# Cryptography:
# Encryption, Hash Functions, Certificates

# Cryptographic Libraries (and APIs)

- C:
  - OpenSSL: https://www.openssl.org
  - Cryptlib: https://www.cryptlib.com
  - NaCL (Networking and Cryptography): https://nacl.cr.yp.to
- Java:
  - JCA/JCE architecture and its implementations (Bouncy Castle: https://www.bouncycastle.org/fr/)
  - JAAS for authentication & authorization – notably for J2EE
  - Apache Commons Crypto (OpenSSL wrapper): https://github.com/apache/commons-crypto
  - jNaCL: https://github.com/neilalexander/jnacl
- Python:
  - NaCL (Networking and Cryptography): https://nacl.cr.yp.to
  - pyOpenSSL: https://www.pyopenssl.org/en/stable/index.html
  - cryptography: https://github.com/pyca/cryptography
- Web:
  - W3C low-level Crypto API (available in browsers): https://www.w3.org/TR/WebCryptoAPI/
  - IETF Javascript Object Signing and Encryption (jose): https://datatracker.ietf.org/wg/jose/about/

# Encryption

- Transforms data (also called plaintexts or cleartexts) into a ciphertext
  - The ciphertext cannot be read by anybody outside those who possess a secret
  - The secret is also called an encryption or decryption key
- Encryption is a mechanism ensuring the confidentiality property
- Kerckhoffs principle (1883):
  - The security of a system must only rely on the ignorance of the key, not that of another parameter
  - No ”security by obscurity”

Enigma Machine

Augustin Kerckhoffs

# Symmetric Key Cryptography


- Encryption and decryption are symmetrical and use the same key
- Main problem of symmetric key encryption: key exchange
  - The keys must be exchanged between each pair of communicating entities
  - In practice, one must also distribute secret keys that will be used without these keys being intercepted by eavesdroppers.

# Encryption - approaches

- Substitution Ciphers
  - The alphabet is shifted (rotation)
  - Exemple: Caesar's cipher (3 shifts)
  - Monoalphabetic Ciphers
    - Each letter is substituted by another chosen arbitrarily
    - Weakness: frequential analysis (frequency of occurrence of letters …)
  - Polyalphabetic substitution Ciphers
    - Permutation of alphabetic symbols
    - Using multiple ciphering alphabets (1 permutation per symbol)
    - The key selects which alphabet is used for each letter
    - Exemple: Vigenère's cipher
- Transposition Ciphers (or Permutation Ciphers)
  - The letter order is modified without changing their value
  - Permutating input symbols
  - The key identifies the permutation
- Product Ciphers
  - Combination of substitutions andpermutations
  - Foundation of modern encryption

# Encryption : Transposition

| 1 | 2 | 3 | 4 |
|---|---|---|---|
| r | e | n | d |
| e | z | v | o |
| u | s | d | e |
| m | a | i | n |
| a | l | u | n |
| i | v | e | r |
| s | i | t | e |

Encryption Key: «3 1 4 2»

First line : «nrde»

Ciphertext:

«nrdeveozduesimnuanleirvtsei»

Source: Jacques Savoy

# Encryption: Substitution

For the « B » letter in the key, a shift of +1 is used in the alphabet.

If the plaintext contains a « R », and the key a « B », the cleartext will be « S »

If the plaintext contains a « E », and the key a « A », the cleartext will remain a « E »

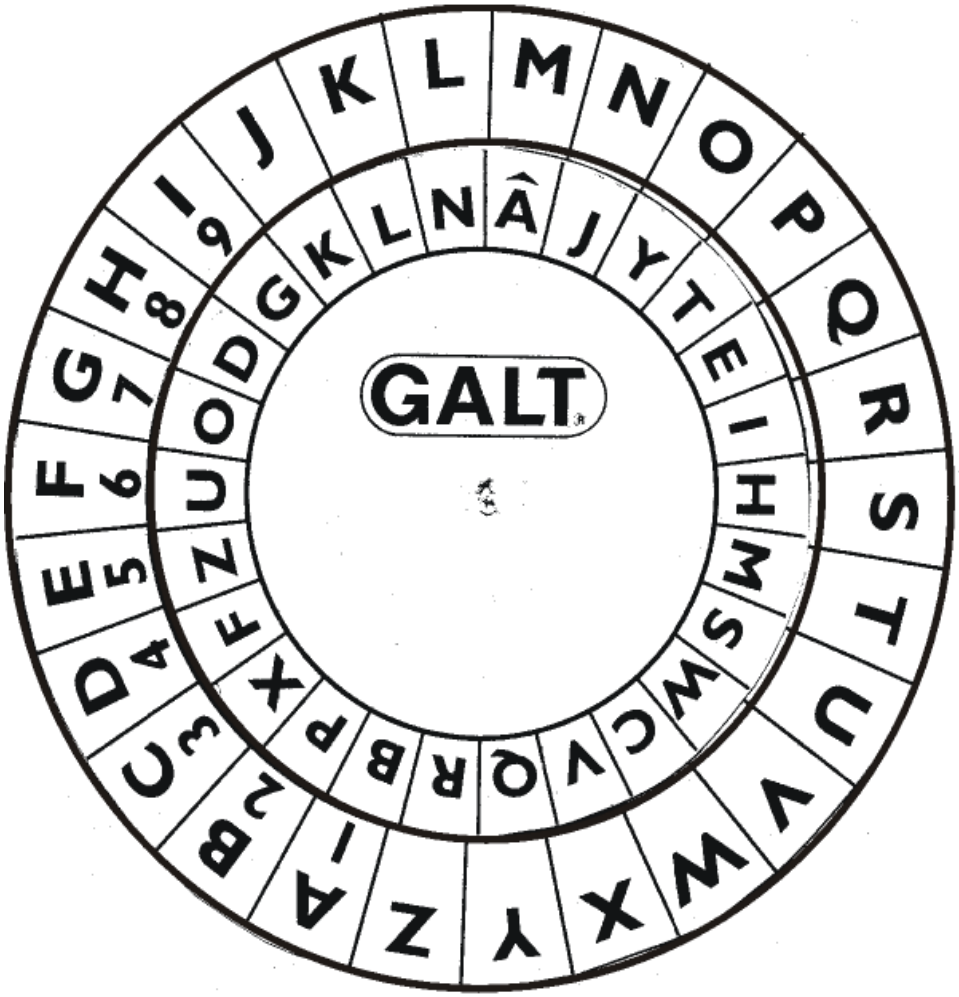
| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| clair = | R | E | N | A | I | S | S | A | N | C | E |
| clé = | **B** | **A** | **N** | **D** | B | A | N | D | B | A | N |
| chiffrement = | S | E | A | D | J | S | F | D | O | C | R |

Blaise de Vigenère

Source: Jacques Savoy

# Encryption: Substitution

More complex and faster substitutions have been supported with hardware




Source: Jacques Savoy

# Block Ciphers

- Confusion :
  - The relationship between plaintext P and ciphertext C statistics must be overly complex to be exploited through cryptanalysis
  - Base Technique: Substitution

- Diffusion :
  - Each symbol of P and/or K must influence several symbols of C
  - The plaintext redundancy must be distributed over a ciphertext
  - Base Technique: Permutation (Transposition)

# DES

In 1973, the *National Bureau of Standards* of the USA launches a call for a cryptographic system.

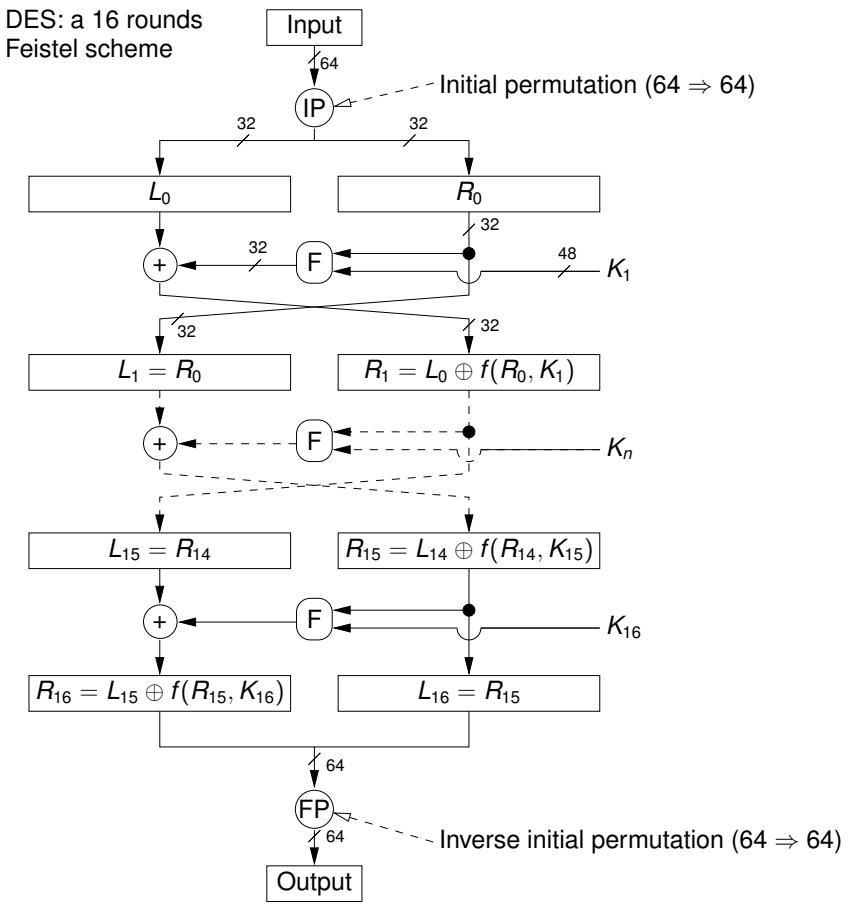
In 1975, the Data Encryption Standard (DES), developed by IBM (under the name « Lucifer ») is adopted.

- 64 bit Bloc Cipher
- 56 bit Key (72 057 594 037 927 936 keys)

# Encryption with DES

- DES - Data Encryption Standard (1976)
  - Feistel scheme (iterative block encryption)
  - Optimised for hardware implementation
  - 56 bit key, vulnerable to exhaustive key search ($2^{56}$ possibilities)
  - A computer with 1024 processors running at 1 GHz can explore all keys in a day
  - DES is no longer secure but is now used as Triple DES (DED or EDE).

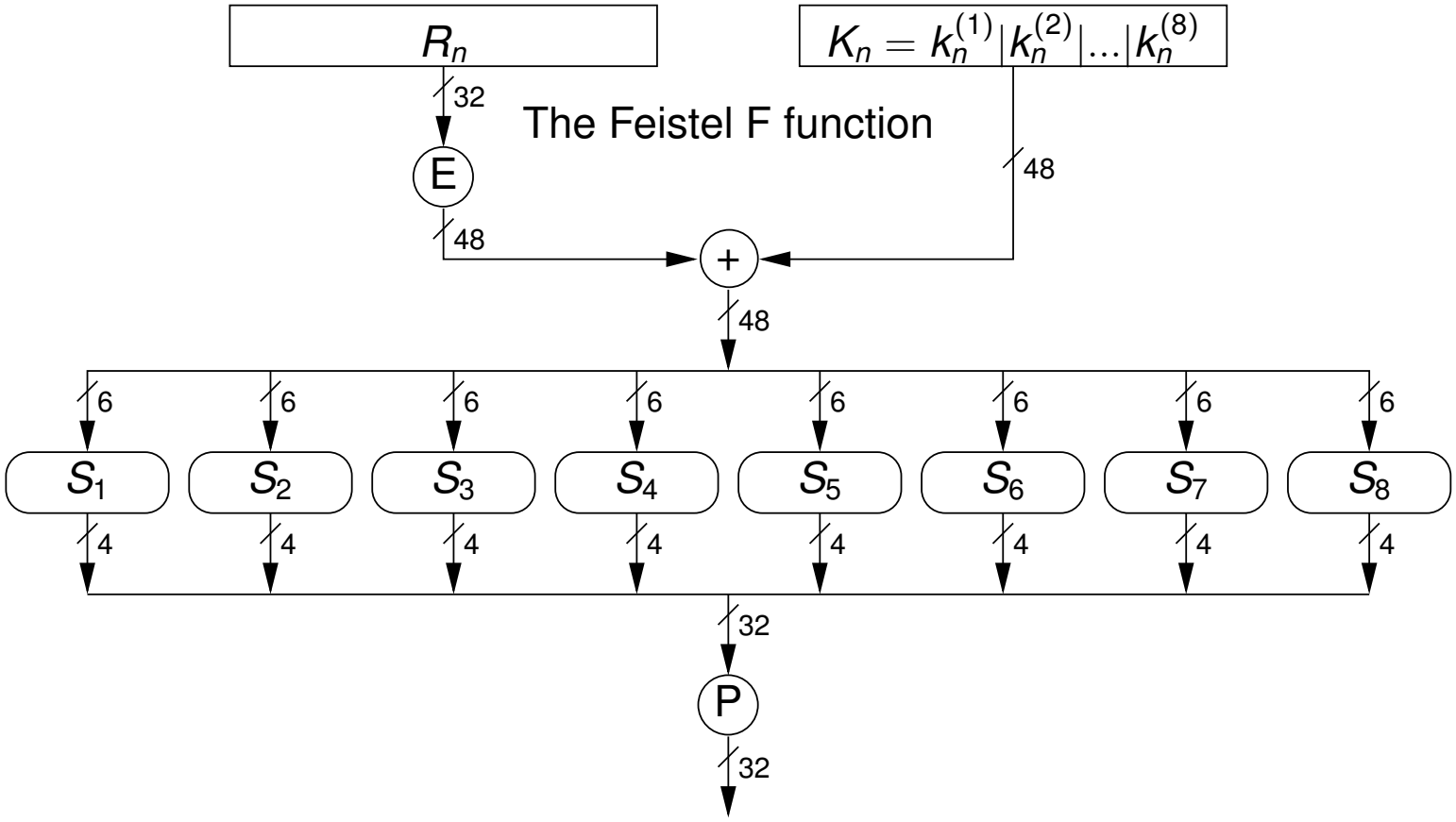
- Newer standard: AES – Advanced Encryption Standard (2000)
  - 128-256 bit keys



# The Feistel *f* function in the DES algorithm

1) 32 bit transposition towards 48 bits
2) Precomputed substitution (stored in tables called S-boxes)

S-Box design (pre-computed substitutions) was wildly discussed



# Chaining Modes

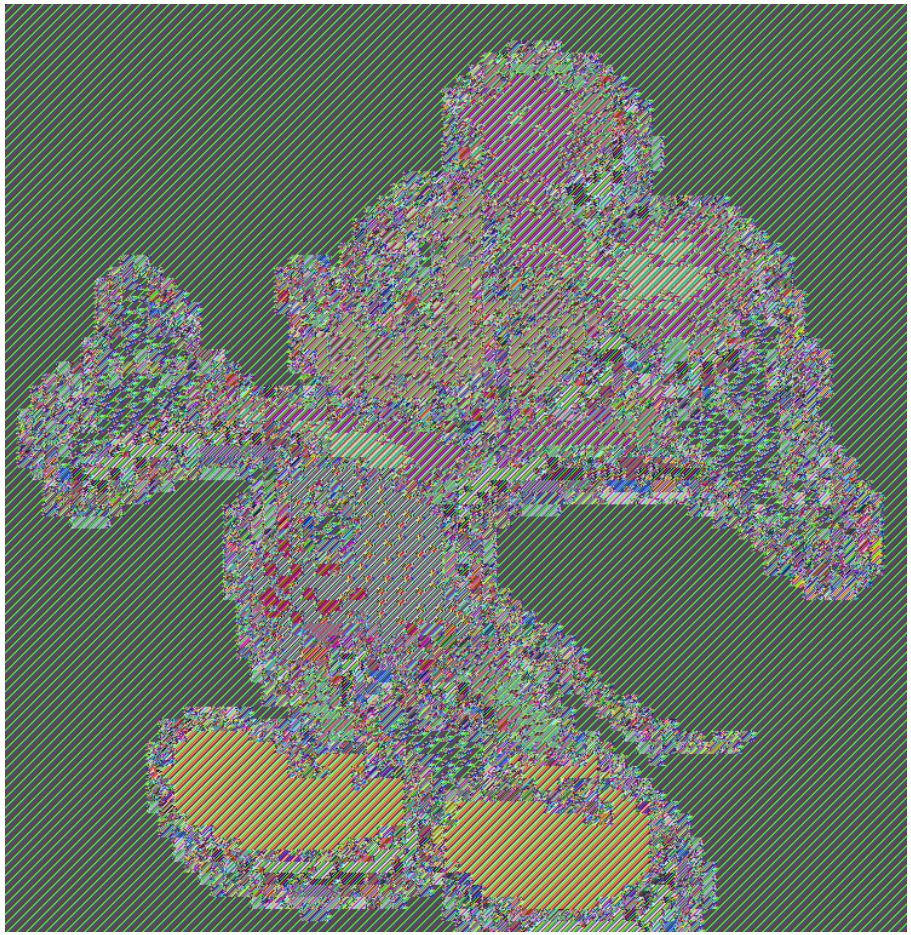
- Electronic Code Book (ECB) **– Statistical Attacks** against DES in ECB mode !

- Cipher Block Chaining (CBC)
- Counter (CTR)
- Cipher Feedback (CFB)

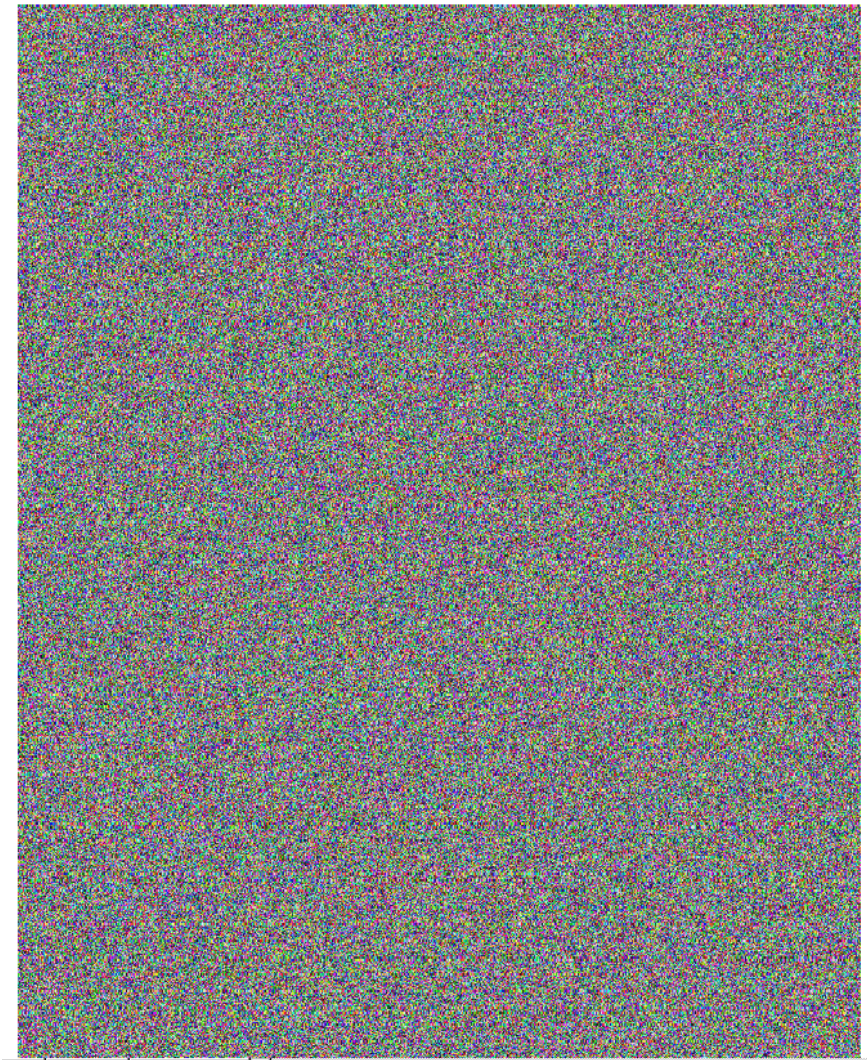
**Chaining blocks introduces problems with respect to error propagation and resynchronization …**

# Plaintext

# DES-ECB Encryption



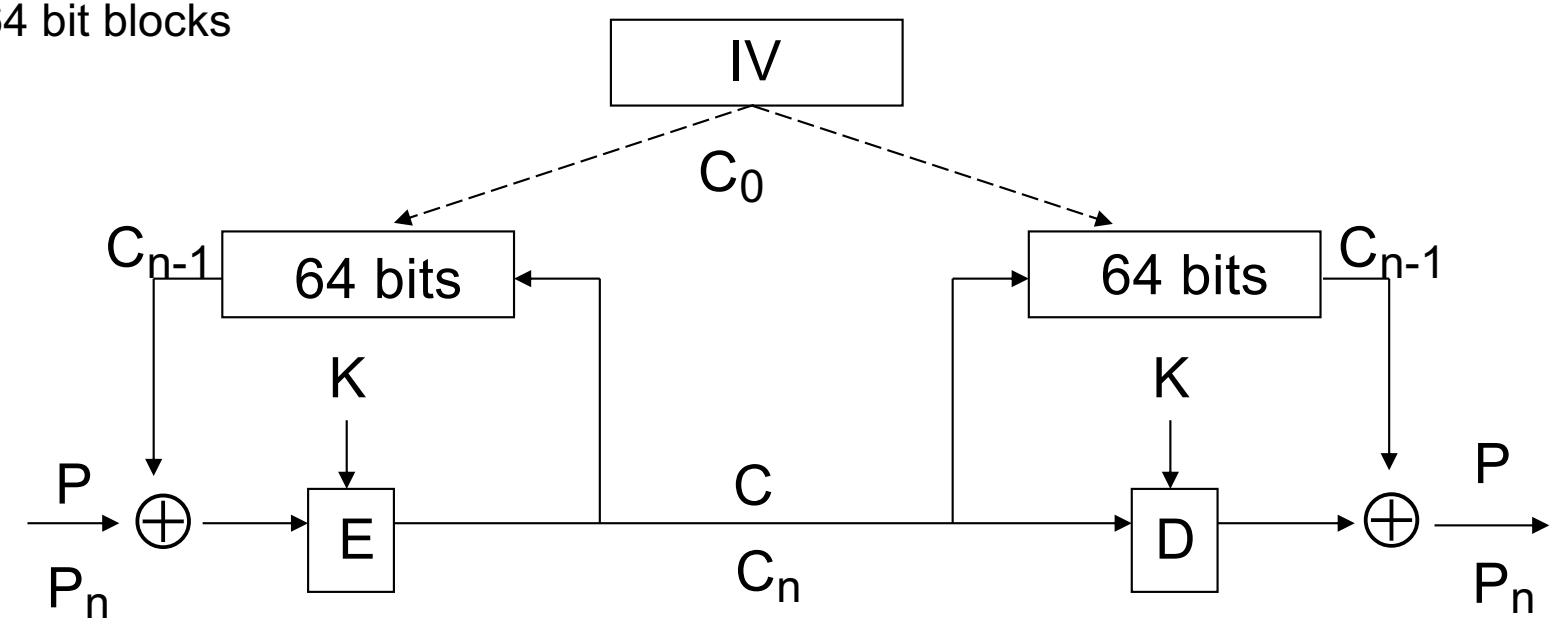
# DES-CBC Encryption



# CBC Mode

Cipher Block Chaining

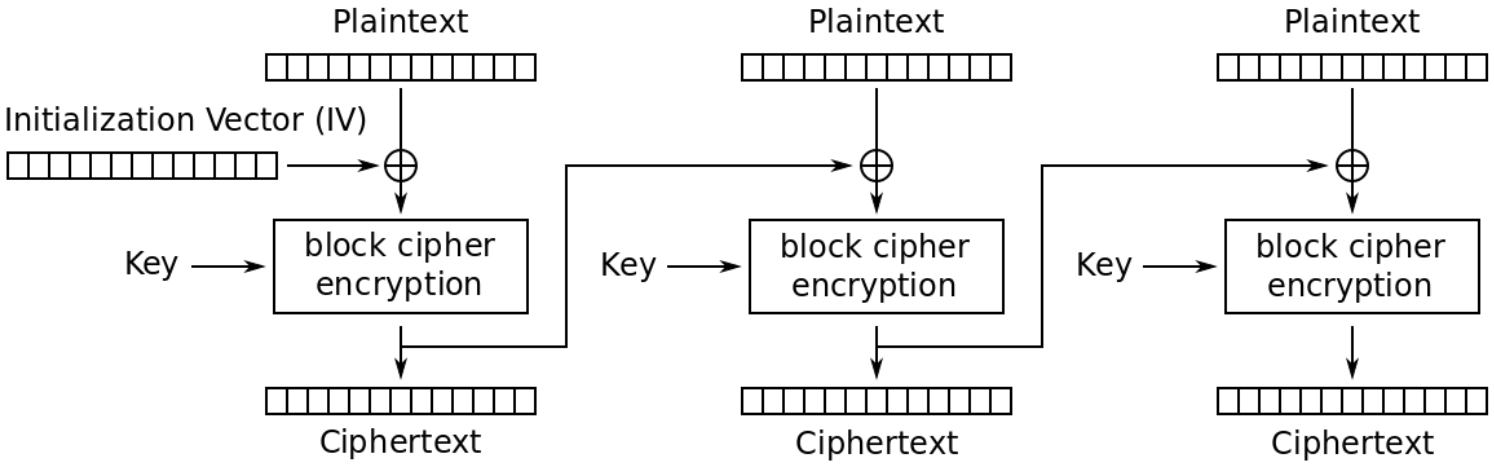
64 bit blocks



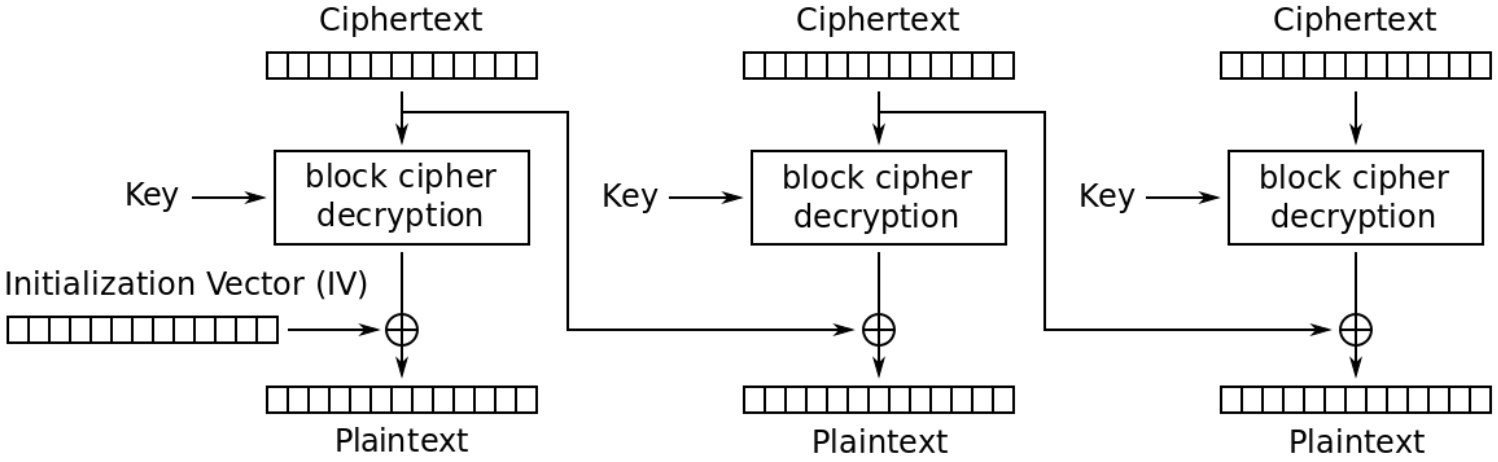

- $C_i = E_K(P_i \oplus C_{i-1})$
- $C_0 = E_K(P_0 \oplus IV)$, IV (Initialization Vector) transmitted in clear
- $P_i = D_K(C_i) \oplus C_{i-1}$

- chaining effect : $C_i$ depends on all $P_j$ with $j \leq i$
- last block in C : depends on all cleartext blocks
- converts DES into a stream cipher
- 1 encryption / decryption operation per 64 bits

# Cipher Block Chaining (CBC) Mode

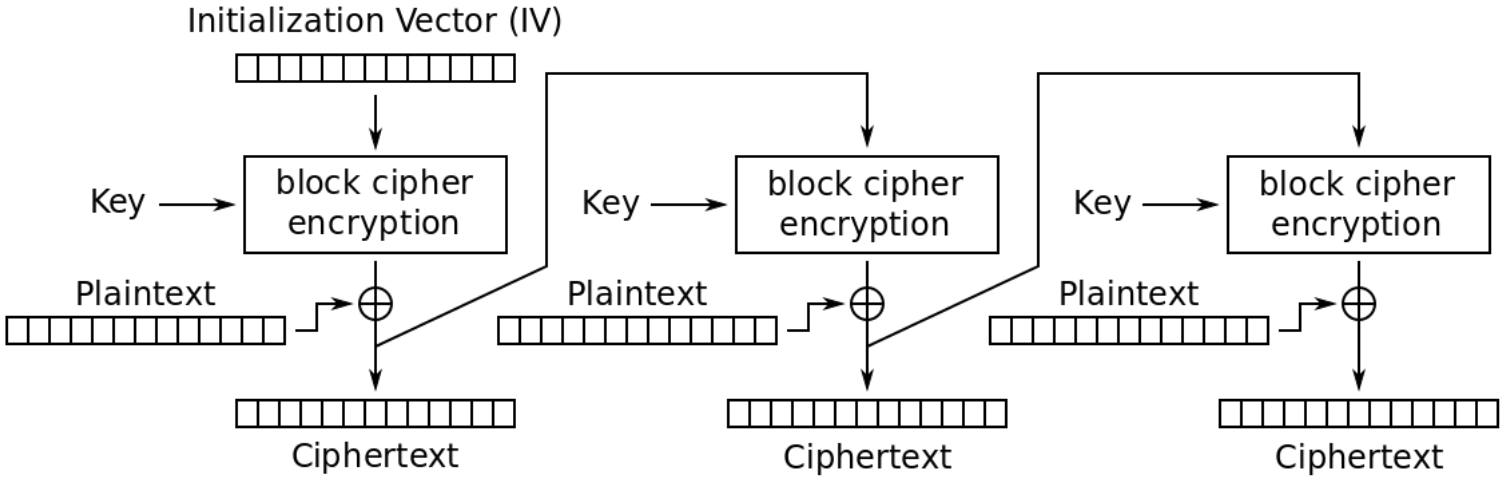



Cipher Block Chaining (CBC) mode encryption

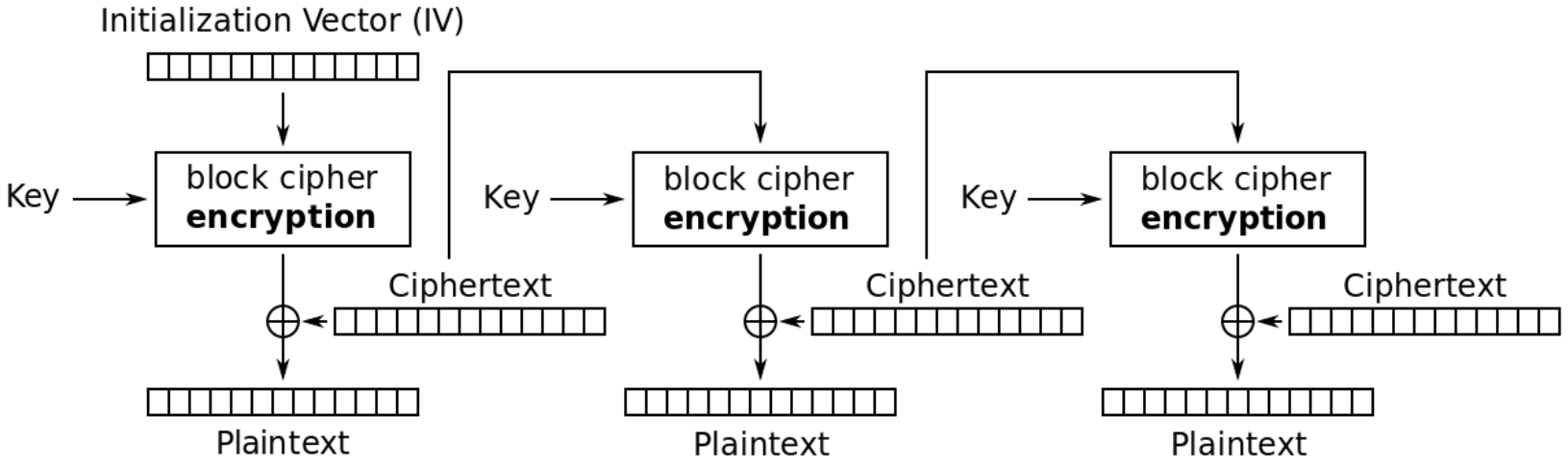



Cipher Block Chaining (CBC) mode decryption

# Cipher Feedback (CFB) Mode

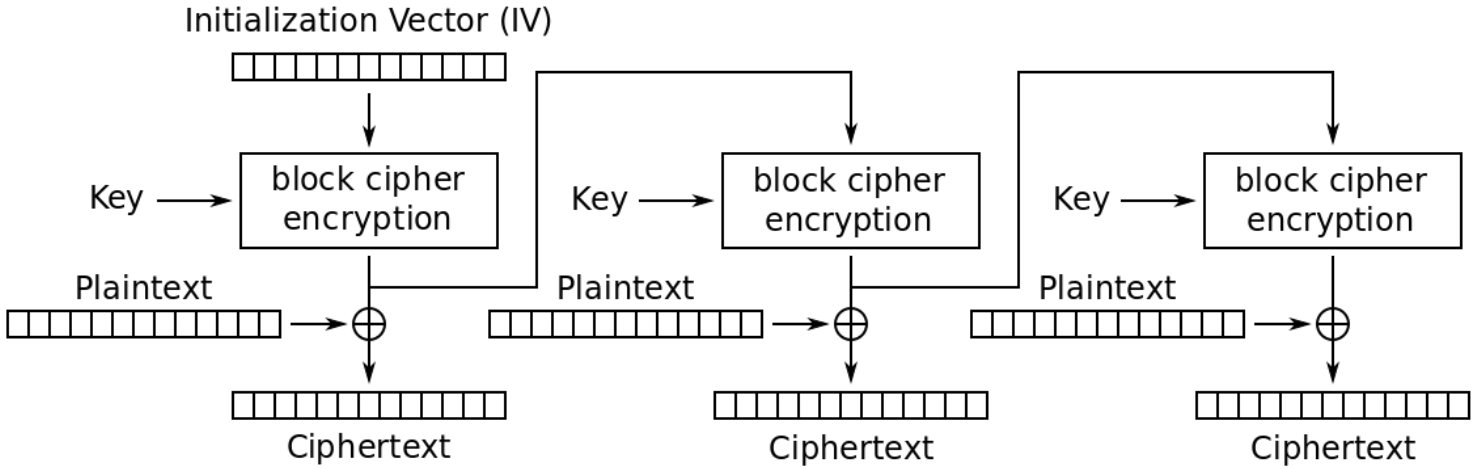



Cipher Feedback (CFB) mode encryption

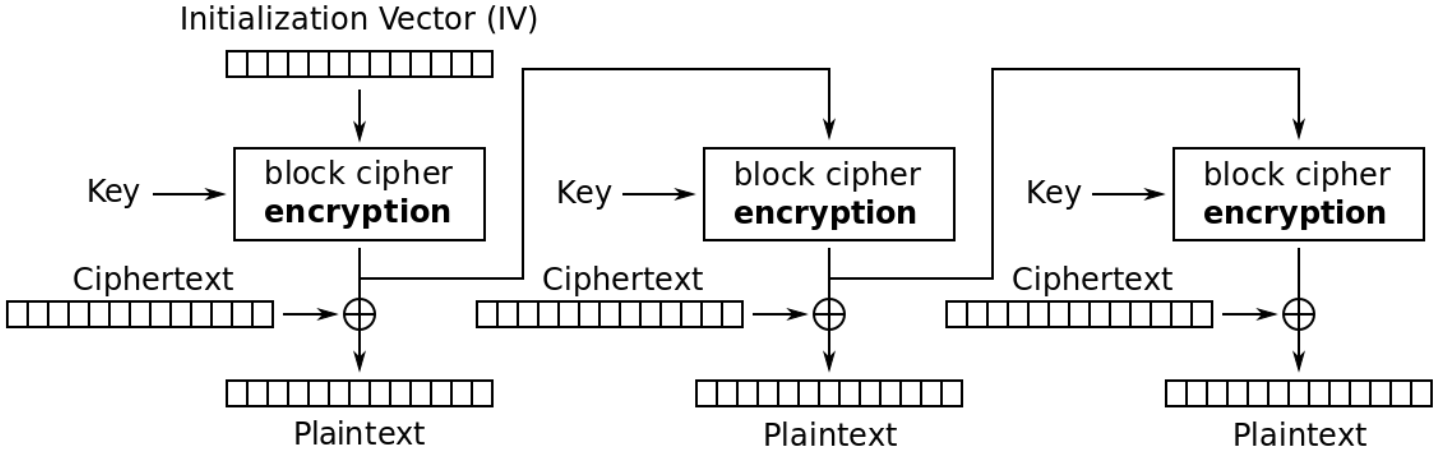



Cipher Feedback (CFB) mode decryption

# Output Feedback (OFB) Mode

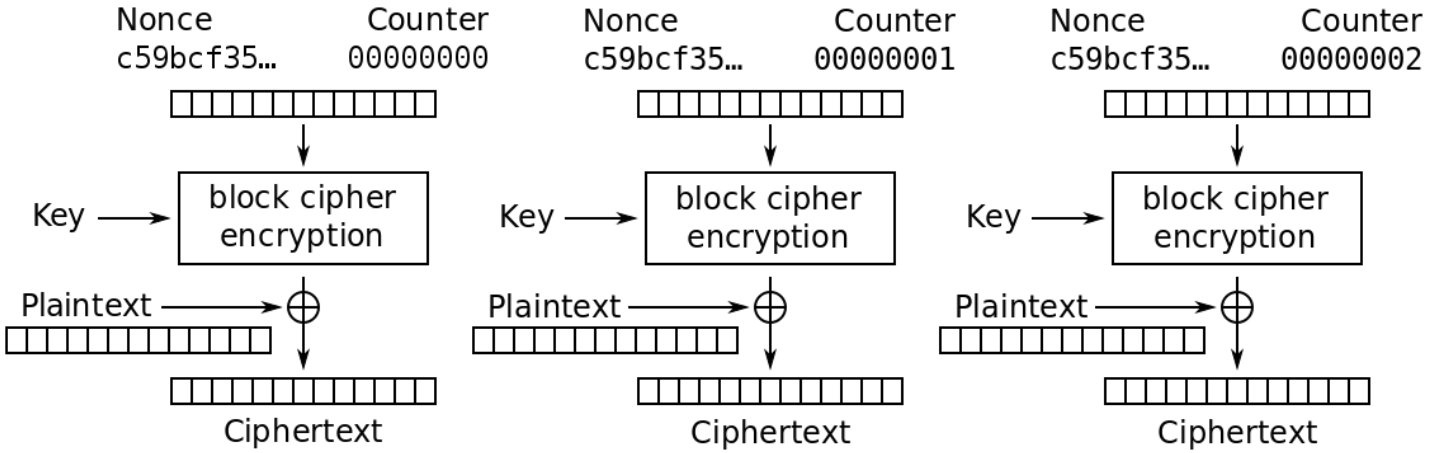



Output Feedback (OFB) mode encryption

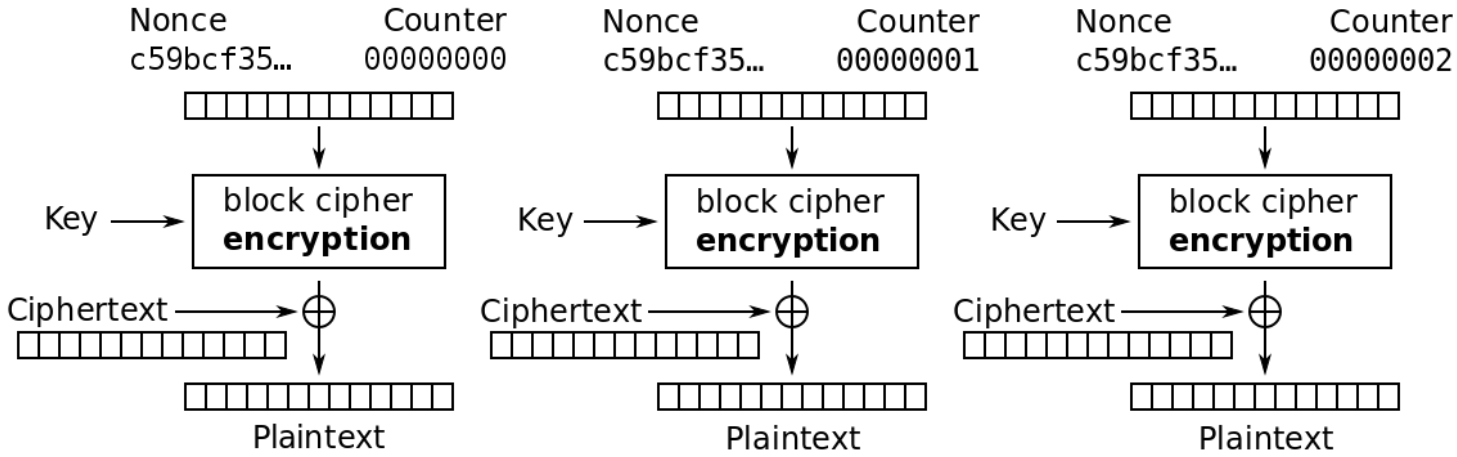



Output Feedback (OFB) mode decryption

# Counter (CTR) Mode

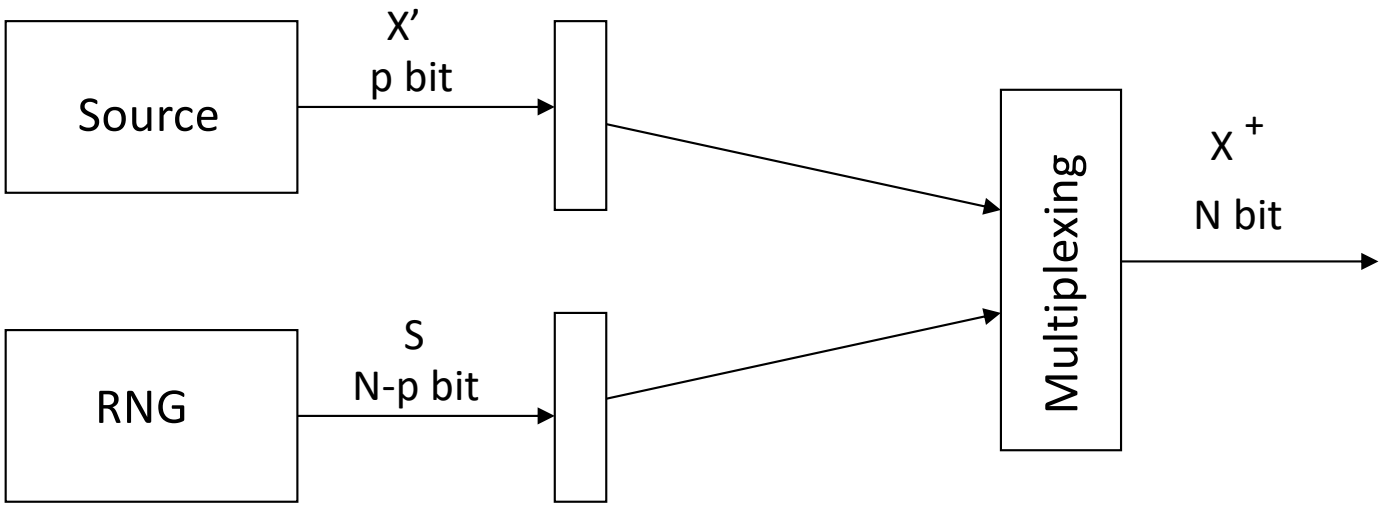



Counter (CTR) mode encryption




Counter (CTR) mode decryption

# Padding




- Two applications:
  - fighting cryptanalysis (e.g., frequency analysis)
  - Adapting cleartext to algorithms (input format size, e.g., 64 bits in DES)
- Example
  - Encryption Algorithme : 56-bit DES key, H(K) = 56 bits
  - Cleartext: 8-bit English coded in ASCII, r= 6,7 bits/character
  - Padding of every source bit with 63 random bits, p=1
- Standards:
  - PKCS#5 and #7 (Public Key Cryptography Standard), PKCS#1 - RSA-OAEP (Optimal Asymmetric Encryption Padding), zero padding for hashes (ISO/EIC 10118-1) …

# Types of attacks

- ***Ciphertext only attacks***:
  the attacker knows C with $C=E_K(P)$
  objectives: find P and K
- ***Known cleartext attacks***:
  the attacker knows $(P_i , C_i)$ with $C_i = E_K(P_i )$
  objectives: find K
- ***Chosen cleartext attacks***:
  the attacker can obtain $C_i$ starting from a chosen $P_i$
  objectives : find K
  the attacks can be adaptive
- ***Chosen ciphertext attacks***:
  the attacker can obtain $P_i$ for a chosen $C_i$
  objectives : find K
  the attacks can also be adaptive

# Security Evaluation

- **Unconditional Security (Perfect Secrecy)** = the system is secure against an attacker with an unlimited amount of time or resources
  - so: is there enough information to compromise the system security?

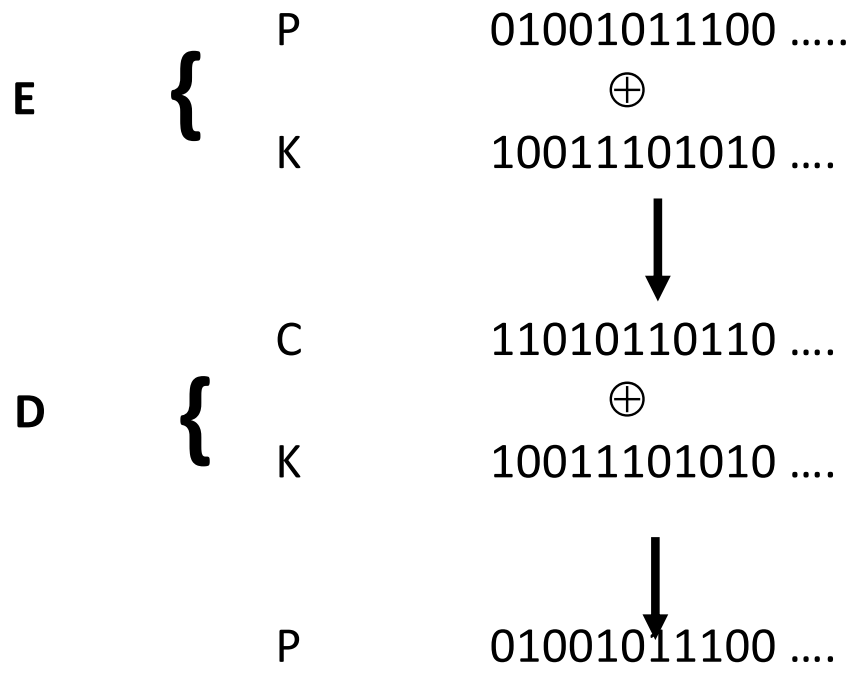
- **Security through theoretical complexity** = proving that the system is secure against an adversary with polynomial capability.

- **Provable Security** = proving that compromising the system security amounts to solving a problem known as “difficult” (e.g., discrete logarithm, factoring big integers).

- **Computational Security** (practical security) = the system is secure against an attacker with a given time and resources.

# One-time pad: perfect secrecy

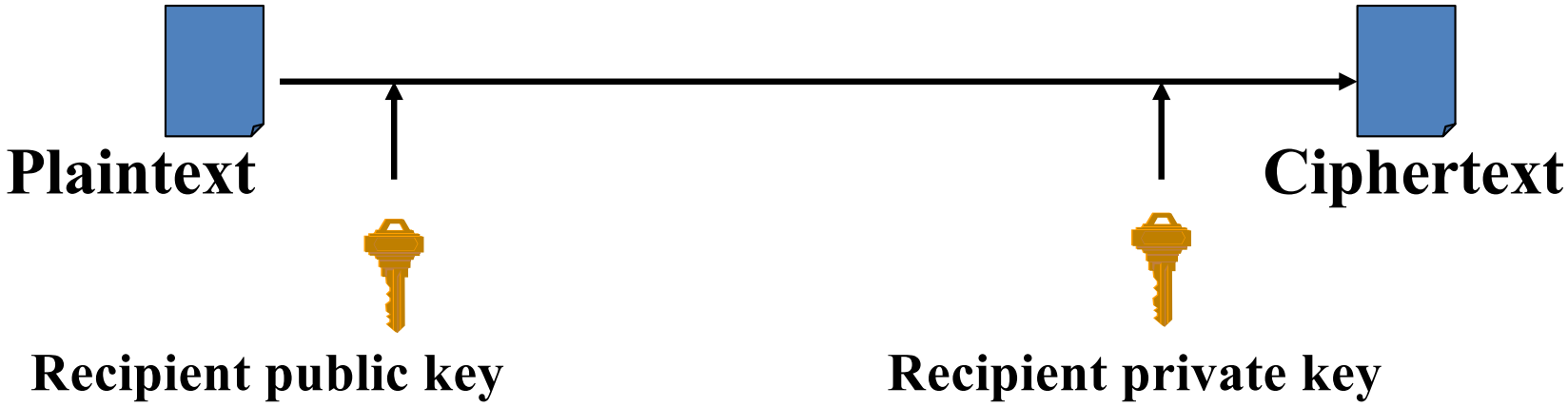
***Vernam Cipher***




Gilbert Vernam

- Additive group operation
- Implementation using exclusive or (XOR)

# Asymmetric Key Cryptography

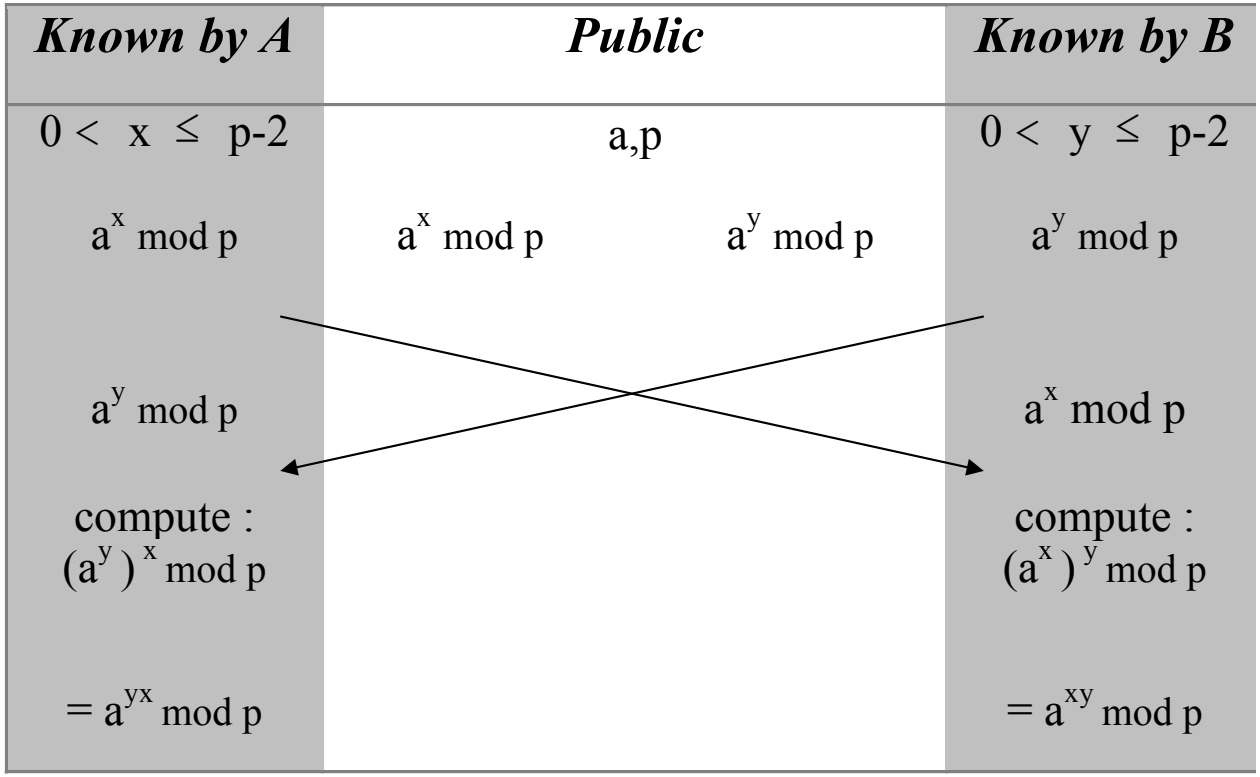



- Encryption based on the difficulty to find the inverse solution to a mathematical problem
  - Much more costly than symmetric key encryption

- Diffie-Hellman (1977)
  - Secret sharing : $(g^a)^b=(g^b)^a)=g^{ab} \bmod p$
- RSA : Rivest-Shamir-Adleman (1978)
  - Standard solution today
  - $E_{KP}(P)=M^e \bmod n$ et $D_{KS}(C)=M^d \bmod n$
  - Les clés KP=(e,n) et KS=(d,n) are connected by a mathematical relationship
- Elliptic curves: the new breed

# Diffie-Hellman algorithm

*p* is a large prime, *a* a primitive element of $Z_p^*$




- A and B establish a shared secret ($a^{xy} \bmod p$) without exchanging any secret information
- $a^{xy} \bmod p$ may be used as a secret key with a symmetric key algorithm in order to encrypt data
- relies on the hardness to compute the discrete logarithm

Adapté d'après R. Molva

# Public key encryption

The algorithm on an operation whose inverse computation is hard

Modulus:

| X | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| $3^x$ | 3 | 9 | 27 | 81 | 243 | 729 |
| $3^x \bmod 7$ | 3 | 2 | 6 | 4 | 5 | 1 |

# RSA: Inversibility

**Euler Function**

For an integer n, $z = \phi(n)$ is the number of primes with n.

- if n is prime $\phi(n) = n-1$
- if n = p.q with p and q prime

$$\phi(n) = (p-1)(q-1)$$

**Euler Theorem**

If a and n are respectivemy prime

$$a^{\phi(n)} \bmod n = 1$$

**Why RSA works**

$$D_K(E_k(M)) = ((M)^e \bmod n)^d \bmod n$$
$$= (M^e)^d \bmod n = M^{e.d} \bmod n$$

But we chose $e.d = 1 \bmod z$

Thus there exists an integer j such that $e.d = j\,z + 1$

$M^{e.d} = M^{j.z}\ M \bmod n = M \bmod n$

According to Euler theorem:

$M^{j.z} \bmod n = (M^z)^j \bmod n = (1)^j = 1$

# Example (B. Schneier)

1) Given two primes p =47,q =71

$n = p.q = 3337$

2) $z = (p-1)(q-1) = 46 \cdot 70 = 3220$

Let us chose e = 79 (prime with n)

3) Computing the inverse of e modulo z

A possible solution: Euler theorem

$e^{\phi(n)} = e\, e^{\phi(n)-1} \bmod z = 1 \bmod z$

Thus : $d = e^{-1} = e^{\phi(n)-1} \bmod z$

Numerically : $79^{78} \pmod{3220} = 1019$

4) To encrypt M = 6882326879666683

Let us decompose M into blocks whose value is less than n= 3337

=> 3 digit blocks

M= 688 232 687 966 668 3

Let us encrypt 688: $688^{79} \bmod 3337 = 1570$

E(M) = 1570 2756 2091 2276 2423 158

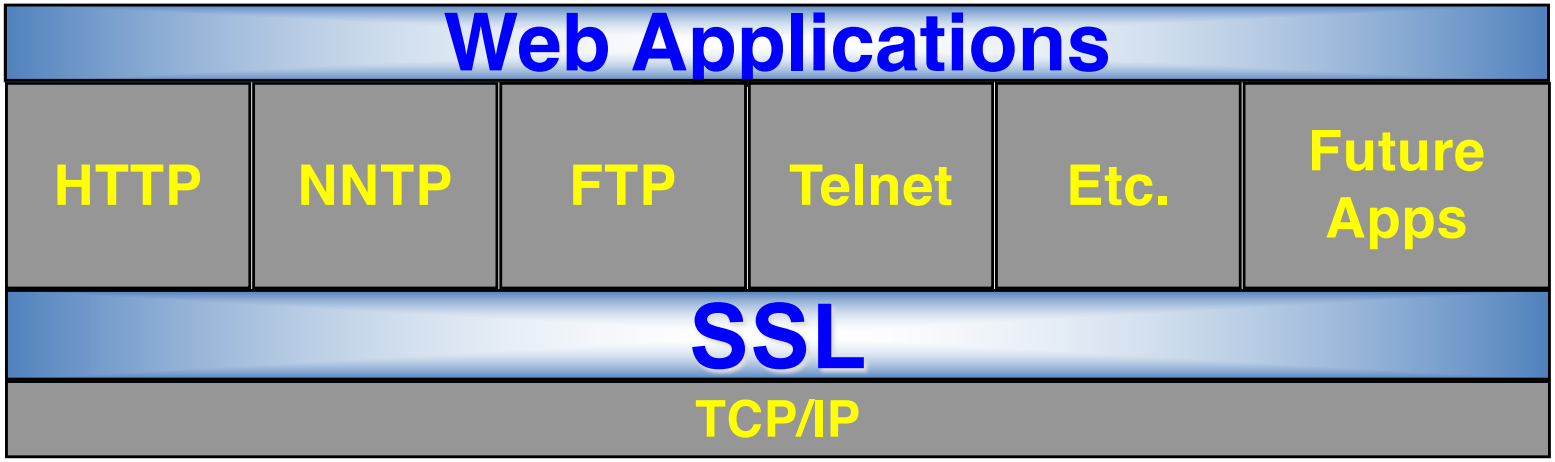
Let us decrypt 1570: $1570^{1019} \bmod 3337 = 688$

# RSA: implementation weaknesses

- Never use a value overly small for *n*,
- Never use an overly short private key
- Use only strong keys such that p-1 and q-1 have a large prime factor
- Do not encrypt overly short blocks (always complete them to n-1 bits, so as to destroy any syntaxic structure
- Do not use a n factor common to several keys if those keys can be used to encrypt the same message
- If a private key *(d,n)* is compromised, do not use the other keys using *n* as a modulus
- Never encrypt or authenticate a message from a third party without modifying it (by adding a few random bytes for instance

# Examples of cryptographic protocol deployments

- Secure transport protocols:
  - SSL/TLS (Secure Socket Layer / Transport Layer Security) – independent from applications
    - Services: server authentication / client authentication / integrity (check values) / confidentiality (encryption)




- Secure Payloads:
  - S-HTTP: individual encryption of HTTP messages
    - Services: authentication, integrity, confidentiality + digital signatures (adds non-repudiation)
  - S/MIME (Secure Internet Multipurpose Extension)

# Hash Functions

- A hash function h is a function which associates to a message M of any length a message h(M) (also denoted as $\{M\}^h$) of a constant (generally short) length.
- While M can be arbitrarily large while $\{M\}^h$ has a given length

Example: file system hashcodes, error detection codes

# Cryptographic hash functions

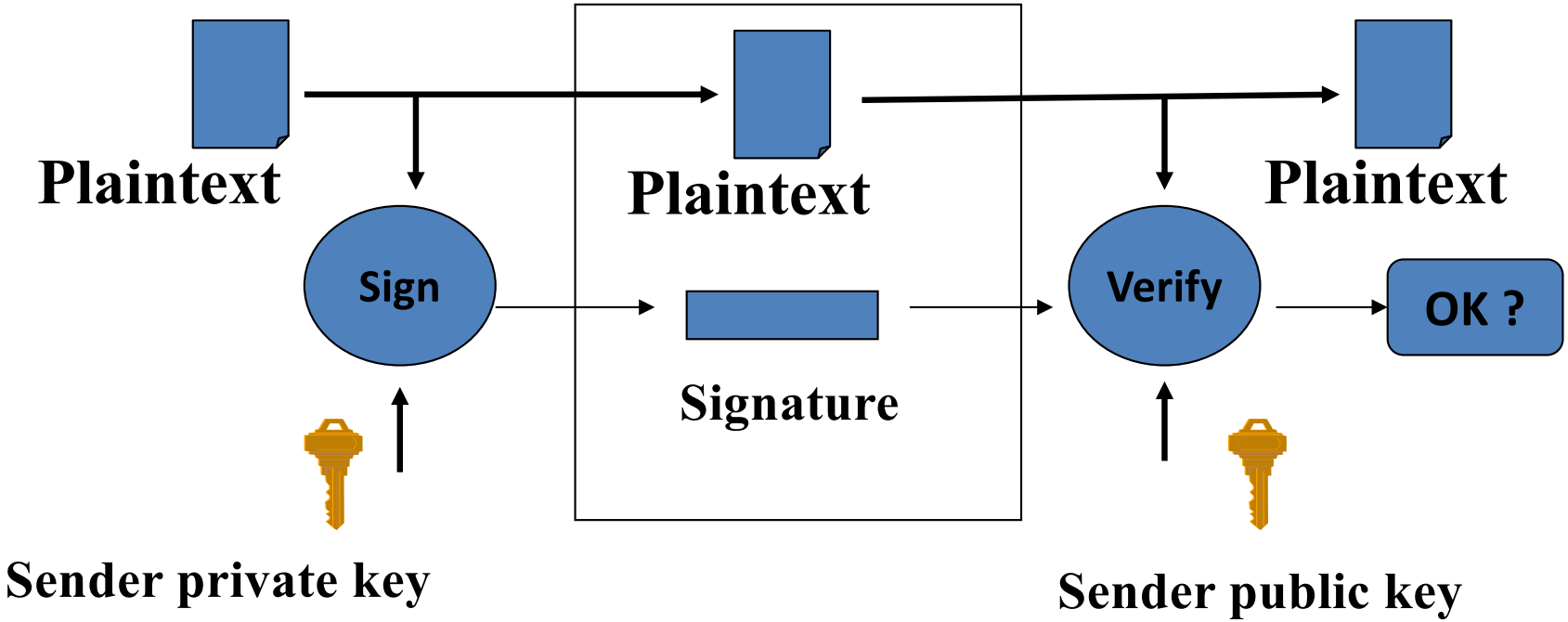
Notable usage: integrity protection, password « storage »

MD5 (don't use !), SHA-1 (don't use!), SHA-256, bcrypt, scrypt, Argon2, …

- 3 properties:
- Preimage resistance: given K, it is computationally hard to find M such that h(M)=K
- 2nd Preimage resistance: given M, it is computationally hard to find M' distinct from M such that h(M)=h(M')=K
- Collision-free: it is computationally hard to find M and M' distinct from M such that h(M)=h(M')
    - Birthday attacks …

- It is keyed if its computation depends on a secret information (termed MAC – Message Authentication Code)

# Digital Signature




- Comes with a message (which can be encrypted or in cleartext)
- Ensures:
  - The authentication of the origin of a message
  - The protection of the integrity of a message
  - The non repudiation of a message

# Signature using a public key algorithm

E, D : public key algorithm

Generating the signature of A over message M :

$$S = E_{KSa}(h(M))$$

Verifying the signature :

- compute h(M)
- verify whether $D_{KPa}(S) = h(M)$

# Public Key Infrastructure (PKI)

- Public key certificates
  - Certifies the association of a name and a key
  - Some certificates can also reference authorizations (permissions)
- Foundational to today's secure and commercial Internet
- Assurances over identity, not trust!
  - Trust = external knowledge
- Example of usage: Thunderbird + Enigmail

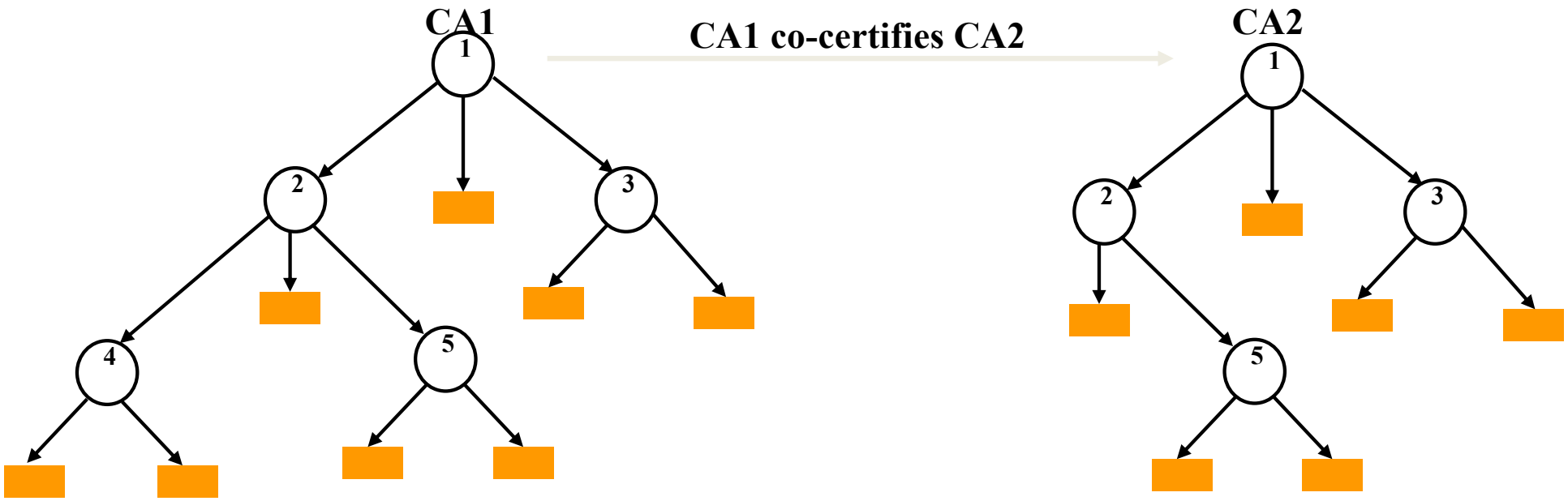
# Certificates

- Introduced in 1978 [Kohnfelder]
- Certificates ≠ Signature
  - Certificats are *implemented using* signatures
- Certificates ≠ Authentication / Authorizations
  - Authentication (resp. authorizations) *can be implemente with* certificates

- X.509 – the current "standard"
  - v.1 (1988) and v.2 – not extensible
  - v.3 (1997) current standard – optional extensions
  - Numerous other proposals extend X.509
- Other standards
  - PGP (notably OpenPGP), SPKI, etc.

# Trust Models in X.509

- Hierarchical Infrastructure
- Possibility to certify between 2 CAs belonging to different trees (co-certification)



# Countermeasures and More Vulnerabilities

- Countermeasures can lead to new vulnerabilities
  - E.g., if we only allow three incorrect logins, as a countermeasure to brute-force attacks (account is frozen), which new vulnerability do we introduce?
    - **Denial of Service attack**
  - If a countermeasure relies on new software, bugs in this new software may mean
    - that it is ineffective, or
    - worse still, that it introduces more weaknesses
    - E.g., Witty worm appeared in Mar 2004 exploited ISS security software
      - http://en.wikipedia.org/wiki/Witty_%28computer_worm%29

# Caveat: insecurities in cryptography

- SSH was meant to provide security, namely as countermeasure against eavesdropping on network, but is a source of new vulnerabilities

- Cryptography is not the cause of these vulnerabilities, and could not solve/prevent these vulnerabilities
  - Protocol, implementation errors (e.g., WEP in WiFi)
  - Programming errors (buffer overflow)
  - Distribution errors (trojan)

- Bruce Schneier: *"Currently encryption is the strongest link we have. Everything else is worse: software, networks, people. There's absolutely no value in taking the strongest link and making it even stronger"*

# Authentication / Identification / Platform Integrity

# Authentication

- What I know: passwords, secret questions, secret keys
- What I own: smartcard, bank card, RSA key, mobile phone
- What I am: biometrics – physiology (fingerprints, iris, veins, face, voice …) or behavior (signature, gestures …)

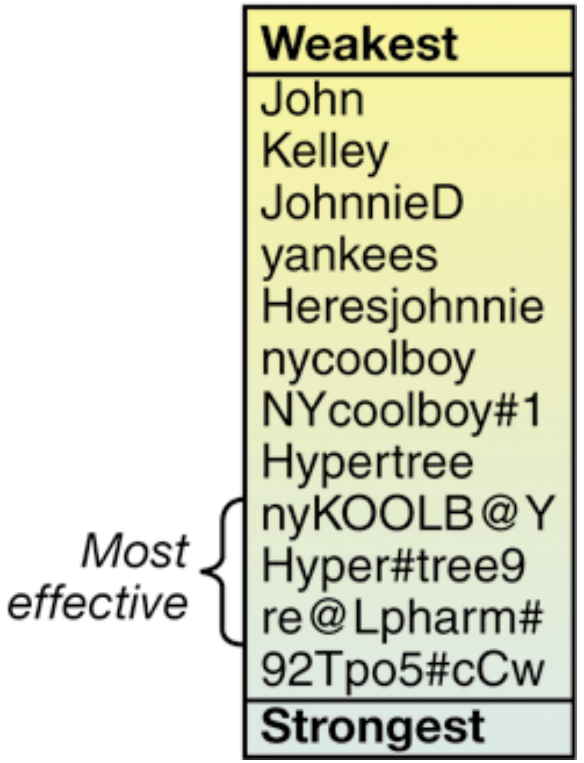
- Increasing association of two mechanims
  - two-factor authentication from different categories

# Passwords

- Strictly personal
- Hard to find, easy to remember (no need to write it down)
  - Minimum number of characters (including ; / ! % ....)
  - Does not correspond to a dictionary word
- Avoid
  - Names or first names of relatives
  - Telephone number…
- Must be changed periodically
- Key phrase method
  - Saying, movie title….

**FIGURE 11.15 • Passwords, weakest to strongest**

Strong passwords use words that are unrelated to your interests and include upper- and lower-case letters, numbers, and symbols.

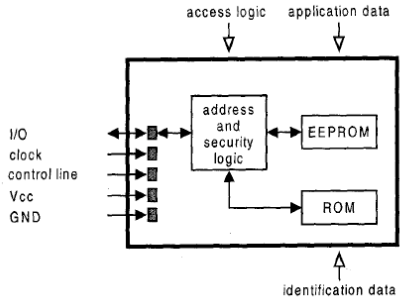


# Smartcard

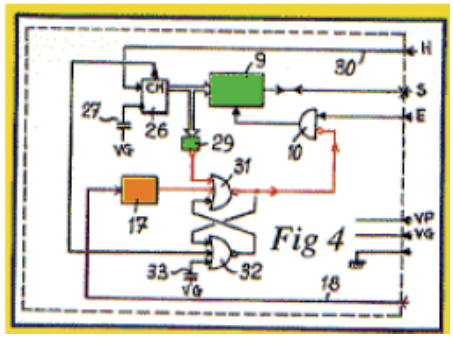
- Smart card, IC card, chip card, pin card …
- Main features:
  - Portable object storing data and/or procedures.
  - Secure object
  - Prevents reading data stored in the card memory (secret keys…)
  - Code executed in a trusted space
  - Low cost object customizable for hundreds of millions of users.
    - 1-5$ for SPOM / 0,1-0,5$ for magnetic cards
  - Can’t work alone and requires
    - A card reader to deliver energy
    - *A clock*
    - *A communication link*

# The card and its technical features

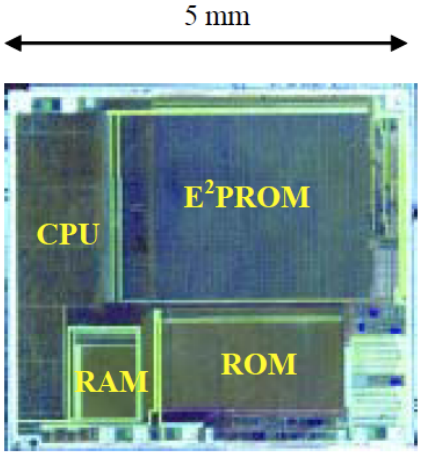
- Memory card:
  - Simple memory (reading / writing) (EPROM / EEPROM)
  - Not standardised


- Cabled logic Card :
  - The card contains a cabled device for protecting the data
  - Dedicated electronics to connect input and output pins
  - Not standardised




- Microprocessor card (SPOM/MAM) :
  - Memory + processor → programmable
  - Security algorithms (ex: DES, RSA)
  - ISO 7816 standards
  - Contact-based or contactless card
  - 1st implementation Bull CP8: 36b of RAM, 1 Kb EPROM, 1,6 Kb ROM



# Biometrics

- Goals:
  - The user does not risk losing his authentication mechanism
  - Some authentification modes are well intégrated – ex: fingerprints (iPAQ, iPhone 5s)


- Problems:
  - Biometric factor theft: never use 1-factor identification!
  - Sometimes easy counterfeating (fingerprints…)
  - The data should be preserved under the control of their owner (ex: smartcard)

**FIGURE 11.17 • Facial recognition**

By taking measurements of 128 facial features and matching them to the measurements of known faces, biometric software can help identify people.

# SW Platform Security: Java

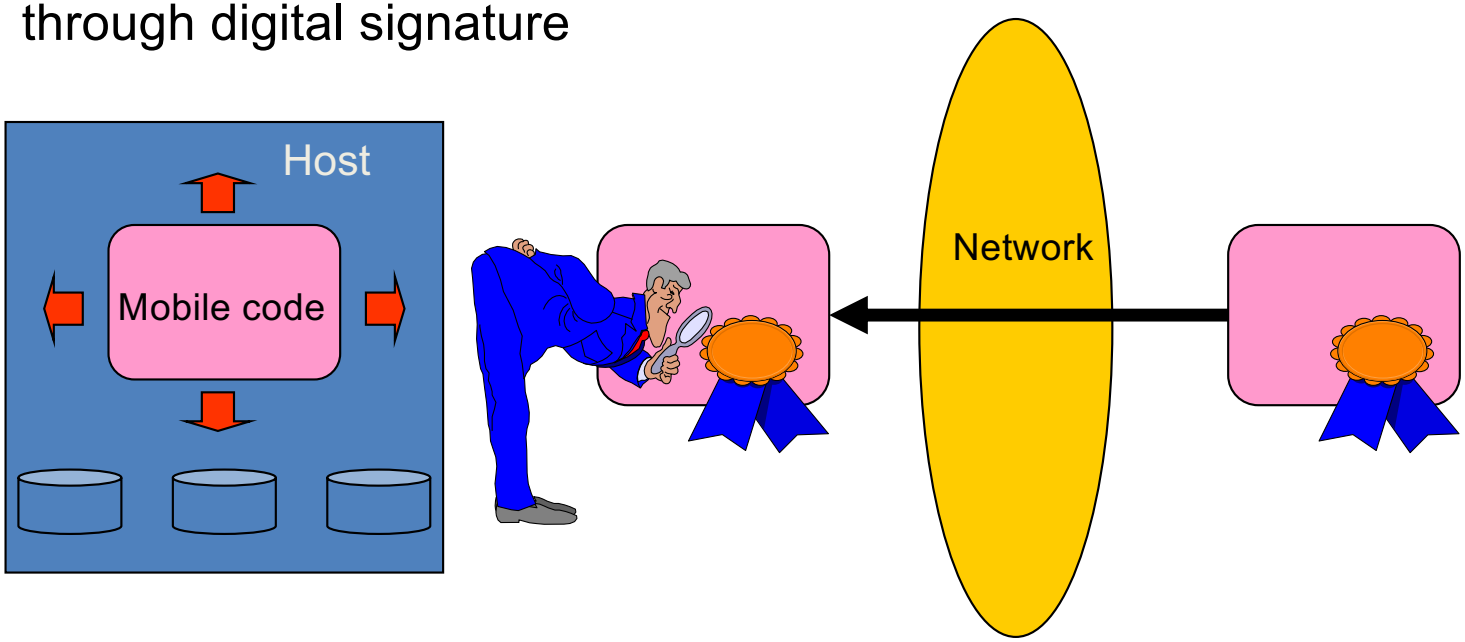
- Host and Mobile code bear separate identities
  - Authentication of the mobile code's origin
- Mobile code is exposed through the network
  - Verification of the mobile code's integrity
- Mobile code is generated by a different party
  - Access control
  - Semantic verification

# Java: Security Manager

- Purpose:
  - establish a custom security policy
  - define the boundaries of the sandbox
- installing an application security manager:
  - in charge for the application lifetime
    - not replaceable
  - policy can depend upon information collected elsewhere
    - e.g. class loader
- To customize the sandbox:
  - extend the java.lang.SecurityManager class
  - override methods authorizing resource access

# Authentication / Integrity - Code Signing

Strong authentication and code integrity through digital signature




Problem: no accountability for mobile code actions

# Java Signed Applets

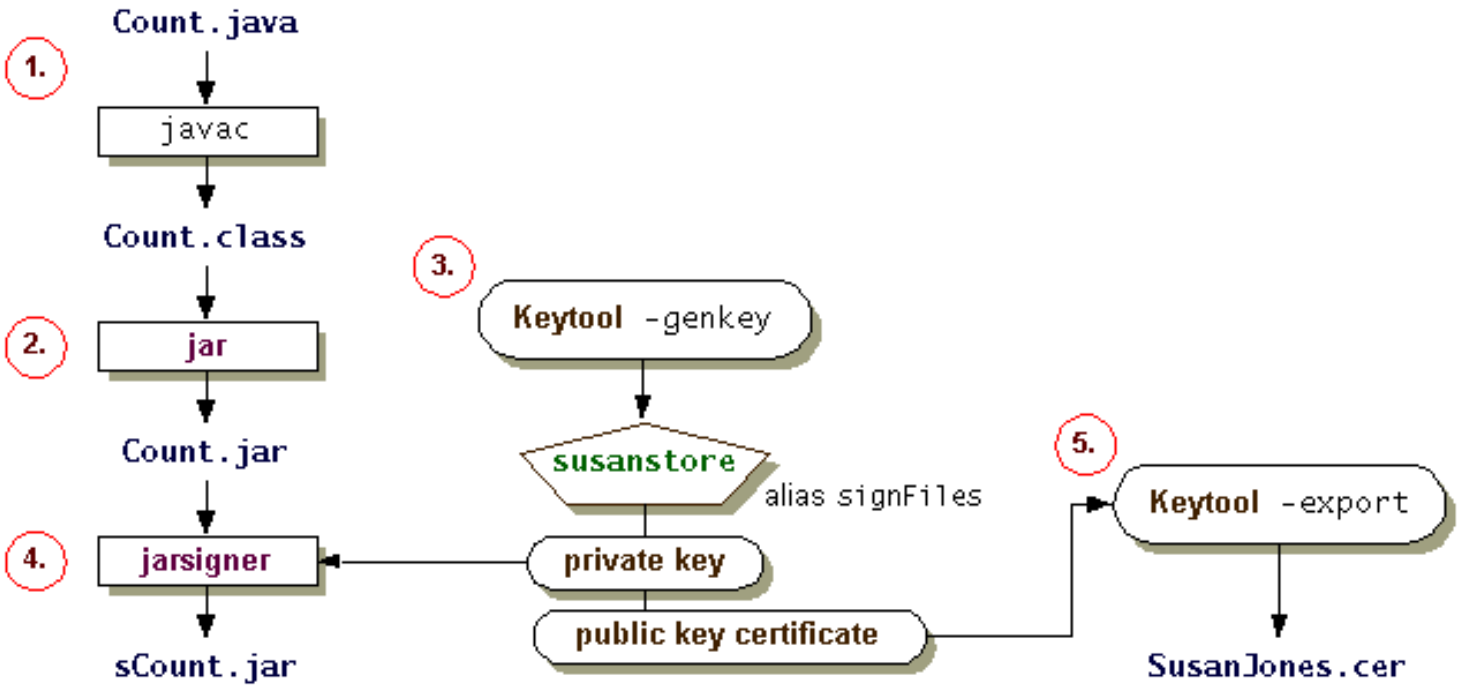
- Available starting from JDK 1.1
- Similar to signed ActiveX components

- Policy:
  - applet authenticated $\Rightarrow$ full access to local resources
  - no certificate $\Rightarrow$ the applet is sandboxed

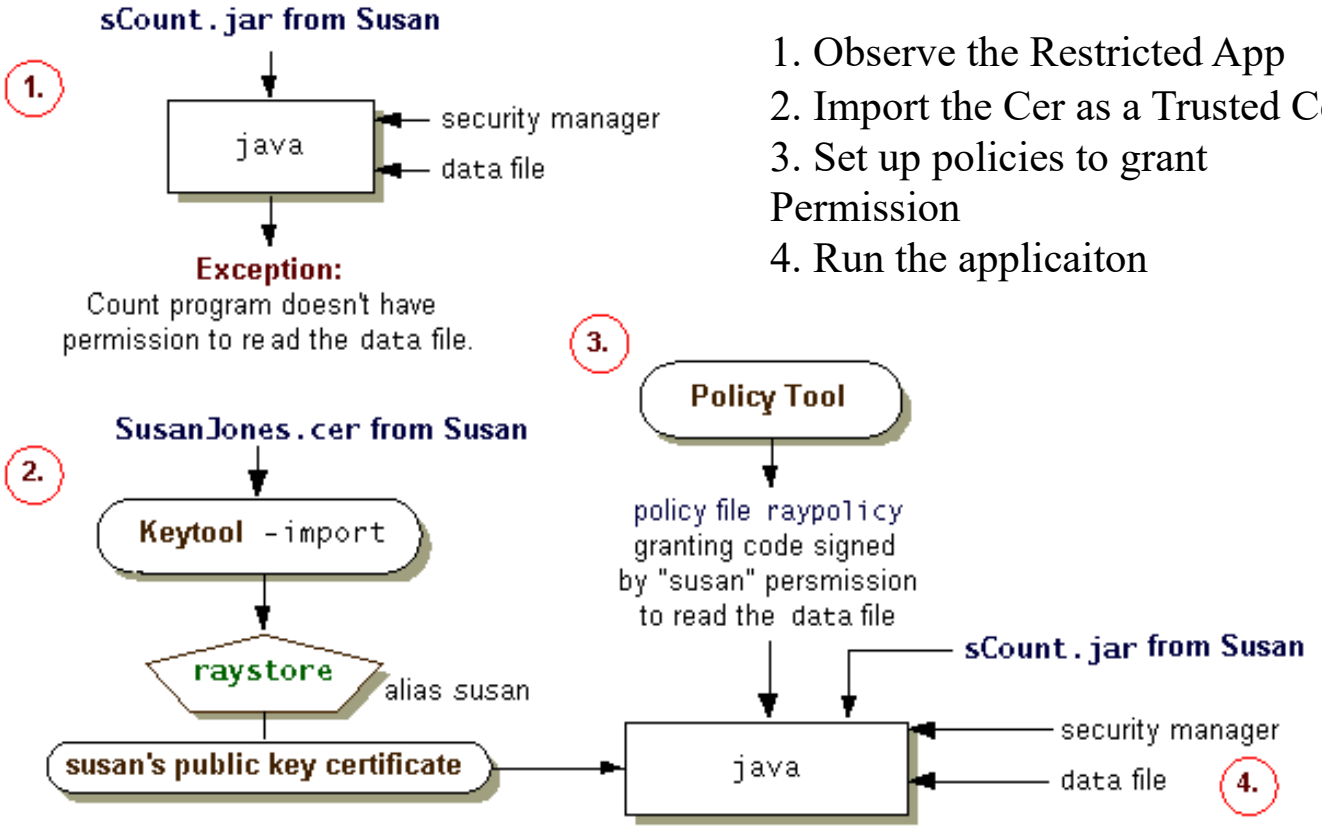
- Archive (JAR) file holds [applet(s) + certificate]

# Signed Code

- Generate keys : (3) keytool –genkey –alias signedFiles –keypass kpi135 –keystore susanstore –storepass ab987c
- (4) jarsigner –keystore susanstore –signedjar sCount.jar Count.jar signFiles
- (5) keytool –export –keystore susanstore –alias signFiles –file SusanJones.cer



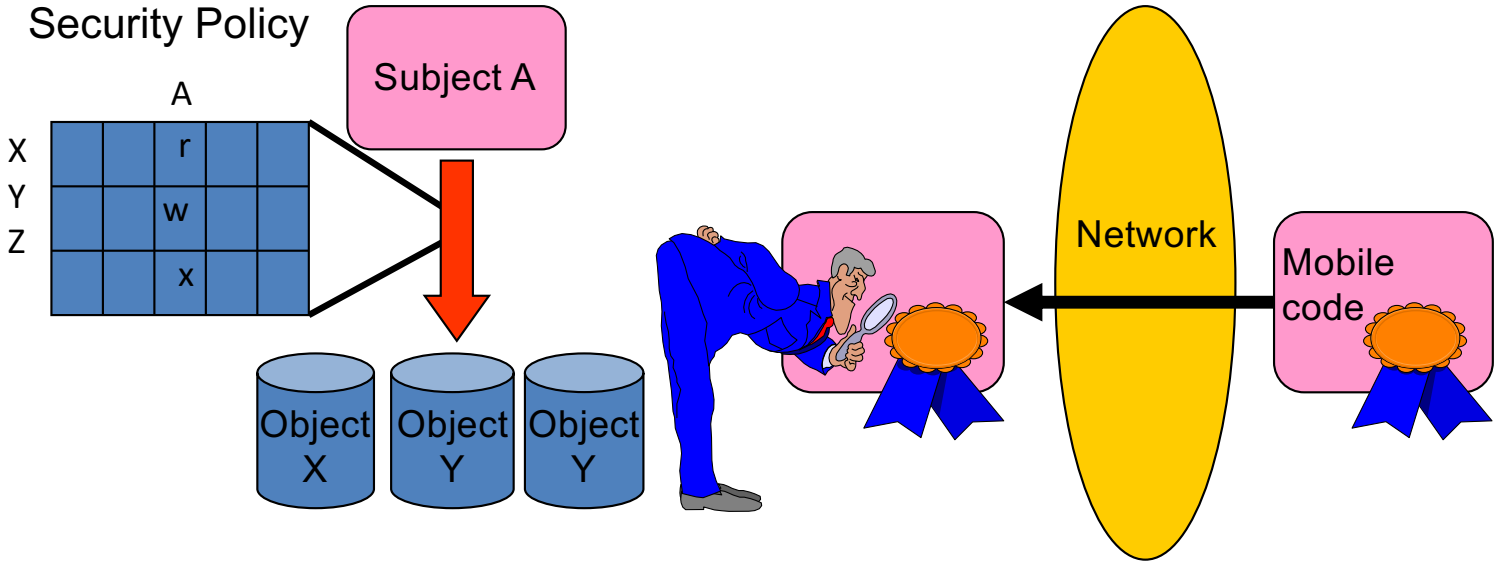
# On Receiving a Signed Code



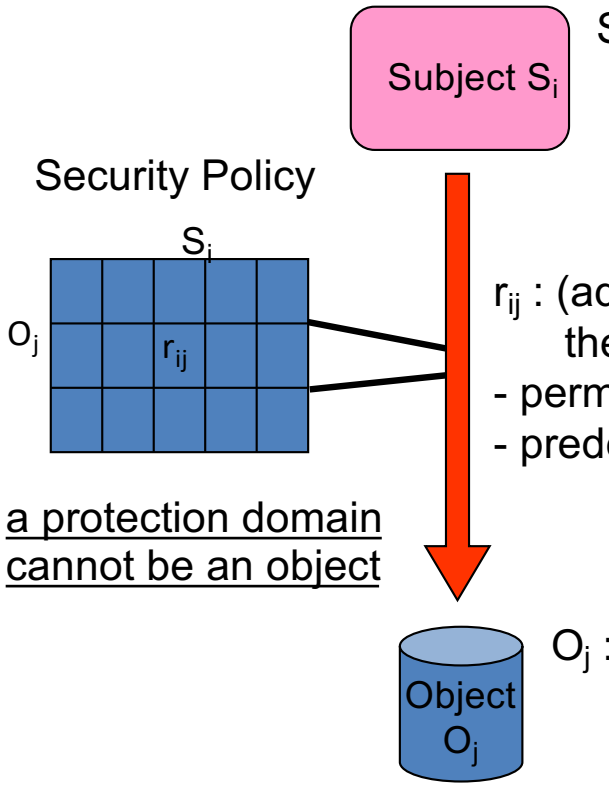

1. Observe the Restricted App
2. Import the Cer as a Trusted Cer
3. Set up policies to grant Permission
4. Run the applicaiton

# Java Access Control - Security Policy

Access Control based on a variable Security Policy



# Java Access Control Model




$S_i$ : code inside a "Protection Domain"
identified by: code base + code signature

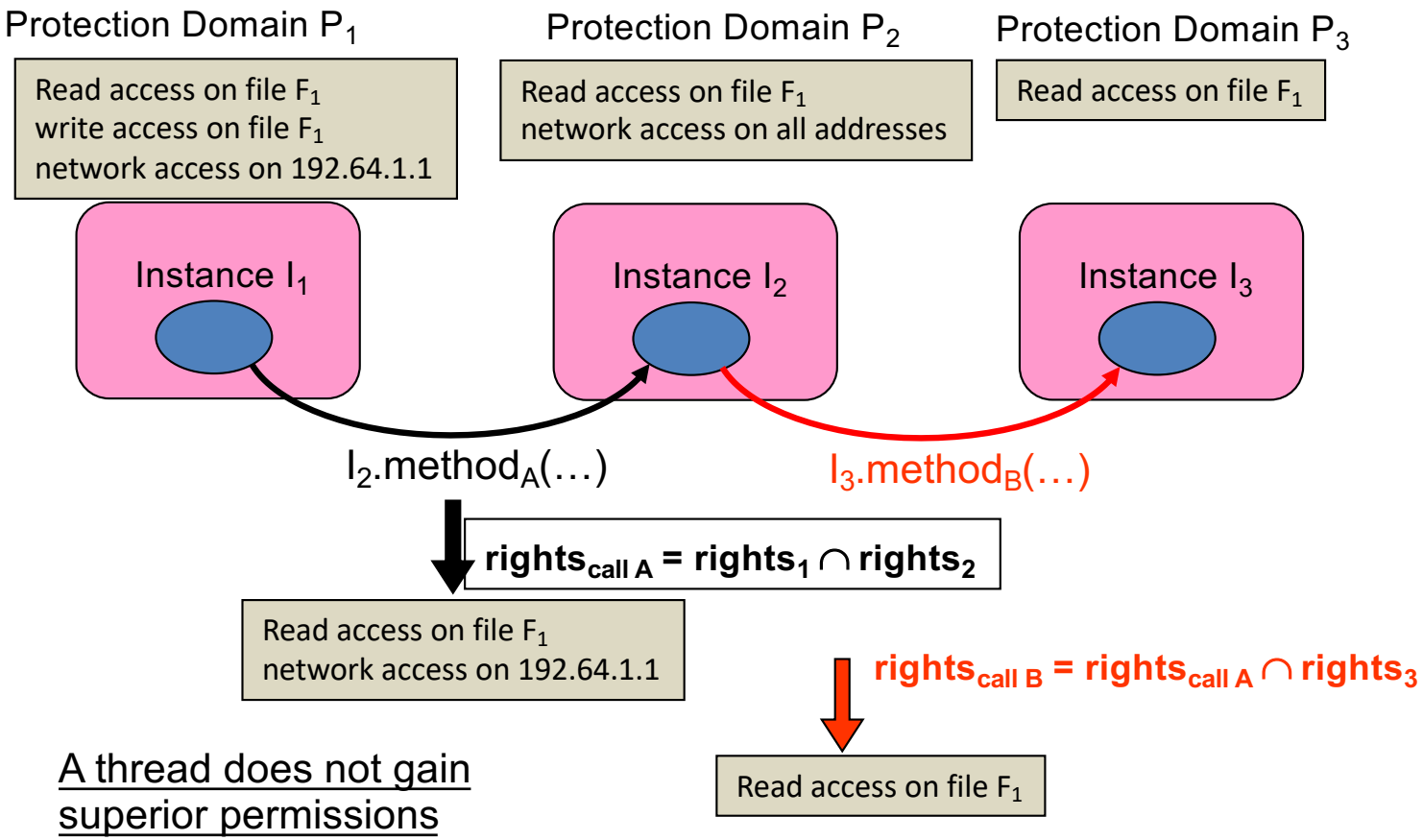
codeBase "http://www.ci.net/mycode/"
signedBy "Bob"

$r_{ij}$ : (administrator + user) "permissions" granted to the thread running $S_i$
- permissions = authorized operations on object $O_j$
- predefined or user-defined permissions types

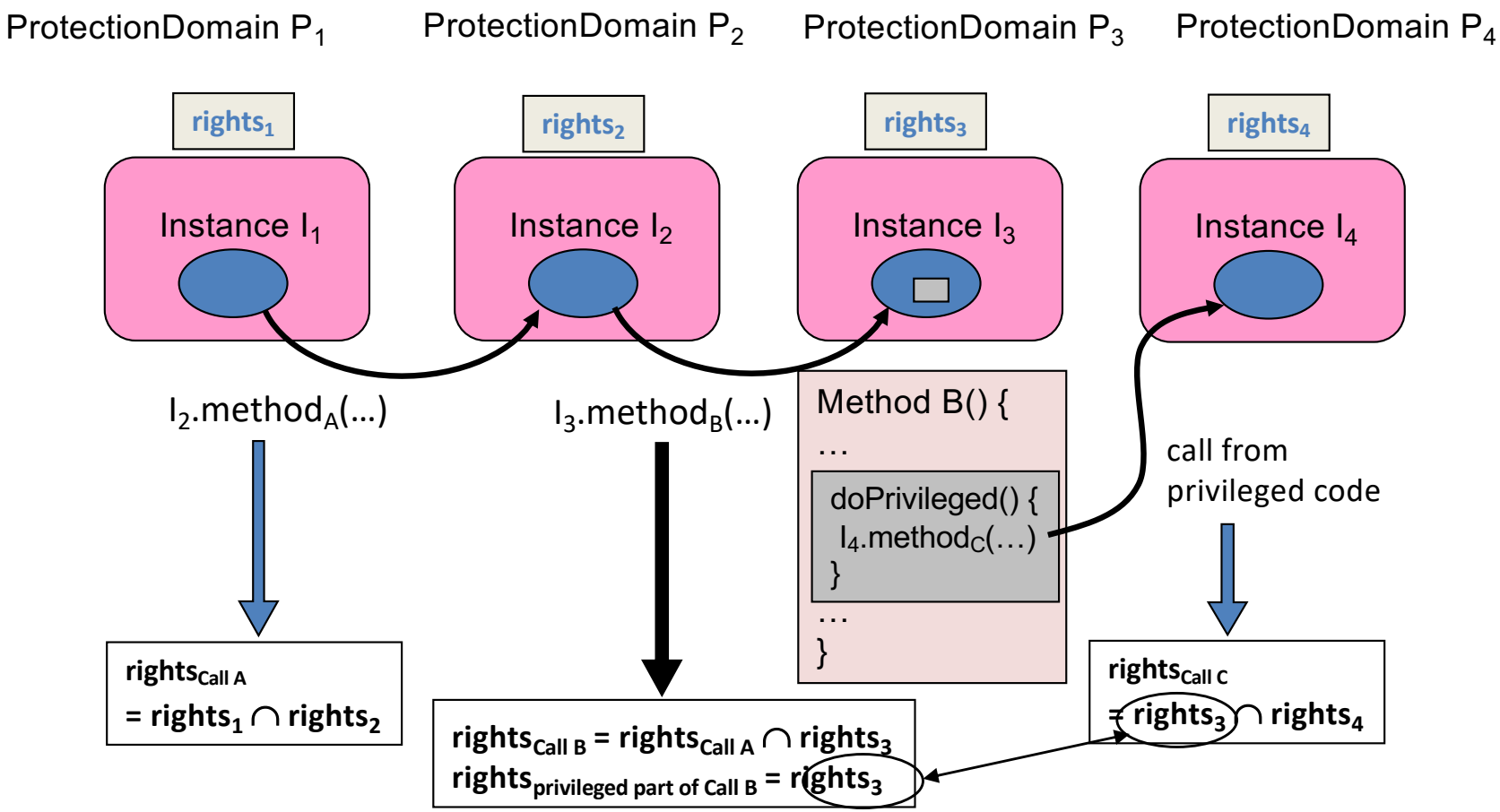
Permission java.io.FilePermission "/tmp/tst", "read"

$O_j$ : resource accessed through a Java object
(network access, file, GUI,
serialization, reflexive layer, …)

# Java: rights across protection domains



# Java: privileged code (POLP)



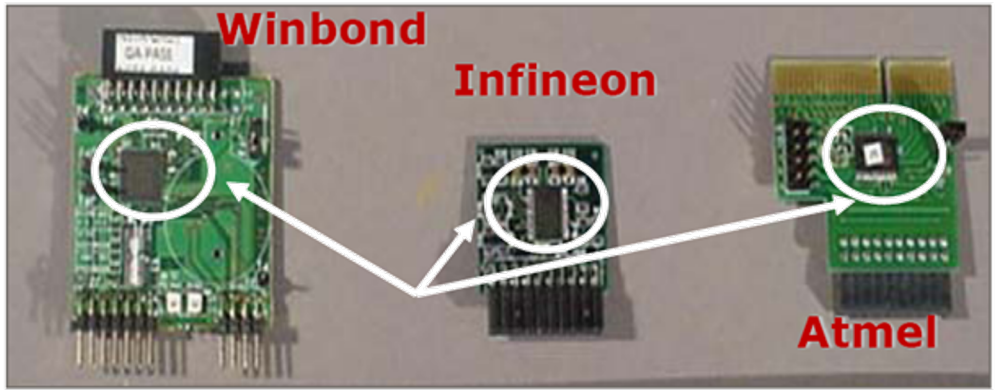
# HW Platform Integrity: Trusted Computing

- TCG consortium, founded in 1999
  - Main members (more than 200):
    - AMD, HP, IBM, Infineon, Intel, Lenovo, Microsoft, Sun
- Architecture has been adopted by other vendors
  - ARM TrustZone, Apple Secure Enclave, Google Titan-M

- **Hardware protected (encrypted) storage**:
  - Only “authorized” software can decrypt data
  - e.g., protecting key for decrypting file system
- **Secure boot**: method to “authorize” software
- **Attestation**: Prove to remote server what software is running on my machine

# Architecture

- HW coprocessor:
  - TCG's TPM chip (Trusted Platform Module)
  - Apple T2
  - Google Titan-M
  - laptops, mobile phones, …




- Software:
  - Modifications to BIOS, OS, Applications
  - Typically used for file/disk encryption (e.g. Bitlocker)
  - Client-side single sign-on linked to a workstation
  - Secure boot / platform attestation before/together with criticial operation (login, signature)

# Hardware Security Module

## TPM 1.2 and 2.0

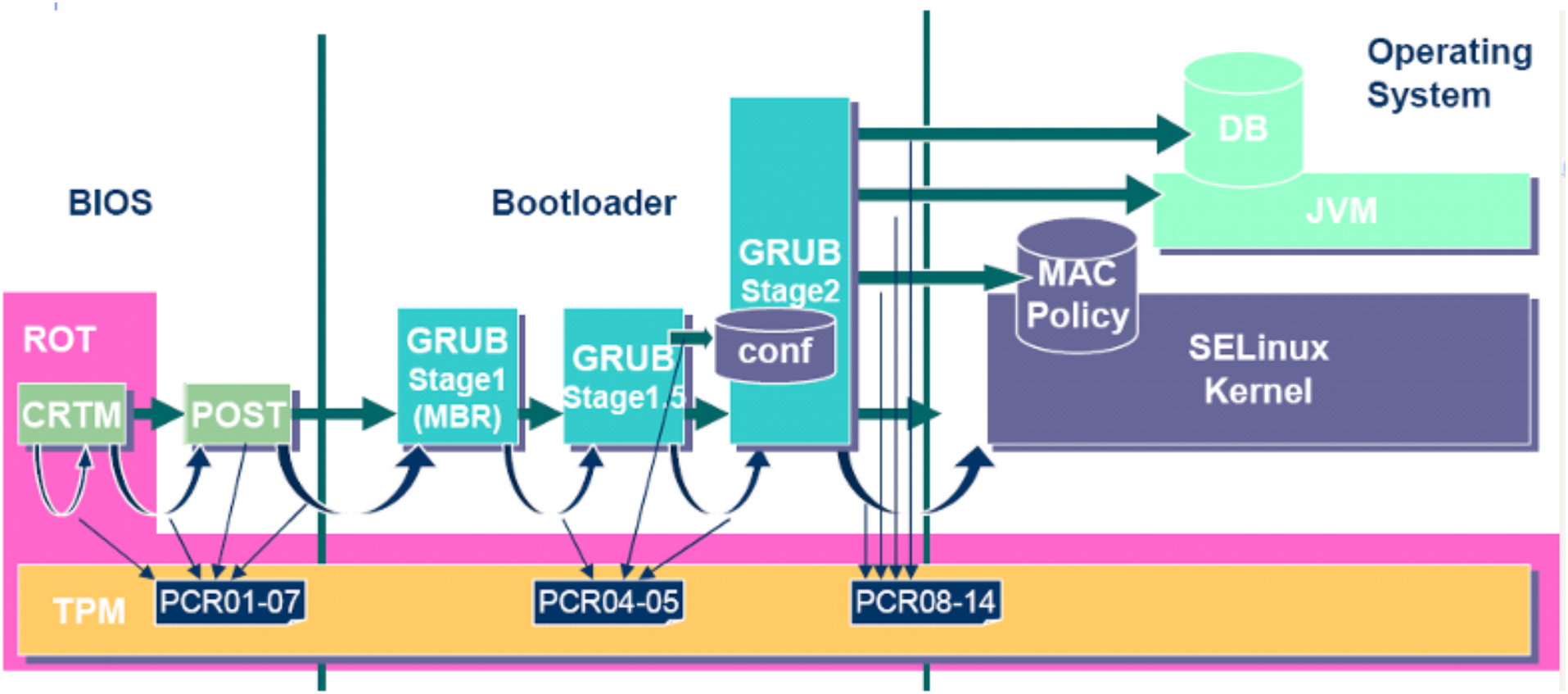
- **Platform integrity.** Makes sure the boot process is correct.
- **Disk encryption.** The encryption key used for dis encryption is fully or partially defined by the SRK.
- **Password protection.** The storage of the user's password in the chip.

**Crypto-processor**
- Random number generator.
- RSA/ECC key generator.
- HMAC generator.
- SHA-1/SHA-256 hash generator.
- Signature engine


**Versatile memory:**
- Platform Configuration Registers (PCR).
- Attestation Identity Keys (AIK).
- Storage Keys.

**Persistent memory:**
- Endorsement Key (EK). A private key from a key pair. A user wishing to send a message to this TPM uses the public key to encrypt. Public key is also stored on TPM.
- Storage Root Key (SRK). Used to encrypt disk storage.

# Secure Boot Example: Trusted Grub (IBM'05)



# Trusted Computing: conclusions

- Widely used today
- APIs to access attestation features and protected storage: extends SW functionality
- Attacks are however POSSIBLE:
  - Faulty validation or attestation processes (e.g., CVE-2018-6622, CVE-2017-16837)
  - HW attacks: e.g., Bitlocker key sniffing (https://pulsesecurity.co.nz/articles/TPM-sniffing)