

4- Cryptography

Segurança Informática em Redes e Sistemas
2021/22

Ack: Ricardo Chaves, Miguel Pardal, Miguel P. Correia,
Carlos Ribeiro

Roadmap

- Introduction
- Symmetric Ciphers
- Hash Functions
- Message Authentication Codes
- Asymmetric Ciphers
- Digital Signatures
- Homomorphic Ciphers

Roadmap

- **Introduction**
- Symmetric Ciphers
- Hash Functions
- Message Authentication Codes
- Asymmetric Ciphers
- Digital Signatures
- Homomorphic Ciphers

Cryptography: terminology

- Cryptography
 - Art or science of writing in a concealed form
 - from Greek: *kryptós*, hidden + *graph*, r. de *graphein*, write
 - Used to assure confidentiality of data (until the 1970s)
 - Steganography
 - from Greek: *steganós*, covered + *graph*, r. de *graphein*, write
- Cryptanalysis
 - The art or science of breaking cryptographic systems or ciphered data
- Cryptology
 - Cryptography + Cryptanalysis

Cryptography

- Widespread and dangerous belief:
 - Encrypting everything provides protection against anything
- A simple example to prove the contrary:
 - Money transfer from one bank to the other
 - The bank encrypts the whole message
 - The attacker:
 - Might not be able to understand the message! (or would he?)
 - But he might be able to:
 - Divert the message into his account (maybe not!)
 - Could get rich by:
 - Diverting or stopping debit messages
 - Allow the passage of all credit messages
 - He might be able to distinguish the two merely by looking at their size
 - Crash the bank by:
 - Injecting random messages

Information

- Unlike mass or energy, information (and data) is non-conservative
 - It can be created
 - It can be destroyed
 - It can be duplicated
- This is the root of most security issues
 - e.g., we cannot infer that it has not been stolen merely because we still have it!
- Conclusion: we need more than just encryption

Attacks on information

- We want to protect the information (or data) against:
 - Unauthorized insertion of information
 - Loss of authenticity
 - Unauthorized modification of information in transit
 - Loss of integrity
 - Unauthorized replay of information
 - From an earlier legitimate data transmission
 - Loss of authenticity
 - Unauthorized access to information
 - Loss of confidentiality
- Which cryptographic services can we use to prevent this?

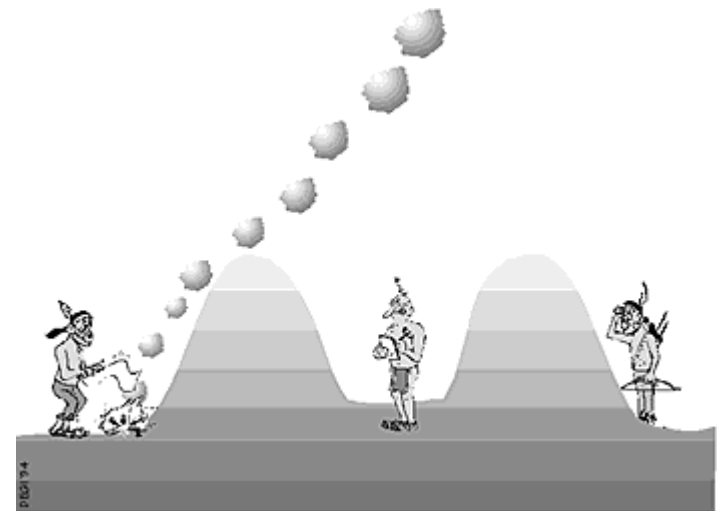
Cryptographic services

- We need the following cryptographic services
 - Confidentiality
 - Data Integrity
 - Authentication
 - Data origin authentication
 - Entity authentication
 - Identification
 - Authorization
 - Non-Repudiation



Confidentiality

- Confidentiality:
 - Is a service used to keep the content of the information from all, but those entities authorized to have it
 - i.e. making the information unintelligible to all but those who possess some secret.
 - Typically operates by encryption:
 - Encryption: the process of converting plaintext to ciphertext
 - Using:
 - » Cryptographic algorithms
 - » Cryptographic key



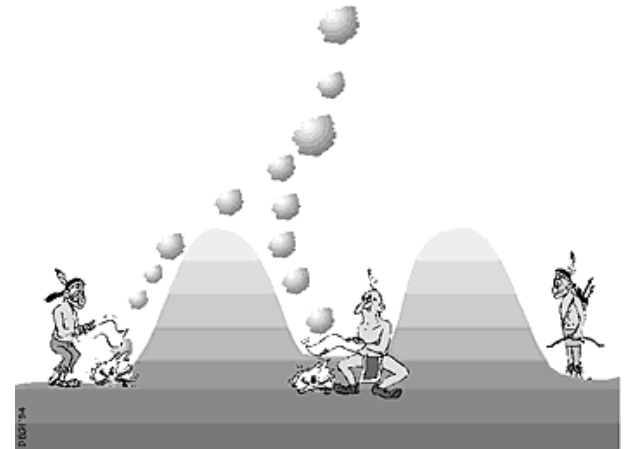
Confidentiality

- Problems:
 - Makes debugging harder
 - Software
 - Systems
 - Protocols
 - Information loss
 - If the key is permanently lost, so is the information
 - Sometimes misused
 - Other, more appropriate, services can be used



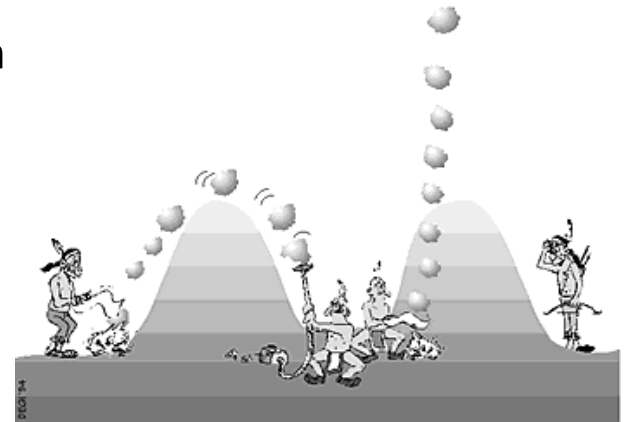
Data Integrity

- Data integrity:
 - Addresses the unauthorized modification of data
 - Detects data manipulation by unauthorized entities
 - An intruder should not be able to substitute a false message for a legitimate one
 - Not the same thing as error detection codes
 - E.g. Cyclic Redundancy Codes (CRC)
 - Are not cryptographically strong
 - » do not protect against intentional alterations of the message



Authentication

- Authentication of messages:
 - Authentication includes Integrity and Freshness
 - It is a service used to ascertain the identity or the origin of a message:
 - Guaranties that entities are who they claim to be
 - Data origin authentication
 - Entity authentication (Identification)
 - An intruder should not be able to masquerade as someone else
 - Authentication \neq secrecy
 - Until mid 1970's secrecy and authentication intrinsically connected
 - Related with the misconception that:
 - » encrypting everything achieves all goals



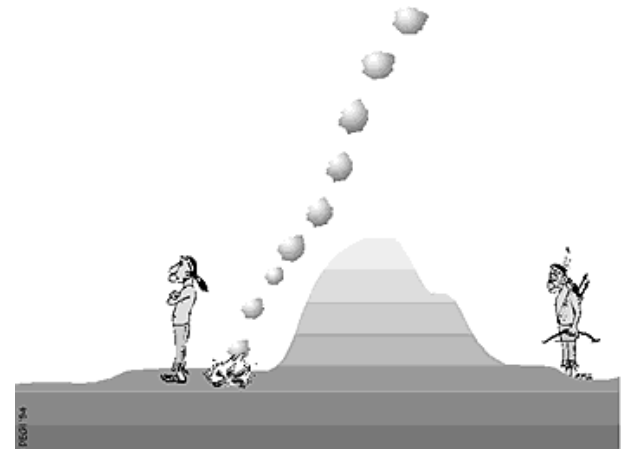
Authentication

- Entity authentication (aka Identification):
 - Applies to real time message exchange
 - If Alice authenticates herself to Bob, then Bob accepts Alice's identity
 - Must be infeasible for a third party to impersonate Alice
 - All these features remain true even after a large number of honest exchanges between Alice and Bob
- Data origin authentication:
 - identify the originator/creator of the message



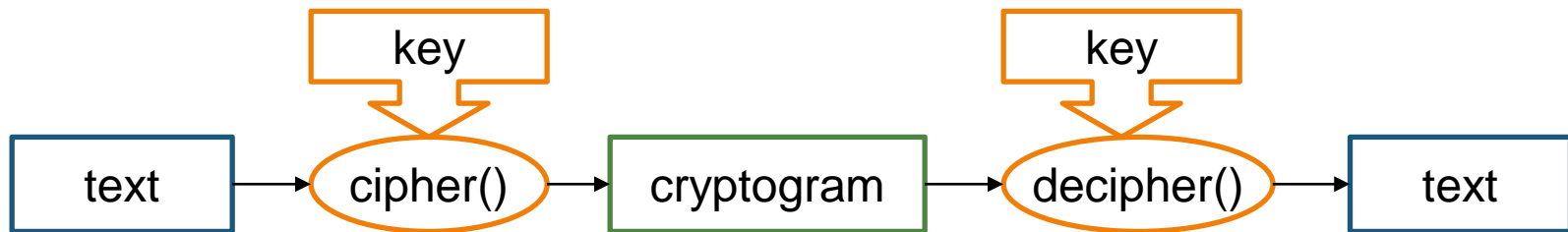
Non-Repudiation

- Non-Repudiation:
 - It is a service which prevents an entity from denying previous commitments or actions
 - Such as:
 - a sent message
 - a signed document
 - ...



Ciphers: terminology

- Cipher
 - Specific cryptographic technique
- Cipher Procedure
 - Cipher: plaintext \rightarrow cryptogram (aka ciphertext)
 - Decipher: cryptogram \rightarrow plaintext
 - Algorithm: data transformation procedure
 - Key: algorithm parameter

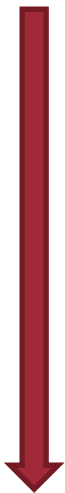


Cryptanalysis

what cryptography has to protect from

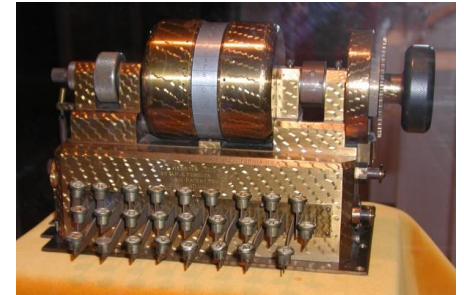
- Basic assumption: the algorithm is known
 - If not public, might be obtained (e.g., stolen)
- Attacks:
 - **Ciphertext-only**: cryptanalyst has access to ciphertexts
 - Without them, no cryptanalysis is possible
 - **Known-plaintext**: cryptanalyst has a set of ciphertexts to which he knows the corresponding plaintext
 - Often easy to get at least partial plaintext, e.g., message beginning
 - **Chosen-plaintext**: cryptanalyst can obtain the ciphertexts corresponding to plaintexts of his choice; or:
 - **Chosen-ciphertext**: cryptanalyst can obtain the plaintexts corresponding to ciphertexts of his choice

less probable and easier



Old (broken) ciphers

- Caesar cipher
 - Shift by 3
- Substitution cipher
 - A -> C
 - ...
- Vigenere cipher (1500s)
 - + mod 26
- Rotor machines (1870-1943)
 - Single rotor: Hebern
 - 3-5 rotors: Enigma
- DES (Digital Encryption Standard) - 1974
 - 56 bits key, 64 bits block



Hebern machine



Enigma machine

Modern cipher types

- Regarding the procedure
 - Stream
 - Block
- Regarding the type of key
 - Symmetric (secret key, a shared secret)
 - Asymmetric (public key and private key)

Ciphers	Block	Stream
Symmetric	yes	yes
Asymmetric	yes	no

Roadmap

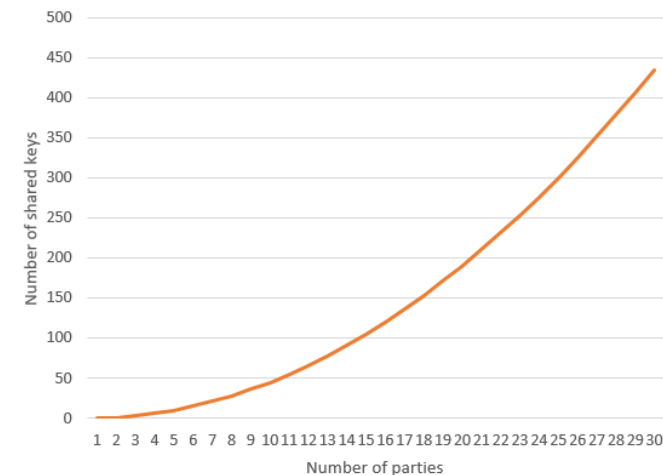
- Introduction
- **Symmetric Ciphers**
- Hash Functions
- Message Authentication Codes
- Asymmetric Ciphers
- Digital Signatures
- Homomorphic Ciphers

Symmetric Ciphers

- Symmetric Ciphers
 - Stream ciphers
 - Block ciphers
 - DES
 - AES
 - Block cipher modes

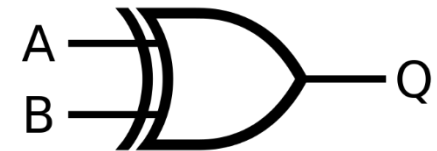
Symmetric Ciphers

- Secret key
 - Shared by 2 or more communicating parties
- Allow for
 - Confidentiality to all who possess the key
 - Message authentication
- Advantages
 - Performance (typically very efficient)
- Disadvantages
 - N communicating parties, 1 to 1 secretly
→ $N \times (N-1)/2$ keys
- Problems
 - Key distribution



Perfectly Secure Cipher: One-Time Pad

- Mauborgne/Vernam [1917]
- XOR (\oplus):
 - $0 \oplus 0 = 0$ $1 \oplus 0 = 1 \rightarrow a \oplus 0 = a$
 - $0 \oplus 1 = 1$ $1 \oplus 1 = 0 \rightarrow a \oplus 1 = \text{not } a$
$$a \oplus b \oplus b = a$$
- Encrypt
 - $E(P, K) = P \oplus K = C$
 - P = plaintext; K = key
- Decrypt
 - $D(C, K) = C \oplus K = (P \oplus K) \oplus K = P$



Perfectly Secure Cipher: One-Time Pad

- One-Time Pad (XOR message with key)
- Example:
 - Message: ONETIMEPAD
 - Key: TBFRGFARFM
 - Ciphertext: IPKLPSFHGQ
 - The key TBFRGFARFM decrypts the message to ONETIMEPAD
 - The key POYYAEAAZX decrypts the message to SALMONEGGS
 - The key BXFGBMTMXM decrypts the message to GREENFLUID

One-Time Pad problems

- Security is based on the assumption that K is never reused
 - What if one has two encrypted messages:

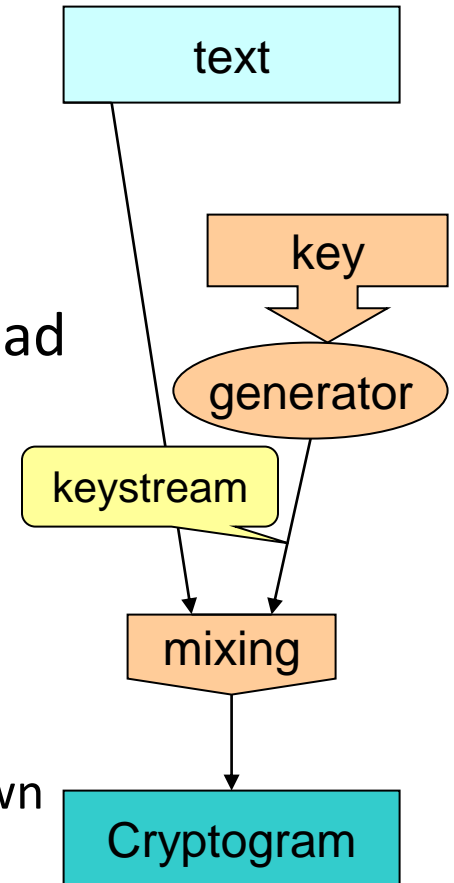
$$C_1 = P_1 \oplus K \text{ and } C_2 = P_2 \oplus K$$

$$\begin{aligned} C_1 \oplus C_2 &= P_1 \oplus K \oplus P_2 \oplus K \\ &= P_1 \oplus P_2 \end{aligned}$$

- Need to generate truly random bit sequence
 - As long as all messages
- Need to securely distribute key bit sequence (!)

Stream ciphers

- Practical approximation to the One-Time Pad
- Keystreams generated in a deterministic way
 - From a fixed size key
 - Approximation to real random sequence generators
- Encryption and decryption as with a one-time pad
 - i.e., by doing XOR with the keystream (mixing)
- Practical aspects of stream ciphers security:
 - If the plain text is known, the keystream is exposed
 - The repetition of cycles (reuse of the keystream) facilitates cryptanalysis
 - if the cycle period or part of the plain text is known
 - Integrity control must exist
 - Easy to modify the cryptogram in a deterministic way

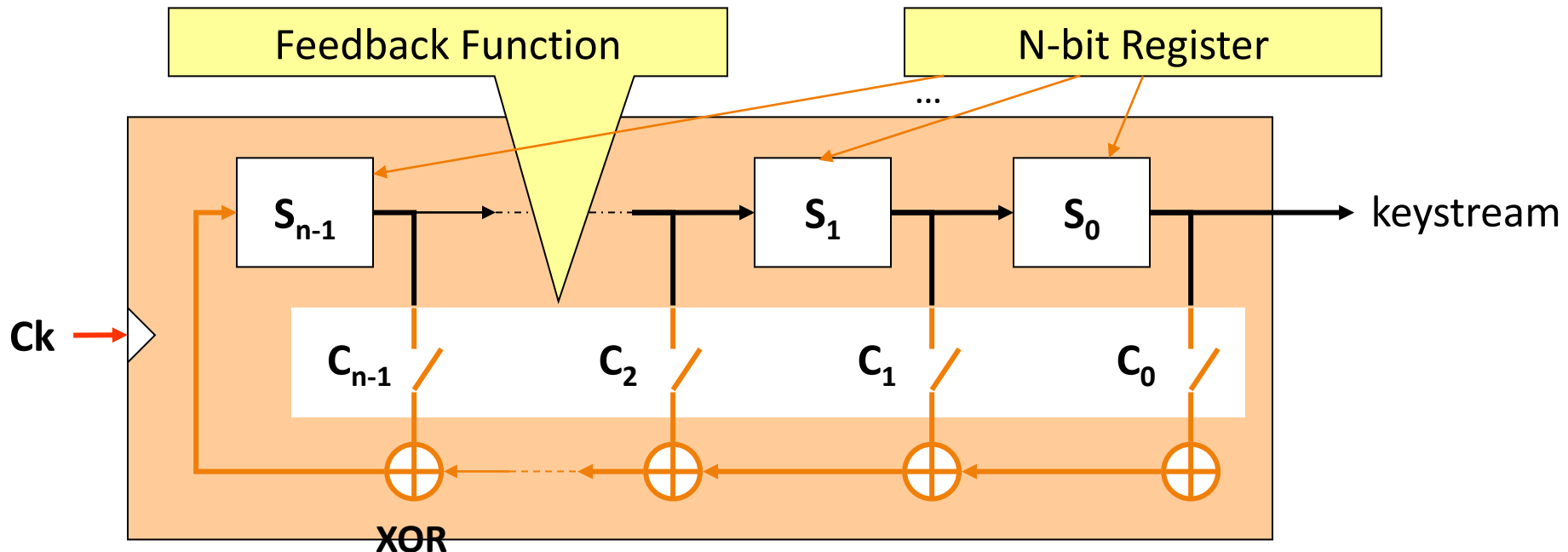


Symmetric stream ciphers

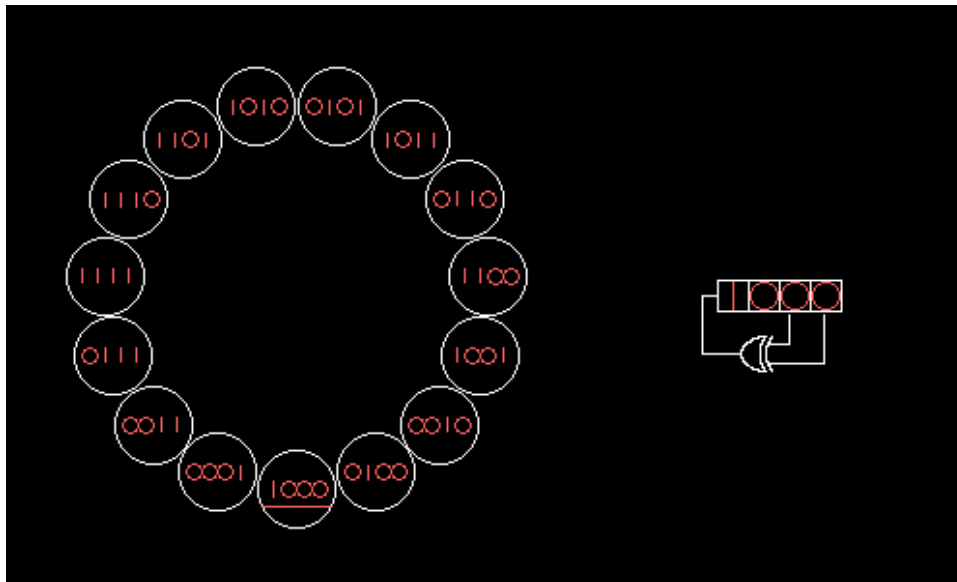
- Used approximations
 - Secure pseudo-random generators
 - Based on LFSRs (Linear Feedback Shift Registers) → next
 - Based on block ciphers → later
 - Other approximations (nonlinear functions, etc.)
 - Usually without self-synchronization
 - Receiver must know when encrypted data begins
 - Typically without the possibility of fast random access
- Most common algorithms
 - A5 (GSM)
 - RC4
 - SEAL (with fast random access)

Linear Feedback Shift Register (LFSR)

- State machine that produces a cyclic sequence of bits
 - The sequence depends on the **key** = initial state of the register
 - S_0, \dots, S_{n-1} = **register's** bits; C_0, \dots, C_{n-1} = **coefficients** of the function
 - Max. period of the cycle is $2^n - 1$



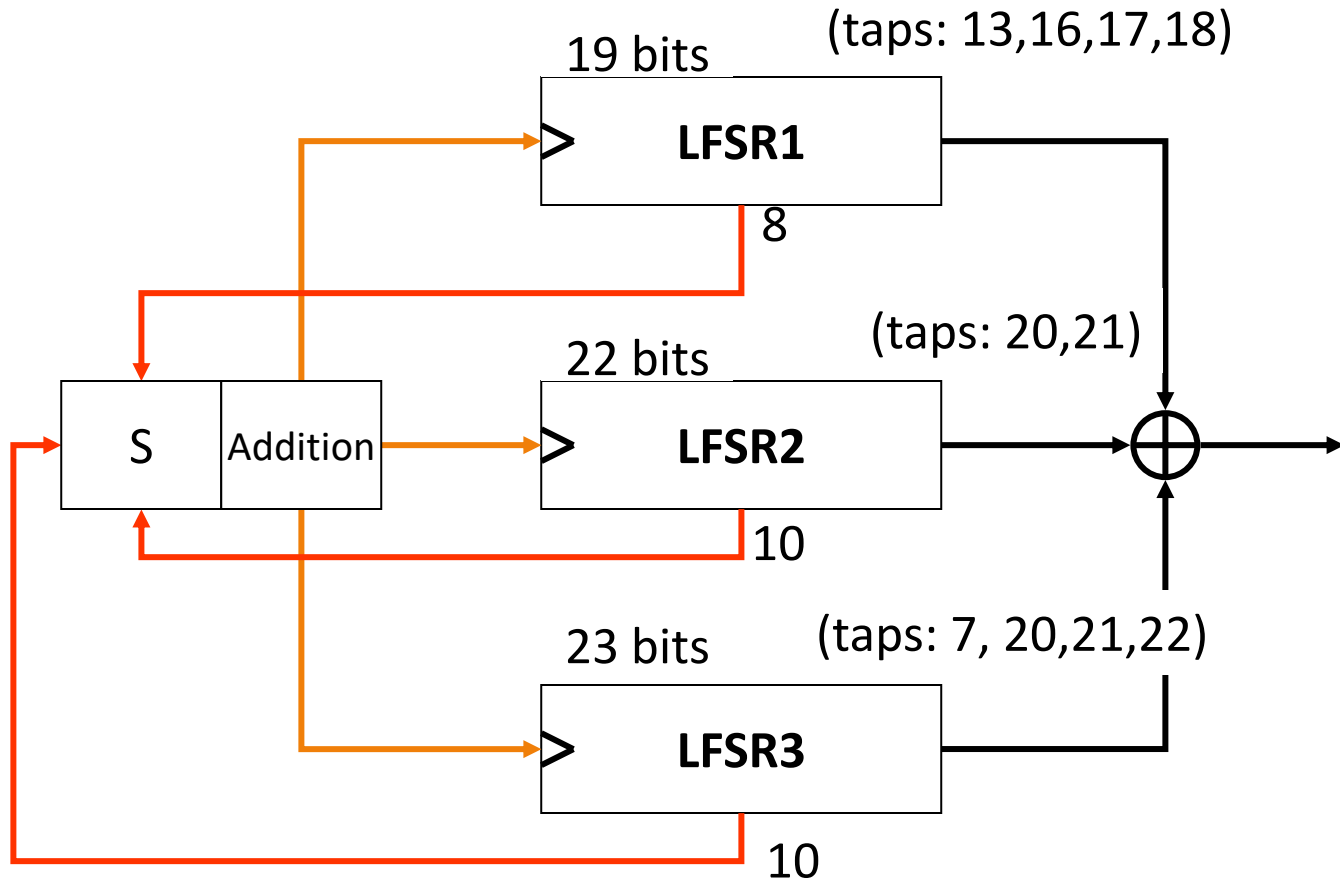
4 bit LFSR example



Bits 2 and 3 are **tapped**

Taps are the bits that affect the next state

LFSR structure: A5/1 (GSM)



Symmetric block ciphers

- Also based on approximations, using Shannon's notions of confusion and diffusion:
 - **Confusion**: repeated application of a complex function to a large block (e.g. 64 bits)
 - **Diffusion**: basic operations:
 - **Permutation**: exchange bits without losing or adding bits
 - **Substitution**: change bits for others using a substitution table
 - **Expansion**: introducing new bits
 - **Compression**: deleting some bits
- Some relevant symmetric encryption algorithms:
 - **DES** – Data=64; Key=56 – insecure, never use
 - **AES** – D=128; K=128, 192, 256 – current standard
 - Others (Salsa20, Blowfish, CLEFIA, RC5, etc.)

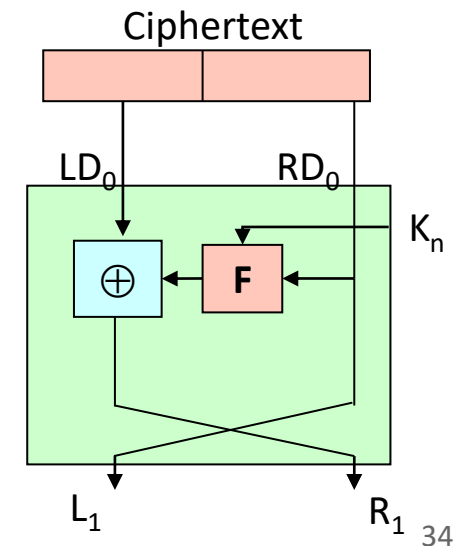
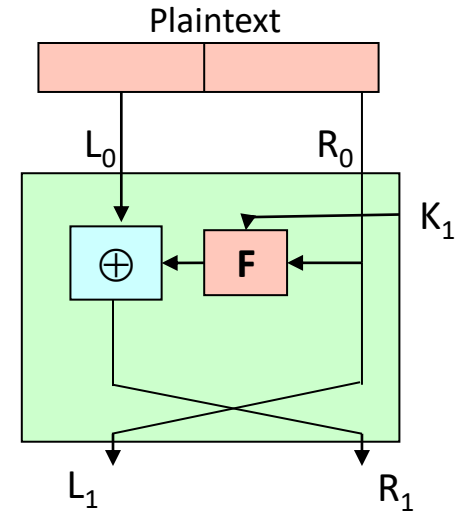
DES Avalanche

[illegible]

* is a changed bit that propagates changes

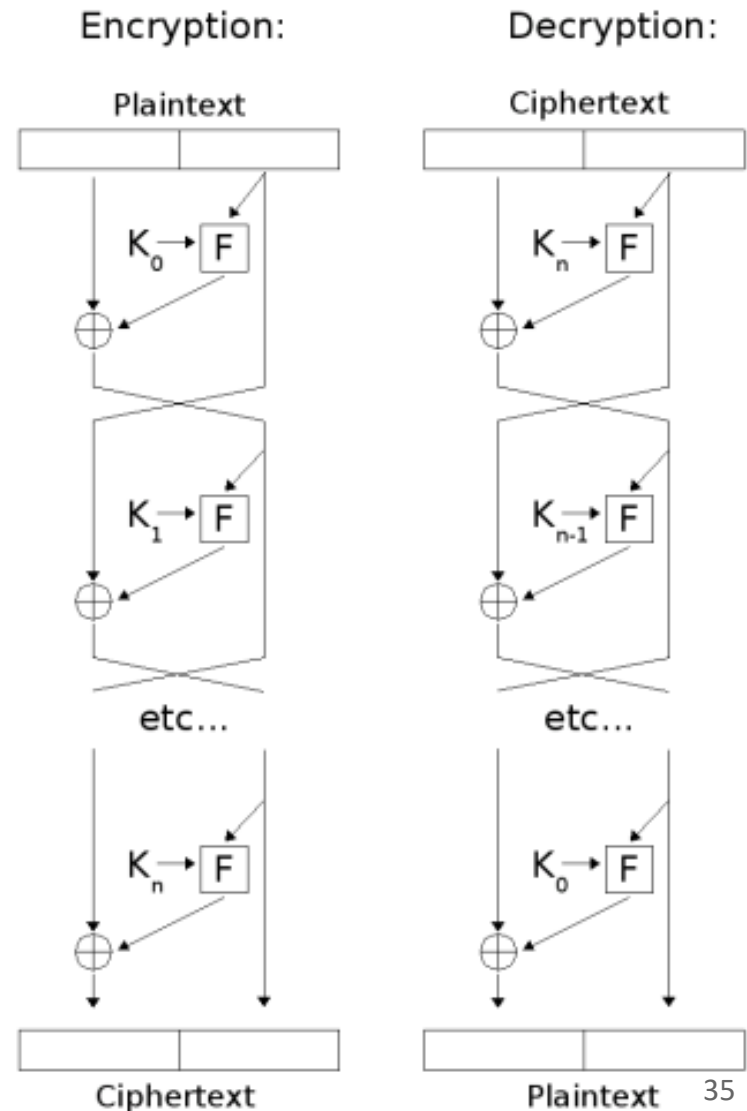
Feistel Network

- Complex function most commonly used in block cipher algorithms
- Applies a **round function (F)** over multiple rounds
 - **F** can be a pseudo-random generator
 - Each round uses a different **round key (K_i)**
 - Round keys are obtained from the key (K)
 - Text is split in **left (L)** and **right (R)** parts



Feistel Network

- Cipher and decipher processes are the same
 - Keys are used in the inverse order



DES Algorithm

- DES:
 - 64-bit blocks
 - Key is 56 bits
- Possible key combinations
 - $2^{56} = 7.2 \times 10^{16} = 72,000,000,000,000,000$
 - Try 1 per second = 9,000 Million years to search the entire key space
- Distributed attacks on DES:
 - RSA's DES challenges:
 - 1997: 96 days (using 70,000 machines)
 - 1998: 41 days (distributed.net)
 - 2008: less than 1 day (array of 128 Spartan FPGAs ~ €5k)
 - 2016: about 15 days on a standard PC with a GPU (~€800)

Block ciphers: reinforcement

- Multiple ciphers

- Double cipher

- Breakable by brute force in max. 2^{n+1} attempts instead of expected 2^{2n}

- Triple cipher, typ. EDE = Encrypt-Decrypt-Encrypt – 3DES-EDE

- $C_i = E_{K_1}(D_{K_2}(E_{K_3}(P_i)))$
 - $P_i = D_{K_3}(E_{K_2}(D_{K_1}(C_i)))$
 - Typically, $K_1=K_3$ is used
 - Effective key size = 56 + 56 bits = 112 bits

- Key whitening (DESX)

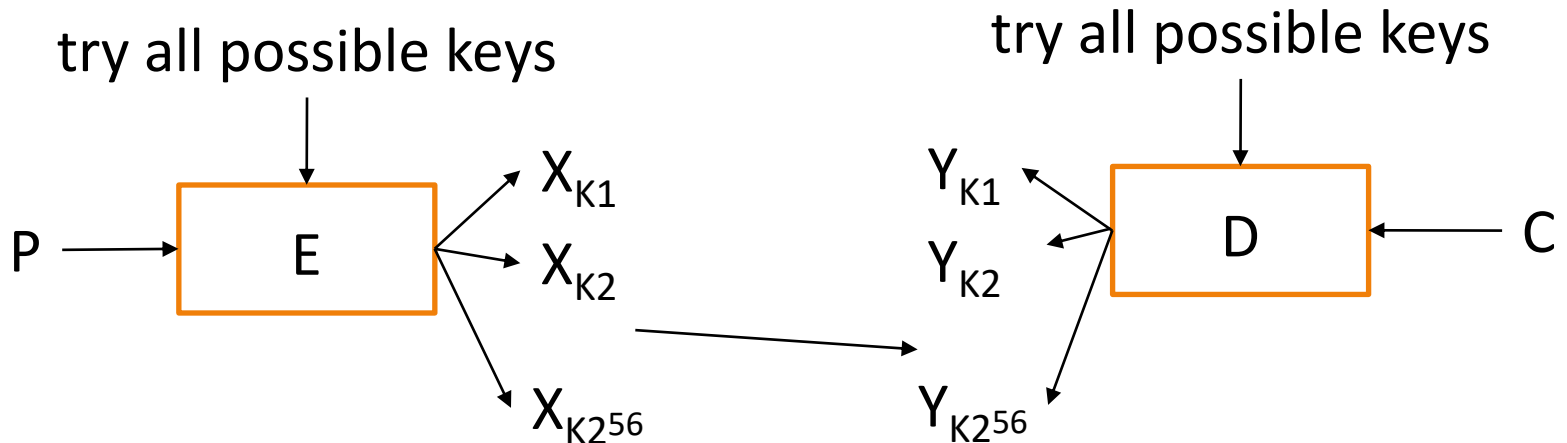
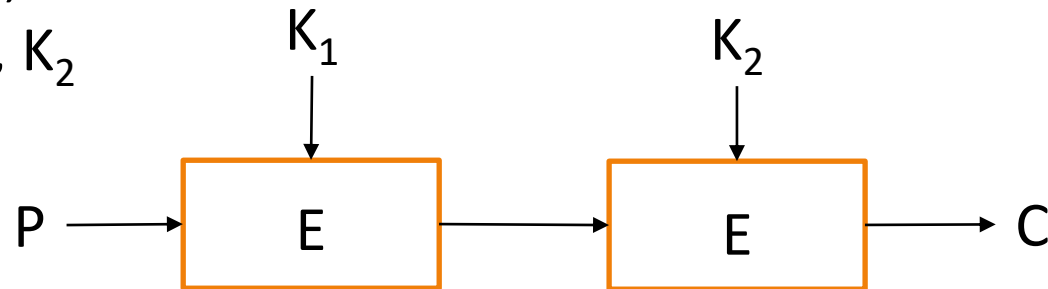
- $DES-X(P_i) = K_2 \oplus DES_K(K_1 \oplus P_i)$
 - $C_i = E_K(K_1 \oplus P_i) \oplus K_2$
 - $P_i = K_1 \oplus D_K(K_2 \oplus C_i)$

Meet-in-the-middle attack

- Double DES
 - $C = E_{K_2}(E_{K_1}(P))$
 - Effective key size of Double DES?
 $= 2^{56} * 2^{56} = 2^{112}$ **WRONG!**

Meet-in-the-middle attack a known-plaintext attack

Attacker knows P , C
doesn't know K_1 , K_2



One $X_{K_i} = Y_{K_j}$ means $K_1 = K_i$ and $K_2 = K_j$
so maximum effort is $2 * 2^{56}$ tries

Meet-in-the-middle attack

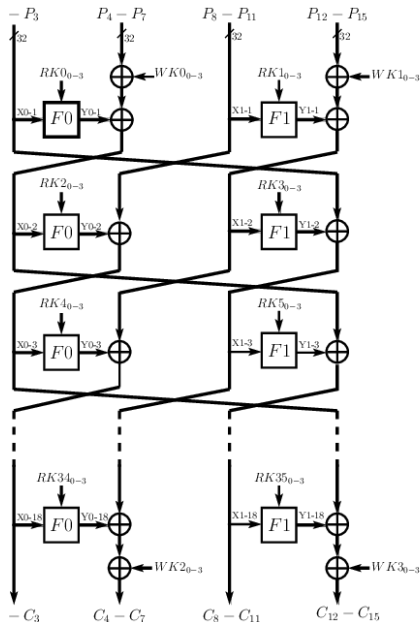
- Double DES

- $C = E_{K_2}(E_{K_1}(P))$
- Effective key size of Double DES?
 $= 2^{56} * 2^{56} = 2^{112}$ **WRONG!**

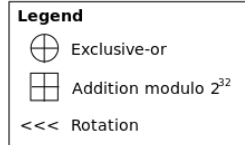
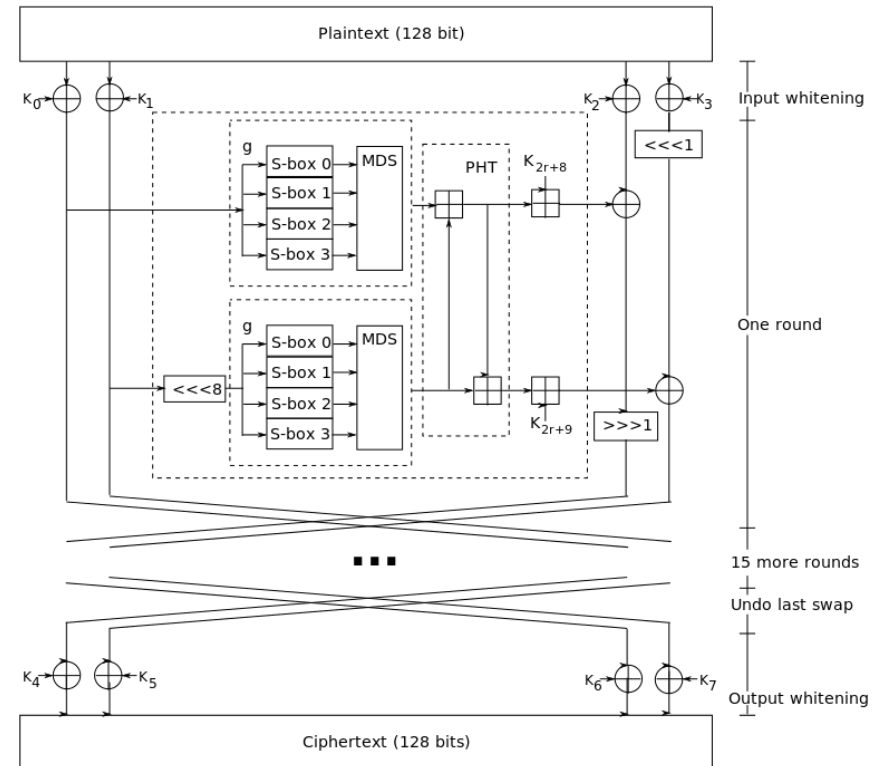
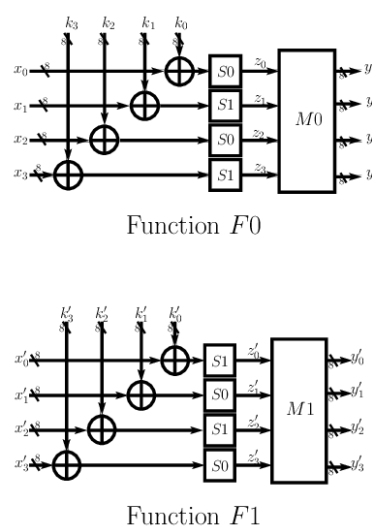
- Meet-in-the-Middle Attack

- $C = E_{K_2}(E_{K_1}(P))$
- $X = E_{K_1}(P) = D_{K_2}(C)$
- Brute force attack (given one P/C pair):
 1. calculate $E_{K_1}(P)$ for all keys (2^{56} work)
 2. calculate $D_{K_2}(C)$ for all keys (2^{56} work)
 3. the match gives the keys
- Total work = $2 * 2^{56} = 2^{57}$

Other Feistel Based Ciphers



CLEFIA



TWOFISH

AES – current encryption standard

- AES (Advanced Encryption Standard) parameters:
 - Key size: 128, 192, 256 bits
 - Input block length: 128 bits
- Runs 10/12/14 rounds in which it:
 - substitutes bytes (one S-box used on every byte)
 - shifts rows (permute bytes between columns)
 - mixes columns (substitute using column multiplication)
 - adds round key (XOR *state* with key material)
 - round key size: 128
 - With fast XOR & table lookup implementations

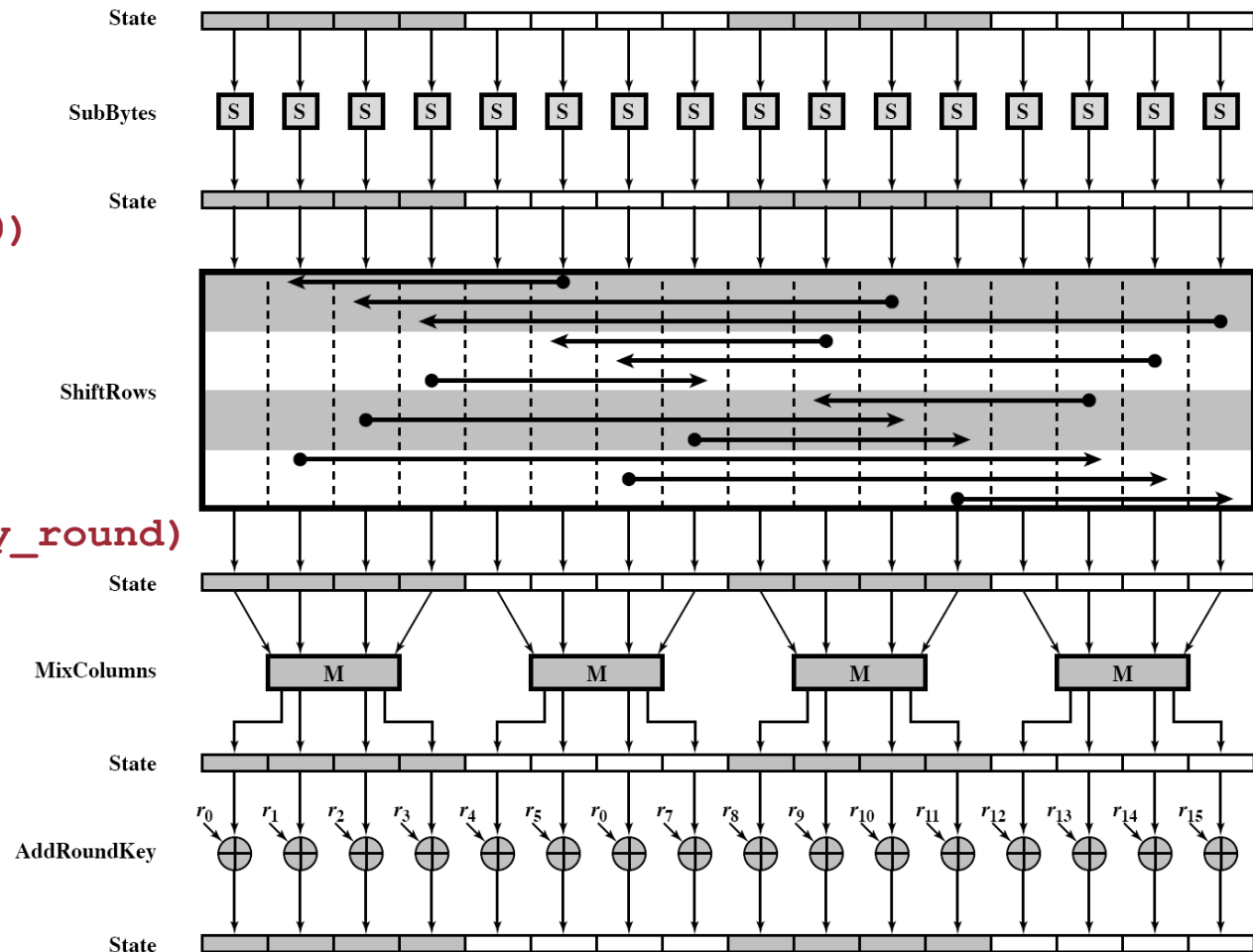
AES - Main Encryption Round

`state = input`

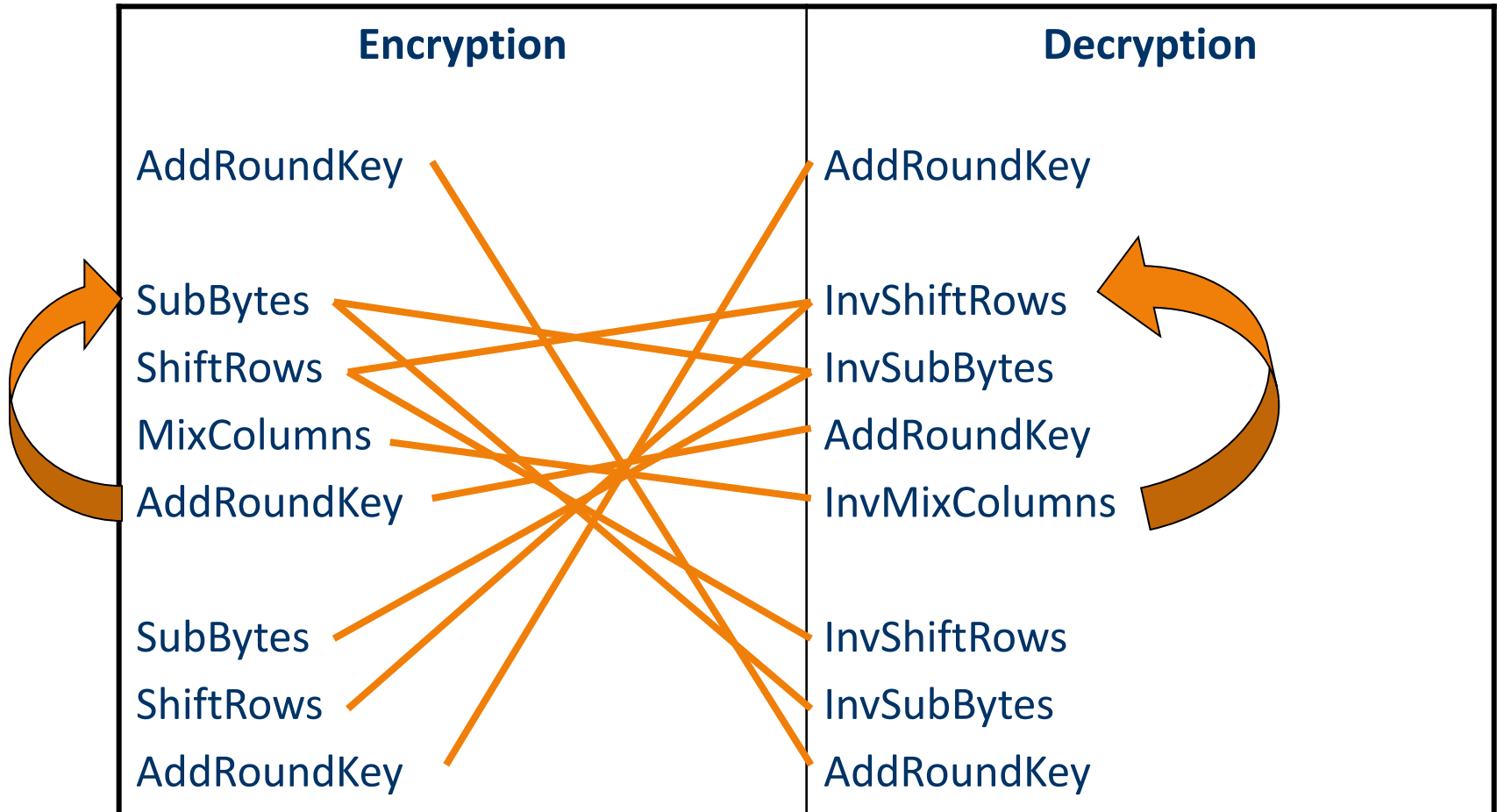
`AddRoundKey(Key0)`

```
for #rounds{
  SubBytes()
  ShiftRows()
  MixColumns()
  AddRoundKey(Key_round)
}
```

`out = state`



AES – Encrypt vs Decrypt



Which algorithm / key size to use?

NIST Special Publication 800-131A
Revision 2

Transitioning the Use of Cryptographic Algorithms and Key Lengths

Elaine Barker
Allen Roginsky

This publication is available free of charge from:
<https://doi.org/10.6028/NIST.SP.800-131Ar2>

C O M P U T E R S E C U R I T Y

NIST
National Institute of
Standards and Technology
U.S. Department of Commerce

March 2019

Table 1: Approval Status of Symmetric Algorithms Used for
Encryption and Decryption

Algorithm	Status
Two-key TDEA Encryption	Disallowed
Two-key TDEA Decryption	Legacy use
Three-key TDEA Encryption	Deprecated through 2023 Disallowed after 2023
Three-key TDEA Decryption	Legacy use
SKIPJACK Encryption	Disallowed
SKIPJACK Decryption	Legacy use
AES-128 Encryption and Decryption	Acceptable
AES-192 Encryption and Decryption	Acceptable
AES-256 Encryption and Decryption	Acceptable

Cipher modes

- Problem
 - How to use a block cipher with a fixed block size with plaintext of a different size?
 - Plaintext may be much larger than the block size
 - Size of the plaintext may not be a multiple of the size of the block
- The problem is solved by cipher modes

Block cipher modes

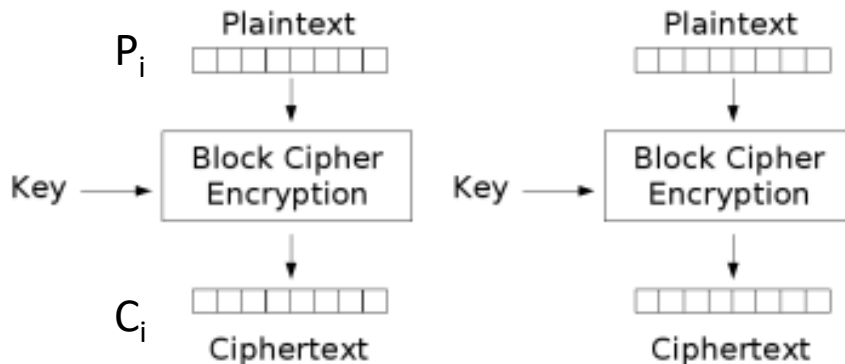
- Initially proposed for DES
 - ECB (Electronic Code Book)
 - CBC (Cipher Block Chaining)
 - OFB (Output FeedBack mode)
 - CTR (CounTeR mode)
 - GCM (Galois Counter Mode)
- Final sub-block processing
 - Alignment with padding
 - Augments the cryptogram by adding padding to the plaintext

Block cipher modes: ECB

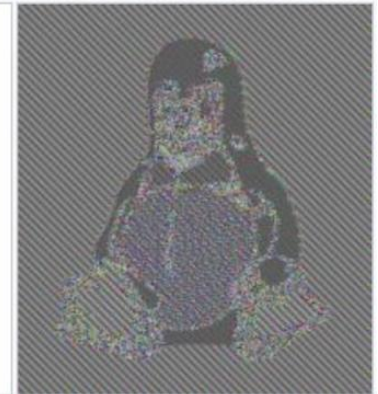
ECB (Electronic Code Book)

$$C_i = E_k(P_i)$$

$$P_i = D_k(C_i)$$



Original image



Encrypted using ECB mode

Original by Larry Ewing's

- **Strength:**
 - Simple
- **Weakness:**
 - Repetitive information in the plaintext may show in the ciphertext, if aligned with blocks
 - If same message is encrypted with the same key and resent, their cipher texts are the same
- Typical use: sending short pieces of data, e.g., encryption key

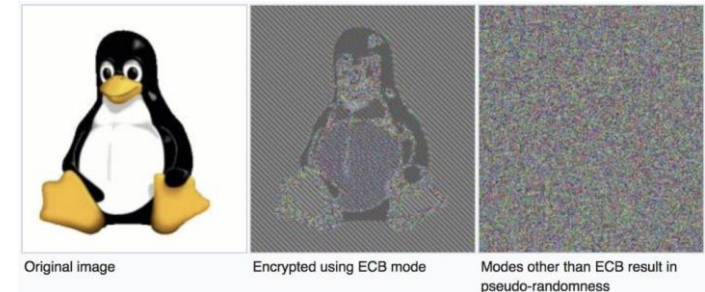
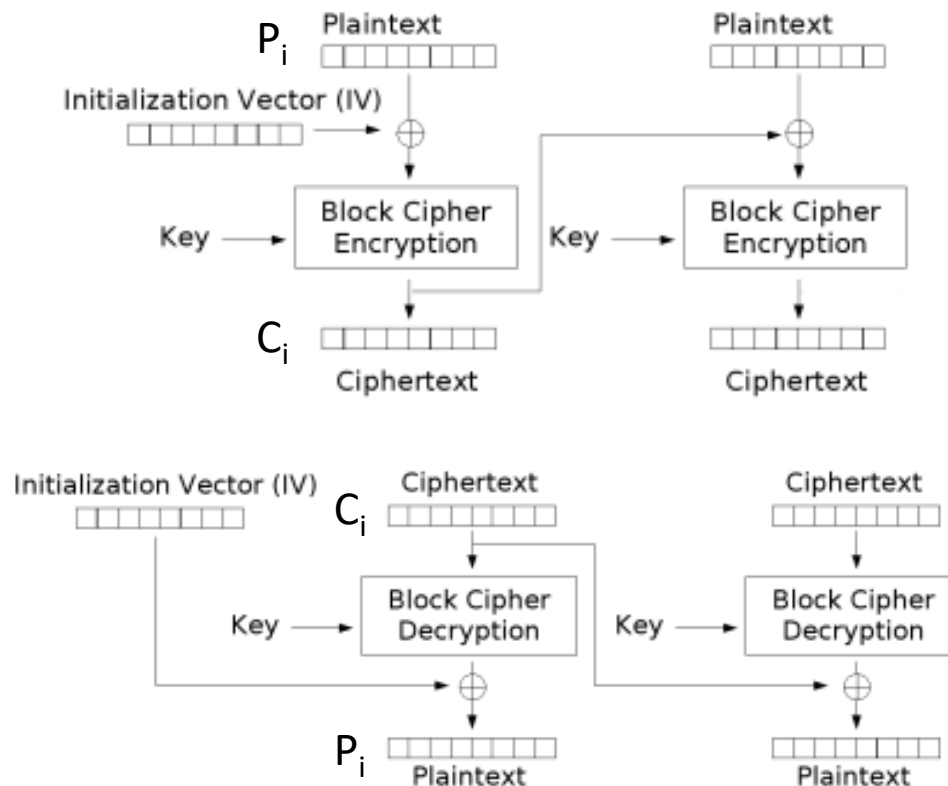
Block cipher modes: CBC

CBC (Cipher-Block Chaining)

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

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

$$C_0 = IV$$

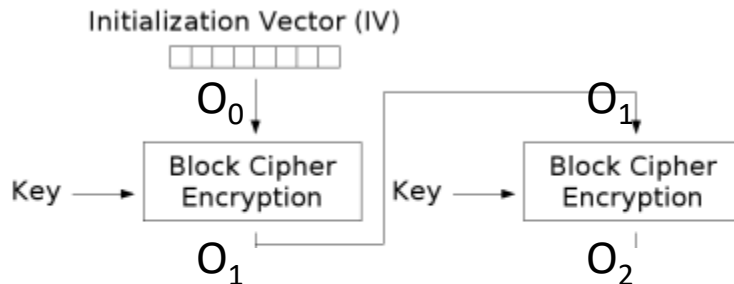


- **Strength:**
 - Repeated plaintext blocks result in different ciphered blocks
 - A ciphertext block depends on **all blocks** before it
- **Weakness:**
 - More complex
 - A corrupted bit in the ciphertext will affect all bits in its block and **one bit** in the next block

Block cipher modes: OFB / CTR

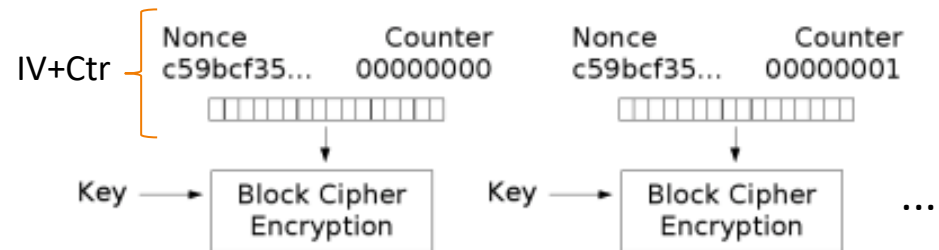
OFB (Output FeedBack)

$$\begin{aligned} C_i &= P_i \oplus O_i \\ P_i &= C_i \oplus O_i \end{aligned} \left. \vphantom{\begin{aligned} C_i &= P_i \oplus O_i \\ P_i &= C_i \oplus O_i \end{aligned}} \right\} \begin{array}{l} \text{Transforms} \\ \text{block cipher into} \\ \text{stream cipher} \end{array}$$
$$O_i = E_k(O_{i-1})$$
$$O_0 = IV$$



CTR (CounTer) mode or ICM (Integer Counter Mode)

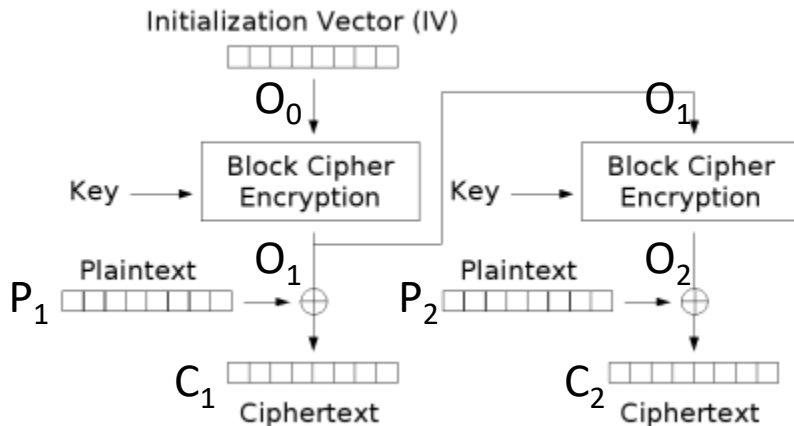
$$\begin{aligned} C_i &= P_i \oplus E_k(IV + Ctr_i) \\ P_i &= C_i \oplus E_k(IV + Ctr_i) \end{aligned} \left. \vphantom{\begin{aligned} C_i &= P_i \oplus E_k(IV + Ctr_i) \\ P_i &= C_i \oplus E_k(IV + Ctr_i) \end{aligned}} \right\} \begin{array}{l} \text{Transforms} \\ \text{block cipher into} \\ \text{stream cipher} \end{array}$$
$$Ctr_i = Ctr_i + 1$$



Block cipher modes: OFB / CTR

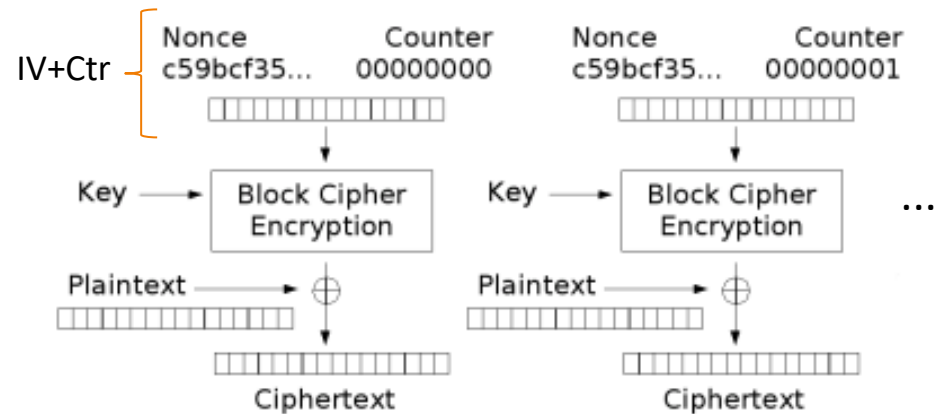
OFB (Output FeedBack)

$$\left. \begin{aligned} C_i &= P_i \oplus O_i \\ P_i &= C_i \oplus O_i \\ O_i &= E_k(O_{i-1}) \\ O_0 &= IV \end{aligned} \right\} \begin{array}{l} \text{Transforms} \\ \text{block cipher into} \\ \text{stream cipher} \end{array}$$



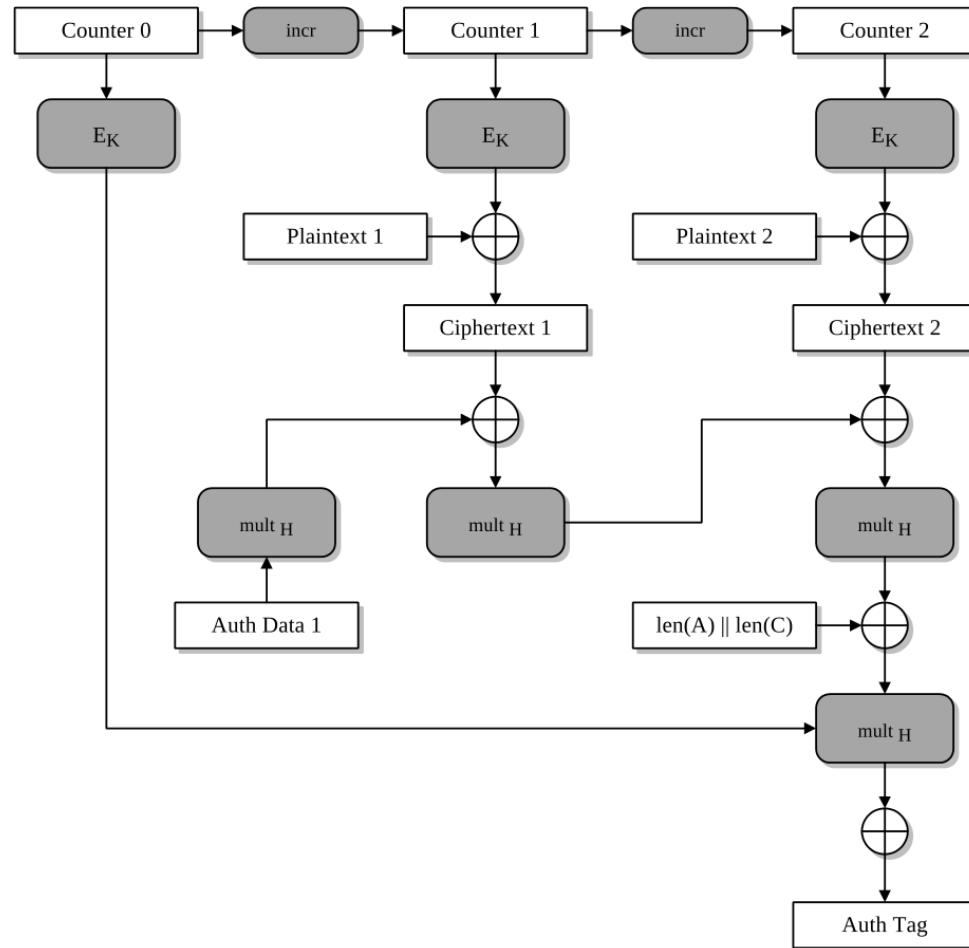
CTR (CounTer) mode or ICM (Integer Counter Mode)

$$\left. \begin{aligned} C_i &= P_i \oplus E_k(IV+Ctr_i) \\ P_i &= C_i \oplus E_k(IV+Ctr_i) \\ Ctr_i &= Ctr_i + 1 \end{aligned} \right\} \begin{array}{l} \text{Transforms} \\ \text{block cipher into} \\ \text{stream cipher} \end{array}$$



Block cipher modes: GCM

Galois Counter Mode



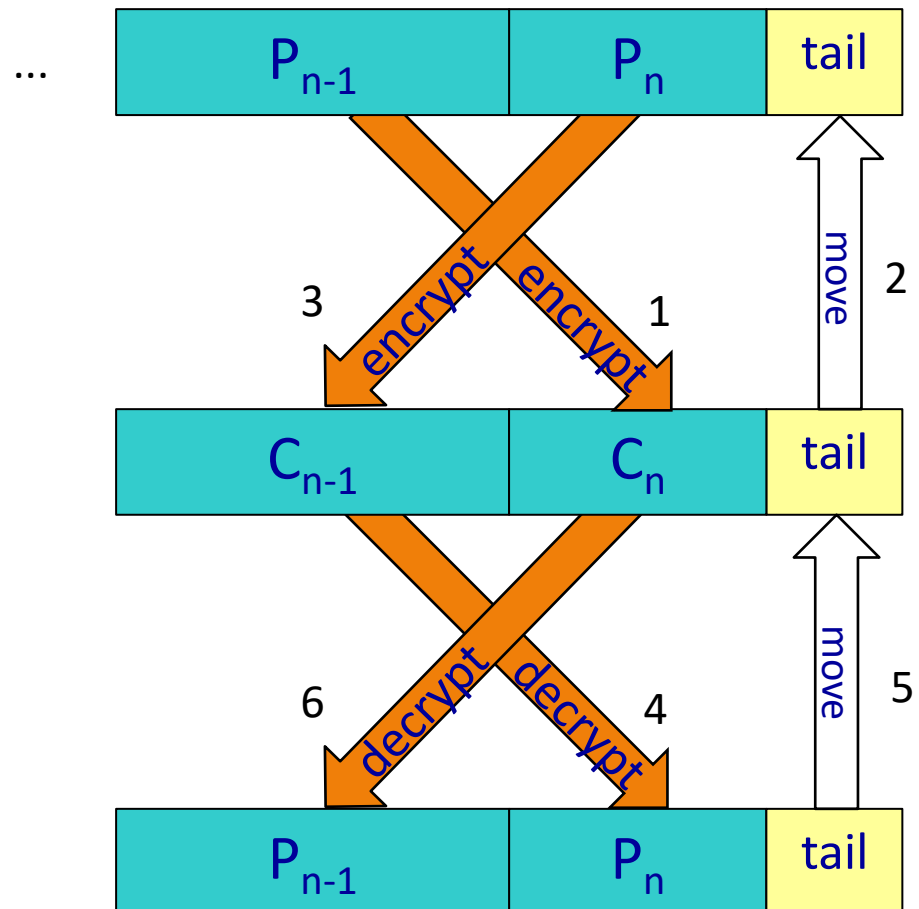
Padding Schemes

- $M \bmod B \neq 0 \Rightarrow$ **padding** of last block (M=message size; B=block size)
- Options:
 - Pad with zero (null) bytes, spaces (0x20), all bytes of the same value
 - Pad with random bits
 - Pad with 0x80 (1000 0000) followed by zero (null) characters
 - **PKCS#5 scheme**
 - Sequence of bytes, each of which equal to the number of padding bytes
 - Example: if 24 bits of padding need to be added, the padding string is "03 03 03" (3 bytes times 8 bits equals 24 bits)



Cipher Text Stealing: ECB mode

- Avoids the need of padding
- Plays with the last 2 blocks
- Benefit: no need to send the padding bits!



Roadmap

- Introduction
- Symmetric Ciphers
- **Hash Functions**
- Message Authentication Codes
- Asymmetric Ciphers
- Digital Signatures
- Homomorphic Ciphers

Hash functions / message digests

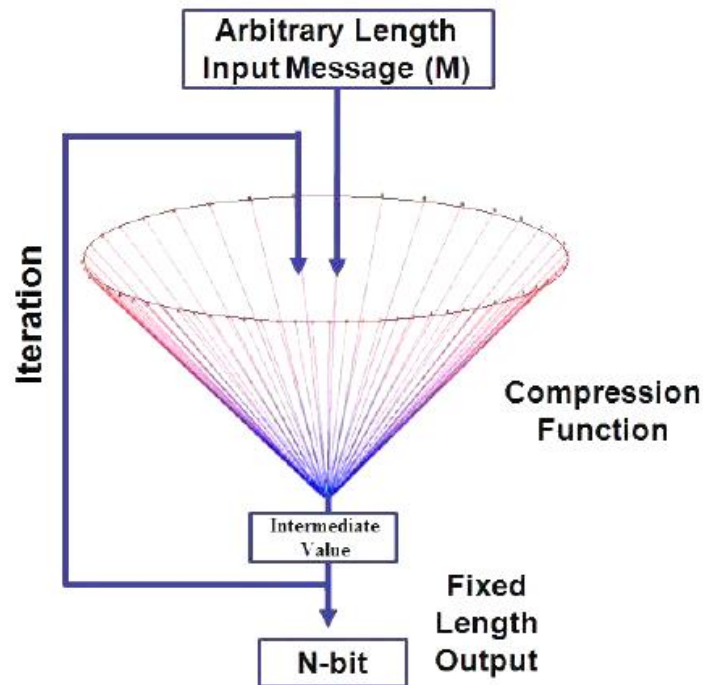
- Aka **cryptographic hash functions** or **collision-resistant hash functions**
 - don't confuse with hash functions used in hash tables (that you learned in IAED)
- They are cryptographic, but not ciphers
 - not used for encryption

Hash functions properties

- Generate very different output values – **hash** value – for similar inputs
- **One-way** (non-invertible):
 - Collision resistance
 - Computationally infeasible to find two inputs that give the same hash
 - Preimage resistance (strong collision resistance)
 - Given a hash, it's computationally infeasible to find an input that produces that hash
 - 2nd preimage resistance (weak collision resistance)
 - Given a hash value and the corresponding input, it's computationally infeasible to find a second input that generates that same hash

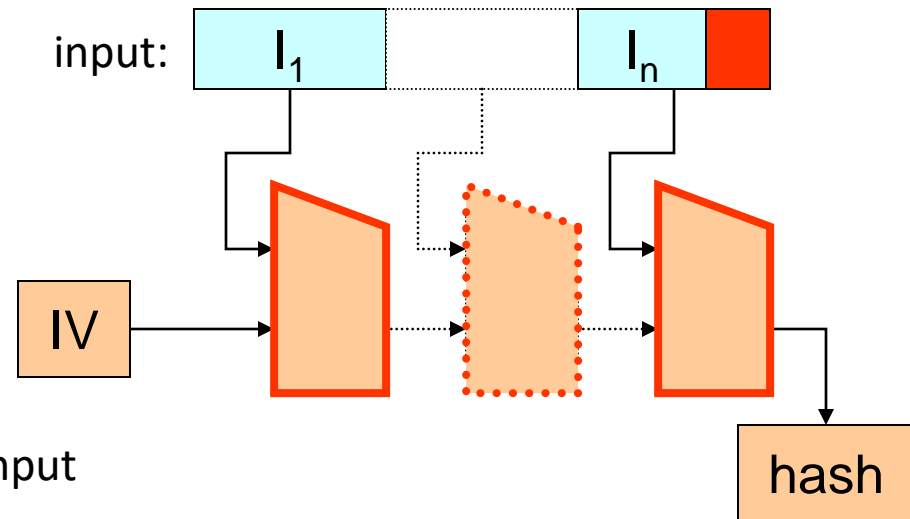
Hash functions

- Generate a fixed size value based on arbitrary size input
 - Iterative usage of a compression function with a fixed parameter input
 - The input text is aligned to the input blocks



Hash functions

- Some mechanisms used:
 - Shannon's diffusion & confusion
 - Iterative compression
 - MD Strengthening
 - Padding with 10000....000
 - Plus the number of bits of the input

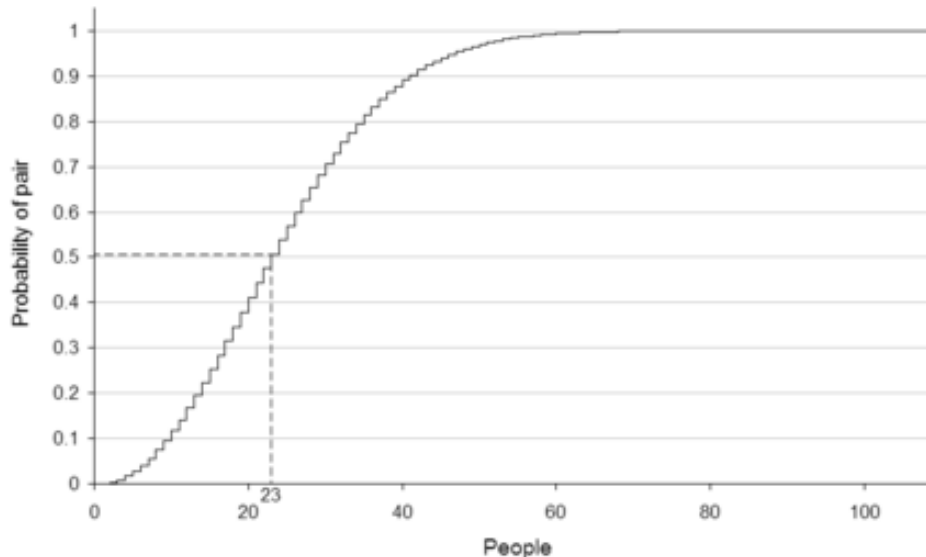


Hash functions

- Most used algorithms
 - Brute force attacks find collisions in $2^{m/2}$ tries, where m is the n. of bits
 - **MD5** (128 bits) *Very weak*
 - Collisions found in 2^{39} tries $\ll 2^{64}$
 - **SHA-1** (Secure Hash Algorithm, 160 bits) *Weak*
 - Collisions found in 2^{63} tries $\ll 2^{80}$
 - **SHA-2** (256 to 512 bits) *Ok*
 - Collisions found in 2^{128} or 2^{256} tries (secure, for now)
 - **SHA-3** (256 to 512 bits) *Ok*

Birthday paradox

- For there to be a 50% chance that someone in a room shares your birthday, you need 253 people
- However, for 50% chance that any two people in the room have the same birthday, you only need 23 people
 - It only takes 23 people to form 253 pairs when cross-matched



Birthday paradox

- Paradox

- *$P(\text{me})$: probability n students having the same birthday as me (“collision”)*

- Probability A’s birthday date is the same as mine = $1/365 \approx 0.003$
 - Probability A’s birthday date different from mine = $1 - 1/365 \approx 0.997$
 - Probability A and B’s dates different from mine = $(1 - 1/365)^2 \approx 0.994$
 - $P(\text{me}) = 1 - (1 - \frac{1}{365})^n$
 - $P(\text{me}) > 0,5 \Leftrightarrow n \geq 253$

- *$P(\text{pair})$: probability of pair with the same birthday*

- $P(\text{pair}) = 1 - (1 - \frac{1}{365})(1 - \frac{2}{365}) \dots (1 - \frac{n-1}{365})$
 - $P(\text{pair}) > 0,5 \Leftrightarrow n \geq 23 \quad !!!$

Hash functions attacks

- Objective is to find a **collision**
- **Brute forcing**
 - Attack: pick $H(M)$ and see if it's equal to $H(M')$, $H(M'')$, $H(M''')$,...
 - $P(\text{collision}) > 0.5 = 2^m/2 = 2^{m-1}$
 - where m is the number of bits of the hash function
- **Birthday attack**
 - Allows finding collision much faster
 - Attack: pick M , M' , M'' , M''' ... and obtain hashes until any 2 are identical
 - Finding 2 messages with $H(M) = H(M')$:
 - $P(\text{collision}) > 0.5 \approx 2^{m/2}$ tries (only)
- Cryptanalysis leads to even faster attacks

SHA-1 is not secure

SHAttered

The first concrete collision attack against SHA-1
<https://shattered.io>



Marc Stevens
Pierre Karpman



Elie Bursztein
Ange Albertini
Yarik Markov

SHAttered

The first concrete collision attack against SHA-1
<https://shattered.io>



Marc Stevens
Pierre Karpman



Elie Bursztein
Ange Albertini
Yarik Markov

Different PDF files, same SHA-1 hash

Transitioning the Use of Cryptographic Algorithms and Key Lengths

Elaine Barker
Allen Roginsky

This publication is available free of charge from:
<https://doi.org/10.6028/NIST.SP.800-131Ar2>

C O M P U T E R S E C U R I T Y



Table 8: Approval Status of Hash Functions

Hash Function	Use	Status
SHA-1	Digital signature generation	Disallowed, except where specifically allowed by NIST protocol-specific guidance.
	Digital signature verification	Legacy use
	Non-digital-signature applications	Acceptable
SHA-2 family (SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224 and SHA-512/256)	Acceptable for all hash function applications	
SHA-3 family (SHA3-224, SHA3-	Acceptable for all hash function applications	

Roadmap

- Introduction
- Symmetric Ciphers
- Hash Functions
- **Message Authentication Codes**
- Asymmetric Ciphers
- Digital Signatures
- Homomorphic Ciphers

Message Authentication Code (MAC)

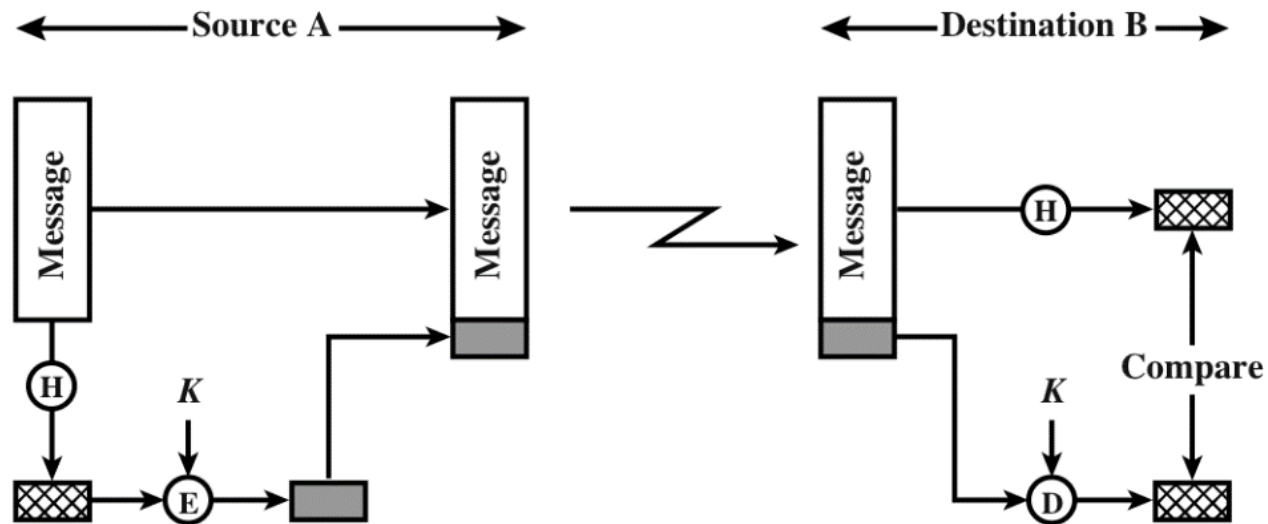
- Objective: **authenticating** a message or equivalent
 - Allows checking its authenticity and integrity
- Assumption: sender and recipient have a **shared secret key K**
- Idea:
 - Send the message + MAC
 - If the message (or the MAC) is modified by an attacker, the recipient will be able to detect it
 - Attacker can't create a valid MAC because he doesn't have K
- What about CRCs and checksums?
 - Attacker can modify the message and CRC/checksum accordingly, as he knows the algorithm and there is no secret involved

Message Authentication Code (MAC)

- MAC is a hash generated using a secret key
 - 1- Hash the message and encrypt the digest
 - For example, with a symmetric block cipher
 - 2- Using a keyed-function
 - ANSI X9.9 (a.k.a. DES-MAC) with DES-CBC (64 bits) – next
 - 3- Hash the message along with a shared key
 - Keyed-MD5 (128 bits)
 - HMAC (size of the used hash function)

Message Authentication Code (MAC)

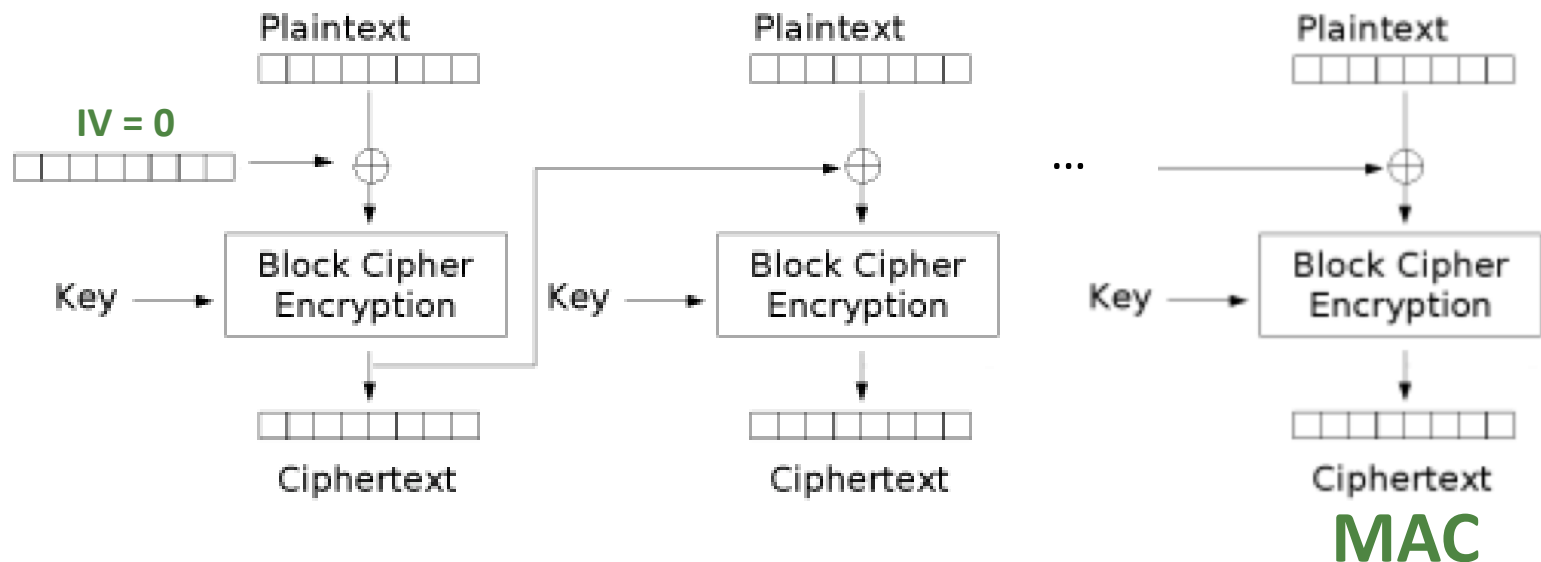
- 1- Hash the message and encrypt the digest



If attacker modifies the message (or the hash), the hash calculated at the destination won't match the hash received

Message Authentication Code (MAC)

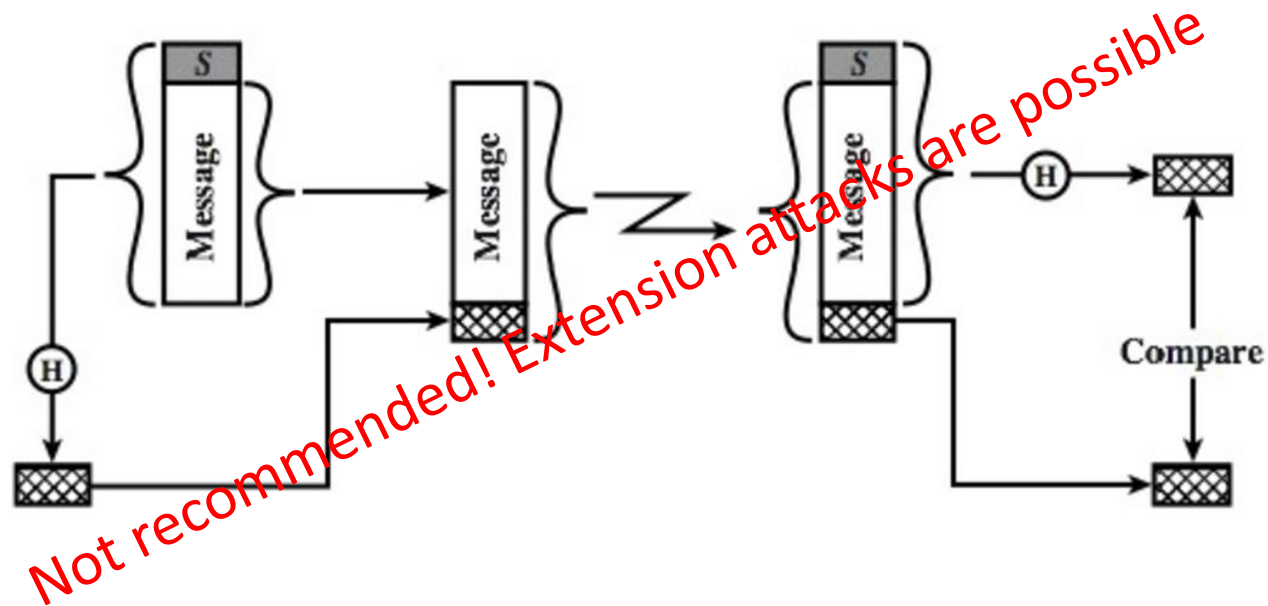
- 2- CMAC (Cipher-based MAC)
 - Use a block cipher in CBC mode (*use a secure cipher!*)
 - MAC is the last block
 - Must be CBC for MAC to depend on the whole message



At the destination do the same and check if the MAC obtained is the same

Message Authentication Code (MAC)

- 3- Hash the message along with a shared key
 - Ad hoc algorithm showing the basic idea:



Message Authentication Code (MAC)

- 3- HMAC algorithm
 - FIPS Standard / RFC 2104: keyed hash MAC
 - $\text{HMAC}(m,k) = \text{hash}(k \oplus \text{opad} || \text{hash}(k \oplus \text{ipad} || m))$
 - ipad – inner padding – 0x36363636
 - opad – outer padding – 0x5c5c5c5c
 - HMAC in practice: SSL/TLS, WTLS, IPSec

Transitioning the Use of Cryptographic Algorithms and Key Lengths

Elaine Barker
Allen Roginsky







This publication is available free of charge from:
<https://doi.org/10.6028/NIST.SP.800-131Ar2>

COMPUTER SECURITY



CMAC, GMAC, KMAC
são standards do NIST

Table 9: Approval Status of MAC Algorithms

MAC Algorithm	Implementation Details	Status
HMAC Generation	Key lengths < 112 bits	Disallowed
	Key lengths \geq 112 bits	Acceptable 
HMAC Verification	Key lengths < 112 bits	Legacy use
	Key lengths \geq 112 bits	Acceptable 
CMAC Generation	Two-key TDEA	Disallowed
	Three-key TDEA	Deprecated through 2023 Disallowed after 2023
	AES	Acceptable 
CMAC Verification	Two-key TDEA	Legacy use
	Three-key TDEA	Legacy use
	AES	Acceptable 
GMAC Generation and Verification	AES	Acceptable 
KMAC Generation and Verification	Key lengths < 112 bits	Disallowed
	Key lengths \geq 112 bits	Acceptable 

Roadmap

- Introduction
- Symmetric Ciphers
- Hash Functions
- Message Authentication Codes
- **Asymmetric Ciphers**
- Digital Signatures
- Homomorphic Ciphers

Asymmetric ciphers: overview

- Aka **public-key cryptography**
- Appeared with the objective of solving a hard problem: **key distribution**
- They use a **pair of keys**
 - One is **private**, personal, non-transmissible
 - One is **public**, can/should be widely known
- Allow for
 - **Confidentiality** with the exchange of secret keys
 - **Authentication** with digital signatures
 - **Integrity** with digital signatures

Asymmetric ciphers: assessment

- **Advantages**

- N communicating parties \Rightarrow N key pairs
- instead of $N(N-1)/2$ as symmetric ciphers

- **Disadvantages**

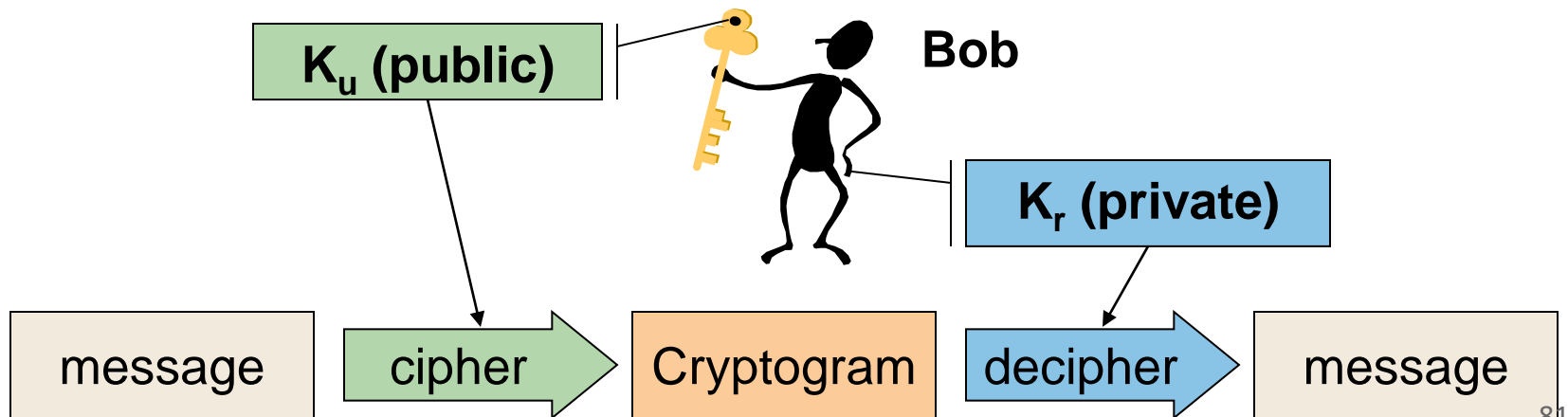
- Performance: typically inefficient in comparison to symmetric ciphers (but signatures are not too slow today)

- **Problems**

- Distribution of public keys, although much simpler than distributing secret keys
- Life-time of the key pairs (revocation)

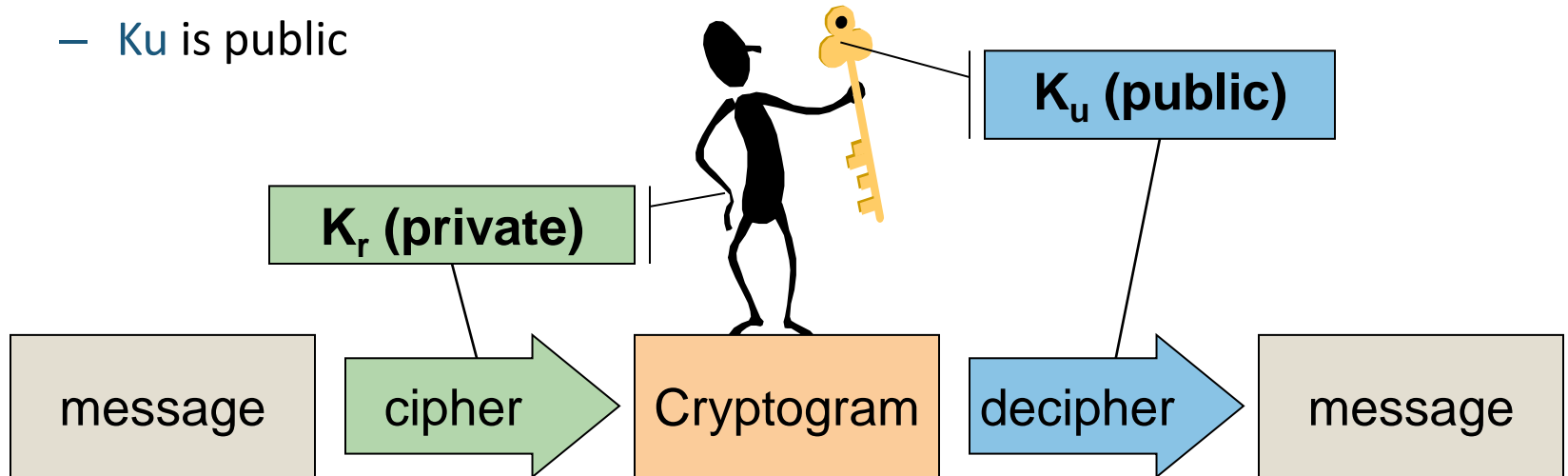
Asymmetric ciphers: Confidentiality

- Cipher / decipher with a pair of keys
 - $C = E(K_u, P)$ $P = D(K_r, C)$
 - To communicate with confidentiality with X we only need to know K_u
 - Very inefficient for large P, so not used like this for encrypting messages/files
- No authentication
 - Bob does not know who generated the cryptogram

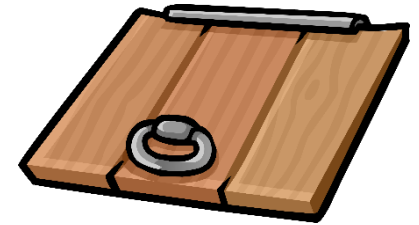


Asymmetric ciphers: Authentication

- The cryptogram cannot be modified
 - $C = E(K_r, P)$ $P = D(K_u, C)$
 - Only X knows the key K_r used to generate the cryptogram
 - Better idea to encrypt $H(P)$: $C = E(K_r, H(P))$
- No confidentiality
 - Whoever knows the key K_u is able to decipher the cryptogram
 - K_u is public



Trapdoor functions



- Functions with the properties:
 - Computing in one direction is easy $A \rightarrow B$
 - Computing the inverse is very hard (unless you know a secret) $A \nleftarrow B$
- Trapdoor functions are the core concept of Asymmetric cryptography
 - If you find mathematical functions with these properties, you can build a cryptographic scheme around it

Asymmetric Ciphers

- Used mathematical approximations
 - Factorization of large numbers
 - Discrete logarithm problem
 - Knapsack problem
- Most common encryption algorithms
 - RSA (Rivest-Shamir-Adleman) – first and most used asymmetric encryption algorithm
 - ECC (Elliptic Curve Cryptography) – becoming frequently used today
- Other public-key techniques (not encryption)
 - Diffie-Hellman (DH), also known as Diffie-Hellman-Merkle – originated the area of asymmetric crypto

Modular arithmetic

- Idea: do arithmetic normally ($+$ $-$ \times \div) but result is the mod of what you obtained dividing by a modulus
 - i.e., the result is the remainder of the integer division of the result by the modulus
 - Performs arithmetic operations not on a line but on a circle
 - mod can be applied “as often as you want”
 - Examples:
 - $1537 \times 4248 \pmod{10} = 6529176 \pmod{10} = 6$
 - $1537 \pmod{10} \times 4248 \pmod{10} = 7 \times 8 \pmod{10} = 56 \pmod{10} = 6$

RSA

- RSA (Rivest-Shamir-Adleman)
- First **asymmetric encryption** algorithm
 - But not the first asymmetric cryptography algorithm (that was Diffie-Hellman-Merkle)
- A major step for cryptography

Programming
Techniques

S.L. Graham, R.L. Rivest*
Editors

A Method for Obtaining Digital Signatures and Public-Key Cryptosystems

R. L. Rivest, A. Shamir, and L. Adleman
MIT Laboratory for Computer Science
and Department of Mathematics

An encryption method is presented with the novel property that publicly revealing an encryption key does not thereby reveal the corresponding decryption key. This has two important consequences:

(1) Couriers or other secure means are not needed to transmit keys, since a message can be enciphered using an encryption key publicly revealed by the intended recipient. Only he can decipher the message, since only he knows the corresponding decryption key. (2) A message can be "signed" using a privately held decryption key. Anyone can verify this signature using the corresponding publicly revealed encryption key. Signatures cannot be forged, and a signer cannot later deny the validity of his signature. This has obvious applications in "electronic mail" and "electronic funds transfer" systems. A message is encrypted by representing it as a number M , raising M to a publicly specified power e , and then taking the remainder when the result is divided by the publicly specified product, n , of two large secret prime numbers p and q . Decryption is similar; only a different, secret, power d is used, where $e * d = 1 \pmod{(p - 1) * (q - 1)}$. The security of the system rests in part on the difficulty of factoring the published divisor, n .

Key Words and Phrases: digital signatures, public-key cryptosystems, privacy, authentication, security, factorization, prime number, electronic mail, message-passing, electronic funds transfer, cryptography.

CR Categories: 2.12, 3.15, 3.50, 3.81, 5.25

General permission to make fair use in teaching or research of all or part of this material is granted to individual readers and to nonprofit libraries acting for them provided that ACM's copyright notice is given and that reference is made to the publication, to its date of issue, and to the fact that reprinting privileges were granted by permission of the Association for Computing Machinery. To otherwise reprint a figure, table, other substantial excerpt, or the entire work requires specific permission as does republication, or systematic or multiple reproduction.

This research was supported by National Science Foundation grant MCS76-14294, and the Office of Naval Research grant number N00014-67-A-0204-0063.

* Note. This paper was submitted prior to the time that Rivest became editor of the department, and editorial consideration was completed under the former editor, G. K. Manacher.

Authors' Address: MIT Laboratory for Computer Science, 545 Technology Square, Cambridge, MA 02139.

© 1978 ACM 0001-0782/78/0200-0120 \$00.75

I. Introduction

The era of "electronic mail" [10] may soon be upon us; we must ensure that two important properties of the current "paper mail" system are preserved: (a) messages are *private*, and (b) messages can be *signed*. We demonstrate in this paper how to build these capabilities into an electronic mail system.

At the heart of our proposal is a new encryption method. This method provides an implementation of a "public-key cryptosystem", an elegant concept invented by Diffie and Hellman [1]. Their article motivated our research, since they presented the concept but not any practical implementation of such a system. Readers familiar with [1] may wish to skip directly to Section V for a description of our method.

II. Public-Key Cryptosystems

In a "public-key cryptosystem" each user places in a public file an encryption procedure E . That is, the public file is a directory giving the encryption procedure of each user. The user keeps secret the details of his corresponding decryption procedure D . These procedures have the following four properties:

(a) Deciphering the enciphered form of a message M yields M . Formally,

$$D(E(M)) = M. \quad (1)$$

(b) Both E and D are easy to compute.

(c) By publicly revealing E the user does not reveal an easy way to compute D . This means that in practice only he can decrypt messages encrypted with E , or compute D efficiently.

(d) If a message M is first deciphered and then enciphered, M is the result. Formally,

$$E(D(M)) = M. \quad (2)$$

An encryption (or decryption) procedure typically consists of a *general method* and an *encryption key*. The general method, under control of the key, enciphers a message M to obtain the enciphered form of the message, called the *ciphertext* C . Everyone can use the same general method; the security of a given procedure will rest on the security of the key. Revealing an encryption algorithm then means revealing the key.

When the user reveals E he reveals a *very inefficient* method of computing $D(C)$: testing all possible messages M until one such that $E(M) = C$ is found. If property (c) is satisfied the number of such messages to test will be so large that this approach is impractical.

A function E satisfying (a)-(c) is a "trap-door one-way function;" if it also satisfies (d) it is a "trap-door one-way permutation." Diffie and Hellman [1] introduced the concept of trap-door one-way functions but

RSA overview

- **Base values and modulus**
 - p, q – **secret**; large **prime** numbers
 - $n = p \times q$ – **public**; large (1K, 2K, 3K bits); used as **modulus**; key length
- **Public key** – exponent e and n ($e < n$)
- **Private key** – exponent d ($d < n$) and primes p and q
- **Key generation**: choose p, q large primes
 - e chosen such that it is coprime with $z = \phi(n) = (p-1)(q-1)$
 - *coprime* means that the only factor that divides both is 1
 - $\phi(n)$ is the Euler's totient function; counts factors up to n that are coprime with n
 - d chosen such that $e \cdot d \equiv 1 \pmod{z} \iff e \cdot d \bmod z = 1$
- **Encryption with public key**: $C = P^e \bmod n$
- **Decryption with private key**: $P = C^d \bmod n$
 $= (P^e \bmod n)^d \bmod n = P^{ed} \bmod n = P^{ed \bmod z} \bmod n = P^1 \bmod n = P$

Public key example

- Taken from a browser

Public Key Info

Algorithm RSA Encryption (1.2.840.113549.1.1.1)

Parameters None

Public Key

= modulus

256 bytes: D0 18 CF 45 D4 8B CD D3 9C E4 40
EF 7E B4 DD 69 21 1B C9 CF 3C 8E 4C 75 B9 0F
31 19 84 3D 9E 3C 29 EF 50 0D 10 93 6F 05 80
80 9F 2A A0 BD 12 4B 02 E1 3D 9F 58 16 24 FE
30 9F 0B 74 77 55 93 1D 4B F7 4D E1 92 82 10 F6
51 AC 0C C3 B2 22 94 0F 34 6B 98 10 49 E7 0B
9D 83 39 DD 20 C6 1C 2D EF D1 18 61 65 E7 23
83 20 A8 23 12 FF D2 24 7F D4 2F E7 44 6A 5B
4D D7 50 66 B0 AF 9E 42 63 05 FB E0 1C C4 63
61 AF 9F 6A 33 FF 62 97 BD 48 D9 D3 7C 14 67
DC 75 DC 2E 69 E8 F8 6D 78 69 D0 B7 10 05 B8
F1 31 C2 3B 24 FD 1A 33 74 F8 23 E0 EC 6B 19 8A
16 C6 E3 CD A4 CD 0B DB B3 A4 59 60 38 88 3B
AD 1D B9 C6 8C A7 53 1B FC BC D9 A4 AB BC DD
3C 61 D7 93 15 98 EE 81 BD 8F E2 64 47 20 40
06 4E D7 AC 97 E8 B9 C0 59 12 A1 49 25 23 E4
ED 70 34 2C A5 B4 63 7C F9 A3 3D 83 D1 CD 6D
24 AC 07

exponent e

Exponent 65537 = 1 0000 0000 0000 0001 b

Key Size 2 048 bits = size of the modulus

RSA example with small numbers

- Key generation

- Pick primes **p** and **q**: 3 and 11
- Calculate modulus **n** = $p \times q = 3 \times 11 = 33$
- Calculate **z** = $\phi(n) = (p-1) \times (q-1) = 2 \times 10 = 20$
- Choose **e** such that **e** is co-prime with **z** and $e < n$: $e = 7$
- Public key: **e** = 7, **n** = 33
- Calculate **d**
 - $e \cdot d = 1 \pmod{z}$ and $d < n \Leftrightarrow$
 - $7 \cdot d \pmod{20} = 1$ and $d < 33 \Rightarrow$
 - **d** = 3

- Encrypt w/ public key **P** = 14

- $C = P^e \pmod{n}$
- $C = 14^7 \pmod{33}$
- $C = 20$

- Decrypt w/ private key

- $P = C^d \pmod{n}$
- $P = 20^3 \pmod{33}$
- $P = 14$

Why is RSA difficult to break?

- No one knows how to compute the **inverse RSA** ("decryption") without knowing the prime factors of the modulus **n**
- No one knows how to efficiently do **integer factorization**, i.e., recover the prime factors from **n** alone
 - The private key is knowledge of the prime factors
- Knowing the **public key** does not allow decryption
- Future crisis: **Shor's quantum algorithm** for integer factorization
 - Current quantum technology can only factorize small numbers
 - ≥ 2048 -bit numbers are safe (for now...)
 - **Post-quantum** crisis; need for **quantum resistance**

RSA signatures

- RSA is often used to produce **signatures**
 - Public key – e, n
 - Private key – d, p, q
- **Signing the message M :**
 - Obtain S such that: $S = (\text{hash}(M))^d \bmod n$
 - Only the private key owner can generate the signature
- **Verifying the signature:**
 - Check if: $\text{hash}(M) == S^e \bmod n$
 - Anyone with the public key can verify the signature

Elliptic Curve Cryptography (ECC)

- RSA and DH use integer and polynomial arithmetic with very large numbers/polynomials
 - Impose a significant load in storing and processing keys and messages
- **Elliptic curves** are an alternative
 - Offer same security with smaller bit sizes
 - 224-bit ECC ~ 2048-bit RSA
 - 500-bit ECC ~ 15k-bit RSA
 - “Newer”, not as well analyzed, but commonly used
 - Many curves exist, each with their tradeoffs
 - Not necessarily faster than RSA for the same equivalent security
 - Normally used for **key agreement or signing**, not ciphering

Elliptic Curve Cryptography (ECC)

- Analogy:
 - ECC addition is analogous to modulo multiplication
 - ECC repeated addition is analogous to modulo exponentiation
- The “hard” problem is the **elliptic curve logarithm problem**
 - $Q = k \cdot P$, where Q, P belong to an elliptic curve; eg $y^2 = x^3 - 3x + b \text{ modulo } p$
 - Easy to compute Q given k, P
 - Hard to find k given Q, P
- Public Key operations $Q(x,y) = kP(x,y)$:
 - Q = public key
 - k = private key
 - P = base point (a parameter of the curve)

ECC Encryption/Decryption

- There are several alternatives but we consider the simplest: to encode the message **M** as a point **Q_M** on the elliptic curve
- Key generation
 - Select suitable curve and base point P
 - Bob chooses a **private key**: $k_B < n$
 - Bob computes the **public key**: $Q_B = k_B \cdot P$
- Alice **encrypts** **Q_M** using Bob's **public key** **Q_B**:
 $C_M = \{r \cdot P, Q_M + r \cdot Q_B\}$ – r is a random number; C_M is a point in the curve
- Bob **decrypts** **C_M** using its **private key** **k_B**:

$$\underbrace{Q_M + r \cdot Q_B - k_B r \cdot P}_{C_M} = Q_M + r(k_B \cdot P) - k_B(r \cdot P) = Q_M$$

Since $Q_B = k_B P$

Roadmap

- Introduction
- Symmetric Ciphers
- Hash Functions
- Message Authentication Codes
- Asymmetric Ciphers
- **Digital Signatures**
- Homomorphic Ciphers

Digital signatures – objectives

- Authenticate the content of a document
- Authenticate its signer
- Being able to assure authentication towards a third party
- Signer cannot repudiate the signature



Legislação Consolidada

Decreto-Lei n.º 290-D/99 - Diário da República n.º 178/1999, 1º Suplemento, Série I-A de 1999-08-02

Aprova o regime jurídico dos documentos electrónicos e da assinatura digital

Assinatura Digital

A assinatura digital qualificada permite ao titular de um Cartão de Cidadão, por vontade própria, assinar com a chave pessoal existente no seu Cartão de Cidadão.

Qualquer entidade pode verificar a assinatura digital recorrendo ao uso do certificado digital pessoal do cidadão e a meios de verificação da validade deste certificado.

CMD Assinatura

A assinatura eletrónica qualificada através da Chave Móvel Digital (CMD) permite ao cidadão, português ou estrangeiro, por vontade própria, assinar com a palavra-chave por si escolhida e respetivo código de segurança.

Digital signatures

- Used approximations: asymmetric cipher + hash function
- **Basic algorithm** (K_x^r - private key K_x^u - public key)
 - Signature: $S_x(\text{doc}) = E(K_x^r, \text{hash}(\text{doc}))$
 - Validation: check if $D(K_x^u, S_x(\text{doc})) = \text{hash}(\text{doc})$
- Why can't adversary provide a fake signature?
- Another option:
 - Signature: $S_x(\text{doc}) = \text{info} + E(K_x^r, \text{hash}(\text{doc} + \text{info}))$
 - Validation: $D(K_x^u, S_x(\text{doc})) = \text{hash}(\text{doc} + \text{info})$
 - Example **info**: timestamp, used algorithm, ...
- Today often **ECDSA** is adopted (with elliptic curve encryption)

Transitioning the Use of Cryptographic Algorithms and Key Lengths



Elaine Barker
Allen Roginsky

This publication is available free of charge from:
<https://doi.org/10.6028/NIST.SP.800-131Ar2>

C O M P U T E R S E C U R I T Y



**Table 2: Approval Status of Algorithms Used for Digital Signature
Generation and Verification**

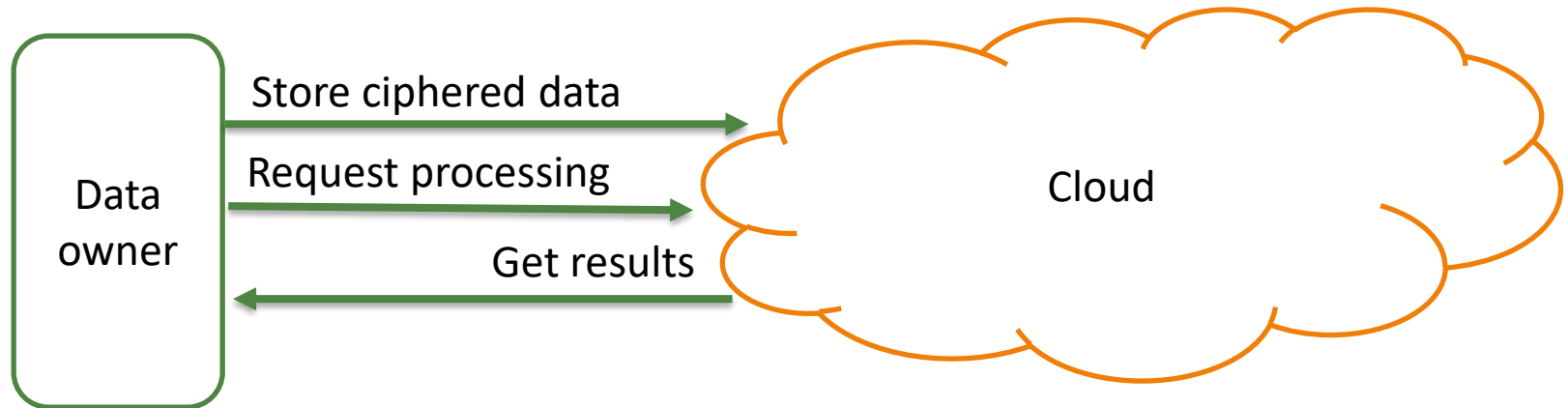
Digital Signature Process	Domain Parameters	Status
Digital Signature Generation	<p>< 112 bits of security strength:</p> <p>DSA: $(L, N) \neq (2048, 224), (2048, 256)$ or $(3072, 256)$</p> <p>ECDSA: $\text{len}(n) < 224$</p> <p>RSA: $\text{len}(n) < 2048$</p>	Disallowed
	<p><u>≥ 112 bits of security strength:</u></p> <p>DSA: $(L, N) = (2048, 224), (2048, 256)$ or $(3072, 256)$</p> <p>ECDSA or EdDSA: $\text{len}(n) \geq 224$</p> <p>RSA: $\text{len}(n) \geq 2048$</p>	Acceptable 
Digital Signature Verification	<p>< 112 bits of security strength:</p> <p>DSA³²: $((512 \leq L < 2048)$ or $(160 \leq N < 224))$</p> <p>ECDSA: $160 \leq \text{len}(n) < 224$</p> <p>RSA: $1024 \leq \text{len}(n) < 2048$</p>	Legacy use
	<p><u>≥ 112 bits of security strength:</u></p> <p>DSA: $(L, N) = (2048, 224), (2048, 256)$ or $(3072, 256)$</p> <p>ECDSA and EdDSA: $\text{len}(n) \geq 224$</p> <p>RSA: $\text{len}(n) \geq 2048$</p>	Acceptable 

Roadmap

- Introduction
- Symmetric Ciphers
- Hash Functions
- Message Authentication Codes
- Asymmetric Ciphers
- Digital Signatures
- **Homomorphic Ciphers**

Homomorphic Encryption

- How to **outsource the processing of ciphered data**?
 - Basic assumption: data processed is encrypted **with the same key**
 - Data is processed in the cloud, not downloaded before processing



Homomorphic Encryption

- **Homomorphic encryption**
 - Generate a ciphered result that, when deciphered, matches the result of the operations as if they had been performed on the plaintext
- **Fully homomorphic encryption (FHE)**
 - A cryptosystem that allows arbitrary computation on ciphertexts, i.e., that allow creating programs with any functionality
 - First implemented by Gentry in 2009
 - Currently very inefficient
- In practice, aim lower, e.g., queries over encrypted databases, e.g., **CryptDB** – <https://css.csail.mit.edu/cryptdb/>

Homomorphic Encryption

Example practical schemes (1/2)

- **Deterministic encryption** – allows equality comparison
 - Encrypt with some deterministic symmetric cipher E , e.g., AES in CBC mode with **fixed IV**
 - If $E(A, K) = E(B, K)$ then $A = B$
 - Less secure than randomized scheme (random IV), reveals equality
- **Order Preserving Encryption** – allows inequality comparison
 - Less secure than deterministic as it reveals order

Homomorphic Encryption

Example practical schemes (2/2)

- **Multiplication using RSA**

- Both keys are kept **secret**: **e**, **d**, **p**, **q** (even **e** that is usually **public**)
- Recall encryption with RSA: $C = P^e \bmod n$
- Consider two pieces of data encrypted with **e**: C_a and C_b
- Multiplication: $C_a \times C_b = P_a^e \times P_b^e \bmod n = (P_a \times P_b)^e \bmod n$
- Decrypting with **d** we get $P_a \times P_b$
- i.e., we multiply two encrypted values and, when we decrypt them, we get the multiplication of the two plaintext numbers

Summary

- Introduction
- Symmetric Ciphers
- Hash Functions
- Message Authentication Codes
- Asymmetric Ciphers
- Digital Signatures
- Homomorphic Ciphers