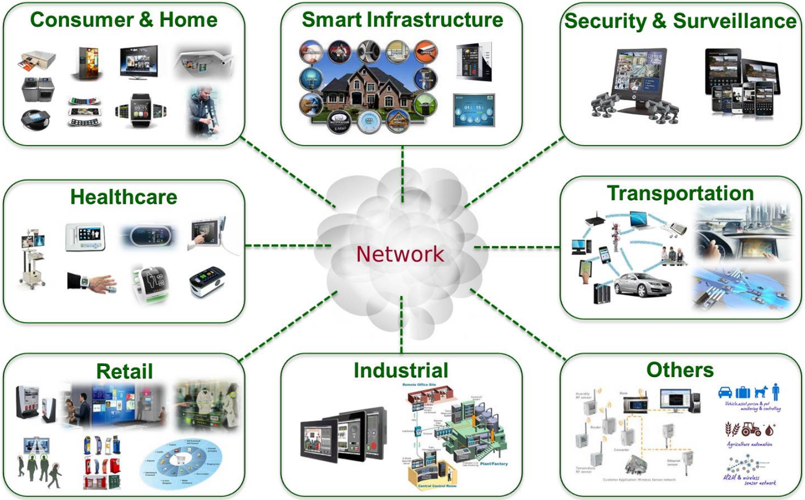
Security of IoT

Pascal Lafourcade

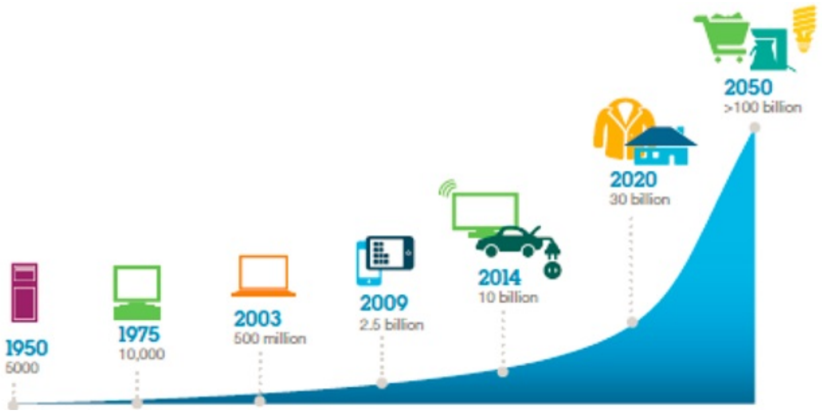
October 2020

Internet of Thing (IoT)

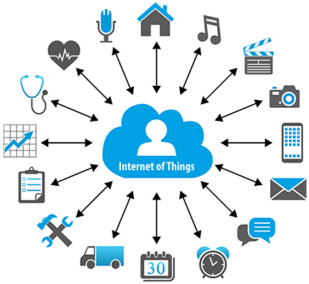


Vivante and the Vivante logo are trademarks of Vivante Corporation. All other product, image or service names in this presentation are the property of their respective owners. © 2013 Vivante Corporation

Increasing Succes of IoT



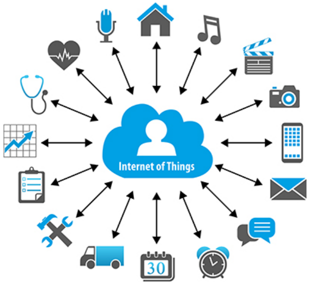
Reasons of the Succes of IOT



Technology

- Wireless Communications:
  Wifi, 3G, 4G, Bluethooth, Sigfox ...
- Batteries
- CPU
- Sensors
- Price

Reasons of the Succes of IOT



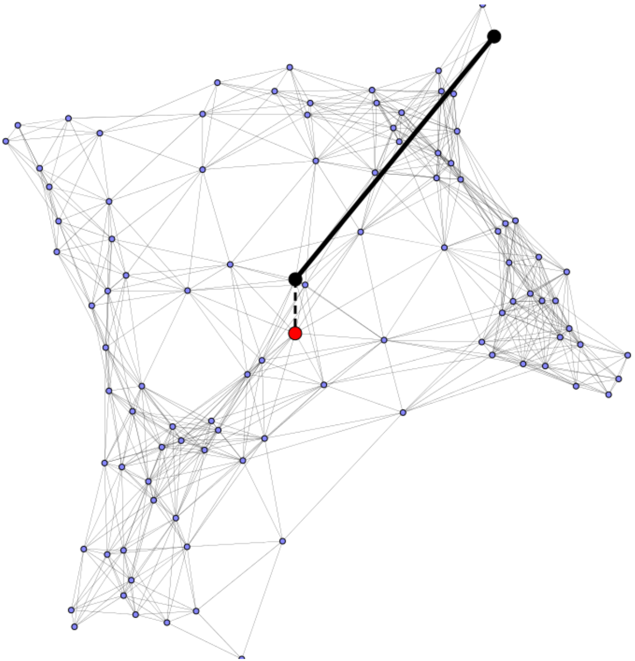
Technology

- Wireless Communications:
  Wifi, 3G, 4G, Bluethooth, Sigfox ...
- Batteries
- CPU
- Sensors
- Price

Usage

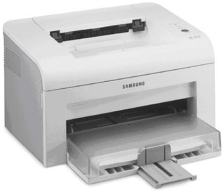
- Monitoring services
- Hyperconnectivity
- Avaibility

Wireless communications ⇒ Wormhole Attack

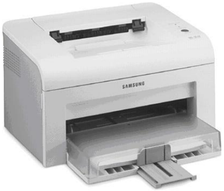


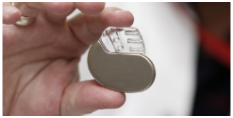
Real attacks on IoT from 2007 ...

Real attacks on IoT from 2007 ...

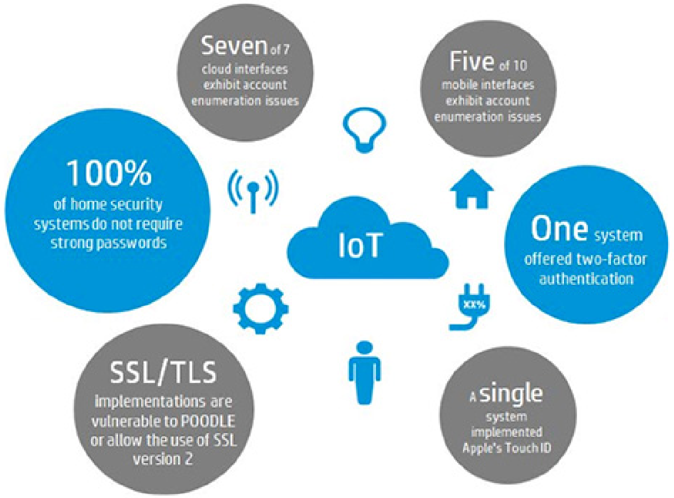


Real attacks on IoT from 2007 ...





Insecurity of IoT by HP in 2015


Seven of 7 cloud interfaces exhibit account enumeration issues
Five of 10 mobile interfaces exhibit account enumeration issues
100% of home security systems do not require strong passwords
IoT
One system offered two-factor authentication
XX%
SSL/TLS implementations are vulnerable to POODLE or allow the use of SSL version 2
A single system implemented Apple's Touch ID

POODLE: Padding Oracle On Downgraded Legacy Encryption

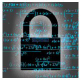
TOP 10: Vulnerabilities of IoT

1. Insecure Web Interface (weak passwords, account protection)
2. Unsufficient Authtneitcation/Authorization
3. Insecure Newtork Services (ports open, DoS)
4. Lack of Transport Encryption
5. Privacy Concerns (leak of personal informations)
6. Insecure Cloud interfaces
7. Insecure Mobile Interfaces
8. Insufficient Security Configurability
9. Insecure Software/Firmeware
10. Poor Physical Security

`https://www.owasp.org/images/8/8e/Infographic-v1.jpg`

How to Secure IoT

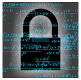
Cryptography:



- Primitives: RSA, Elgamal, AES, DES, SHA-3 ...
- Protocols: Distributed Algorithms

How to Secure IoT

Cryptography:



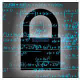
- Primitives: RSA, Elgamal, AES, DES, SHA-3 ...
- Protocols: Distributed Algorithms

Properties:

- Secrecy,
- Authentication,
- Privacy
- Non Repudiation ...

How to Secure IoT

Cryptography:



- Primitives: RSA, Elgamal, AES, DES, SHA-3 ...
- Protocols: Distributed Algorithms

Properties:

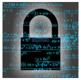
- Secrecy,
- Authentication,
- Privacy
- Non Repudiation ...

Intruders:

- Passive, active
- CPA, CCA ...

How to Secure IoT

Cryptography:



- Primitives: RSA, Elgamal, AES, DES, SHA-3 ...
- Protocols: Distributed Algorithms

Properties:

- Secrecy,
- Authentication,
- Privacy
- Non Repudiation ...

Intruders:

- Passive, active
- CPA, CCA ...

Designing such **secure** protocols is **difficult**

Is it preserving your privacy?

Is it preserving your privacy?

4096 RSA encryption

Is it preserving your privacy?

4096 RSA encryption

Environs 60 températures possibles: 35 ... 41

Is it preserving your privacy?

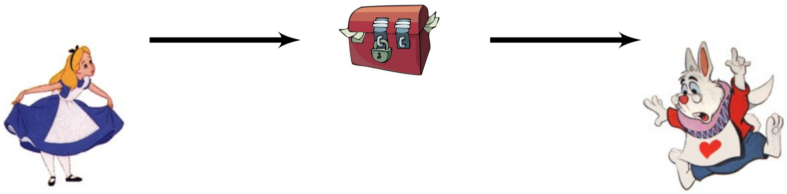
4096 RSA encryption

Environs 60 températures possibles: 35 ... 41

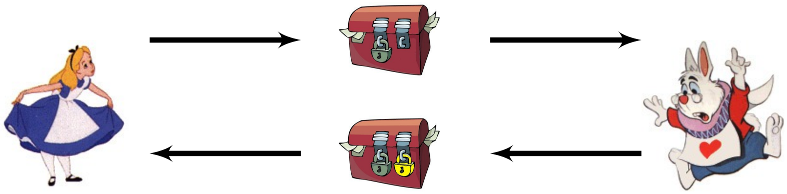
$$\{35\}_{pk}, \{35,1\}_{pk}, \ldots, \{41\}_{pk}$$

3-Pass Shamir

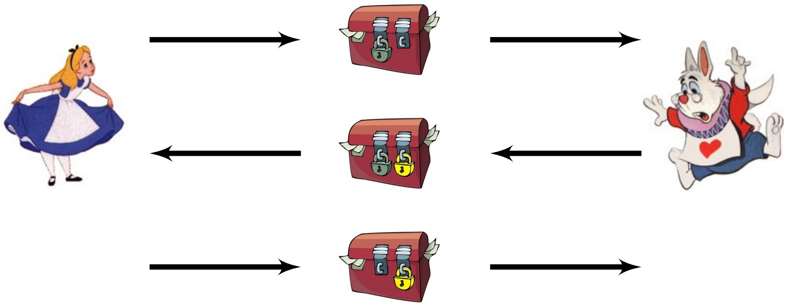
3-Pass Shamir



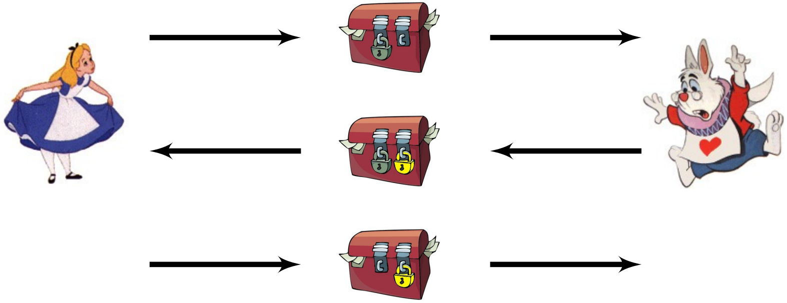
3-Pass Shamir



3-Pass Shamir



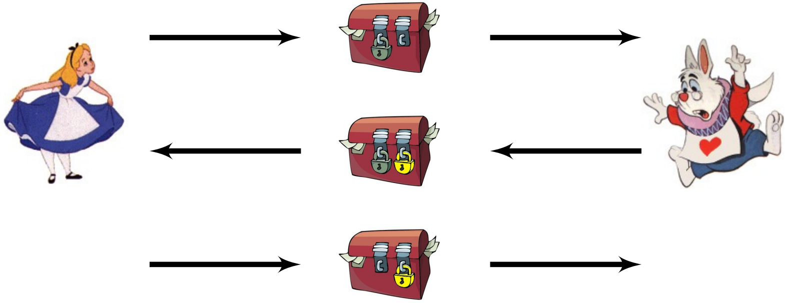
3-Pass Shamir



Abstract Representation

$$1 \quad A \quad \rightarrow \quad B \quad : \quad \{m\}_{K_A}$$

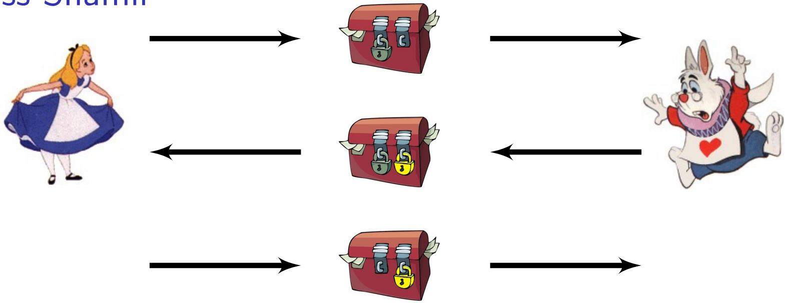
3-Pass Shamir



Abstract Representation

1	A	$\rightarrow$	B	:	$\{m\}_{K_A}$
2	B	$\rightarrow$	A	:	$\{\{m\}_{K_A}\}_{K_B}$

3-Pass Shamir

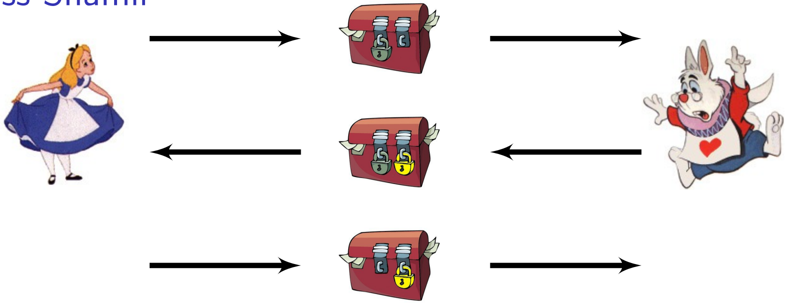


Abstract Representation

$$\begin{array}{lllll} 1 & A & \rightarrow & B & : \{m\}_{K_A} \\ 2 & B & \rightarrow & A & : \{\{m\}_{K_A}\}_{K_B} = \{\{m\}_{K_B}\}_{K_A} \end{array}$$

Commutative
Encryption

3-Pass Shamir



Abstract Representation

$$\begin{array}{lllll} 1 & A & \rightarrow & B & : \{m\}_{K_A} \\ 2 & B & \rightarrow & A & : \{\{m\}_{K_A}\}_{K_B} = \{\{m\}_{K_B}\}_{K_A} \\ 3 & A & \rightarrow & B & : \{m\}_{K_B} \end{array}$$

Commutative Encryption

Logical Attack on Shamir 3-Pass Protocol (I)

Perfect encryption one-time pad (Vernam Encryption)

$\{m\}_k = m \oplus k$

XOR Properties (ACUN)

- $(x \oplus y) \oplus z = x \oplus (y \oplus z)$ **A**ssociativity
- $x \oplus y = y \oplus x$ **C**ommutativity
- $x \oplus 0 = x$ **U**nity
- $x \oplus x = 0$ **N**ilpotency

Logical Attack on Shamir 3-Pass Protocol (I)

Perfect encryption one-time pad (Vernam Encryption)

$\{m\}_k = m \oplus k$

XOR Properties (ACUN)

$(x \oplus y) \oplus z = x \oplus (y \oplus z)$	**A**ssociativity
$x \oplus y = y \oplus x$	**C**ommutativity
$x \oplus 0 = x$	**U**nity
$x \oplus x = 0$	**N**ilpotency

Vernam encryption is a commutative encryption :

$$\{\{m\}_{K_A}\}_{K_I} = (m \oplus K_A) \oplus K_I = (m \oplus K_I) \oplus K_A = \{\{m\}_{K_I}\}_{K_A}$$

Logical Attack on Shamir 3-Pass Protocol (II)

Perfect encryption one-time pad (Vernam Encryption)

$\{m\}_k = m \oplus k$

Shamir 3-Pass Protocol

$$\begin{array}{lllll} 1 & A & \rightarrow & B: & m \oplus K_A \\ 2 & B & \rightarrow & A: & (m \oplus K_A) \oplus K_B \\ 3 & A & \rightarrow & B: & m \oplus K_B \end{array}$$

Logical Attack on Shamir 3-Pass Protocol (II)

Perfect encryption one-time pad (Vernam Encryption)

$\{m\}_k = m \oplus k$

Shamir 3-Pass Protocol

$$\begin{array}{lllll} 1 & A & \rightarrow & B: & m \oplus K_A \\ 2 & B & \rightarrow & A: & (m \oplus K_A) \oplus K_B \\ 3 & A & \rightarrow & B: & m \oplus K_B \end{array}$$

Second Example

Needham Schroeder Key Echange 1976

$$A \rightarrow B : \{A, N_A\}_{Pub(B)}$$
$$B \rightarrow A : \{N_A, N_B\}_{Pub(A)}$$
$$A \rightarrow B : \{N_B\}_{Pub(B)}$$

- Use cryptography
- Small programs
- Distributed

Cryptography is not sufficient !

Example : Needham Schroeder Key Echange

$$A \rightarrow B : \{A, N_A\}_{Pub(B)}$$

$$B \rightarrow A : \{N_A, N_B\}_{Pub(A)}$$

$$A \rightarrow B : \{N_B\}_{Pub(B)}$$

Cryptography is not sufficient !

Example : Needham Schroeder Key Echange

$$A \to B : \{A, N_A\}_{Pub(B)}$$

$$B \to A : \{N_A, N_B\}_{Pub(A)}$$

$$A \to B : \{N_B\}_{Pub(B)}$$

Broken 17 years after, by G. Lowe

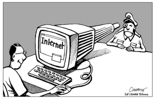
$$\begin{array}{ll}
A \to I : \{A, N_A\}_{Pub(I)} & I \to B : \{A, N_A\}_{Pub(B)} \\
A \leftarrow I : \{N_A, N_B\}_{Pub(A)} & I \leftarrow B : \{N_A, N_B\}_{Pub(A)} \\
A \to I : \{N_B\}_{Pub(I)} & I \to B : \{N_B\}_{Pub(B)}
\end{array}$$

Security Challenges for IoT

Data exchanged should be protected.

Security Properties

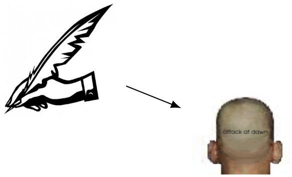
- Data Integrity
- Data Confidentiality
- Data Privacy
- Authentication
- Non-repudiation
- Avaibility



Outline

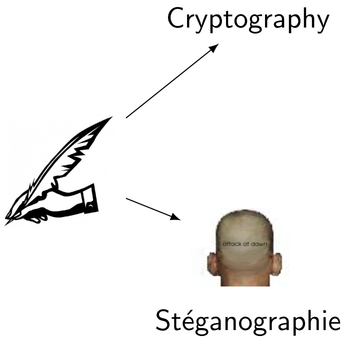
Information hiding

Information hiding

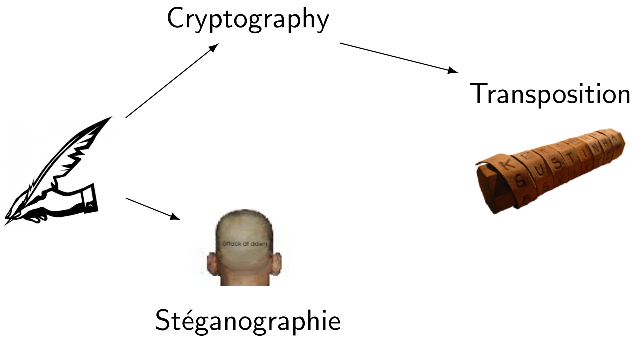


Stéganographie

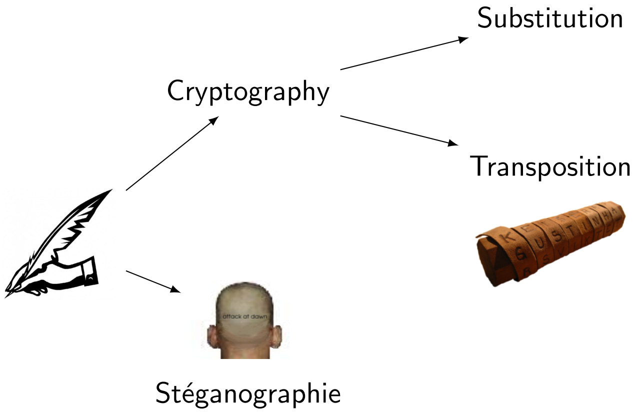
Information hiding



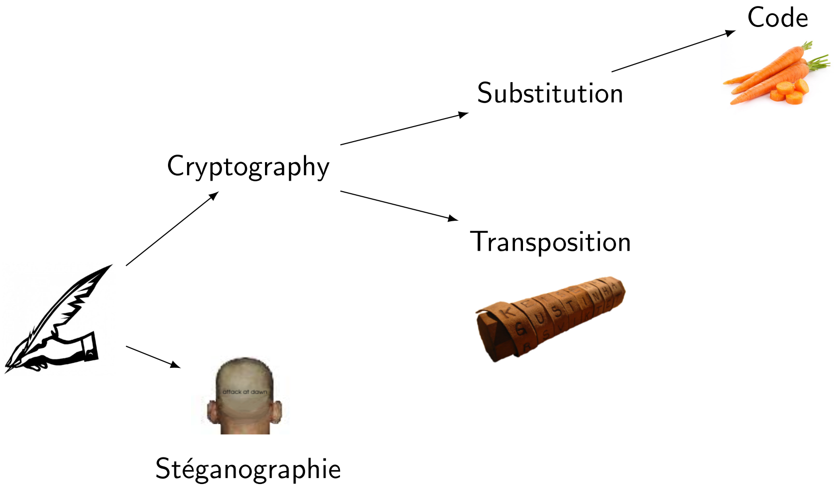
Information hiding



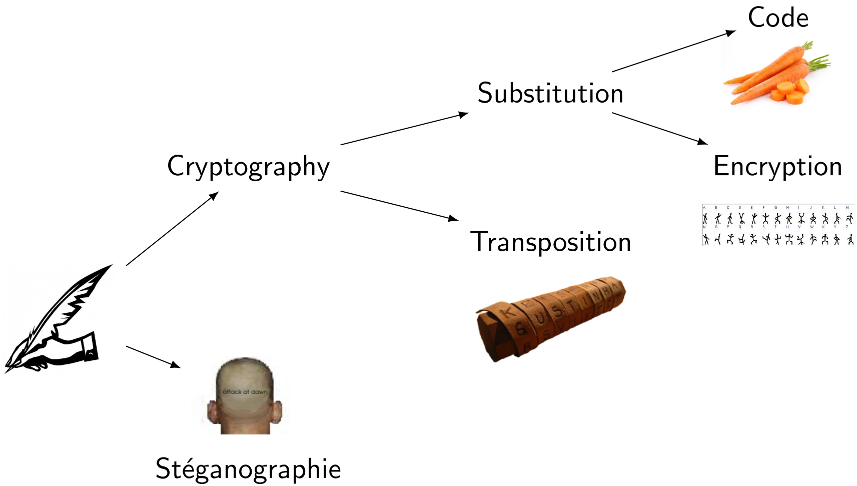
Information hiding



Information hiding

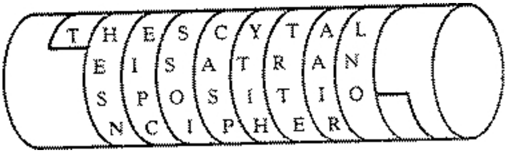


Information hiding



Greeks and the Scythale

Greeks and the Scythale



Transposition

Transposition ciphers

- For block length t, let K be the set of permutations on $\{1,\ldots,t\}$. For each $e \in K$ and $m \in M$

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

- The set of all such transformations is called a transposition cipher.
- To decrypt $c = c_1 c_2 \cdots c_t$ compute $D_d(c) = c_{d(1)} c_{d(2)} \cdots c_{d(t)}$, where d is inverse permutation.
- Letters unchanged so frequency analysis can be used to reveal if ciphertext is a transposition. Decrypt by exploiting frequency analysis for diphthongs, tripthongs, words, etc.

Romans

Caesar Encryption

Substitution $+3$

Romans

Caesar Encryption
Substitution $+3$

Dyh Fhvdu

Romans

Caesar Encryption
Substitution $+3$

Dyh Fhvdu

Ave Cesar

Mono-alphabetic substitution ciphers

- Simplest kind of cipher. Idea over 2,000 years old.
- Let K be the set of all permutations on the alphabet A. Define for each $e \in K$ an encryption transformation E_e on strings $m = m_1 m_2 \cdots m_n \in M$ as

$$E_e(m) = e(m_1)e(m_2)\cdots e(m_n) = c_1 c_2 \cdots c_n = c\,.$$

- To decrypt c, compute the inverse permutation $d = e^{-1}$ and

$$D_d(c) = d(c_1)d(c_2)\cdots d(c_n) = m\,.$$

- E_e is a simple substitution cipher or a mono-alphabetic substitution cipher.

Substitution cipher examples

- KHOOR ZRUOG

Substitution cipher examples

- KHOOR ZRUOG = HELLO WORLD
  Caesar cipher: each plaintext character is replaced by the character three to the right modulo 26.

Substitution cipher examples

- KHOOR ZRUOG = HELLO WORLD
  Caesar cipher: each plaintext character is replaced by the character three to the right modulo 26.
- Zl anzr vf Nqnz

Substitution cipher examples

- KHOOR ZRUOG = HELLO WORLD
  Caesar cipher: each plaintext character is replaced by the character three to the right modulo 26.
- ZI anzr vf Nqnz = My name is Adam
  ROT13: shift each letter by 13 places.
  Under Unix: `tr a-zA-Z n-za-mN-ZA-M`.
- 2-25-5 2-25-5

Substitution cipher examples

- KHOOR ZRUOG = HELLO WORLD
  Caesar cipher: each plaintext character is replaced by the character three to the right modulo 26.
- Zl anzr vf Nqnz = My name is Adam
  ROT13: shift each letter by 13 places.
  Under Unix: `tr a-zA-Z n-za-mN-ZA-M`.
- 2-25-5 2-25-5 = BYE BYE
  Alphanumeric: substitute numbers for letters.

How hard are these to cryptanalyze? Caesar? General?

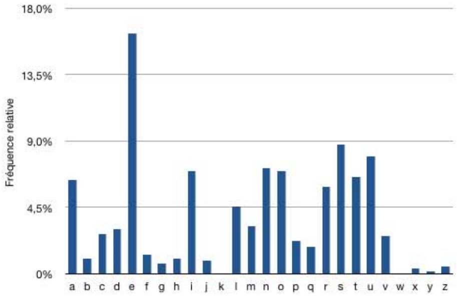
Is it secure?

Is it secure?

Key spaces are typically huge. 26 letters $\rightsquigarrow$ 26! possible keys.

Is it secure?

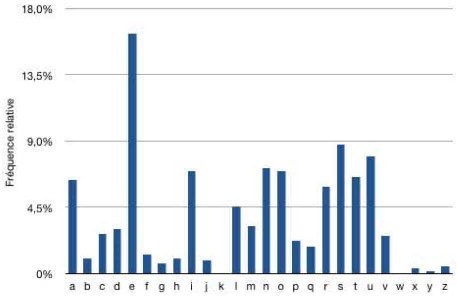
Key spaces are typically huge. 26 letters $\rightsquigarrow$ 26! possible keys.



Frequency analysis

Is it secure?

Key spaces are typically huge. 26 letters $\rightsquigarrow$ 26! possible keys.



Frequency analysis

Except for short, atypical texts
From Zanzibar to Zambia and Zaire, ozone zones make zebras run zany zigzags.
$\Rightarrow$ More sophistication required to mask statistical regularities

Homophonic substitution ciphers

- To each $a \in A$, associate a set $H(a)$ of strings of t symbols, where $H(a), a \in A$ are pairwise disjoint. A homophonic substitution cipher replaces each a with a randomly chosen string from $H(a)$. To decrypt a string c of t symbols, one must determine an $a \in A$ such that $c \in H(a)$. The key for the cipher is the sets $H(a)$.

Homophonic substitution ciphers

- To each $a \in A$, associate a set $H(a)$ of strings of t symbols, where $H(a), a \in A$ are pairwise disjoint. A homophonic substitution cipher replaces each a with a randomly chosen string from $H(a)$. To decrypt a string c of t symbols, one must determine an $a \in A$ such that $c \in H(a)$. The key for the cipher is the sets $H(a)$.

Example:

$A = \{a, b\}$, $H(a) = \{00, 10\}$, and $H(b) = \{01, 11\}$. The plaintext *ab* encrypts to one of 0001, 0011, 1001, 1011.

Rational: makes frequency analysis more difficult.

Cost: data expansion and more work for decryption.

Polyalphabetic substitution ciphers

- Leon Alberti: conceal distribution using family of mappings.

- A polyalphabetic substitution cipher is a block cipher with block length t over alphabet A where:
  - the key space K consists of all ordered sets of t permutations over A, $(p_1, p_2, \ldots, p_t)$.
  - Encryption of $m = m_1 \cdots m_t$ under key $e = (p_1, \cdots, p_t)$ is $E_e(m) = p_1(m_1) \cdots p_t(m_t)$.
  - Decryption key for e is $d = (p_1^{-1}, \cdots p_t^{-1})$.

Example: Vigenère ciphers 1553

- Key given by sequence of numbers $e = e_1, \ldots, e_t$, where

$$p_i(a) = (a + e_i) \bmod n$$

  defining a permutation on an alphabet of size n.
- Example: English ($n = 26$), with k = 3,7,10

  m = THI SCI PHE RIS CER TAI NLY NOT SEC URE

then

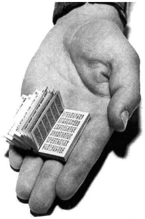
$E_e(m) =$ WOS VJS SOO UPC FLB WHS QSI QVD VLM XYO

One-time pads (Vernam cipher)

- A one-time pad is a cipher defined over $\{0,1\}$. Message $m_1 \cdots m_n$ is encrypted by a binary key string $k_1 \cdots k_n$.

$$\begin{aligned} E_{k_1 \cdots k_n}(m_1 \cdots m_n) &= (m_1 \oplus k_1) \cdots (m_n \oplus k_n) \\ D_{k_1 \cdots k_n}(c_1 \cdots c_n) &= (c_1 \oplus k_1) \cdots (c_n \oplus k_n) \end{aligned}$$

- Unconditional (information theoretic) security, if key isn't reused!



One-Time Pad (Vernam 1917)

Example:

$$\begin{array}{rcl} m & = & 010111 \\ k & = & 110010 \\ \hline c & = & 100101 \end{array}$$

Problem?

One-Time Pad (Vernam 1917)

Example:

$$\begin{array}{rcl} m & = & 010111 \\ k & = & 110010 \\ \hline c & = & 100101 \end{array}$$

Problem? Securely exchanging and synchronizing long keys.

Kerchoff's Principle

In 1883, a Dutch linguist Auguste Kerchoff von Nieuwenhof stated in his book "La Cryptographie Militaire" that:

"the security of a crypto-system must be totally dependent on the secrecy of the key, not the secrecy of the algorithm."

Author's name sometimes spelled Kerckhoff

Chiffrement : Enigma (Seconde guerre mondiale)

$+$ $=$

Chiffrement : Enigma (Seconde guerre mondiale)

$+$

$=$

Chiffrement : Enigma (Seconde guerre mondiale)

+ =

+ =

Chiffrement : Enigma (Seconde guerre mondiale)

$+$ $=$

$+$ $=$

Chiffrement : Enigma (Seconde guerre mondiale)

$+$

$=$

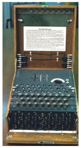
$+$

$=$

$+$

$+ \ldots +$

$+$

$=$


Shannon's Principle 1949

Confusion

The purpose of confusion is to make the relation between the key and the ciphertext as complex as possible.

Ciphers that do not offer much confusion (such as Vigenere cipher) are susceptible to frequency analysis.

Shannon's Principle 1949

Confusion

The purpose of confusion is to make the relation between the key and the ciphertext as complex as possible.

Ciphers that do not offer much confusion (such as Vigenere cipher) are susceptible to frequency analysis.

Diffusion

Diffusion spreads the influence of a single plaintext bit over many ciphertext bits.

The best diffusing component is substitution (homophonic)

Shannon's Principle 1949

Confusion

The purpose of confusion is to make the relation between the key and the ciphertext as complex as possible.

Ciphers that do not offer much confusion (such as Vigenere cipher) are susceptible to frequency analysis.

Diffusion

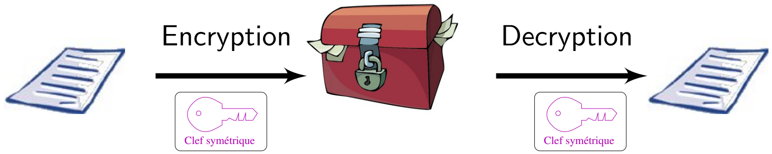
Diffusion spreads the influence of a single plaintext bit over many ciphertext bits.

The best diffusing component is substitution (homophonic)

Principle

A good cipher design uses Confusion and Diffusion together

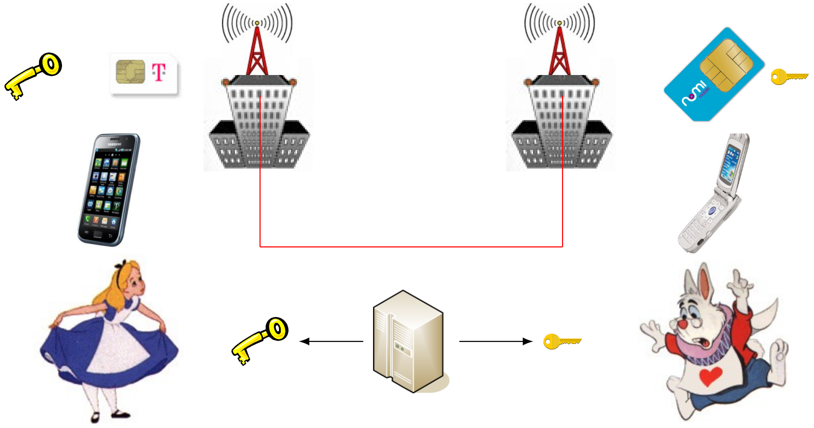
Symmetric Encryption



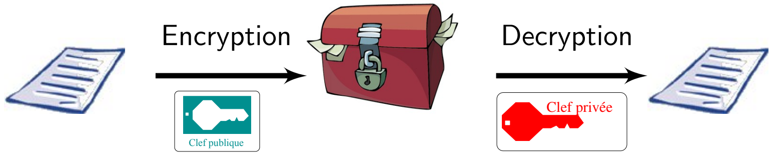
Examples

- DES
- AES

Cellphone Communications



Public Key Encryption



Examples

- RSA : $c = m^e \mod n$
- ElGamal : $c \equiv (g^r, h^r \cdot m)$

Comparison

- Size of the key
- Complexity of computation (time, hardware, cost ...)
- Number of different keys ?
- Key distribution
- Signature only possible with asymmetric scheme

Computational cost of encryption

2 hours of video (assumes 3Ghz CPU)

	DVD 4,7 G.B		Blu-Ray 25 GB	
Schemes	encrypt	decrypt	encrypt	decrypt
RSA 2048(1)	22 min	24 h	115 min	130 h
RSA 1024(1)	21 min	10 h	111 min	53 h
AES CTR(2)	20 sec	20 sec	105 sec	105 sec

Outline

One-way function and Trapdoor

Definition

A function is *One-way*, if :

- it is easy to compute
- its inverse is hard to compute :

$$\Pr[m \xleftarrow{r} \{0,1\}^*; y := f(m) : f(A(y,f)) = y]$$

  is negligible.

Trapdoor:

- Inverse is easy to compute given an additional information (an inverse key *e.g.* in RSA).

Integer Factoring

$\rightarrow$ Use of algorithmically hard problems.

Factorization

- $p, q \mapsto n = p.q$ easy (quadratic)
- $n = p.q \mapsto p, q$ difficult

RSA

RSA function $n = pq$, p and q primes.
e: public exponent

- $x \mapsto x^e \mod n$ easy (cubic)
- $y = x^e \mapsto x \mod n$ difficult
  $x = y^d$ where $d = e^{-1} \mod \phi(n)$

Soundness

Assume $n = pq$, $gcd(e, \phi(n)) = 1$ and $d = e^{-1} \mod \phi(n)$.
$c^d = m^{de} = m.m^{k\phi(n)} \mod n$
According to the Fermat Little Theorem $\forall x \in (Z/nZ)^*, x^{\phi(n)} = 1$

Example RSA

Example

- $p = 61$ (destroy this after computing E and D)
- $q = 53$ (destroy this after computing E and D)
- $n = pq = 3233$ modulus (give this to others)
- $e = 17$ public exponent (give this to others)
- $d = 2753$ private exponent (keep this secret!)

Your public key is (e, n) and your private key is d.
$encrypt(T) = (T^e) \bmod n = (T^{17}) \bmod 3233$
$decrypt(C) = (C^d) \bmod n(C^{2753}) \bmod 3233$

- $encrypt(123) = 123^{17} \bmod 3233$
  $= 337587917446653715596592958817679803 \bmod 3233$
  $= 855$
- $decrypt(855) = 855^{2753} \bmod 3233$

Complexity Estimates

Estimates for integer factoring Lenstra-Verheul 2000

Modulus (bits)	Operations ($\log_2$)
512	58
1024	80
2048	111
4096	149
8192	156

$\approx 2^{60}$ years

$\rightarrow$ Can be used for RSA too.

ElGamal Encryption Scheme

Key generation: Alice chooses a prime number p and a group generator g of $(Z/pZ)^*$ and $a \in (Z/(p-1)Z)^*$.

Public key: (p, g, h), where $h = g^a \mod p$.

Private key: a

Encryption: Bob chooses $r \in_R (Z/(p-1)Z)^*$ and computes $(u, v) = (g^r, Mh^r)$

Decryption: Given (u, v), Alice computes $M \equiv_p \frac{v}{u^a}$

Justification: $\frac{v}{u^a} = \frac{Mh^r}{g^{ra}} \equiv_p M$

Remarque: re-usage of the same random r leads to a security flaw:

$$\frac{M_1 h^r}{M_2 h^r} \equiv_p \frac{M_1}{M_2}$$

Practical Inconvenience: Cipher is twice as long as plain text.

Optimal Asymmetric Encryption Padding (OAEP)

The OAEP cryptosystem (K, E, D) obtained from a permutation f, whose inverse is denoted by g. And two hash functions:

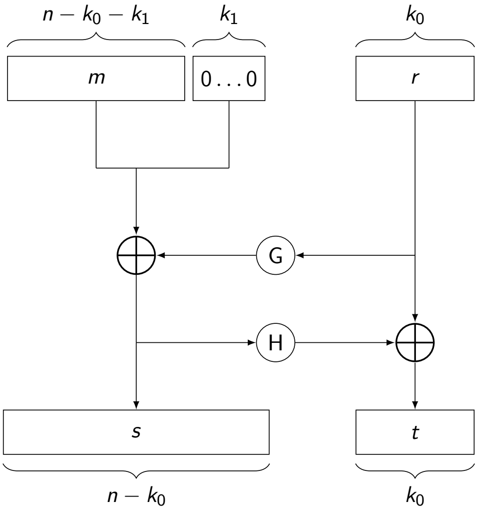
$$G : \{0,1\}^{k_0} \to \{0,1\}^{k-k_0}$$
$$H : \{0,1\}^{k-k_0} \to \{0,1\}^{k_0}$$

$K(1^k)$: specifies an instance of the function f, and of its inverse g. The public key pk is therefore f and the private key sk is g.

OAEP: Encryption

$$E_{pk}(m, r) = c = f(s, t) \text{ with } m \in \{0,1\}^n, \text{ and } r \leftarrow \{0,1\}^{k_0}$$

$$s = (m||0^{k_1}) \oplus G(r), t = r \oplus H(s)$$



OAEP: Decryption

$D_{sk}(c)$

$$g(c) = (s, t)$$

$$r = t \oplus H(s)$$

$$M = s \oplus G(r)$$

If $[M]_{k_1} = 0^{k_1}$, the algorithm returns $[M]^n$, otherwise it returns "Reject"

- $[M]_{k_1}$ denotes the k_1 least significant bits of M
- $[M]^n$ denotes the n most significant bits of M

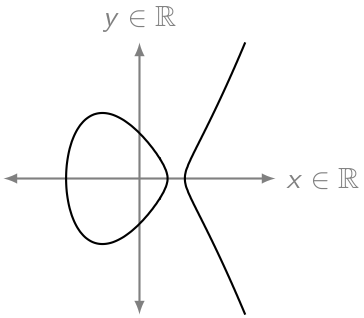
Others Cryptosystems

- Bellare & Rogaway'93:
$$f(r)||x \oplus G(r)||H(x||r)$$
- Zheng & Seberry'93:
$$f(r)||G(r) \oplus (x||H(x))$$
- OAEP'94 (Bellare & Rogaway):
$$f(s||r \oplus H(s))$$
where $s = x0^k \oplus G(r)$
- OAEP+'02 (Shoup):
$$f(s||r \oplus H(s))$$
where $s = x \oplus G(r)||H'(r||x)$.
- Fujisaki & Okamoto'99:
$$E((x||r); H(x||r))$$
where E is IND-CPA.

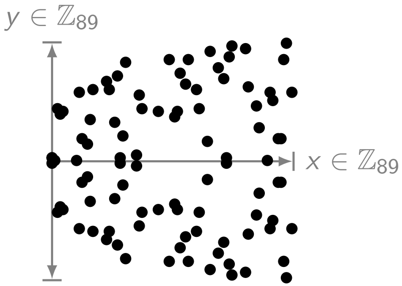
Outline

Introduction

$$y^2 = x^3 + ax + b$$



$y^2 = x^3 - 2x + 1$ over $\mathbb{R}$



$y^2 = x^3 - 2x + 1$ over $\mathbb{Z}_{89}$

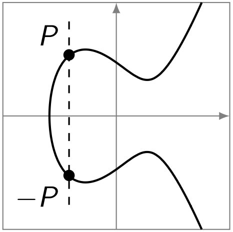
$E(K) = \{(x, y)$ such that $y^2 = x^3 + ax + b\}$ plus an extra point "at infinite"
Weierstrass form if $\Delta = -16(4a^3 + 27b^2) \neq 0$ (if K is not of characteristic 2 or 3).

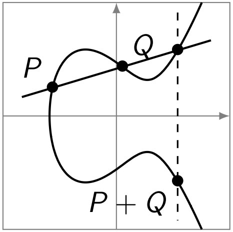
Laws

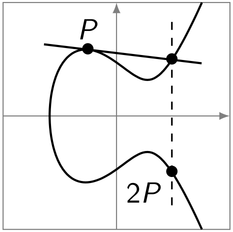
Theorem

- Addition law on $E(K)$
  - Associativity: $(P_1 + P_2) + P_3 = P_1 + (P_2 + P_3)$
  - Commutativity: $P_1 + P_2 = P_2 + P_1$
  - Neutral element is ∞: $P + \infty = P$
  - Inverse: Given P on E, there exists P' on E with $P + P' = \infty$ (usually denoted $-P$)
- Three aligned points sum to neutral element often denoted zero

Laws







Inverse element $-P$

Addition $P + Q$
"Chord rule"

Doubling $P + P$
"Tangent rule"

$$P + R + Q = 0 \Rightarrow R = -(P + Q)$$

$$R + S + 0 = 0 \Rightarrow R = -S$$

“Elliptic Discrete Logarithm”

Hard Problem

Finding k, given P and $Q = kP$. is computationally intractable for large values of k.

Cryptosystem: ECDH

Exercice: Give an Elliptic curve of ElGamal.

Cryptosystem: ECDH

Alice's key is (d_A, Q_A) where $Q_A = d_A G$.

DH like Protocol

1. Alice sends Q_A, G to Bob.
2. Bob computes $k = d_B Q_A$.
3. Bob sends to Alice Q_B
4. Alice computes $k = d_A Q_B$.

The shared key is x_k (the x coordinate of the point).

The number calculated by both parties is equal, because
$k = d_A Q_B = d_A d_B G = d_B d_A G = d_B Q_A = k$.

ECDSA (Digital Signature Algorithm) I

Alice private key d_A and a public key Q_A (where $Q_A = d_A G$).

Signature generation algorithm

1. Calculate $e = HASH(m)$, where HASH is a cryptographic hash function, such as SHA-1.
2. Select a random integer k from $[1, n-1]$.
3. Calculate $r = x_1 (\mod n)$, where $(x_1, y_1) = kG$.
   If $r = 0$, go back to step 2.
4. Calculate $s = k^{-1}(e + rd_A)(\mod n)$.
   If $s = 0$, go back to step 2.
5. The signature is the pair (r, s).

ECDSA (Digital Signature Algorithm) II

Signature verification algorithm

1. Verify that r and s are integers in $[1, n-1]$.
   If not, the signature is invalid.
2. Calculate $e = HASH(m)$, where HASH is the same function used in the signature generation.
3. Calculate $w = s^{-1} (\bmod n)$.
4. Calculate $u_1 = ew (\bmod n)$ and $u_2 = rw (\bmod n)$.
5. Calculate $(x_1, y_1) = u_1 G + u_2 Q_A$.
6. The signature is valid if $r = x_1 (\bmod n)$, invalid otherwise.

ECDSA (Digital Signature Algorithm)

$$s = k^{-1}(e + rd_A)(\mod n)$$

Hence
$k = s^{-1}(e + rd_A)(\mod n) = w(e + rd_A) = we + wrd_A = u_1 + u_2 d_A$
since $w = s^{-1}$, $u_1 = we$ and $u_2 = wr$

$$(x_1, y_1) = u_1 G + u_2 Q_A$$

Hence $(x_1, y_1) = u_1 G + u_2 d_A G = kG$
because $Q_A = d_A G$ and $k = u_1 + u_2 d_A$
We conclude that $r = x_1(\mod n)$ by construction.

Outline

Data Encryption Standard, (call in 1973)

Lucifer designed in 1971 by Horst Feistel at IBM.

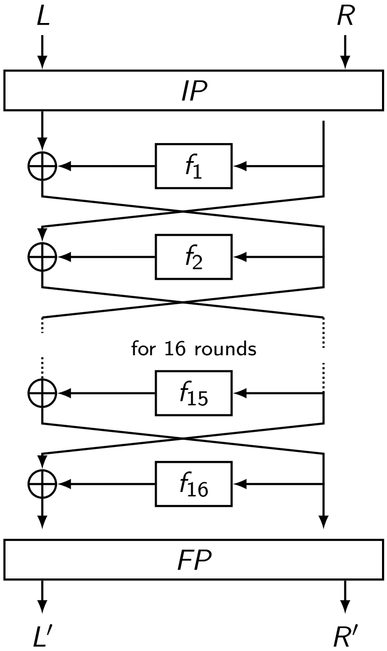
- Block cipher, encrypting 64-bit blocks
  Uses 56 bit keys
  Expressed as 64 bit numbers (8 bits parity checking)
- First cryptographic standard.
  - 1977 US federal standard (US Bureau of Standards)
  - 1981 ANSI private sector standard

DES — overall form

- 16 rounds Feistel cipher + key-scheduler.
- Key scheduling algorithm derives subkeys K_i from original key K.
- Initial permutation at start, and inverse permutation at end.
- f_i consists of two permutations and an s-box substitution.

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

DES — overall form



DES — Subkey generation

First, produce two subkeys K1 and K2:

$$K1 = P8(LS1(P10(key)))$$

$$K2 = P8(LS2(LS1(P10(key))))$$

where P8, P10, LS1 and LS2 are bit substitution operators.

- P10 : 10 bits to 10 bits

3	5	2	7	4	10	1	9	8	6

- P8 : 10 bits to 8 bits

6	3	7	4	8	5	10	9

- LS1 ("left shift 1 bit" on 5 bit words) : 10 bits to 10 bits

2	3	4	5	1	7	8	9	10	6

- LS2 ("left shift 2 bit" on 5 bit words) : 10 bits to 10 bits

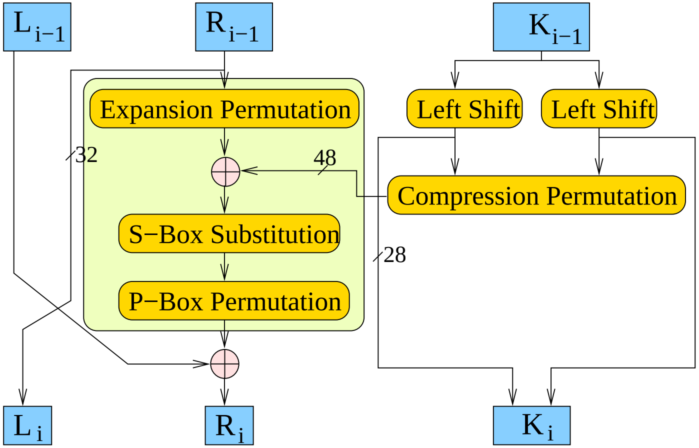
3	4	5	1	2	8	9	10	6	7

DES — Before round subkey

Each half of the key schedule state is rotated left by a number of places.

# Rds	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Left	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

DES — 1 round



$(b_1 b_6, b_2 b_3 b_4 b_5)$, C_j represents the binary value in the row $b_1 b_6$ and column $b_2 b_3 b_4 b_5$ of the S_j box.

S-Boxes: S1, S2, S3, S4

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

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

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

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

S-Boxes: S5, S6, S7 and S8

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

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

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

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

Permutation P

58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

Decryption DES

Use inverse sequence key.

- $IP(C) = IP(IP^{-1}(R_{16}||L_{16})$
- $L'_0 = R_{16}$ and $R'_0 = L_{16}$

$$L'_1 = R'_0 = L_{16} = R_{15}$$

$$R'_1 = L'_0 \oplus f(R'_0, K'_0)$$
$$R'_1 = R_{16} \oplus f(L_{16}, K_{15})$$
$$R'_1 = R_{16} \oplus f(R_{15}, K_{15})$$
$$R'_1 = L_{15}$$

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

Property of DES

DES exhibits the complementation property, namely that

$$E_K(P) = C \Leftrightarrow E_{\overline{K}}(\overline{P}) = \overline{C}$$

where $\overline{x}$ is the bitwise complement of x. E_K denotes encryption with key K. Then P and C denote plaintext and ciphertext blocks respectively.

Anomalies of DES

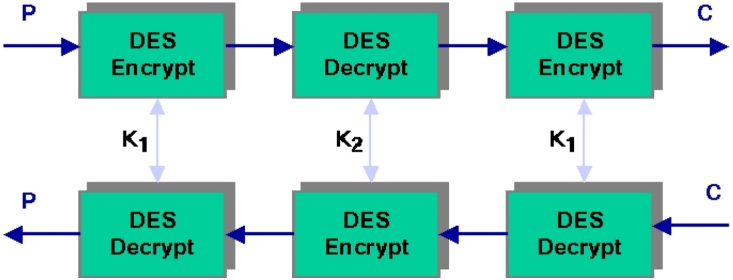
- Existence of 6 pairs of *semi-weak keys*: $E_{k_1}(E_{k_2}(x)) = x$.
  - 0x011F011F010E010E and 0x1F011F010E010E01
  - 0x01E001E001F101F1 and 0xE001E001F101F101
  - 0x01FE01FE01FE01FE and 0xFE01FE01FE01FE01
  - 0x1FE01FE00EF10EF1 and 0xE01FE01FF10EF10E
  - 0x1FFE1FFE0EFE0EFE and 0xFE1FFE1FFE0EFE0E
  - 0xE0FEE0FEF1FEF1FE and 0xFEE0FEE0FEF1FEF1

Security of DES

- No security proofs or reductions known
- Main attack: exhaustive search
  - 7 hours with 1 million dollar computer (in 1993).
  - 7 days with $10,000 FPGA-based machine (in 2006).
- Mathematical attacks
  - Not know yet.
  - But it is possible to reduce key space from 2^{56} to 2^{43} using (linear) cryptanalysis.
    - To break the full 16 rounds, differential cryptanalysis requires 2^{47} chosen plaintexts (Eli Biham and Adi Shamir).
    - Linear cryptanalysis needs 2^{43} known plaintexts (Matsui, 1993)

Triple DES

- Use three stages of encryption instead of two.



- Compatibility is maintained with standard DES ($K_2 = K_1$).
- No known practical attack
  $\Rightarrow$ brute-force search with 2^{112} operations.

Advanced Encryption Standard

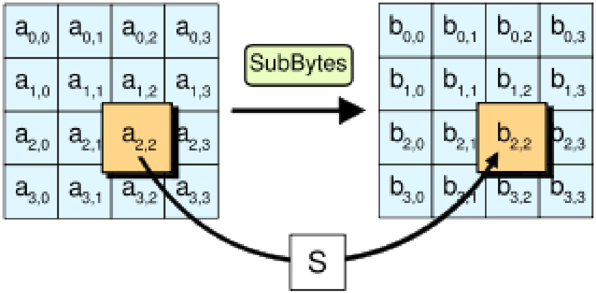
- Block cipher, approved for use by US Government in 2002. Very popular standard, designed by two Belgian cryptographers.
- Block-size = 128 bits, Key size = 128, 192, or 256 bits.
- Uses various substitutions and transpositions + key scheduling, in different rounds.
- Algorithm believed secure. Only attacks are based on side channel analysis, i.e., attacking implementations that inadvertently leak information about the key.

Key Size	Round Number
128	10
192	12
256	14

AES: High-level cipher algorithm

- KeyExpansion using Rijndael's key schedule
- Initial Round: AddRoundKey
- Rounds:
    1. SubBytes: a non-linear substitution step where each byte is replaced with another according to a lookup table.
    2. ShiftRows: a transposition step where each row of the state is shifted cyclically a certain number of steps.
    3. MixColumns: a mixing operation which operates on the columns of the state, combining the four bytes in each column
    4. AddRoundKey: each byte of the state is combined with the round key; each round key is derived from the cipher key using a key schedule.
- Final Round (no MixColumns)
    1. SubBytes
    2. ShiftRows
    3. AddRoundKey

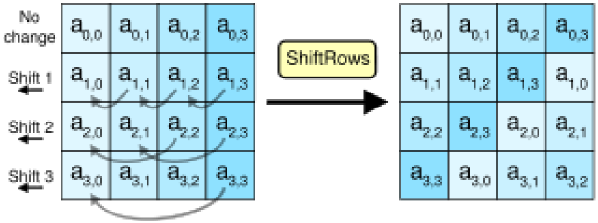
AES: SubBytes



$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,0}$ $a_{1,1}$ $a_{1,2}$ $a_{1,3}$
$a_{2,0}$ $a_{2,1}$ $a_{2,2}$ $a_{2,3}$
$a_{3,0}$ $a_{3,1}$ $a_{3,2}$ $a_{3,3}$
SubBytes
$b_{0,0}$ $b_{0,1}$ $b_{0,2}$ $b_{0,3}$
$b_{1,0}$ $b_{1,1}$ $b_{1,2}$ $b_{1,3}$
$b_{2,0}$ $b_{2,1}$ $b_{2,2}$ $b_{2,3}$
$b_{3,0}$ $b_{3,1}$ $b_{3,2}$ $b_{3,3}$
S

SubBytes: a non-linear substitution step where each byte is replaced with another according to a lookup table.

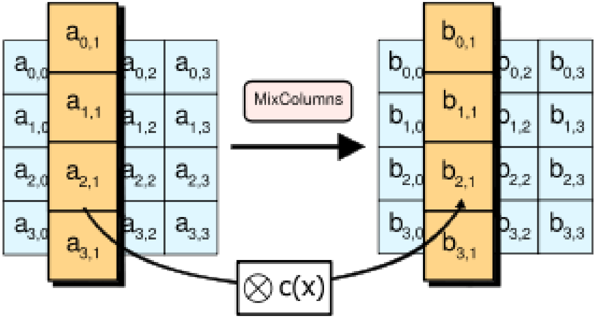
AES: ShiftRows



No change
Shift 1
Shift 2
Shift 3
$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,0}$ $a_{1,1}$ $a_{1,2}$ $a_{1,3}$
$a_{2,0}$ $a_{2,1}$ $a_{2,2}$ $a_{2,3}$
$a_{3,0}$ $a_{3,1}$ $a_{3,2}$ $a_{3,3}$
ShiftRows
$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,1}$ $a_{1,2}$ $a_{1,3}$ $a_{1,0}$
$a_{2,2}$ $a_{2,3}$ $a_{2,0}$ $a_{2,1}$
$a_{3,3}$ $a_{3,0}$ $a_{3,1}$ $a_{3,2}$

ShiftRows: a transposition step where each row of the state is shifted cyclically a certain number of steps.

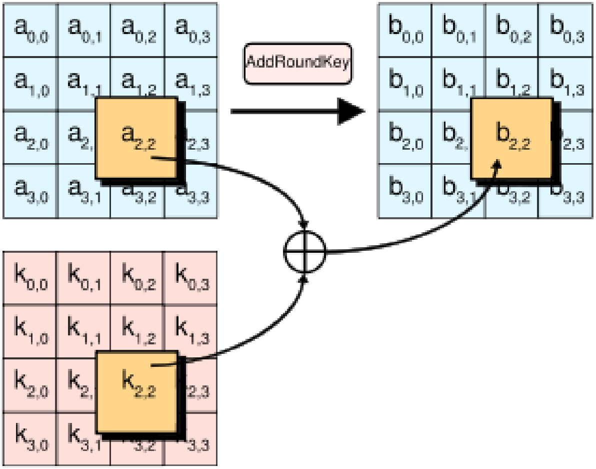
AES: MixColumns



$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,0}$ $a_{1,1}$ $a_{1,2}$ $a_{1,3}$
$a_{2,0}$ $a_{2,1}$ $a_{2,2}$ $a_{2,3}$
$a_{3,0}$ $a_{3,1}$ $a_{3,2}$ $a_{3,3}$
MixColumns
$\otimes c(x)$
$b_{0,0}$ $b_{0,1}$ $b_{0,2}$ $b_{0,3}$
$b_{1,0}$ $b_{1,1}$ $b_{1,2}$ $b_{1,3}$
$b_{2,0}$ $b_{2,1}$ $b_{2,2}$ $b_{2,3}$
$b_{3,0}$ $b_{3,1}$ $b_{3,2}$ $b_{3,3}$

MixColumns: a mixing operation which operates on the columns of the state, combining the four bytes in each column

AES: AddRoundKey



$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,0}$ $a_{1,1}$ $a_{1,2}$ $a_{1,3}$
$a_{2,0}$ $a_{2,2}$
$a_{3,0}$ $a_{3,1}$
AddRoundKey
$b_{0,0}$ $b_{0,1}$ $b_{0,2}$ $b_{0,3}$
$b_{1,0}$ $b_{1,1}$ $b_{1,2}$ $b_{1,3}$
$b_{2,0}$ $b_{2,2}$ $b_{2,3}$
$b_{3,0}$ $b_{3,3}$
$k_{0,0}$ $k_{0,1}$ $k_{0,2}$ $k_{0,3}$
$k_{1,0}$ $k_{1,1}$ $k_{1,2}$ $k_{1,3}$
$k_{2,0}$ $k_{2,2}$
$k_{3,0}$ $k_{3,1}$

AddRoundKey: each byte of the state is combined with the round key; each round key is derived from the cipher key using a key schedule.

Outline

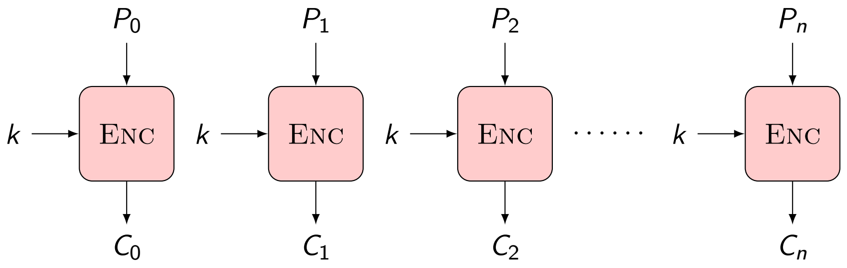
Electronic Book Code (ECB)

Each block of the same length is encrypted separately using the same key K. In this mode, only the block in which the flipped bit is contained is changed. Other blocks are not affected.

ECB Encryption Algorithm

```
**algorithm** $E_K(M)$
if $(|M| \bmod n \neq 0$ or $|M| = 0)$ then return FAIL
Break $M$ into n-bit blocks $M[1] \dots M[m]$
for $i = 1$ to $m$ do $C[i] = E_K(M[i])$
$C = C[1] \dots C[m]$
return $C$
```

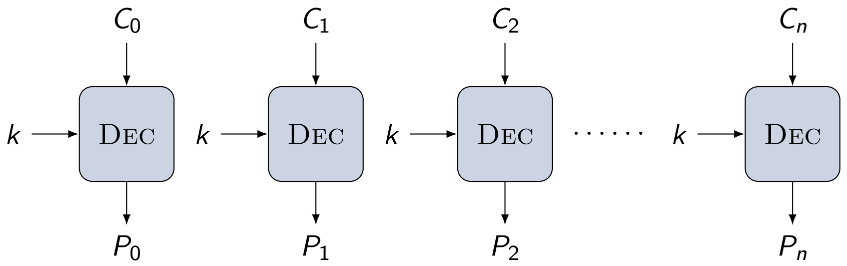
ECB Encryption



ECB Decryption Algorithm

```
**algorithm** $D_K(C)$
if ($|C| \bmod n \neq 0$ or $|C| = 0$) then return FAIL
Break $C$ into n-bit blocks $C[1] \ldots C[m]$
for $i = 1$ to $m$ do $M[i] = D_K(C[i])$
$M = M[1] \ldots M[m]$
return $M$
```

ECB Decryption



Cipher-block chaining (CBC)

If the first block has index 1, the mathematical formula for CBC encryption is

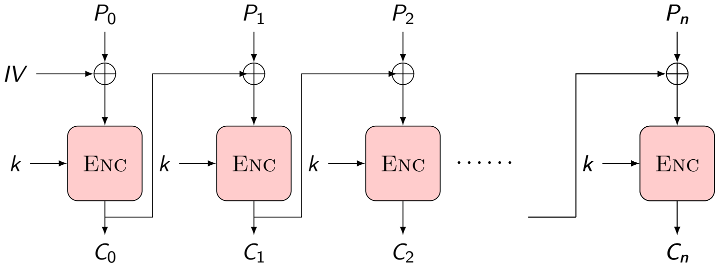
$$C_i = E_K(P_i \oplus C_{i-1}), C_0 = IV$$

while the mathematical formula for CBC decryption is

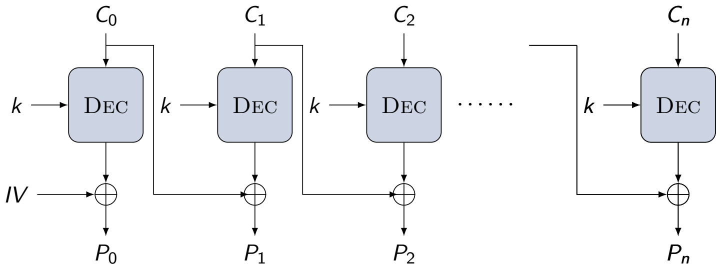
$$P_i = D_K(C_i) \oplus C_{i-1}, C_0 = IV$$

CBC has been the most commonly used mode of operation.

CBC Encryption



CBC Decryption



The cipher feedback (CFB)

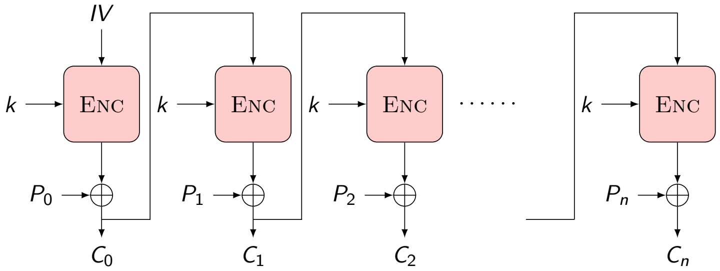
A close relative of CBC:

$$C_i = E_K(C_{i-1}) \oplus P_i$$

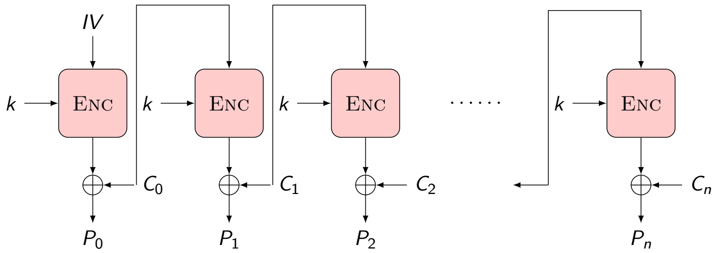
$$P_i = E_K(C_{i-1}) \oplus C_i$$

$$C_0 = \text{ IV}$$

CFB Encryption



CFB Decryption



Output feedback (OFB)

Because of the symmetry of the XOR operation, encryption and decryption are exactly the same:

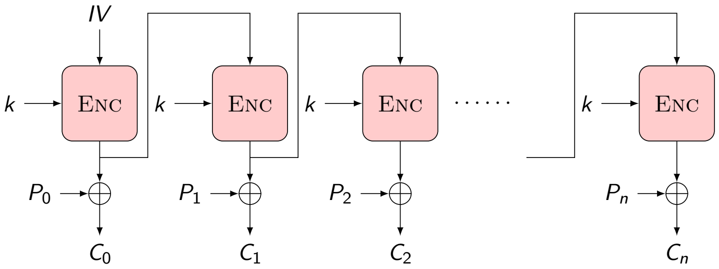
$$C_i = P_i \oplus O_i$$

$$P_i = C_i \oplus O_i$$

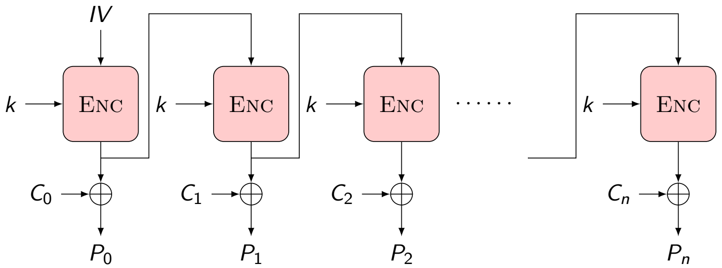
$$O_i = E_K(O_{i-1})$$

$$O_0 = \text{IV}$$

OFB encryption



OFB Decryption



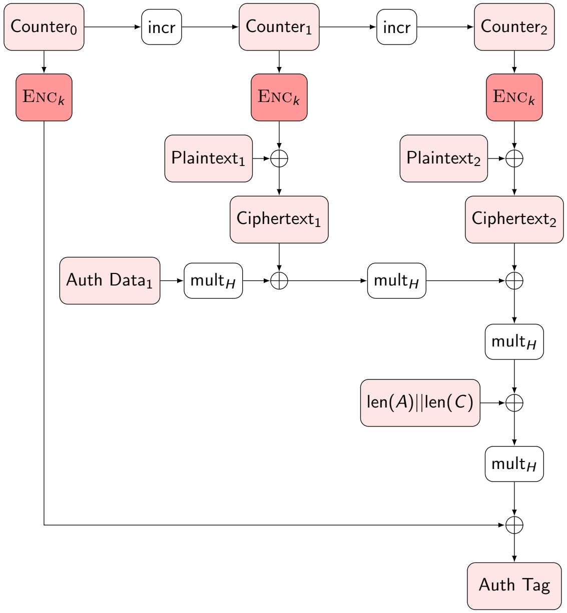
Counter Mode (CTR)

$$C_0 = IV$$

$$C_i = P_i \oplus E_k(IV + i - 1)$$

$$P_i = C_i \oplus E_k(IV + i - 1)$$

GCM Galois/Counter Mode by D. McGrew and J. Viega



GCM

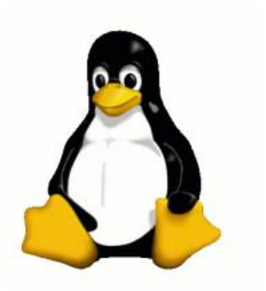
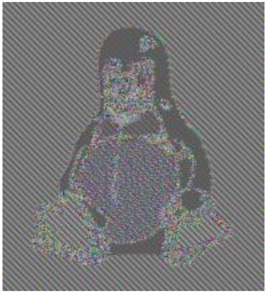
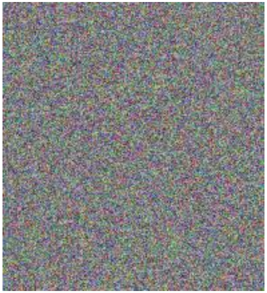
$GF(2^{128})$ est défini par $x^{128} + x^7 + x^2 + x + 1$

$$S_i = \begin{cases} A_i & \text{for } i = 1, \ldots, m-1 \\ A_m^* \parallel 0^{128-v} & \text{for } i = m \\ C_{i-m} & \text{for } i = m+1, \ldots, m+n-1 \\ C_n^* \parallel 0^{128-u} & \text{for } i = m+n \\ \operatorname{len}(A) \parallel \operatorname{len}(C) & \text{for } i = m+n+1 \end{cases}$$

where $len(A)$ and $len(C)$ are the 64-bit representations of the bit lengths of A and C, respectively, $v = len(A) \mod 128$ is the bit length of the final block of A, $u = len(C) \mod 128$ is the bit length of the final block of C.

$$X_i = \sum_{j=1}^{i} S_j \cdot H^{i-j+1} = \begin{cases} 0 & \text{for } i = 0 \\ (X_{i-1} \oplus S_i) \cdot H & \text{for } i = 1, \ldots, m+n+1 \end{cases}$$

ECB vs Others

Outline

“Classifications” of Hash Functions

Unkeyed Hash function

- Modification Code Detection (MDC)
- Data integrity
- Fingerprints of messages
- Other applications

Keyed Hash function

- Message Authentication Code (MAC)
- Password Verification in uncrypted password-image files.
- Key confirmation or establishment
- Time-stamping
- Others applications

Hash Functions

A hash function H takes as input a bit-string of any finite length and returns a corresponding 'digest' of fixed length.

$$h : \{0,1\}^* \rightarrow \{0,1\}^n$$

$$H(Alice) =$$

Definition (Pre-image resistance (One-way) OWHF)

Given an output y, it is computationally infeasible to compute x such that

$$h(x) = y$$

Properties of hash functions

2nd Pre-image resistance (weak-collision resistant) CRHF

Given an input x, it is computationally infeasible to compute x' such that

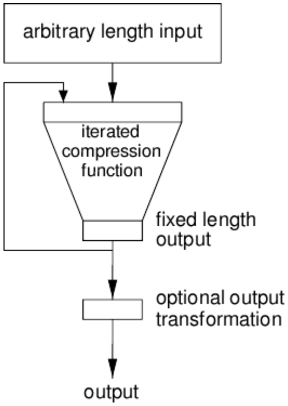
$$h(x') = h(x)$$

Collision resistance (strong-collision resistant)

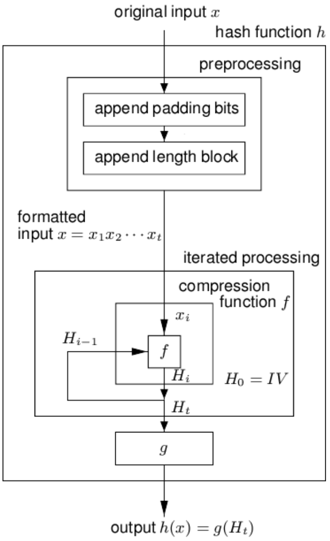
It is computationally infeasible to compute x and x' such that

$$h(x) = h(x')$$

Basic construction of hash functions



Basic construction of hash functions



Basic construction of hash functions (Merkle-Damgård)

$$f : \{0,1\}^m \to \{0,1\}^n$$

1. Break the message x to hash in blocks of size $m - n$:

$$x = x_1 x_2 \dots x_t$$

2. Pad x_t with zeros as necessary.
3. Define x_{t+1} as the binary representation of the bit length of x.
4. Iterate over the blocks:

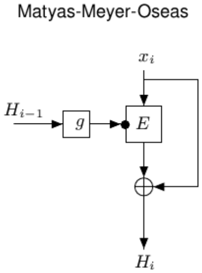
$$\begin{aligned} H_0 &= 0^n \\ H_i &= f(H_{i-1} || x_i) \\ h(x) &= H_{t+1} \end{aligned}$$

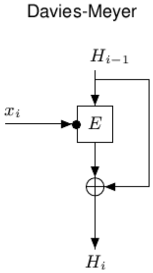
Basic construction of hash functions

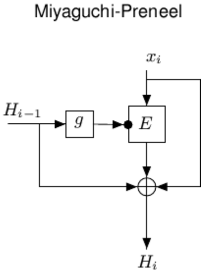
Theorem

If the compression function f is collision resistant, then the obtained hash function h is collision resistant.

Hash functions based on (MDC) block ciphers

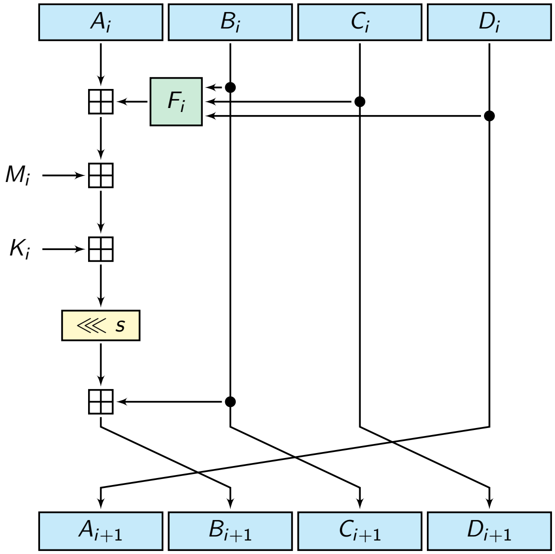






MD5 by Ron Rivest in 1991

For each 512-bit block of plaintext



Ki denotes a 32-bit constant, different for each operation Addition

MD5 by Ron Rivest in 1991

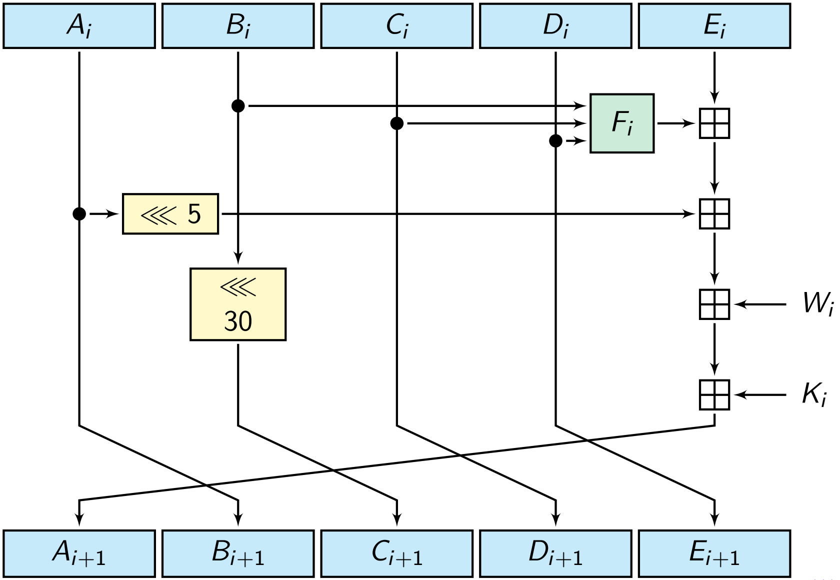
There are four possible functions F; a different one is used in each round:

- $F(B,C,D) = (B \wedge C) \vee (\neg B \wedge D)$
- $G(B,C,D) = (B \wedge D) \vee (C \wedge \neg D)$
- $H(B,C,D) = B \oplus C \oplus D$
- $I(B,C,D) = C \oplus (B \vee \neg D)$

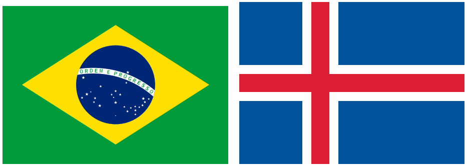
MD5 Cryptanalysis

- In 1993, Den Boer and Bosselaers gave a "pseudo-collision" two different initialization vectors of compression function which produce an identical digest.
- In 1996, Dobbertin announced a collision of the compression function of MD5.
- 17 August 2004, collisions for the full MD5 by Xiaoyun Wang, Dengguo Feng, Xuejia Lai, and Hongbo Yu.
- On 1 March 2005, Arjen Lenstra, Xiaoyun Wang, and Benne de Weger demonstrated construction of two X.509 certificates with different public keys and the same MD5 hash value.
- A few days later, Vlastimil Klima able to construct MD5 collisions in a few hours on a single notebook computer.
- On 18 March 2006, Klima published an algorithm that can find a collision within one minute on a single notebook computer, using a method he calls tunneling.
- On 24 December 2010, Tao Xie and Dengguo Feng announced the first published single-block (512 bit) MD5 collision.

SHA-1



Collision for PDF



SHA1(TP/SHA1/a.pdf)=
a5a678701d8b2ab07c96d101b3331fb4992f0980
SHA1(TP/SHA1/b.pdf)=
a5a678701d8b2ab07c96d101b3331fb4992f0980

List of Hash Functions

Algorithm	Output size	Internal state size	Block size	Length size	Word size	Collision
HAVAL	256/.../128	256	1024	64	32	Yes
MD2	128	384	128	No	8	Almost
MD4	128	128	512	64	32	Yes
MD5	128	128	512	64	32	Yes
PANAMA	256	8736	256	No	32	Yes
RadioGatún	Arbitrarily long	58 words	3 words	No	1-64	No
RIPEMD	128	128	512	64	32	Yes
RIPEMD	128/256	128/256	512	64	32	No
RIPEMD	160/320	160/320	512	64	32	No
SHA-0	160	160	512	64	32	Yes
SHA-1	160	160	512	64	32	With flaws
SHA-256/224	256/224	256	512	64	32	No
SHA-512/384	512/384	512	1024	128	64	No
Tiger(2)	192/160/128	192	512	64	64	No
WHIRLPOOL	512	512	512	256	8	No

SHA-3 Zoo

64 Submissions, 54 selected,

1. * BLAKE Jean-Philippe Aumasson
2. Blue Midnight Wish Svein Johan Knapskog
3. CubeHash Daniel J. Bernstein preimage
4. ECHO Henri Gilbert
5. Fugue Charanjit S. Jutla
6. * Grøstl Lars R. Knudsen
7. Hamsi Özgül Küçk̈
8. * JH Hongjun Wu preimage
9. * Keccak The Keccak Team
10. Luffa Dai Watanabe
11. Shabal Jean-François Misarsky
12. SHAvite-3 Orr Dunkelman
13. SIMD Gaëtan Leurent
14. * Skein Bruce Schneier

SHA-3 = Keccak (sponge + compression)

Authors

- Guido Bertoni (Italy) of STMicroelectronics,
- Joan Daemen (Belgium) of STMicroelectronics,
- Michaël Peeters (Belgium) of NXP Semiconductors, and
- Gilles Van Assche (Belgium) of STMicroelectronics.

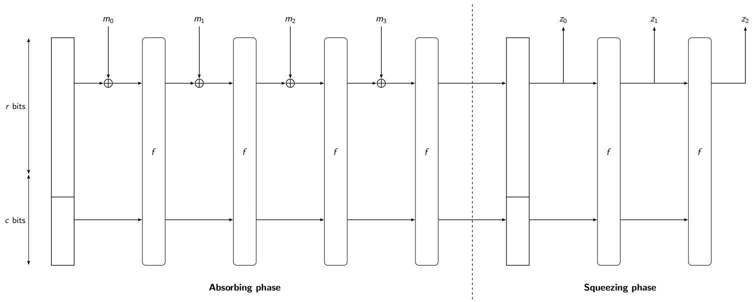
SHA-3 = Keccak

$$h : \{1,0\}^* \to \{1,0\}^n$$

- MD5: $n = 128$ (Ron Rivest, 1992)
- SHA-1: $n = 160$ (NSA, NIST, 1995)
- SHA-2: $n \in \{224, 256, 384, 512\}$ (NSA, NIST, 2001)
- SHA-3: n is arbitrary (NSA, NIST, 2012)

SHA-3 = Keccak is a sponge based hash

$$H(P_0|P_1|\ldots|P_i) = Z_0|Z_1|\ldots|Z_l$$



$b = r + c$

- r bits of rate
- c bits of capacity (security parameter)

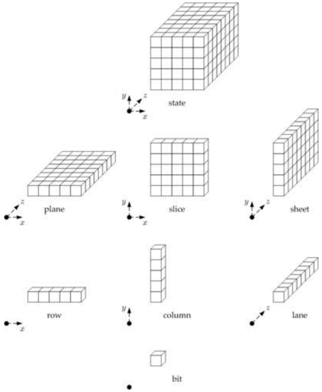
Inside Keccak

- 7 permutations: $b \in \{25, 50, 100, 200, 400, 800, 1600\}$
- ... from toy over lightweight to high-speed ...
- SHA-3 instance: r = 1088 and c = 512
  - permutation width: 1600
  - security strength 256: post-quantum sufficient
- Lightweight instance: r = 40 and c = 160
  - permutation width: 200
  - security strength 80: same as (initially expected from) SHA-1

SHA-3 = Keccak f Setting

Defined for word of size, $w = 2^l$ bits (if $l = 6$ 64-bit words)

State is $5 \times 5 \times w$ array of bits (a[i][j][k])



- state = 5×5 lanes , each containing 2^l bits
- (5×5)-bit slices, 2^l of them

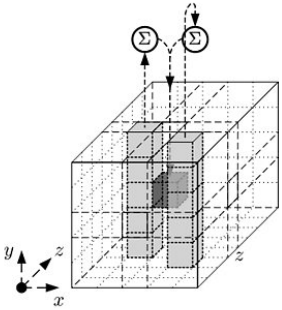
SHA-3 = Keccak

The basic block permutation function consists of $12 + 2 \times l$ iterations of following sub-rounds.

1. step Θ
2. step ρ
3. step π
4. step χ
5. step ι

Keccak Θ

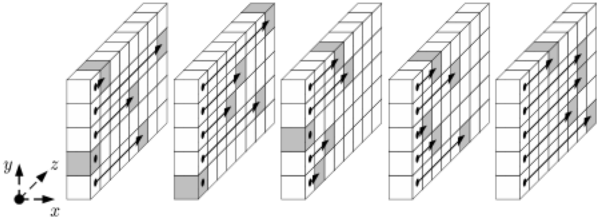
1. Compute the parity of each of the 5-bit columns
2. ⊕ the sum of a[x-1][][z] and of a[x+1][][z-1] into a[x][y][z].



$$a[i][j][k] \oplus = parity(a[0..4][j-1][k]) \oplus parity(a[0..4][j+1][k-1])$$

Keccak ρ

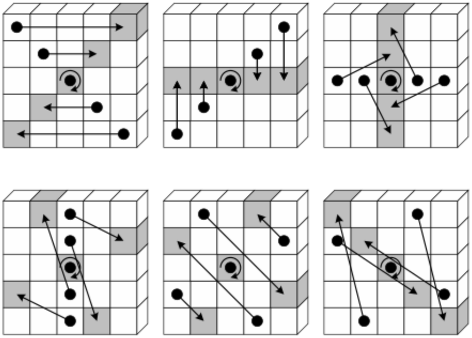
Bitwise rotate each of the 25 words by a different rotation.



$a[0][0]$ is not rotated, and for all $0 \leq t < 24$
$a[i][j][k] = a[i][j][k - (t+1)(t+2)/2]$, where
$\begin{pmatrix} i \\ j \end{pmatrix} = \begin{pmatrix} 3 & 2 \\ 1 & 0 \end{pmatrix}^t \begin{pmatrix} 0 \\ 1 \end{pmatrix}.$

Keccak π

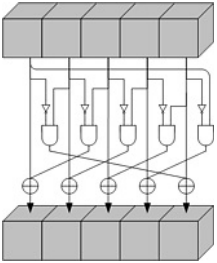
Permute the 25 words in a fixed pattern.



$$a[i][j] = a[j][2i + 3j]$$

Keccak χ

Bitwise combine along rows, using $a = a \oplus (\neg b \& c)$.



$$a[i][j][k] \oplus = \neg a[i][j+1][k] \& a[i][j+2][k]$$

This is the only non-linear operation in SHA-3.

Keccak ι

Exclusive-or a round constant into one word of the state.

- In round n, for $0 \le m \le l$, $a[0][0][2m - 1]$ is exclusive-ORed with bit $m + 7n$ of a degree-8 LFSR (Linear Feedback Shift Register) sequence.

This breaks the symmetry that is preserved by the other sub-rounds.

Hash Functions

A hash function $\mathcal{H}$ takes as input a bit-string and returns a corresponding 'digest' of fixed length. Good hash functions are :

- collision-free: $\mathcal{H}(x) = \mathcal{H}(y) \Rightarrow x = y$
- non-malleable: $x\mathcal{R}y \nRightarrow \mathcal{H}(x)\mathcal{R}\mathcal{H}(y)$

$$\mathcal{H}(Alice) \quad =\neq \quad \mathcal{H}(Bob)$$

Properties of hash functions

Definition (Preimage resistance)

Given an output y, it is computationally infeasible to compute x such that

$$h(x) = y$$

Definition (2nd Preimage resistance)

Given an input x, it is computationally infeasible to compute x' such that

$$h(x') = h(x)$$

Properties of hash functions

Definition (Collision resistance)

It is computationally infeasible to compute x and x' such that

$$h(x) = h(x')$$

Properties of hash functions

Alternate terminology:

- *pre-image resistant* $\equiv$ *one-way*
- *2nd pre-image resistant* $\equiv$ *weak collision resistant*
- *collision resistant* $\equiv$ *strong collision resistant*

Use of hash functions

Idea: compute a condensed message y from a given message m.
The condensed should be specific to the message.

- Use y in place of m in a trustworthy way.
- *"Did you get m correctly? Here's y to check."* (file-sharing)
- *"Could you decrypt c correctly?"*
- *"I sign y to prove that I wrote m."*

List of Hash Functions

Algorithm	Output size	Internal state size	Block size	Length size	Word size	Collision
HAVAL	256/.../128	256	1024	64	32	Yes
MD2	128	384	128	No	8	Almost
MD4	128	128	512	64	32	Yes
MD5	128	128	512	64	32	Yes
PANAMA	256	8736	256	No	32	Yes
RadioGatún	Arbitrarily long	58 words	3 words	No	1-64	No
RIPEMD	128	128	512	64	32	Yes
RIPEMD	128/256	128/256	512	64	32	No
RIPEMD	160/320	160/320	512	64	32	No
SHA-0	160	160	512	64	32	Yes
SHA-1	160	160	512	64	32	With flaws
SHA-256/224	256/224	256	512	64	32	No
SHA-512/384	512/384	512	1024	128	64	No
Tiger(2)	192/160/128	192	512	64	64	No
WHIRLPOOL	512	512	512	256	8	No

SHA-3 Zoo

64 Submissions, 54 selected,

1. * **BLAKE** by Jean-Philippe Aumasson
2. **Blue Midnight Wish** by Svein Johan Knapskog
3. **CubeHash** by Daniel J. Bernstein preimage
4. **ECHO** byHenri Gilbert
5. **Fugue** by Charanjit S. Jutla
6. * **Grøstl** byLars R. Knudsen
7. **Hamsi** by Özgül Küçk̈
8. * **JH** by Hongjun Wu preimage
9. * **Keccak** by The Keccak Team
10. **Luffa** by Dai Watanabe
11. **Shabal** by Jean-François Misarsky
12. **SHAvite-3** by Orr Dunkelman
13. **SIMD** by Gaëtan Leurent
14. * **Skein** by Bruce Schneier

DMAC (CBC-MAC variant)

Example

```
$c_1 := m_1$;
for $i = 2$ to $n$ do:
    $z_i := c_{i-1} \oplus m_i$
    $c_i := E(z_i)$;
$tag := E'(c_n)$;
```

HMAC

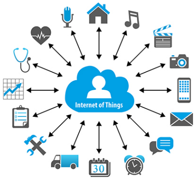
Example

```
$z_1 := k \| m_1;$
$c_1 := \mathcal{H}(z_1);$
for $i = 2$ to $n$ do:;
    $z_i := c_{i-1} \| m_i$
    $c_i := \mathcal{H}(z_i)$
$z' := k' \| c_n;$
$tag := \mathcal{H}(z');$
```

Outline

5 Things to Bring Home

1. Severals **security challenges** in IoT
2. Security has to be taken **at the design** of IoT
3. Designing secure protocols is **difficult**
4. **Tradeoff** between security, battery, CPU and price.
5. Use the **adpated cryptographic primitives**.



Protocol + Properties + Intruder ⇒ Security