques	→	integrity, authenticity
identification/entity authentication techniques	→	authenticity
digital signatures	→	authenticity, integrity, non-repudiation

Basic Setup for Cryptosystems



Sender Alice communicates with receiver Bob over an insecure channel (e.g., telephone line, or computer network) such that and opponent Oscar (often called Eve) cannot determine what was said.

Some Definitions (1)

Definition 1.7.

A symmetric cryptosystem is a five-tuple $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$, where the following conditions are satisfied:

1. $\mathcal{P}$ is a finite set of possible plaintexts
2. $\mathcal{C}$ is a finite set of possible ciphertexts
3. The keyspace $\mathcal{K}$ is a finite set of possible keys
4. For each $K \in \mathcal{K}$, there is an encryption rule $e_K \in \mathcal{E}$ and a corresponding decryption rule $d_K \in \mathcal{D}$ where $e_K : \mathcal{P} \to \mathcal{C}$ is injective[1] and $d_K : \mathcal{C} \to \mathcal{P}$ such that $d_K(e_K(x)) = x$ for every plaintext element $x \in \mathcal{P}$.

[1] one-to-one, $e_K(P_1) = e_K(P_2) \Longrightarrow P_1 = P_2$

Some Definitions (2)

- **Remark 1.8.**
  If $\mathcal{P} = \mathcal{C}$ then encryption is a permutation.
- Usage of the cryptosystem $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$:
  - Alice and Bob first agree on a random key $k \in \mathcal{K}$ using a secure channel.
  - In order to send $x = x_1 x_2 \ldots x_n$, $(n \geq 1, x_i \in \mathcal{P})$, Alice computes $y_i = e_K(x_i)$ $(1 \leq i \leq n)$.
  - $y = y_1 y_2 \ldots y_n$ is sent to Bob.

  (Block cipher)

Some Principles

- **Arbitrary Adversary Principle**:
  - No assumption on adversarial strategy
  - Instead, focus on adversaries determined by, e.g., access restrictions or computational restrictions.
  - Show no adversary in this class can break cryptosystem.
- **Kerckhoffs' principle**: (A. Kerckhoffs, 1835-1903) consider the security of cryptosystems under the assumption that the algorithms are known to the adversary (and only the keys are secret).
- **Key length principle**: in order for a cryptosystem to be secure, it is necessary that the keyspace is large enough that exhaustive key search is infeasible.

Remark 1.9.
Kerckhoffs' principle does not state that the algorithm has to be published, it just states that it's security must not rely on the secrecy of the algorithmic details.

2. Modular Arithmetic (1)

Definition 1.10.

Let a, b be integers, i.e., $a, b \in \mathbb{Z}$ and m be a positive integer, i.e., $m \in \mathbb{Z}^+$. We write $a \equiv b \pmod{m}$ if m divides $b - a$. The congruence $a \equiv b \pmod{m}$ is read "a is congruent to b modulo m". The integer m is the modulus. We define $a \bmod m$ to be the unique value $b \in \mathbb{Z}_m$ such that $a \equiv b \pmod{m}$. If we replace a by $a \bmod m$, we say that a is reduced modulo m.

Definition 1.11. Modular Arithmetic

$\mathbb{Z}_m$ is defined to be the set $\{0, \ldots, m-1\}$, with two operators $+$ and $\times$. Addition and multiplication in $\mathbb{Z}_m$ work exactly like integer addition and multiplication, except that the results are reduced modulo m.

Example 1.12.

Compute 4×3 in $\mathbb{Z}_5$: $4 \times 3 \bmod 5 = 12 \bmod 5 = 2$.

Modular Arithmetic (2)

Theorem 1.13.

The modular arithmetic satisfies the following properties:

1. addition is closed, i.e., for any $a, b \in \mathbb{Z}_m$, $a + b \in \mathbb{Z}_m$.
2. addition is commutative, i.e., for any $a, b \in \mathbb{Z}_m$, $a + b = b + a$.
3. addition is associative, i.e., for any $a, b, c \in \mathbb{Z}_m$, $(a + b) + c = a + (b + c)$.
4. 0 is an additive identity, i.e., for any $a \in \mathbb{Z}_m$, $a + 0 = 0 + a = a$.
5. the additive inverse of any $a \in \mathbb{Z}_m$ is $m - a$, i.e., $a + (m - a) = (m - a) + a = 0$.
6. multiplication is closed, i.e., for any $a, b \in \mathbb{Z}_m$, $ab \in \mathbb{Z}_m$.
7. multiplication is commutative, i.e., for any $a, b \in \mathbb{Z}_m$, $ab = ba$.
8. multiplication is associative, i.e., for any $a, b, c \in \mathbb{Z}_m$, $(ab)c = a(bc)$.
9. 1 is a multiplicative identity, i.e., for any $a \in \mathbb{Z}_m$, $a \times 1 = 1 \times a = a$.
10. the distributive property, i.e., for any $a, b, c \in \mathbb{Z}_m$, $(a + b)c = (ac) + (bc)$ and $a(b + c) = (ab) + (ac)$.

Groups (1)

Definition 1.14.

A binary operation $*$ on a set S is a mapping from $S \times S \to S$.

Definition 1.15.

A group $(G, *)$ consists of a set G with a binary operation $*$ on G satisfying the three axioms:

1. $*$ is associative, i.e., $a * (b * c) = (a * b) * c \quad \forall a, b, c \in G$.
2. there is an identity element $1 \in G$, s.t. $a * 1 = 1 * a = a \quad \forall a \in G$.
3. $\forall a \in G \; \exists a^{-1} \in G$ s.t. $a * a^{-1} = a^{-1} * a = 1$ (inverse).

A group is Abelian (or commutative), if

$$a * b = b * a \, \forall a, b \in G.$$

Groups (2)

Definition 1.16.

The order of a group G, denoted $|G|$, is the number of elements in G. G is a finite group if $|G|$ is finite.

Example 1.17.

- $(\mathbb{Z}, +)$ is a group with identity 0.
- $(\mathbb{Z}_n, +)$ is a group of order n, with identity 0.
- $(\mathbb{Z}_n, *)$ is not a group.
- $(\mathbb{Z}_n^*, *)$ is a group of order $\phi(n)$ with identity 1.[2]

Definition 1.18.

A subset U of G is called a subgroup of G if U with the group operation of G is a group.

[2] $\phi(n)$ - Euler's totient function: number of positive integers less than or equal to n that are coprime to n

Groups (3)

Definition 1.19.

The order m of an element $g \in G$ is the order of the generated subgroup, i.e., m is the smallest positive integer such that $g^m = 1$.

Definition 1.20.

A group G is called cyclic if there exists an element $g \in G$, called generator, having order $|G|$, i.e., $G = \langle g \rangle$.

Example 1.21.

- $(\mathbb{Z}_p, *)$ with p prim is a cyclic group of order $p - 1$
- Any group of prime order is cyclic.
- The base point $G = (x_G, y_G) \in E(F_p)$,

$$E(F_p) := \{(x, y) \in F_p \times F_p : y^2 = x^3 + ax + b \pmod{p}\} \cup \mathcal{O}$$

the elliptic curve E over F_p, generates a cyclic subgroup of order $n = Ord(G)$. The parameters are chosen in such a way that n is prime.

Rings

Definition 1.22.

A ring $(R, +, *)$ consists of a set R with two binary operations on R satisfying the following axioms:

1. $(R, +)$ is an Abelian group with identity 0.
2. $*$ is associative, i.e., $(a * b) * c = a * (b * c)\ \forall a, b, c \in R$
3. $\exists$ multiplicative identity $1 \neq 0$ s.t. $1 * a = a * 1 = a\ \forall a \in R$.
4. $*$ distributive over $+$, i.e., $a * (b + c) = (a * b) + (a * c)$ and $(b + c) * a = (b * a) + (c * a)\ \forall a, b, c \in R$

The ring is commutative if $a * b = b * a\ \forall a, b \in R$.

Example 1.23.

$(\mathbb{Z}, +, *)$ is a commutative ring, and so is $(\mathbb{Z}_n, +, *)$.

Fields

Definition 1.24.

An element $a \in R$ is called a unit or invertible element if there is an element $b \in R$ s.t. $a * b = 1$.

Definition 1.25.

A field is a commutative ring in which all non-zero elements have a multiplicative inverse.

Polynomials

Definition 1.26.

If F is a field, the ring of polynomials with coefficients in F, is denoted by $F[x]$.

$$f(x) := a_n x^n + \cdots + a_1 x + a_0, \quad q_i \in F, n \geq 0$$

$$\max\{m \mid a_m \neq 0\} =: \deg f \qquad \text{degree of polynomial}$$

Definition 1.27.

Let $f \in F[x]$ be a polynomial of degree at least 1. Then f is said to be irreducible over F if it cannot be written as the product of two polynomials in $F[x]$, each of positive degree.

Example 1.28.

The polynomial $f(x) := x^2 + 1$ is reducible $(f(x) = (x - i)(x + i))$ over $\mathbb{C}$, irreducible over $\mathbb{Z}$, and reducible $(f(x) = (x + 1)^2)$ over $\mathbb{Z}/2\mathbb{Z}$.

Finite Fields

In cryptography, we mainly use finite fields, i. e., fields with a finite number of elements.

Theorem 1.29. Existence and Uniqueness of Finite Fields
For every prime p and every positive integer n there exists a finite field with p^n elements. Any finite field with $q = p^n$ elements is isomorphic to the splitting field of $x^q - x$ over $GF(p)$.

The finite field is denoted by F_q, F_{p^n}, $GF(q)$ or $GF(p^n)$ with order[3] q and characteristics[4] p.

[3] number of elements

[4] smallest number of times one must add the multiplicative identity element (1) to itself to get the additive identity element (0)

Computing in Finite Fields

- Prime fields F_p:
  - Modular arithmetic $\Longrightarrow$ long term modular arithmetic unit
  - Carry propagation during addition
- Binary fields F_{2^n}:
  - Elements are binary polynomials of order n
  - Addition with no carry (XOR)
  - Multiplication = polynomial multiplication modulo irreducible polynomial
- Extension fields F_{p^n}:
  - Elements are polynomials of order n with coefficients in F_p
  - Addition = usual polynomial addition with coefficient arithmetic in F_p
  - Multiplication = polynomial multiplication modulo irreducible polynomial
  - Explicit construction of finite field F_{p^n} requires Galois theory : $F_p[X]/(f(X))$ with $f(X)$ – monic irreducible polynomial of degreee n in $F_p[X]$, $(f(X))$ ideal generated by $f(X)$, $./.$ quotient ring = set of polynomials in X with coefficients in F_p mod $((f(X))$.

Cryptographic Requirements for Finite Field Arithmetic

Remark 1.30. Types of arithmetic

- RSA: F_n (1024, 2048 bit)
- DSA: F_n (1024, 2048 bit), F_p (160–256 bit)
- ECC over F_p: F_p, F_n (160–256 bit)
- ECC over F_{2^n}: F_{2^n}, F_p (160–256 bit)
- Galois Counter Mode for AES, XTS-AES Mode for Disk Encryption: $F_{2^{128}}$

Remark 1.31. Arithmetic operations

- Modular addition, subtraction
- Modular multiplication
- Modular inverses (Fermat or Extended Euclidean Algorithm)
- Euclidean Algorithm (gcd), Extended Euclidean Algorithm (inverse)
- Modular exponentation (RSA), Scalar Multiplication (ECC)

3. Caesar Cipher (1)

- Reportedly used by Julius Caesar
- Encoded message is obtained from original message by rotating the 26 letters of the alphabet by 3 positions:

A	→	D	B	→	E	C	→	F	D	→	G	E	→	H	F	→	I
G	→	J	H	→	K	I	→	L	J	→	M	K	→	N	etc.		

Thus, the message “HELLO” is encoded as “KHOOR”.

- Security?

Caesar Cipher (2)

- Problems:
  - Easy to decipher if algorithm is known
  - Easy to determine algorithm if any pair of original/encrypted message is given, or original message of an encoded message can be recognized (frequency analysis).
- Slight improvement: introduce a key (secret known to sender and receiver only) and rotate by K positions.
  BUT: Solution suffers from same problems as above: attacker can try all 26 possibilities and find the correct one.

Shift Cipher (1)

$\mathcal{P} = \mathcal{C} = \mathcal{K} = \mathbb{Z}_{26}$. For $0 \leq K \leq 25$, define

$$e_K(x) = (x + K) \bmod 26$$

and

$$d_K(y) = (y - K) \bmod 26 \text{ (with } x, y \in \mathbb{Z}_{26}\text{)}.$$

Remark 1.32.
The shift cipher is a formalization of the keyed Caesar cipher.

Remark 1.33.
In order to use the cipher to encrypt ordinary English text, we use the following correspondence between alphabetic characters and residues modulo 26: $A \leftrightarrow 0$, $B \leftrightarrow 1$, $C \leftrightarrow 2$, $D \leftrightarrow 3$, . . ., $Z \leftrightarrow 25$.

Shift Cipher (2)

In order to be of practical use, a cryptosystem must satisfy two properties:

1. Given the key, encryption and decryption can be computed efficiently.
2. Without knowing the key, determining (any information about) the cleartext from the ciphertext should be infeasible.

Substitution Cipher

$\mathcal{P} = \mathcal{C} = \mathbb{Z}_{26}$. $\mathcal{K}$ is the set of all permutations of the 26 symbols $0, 1, 2, 3, \ldots, 25$. For each permutation $\pi \in \mathcal{K}$, define

$$e_\pi(x) = \pi(x)$$

and

$$d_\pi(y) = \pi^{-1}(y)$$

where π^{-1} is the inverse permutation to π.

Permutation Cipher

Let m be a positive integer, and let $\mathcal{P} = \mathcal{C} = (\mathbb{Z}_{26})^m$, i.e., $\mathbb{Z}_{26} \times \cdots \times \mathbb{Z}_{26}$, m times. Let $\mathcal{K}$ be the set of all permutations of the set $\{1, \ldots, m\}$. For each $\pi \in \mathcal{K}$:

$$e_\pi(x_1, \ldots, x_m) = x_{\pi(1)}, x_{\pi(2)}, \ldots, x_{\pi(m)}$$

and

$$d_\pi(y_1, \ldots, y_m) = y_{\pi^{-1}(1)}, \ldots, y_{\pi^{-1}(m)}$$

where π^{-1} is the inverse permutation of π.

Remark 1.34.

Given ciphertext only, it can be difficult to break the permutation cipher for large m. Knowledge about plaintext language/format can improve efficiency of attacks.

Remark 1.35.

Even though permutation and substitution ciphers alone are quite weak, repeated application of them appears to be much stronger. Such combinations are the basis of modern block ciphers such as DES, as we will see later.

Vigenère Cipher

Let m be a positive integer. $\mathcal{P} = \mathcal{C} = \mathcal{K} = (\mathbb{Z}_{26})^m$. For a key $K = (k_1, k_2, \ldots, k_m)$,

$$e_k(x_1, x_2, \ldots, x_m) = ((x_1 + k_1) \bmod 26, \ldots, (x_m + k_m) \bmod 26)$$

and

$$d_k(y_1, y_2, \ldots, y_m) = ((y_1 - k_1) \bmod 26, \ldots, (y_m - k_m) \bmod 26).$$

Remark 1.36.

Unlike the cryptosystems introduced so far, the Vigenère cipher is a polyalphabetic cryptosystem. That is, each plaintext character can be encrypted as one of m possible ciphertext characters, depending on where in the plaintext it occurs.

Remark 1.37.

Vigenère Cipher = series of different Caesar cipher based on the letters of a keyword; developed around 1467, remained unbroken for about 400 years

Remark 1.38. Idea for cryptanalysis

try to find key length (Kasiski's test = search the ciphertext for pairs of identical segments)

Stream Ciphers (1)

Idea: Successive plaintext elements are encrypted using a different key.

Definition 1.39.

A stream cipher is a tuple $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{L}, \mathcal{E}, \mathcal{D})$ together with a function g, such that the following conditions are satisfied:

1. $\mathcal{P}, \mathcal{C}, \mathcal{K}$ are defined as before
2. $\mathcal{L}$ is a finite set called the keystream alphabet.
3. g is the keystream generator. g takes a key K as input, and generates an infinite string $z_1 z_2 \ldots$ called the keystream, where $z_i \in \mathcal{L}$ for $i \geq 1$.
4. For each $z \in \mathcal{L}$, there is an encryption rule $e_z \in \mathcal{E}$ and a corresponding decryption rule $d_z \in \mathcal{D}$ such that $e_z : \mathcal{P} \to \mathcal{C}$, $d_z : \mathcal{C} \to \mathcal{P}$ and $d_z(e_z(x)) = x$ for every plaintext element $x \in \mathcal{P}$.

Stream Ciphers (2)

Remark 1.40.

1. The Vigenère cipher can be defined as a stream cipher: Let m be the keyword length, $\mathcal{K} = (\mathbb{Z}_{26})^m$ and $\mathcal{P} = \mathcal{C} = \mathcal{L} = \mathbb{Z}_{26}$. Define

$$e_z(x) = (x + z) \bmod 26$$

and

$$d_z(y) = (y - z) \bmod 26.$$

The keystream $z_1 z_2 \ldots$ is defined as

$$z_i = \begin{cases} k_i & \text{if } 1 \leq i \leq m \\ z_{i-m} & \text{if } i \geq m+1, \end{cases}$$

where $K = (k_1, \ldots, k_m)$. This generates the keystream

$$k_1 k_2 \ldots k_m k_1 k_2 \ldots k_m k_1 k_2 \ldots$$

Stream Ciphers (3)

2. A block cipher can be viewed as a special case of a stream cipher where the key stream is constant: $z_i = K$ for all $i \geq 1$.
3. Stream ciphers are often described in terms of binary alphabets, i.e., $\mathcal{P} = \mathcal{C} = \mathcal{L} = \mathbb{Z}_2$. Encryption and decryption operations are simply additions modulo 2 (respectively the XOR operation $\oplus$):

$$e_z(x) = (x + z) \bmod 2 = x \oplus z$$

and

$$d_z(y) = (y + z) \bmod 2 = y \oplus z, \text{ for } x, y, z \in \mathbb{Z}_2.$$

We will see later that this cipher is perfectly secure under certain assumptions. In that context, this is referred to as one time pad.

4. Examples for common stream ciphers: RC4 (WLAN-WEP), A5/1, A5/2 (GSM encryption), E0 (Bluetooth)

Stream Ciphers (4)

Types of Stream Ciphers

- Synchronous: generate key stream independent of plaintext
- Asynchronous: use previous ciphertext

LFSR (Linear Feedback Shift Register)

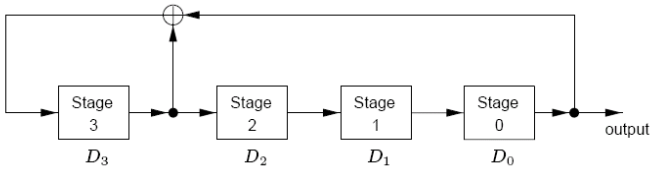
$$z_{m+i} = \sum_{j=0}^{m-1} c_j z_{i+j} \bmod 2 \qquad i \geq 1$$

- Initial state: m-tupel $(z_1, \ldots, z_m) = (k_1, \ldots, k_m)$,
  taps $c_0, \ldots, c_{m-1} \in \mathbb{Z}_2$ (generating polynomial $1 + c_0 x + \cdots + c_{m-1} x^m$)
- Nice mathematical structure, easy to analyze: maximal period $2^m - 1$ for all non-zero initial states if generating polynomial c is primitive[5].
- Insecure when used on their own (Berlekamp-Massey algorithm to compute linear complexity)

[5] An irreducible polynomial $c(x) \in \mathbb{Z}_2$ of degree m is called primitive if x is a generator of $F_{2^m}^*$.

Stream Ciphers (5)

Example 1.41. LFSR of length $m = 4$ with maximal period: $c = 1 + x + x^4$
Consider initial state $(0, 1, 1, 0)$:



t	3	2	1	0
0	0	1	1	0
1	0	0	1	1
2	1	0	0	1
3	0	1	0	0
4	0	0	1	0
5	0	0	0	1
6	1	0	0	0
7	1	1	0	0

t	3	2	1	0
8	1	1	1	0
9	1	1	1	1
10	0	1	1	1
11	1	0	1	1
12	0	1	0	1
13	1	0	1	0
14	1	1	0	1
15	0	1	1	0

Stream Ciphers (6) — Pro's and Con's

+ very fast
+ little hardware resources

$\Longleftrightarrow$

- difficult to analyze (no proofs for real-world stream ciphers)
- difficult to use, e.g., RC4 in WEP

Recent projects

- NESSIE (New European Schemes for Signatures, Integrity, and Encryption): EU 2000–2003: open competition, no selection for stream ciphers
- eSTREAM: EU 2004–2008: open competition for new stream ciphers

Stream Ciphers (7) — eSTREAM results

- eSTREAM released portfolio of 8 stream ciphers in April 2008

Profile 1: software	Profile 2: hardware
HC-128	Trivium
Rabbit	Grain
Salsa	MICKEY
SOSEMANUK	F-FCSR

- F-FCSR was removed from portfolio in September 2008 due to cryptanalytic attacks
- Status of stream ciphers IMHO still disappointing
- In case of doubt, use AES-CTR!

4. Attack Models for Encryption Schemes

1. **Ciphertext only attack:** The opponent possesses only one or more strings of ciphertext.

2. **Known plaintext attack:** The adversary has one or more pairs of plaintext/ciphertext.

3. **Chosen plaintext attack:** The adversary has one or more pairs of plaintext/ciphertext, for plaintext of his choosing.

4. **Chosen ciphertext attack:** The adversary has one or more pairs of plaintext/ciphertext, for ciphertexts of his choosing.

Adversary Goals for Encryption Schemes

1. **Total break:** Adversary learns the decryption key (with non-negligible probability).

2. **Partial break:** The adversary is able to decrypt a ciphertext for which he has not seen the plaintext.

3. **Distinguishability of ciphertexts:** With probability non-negligibly greater than 0.5, the adversary can distinguish between the encryptions of two plaintexts he has not seen before. A cryptosystem that does not permit distinguishability of ciphertexts is said to be semantically secure.

Semantic security is only appropriate for passive adversaries, for active attackers more advanced security notions are required.

5. Notions of Security of a Cryptosystem

- **Computational Security:** Quantifying the computational effort (or work factor) required to break a cryptosystem. A system is secure if it requires at least N operations to break, for some specified (and hopefully large) N.

- **Provable Security:** Showing security relative to a specific assumption. That is, to reduce the security of the cryptosystem to a well-studied problem that is thought to be difficult.

- **Unconditional/Perfect Security:** Even with infinite computational resources, the cryptosystem cannot be broken. This is also called information-theoretic security.

Exhaustive key search dimensions

number (of)	$\log_2$ of number
1000	$9.97 \approx 10$
1000000	≈ 20
seconds in a year	24.9
age of solar system (in years)	32.6
clock cycles per year (1000 MHz)	54.9
DES keys	56
AES keys	128, 192, 256
electrons in the universe	259
number of 512-bit prime numbers	502

Source: Ueli Maurer, Advanced Technology Seminars, 2001.

Modern Security Notions for Encryption Schemes

Remark 1.42.

- In modern cryptographic theory, more advanced security models are used, e. g.
  - Indistinguishability: It is not possible to distinguish pairs of ciphertexts based on the message they encrypt.
  - Non-malleability: It is not possible to transform a ciphertext into another ciphertext which decrypts to a related, meaningful plaintext.
- Properties of Encryption Schemes
  - IND-CPA=indistinguishability (IND) under chosen plaintext attacks (CPA) given access to an encryption oracle
  - IND-CCA1=indistiguishability (IND) under non-adaptive (1) chosen-ciphertext attacks (CCA) given access to an decryption oracle
  - IND-CCA2=indistiguishability (IND) under adaptive (2) chosen-ciphertext attacks (CCA) given access to an decryption oracle
- IND-CPA=semantically secure cryptosystem. It holds IND-CCA2 $\rightarrow$ IND-CCA1 $\rightarrow$ IND-CPA.

Part 1: Outline — Section 2: Probability Theory and Perfect Secrecy

- 1. Review of basic probability theory
- 2. Perfect secrecy
- 3. One-time pad
- 4. Product cryptosystems

1. Review of Basic Probability Theory

Definition 2.1.

A discrete random variable $\mathbf{X}$ consists of a finite set X and a probability distribution defined on X. The probability that the random variable $\mathbf{X}$ takes on the value $x \in X$ is denoted $\Pr[\mathbf{X} = x]$, sometimes abbreviated as $\Pr[x]$ if $\mathbf{X}$ is fixed. Two properties must be satisfied:

$$0 \leq \Pr[x] \; \forall x \in \mathbf{X}$$

$$\sum_{x \in X} \Pr[x] = 1$$

Definition 2.2.

An event E is a subset of X. The probability that E occurs, denoted $\Pr[E]$ is:

$$\Pr[E] = \Pr[x \in E] = \sum_{x \in E} \Pr[x]$$

The complementary event is denoted by $\overline{E}$.

Review of Basic Probability Theory (2)

Theorem 2.3.
Let $E \subseteq X$:

1. $0 \leq \Pr[E] \leq 1, \Pr[\mathbf{X}] = 1, \Pr(\emptyset) = 0$
2. $\Pr[\overline{E}] = 1 - \Pr[E]$
3. If the outcomes in X are equally likely, then $\Pr[E] = \frac{|E|}{|X|}$

Definition 2.4.
Suppose $\mathbf{X}$ and $\mathbf{Y}$ are random variables defined on finite sets X and Y. The joint probability $\Pr[x, y]$ is the probability that $\mathbf{X} = x$ and $\mathbf{Y} = y$. The conditional probability $\Pr[x|y]$ denotes the probability that $\mathbf{X} = x$ given $\mathbf{Y} = y$. Conditional probability and joint probability are related: $\Pr[x, y] = \Pr[x|y] \Pr[y]$.
$\mathbf{X}$ and $\mathbf{Y}$ are independent if $\Pr[x, y] = \Pr[x] \Pr[y]$ for all $x \in \mathbf{X}$ and $y \in \mathbf{Y}$.

Theorem 2.5. (Bayes' Theorem)
If x, y are events with $\Pr[y] > 0$, then

$$\Pr[x|y] = \frac{\Pr[x] \Pr[y|x]}{\Pr[y]}. \tag{1}$$

2. Perfect Secrecy (1)

Assumptions for this section:

- For a cryptosystem $\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D}$, a particular key $K \in \mathcal{K}$ is used only once (i.e., to encrypt a single plaintext element).
- There is a probability distribution on the plaintext space $\mathcal{P}$. We denote the *a priori* probability that plaintext x occurs by $\Pr[\mathbf{x} = x]$.
- Key K is chosen using some fixed probability distribution. The corresponding random variable is denoted by $\mathbf{K}$.
- The key and the plaintext are independent random variables.

The probability distributions on $\mathcal{P}$ and $\mathcal{K}$ induce a probability distribution on $\mathcal{C}$. Thus, the ciphertext element is also a random variable $\mathbf{y}$. We denote the set of possible ciphertexts if K is the key by $C(K) = \{e_K(x) : x \in \mathcal{P}\}$. Thus,

$$\Pr[\mathbf{y} = y] = \sum_{\{K : y \in C(K)\}} \Pr[\mathbf{K} = K] \Pr[\mathbf{x} = d_K(y)] \quad \text{for every } y \in \mathcal{C}. \tag{2}$$

Perfect Secrecy (2)

For any $y \in \mathcal{C}$ and $x \in \mathcal{P}$:

$$\Pr[\mathbf{y} = y | \mathbf{x} = x] = \sum_{\{K : x = d_K(y)\}} \Pr[\mathbf{K} = K]. \tag{3}$$

Applying Bayes' theorem (1), anyone who knows the probability distributions (2) and (3) can compute the probability that x is the plaintext given that y is the ciphertext:

$$\Pr[\mathbf{x} = x | \mathbf{y} = y] = \frac{\Pr[\mathbf{X} = x] \times \sum_{\{K : x = d_K(y)\}} \Pr[\mathbf{K} = K]}{\sum_{\{K : y \in C(K)\}} \Pr[\mathbf{K} = K] \Pr[\mathbf{x} = d_K(y)]} \tag{4}$$

Since an attacker should not obtain any information about the plaintext by observing the ciphertext, perfect secrecy is formulated as follows:

Definition 2.6.
A cryptosystem has perfect secrecy if $\Pr[x|y] = \Pr[x]$ for all $x \in \mathcal{P}$, $y \in \mathcal{C}$.

Perfect Secrecy (3) — Toy Example for missing perfect secrecy

Plaintext $\mathcal{P} = \{a, b\}$, $\Pr[a] = 1/4$, $\Pr[b] = 3/4$,
Key $\mathcal{K} = \{K_1, K_2, K_3\}$, $\Pr[K_1] = 1/2$, $\Pr[K_2] = \Pr[K_3] = 1/4$,
Ciphertext $\mathcal{C} = \{1, 2, 3, 4\}$
Encryption matrix

	a	b
K_1	1	2
K_2	2	3
K_3	3	4

Probability distributions $\Pr[\mathbf{y} = y]$ on $\mathcal{C}$ using (2):

$$\Pr[1] = \frac{1}{2} \times \frac{1}{4} = \frac{1}{8} \qquad \Pr[2] = \frac{1}{2} \times \frac{3}{4} + \frac{1}{4} \times \frac{1}{4} = \frac{7}{16}$$

$$\Pr[3] = \frac{1}{4} \times \frac{3}{4} + \frac{1}{4} \times \frac{1}{4} = \frac{4}{16} \qquad \Pr[4] = \frac{1}{4} \times \frac{3}{4} = \frac{3}{16}$$

Perfect Secrecy (4) — Toy Example for missing perfect secrecy (2)

Conditional probability $\Pr[\mathbf{x} = x|\mathbf{y} = y]$ using (4):

$$\Pr[a|1] = 1 \qquad \Pr[b|1] = 0$$

$$\Pr[a|2] = \frac{\frac{1}{4} \times \frac{1}{4}}{\frac{7}{16}} = \frac{1}{7} \qquad \Pr[b|2] = \frac{\frac{3}{4} \times \frac{1}{2}}{\frac{7}{16}} = \frac{6}{7}$$

$$\Pr[a|3] = \frac{\frac{1}{4} \times \frac{1}{4}}{\frac{4}{16}} = \frac{1}{4} \qquad \Pr[b|3] = \frac{\frac{3}{4} \times \frac{1}{4}}{\frac{4}{16}} = \frac{3}{4}$$

$$\Pr[a|4] = 0 \qquad \Pr[b|4] = 1$$

Hence, for $y = 3$: $\Pr[x|y] = \Pr[x]$ for $x = \{a, b\}$,
but for $y = 1, 2, 4$: $\Pr[x|y] \neq \Pr[x]$ for $x = \{a, b\}$ no perfect secrecy

Perfect Secrecy (5)

Theorem 2.7.
If the 26 keys in the Shift Cipher are used with equal probability, then for any plaintext probability distribution, the Shift Cipher has perfect secrecy.

Proof: see textbook. Recall that this is under the assumption that the key is used only to encrypt a single letter.

Theorem 2.8.
Suppose $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$ is a cryptosystem with $\Pr[y] > 0$ (for all $y \in \mathcal{C}$) that provides perfect secrecy. Then $|\mathcal{K}| \geq |\mathcal{P}|$.

Proof: Consider $x \in \mathcal{P}$ with $\Pr[x] > 0$. Using Bayes' Theorem
$\Pr[x|y] = \Pr[y]$ for all $y \in \mathcal{C} \iff \Pr[y|x] = \Pr[y]$ for all $y \in \mathcal{C}$.
Fix any $x \in \mathcal{P}$. For each $y \in \mathcal{C}$: $\Pr[y|x] = \Pr[y] > 0$. Hence, there exists at least one key K with $e_K(x) = y$, i.e., $|\mathcal{K}| \geq |\mathcal{C}|$. Because of injectivity of encoding rule, it always holds $|\mathcal{C}| \geq |\mathcal{P}|$, thus $|\mathcal{K}| \geq |\mathcal{P}|$. □

3. One-time Pad (1)

Let $n \geq 1$ be an integer, and $\mathcal{P} = \mathcal{C} = \mathcal{K} = (\mathbb{Z}_2)^n$. For $K = (K_1, \ldots, K_n) \in \mathcal{K}$ define

$$e_K(x) = x \oplus K = (x_1 \oplus K_1, \ldots, x_n \oplus K_n) \bmod 2$$
$$d_K(y) = y \oplus K = (y_1 \oplus K_1, \ldots, y_n \oplus K_n) \bmod 2$$

Theorem 2.9.
The one-time pad provides perfect secrecy.

Proof: If K is chosen uniformly at randomly, then $y = x \oplus K$ is uniformly random, regardless of the distribution on $\mathcal{P}$. That is, $\Pr[y|x] = \Pr[y]$. Hence, it follows from Bayes' theorem that $\Pr[x|y] = \Pr[x]$, i.e., perfect secrecy is obtained.

Remark 2.10.
The one-time pad was first realized by Gilbert Vernam in 1917 (Vernam chiffre). Its perfect security was proven by Shannon 1948 in his landmark paper "Communication Theory of Secrecy Systems".

One-time Pad (2)

Questions

- Is it possible to efficiently use the one-time pad in practice?
- What is the implication with respect to perfect secrecy if the same key is used to encrypt two different plaintexts, i.e.,

$$e_K(x_1) = x_1 \oplus K = y_1$$

and

$$e_K(x_2) = x_2 \oplus K = y_2?$$

4. Product Cryptosystems

Given cryptosystems S_1 and S_2 with $\mathcal{C}_1 = \mathcal{P}_1 = \mathcal{C}_2 = \mathcal{P}_2$, with encryption functions e^1 and e^2 and decryption functions d^1 and d^2, $S_1 \times S_2$ is the system with

$$e_{k_1,k_2}(x) = e^2_{k_2}(e^1_{k_1}(x))$$

$$d_{k_1,k_2}(y) = d^1_{k_1}(d^2_{k_2}(y))$$

Definition 2.11.

A cryptosystem is idempotent if $S \times S = S$.

$\Longrightarrow$ No extra security. No point in using $S \times S$ instead of S.

Example 2.12.

The shift cipher is idempotent.

The substitution and permutation ciphers are idempotent.

But: Use combination of ciphers which are non-idempotent to increase security!

Part 1: Outline — Section 3: Block Ciphers

- 1. Block ciphers: Introduction, Iterated Block Ciphers
- 2. Substitution-Permutation networks
- 3. Cryptanalysis
- 4. Data Encryption Standard DES
- 5. Advanced Encryption Standard AES
- 6. Modes of operation

- Most block ciphers are product ciphers, incorporating a sequence of permutation and substitution operations.
- Most commonly used design is that of an iterated cipher.

Definition 3.1. Iterated Block Cipher

An iterated block cipher is a product cipher consisting of repeated application of similar rounds of encryption. It requires specification of a round function and a key schedule. Encryption then takes place in Nr similar rounds.

- The key K is used to construct the key schedule, consisting of Nr round keys $K^1, \ldots, K^{\mathrm{Nr}}$.
- The round function g takes two inputs: round key K^r and a current state w^{r-1}. The next state is defined as $w^r = g(w^{r-1}, K^r)$ with $w^0 = x$ (plaintext).
- The ciphertext is $y = w^{\mathrm{Nr}}$.

Remark 3.2.

In order for decryption to be possible, the round function g has to be injective if its second argument is fixed. This is equivalent to the existence of g^{-1} with

$$g^{-1}(g(w, y), y) = w$$

for all w and for all y.

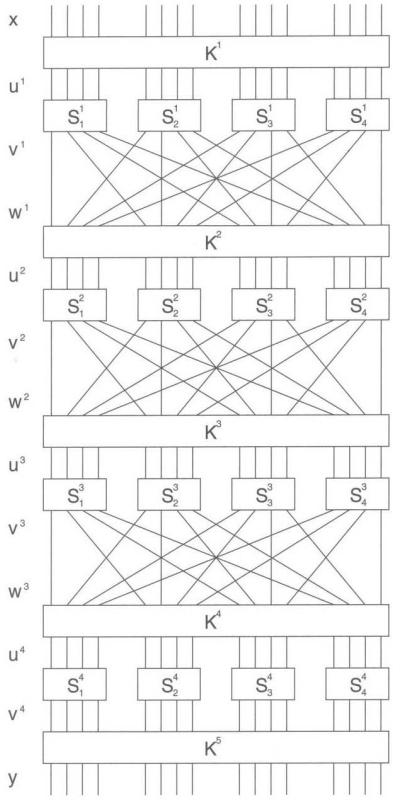
2. Substitution-Permutation Networks (SPN)

- Special type of iterated cipher.
- ℓ, m positive integers, block length is ℓm. Components of the SPN:

$$\pi_S : \{0,1\}^\ell \to \{0,1\}^\ell \qquad \text{(S-box)}$$
$$\pi_P : \{1,\ldots,\ell m\} \to \{1,\ldots,\ell m\} \qquad \text{(Permutation)}$$

- π_S permutates ℓ bits, π_P permutates ℓm bits
- Plaintext: ℓm-bit binary string $x = (x_1,\ldots,x_{\ell m}) = x_{\langle 1\rangle}||\ldots||x_{\langle m\rangle}$, i.e., concatenation of m ℓ-bit substrings, where $x_{\langle i\rangle} = (x_{\ell(i-1)+1},\ldots,x_{\ell i})$.
- SPN consists of Nr rounds. In each round (except the last) π_S is followed by π_P.
- Key Schedule $(K^1,\ldots,K^{\mathrm{Nr}+1})$, $K^i \in \{0,1\}^{\ell m}$
- Diagram, see next slide
- Design can be implemented efficiently in software and hardware.
- AES is a typical SPN cipher with some modifications.

Diagram of a substitution-permutation network



Encryption with SPN

Algorithm SPN $\left(x, \pi_S, \pi_P, \left(K^1, \ldots, K^{Nr+1}\right)\right)$

$w^0 := x$; /* initial state */
for $r := 1$ **to** $Nr - 1$ **do**
  begin
    $u^r := w^{r-1} \oplus K^r$; /* round key mixing, input to S-box */
    for $i := 1$ **to** m **do**
      $v^r_{\langle i \rangle} := \pi_S\left(u^r_{\langle i \rangle}\right)$; /* S-box substitution */
    $w^r := \left(v^r_{\pi_P(1)}, \ldots, v^r_{\pi_P(\ell m)}\right)$; /* permutation */
  end
$u^{Nr} := w^{Nr-1} \oplus K^{Nr}$; /* round key mixing, input to S-box */
for $i := 1$ **to** m **do**
  $v^{Nr}_{\langle i \rangle} := \pi_S\left(u^{Nr}_{\langle i \rangle}\right)$; /* S-box substitution, no permutation in last round */
$y := v^{Nr} \oplus K^{Nr+1}$;
return y

3. Cryptanalysis of Block Ciphers

- **Exhaustive search:** Can be carried out with one known plaintext/ciphertext pair, or with one or more ciphertexts looking for meaningful messages. 2^n possibilities for an n-bit key. On average, one would expect to test half the keys before finding the right one.
- **Linear cryptanalysis:** known-plaintext total break attack requiring a large number of plaintext/ciphertext pairs. The attack requires finding of certain linear relationships between plaintext bits and a subset of state bits immediately preceding the substitutions performed in the last round. Works best for low round variants. (More details available in text.)
- **Differential cryptanalysis:** chosen-plaintext total break attack requiring a large number of plaintext/ciphertext pairs encrypted with the same key K. The attacker gets to specify the plaintexts. In particular, the attacker uses plaintexts with fixed difference $d = x_1 \oplus x_2$. (More details available in text.)

4. Data Encryption Standard (1)

- In 1973, the National Bureau of Standards published a solicitation for cryptosystems in the Federal Register.
- IBM proposed Lucifer (designed by Feistel, Meyer and Tuchman).
- DES (a modification of Lucifer) was published on March 17, 1975.
- In January 1977, DES was adopted as standard, and reviewed every 5 years.
- Despite eventual feasible attacks, particularly exhaustive key search, DES held up remarkably well throughout (and even beyond) its expected 20-year lifetime.
- In 1997, National Institute of Standards and Technology (NIST) solicited candidates for a new advanced encryption standard which was adopted in 2000.
- January 1999, last renewal of DES
- In May 2005, NIST has withdrawn FIPS 46-3 which approved DES as standard for Federal use.
- Two-Key Triple-DES and Three-Key Triple DES are still widely used.
- NIST approves Two-Key Triple-DES only until 2009. Three-Key Triple-DES will probably coexist with AES until 2030.

DES (2)

- Criticisms:
  - Short keys
  - Design of S-boxes
  - People were suspicious about NSA's involvement (possible inclusion of trapdoor function)
- Attacks:
  - Differential Cryptanalysis (Biham and Shamir, 1990): requires 2^{47} chosen plaintexts
  - Linear Cryptanalysis (Matsui, 1993): requires 2^{43} known plaintexts
  - Special purpose hardware for speeding up exhaustive key search, e.g., COPACOBANA, Bochum, 2006.
- DES Challenges:
  - DES Challenge I: in 1997 - 96 days
  - DES Challenge II-1: in February 1998 - 41 days
  - DES Challenge II-2: in July 1998 - 56 hours
  - DES Challenge III: in January 1999 - 22 hours

DES (3)

Definition 3.3.

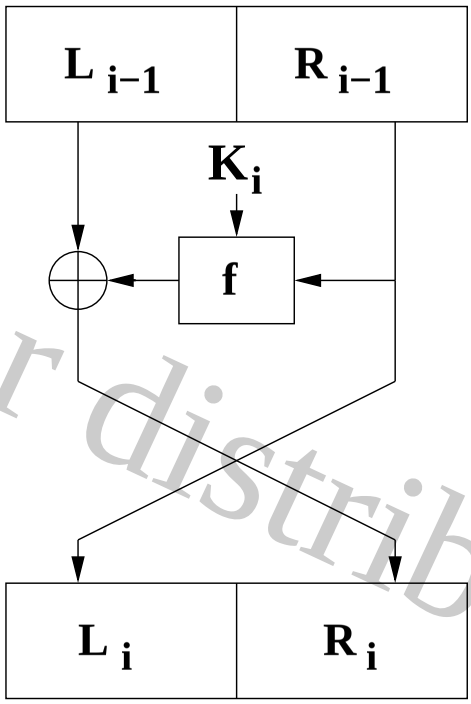
A Feistel cipher is an iterated cipher mapping a $2t$-bit plaintext (L_0, R_0) to a ciphertext (R_r, L_r), through an r round process where $r \geq 1$. For $1 \leq i \leq r$, round i maps (L_{i-1}, R_{i-1}) to (L_i, R_i) using K_i as follows:

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

Remark 3.4.

- Decryption: same r round process with reversed key schedule
- f need not be invertible
- DES is a Feistel cipher enclosed by two permutations IP and IP^{-1}. While they are not needed from a cryptographic point of view, they were added in the design to slow down software implementations.

DES (4)



One round of a Feistel cipher

DES (5)

Parameters:

- $r = 16$ rounds
- $\mathcal{P} = \mathcal{C} = \{0,1\}^{64}$, i.e., 64 bit block width
- $\mathcal{K} = \{(k_1, \ldots, k_{64}) \in \{0,1\}^{64} : \sum_{j=1}^{8} k_{8l+j} \equiv 1 \pmod 2, 0 \leq l \leq 7\}$

  56 key bits + 8 parity bits

  $\Longrightarrow 2^{56} \approx 7.2 \cdot 10^{16}$ DES keys
- 48 bit round keys K_i, $i = 1, \ldots, 16$
- 8 weak keys (lead to identical round keys)
- DES is bit-oriented, thus slow in software (cf. byte-oriented AES)

DES round function

- $f(R_{i-1}, K_i) := P(S(E(R_{i-1}) \oplus K_i))$ – round function
- $E : \{0,1\}^{32} \to \{0,1\}^{48}$ – expansion of right side
- $S : \{0,1\}^{8\times 6} \to \{0,1\}^{8\times 4}$ – S-boxes, 8 different S-Boxes with 6 bit input and 4 bit output
- $P : \{0,1\}^{32} \to \{0,1\}^{32}$ – round permutation
- The design of the S-boxes (only non-linear part) is crucial for the security of DES. In 1974, IBM together with NSA already considered differential cryptanalysis, discovered in public only 1990.

DES Key Schedule

Algorithm DES Encryption Key Schedule

Input: 64-bit Key $K = k_1, \ldots, k_{64}$

Output: sixteen 48-bit round keys K_i, $i = 1, \ldots, 16$

1. number of left rotations per round i: $v_i = 1$ for $i = 1, 2, 9, 16$, $v_i = 2$ else
2. $(C_0, D_0) = PC1(K)$; $PC1$ – permutated choice 1, C_0, D_0 28-bit halves, $PC1$ discards the 8 parity bits in K
3. **for** $i := 1$ **to** 16 **do**
   $C_i := C_{i-1} \lll v_i$; $D_i := D_{i-1} \lll v_i$;
   $K_i := PC2(C_i, D_i)$; $PC2$ – permutated choice 2, 56 bit

DES Decryption Schedule can be generated by the same algorithm and using the round keys in reverse order or directly by replacing left-shift rotates by right-shift rotates and changing $v_1 = 1$ to 0.

DES Algorithm

Algorithm Data Encryption Standard (DES)

Input: plaintext $m = m_1, \ldots, m_{64}$, 64-bit Key $K = k_1, \ldots, k_{64}$

Output: ciphertext $c = c_1, \ldots, c_{64}$

1. DES Key Schedule: compute sixteen 48-bit round keys K_i, $i = 1, \ldots, 16$
2. $(L_0, R_0) = \mathrm{IP}(m_1, \ldots, m_{64})$; initial permutation
3. **for** $i := 1$ **to** 16 **do**
   $L_i = R_{i-1}$;
   $R_i = L_{i-1} \oplus P(S(E(R_{i-1}) \oplus K_i))$;
4. $c = \mathrm{IP}^{-1}(R_{16}, L_{16})$; exchange final blocks L_{16}, R_{16}; final permuatation

DES in Practice

- Extremely fast in hardware, e.g., in 1992: DEC chip with 50K transistors could encrypt at rate of 1 Gbit/second using clock rate of 250 MHz.
- Important applications: bank transactions
- Two-key triple-DES adopted in ANSI X9.17 and ISO 8732:

$$E_{K_1}(E_{K_2}^{-1}(E_{K_1}(M)))$$

- Triple-key triple-DES:

$$E_{K_1}(E_{K_2}^{-1}(E_{K_3}(M)))$$

Remark 3.5.

Double-DES (and double encryption in general) is susceptible to a “meet-in-the-middle” attack. While this also extends to triple-key triple DES, in this case it requires storing of a large look-up table.

DES in Practice — Implementation Aspects

- DES key scheduling is typically more expensive than a pure encryption. Therefore, in software a separate key schedule (16 round keys a 48 bit) is computed in advance. In hardware, the round keys are usually generated on the fly.
- Size of S-boxes: $8 \times 4 \times 16 \times 0.5$ byte = 256 bytes (LUT in ROM)
- To increase software performance, $P(S(.))$ is often implemented as one large SP-table of size 2 Kbytes.
- Typical software performance: 80 cycles per byte (cf. 14-18 cycles per byte for AES and approx. 240 cycles for Triple DES!)
- Side channel resistant DES implementations using masking as countermeasure require variable S-boxes in RAM!
- Decryption is the same process as encryption with reversed round keys, so no additional hardware overhead, cf. AES.

Remark 3.6.

DES has a complementation property: $\mathrm{Enc}_{\bar{K}}(\bar{P}) = \overline{\mathrm{Enc}_K(P)}$ (Non-Malleability?)

5. Advanced Encryption Standard (AES)

- January 2, 1997: NIST announced the initiation of an effort to develop AES
- September 12, 1997: formal call for AES algorithms with the goal to specify an unclassified, publicly disclosed encryption algorithm available royalty-free, worldwide.

  Requirements:

  - symmetric-key cryptography implemented as block cipher
  - support block size of 128 bits
  - support key sizes of 128, 192 and 256 bits[6]
- August 20, 1998: First AES conference:
  - 15 candidates (industry and academia submitters from 12 countries): CAST-256, Crypton, DEAL, DFC, E2, FROG, HPC, LOKI97, MAGENTA, MARS, RC6, RIJNDAEL, SAFER+, SERPENT, TWOFISH

[6] AES-256 even resists quantum computing brute force attacks

AES (2)

- March 1999: Second AES conference
- August 1999: Announcement of five finalist algorithms:
  - MARS (IBM)
  - RC6 (RSA Laboratories)
  - Rijndael (Joan Daemen, Vincent Rijmen)
  - Serpent (Ross Anderson, Eli Biham, Lars Knudsen)
  - Twofish (Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall, Niels Ferguson)
- April 2000: Third AES conference
- May 15, 2000: End of public comment period for reviewing finalists
- October 2, 2000: NIST announced that Rijndael was selected as the proposed AES
- NIST. *FIPS-197: Advanced Encryption Standard, November 2001*. Available online at `http://www.itl.nist.gov/fipspubs/`

AES (3)

- Evaluation criteria:
  - Security
  - Cost (computational efficiency, memory requirements, etc.)
  - Algorithm and implementation characteristics (flexibility, hardware and software suitability, algorithm simplicity, etc.)
- June 2003: AES-128 also for US classified information and AES-192/-256 for secret and top secret information!

  > *The implementation of AES in products intended to protect national security systems and/or information must be reviewed and certified by NSA prior to their acquisition and use.*

- May 2004, Fourth AES conference, Bonn: cryptanalytic and algebraic attacks, hardware implementation, state of the art of data encryption with AES
- July 2009: Attacks on round-reduced AES-256 by Biryukov, Dunkelman, Keller, Khovratovich and Shamir. Not relevant for practice.

Why Rijndael?

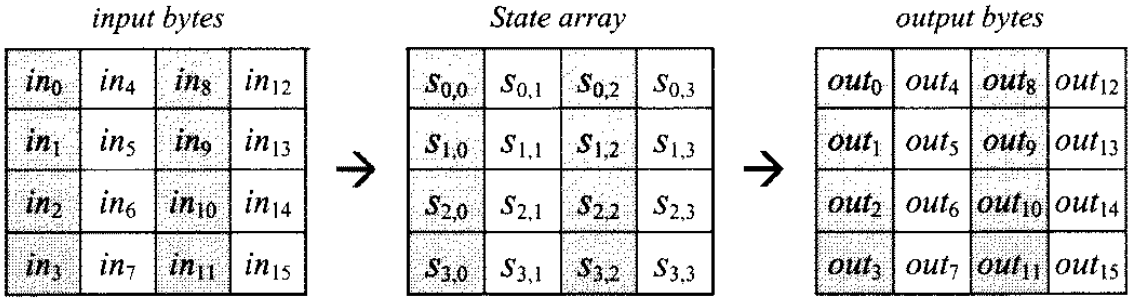
- Good performer in HW/SW across wide range of computing environments
- Key setup time is excellent
- Low memory requirements make it well suited for restricted space environments
- Operations are among the easiest to defend against power and timing attacks (*)
- Flexibility in terms of block and key sizes, can accommodate alterations in number of rounds
- Potential to benefit from instruction-level parallelism

Remark 3.7.

While the authors claim (*) it should be noted that it is still more difficult to defend than in case of DES. This is due to the fact that DES is fully bit-oriented, while AES makes use of both structures GF[2] and GF[2^8].

Rijndael (1)

- Input/output: sequences of 128 bits[7]
- Key: sequence of 128, 192 or 256 bits, $\mathrm{Nk} = 4$, 6 or 8 (number of key words)
- Internally: operations are performed on array of bytes (state)
- State = array of 4 rows with Nb bytes each, $\mathrm{Nb} = 4$
  (number of 32-bit words = number of columns)



Source: FIPS Publication 197

[7] The original Rijndael specification allows additional block widths of 192 and 256 bits.

Rijndael (2)

Key-Block/Round combination:

	Key Length Nk words	Block size Nb words	Number of Rounds Nr
AES-128	4	4	10
AES-192	6	4	12
AES-256	8	4	14

Parameters for next slide:

`in` – 16 byte Input,

`out` – 16 byte Output,

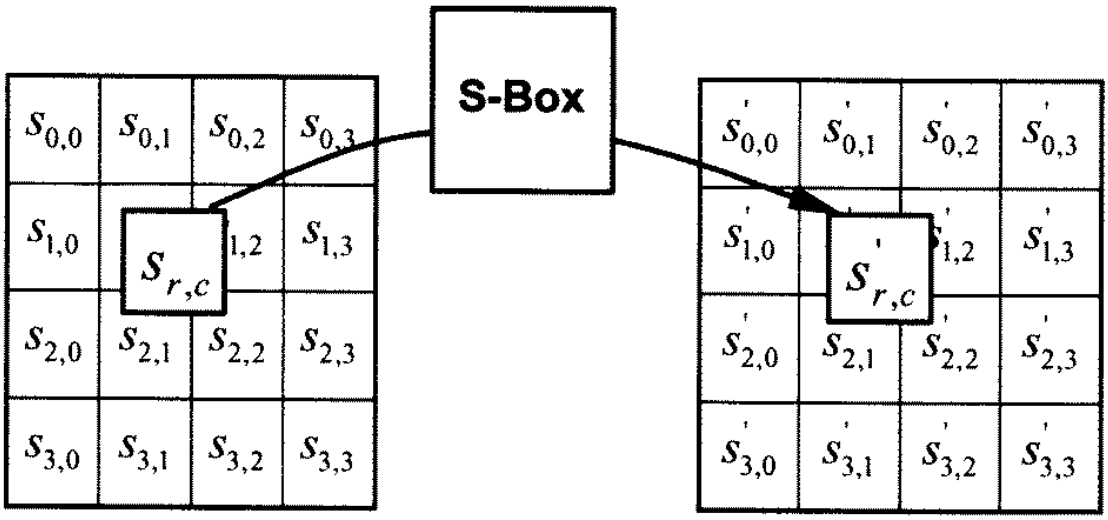
`w` – $\mathrm{Nk} \times (\mathrm{Nr} + 1) \times 32$ bit Key Schedule

Cipher

```
Cipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin
   byte  state[4,Nb]

   state = in

   AddRoundKey(state, w[0, Nb-1])

   for round = 1 step 1 to Nr-1
      SubBytes(state)
      ShiftRows(state)
      MixColumns(state)
      AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
   end for

   SubBytes(state)
   ShiftRows(state)
   AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])

   out = state
end
```

Source: FIPS Publication 197

S-Box (1)



Source: FIPS Publication 197

S-Box (2)

x \ y	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Source: FIPS Publication 197

SubBytes

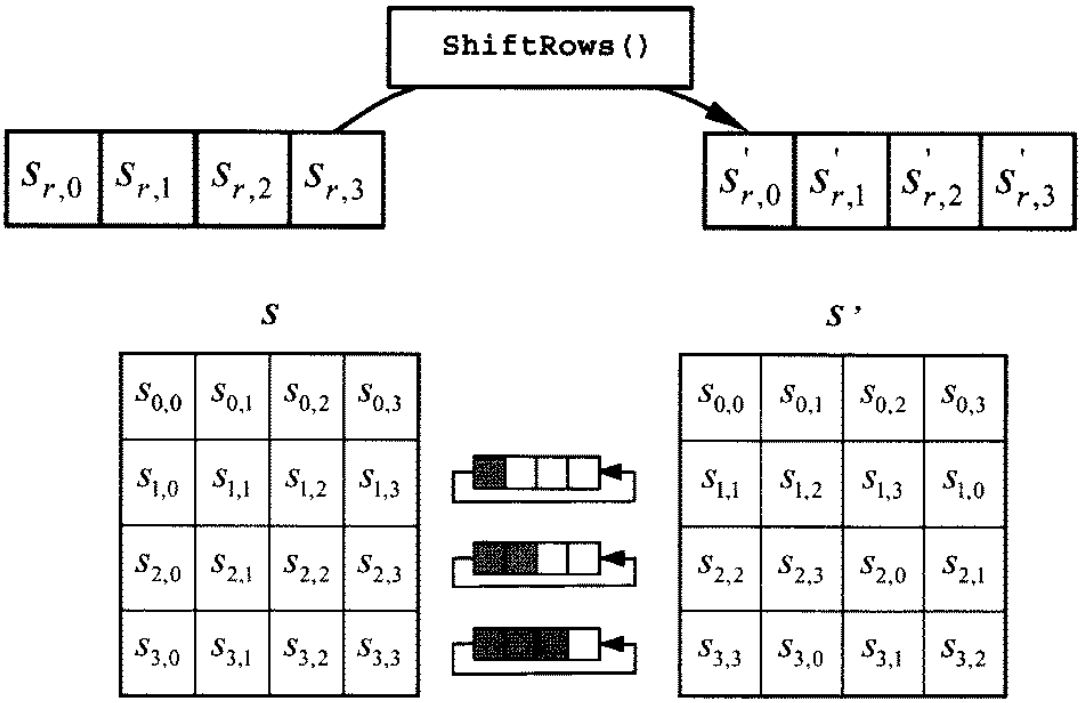
- Non-linear, invertible byte substitution that operates on each byte.
- Composition of two transformations:

1. For byte $b = \{b_7, b_6, \ldots, b_0\}$ and corresponding polynomial $b_7x^7 + \cdots + b_0 \in \mathbb{Z}_2[x]$ compute its multiplicative inverse modulo $m(x) = x^8 + x^4 + x^3 + x + 1$. The element 00 is mapped to itself.
2.

$$\begin{pmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{pmatrix} \oplus \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix} \mod 2$$

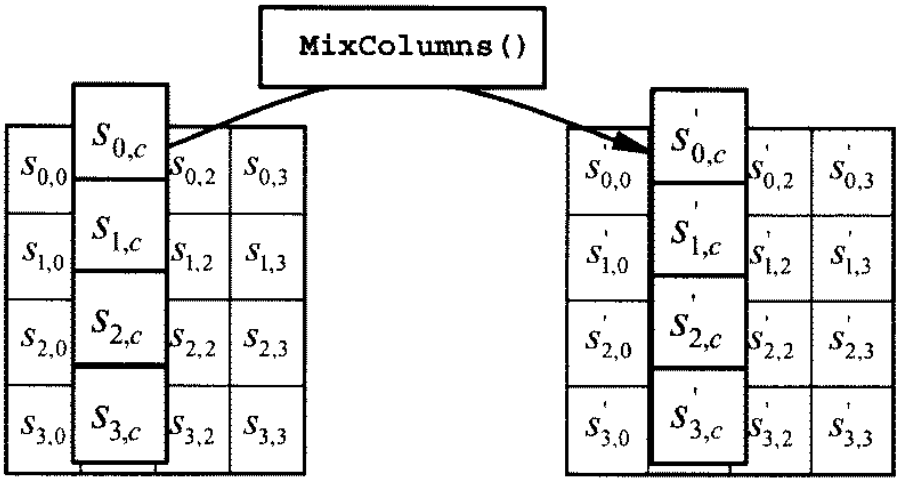
- The operation 1. in SubBytes can be viewed as inversion in the finite field GF[2^8].

ShiftRows



Source: FIPS Publication 197

MixColumns (1)

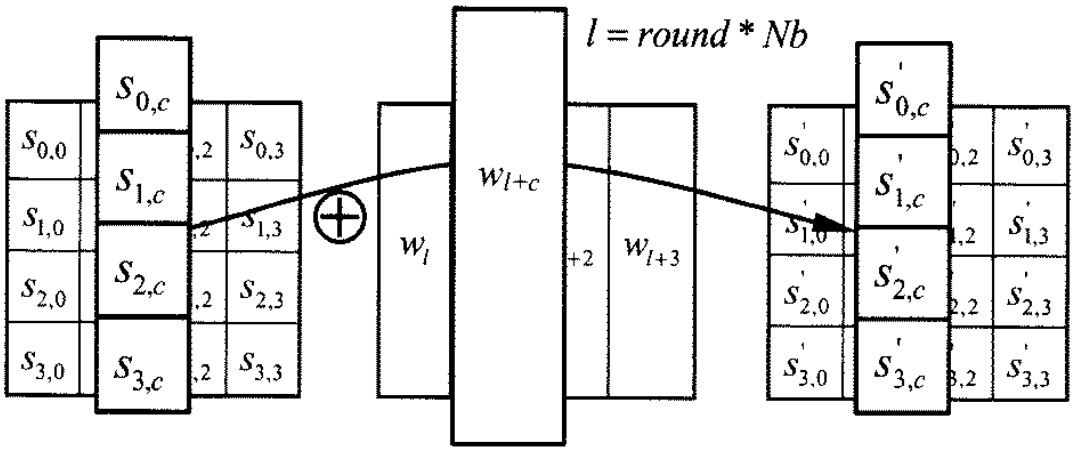


Source: FIPS Publication 197

MixColumns (2)

$$\begin{pmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \begin{pmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{pmatrix} \quad \text{for } 0 \leq c < \text{Nb}$$

AddRoundKey



Source: FIPS Publication 197

Only Nk words of whole key schedule are used in one iteration!

Key Expansion

```
KeyExpansion(byte key[4*Nk], word w[Nb*(Nr+1)], Nk)
begin
   word  temp

   i = 0

   while (i < Nk)
      w[i] = word(key[4*i], key[4*i+1], key[4*i+2], key[4*i+3])
      i = i+1
   end while

   i = Nk

   while (i < Nb * (Nr+1)]
      temp = w[i-1]
      if (i mod Nk = 0)
         temp = SubWord(RotWord(temp)) xor Rcon[i/Nk]
      else if (Nk > 6 and i mod Nk = 4)
         temp = SubWord(temp)
      end if
      w[i] = w[i-Nk] xor temp
      i = i + 1
   end while
end
```

Source: FIPS Publication 197

Key Expansion (2)

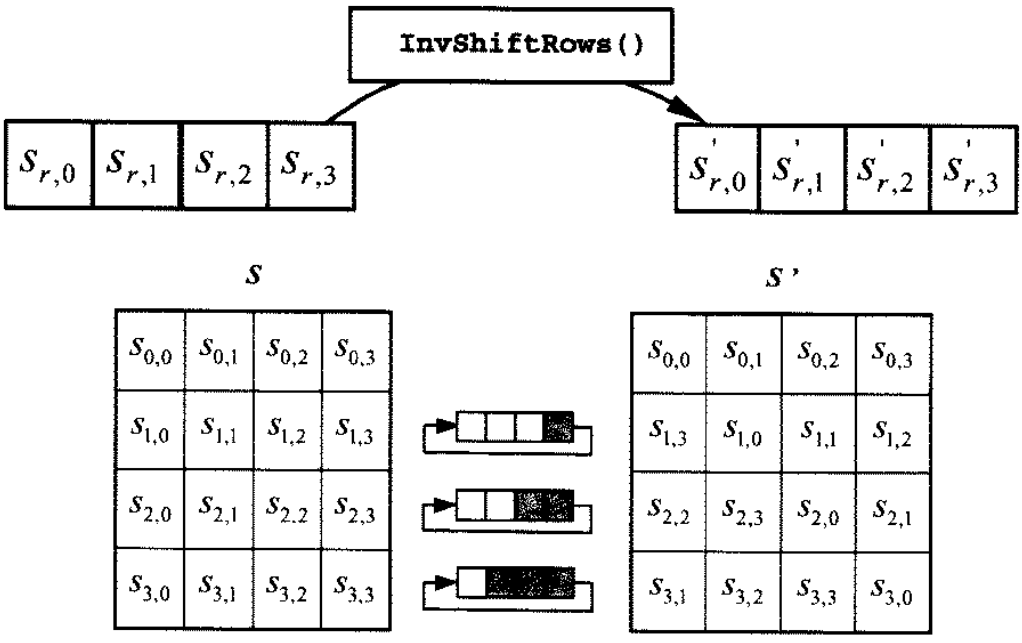
- `SubWord()`: 32 bit input, send each byte to `SubBytes()`
- `RotWord()`: rotate 32 bit word cyclically by 8 bits
- `Rcon`: $rcon(i) = x^{(254+i)} \pmod{m}$ $i = 1, \ldots, 255$
- Expanded key w: $\mathrm{Nk} * (\mathrm{Nr} + 1)$ 32 bit words
- AES-128: $\Longrightarrow \mathrm{Nr} = 10, \mathrm{Nk} = 4$
  default minimum for round keys $w = 11 * 4 * 4 = 176$ bytes,
  memory-saving implementation = 16 bytes

Inverse Cipher

```
InvCipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin
   byte  state[4,Nb]

   state = in

   AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])

   for round = Nr-1 step -1 downto 1
      InvShiftRows(state)
      InvSubBytes(state)
      AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
      InvMixColumns(state)
   end for

   InvShiftRows(state)
   InvSubBytes(state)
   AddRoundKey(state, w[0, Nb-1])

   out = state
end
```

Source: FIPS Publication 197

InvShiftRows



Source: FIPS Publication 197

Inverse S-Box

x \ y	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	52	09	6a	d5	30	36	a5	38	bf	40	a3	9e	81	f3	d7	fb
1	7c	e3	39	82	9b	2f	ff	87	34	8e	43	44	c4	de	e9	cb
2	54	7b	94	32	a6	c2	23	3d	ee	4c	95	0b	42	fa	c3	4e
3	08	2e	a1	66	28	d9	24	b2	76	5b	a2	49	6d	8b	d1	25
4	72	f8	f6	64	86	68	98	16	d4	a4	5c	cc	5d	65	b6	92
5	6c	70	48	50	fd	ed	b9	da	5e	15	46	57	a7	8d	9d	84
6	90	d8	ab	00	8c	bc	d3	0a	f7	e4	58	05	b8	b3	45	06
7	d0	2c	1e	8f	ca	3f	0f	02	c1	af	bd	03	01	13	8a	6b
8	3a	91	11	41	4f	67	dc	ea	97	f2	cf	ce	f0	b4	e6	73
9	96	ac	74	22	e7	ad	35	85	e2	f9	37	e8	1c	75	df	6e
a	47	f1	1a	71	1d	29	c5	89	6f	b7	62	0e	aa	18	be	1b
b	fc	56	3e	4b	c6	d2	79	20	9a	db	c0	fe	78	cd	5a	f4
c	1f	dd	a8	33	88	07	c7	31	b1	12	10	59	27	80	ec	5f
d	60	51	7f	a9	19	b5	4a	0d	2d	e5	7a	9f	93	c9	9c	ef
e	a0	e0	3b	4d	ae	2a	f5	b0	c8	eb	bb	3c	83	53	99	61
f	17	2b	04	7e	ba	77	d6	26	e1	69	14	63	55	21	0c	7d

Source: FIPS Publication 197

InvMixColumns

$$\begin{pmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{pmatrix} = \begin{pmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{pmatrix} \begin{pmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{pmatrix} \text{ for } 0 \leq c < \text{Nb}$$

Function of the Steps

- ByteSub: non-linearity
- ShiftRow: inter-column diffusion
- MixColumn: inter-byte diffusion within columns
- Round key addition

AES in Practice — Implementation Aspects

- Block width 128 $\Longrightarrow$ efficient 8, 16, 32 or 128 bit implementations
- Most expensive operation: SubBytes $S[.]$
  - Look-Up Tables, e.g., 256×8 bit
  - Field Towers: $A \in GF(2^8) \xrightarrow{\text{isomorphism}} (A_1, A_2), A_i \in GF((2^2)^2)$
  - Arithmetic
- Decryption requires different tables than encryption (difference to DES)
  $\Longrightarrow$ look for protocols with encryption only (CTR mode)
- Simple 8-bit implementation: basic operation = multiplication in $GF(2^8)$ with $\{02\} = x$
  1. multiply with $\{02\} = x$: left shift
  2. reduce, if necessary, by $m(x) = x^8 + x^4 + x^3 + x + 1 = \{11B\}$: if MSB is set, add/XOR with $\{11B\}$
  3. `#define xtimes(x)  ((x<<1) ^ (((x>>7) & 1) * 0x1B))`

AES in Practice — Implementation Aspects (2)

- Software performance: fastest implementations on Intel processors achieve 11 cycles/byte (Dan Bernstein/Peter Schwabe 2008); 14 cycles/byte (Helger Lipmaa) or 16 cycles/byte (Brian Gladman)
- Bitsliced implementations on Core 2 achieve 9.2 cycles/byte (Matsui)
- Hardware performance:
  - small: 992 cycles@100 kHz, 3600 GE (Feldhofer, Dominikus, Wolkerstorfer, IAIK Graz, 2004)
  - fast: 68 Gbits/s, 250000 gates (Hodjat, Verbauwhede, KU Leuven, 2004)

AES in Practice — Implementation Aspects (3)

- Fast implementation on 32-bit processors $\Longrightarrow$ combine the transformations SubBytes, ShiftRows, MixColumns and AddRoundKey for one column

$$\begin{pmatrix} s_{3,c} \\ s_{2,c} \\ s_{1,c} \\ s_{0,c} \end{pmatrix}' = \begin{bmatrix} \{02\} & \{01\} & \{01\} & \{03\} \\ \{03\} & \{02\} & \{01\} & \{01\} \\ \{01\} & \{03\} & \{02\} & \{01\} \\ \{01\} & \{01\} & \{03\} & \{02\} \end{bmatrix} \begin{pmatrix} \mathrm{S}[s_{3,c(3)}] \\ \mathrm{S}[s_{2,c(2)}] \\ \mathrm{S}[s_{1,c(1)}] \\ \mathrm{S}[s_{0,c(0)}] \end{pmatrix} \oplus \begin{pmatrix} k_{3,c} \\ k_{2,c} \\ k_{1,c} \\ k_{0,c} \end{pmatrix}$$

- Define four tables $T_i[x]$, $i = 3, \ldots, 0$, each of 256 4-byte words (for $0 \leq x \leq 255$) as follows:

$$T_3[.] = \begin{bmatrix} \{02\} \otimes \mathrm{S}[.] \\ \{03\} \otimes \mathrm{S}[.] \\ \mathrm{S}[.] \\ \mathrm{S}[.] \end{bmatrix}, T_2[.] = \begin{bmatrix} \mathrm{S}[.] \\ \{02\} \otimes \mathrm{S}[.] \\ \{03\} \otimes \mathrm{S}[.] \\ \mathrm{S}[.] \end{bmatrix}, T_1[.] = \begin{bmatrix} \mathrm{S}[.] \\ \mathrm{S}[.] \\ \{02\} \otimes \mathrm{S}[.] \\ \{03\} \otimes \mathrm{S}[.] \end{bmatrix}, T_0[.] = \begin{bmatrix} \{03\} \otimes \mathrm{S}[.] \\ \mathrm{S}[.] \\ \mathrm{S}[.] \\ \{02\} \otimes \mathrm{S}[.] \end{bmatrix}$$

AES in Practice — Implementation Aspects (4)

- AES round with four table lookups and four XOR

$$\begin{pmatrix} s_{3,c} \\ s_{2,c} \\ s_{1,c} \\ s_{0,c} \end{pmatrix}' = T_3[s_{3,c(3)}] \oplus T_2[s_{2,c(2)}] \oplus T_1[s_{1,c(1)}] \oplus T_0[s_{0,c(0)}] \oplus k_{\text{round},c},$$

  where $c(r) = c + r \bmod 4$ with $c(0) = c$, c – actual column index, and $k_{\text{round},c}$ is word c of round key round.
- MixColumns is missing in last round $\Longrightarrow$ different tables for last encryption round $\Longrightarrow 256 \times 4 \times 4 \times 2 = 8192$ byte table space for fast encryption
- Space can be reduced by a factor of four at the expense of three additional rotations in the calculation of each column of the state.
- Similar techniques can be applied for decryption.

More Information on AES

NIST:

`http://www.nist.gov/aes`

AES Lounge:

`http://www.iaik.tu-graz.ac.at/research/krypto/AES/`

AES animation:

`rijndael_ingles2004.swf`

Common block ciphers — Overview

cipher	year	authors	key length	block width	remarks
DES	1975	IBM	56	64	insecure today
2DES, 3DES	1976	IBM	112, 168	64	slow
AES	2001	Rijmen, Daemen	128, 192, 256	128	new standard
IDEA	1990	Massay, Lai	128	64	patented, PGP
TEA	1994	Wheeler, Needham	128	64	fast, but not very secure
Twofish	1998	Schneier et al.	128, 192, 256	128	AES candidate, popular in open source

6. Modes of Operation

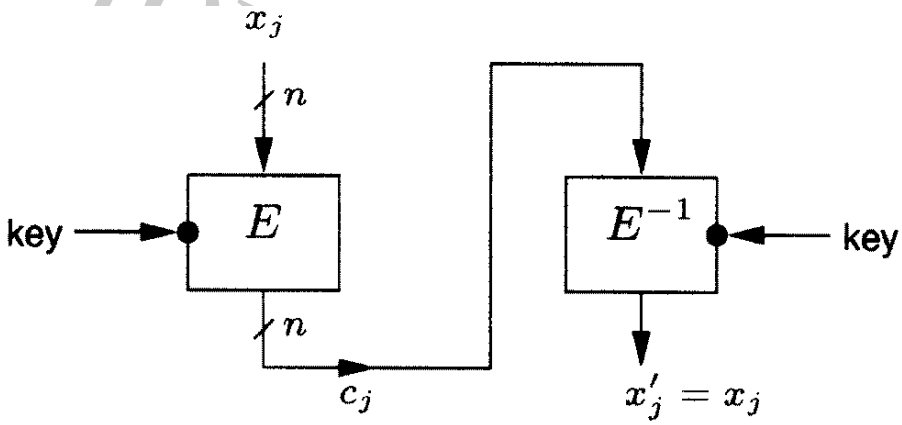
- Use of block ciphers for arbitrarily long messages
- Message x decomposed into blocks $x_1, \ldots, x_t$ of length n resp. r, i.e., $M = x_1 || \ldots || x_t$.
- Note: Padding is not only required for messages with message length $\neq$ integer multiple of block length, it is an essential part of the security system!
- Many flaws are introduced by improper padding!

Electronic Codebook Mode (ECB)

Each block is encrypted with the key to produce the corresponding ciphertext block.

Properties:

- Identical blocks result in identical ciphertext
- Chaining dependencies: blocks are encrypted independently
- No error propagation



Source: Handbook of Cryptography

Electronic Codebook Mode (ECB) (2)

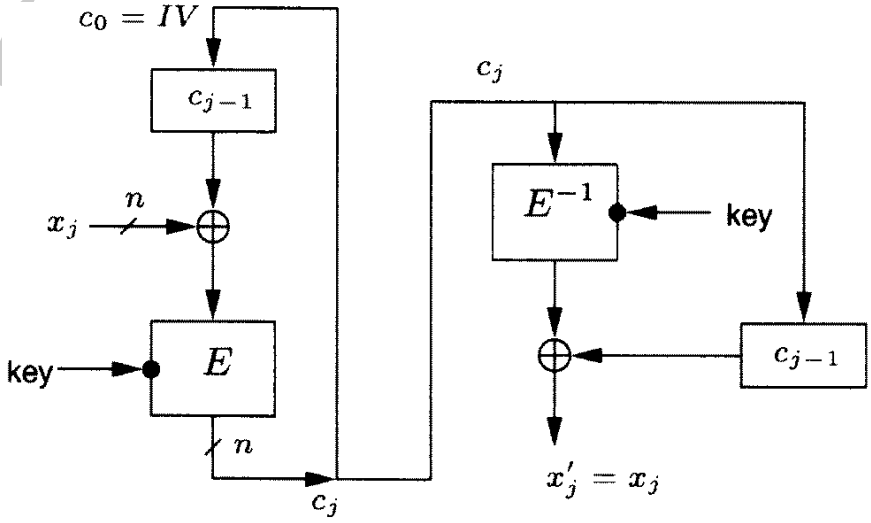
$$C_j = \mathrm{Enc}_K(X_j) \qquad j = 1, \ldots, t \qquad \text{(Encryption)}$$
$$X_j = \mathrm{Dec}_K(C_j) \qquad j = 1, \ldots, t \qquad \text{(Decryption)}$$

- Insertion attacks possible
- Multiple blocks leak information (data patterns)
- Requires padding
- Not recommended for messages longer than one block

Cipherblock Chaining Mode (CBC)

Properties:

- Identical plaintexts result in same ciphertexts if same key and IV is used.
- Chaining dependencies: ciphertext C_j depends on X_j and all preceding blocks



Source: Handbook of Cryptography

Cipherblock Chaining Mode (CBC) (2)

Encryption:

$$C_0 = IV \tag{5a}$$
$$C_j = \mathrm{Enc}_K(C_{j-1} \oplus X_j) \qquad j = 1, \ldots, t \tag{5b}$$

Decryption:

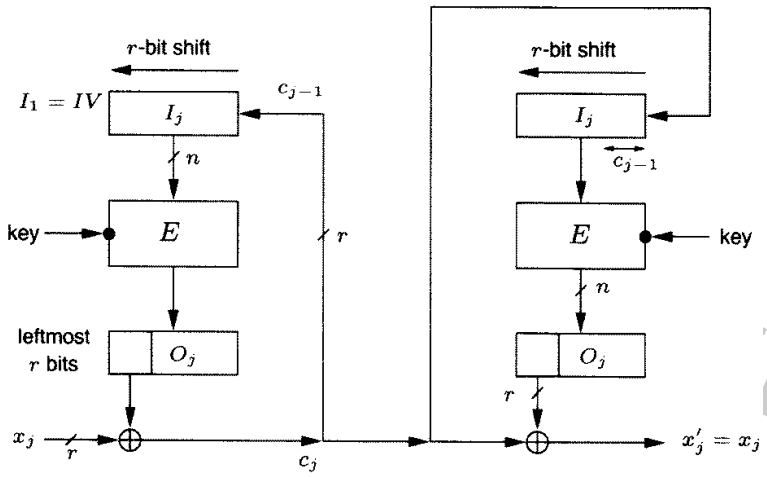
$$C_0 = IV \tag{6a}$$
$$X_j = \mathrm{Dec}_K(C_j) \oplus C_{j-1} \quad j = t, \ldots, 1 \tag{6b}$$

- Workhorse of cryptography
- IV random, authentic for encryption
- Vulnerable to bit flipping attacks
- Main fields: CBC-encryption and CBC-MAC

Cipher Feedback Mode (CFB)

Properties:

- Identical plaintexts result in same ciphertexts if same key and IV is used.
- Chaining dependencies: ciphertext C_j depends on X_j and all preceding blocks
- Throughput is decreased: each execution yields only r bits of ciphertext

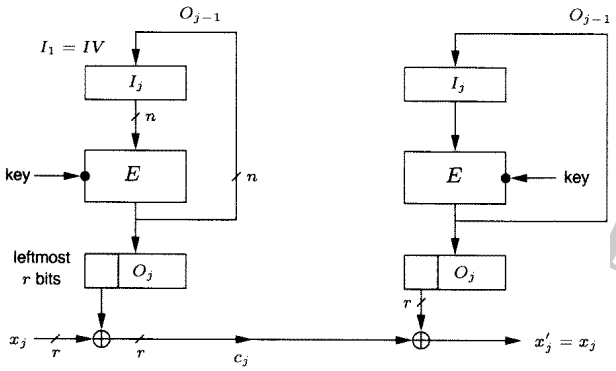


Source: Handbook of Cryptography

Output Feedback Mode (OFB)

Properties:

- Identical plaintexts result in same ciphertexts if same key and IV is used.
- Chaining dependencies: plaintext-independent preceding blocks
- Error propagation and recovery: one or more bit errors in C_j affect only its decryption. Thus, system recovers from ciphertext bit errors
- Throughput is decreased



Source: Handbook of Cryptography

Counter Mode (CTR)

- DES mode of operations: ECB, CBC, OFB, CFB; new for AES: CTR

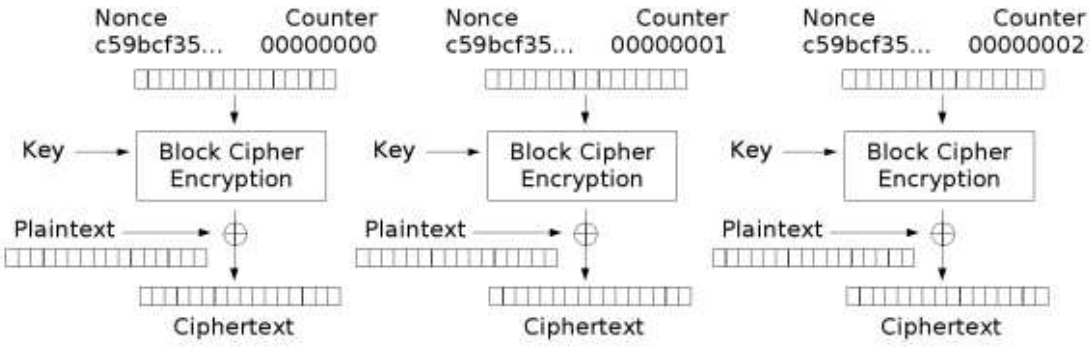
Let $T_j = \text{Nonce}||\text{Counter}_j, j = 1, \ldots, t$.
CTR-Encryption:

$$O_j = \text{Enc}_K(T_j) \qquad j = 1, \ldots, t$$
$$C_j = X_j \oplus O_j \qquad j = 1, \ldots, t-1$$
$$C_t = X_t \oplus MSB_r(O_t)$$

CTR-Decryption:

$$O_j = \text{Enc}_K(T_j) \qquad j = 1, \ldots, t$$
$$X_j = C_j \oplus O_j \qquad j = 1, \ldots, t-1$$
$$X_t = C_t \oplus MSB_r(O_t)$$

Counter Mode (CTR) (2)



Counter (CTR) mode encryption

Source: Wikipedia

Properties:

- Identical plaintexts result in same ciphertexts if same key and nonce is used.
- No chaining, fully parallelizable, requires only encryption primitive
- Error propagation and recovery: one or more bit errors in C_j affect only its decryption. Thus, system recovers from ciphertext bit errors.

Further Modes of Operations

- Other modes are under discussion, e.g., CCM and GCM as NIST Special Publications. These modes mostly combine confidentiality and authentication in one or two passes.
- CTR mode encryption replaces CBC encryption in many applications
- CCM: NIST SP 800-38C, 2004
  - provides authenticated encryption
  - CBC-MAC + CTR mode encryption
- GCM: Galois/Counter Mode, NIST SP 800-38D, 2007
  - provides authenticated encryption
  - universal hashing by $GF(2^{128})$ multiplication + CTR mode encryption
- There are many more authenticated encryption schemes, many patented!
- XTS: IEEE P1619 2007 – tweakable encryption mode for disk encryption, requires encryption of tweak and $GF(2^{128})$ multiplication

Padding Options for Encryption Modes

The following padding options are often used for ECB and CBC mode:

- No padding: only possible if length of plaintext is a multiple of block length
- ANSI X9.23/ISO 10126-2 padding: fill with null bytes or random bytes, last padded byte contains number of padding bytes
- PKCS#5/IETF RFC 1423 padding: contents of padding bytes = number of padding bytes

If expansion of ciphertext blocks is not desired, Ciphertext Stealing can be used: process last two plaintext blocks separately, pad the last plaintext block with the low order bits from the second to last ciphertext block.

Part 1: Outline — Section 4: Cryptographic Hash Functions

- 1. Cryptographic Hash Functions
  - Definitions
  - Random Oracle Model
  - Problems (collision, 2nd preimage, preimage resistance)
- 2. Iterated Hash Functions
  - Structure
  - SHA-0, SHA-1
  - Attacks

1. Cryptographic Hash Functions (1)

- Related to conventional hash functions.
- Used for data integrity and message authentication.
- Take a message of variable length as input, produce an output of fixed length referred to as hash-code, hash-result, hash-value or simply hash.
- Basic idea of cryptographic hash functions is that hash-value serves as compact representative image (called imprint, digital fingerprint, message digest) of an input string.
- Main applications of hash functions
  - data integrity, message authentication, signatures: protect only hash-value instead of entire message; break homomorphic property of signature schemes
  - design of MAC schemes: distinct calls of hash functions that allows message authentication by symmetric techniques. Takes two inputs: message and secret key.
  - pseudo-random number generation
  - key management: key derivation functions

Cryptographic Hash Functions (2)

Definition 4.1.

A hash function is a function $h : \mathcal{X} \to \mathcal{Y}$

1. $\mathcal{X}$ is the set of possible messages
2. $\mathcal{Y}$ is a finite set of possible message digests or authentication tags.

Remark 4.2.

$\mathcal{X}$ can be finite or infinite. We always assume $|\mathcal{X}| \geq |\mathcal{Y}|$, and often even $|\mathcal{X}| \gg |\mathcal{Y}|$.

Example 4.3.

MD5: input size $< 2^{64}$, output size 128
SHA-1, RIPEMD-160: input size $< 2^{64}$, output size 160
SHA-224, SHA-256: input size $< 2^{64}$, output size $224, 256$
SHA-384, SHA-512: input size $< 2^{128}$, output size $384, 512$

Cryptographic Hash Functions (3)

Definition 4.4.

A family of keyed hash functions is a 4-tuple $\mathcal{X}, \mathcal{Y}, \mathcal{K}, \mathcal{H}$ where:

1. $\mathcal{X}$ is a set of possible messages
2. $\mathcal{Y}$ is a finite set of possible message digests
3. $\mathcal{K}$ is a finite set of possible keys
4. For each $K \in \mathcal{K}$, there is a hash function $h_K : \mathcal{X} \to \mathcal{Y}$.

Remark 4.5.

- If $\mathcal{X}$ is finite, a hash function is sometimes called a compression function.
- The pair $(x, y) \in \mathcal{X} \times \mathcal{Y}$ is valid under the key K, if $h_K(x) = y$.
- Let $\mathcal{F}^{\mathcal{X} \times \mathcal{Y}}$ denote the set of all functions from $\mathcal{X}$ to $\mathcal{Y}$. If $|\mathcal{X}| = N$ and $|\mathcal{Y}| = M$, then $|\mathcal{F}^{\mathcal{X} \times \mathcal{Y}}| = M^N$. Any hash family $\mathcal{F}$ is a subset of $\mathcal{F}^{\mathcal{X} \times \mathcal{Y}}$, and is called an (N, M)-hash family.
- An unkeyed hash function can be thought of as a hash family with only one possible key, i.e., $|\mathcal{K}| = 1$.

Cryptographic Hash Functions (4)

Desirable properties:

1. compression: h maps an input x of arbitrary bit length, to an output $h(x)$ of fixed bit length.
2. ease of computation: given h and an input x, $h(x)$ is easy to compute.
3. pre-image resistance (also called one-wayness): given a message digest z, it is computationally infeasible to find a message x such that $h(x) = z$.
4. second pre-image resistance: given a message x, it is computationally infeasible to find a message $x' \neq x$ such that $h(x) = h(x')$.
5. collision resistance: it is computationally infeasible to find any messages $x' \neq x$ such that $h(x) = h(x')$.
6. randomness: given an element x chosen uniformly at random from $\mathcal{X}$, $h(x)$ should be uniformly at random element of $\mathcal{Y}$.

Random Oracle Model

- Idealized model of a hash function that attempts to capture the concept of an “ideal” hash function.
- Mathematically captures desired randomness properties as well as intuition that ideally the only efficient way to compute $h(x)$ is to evaluate h at x, even if some hash values are already known.
- In the random oracle model, $h : \mathcal{X} \to \mathcal{Y}$ is chosen randomly from $\mathcal{F}^{\mathcal{X}\times\mathcal{Y}}$. Only oracle access to the function h is permitted.

Theorem 4.6.

Suppose that $h \in \mathcal{F}^{\mathcal{X}\times\mathcal{Y}}$ is chosen randomly, and let $\mathcal{X}_0 \subseteq \mathcal{X}$. Suppose that the values $h(x)$ have been determined (by querying the oracle for h) if and only if $x \in \mathcal{X}_0$. Then, $\Pr[h(x) = y] = \frac{1}{M}$ for all $x \in \mathcal{X}\backslash\mathcal{X}_0$ and all $y \in \mathcal{Y}$.

Random Oracle Methodology

Analyze algorithms using hash functions in the random oracle model — that is, as if $h \in \mathcal{F}^{\mathcal{X}\times\mathcal{Y}}$ is chosen randomly and only oracle access to h is allowed.

Remark 4.7.

This is an invalid assumption since h is fixed. But:

- it provides for significantly simpler analysis
- it is good for screening out insecure solutions
- security under random oracle model implies security to many attacks, assuming ideal h is replaced with a suitable h.
- not a complete proof of security, but better than no proof of security.

Warning: Recent results have shown (contrived) cases in which a cryptographic algorithm using a hash function is secure in the random oracle model, but is not secure for any actual hash function. So far, these results have only been shown for contrived cases, not for natural ones.

Problems

1. **Pre-image resistance problem:** given a hash function $h : \mathcal{X} \to \mathcal{Y}$ and an element $y \in \mathcal{Y}$, find a message x such that $h(x) = y$.

2. **Second pre-image problem:** given a hash function $h : \mathcal{X} \to \mathcal{Y}$ and an element $x \in \mathcal{X}$, find a message $x' \neq x$ such that $h(x) = h(x')$.

3. **Collision problem:** given a hash function $h : \mathcal{X} \to \mathcal{Y}$ find messages $x' \neq x$ such that $h(x) = h(x')$.

Randomized Algorithms

We “allow” the adversary to use randomized algorithms, since it is important to understand the existence of a threat even if the adversary has only some probability smaller than 1 of succeeding.

- Las Vegas algorithms: May fail to give an answer, but if it returns answer, it is always correct.
- yes-biased Monte Carlo algorithms: a “yes” answer is always correct, a “no” answer may be incorrect[8]
- worst-case success probability ϵ: if for every problem instance, the algorithm returns a correct answer with probability at least ϵ.
- average-case success probability ϵ: if the probability that the algorithm returns a correct answer, averaged over all problem instances of a specified size, is at least ϵ.
- We say an algorithm is an $(\epsilon; q)$-algorithm if it is a Las Vegas algorithm with average-case success probability ϵ in which the number of oracle queries is at most q.

[8] see, e. g., Miller Rabin test for prime number generation

Problems: Pre-Image

Algorithm FINDPREIMAGE(h, y, q)
choose any $\mathcal{X}_0 \subseteq \mathcal{X}$, $|\mathcal{X}_0| = q$
for each $x \in \mathcal{X}_0$
  do if $h(x) = y$ then return (x)
return ($failure$)

Theorem 4.8.
For any $\mathcal{X}_0 \subseteq \mathcal{X}$ with $|\mathcal{X}_0| = q$, $|\mathcal{Y}| = M$, the average-case success probability of Algorithm FINDPREIMAGE is $\epsilon = 1 - (1 - 1/M)^q$.

Proof: see textbook

Problems: Collisions

Algorithm FINDCOLLISION(h, q)
choose $\mathcal{X}_0 \subseteq \mathcal{X}$, $|\mathcal{X}_0| = q$
for each $x \in \mathcal{X}_0$
  do $y_x := h(x)$
if $y_x = y_{x'}$ for some $x' \neq x$
  then return (x, x')
  else return $(failure)$

Theorem 4.9.
FINDCOLLISION is an $(0.5, \mathcal{O}(\sqrt{M})$-algorithm!

Birthday Paradox

FINDCOLLISION(h, q) benefits from birthday paradox:

In a group of 23 random people, at least two will share a birthday with probability at least $1/2$.

- Suppose there are n birthdays and k people in the room. An elementary event is a tuple $(b_1, \ldots b_k) \in \{1, 2, \ldots, n\}^k$. I.e., the birthday of the i-th person is b_i. Thus, there are n^k elementary events. We assume they are equally probable.
- Let q denote the probability that any two people have different birthdays.
- Let E be the number of vectors with pairwise different entries.
- Hence $E = \prod_{i=0}^{k-1}(n-i)$ and $q = \frac{1}{n^k}\prod_{i=0}^{k-1}(n-i) = \prod_{i=0}^{k-1}(1-\frac{i}{n})$.

Birthday Paradox (2)

- Since $1 + x \leq e^x$ for all real numbers x, we obtain

$$q \leq \prod_{i=0}^{k-1} e^{-i/n} = e^{-\sum_{i=0}^{k-1} i/n} = e^{-k(k-1)/(2n)}.$$

If

$$k \geq (1 + \sqrt{1 + 8n \ln 2})/2,$$

then $q \leq 0.5$. For $n = 365$, $k = 23$.

Remark 4.10.

For $q = 0.5$, $k \approx 1.17\sqrt{n}$. Thus, for a hash function h with $h : \mathcal{X} \to \mathcal{Y}$ and $|\mathcal{Y}| = n$, hashing just over $\sqrt{n}$ random elements of $\mathcal{X}$ yields a collision with probability of 50%.
$\Rightarrow$ imposes lower bound on the sizes of message digests. Today, 160-bit hash functions are widely used. They should be replaced by larger sized hash functions (SHA-256, etc.) by 2010 according to NIST and BSI regulations.

Reductions

- The collision problem can be reduced (in a complexity theoretic sense) to the pre-image problem.
- The collision problem can be reduced to the second pre-image problem.
- While this seems to be trivial for fixed inputs, it is non-trivial in the random oracle model (randomized algorithms).
- Thus, collision resistance is the most demanding security property! The property of collision resistance implies the properties of preimage and second preimage resistance.

2. Iterated Hash Functions

Idea: Turn fixed input length hash functions over a finite domain into variable input length hash functions over an infinite domain

Preprocessing step: Pad input string x to multiple of block length

$$y = x||\text{PAD}(x)$$
$$y = y_1||y_2||\ldots||y_r \qquad y_i \text{ bitstring of length } t$$

Processing step:

$$z_0 = IV \qquad \text{bitstring of length m}$$
$$z_i = \text{COMPRESS}(z_{i-1}||y_i) \qquad i = 1, \ldots, r$$

compression function COMPRESS: $\{0,1\}^{m+t} \to \{0,1\}^m$
Output transformation: $h(x) = g(z_t)$ with output transformation g

Theory of Iterated Hash Functions

- There is a theoretical construction (landmark paper by I. Damgård [1989]) that shows how to construct a collision-resistant hash function from a collision-resistant compression function (=careful, but impractical construction of padding).
- Two similar concepts:
  - Collision resistant compression function (fixed length) $\longrightarrow$ Merkle-Damgård construction $\longrightarrow$ collision resistant hash function (variable length)
  - Cf. Luby-Rackoff Theorem: Pseudo Random Function $f : \{0,1\}^n \to \{0,1\}^n$ $\longrightarrow$ three round Feistel construction $L_i = R_{i-1}, R_i = L_{i-1} \oplus f(R_{i_1}, K_i) \longrightarrow$ Pseudo Random Permutation
- Most practical hash functions today use the Merkle-Damgård construction with simple padding.
- New construction ideas due to hash attacks are underway (e.g., sponge construction) or are being reconsidered (hash functions from block ciphers, e.g., Miyaguchi-Preneel).

The Iterated Hash Functions SHA-0 and SHA-1

1. Padding of the message by appending 1, and a corresponding number of 0, as well as the length of the message encoded as 64-bit integer

$$\Longrightarrow x = x_1 x_2 \ldots x_r \qquad x_i - 512\text{-bit blocks}, i = 1, \ldots, r$$

2. Initialize five 32-bit registers A, B, C, D, E with fixed constants

$$A = 0x67452301, B = 0xEFCDAB89, \ldots \qquad H_0 = IV$$

3. For every message block
   - Copy $A, \ldots, E$ to $AA, \ldots, EE$
   - Apply compression function f to $AA, \ldots, EE$ and message block
   - Add results $AA', \ldots, EE'$ to $A, \ldots, E$

$$H_i = f(H_{i-1}, x_i) \quad i = 1, \ldots, r, \qquad f : (160 \text{ bit}, 512 \text{ bit}) \mapsto 160 \text{ bit}$$

4. Result $A||B||C||D||E$ $\qquad H = H_r$ (160 bit)

The Iterated Hash Functions SHA-0 and SHA-1 (2)

The compression function of SHA-0 and SHA-1

1. Split the 512-bit message block into 16 32-bit words $W_0, W_1, \ldots, W_{15}$
2. Expansion into 80 32-bit words

$$W_i = (W_{i-3} \oplus W_{i-8} \oplus W_{i-14} \oplus W_{i-1}) \underbrace{\lll 1}_{\text{only SHA-1}} \qquad i = 16, \ldots, 79$$

3. Copy $A, \ldots, E$ to $AA, \ldots, EE$ ($A_0 = A, \ldots, E_0 = E$)
4. for $i = 0$ to 79 do

$$\begin{aligned} A_{i+1} &= A_i \lll 5 + f_i(B_i, C_i, D_i) + W_i + K_i \mod 2^{32} \\ B_{i+1} &= A_i \\ C_{i+1} &= B_i \lll 30 \\ D_{i+1} &= C_i \\ E_{i+1} &= D_i \end{aligned}$$

The Iterated Hash Functions SHA-0 and SHA-1 (3)

5. $A_0 = A_{80} + AA \mod 2^{32}, \ldots, E_0 = E_{80} + EE \mod 2^{32}$
6. Round functions $f_i(X, Y, Z)$ and constants K_i

round	function f_i		constant
i	name	definition	K_i
$0 - 19$	IF	$(X \wedge Y) \vee (\neg X \wedge Z)$	$0x5A827999$
$20 - 39$	XOR	$X \oplus Y \oplus Z$	$0x6ED9EBA1$
$40 - 59$	MAJ	$(X \wedge Y) \vee (X \wedge Z) \vee (Y \wedge Z)$	$0x8F1BBCDC$
$60 - 79$	XOR	$X \oplus Y \oplus Z$	$0xCA62C1D6$

The status of hash functions after recent attacks

- den Boer/Bosselears, 1993: Pseudo Collisions in MD5
- Dobbertin, 1994: Collisions in MD4
- Dobbertin, 1996: Free start collisions in MD5
- Joux et al., 2004: Collisions in SHA-0 in 2^{51}
- Wang et al., 2004. Collisions in MD4, MD5, HAVAL, RIPEMD
- Wang et al., 2005: Collisions in SHA-0 in 2^{39}
- Wang et al., 2005: theoretical collisions in SHA-1 in 2^{63}
- de Canniere, Rechberger, 2006: structured collisions for 64 round SHA-1

- $\Longrightarrow$ MD5 is cryptographically dead
- $\Longrightarrow$ no practical collision for SHA-1 yet, but expected very soon
- $\Longrightarrow$ no realistic attacks for RIPEMD-160 yet, but 160 bit are too small
- $\Longrightarrow$ use SHA-224, SHA-256, SHA-384, SHA-512 (similar design as SHA-1)
- $\Longrightarrow$ wait for results of NIST hash competition, expected after 2012

Overview practical hash functions

Algorithm	Output size (bits)	Number of rounds	Internal state (bits)	Block size (bits)	Security
MD4	128	3×16	128	512	insecure
MD5	128	4×16	128	512	insecure
SHA-1	160	4×20	160	512	almost insecure
RIPEMD-160	160	5×16	160	512	secure till 2010
SHA-256/SHA-224	256/224	64	256	512	secure
SHA-512/SHA-384	512/384	80	512	1024	secure
Whirlpool	512	10	512	512	NESSIE, based on AES

Source: Wikipedia and Standards

Hash Functions — Implementation Aspects

- Hashes are mostly slower than block ciphers
- SHA-1 $\approx$ 58 cycles per output byte on Pentium 4 (over twice as slow as AES)
- RIPEMD-160 is approx. 15 percent slower than SHA-1. In addition, it has round dependent rotation, thus larger.
- IETF often uses HMAC constructions while smart cards use CBC-MAC for authentication and integrity protection
- Hashes are not MACs! Example: secret prefix

$$tag := hash(key||message)$$

  Attacker: chooses tag as new inital state, appends one arbitrary block + MD strengthening $\Longrightarrow$ valid tag
- $\Longrightarrow$ Use HMAC which does not leak inner state of hash function

Part 1: Outline — Section 5: Message Authentication Codes

1. Message Authentication Codes (MACs)
2. MAC constructions based on hash functions (Nested MACs, HMAC)
3. MAC constructions based on block ciphers (CBC-MAC)

1. Message Authentication Codes (MACs)

- To verify the authenticity of a message, append a “fingerprint” or “tag” to it that can be verified by the receiver and no one else can forge it.
- In the symmetric key mode this is called a message authentication code. Only a party possessing the key can compute and verify the authentication tag.

2. MACs from Hash Functions

- Prepending the key (secret prefix) is insecure because an adversary can append further blocks to a correctly authenticated message
- Appending the key (secret postfix) allows for a birthday attack
- Keying via IV has the analytical advantage that h_k and thus the MAC naturally represent families of functions, where k chooses a random element of the family.
- The secret prefix/suffix methods require no modifications to the hash functions.
- HMAC, which is a nested MAC, combines both advantages. It is a widely used keyed hash functions. It is mainly used in Internet standards as HMAC-SHA-1 and HMAC-MD5.
- HMAC construction:

$$\mathrm{HMAC}(K, m) = H((K_0 \oplus \mathrm{opad}) || H((K_0 \oplus \mathrm{ipad}) || m))$$

  K_0 padded key K,
  ipad=$363636\ldots$, opad=$5C5C\ldots$ inner and outer padding constants
- HMAC is standardized, e.g., in ISO/IEC 9797-2, 2002 and FIPS 198a, 2002.

3. CBC-based MACs (1)

Idea: Use block cipher in CBC mode with a fixed (public) initialization vector.

Algorithm CBC-MAC(x, K)
$x = x_1 || \ldots || x_n$ (n blocks of length t, where t is block size of cipher)
$\mathrm{IV} \leftarrow 0^t$
$y_0 = \mathrm{IV}$
for $i \leftarrow 1$ **to** n
do $y_i \leftarrow e_K(y_{i-1} \oplus x_i)$
return (y_n)

$y_1, \ldots, y_{n-1}$ must not be published!

CBC-based MACs (2)

Attack: Birthday collision attack = best general attack

- Adversary can request MACs on a large number of messages. If a duplicated MAC is found, then the adversary can construct an additional message and request its MAC. Furthermore, the adversary can produce a new message and corresponding MAC without knowing the secret key:
  1. $n \geq 2$, $x_3, \ldots, x_n$ fixed bit strings of length t, Let $q \approx 1.17 \times 2^{t/2}$. Choose q distinct bit strings $x_1^1, \ldots, x_1^q$ of length t.
  2. Let $x_2^1, \ldots, x_2^q$ be randomly chosen bit strings of length t. For $3 \leq l \leq n$ and $1 \leq i \leq q$, define $x_l^i = x_l$. Define $x^i = x_1^i || \ldots || x_n^i$ for $1 \leq i \leq q$.
  3. Adversary requests MACs of $x^1, \ldots, x^q$.
  4. $h_K(x^i) = h_K(x^j) \Leftrightarrow y_2^i = y_2^j \Leftrightarrow y_1^i \oplus x_2^i = y_1^j \oplus x_2^j$.
  5. If that occurs, let x_δ be any bit string of length t.
  6. Define $v = x_1^i || (x_2^i \oplus x_\delta) || \ldots || x_n^i$ and $w = x_1^j || (x_2^j \oplus x_\delta) || \ldots || x_n^j$.
  7. The adversary requests the MAC for v which is identical with the one for w.
- This constitutes an attack with success probability 0.5 and effort $\mathcal{O}(2^{t/2})$.

CBC-MAC in Practice: Padding Methods

The data string D to be input to the MAC algorithm shall be

1. right-padded with as few (possibly none) ‘0’ bits as necessary to obtain a data string whose length (in bits) is a positive integer multiple of t.
2. right-padded with a single ‘1’ bit. The resulting string shall then be right-padded with as few (possibly none) ‘0’ bits as necessary to obtain a data string whose length (in bits) is a positive integer multiple of t.
3. right-padded with as few (possibly none) ‘0’ bits as necessary to obtain a data string whose length (in bits) is a positive integer multiple of t. The resulting string shall then be left-padded with a block L (binary representation of the length).

CBC-MAC in Practice: Postprocessing — MAC variants

- In practice, the last block y_n is sent to an output transformation and truncated (optional):
  1. transformation 1: transformation = identity transformation, $\mathrm{MAC} := y_n$
  2. transformation 2: encrypt last block with different key K', $\mathrm{MAC} = \mathrm{Enc}_{K'}(y_n)$
  3. transformation 3: decrypt last block with different key K', and encrypt with key K, $\mathrm{MAC} = \mathrm{Enc}_K(Dec_{K'}(y_n))$
  4. truncation: take the m leftmost bits of MAC
- Above MAC constructions are standardized in ISO/IEC 9797-2-2002.
- CBC-MAC alone is only secure for fixed message length and not secure unless the message length is fixed.
- CBC-MAC with transformation 2 is known as Encrypted MAC (EMAC). Petrank/Rackoff proved the security of EMAC if the message length is a positive multiple of n. In other cases, pad with padding method 2.
- CBC-MAC with transformation 3, padding method 2 and truncation of $m = \frac{t}{2}$ bits is known as Retail-MAC (banking sector).

CBC-MAC in Practice: CMAC

CMAC or OMAC1 (One-Key MAC) is a new MAC, secure for variable length messages. It requires only one key, is standardized by NIST SP 800-38b, 2005 and doesn't introduce a new block by padding.

Algorithm Subkey-Generation for CMAC

Input: Key K, Output: Subkeys K_1, K_2 of length b ($b = 128$ for AES)

S0: $R_{128} = 0x0 \ldots 087 = 0^{120}10000111$;

S1: $L := \mathrm{Enc}_K(0^b)$;

S2: **if** (msb(L) = 0) **then** $K_1 := (L << 1)$ **else** $K_1 := (L << 1) \oplus R_b$;

S3: **if** (msb(K_1) = 0) **then** $K_2 := (K_1 << 1)$ **else** $K_2 := (K_1 << 1) \oplus R_b$;

CBC-MAC in Practice: CMAC (2)

Algorithm $\mathrm{CMAC}_K(x)$

1. Apply the subkey procedure to K to produce K_1 and K_2
2. Let $x_1, \ldots, x_{n-1}, x'_n$ be the sequence of message blocks, where $x_1, \ldots, x_{n-1}$ are complete blocks, i.e., $|x_i| = b = 128$, $i = 1, \ldots, n-1$.
3. If x'_n is a complete block, i.e., $|x'_n| = b$, then $x_n := K_1 \oplus x'_n$;
   else $x_n := K_2 \oplus (x'_n \,||\, 10^j)$ with $j = b - |x'_n| - 1$.
4.

$$
\begin{aligned}
H_0 &:= IV = 0^{128} && \text{Initialization} \\
H_i &:= \mathrm{Enc}_K(H_{i-1} \oplus x_i) \quad i = 1, \ldots, n && \text{CBC mode} \\
\mathrm{CMAC}_K(x) &:= \mathrm{MSB}_{127\ldots64}(H_n) && \text{Output transformation}
\end{aligned}
$$