DI-FCT-UNL
Segurança de Redes e Sistemas de Computadores
Network and Computer Systems Security

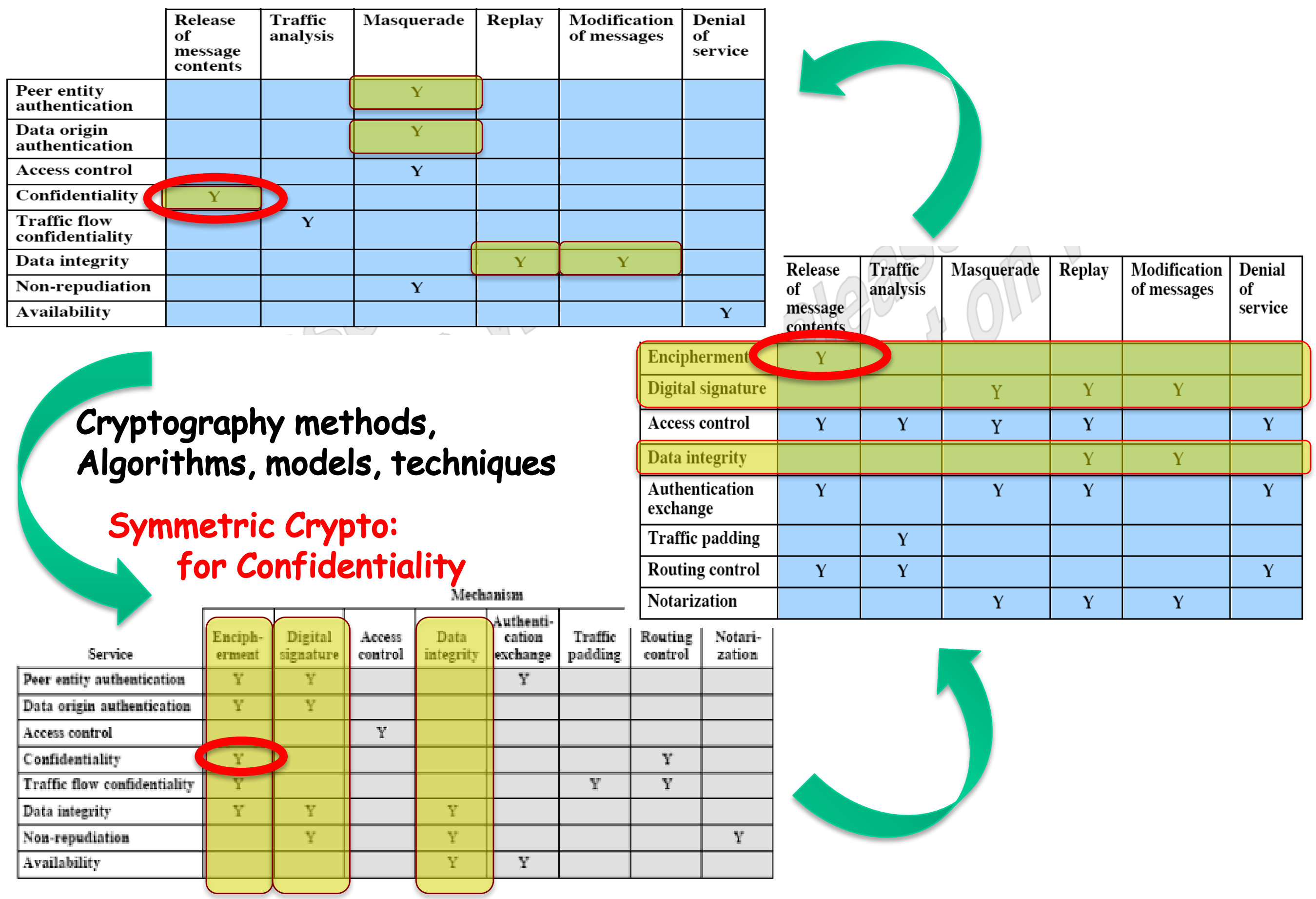
Mestrado Integrado em Engenharia Informática
MSc Course: Informatics Engineering
1st Sem, 2020/2021

More on Symmetric Cryptography

Last lecture …

Overview on Cryptography vs. OSI X.800 Framework

Different methods and algorithms, including Symmetric Cryptography, as Specific Security Mechanisms



Cryptography methods, Algorithms, models, techniques
Symmetric Crypto: for Confidentiality

	Release of message contents	Traffic analysis	Masquerade	Replay	Modification of messages	Denial of service
Peer entity authentication			Y			
Data origin authentication			Y			
Access control			Y			
Confidentiality	Y					
Traffic flow confidentiality		Y				
Data integrity				Y	Y	
Non-repudiation			Y			
Availability						Y

	Release of message contents	Traffic analysis	Masquerade	Replay	Modification of messages	Denial of service
Encipherment	Y					
Digital signature			Y	Y	Y	
Access control	Y	Y	Y	Y		Y
Data integrity				Y	Y	
Authentication exchange	Y		Y	Y		Y
Traffic padding		Y				
Routing control	Y	Y				Y
Notarization			Y	Y	Y	

	Mechanism							
Service	Encipherment	Digital signature	Access control	Data integrity	Authentication exchange	Traffic padding	Routing control	Notarization
Peer entity authentication	Y	Y			Y			
Data origin authentication	Y	Y						
Access control			Y					
Confidentiality	Y						Y	
Traffic flow confidentiality	Y					Y	Y	
Data integrity	Y	Y		Y				
Non-repudiation		Y		Y				Y
Availability				Y	Y			

Symmetric Cryptography Scope (ref. w/ OSI X.800 Framework)

Also named *conventional encryption or shared-key cryptography*

- Primary use: for Confidentiality (Encryption)

  Target: Data, Data-Storage or Message Confidentiality

  (Security Properties for Secure Channels, - rem. X.800 Framework)

  - Connection-Oriented Confidentiality
  - Connectionless or Datagram Oriented Confidentiality
  - Selective Field Confidentiality
  - Traffic-Flow Confidentiality

Symmetric Cryptography Scope (ref. w/ OSI X.800 Framework)

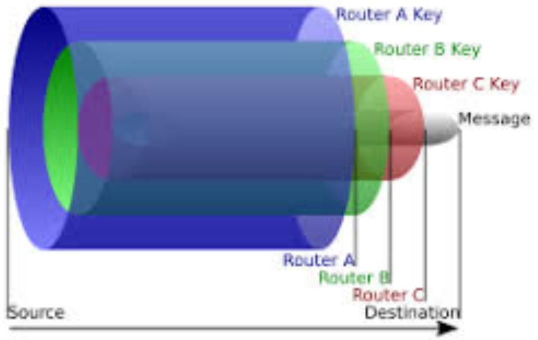
Also named *conventional encryption or shared-key cryptography*

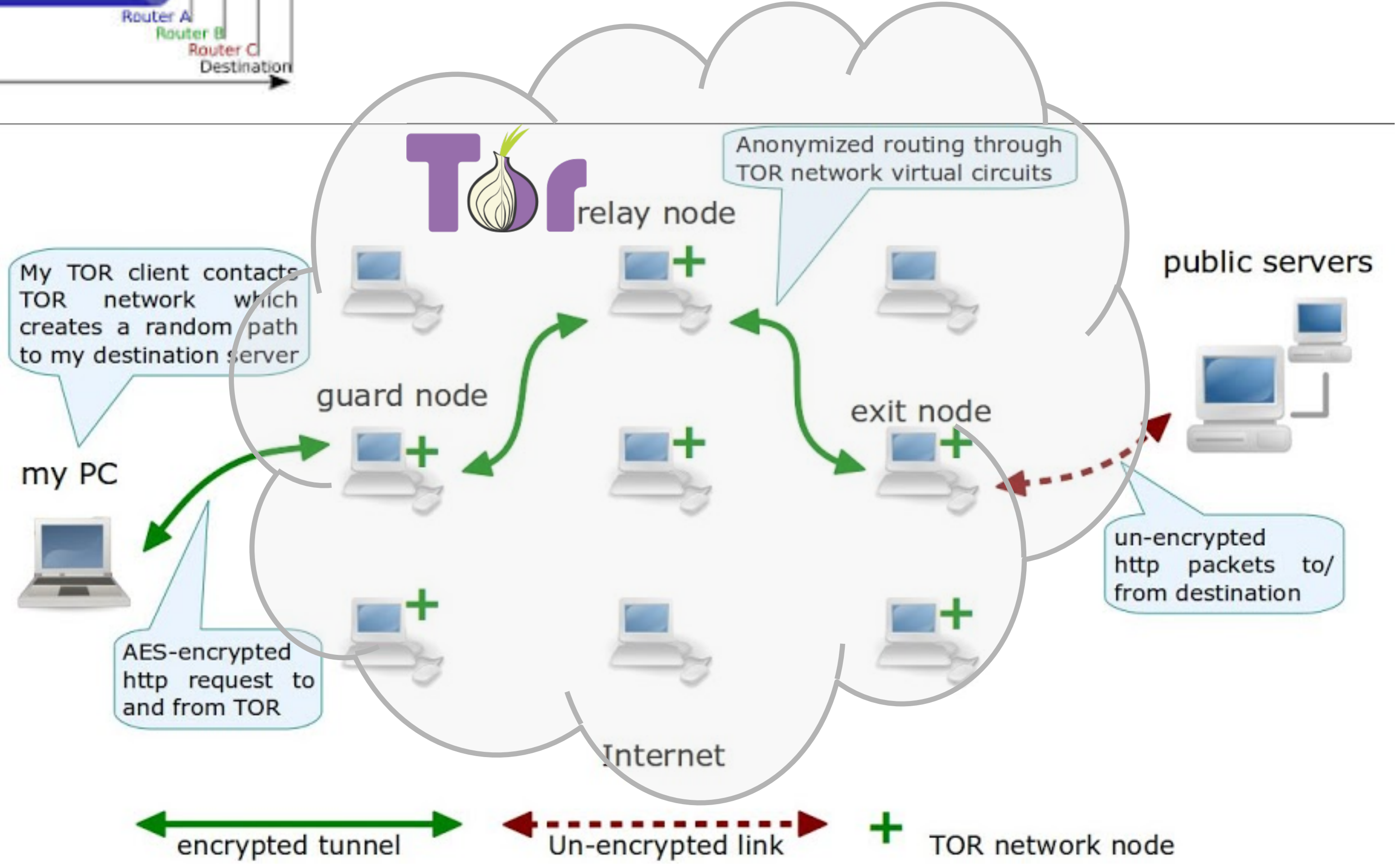
- **... But also as a combined mechanism for other uses (ex):**
  - Data (or Message) Authentication (or Message) Integrity and Proofs (using **CMACs or Cryptographic Authentication Codes**
  - Password-Based Encryption Constructions (**PBEncryption**)
  - Key-wrapping constructions (Keys protected by other keys)
    - Ex., Temporary Keys protected by Master Keys

See aldo in practice on next labs !

  - Enforcement for traffic-flow confidentiality, anonymization or traffic covert-channels, ex., using:
    - Tunneling, Onion-Encryption, Multi-Layered Encryption
    - Onion-Routing for traffic anonymization

Ex., Onion-Routing




relay node
Anonymized routing through TOR network virtual circuits
My TOR client contacts TOR network which creates a random path to my destination server
public servers
guard node
exit node
my PC
un-encrypted http packets to/ from destination
AES-encrypted http request to and from TOR
Internet
encrypted tunnel
Un-encrypted link
TOR network node

Today: More on Symmetric Cyptography

Outline:

- **More on Symmetric Cryptography**
  - More on symmetric cryptography
    - Robustness of symmetric encryption algorithms:
      - Block-Ciphers
      - Stream-Ciphers
    - Cipher block modes of operation
    - Relevance of Padding
    - Randomness and pseudo-randomness
    - Some algorithms:
      - Block Ciphers: DES, 3DES, IDEA, AES
      - Stream Encryption Algorithms and RC4
    - Bit-Stream Ciphers using Block Ciphers

Today: More on Symmetric Cyptography

Outline:

- More on Symmetric Cryptography
  - More on symmetric cryptography
    - Robustness of symmetric encryption algorithms:
      - Block-Ciphers
      - Stream-Ciphers
    - Cipher block modes of operation
    - Relevance of Padding
    - Randomness and pseudo-randomness
    - Some algorithms:
      - Block Ciphers: DES, 3DES, IDEA, AES
      - Stream Encryption Algorithms and RC4
    - Bit-Stream Ciphers using Block Ciphers

Symmetric Encryption Algorithms

Symmetric Encryption (shared key cryptographic method)
Two main families of Symmetric Encryption Algorithms:

- **Symmetric Block Encryption** : Block-Oriented Processing
  - Ex., DES, 3DES, RC5, Blowfish, Skipjack, Katsumi,
    … Camelia, Cha-Cha, Rijaendel, ….
    AES, RC6, Twofish, MARS, Serpent, …
- **Symmetric Stream Encryption**: Stream-Oriented Processing
  - Ex., ARC4, RC4, A5, A8, SEAL, WAKE, SALSA20, CryptMT, …

See available algorithms in crypto provider libraries and their use
See LABs

https://en.wikipedia.org/wikiAdvanced_Encryption_Standard_process
https://en.wikipedia.org/wiki/Symmetric-key_algorithm

Today: More on Symmetric Cyptography

Outline:

- **More on Symmetric Cryptography**
  - More on symmetric cryptography
    - Robustness of symmetric encryption algorithms:
      - Block-Ciphers
      - Stream-Ciphers
    - Cipher block modes of operation
    - Relevance of Padding
    - Randomness and pseudo-randomness
    - Some algorithms:
      - Block Ciphers: DES, 3DES, IDEA, AES
      - Stream Encryption Algorithms and RC4
    - Bit-Stream Ciphers using Block Ciphers

Requirements for security when using symmetric cryptography

- **Robustness of Algorithms** (security and correctness of the symmetric encryption algorithms on their trust-execution criteria)
  - Resistance against brute-force attacks
  - Resistance against cryptanalysis attacks
- **Robust parameterizations for the specific use**

Robust parameterizations

- **Robust parameterizations for the specific use**
  - Need **security association parameters** for the intended purpose
  - **Minimization or Avoidance of key-exposure or other related secrecy parameters (ex. in-memory, in-disks)**
  - Need of **strong keys**: generated w/ randomness, distributed and maintained with security guarantees
    - TRNGs, PRNGs, PRFs for Key-Generation and other parameters
    - Ex., possible use of HSMs, Smartcards, Crypto-Tokens
  - Need of **secure key-distribution and establishment services** (shared keys and related security association parameters)
  - **Fast and secure "rekeying"** services with future and past secrecy
    - Rekeying for temporary session keys
    - Key-Independency: Each generated and established key must be independent of keys generated in the past and keys that will be generated in the future
    - Keys used with OTP (One Time Pad) assumptions

Security of Symmetric Block Ciphers

- **Brute force Attacks**
  - Try all the possible combinations of:
    - Keys
    - Block Sizes
    - Encryption: Plain text Blocks > Found Ciphertext Blocks
    - Decryption: Ciphertext Blocks > Found Plaintext Blocks

- **Cryptanalysis Attacks (or Studies)**
  - Find security vulnerabilities in the internal (core) structure of each algorithm
  - Cryptanalysis types: Cryptanalyst (or Attacker) perspective

	– Ciphertext only	More difficult
	– Known plaintext	↓
IND-CPA	– Chosen plaintext	
IND-CCA	– Chosen ciphertext	
IND CPA&CCA	– Chosen text	Less difficult ? But More relevant ! Why ?

Today: More on Symmetric Cyptography

Outline:

- **More on Symmetric Cryptography**
  - More on symmetric cryptography
    - Robustness of symmetric encryption algorithms:
      - Block-Ciphers
      - Stream-Ciphers
    - Cipher block modes of operation
    - Relevance of Padding
    - Randomness and pseudo-randomness
    - Some algorithms:
      - Block Ciphers: DES, 3DES, IDEA, AES
      - Stream Encryption Algorithms and RC4
    - Bit-Stream Ciphers using Block Ciphers

ABOUT BRUTE FORCE

Brute Force Crypto Attacks

Brute Force Attack (Ataque por força bruta): breaking the key trying every possible keys (knowing algorithm and the key size) to obtain the plaintext:

- Metrics: for N possible keys
  - Success with N/2 keys => break probability = 0.5…
  - N/3 => probability = 0,33; N/10: => probability = 0.1,
  - $N/10^6$ => probability = 10^{-6}
  - $N/2^{256}$ => probability 2^{-256}

- *Rationale* for Brute Force:
  - If there are no inherent mathematical weaknesses and supposing the generation of strong keys and no processing vulnerabilities, a brute force attack is always possible !

- Knowledge of the Algorithm
  - Ex: Block—Based Symmetric Cryptography: Known valid keysizes and blocksizes and assuming the knowledge of the used algorithm

Effort in a Brute Force Crypto Attack

Supposing the quality of (pseudo) random-based key generation (TRGs vs. PRGs), success only dependent from:

- Key size
- Block size
- Processing time to complete (computational cost of the encryption/decryption functions) with the available processing power
  - How many time we need ? What "physics" do we have ?
- Enough Storage to use for the Attack

Ex: effort for BF on a Symmetric-Block-Based Crypto Algorithm:

Keysize = 256 bits

Blocksize = 128 bits

Given a ciphertex Block C:

- We have 2^{128} possible block-candidates
- We have 2^{256} possible keys

# of Cryptographic operations:

$$2^{128}\ 2^{256} = 2^{384}$$

… or (with knonw plaintext)

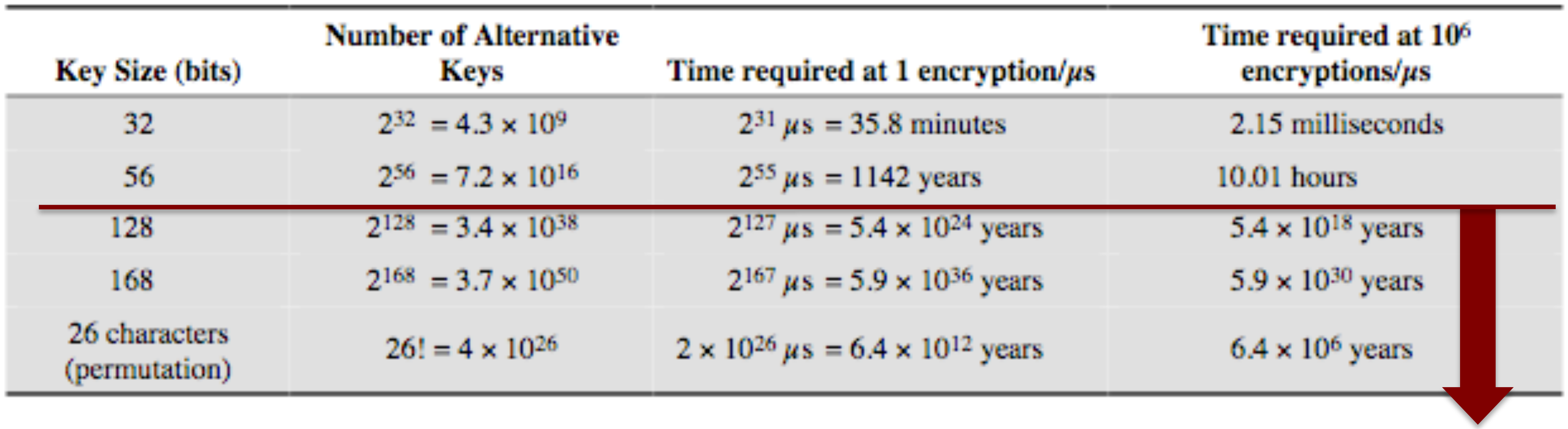
$$2^{256} \sim 1.2 \text{ x } 10^{77}$$

Big numbers [Schneier 1996]

Bruce Schneier, Applied Cryptography, Wiley, 1996

Physical Analogue	Number
Odds of being killed by lightning (per day)	1 in 9 billion (2^{33})
Odds of winning the top prize in a U.S. state lottery	1 in 4,000,000 (2^{22})
Odds of winning the top prize in a U.S. state lottery and being killed by lightning in the same day	1 in 2^{55}
Odds of drowning (in the U.S. per year)	1 in 59,000 (2^{16})
Odds of being killed in an automobile accident(in the U.S. in 1993)	1 in 6100 (2^{13})
Odds of being killed in an automobile accident(in the U.S. per lifetime)	1 in 88 (2^{7})
Time until the next ice age	14,000 (2^{14}) years
Time until the sun goes nova	10^9 (2^{30}) years
Age of the planet	10^9 (2^{30}) years
Age of the Universe	10^{10} (2^{34}) years
Number of atoms in the planet	10^{51} (2^{170})
Number of atoms in the sun	10^{57} (2^{190})
Number of atoms in the galaxy	10^{67} (2^{223})
Number of atoms in the Universe (dark matter excluded)	10^{77} (2^{265})
Volume of the Universe	10^{84} (2^{280}) cm^3
If the Universe is Closed:	
Total lifetime of the Universe	10^{11} (2^{37}) years 10^{18} (2^{61}) seconds
If the Universe is Open:	
Time until low-mass stars cool off	10^{14} (2^{47}) years
Time until planets detach from stars	10^{15} (2^{50}) years
Time until stars detach from galaxies	10^{19} (2^{64}) years
Time until orbits decay by gravitational radiation	10^{20} (2^{67}) years
Time until black holes decay by the Hawking process	10^{64} (2^{213}) years
Time until all matter is liquid at zero temperature	10^{65} (2^{216}) years
Time until all matter decays to iron	$10^{10\text{^}26}$ years
Time until all matter collapses to black holes	$10^{10\text{^}76}$ years

Exhaustive brute-force key-search



Key Size (bits)	Number of Alternative Keys	Time required at 1 encryption/μs	Time required at 10^6 encryptions/μs
32	$2^{32} = 4.3 \times 10^9$	$2^{31}\ \mu s = 35.8$ minutes	2.15 milliseconds
56	$2^{56} = 7.2 \times 10^{16}$	$2^{55}\ \mu s = 1142$ years	10.01 hours
128	$2^{128} = 3.4 \times 10^{38}$	$2^{127}\ \mu s = 5.4 \times 10^{24}$ years	5.4×10^{18} years
168	$2^{168} = 3.7 \times 10^{50}$	$2^{167}\ \mu s = 5.9 \times 10^{36}$ years	5.9×10^{30} years
26 characters (permutation)	$26! = 4 \times 10^{26}$	$2 \times 10^{26}\ \mu s = 6.4 \times 10^{12}$ years	6.4×10^6 years

Important factors:

- Expectable computational effort and cost
  - Try the available computation for a symmetric crypto alg. with a benchmark tool
    - Ex., openssl speed aes for AES Algorithm
- Key sizes (and also the block sizes)
- Degree of "plaintext intelligibility"
  - More difficult in some cases: ex., binary data, pre-processed plain txt transformation, compression, hashed-data, hashed data with mixed "secrets", etc…

Exhaustive brute-force key-search

Key Size (bits)	Number of Alternative Keys	Time req. for 10^9 decryptions/us	Time required at 10^{13} decryptions/us
32	$2^{32} = 4.3 \times 10^9$	2^{31} ns , 3 minutes	~2 us
56	$2^{56} = 7.2 \times 10^{16}$	2^{55} ns , 1.125 years	1 hour
128	$2^{128} = 3.4 \times 10^{38}$	2^{127} ns , 5.3×10^{21} years	5.3×10^{17} years
168	$2^{168} = 3.7 \times 10^{50}$	2^{167} ns , 5.8×10^{33} years	5.8×10^{29} years
26 characters (permutation)	$26! = 4 \times 10^{26}$	2×10^{26} ns , 6.3×10^9 years	6.3×10^6 years
192	$2^{192} = 6,3 \times 10^{37}$	2^{191} ns , 9.8×10^{40} years	9.8×10^{36} years
256	$2^{256} = 1,2 \times 10^{77}$	2^{255} ns , 1.8×10^{60} years	1.8×10^{56} years

- What of we improve the computational (and energy ;-)) capacity:

  Ex., 10^9 /us or … 10^{13} /us

Uhm … How many Encryptions/Decryptions can you Do in your computer ? (Demo in Class)

Optimistic? Something wrong in these parallel "hazing ritual" projections ?

More brute-force attack estimations

Ex., Breaking with $10 Millions in 2000:

Symmetric Crypto ⟷ Asymmetric Crypto ⟷

Symmetric Alg. (key sizes)	Public-Key (ECC)	Public Key (RSA)	*Time to Break*	Storage needs
56	**112**	**420**	**< 5 min**	**Trivial**
80	**160**	**760**	**600 months**	**4GB**
96	**192**	**1020**	**3x10^6 years**	**170GB**
128	**256**	**1620**	**10^16 years**	**120TB**

*Robert Silverman, Technical Report RSA Laboratories, **2000***

- Time-to-Break, with computational power of 10^6 decryptions/μs
- EFF announced in 1998: "DES breaking experience in 3 days with USD$250,000"

… Moore's Law and other predictions

http://en.wikipedia.org/wiki/Moore's_law

- Processing power doubles, each 18 months (will continue ?)
- In 10 years, computers will be 100 times more powerful (in computational resources).
  - A desktop fits now into a smart phone … next in a IoT device ?
  - Storage Capacity
  - Gigabit wireless connectivity everywhere
  - Global computation power: Networks connect our computing devices in the context of collective/global/shareable computing environments and datacenters …
- Other aspects of the future (more difficult to predict)
  - Anyone can not predict what the emergent properties of "100 or 1000x computing power" will bring: new computing physics, and new paradigms of communication.
  - A "100x-1000x … world" will be different, in ways that will be surprising, with possible disruptions (we don't know yet)

More brute-force attack estimations

Ex., Breaking with $10 Millions in 2000:

Symmetric Crypto ⟷ Asymmetric Crypto

Symmetric Alg. (key sizes)	Public-Key (ECC)	Public Key (RSA)	*Time to Break*	Storage needs
56	112	420	< 5 min	Trivial
80	160	760	600 months	4GB
96	192	1020	3x10^6 years	170GB
128	256	1620	10^16 years	120TB

Robert Silverman, Technical Report RSA Laboratories, 2000

Moore Law (computing power):

- double in each 18 months ....

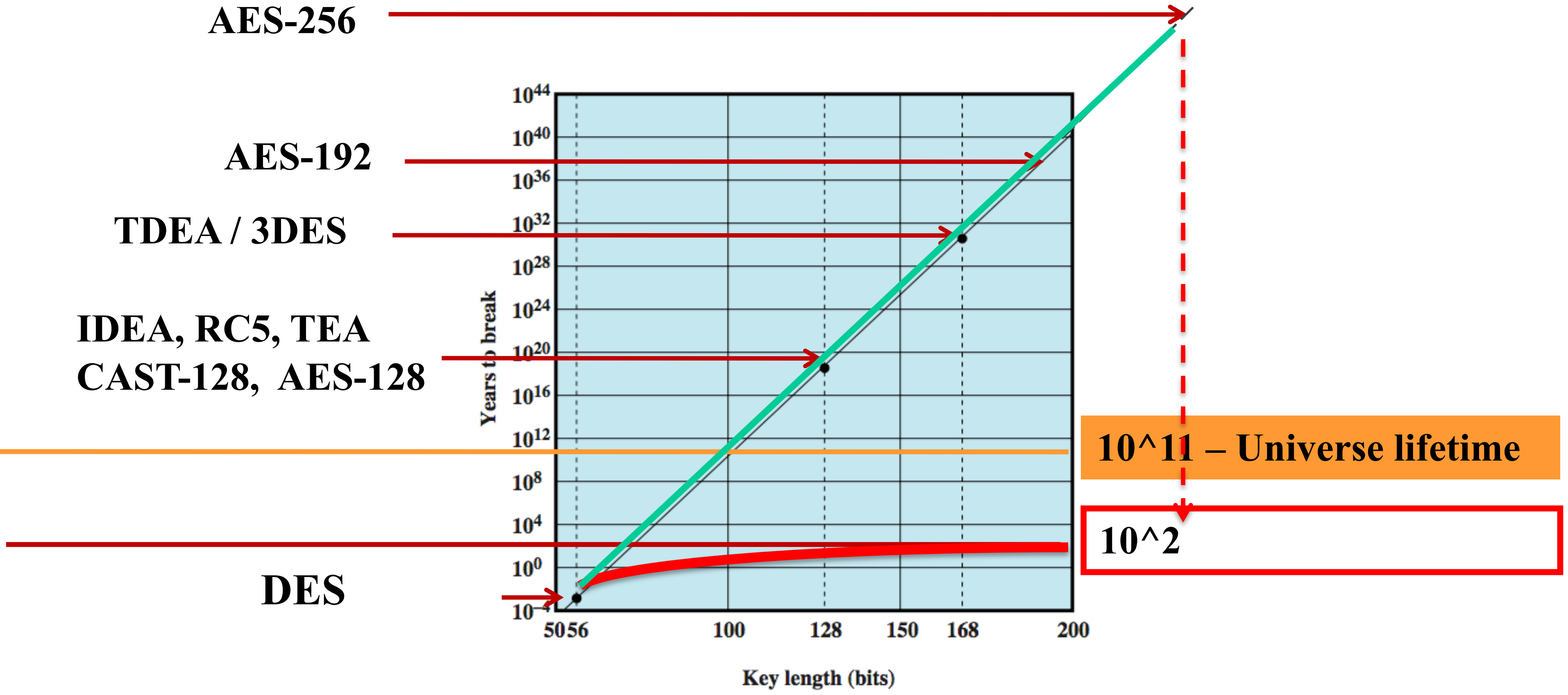
Cost estimation to break

Extrapolation using the Moore law ↓

Key size		Year (attack)
80	———	2010
96	———	2034
112	———	2058
128	———	2082
144	———	2106
160	———	2130
176	———	2154
192	———	2178
208	———	2202
224	———	2226
240	———	2250
256	———	2274

Average time required for a brute-force attack

Examples: DES, IDEA, RC5, TEA, CAST-128, AES-128, AES-192, AES-256



Today: More on Symmetric Cyptography

Outline:

- **More on Symmetric Cryptography**
  - More on symmetric cryptography
    - Robustness of symmetric encryption algorithms:
      - Block-Ciphers
      - Stream-Ciphers
    - Cipher block modes of operation
    - Relevance of Padding
    - Randomness and pseudo-randomness
    - Some algorithms:
      - Block Ciphers: DES, 3DES, IDEA, AES
      - Stream Encryption Algorithms and RC4
    - Bit-Stream Ciphers using Block Ciphers

ABOUT CRYPTANALYSIS

Two important considerations: Unconditionally vs. Computationally Security

- Unconditionally secure: generated cipher text does not contain enough information to determine uniquely the correspondent plaintext
  - Time vs. Space Independent Notion
  - Only theoretically achieved by OTP schemes
- Computationally security: security with two different evaluation criteria:
  - **Breaking cost** > value of encrypted information
    - Cost metrics: money, required resources
  - **Breaking time**: time required to break vs. useful lifetime of the encrypted information
    - Can we approach with "practical infinite time considerations" ? How ?
  - **Physics to break** (or possible physics impossibilities)
    - Can we have the required matter (ex., atoms) in the universe ?

Robustness and computational security criteria in the study of cryptography

- **Robustness:** if it is impossible (**computationally impossible**) to decrypt a message on the basis of the ciphertext plus the knowledge of the encryption/decryption algorithm
- **Computational security criteria:**
  - **Infinite time to break with available computing power and resources (processing power MIPS, memory, storage,…)**
    - what is "infinite time" ?
      - Ex., More than the universe lifetime … 10^11 years ?
      - Ex., More than the universe time in several orders of magnitude ?
      - **Finite time to break** … but time exceeds the useful lifetime of the protected information
  - **Finite time to break but requires infinite or impossible computing power and information resources**
    - what is "impossible computing power" ?
      - Ex., A quantity of computers requiring more silicon or germanium atoms than the total quantity of atoms (estimated) available in our galaxy ?
      - Ex., Processing speed required will cause the fusion of the materials ?
      - **Available computing can break before the useful lifetime of the protected information** … but the cost (money) of such computing resources exceeds the value of the protected information

Security of Symmetric Block Ciphers: What about Cryptanalysis

- **Cryptanalysis Attacks (or Studies)**
  - Find security vulnerabilities in the internal (core) structure of each algorithm
  - Cryptanalysis types: Cryptanalyst (or Attacker) perspective

	– Ciphertext only	More difficult
	– Known plaintext	↓
IND-CPA	– Chosen plaintext	
IND-CCA	– Chosen cyphertext	
IND CPA&CCA	– Chosen text	Less difficult ? But More relevant ! Why ?

See definitions for CPA, IND-CPA and CCA, IND-CCA

Shannon Criteria for Quality of Crypto Systems (1949, C. Channon) 1

- Quantity of secrecy
  - Required effort: time required resources (including $$) to conduct cryptanalsys attacks
    - If time > useful time of the secret information, than the objective is achieved
- Key Sizes
  - Complexity for the transmission/distribution and storage, in the involved entities
    - Ex., this means that Theoretical Perfect OTPs are bed for practical use
- Block Size
  - Cyphertext block sizes must be adequate considering Plaintext blocksizes (ideally meaning smaller or equal, with no cyphertext expansions
  - And blocksizes also have effects on the security assumptions

Shannon Criteria for Quality of Crypto Systems (1949, C. Channon) 2

- Simplicity of realization and exploitation
  - Regarded today in terms of possibly different realizations: HW, SW, tamperproof devices, etc as well as required Key-Generation Processes and Means

- Error propagation and recoverability
  - Transmission errors (ex., bit-changes) in transmitted cyphertext
  - But detection of errors w/ no recoverability in a chain of cyphertext blocks can be a good thing for integrity-detection purposes

Important notions for Robustness w/ Cryptanalysis Attacks (or Studies)

- Without knowing the Key, algorithms must perform as Random Oracles
- Shannon statements:
  - Confusion (entropy and complexity)
  - Diffusion
- Repercussions / Cryptanalysis assessment tests:
  - Degree of the Avalanche-Effect
    - Diffusion in using keys and plaintext blocks
  - Kasiski inspired tests for the evaluation of "periods"
  - Coincidence Indexes (or Displacement Tests)
  - Algebraic (Math. Proofs)
    - Decryption not knowing the key => Coplexity not bounded by polynomial time complexity or in a perfect construction, a NP Complete problem

Specific Cryptanalysis

- Must consider the Internal Structure of each particular algorithm
  - Three Big Families of CORE Internal Processing Structures:
    - **No-algebraic constructions**: complex compositions of substitutions, permutations or expansions/contractions rounds, parameterized by subkeys derived from the application of the same techniques to the input key
    - **Algebraic constructions in GF2**: Rounds of modular arithmetics, computations involving polynomials, and also the use of Boolean operators, matrix-multiplications (and/or matrix columns/rows shifts and permutations)
    - **Hybrid (all of above)**

Internal structure of symmetric block ciphers

Also characterized by:

- **Block size** (larger means more security)
- **Key size** (larger means more security)
- Possible initial block transformation (initial pre-processing of input block)
- **CORE Round transformation function** (greater complexity for diffusion and confusion means more resistance)
- Sub-key generation using the initial key as seed, used to parameterize the transformation in each round
- **A number of core and sub-key generation rounds** (multiple rounds increase security by improving diffusion and confusion)
- Possible final block transformation (post-processed block) to obtain the output ciphertext block)

Block Ciphers and Internal Structure

- A generic structure for block-encryption algorithms: a sequence of block transformation rounds (w/ the core algorithm) performing transformations, successively parameterized by secret subkeys derived from the initial key
  - Sub-keys: derived for each new round from a key-scheduling algorithm
  - Sub-key Generation (scheduling) Funtion
  - Core Encryption and Decryption Functions
    - Transformations in core rounds and related input sub-keys
  - Number of rounds to materialize the effect of a "random-oracle" (or entropy) if the key is not known
- Characteristics of particular algorithms:
  - Base block size
  - Valid key size (or key sizes)
- For practical use (characteristics):
  - Fast encryption/decryption (performance in computation time)
  - SW and HW implementation possibility
  - Core round and sub-key generation algorithm (ease of analysis for cryptanalysis)
  - Complexity analysis and soundness proofs

Characteristics in symmetric block cyphers

Security

- Security assessment criteria (brute-force and cryptanalysis)
- Confusion and Diffusion Criteria or Avalanche Effect
  - No distinguishability from "random" processing (not knowing the key)
  - Soundness of statistical studies and/or mathematical (arithmetical) proofs
- Requires complexity (complex transformations in each round): large number of rounds, large keysize, large blocks will be better

Good tradeoff on security vs. fast operation (performance)

- Speed of the algorithm is a concern
- Simplicity (ex., HW implementations and embedded solutions, fast computation, low computational cost, energy-savings, …

Analysis and Verification: ease for cryptanalysis and implementation verification

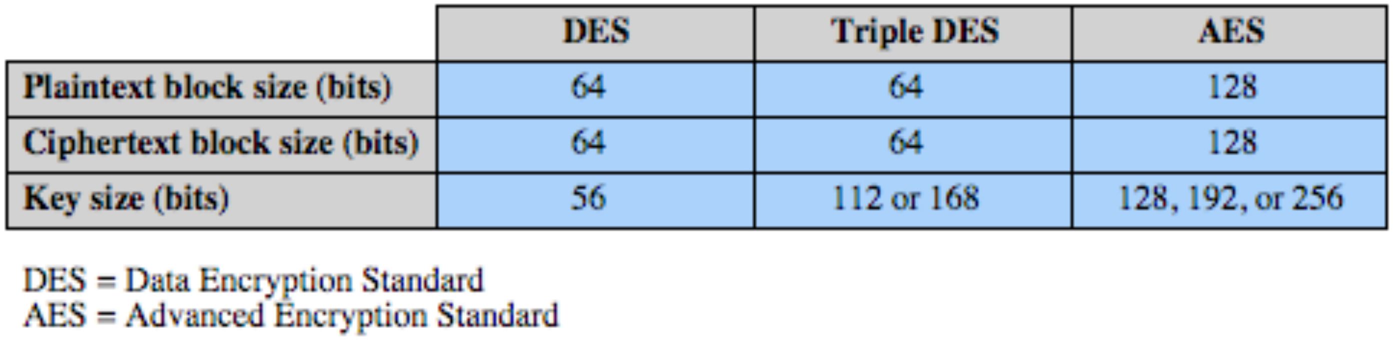
- Simplicity in the internal reference structure
- "analyzable": formalisms and mathematical foundations, as well as, impplementation correction

See for example the NIST requirements for the AES competition: ex., https://competitions.cr.yp.to/aes.html

Some Symmetric Algorithms and Structure:
The past and the emergence of strong algebraic constructions

Alg.	Key size Block size	Rounds	Structure Operations	Ex. Applications (standardization)
DES	56 /64	16	XOR, S-boxes, P-Boxes EP Boxes, Shifts/Rotations, ...	SET, Kerberos
3DES	112,168 /64	48	idem	Financial standards E-commerce, PGP S/MIME, SSL
IDEA	128 /128	8	XOR, addition mod 2^16 Multiplication mod 2^16	PGP, SSL
TEA	128 /128	32	XOR + shift	
Blowfish	< 448 /128	16	XOR + Variable S-boxes	
RC5	< 2048 /128	< 255	Addition, subtraction XOR and rotation operations	SSL
Cast-128	40,128 /128	16	Addition, Subtraction, XOR, Rotation and S-boxes	PGP
AES	128, 192, 256 /128	10,12,14	S boxes, Mod. Arith., Matrix SR-MC, XOR	Many: "The Standard" after 2000

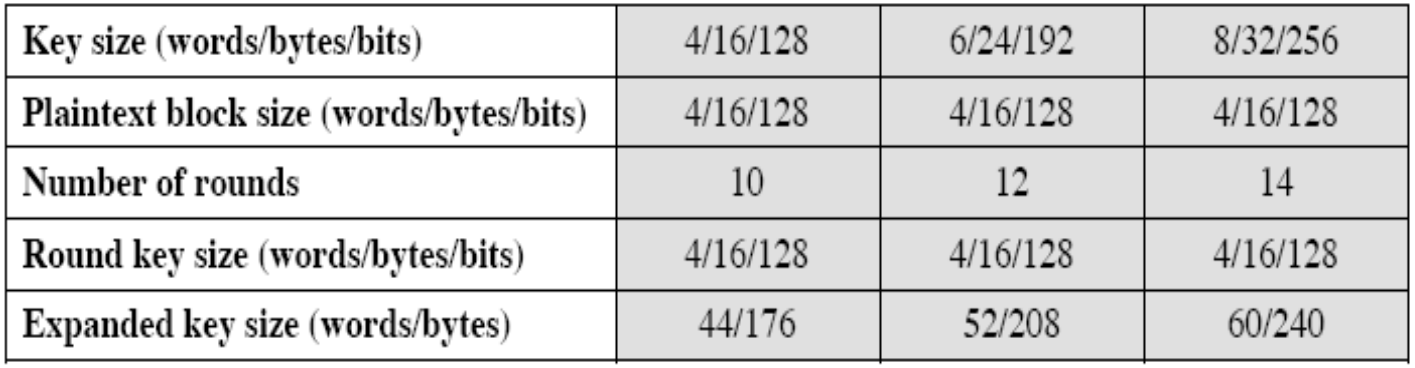
Examples: AES vs. DES vs. 3DES



	DES	Triple DES	AES
Plaintext block size (bits)	64	64	128
Ciphertext block size (bits)	64	64	128
Key size (bits)	56	112 or 168	128, 192, or 256

DES = Data Encryption Standard
AES = Advanced Encryption Standard

Figures in AES:

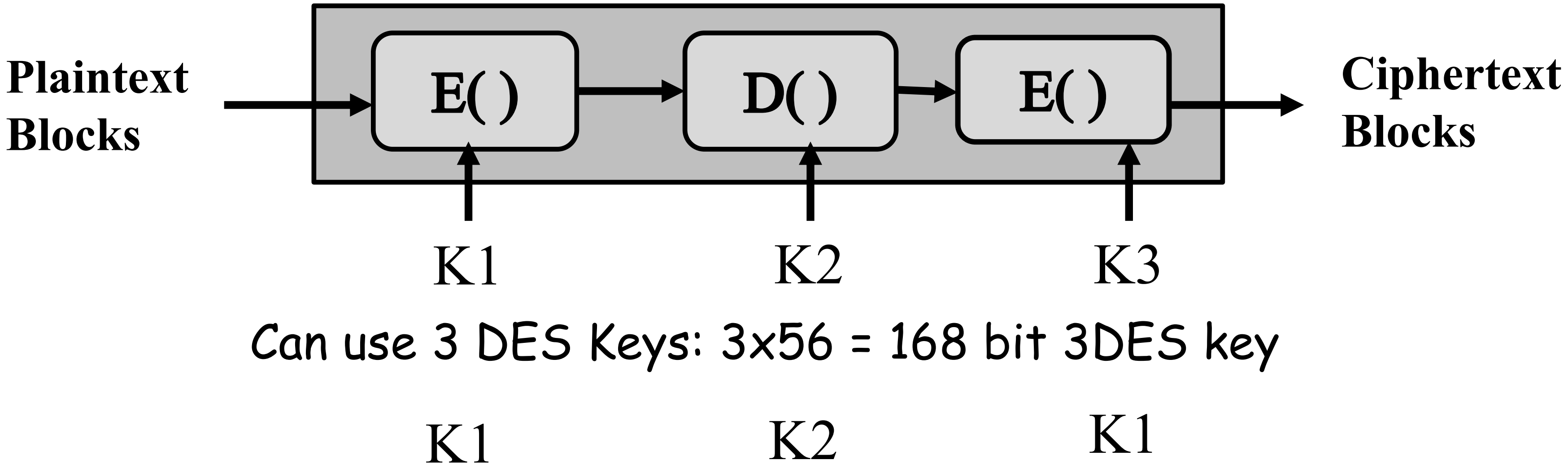


Key size (words/bytes/bits)	4/16/128	6/24/192	8/32/256
Plaintext block size (words/bytes/bits)	4/16/128	4/16/128	4/16/128
Number of rounds	10	12	14
Round key size (words/bytes/bits)	4/16/128	4/16/128	4/16/128
Expanded key size (words/bytes)	44/176	52/208	60/240

Demo: Performance Comparisons in Class
(Ex., `$openssl speed aes des des-ede3`)
See other valid blockizes and keysizes for other symmetric algorithms (LABs)

Triple DES (also called DESede)

- An example of Onion-Encryption/Decryption Technique, in this case using the same Base Algorithm: DES



Can use 3 DES Keys: 3x56 = 168 bit 3DES key

or can use 2 DES Keys: 2x56 = 112 bit 3DES key

- Interesting issues:
  - Why Triple DES ? Why E-D-E Processing ?
  - Can we use Onion-based Multiple Algorithms with Multiple Keys ?
    Is it a good idea ?

Are Symmetric Cryptographic Algorithms vulnerable for Quantum-Computing ?

- **Post-quantum cryptography** (also ref. as **quantum-proof**, **quantum-safe** or **quantum-resistant**) refers to crypto algorithms that are thought to be secure against an attack by a Quantum Computer
  - As of 2020, this is not true for the most popular public-key algorithms: can efficiently broken by a sufficiently strong quantum computer.

- However in contrast to the threat quantum computing poses to current public-key algorithms, most current strong symmetric cryptographic algorithms from the most recent generation (ex., AES, RC6, Twofish, MARS, with 256 bit keys or even with OnionEncryption constructions) and also secure hash functions (ex., SHA2-256/512 or SHA3-256/512) are considered to be relatively secure in practice to be quantum resistant.

- Why, How and in What Sense ? Discussion !

Today: More on Symmetric Cyptography

Outline:

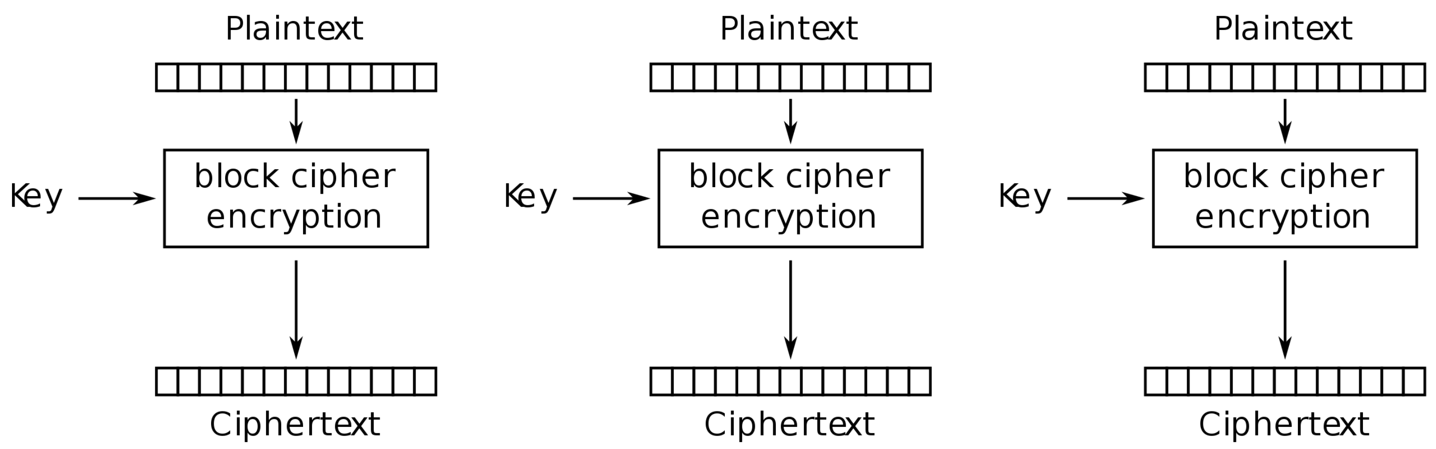
- **More on Symmetric Cryptography**
  - More on symmetric cryptography
    - Robustness of symmetric encryption algorithms:
      - Block-Ciphers
      - Stream-Ciphers
    - Cipher block modes of operation
    - Relevance of Padding
    - Randomness and pseudo-randomness
    - Some algorithms:
      - Block Ciphers: DES, 3DES, IDEA, AES
      - Stream Encryption Algorithms and RC4
    - Bit-Stream Ciphers using Block Ciphers

Modes of Operation (Symmetric Block Ciphers)

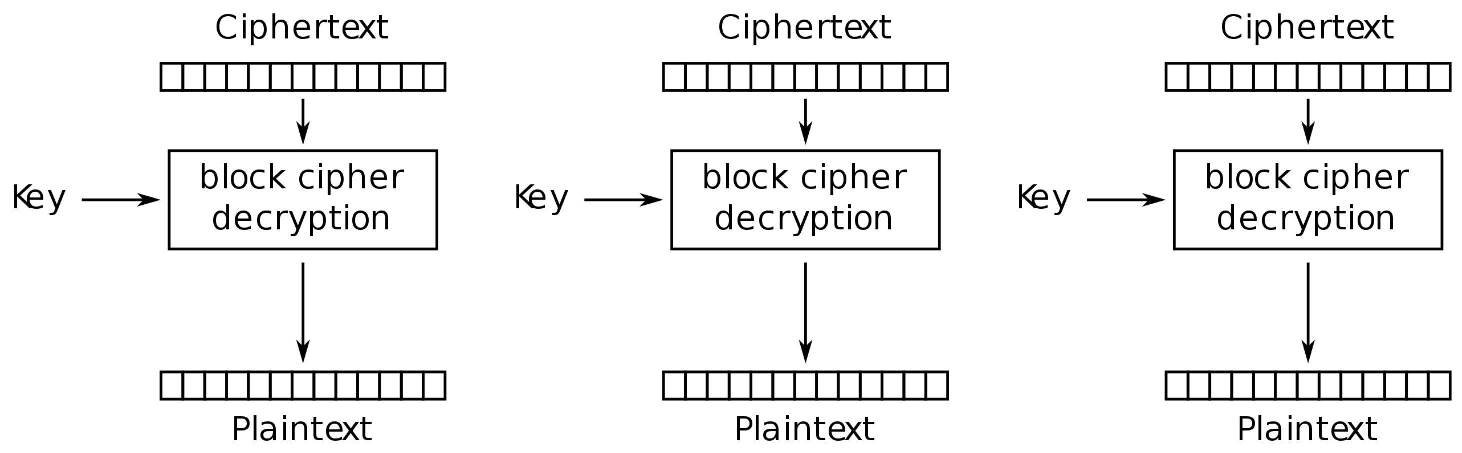
- Remember: Block ciphers encrypt fixed size blocks
- Need some way to encrypt/decrypt arbitrary amounts of data in practice (in a secure way and for the proper use)
- **ANSI X3.106-1983 Modes of Use** (now FIPS 81) defines 4 possible modes: **ECB, CBC, PCBC, CFB, OFB**
- Subsequently 5 more defined for AES & DES: **CTR, CTS**
- Also applicable to any Block-based Symmetric Crypto Algorithm
- Today other modes are also relevant:
  - XTS (NIST SP 800-3E Specification)
  - CCM, GCM, OCB, EAX, ….. CWC, IAPM
- Usable with any Symmetric Encryption Alg. (Block Ciphers or Block-Oriented Algorithms)

Red-Indications: available in JAVA/JCE (ex., BC)
GCM, CCM, OCB used with No Padding
CCM requires "valid" IVs: ex., 7 to 13 bytes, OCB no more than 15 bytes

ECB Mode (Electronic Code Book)



Electronic Codebook (ECB) mode encryption



Electronic Codebook (ECB) mode decryption

ECB Mode Characteristics

- Plaintext blocks encrypted in an independent weay
- Fast (encryption and decryption are parallelizable)
- Random Read Access (block-indexing)

Message Repetitions => Regularities show in ciphertext

- If aligned with message block
- Particularly with data such graphics
- or text-messages or messages that change very little, which become a code-book analysis problem

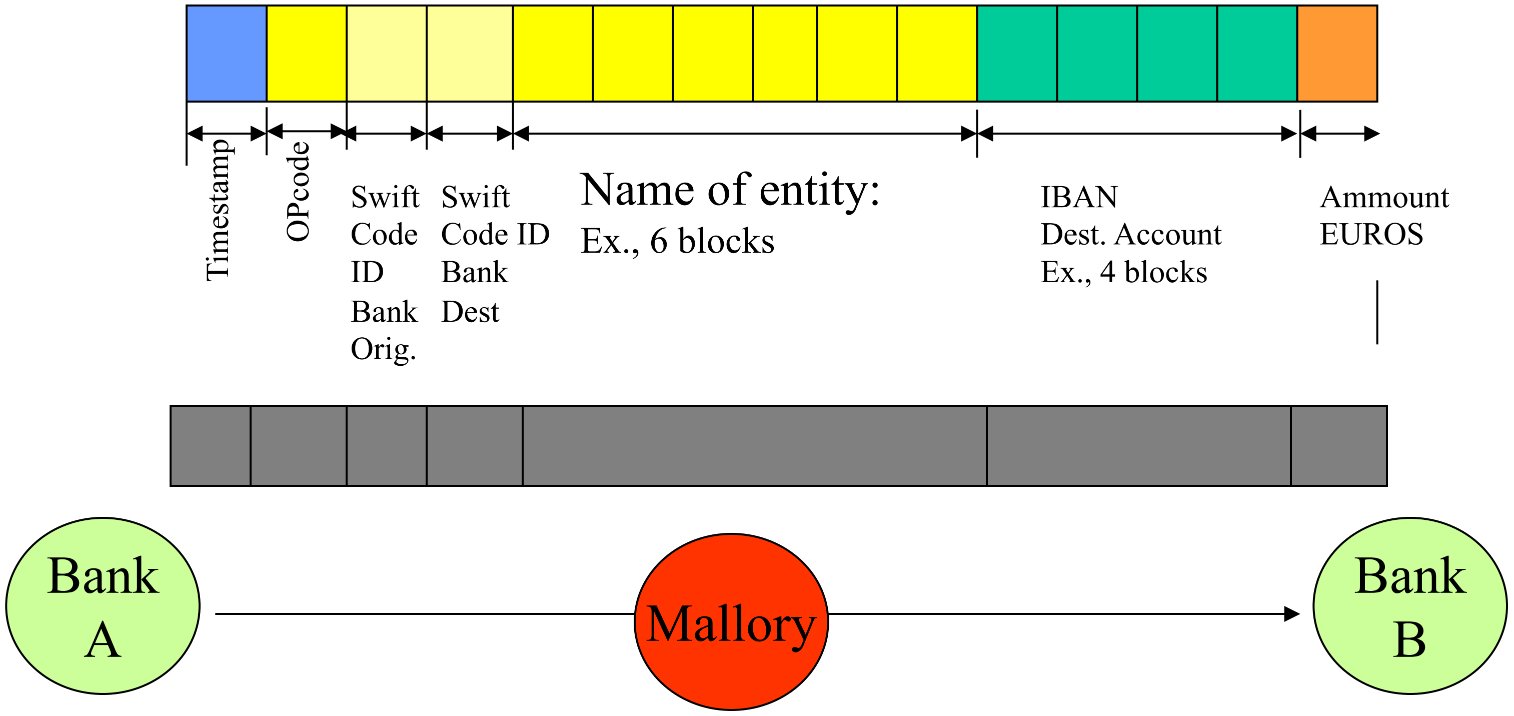
- Weakness is due to the encrypted message blocks being independent: same plaintext blocks produces the same cyphertext blocks (using the same key)

- Main use is sending a few blocks of data, encryption of small blocks (size small than the block size of a given algorithm, small-byte blocks, ex., passwords)

ECB: manipulation of encrypted blocks

Ex., electronic banking transaction, ref. with DES

1 block = 64 bits

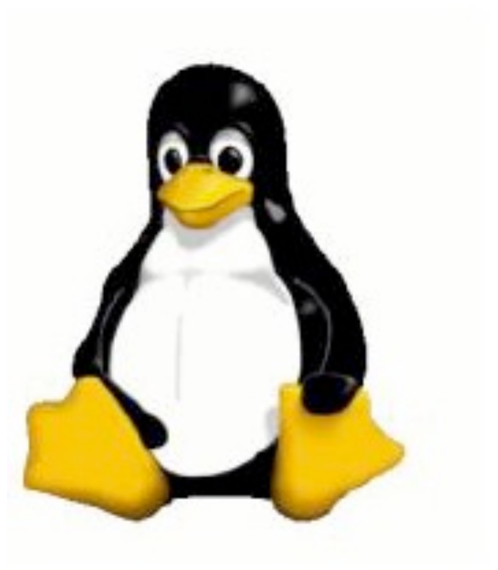


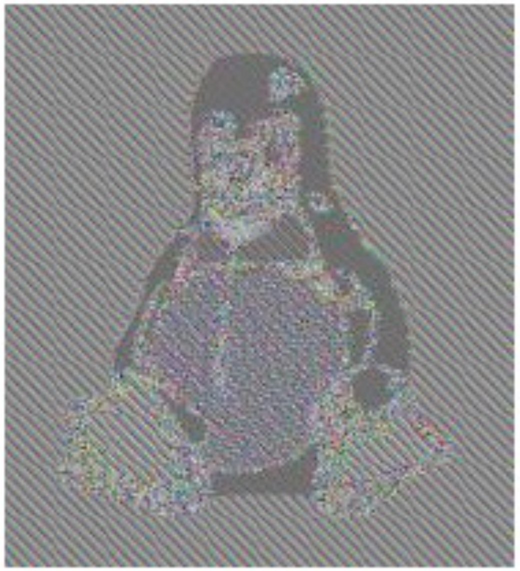
Eavesdropping + Message Replying + Message Tampering (on specific blocks)

ECB (graphics, revelation of patterns)

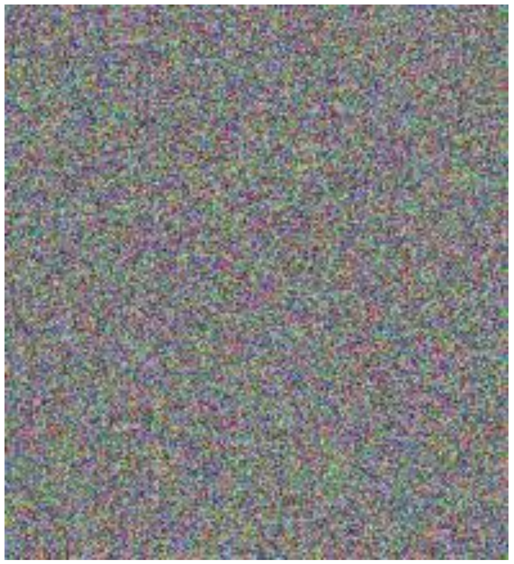
Plaintext and Ciphertex with ECB

What we want ...





Original de Larry Ewing's



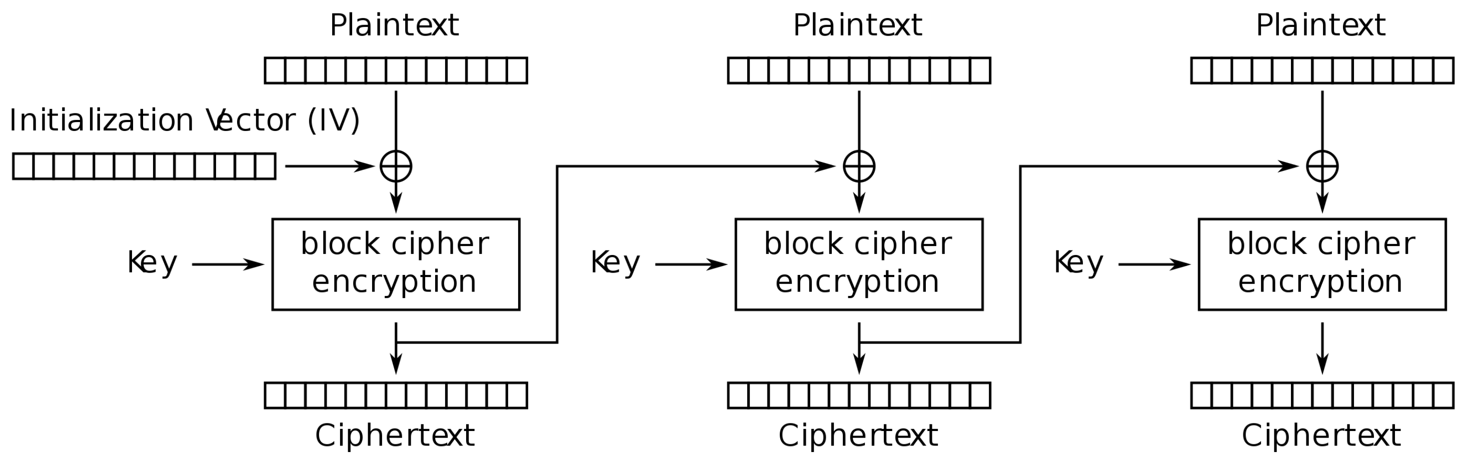
Demonstrated in Labs,
for bmp/gif/jpeg files

ECB Advantages and Drawbacks

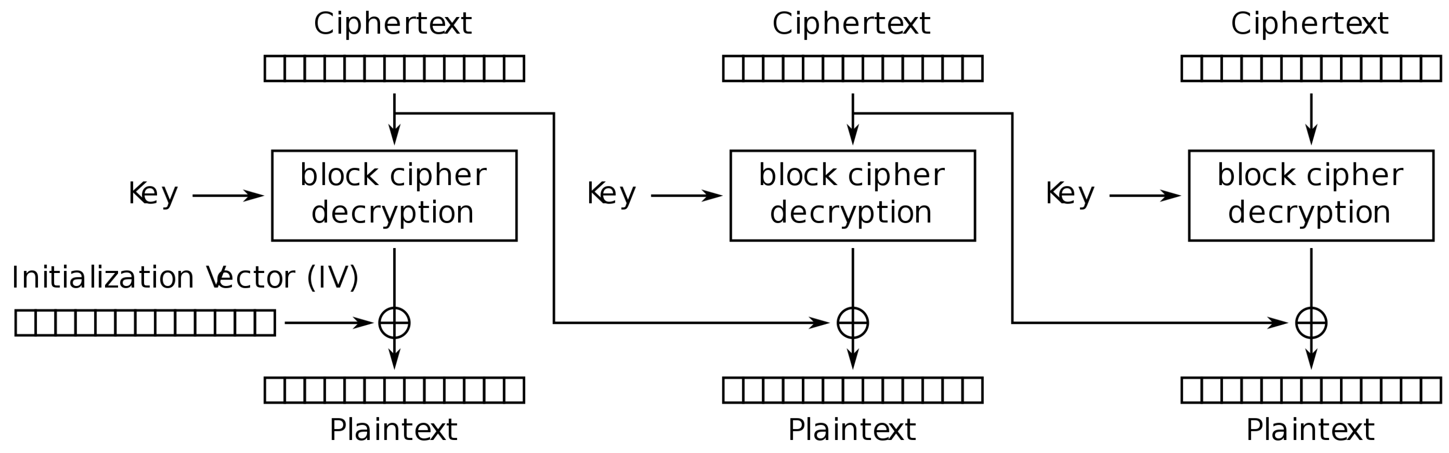
- Simple, Block-By-Block Independent Encryption
- Fast – Easily Parallelizable
  - The faster mode: faster than any other mode
- Easy to recover possible errors: simply retransmit the affected block
  - No Error Propagation

- Same plaintext blocks => Same ciphertext blocks if we use the same key
  - Regularities of plaintext blocks in a sequence of blocks are revealed
- But can be used when not-revealing block-patterns
  - If plaintext is first pre-processed in some way to hide possible regular patterns
  - Example: Compression before Encryption is usual
  - Or if we encrypt data with size lower than the block size (in a given algorithm)
    - Example: Encryption of Initialization Vectors, Counters, Variable Padded Passwords, Nonces/Responses, etc

CBC Mode: Cipher Block Chain



Cipher Block Chaining (CBC) mode encryption



Cipher Block Chaining (CBC) mode decryption

CBC Mode

- Cipher Block Chaining Mode (CBC)
  - The input to the encryption algorithm is the XOR of the current plaintext block and the preceding ciphertext block.
  - Repeating pattern of 64-bits are not exposed

- Need an Initial Vector (IV) to start process

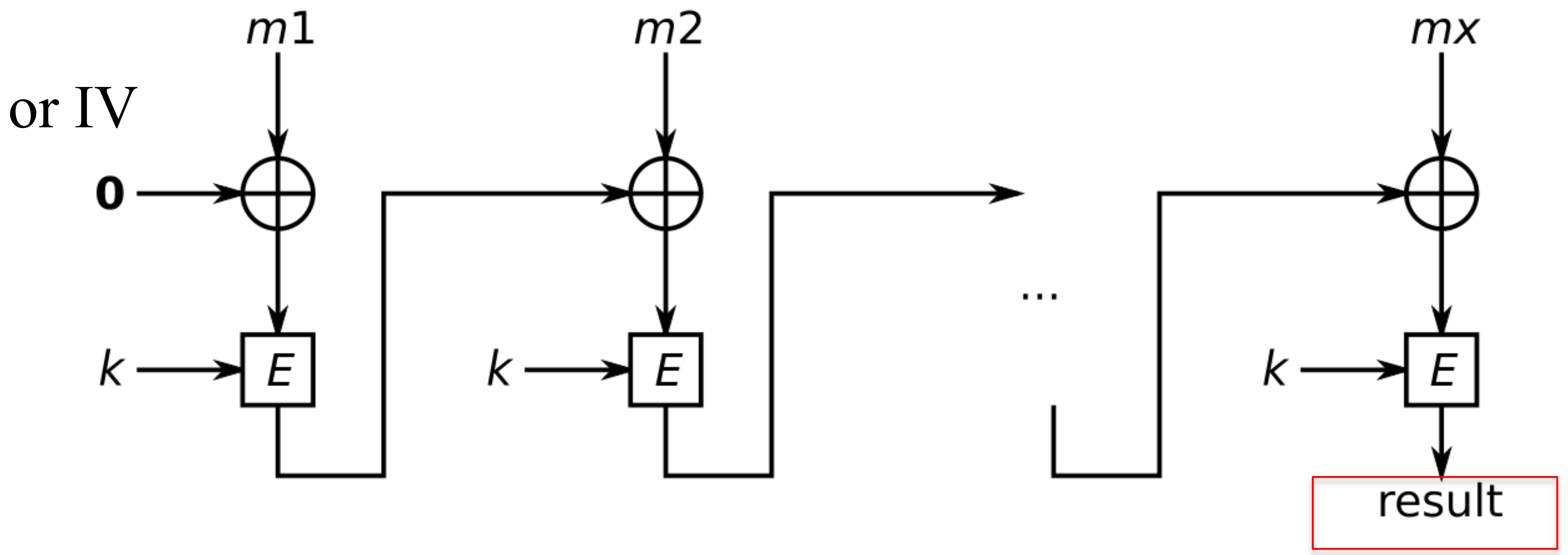
  $C_i = DES_{K1}(P_i \text{ XOR } C_{i-1})$

  $C_{-1} = IV$

- Uses: for bulk data encryption for confidentiality, but also for authentication (example CBC-MAC shemes

- Same plaintext blocks => Different cyphertext blocks, even when the same key is used (if use different IVs)

CBC-MAC Scheme (usable as a CMAC Scheme



CBC-Obtained
Message Authentication Code

Suppose you send …

E_{Km} (M) || CBC-MAC_{KA}

CBC Mode Characteristics (1)

Security

- + Security: plaintext patterns concealed by XORing with previous cyphertext blocks – Input to blocks randomized by the previous cipher text block
- + Any change to a block affects all following ciphertext blocks
- +/- Plaintext somewhat difficult to manipulate
  Only 1st block and last block can be removed)
  Bits changed in the first block: repetition allows controlled changes
- + More than one message can be encrypted with the same key
- + A cyphertext block depends on **all** blocks before it

Performance

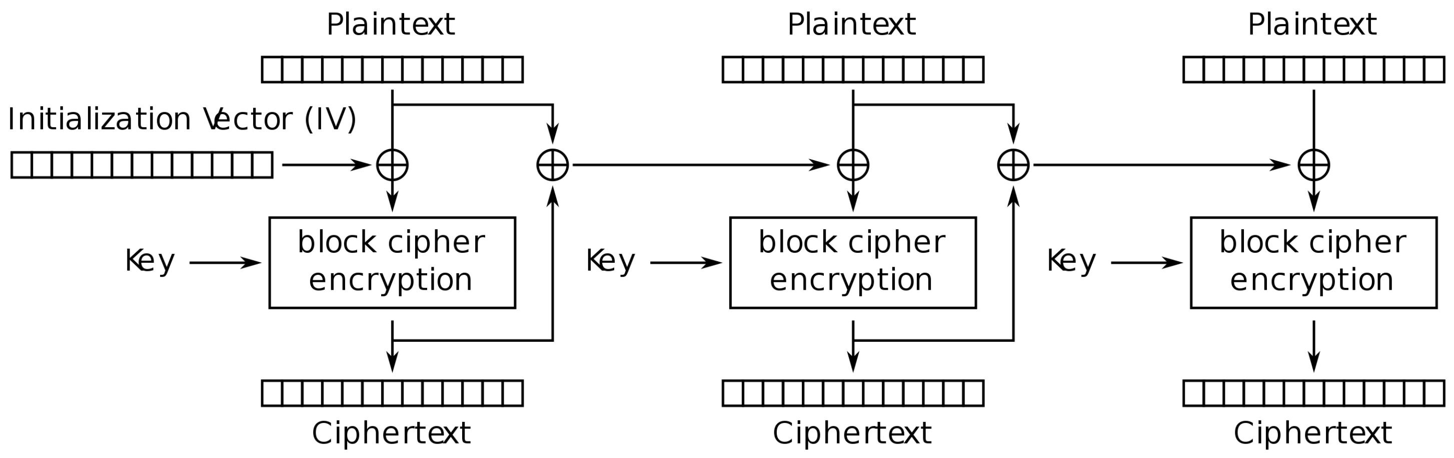
- + Speed in ~the same as block cipher
- - But ciphertext is up to one block longer than the plaintext
- - No pre-processing is possible
- -/+ Encryption is not parallelizable, but decryption is parallelizable and has a random-access property

CBC Mode Characteristics (2)

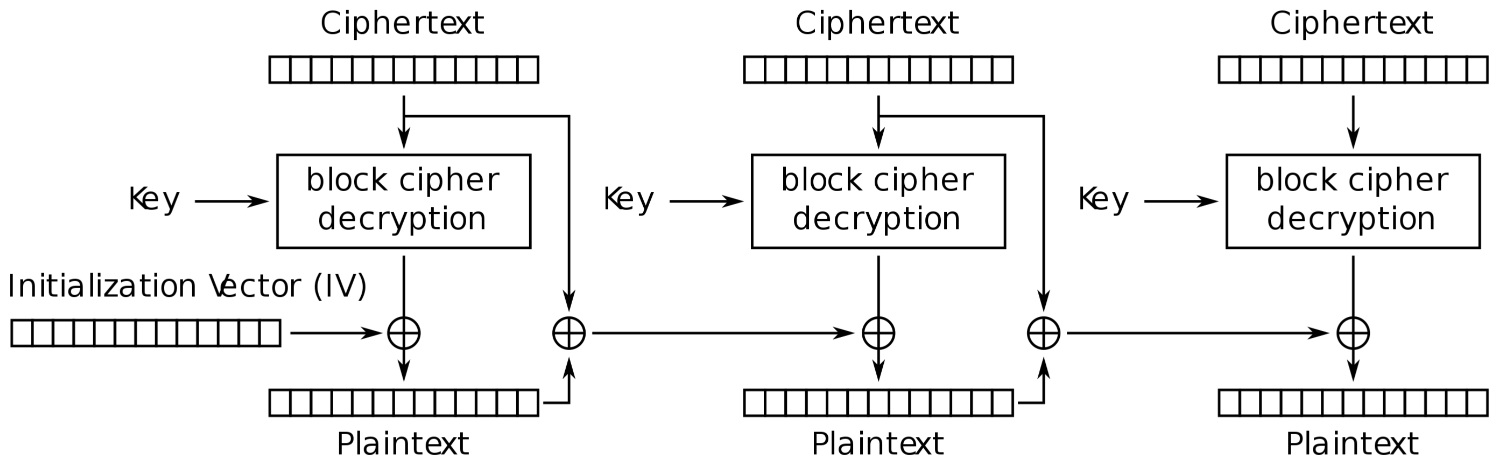
Fault-Tolerance

- - Need **Initialization Vector** (IV)
  - which must be known to sender & receiver
    - Requires synchronization
  - if sent in clear, attacker can change bits of first block, and change IV to compensate
  - hence IV must either be a fixed value (as in EFTPOS)
  - or must be sent encrypted in ECB mode before rest of message
- A cipher text error affects one full plaintext block and the correspondent bit in the next block
- Synchronization errors unrecoverable

PCBC Mode (Propagating CBC mode)



Propagating Cipher Block Chaining (PCBC) mode encryption



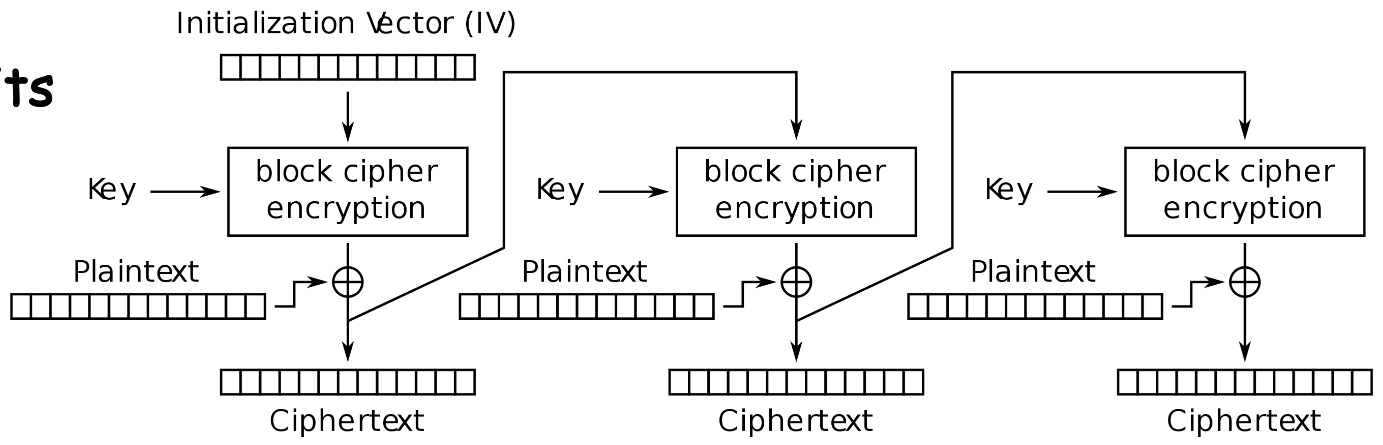
Propagating Cipher Block Chaining (PCBC) mode decryption

PCBC Use

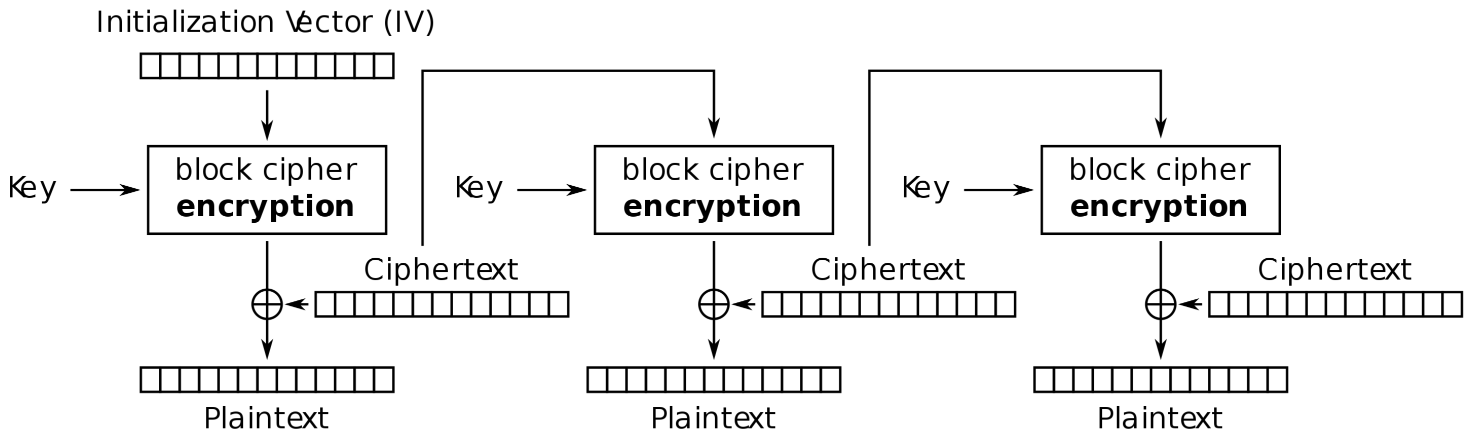
- Most notably standardized and used in some security protocols (ex., Kerberos V4)… but not so common
- Property that can be observed:
  - If two adjacent cyphertext blocks are exchanged this does not affect the decryption of the subsequent blocks, which may be am "insecurity" issue
  - Example: PCBC mode was abandoned in Kerberos V5

CFB mode (Cipher Feedback Mode)

Can use w/ 8,16, … 128 bits



Cipher Feedback (CFB) mode encryption



Cipher Feedback (CFB) mode decryption

CFB Mode Processing

Cipher Feedback Mode (CFB)

Conversion of a block ciphers as stream cipher

(byte-oriented block cipher)

- Message treated as a stream of bits (controlled size)
- Added to the output of the block cipher
- Result is feed back for next stage (hence name)
- Standard allows any number of bit (1,8, 64 or 128 etc) to be feed back
  - Denoted CFB-1, CFB-8, CFB-64, CFB-128 etc
- Most efficient to use all bits in block (64 or 128)

  $C_i = P_i \text{ XOR } D_{K1}(C_{i-1})$

  $C_{-1} = IV$
- Uses: stream data encryption, authentication
- Note: Only need the Encryption Function Implementation !

CFB Mode: Characteristics (1)

Security

- + Appropriate when data arrives in bits/bytes, most common stream mode (without counting)
- + Plaintext concealed, input to block cipher is randomized
- + More than one message can be encrypted with the same key (changing the IV)
- +/- Plaintext somewhat difficult to manipulate

  Only 1st block and last block can be removed)

  Bits changed in the first block: repetition allows controlled changes

Performance

- + Speed is ~the same as the block cipher
- + Cipher text with the same size than plaintext (not counting the IV)
- +/- Encryption not parallelizable, decryption parallelizable and has a random access property
- + Some pre-processing is possible (previous cipher text can be encrypted (before a new plaintext block) – Note that the block cipher is used in **encryption** mode at **both**

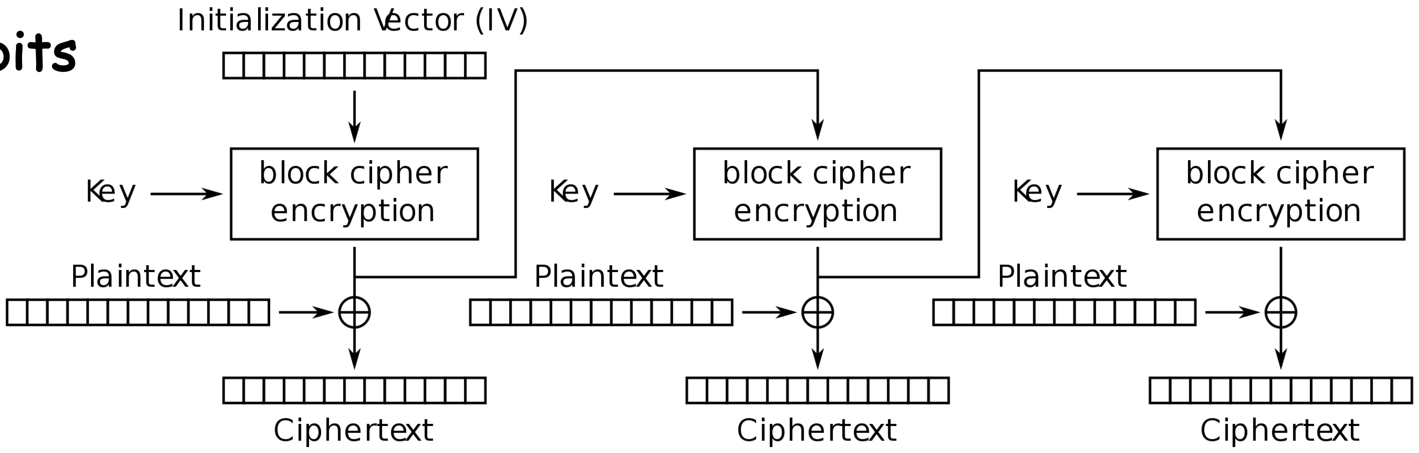
CFB Mode: Characteristics (2)

Fault-tolerance

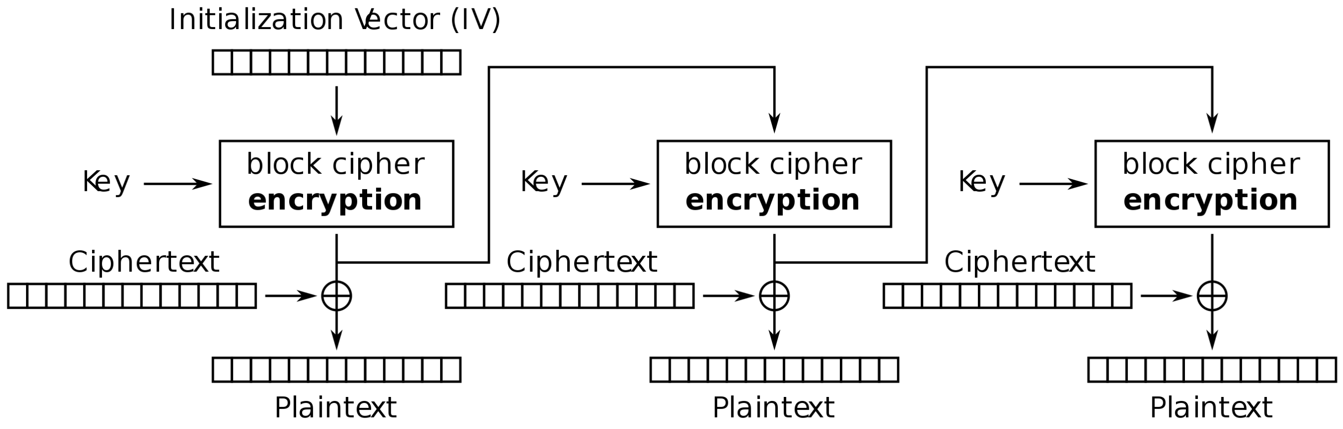
- Limitation is need to stall while do block encryption after every n-bits
- Errors propagate for several blocks after the error: affecting correspondent bits of plaintext and the next full block
- Synchronization errors of full block sizes are recoverable. A 1-bit CFB can recover from the addition or loss of single bits

OFB Mode: Output FeedBack Mode

Can use w/ 8,16, … 128 bits



Output Feedback (OFB) mode encryption



Output Feedback (OFB) mode decryption

OFB Mode Processing

- Message is treated as a stream of bits (allowing sizes of 8, 16, 32. 64, 128 bits)
- Output of cipher is added to message
- Output is then feed back (hence name)
- Feedback is independent of message

- Uses: stream encryption on noisy channels

OFB Mode: Characteristics (1)

Security

- + Plaintext patterns concealed
- + Input to the block cipher is randomized
- + More than one message can be encrypted with the same key (provided that a different IV is used)
- - Plaintext is very easy to manipulate; any change in cipher text directly affects the plaintext - more vulnerable to message stream modification

  A variation of a Vernam cipher
  - hence must **never** reuse the same sequence (key+IV)
- originally specified with m-bit feedback

  (but subsequent research has shown that only **full block feedback** ie CFB-64 or CFB-128) should ever be used for high security)

Performance

- + Speed is ~the same as block cipher
- - Cipher text is the same size as the plaintext, not counting the IV
- + Pre-Processing possible before the message is seen
- -/+ not parallelizable

OFB Mode: Characteristics (2)

Fault-tolerance

- + Bit errors do not propagate - Cipher text error affects only the corresponding bit of plaintext
- - But sender & receiver must remain in sync: synchronization error not recoverable

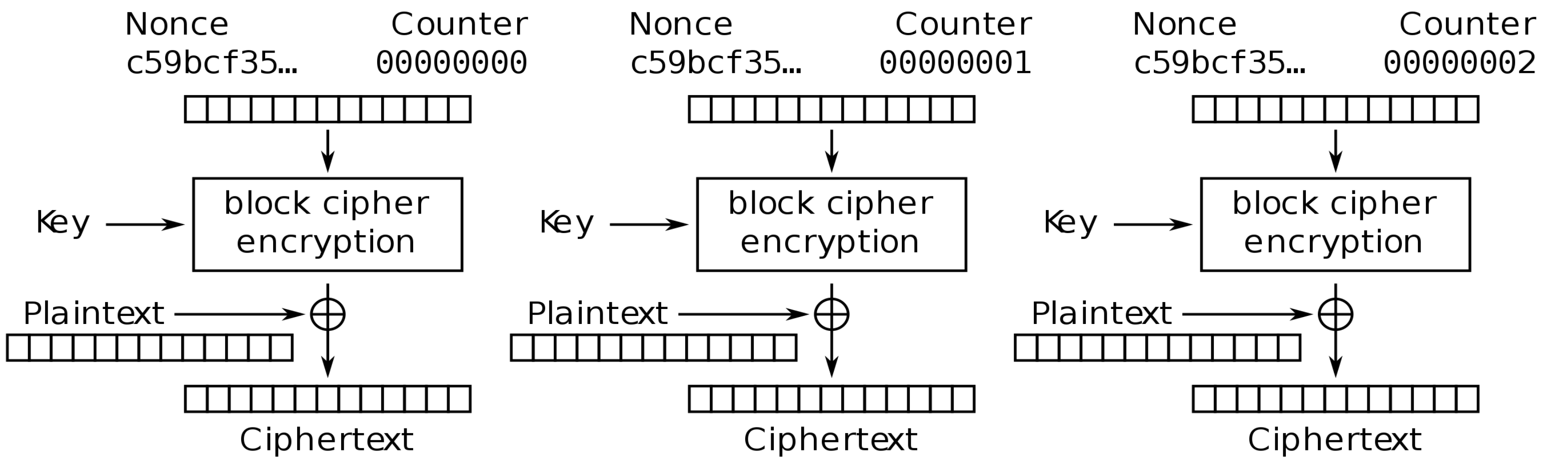
CTR Mode: CounTeR Mode

- A “new” mode, though proposed early on
- Similar to OFB but encrypts counter value rather than any feedback value
- Must have a different key & counter value for every plaintext block (never reused)

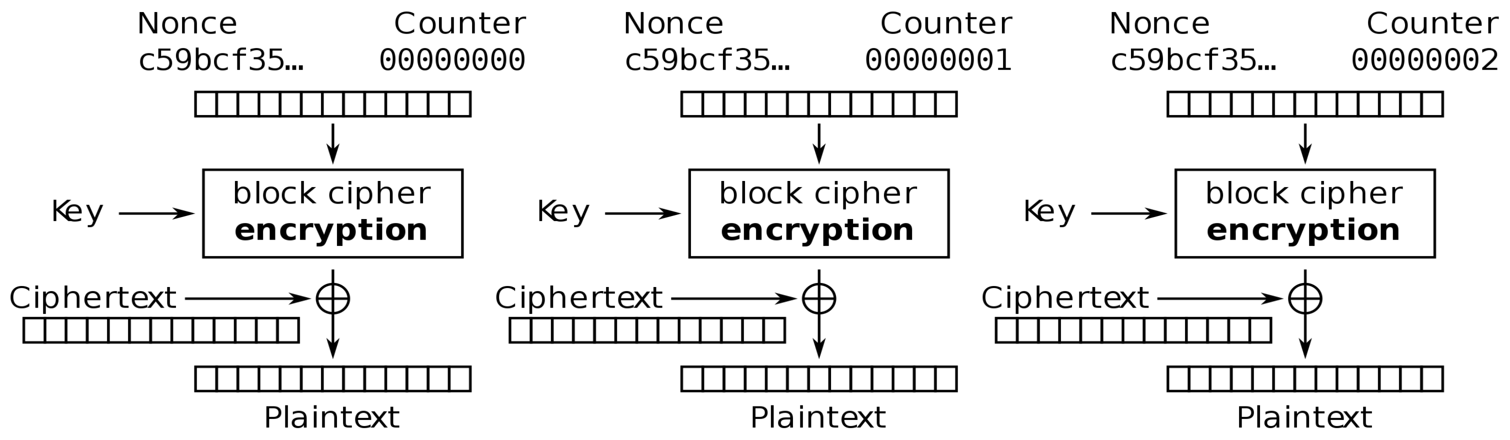
  $C_i = P_i \text{ XOR } O_i$

  $O_i = DES_{K1}(i)$
- Uses: high-speed network encryptions

CTR Mode



Counter (CTR) mode encryption



Counter (CTR) mode decryption

CTR: Characteristics (1)

Security

- Same criteria as OFB
  - Random access to encrypted data blocks
  - Provable security (good as other modes)
  - But must ensure never reuse key/counter values, otherwise could break (cf OFB)

Performance

- Efficiency
  - + ~The same Better than OFB
    - Parallelizable: can do parallel encryptions in h/w or s/w
    - can pre-process in advance of need
    - good for bursty high speed links

Performance

as OFB

Summary of Block-Cipher Modes of Operation: characteristics vs. tradeoffs

Security

- CBC (entire blocks), CFB (mainly with m-bit blocks, more secure => larger m)
- ECB, OFB and Counter: plaintext easy to manipulate

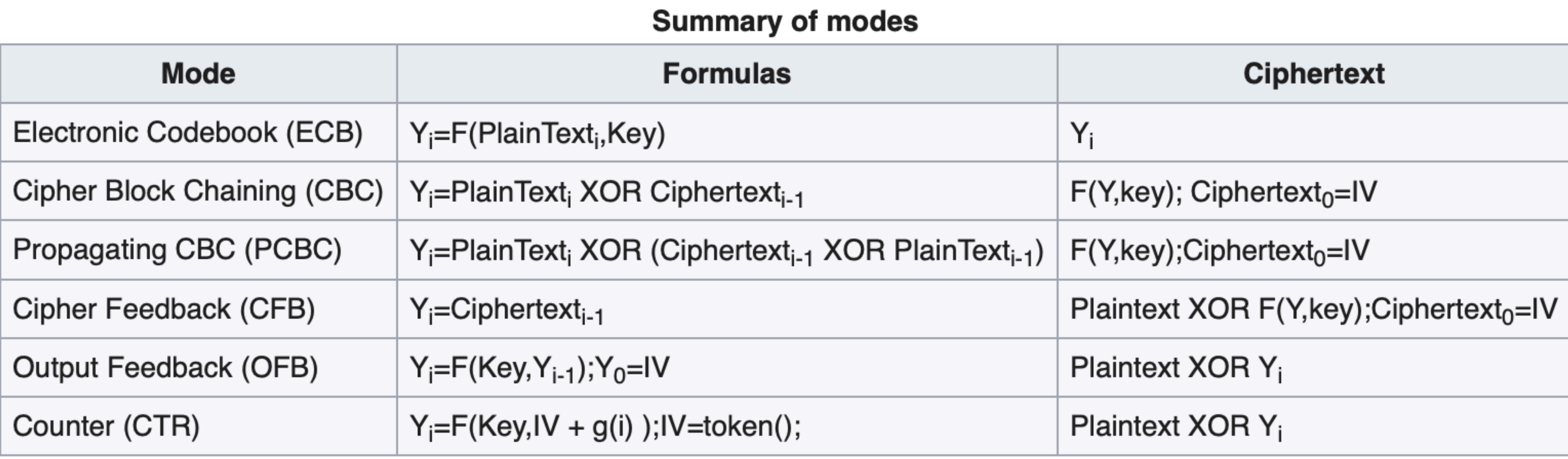
Performance

- ECB, CTR (parallelizable)
- CFB (only partial parallelization in the decryption)
- CBC same but no pre-processing
- OFB no parallelization, no pre-processing

Fault-Tolerance

- CFB - synchronization errors of full block sizes are recoverable, … but cipher text error affect the correspondent bit of plaintext + the next full block), OFB, CTR (ciphertext errors only affect the correspondent bit in plaintext)
- ECB, CBC, OFB, CTR - Sync. Errors unrecoverable,

Symmetric Modes of Operation: Summary of Representation and Formulas (1)



Summary of modes

Mode	Formulas	Ciphertext
Electronic Codebook (ECB)	Y_i=F(PlainText$_i$,Key)	Y_i
Cipher Block Chaining (CBC)	Y_i=PlainText$_i$ XOR Ciphertext$_{i-1}$	F(Y,key); Ciphertext$_0$=IV
Propagating CBC (PCBC)	Y_i=PlainText$_i$ XOR (Ciphertext$_{i-1}$ XOR PlainText$_{i-1}$)	F(Y,key);Ciphertext$_0$=IV
Cipher Feedback (CFB)	Y_i=Ciphertext$_{i-1}$	Plaintext XOR F(Y,key);Ciphertext$_0$=IV
Output Feedback (OFB)	Y_i=F(Key,Y_{i-1});Y_0=IV	Plaintext XOR Y_i
Counter (CTR)	Y_i=F(Key,IV + g(i));IV=token();	Plaintext XOR Y_i

Symmetric Modes of Operation: Summary of Representation and Formulas (2)

ECB: $Ci = E_k(Pi)$; $Pi = D_k(Ci)$

CBC: $Ci = E_k(Pi) \text{ XOR } Ci-1$ | $Co = IV$

$Pi = D_k(Ci) \text{ XOR } C_{i-1}$

PCBC: $Ci = E_k(Ci-1) \text{ XOR } Pi$ | $Co = IV$

$Pi = D_k(Ci-1) \text{ XOR } Ci$

CBF: $Ci = \text{head}(E_k(Si-1), x) \text{ XOR } Pi$ | $Si = ((Si-1 \ll x) + Ci) \bmod 2^n$

$Pi = \text{head}(E_k(Si-1), x) \text{ XOR } Ci$ | $So = IV$

OFB: $Cj = Pj \text{ XOR } Oj$ | $Oj = E_k(Ij)$, $Ij=Oj-1$, $Io= IV$

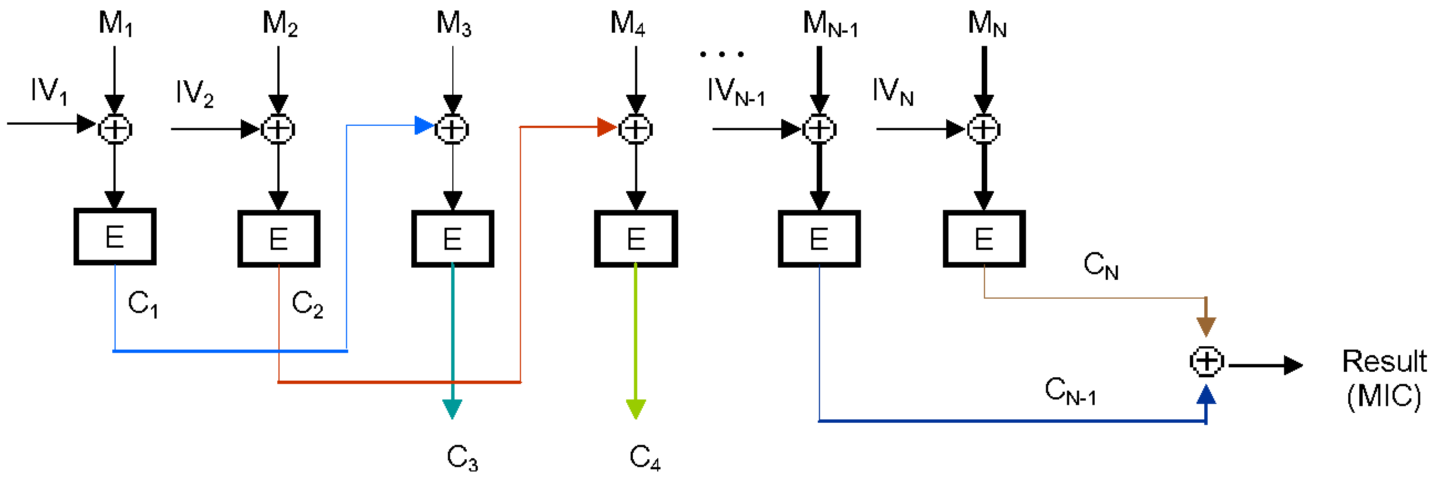
$Pj = Cj \text{ XOR } Oj$

CTR: $Ci = Ek(n\text{-}ctr_i) \text{ XOR } Pi$ | $n\text{-}ctr_{i+1} = Fc(n\text{-}ctr_i)$, $n\text{-}ctr_0 = IV$

$Pi = Ek(n\text{-}ctr_i) \text{ XOR } Ci24$

Other Cipher Modes of Operation: Interleaving and Mux/Demux Techniques

- Interleaving technique (Ex., CBC Interleaving)



- Multiplex/Demultiplex with Interleaving
  Single messages
  Multiple messages

Other Cipher Modes: CTS, CCM and GCM

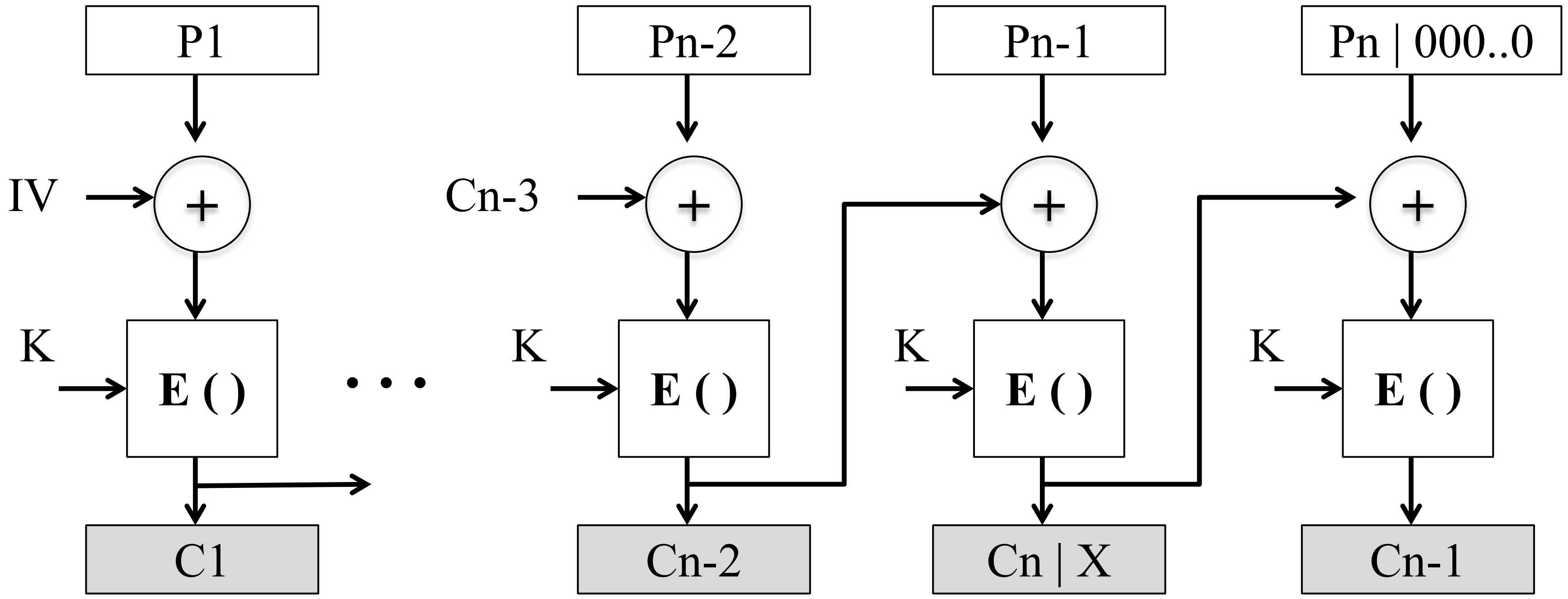
- CTS (Cyphertext Stealing Mode)
- CCM, GCM: Modes w/ Implicit Authentication (CMAC)
  - AES-CCM :
    - IETF RFC 6665 (used in TLS Cryptosuites)
    - IETF RFC 4309 (used in IPSec Ciphersuites)
  - GCM

See more here (papers ref.):

https://en.wikipedia.org/wiki/CCM_mod

https://en.wikipedia.org/wiki/Galois/Counter_Mode

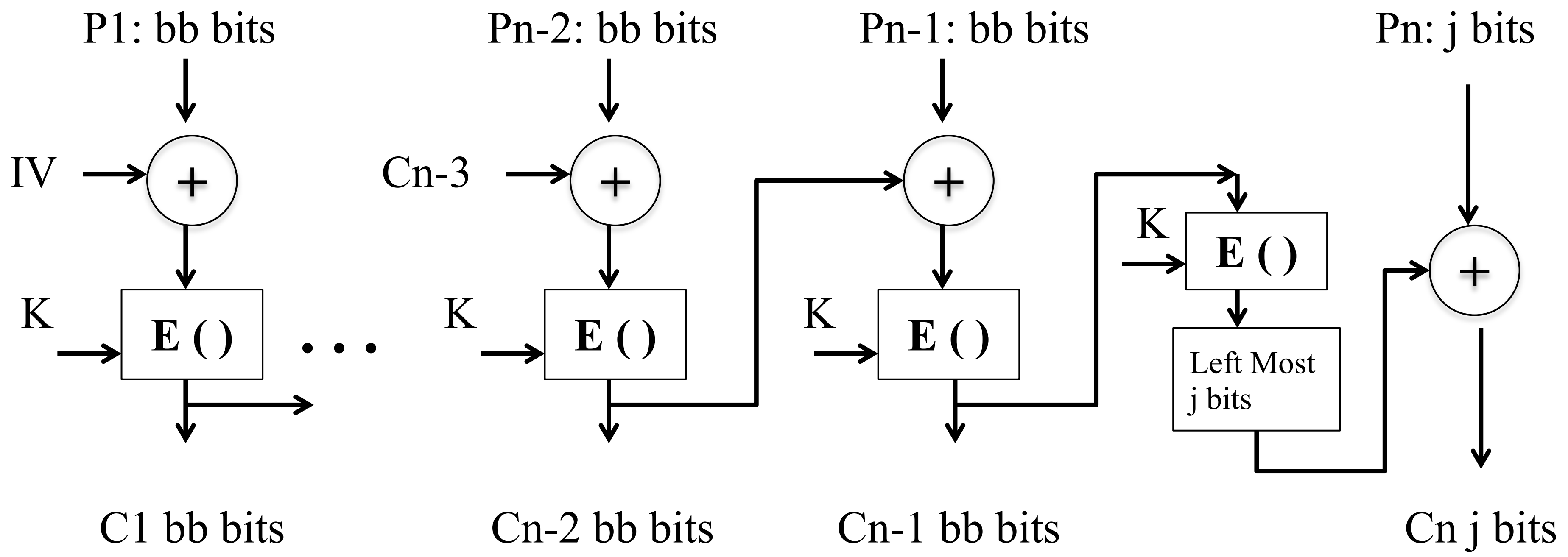
CTS Mode - Ciphertext Stealing



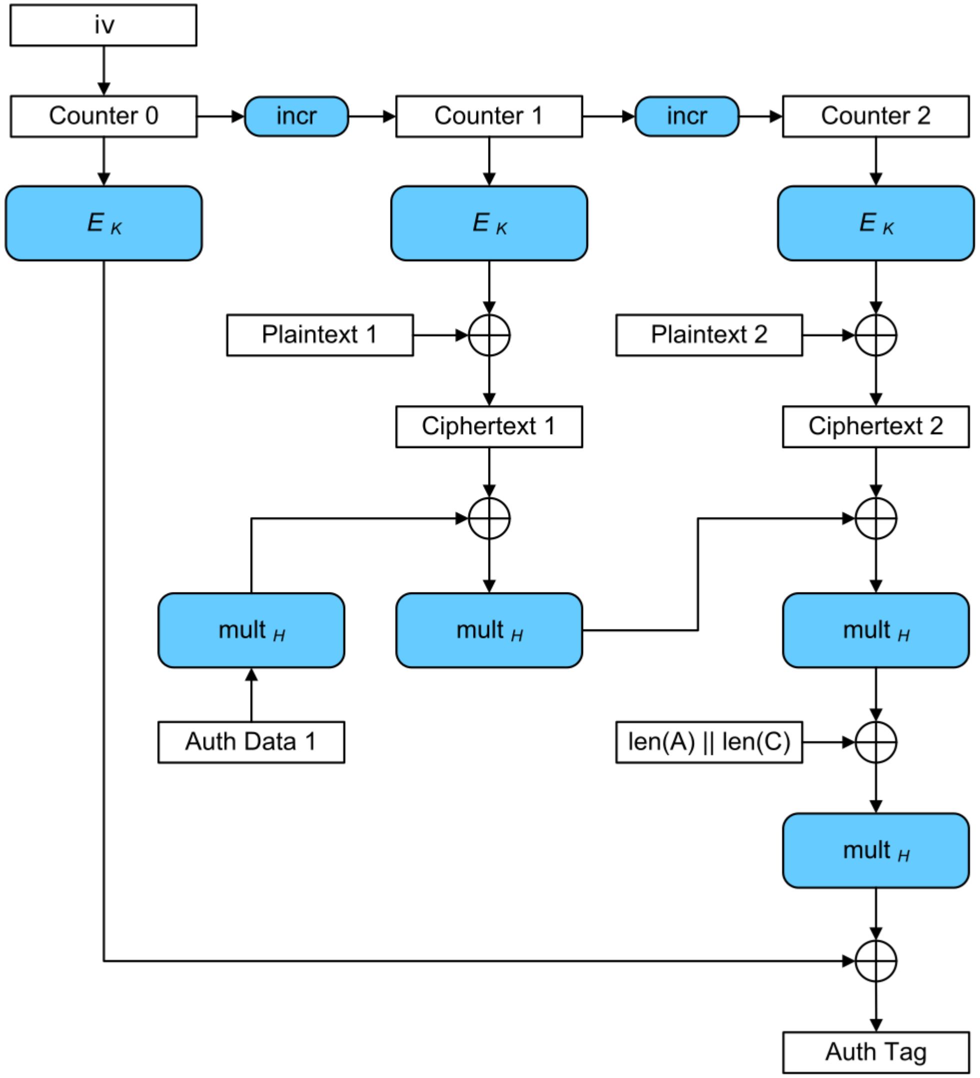
Similar to CBC Considerations but ciphertext will have the Same size as plaintext

(Think if you want to store encrypted data in the same memry buffer initially storing plaintext data)

CTS Mode (alternative)



GCM (Galois Counter Mode)



$\mathrm{GF}(2^{128}) =$
$x^{128} + x^7 + x^2 + x + 1$

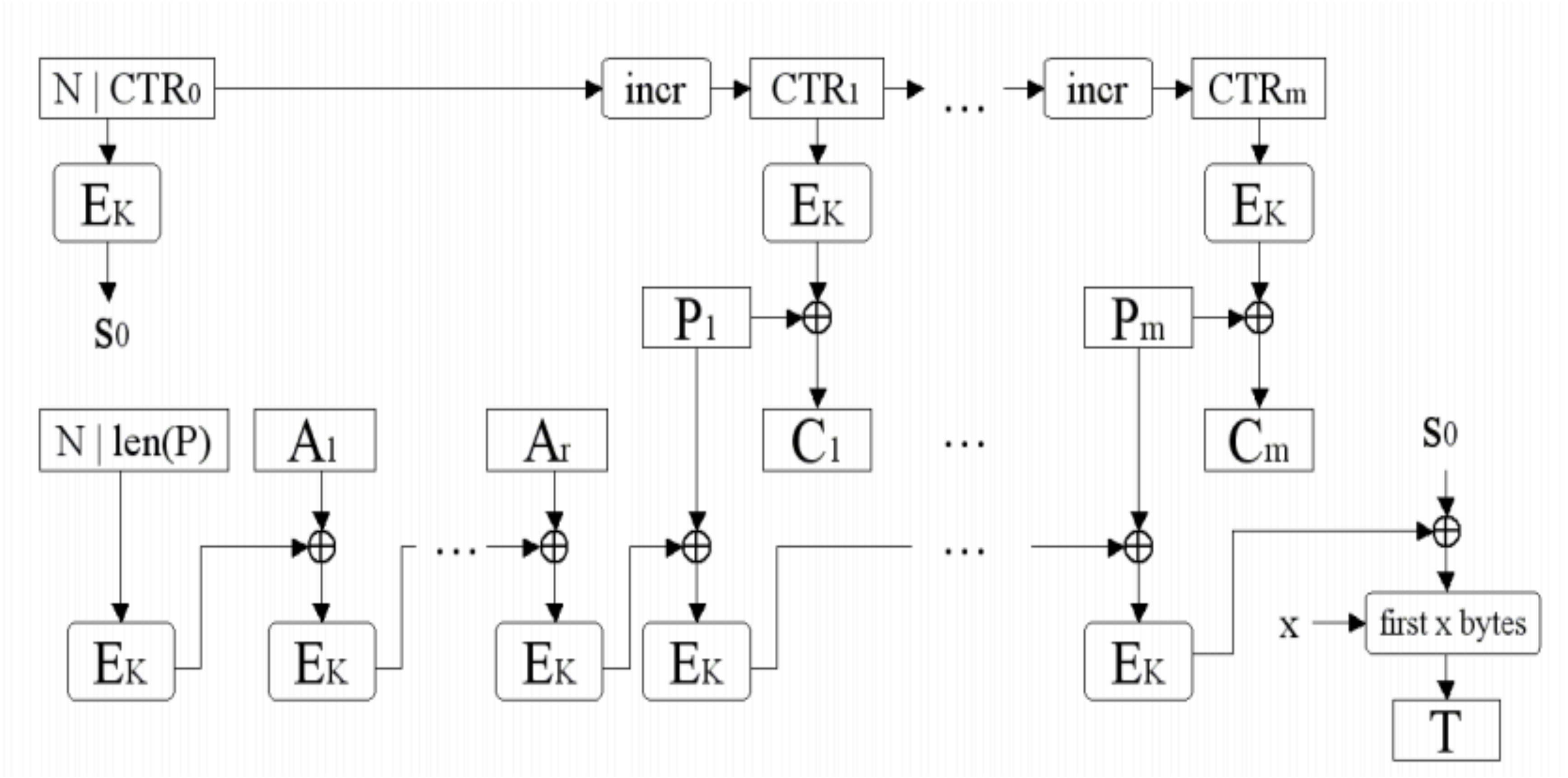
$\mathrm{GHASH}\ (H,A,C) = X_{m+n+1}$

©

Can see in: https://en.wikipedia.org/wiki/Galois/Counter_Mode

CCM (Counter w/ CBC MAC)

Combines CBC-MAC scheme for a data authentication and CTR mode: A "*mac-then-encrypt*" approach



Other modes have been proposed ...

- No so popular/implemented in crypto libraries, some examples:
  - IAPM
  - XCBC
  - EAX
  - CWC

Today: More on Symmetric Cyptography

Outline:

- **More on Symmetric Cryptography**
  - More on symmetric cryptography
    - Robustness of symmetric encryption algorithms:
      - Block-Ciphers
      - Stream-Ciphers
    - Cipher block modes of operation
    - Relevance of Padding
    - Randomness and pseudo-randomness
    - Some algorithms:
      - Block Ciphers: DES, 3DES, IDEA, AES
      - Stream Encryption Algorithms and RC4
    - Bit-Stream Ciphers using Block Ciphers

Padding

- At end of message must handle a possible last short block
  - Which is not as large as blocksize of cipher
  - PAD either with known non-data value (eg nulls)
  - or PAD last block following a standardizd padding scheme
  - When using Padding, it must be required an extra entire block over those in message (when multiples of blocks)

- Padding standardization (next slide):

  See practical classes / labs - examples

- There are other, more esoteric modes, which avoid the need for an extra block

- Easy to adopt secure padding for specific designed protocols

- See this in more detail in LABs

Some Standardized Padding Schemes (1)

ZERO BIT PADDING, RFC 1321, ISO/IEC 9797-1 (called Padding Method 2):

... | 1011 1001 1101 0100 0010 0111 **0000 0000** |

ZERO PADDING

... | DD DD DD DD DD DD DD DD | DD DD DD DD **00 00 00 00** |

ANSI X9.23

... | DD DD DD DD DD DD DD DD | DD DD DD DD **00 00 00 04** |

ISO 10126

... | DD DD DD DD DD DD DD DD | DD DD DD DD **81 A6 23 04** |

PKCS#5, PKCS#7, RFC 5652

... | DD DD DD DD DD DD DD DD | DD DD DD DD **04 04 04 04** |

Some Standardized Padding Schemes (2)

ISO/IEC 7816-4

... | DD DD DD DD DD DD DD DD | DD DD DD DD **80 00 00 00** |
... | DD DD DD DD DD DD DD DD | DD DD DD DD DD DD DD **80** |

What about a more secure padding (for improved paranoia) ?

Ex:

... | DD DD DD DD DD DD DD DD | DD DD 06 XX XX XX XX XX |

Msb_{30} (SecureHash (m))

Today: More on Symmetric Cyptography

Outline:

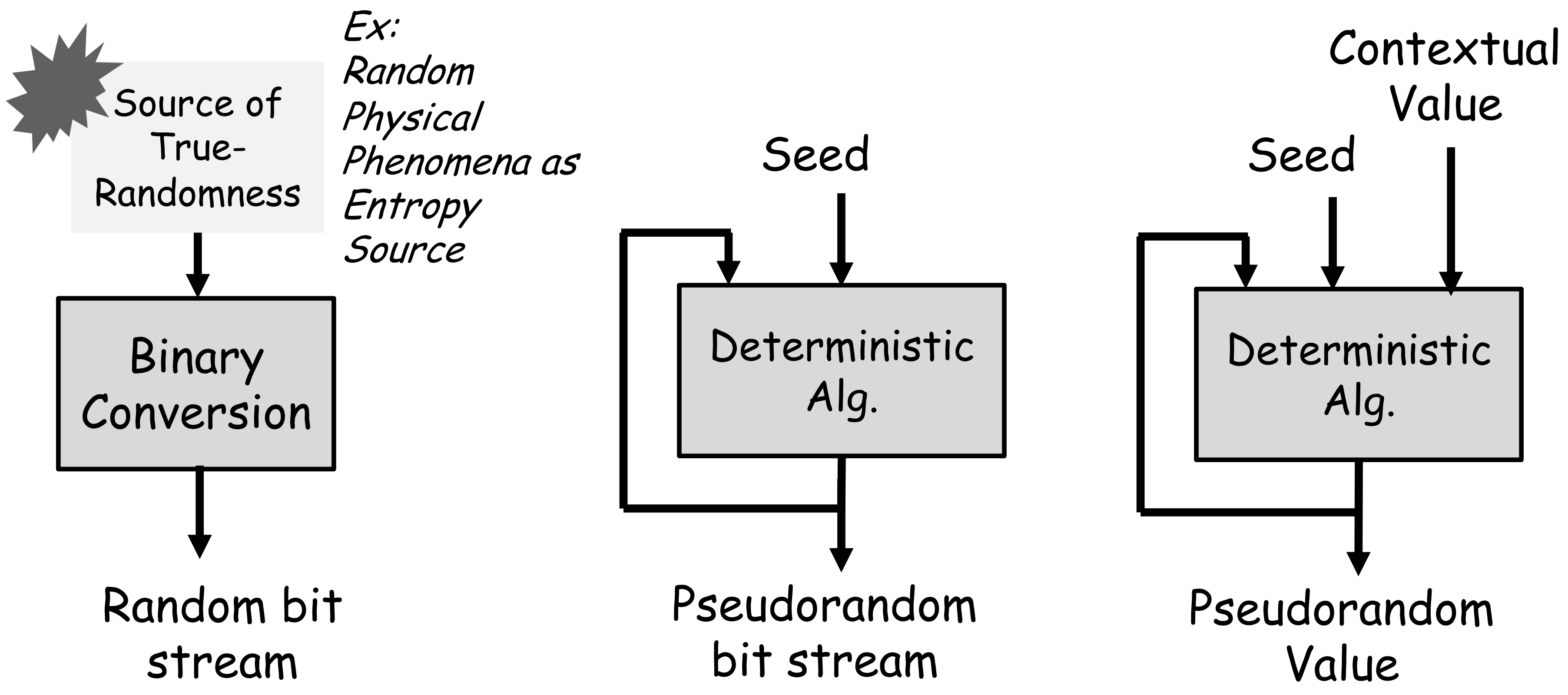
- **More on Symmetric Cryptography**
  - More on symmetric cryptography
    - Robustness of symmetric encryption algorithms:
      - Block-Ciphers
      - Stream-Ciphers
    - Cipher block modes of operation
    - Relevance of Padding
    - Randomness and pseudo-randomness
    - Some algorithms:
      - Block Ciphers: DES, 3DES, IDEA, AES
      - Stream Encryption Algorithms and RC4
    - Bit-Stream Ciphers using Block Ciphers

Random Numbers (RNs) and Pseudo Random Numbers (PRNs)

- Relevant:
  - Key-Generation and generation of other considered secrecy parameters: Passwords, IVs, Nonces, Tokens, …
- Secure random functions must be:
  - **Secure** here means the randomness characteristic:
    - Uniform distribution
      - Probability of bit generation in random words should be exactly the same (for perfect randomness) or approximately the same (for pseudorandmness)
    - Independence: no one subsequence can be inferred from another
  - **Fast** (ex., use in fast rekeying – successive generation of temporary session keys)

TRNG, PRNG and PRF

- Structure of functions:



TRNG
True-Random Number Generator

PRNG
Pseudo-Random Number Generator

PRF
Pseudo-Random Function

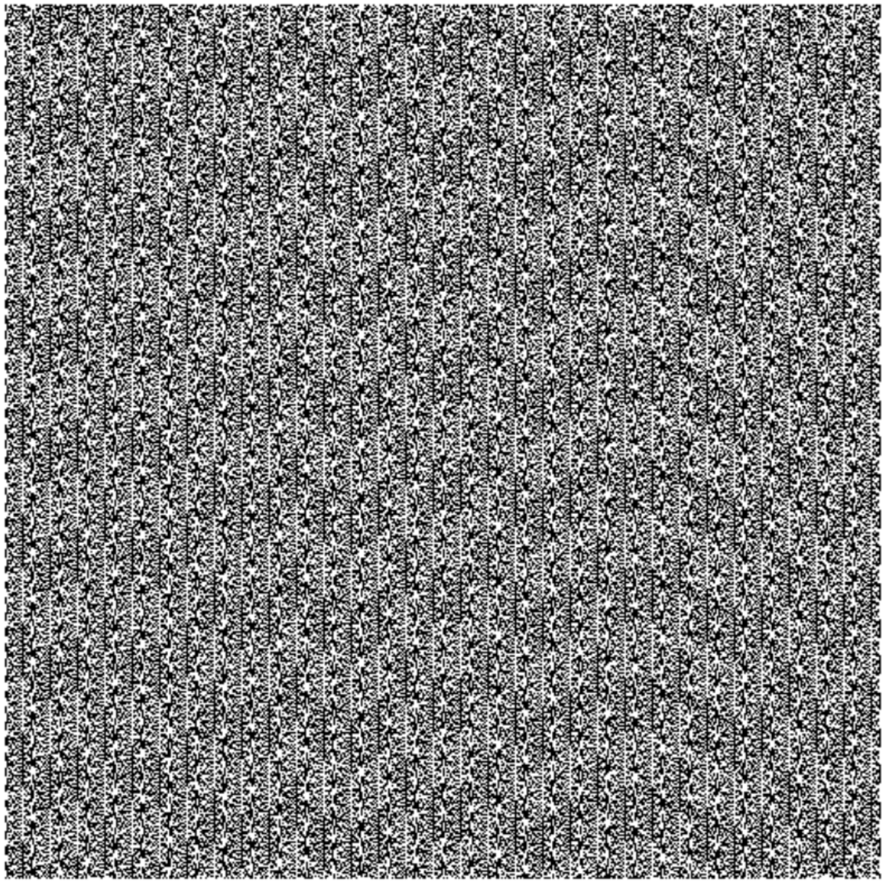
Java Secure (or Strong) Random Generation

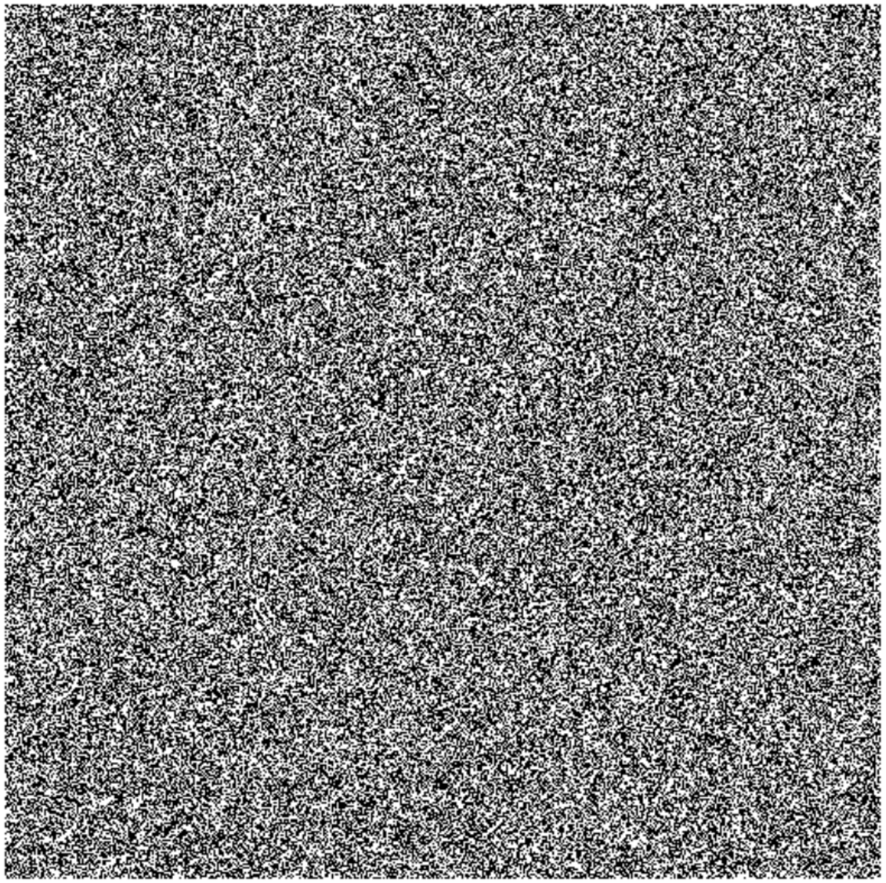
- java.lang.Object
  - java.util.Random
    - java.security.SecureRandom
- **Class SecureRandom**
- FIPS PUB 140-2 Statistical Tests Compliance
- But input usable seeds are determinant

https://docs.oracle.com/javase/8/docs/api/java/security/SecureRandom.html

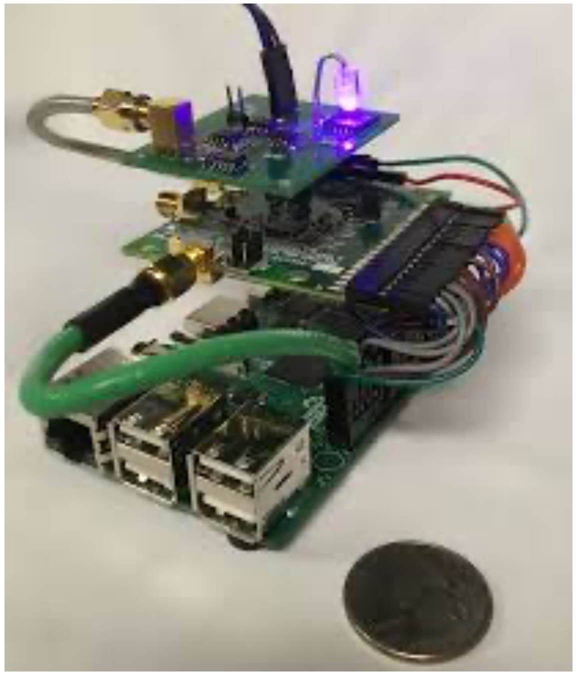
Java Secure (or Strong) Random Generation

- See differences ...


Obtained w/ java.util.Random:
Uses LCG
Linear Congrential Generator Algorithm


Obtained w/ java.security.SecureRandom:
Uses CSPRNG - PaeudoRandom Number
Generation FIPS-PUB 140-2

Example of TRNG Devices



Today: More on Symmetric Cyptography

Outline:

- **More on Symmetric Cryptography**
  - More on symmetric cryptography
    - Robustness of symmetric encryption algorithms:
      - Block-Ciphers
      - Stream-Ciphers
    - Cipher block modes of operation
    - Relevance of Padding
    - Randomness and pseudo-randomness
    - Some algorithms:
      - Block Ciphers: DES, 3DES, IDEA, AES
      - Stream Encryption Algorithms and RC4
    - Bit-Stream Ciphers using Block Ciphers

Internal Structure of Algorithms

- See complementary materials:
  - Feistel Structure for Symmetric Block Encryption/Decryption
  - DES Internal Structure and 3DES
  - AES Internal Structure

See in the end of these slides and in the Book:
Stallings, Network Security Essentials Chap.2

Stream Ciphers Characteristics and Structure

- Operation based on encrypting plaintext streams (ex., bit-stream encryption or byte-stream encryption
- Core based on secure Keystream Generators
  - Input key used as secret seed
    - Different sizes for different algorithms
  - Generated keystreams must have Long Periods
    - In a perfect (ideal) case, perfect random keystream generation with no patterns' repetitions at all !
    - In concrete implementations and practical use: long-repetitions will prevent weaknesses for "finite sized input plaintext streams"
    - Repetitions => Weaknesses
  - Ciphertext streams can be computed from input streams and key-streams with simple, lightweight operations: ex., bit XOR

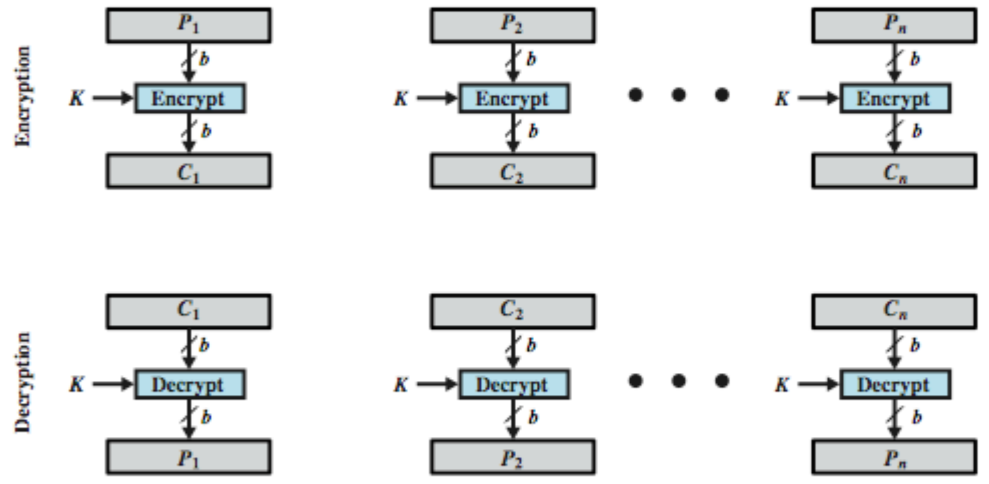
  - => Encryption/Decryption can be very fast !

    Compare with blockciphers:

    see, ex., openssl speed des aes rc4

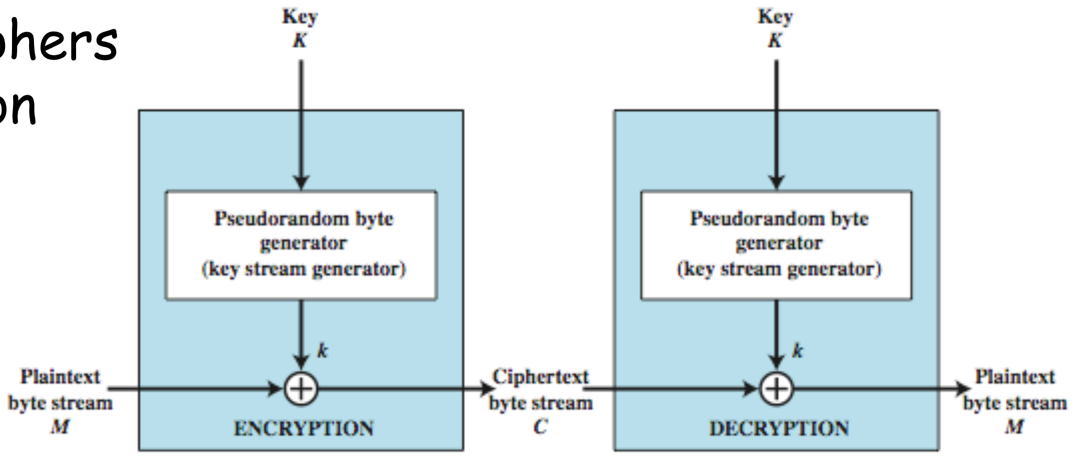
Stream ciphers

- Block cipher (with ECB mode of operation)



(a) Block cipher encryption (electronic codebook mode)

- Structure for stream ciphers
    - Possible implementation with block ciphers.
    - How ?



Key
K
Pseudorandom byte generator (key stream generator)
k
Plaintext byte stream M
Ciphertext byte stream C
ENCRYPTION
DECRYPTION

(b) Stream encryption

Stream ciphers

- Process message bit by bit (as a stream)
  - This is different than the use of byte-oriented block-encryption, like CTR, OFC or CFB
  - Symmetric Block-Encryption (using those modes) is not a stream cypher algorithm
- Have a pseudo random **keystream**
- Combined (XOR) with plaintext bit by bit
- Randomness of **stream key** completely destroys statistically properties in message
  - $C_i = M_i$ XOR StreamKey_i
- But must never reuse stream key: Problem of the Period in Stream Cipher Algorithms
  - otherwise can recover messages (cf book cipher)
- Some design considerations are:
  - long period with no repetitions, Statistically random
  - depends on large enough key
  - large linear complexity
- Properly designed, can be as secure as a block cipher with same size key, but usually simpler & faster

RC4 Algorithm

- Proprietary cipher owned by RSA DSI
  - Code revealed, known-algorithm
- Another Ron Rivest design, simple but effective
- Variable key size, byte-oriented stream cipher
- Widely used (web SSL/TLS, wireless WEP)
- Key forms random permutation of all 8-bit values
- Uses that permutation to scramble input info processed a byte at a time

RC4 Key Schedule

- Starts with an array *S* of numbers: *0..255*
- Use key to well and truly shuffle
- *S* forms **internal state** of the cipher

```
// Initialization, S and T
for i = 0 to 255 do
   S[i] = i
   T[i] = K[i mod keylen])

// Initial Permutation of S
j = 0
for i = 0 to 255 do
   j = (j + S[i] + T[i]) (mod 256)
   swap (S[i], S[j])
```

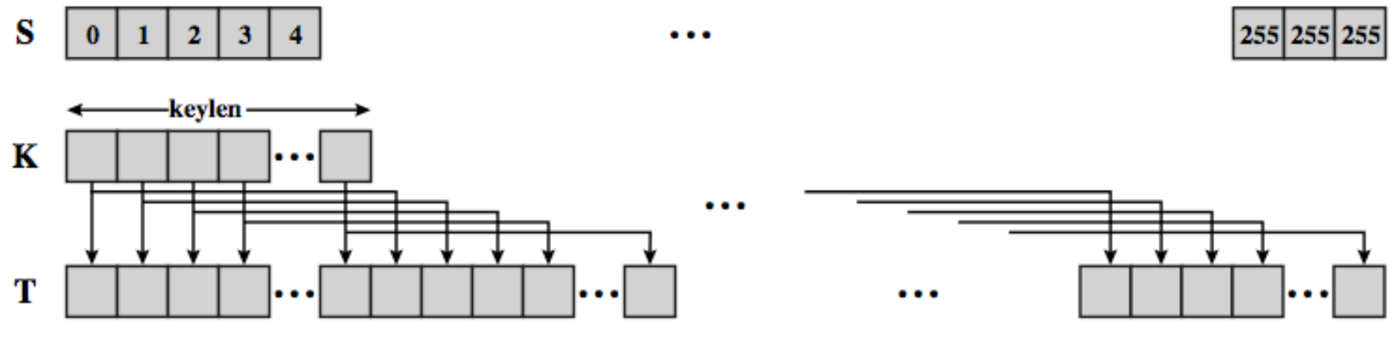
RC4 Encryption

- Encryption continues shuffling array values
- Sum of shuffled pair selects "stream key" value from permutation
- XOR S[t] with next byte of message to en/decrypt

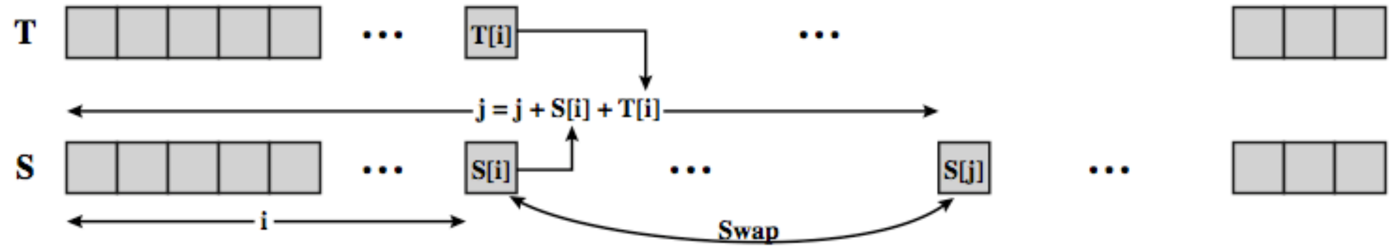
```
i = j = 0
for each message byte $M_i$
    i = (i + 1) (mod 256)
    j = (j + S[i]) (mod 256)
    swap(S[i], S[j])
    t = (S[i] + S[j]) (mod 256)
    $C_i$ = $M_i$ XOR S[t]
```

RC4 encryption model

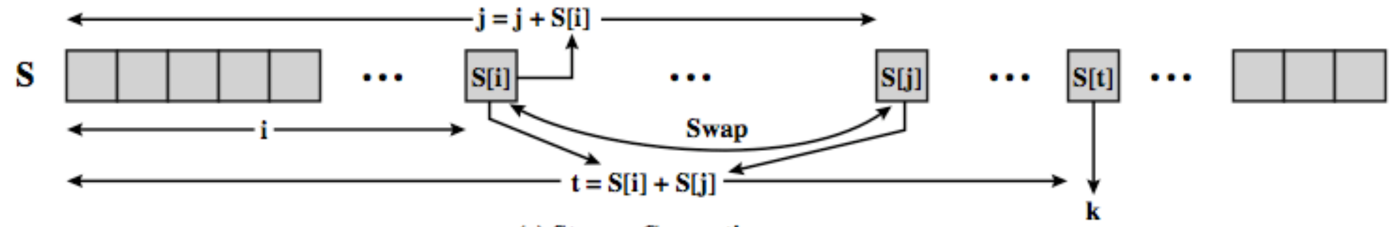
- Algorithm explained in Stallings, Network Security Essentials (see section 2.4, Chapter 2)



(a) Initial state of S and T



(b) Initial permutation of S



(c) Stream Generation

RC4 Security

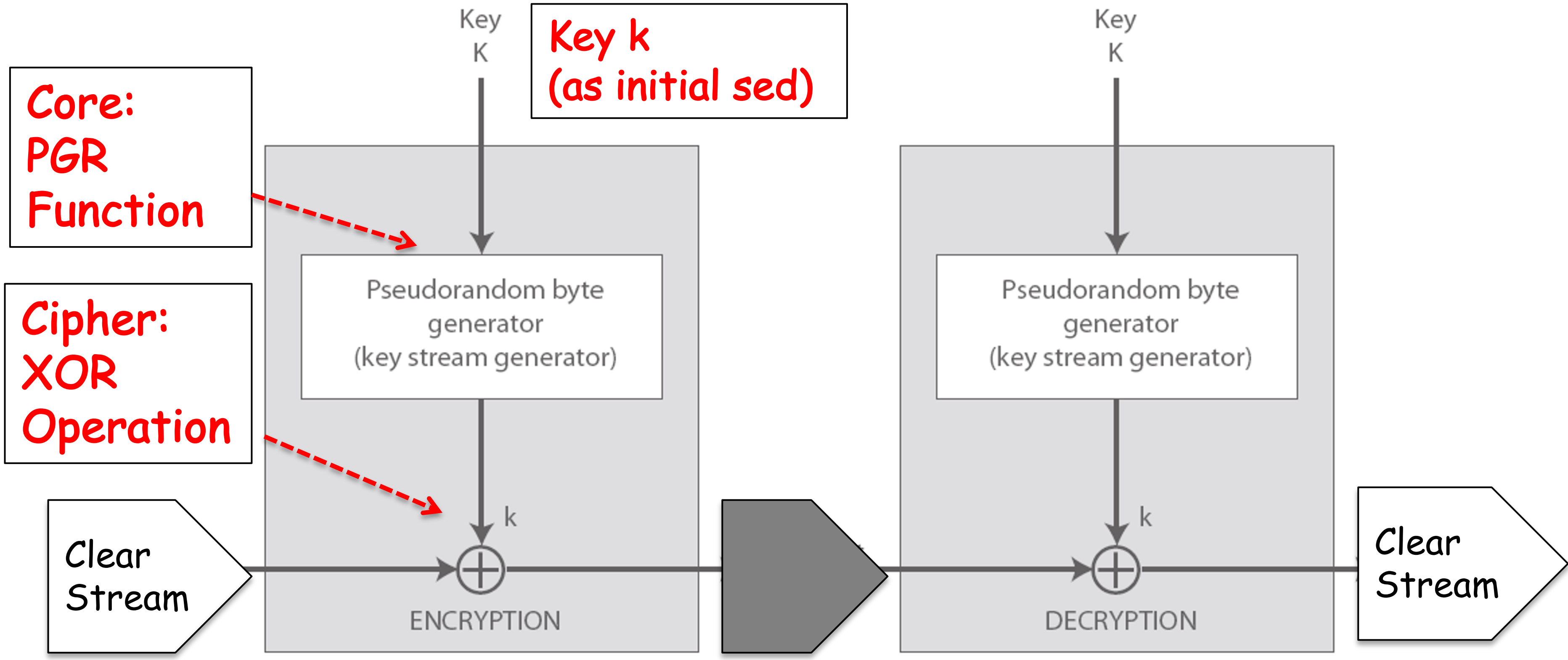
- Claimed secure against known attacks
  - have some analyses, none practical
- Result is very non-linear
- Since RC4 is a stream cipher, must **never reuse a key**
- Have a concern with WEP, **but due to key handling and/or pattern-repetitions** rather than RC4 itself

Other stream ciphers

- SEAL
- "A" symmetric stream ciphers series (ex., A5, A8, in Industry/GSM)
- … Many others …
  - See more: https://en.wikipedia.org/wiki/Stream_cipher

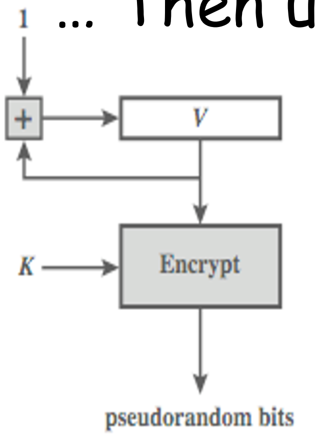
Bit-Stream Encryption using Block Ciphers

Can have real-time bit streaming using only block-ciphers ?
- Yes, look to the generic stream cipher architecture …

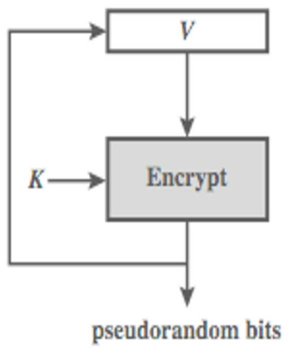


Bit Streaming using Block Ciphers

… Then use Block Ciphers to implement PRG Boxes



(a) CTR Mode



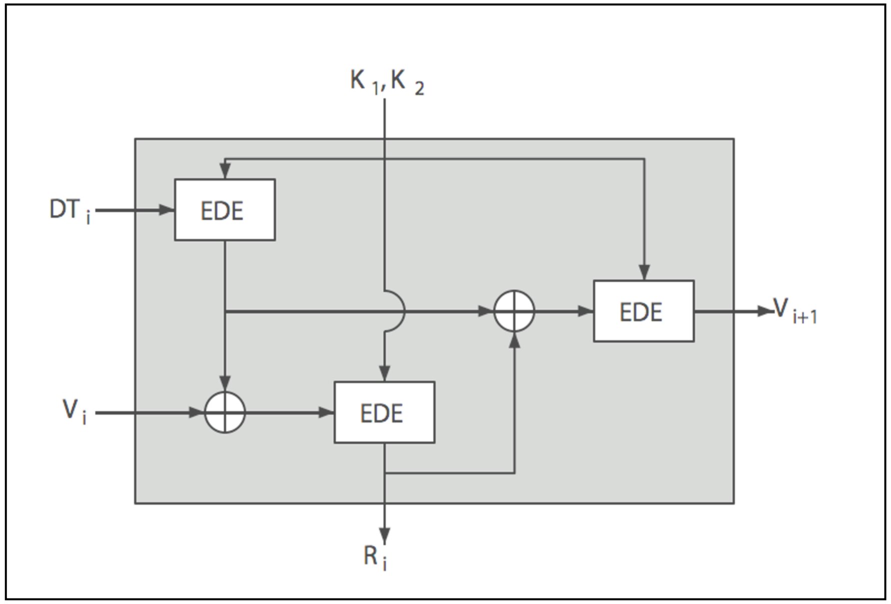
(b) OFB Mode

Example of the
ANSI X9.17 PRG Specification
Using 3DES

Keys K1, K2 (+ opt K3)

DTi
Initial Timestamp
(or counter-state)

Seed Value



Questions ?

Recommended Reading (with the slides)

Symmetric Encryption and Message Confidentiality

Reading:

W. Stallings, Network Security Essentials – Applications and Standards, Part I – Cryptography, Chap.2 – Symmetric Encryption and Message Confidentiality

Complementary Materials

(Informative) ANNEX:

Complementary Materials on:

- Internal Structure of Symmetric Crypto Algorithms
- Feistel Structure and DES Algorithm
- AES Algorithm
- Other Cryptographic Constructions using Symmetric Cryptographic Algorithms

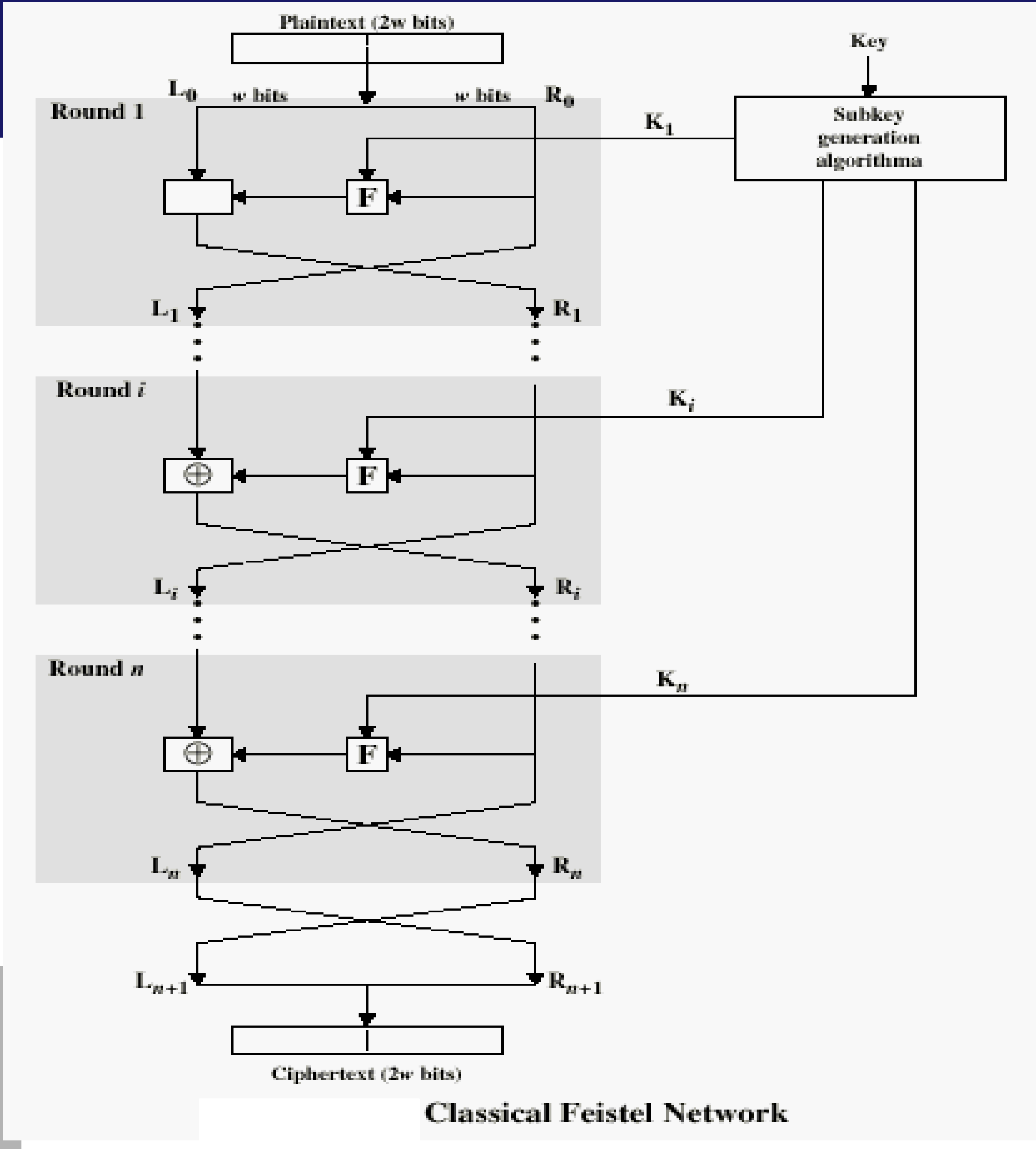
Ref. Feistel Structure

Horst Feistel,
IBM, 1973

Base structure
for many symmetric
algorithms
(ex., DES)

Decryption
implemented as
Encryption
(input: ciphertext,
and use of sub-keys
in the reverse order)

In each round:
Li = Ri-1
Ri = Li-1 xor F(Ri-1, Ki)



Classical Feistel Network

Feistel Structure: summary to retain

- A generic structure for block-encryption algorithms, as a sequence of rounds (of core algorithm) performing substitutions and permutations, parameterized by a secret subkey
- Subkeys: derived in each round from a key-scheduling algorithm
- For a particular instantiation (specific algorithm), we must define:
  - The blocksize,
  - The keysize
  - Sub-key generation (scheduling) algorithm
  - The core round function (and related transformations)
  - Number of rounds for required confusion and diffusion

Taking into account:

- Fast encryption/decryption
- SW and HW implementation possibility
- Core round and sub-key generation algorithm (ease of analysis for cryptanalysis)
- Complexity analysis and soundness proofs

DES Algorithm

DES

Data Encryption Standard (DES)

- The most widely used encryption scheme, until 2000
- Also referred as Data Encryption Algorithm (DEA)
- DES is a block cipher
  - Input (plaintext) Blocks 64-bit blocks
  - The key: 56-bits in length

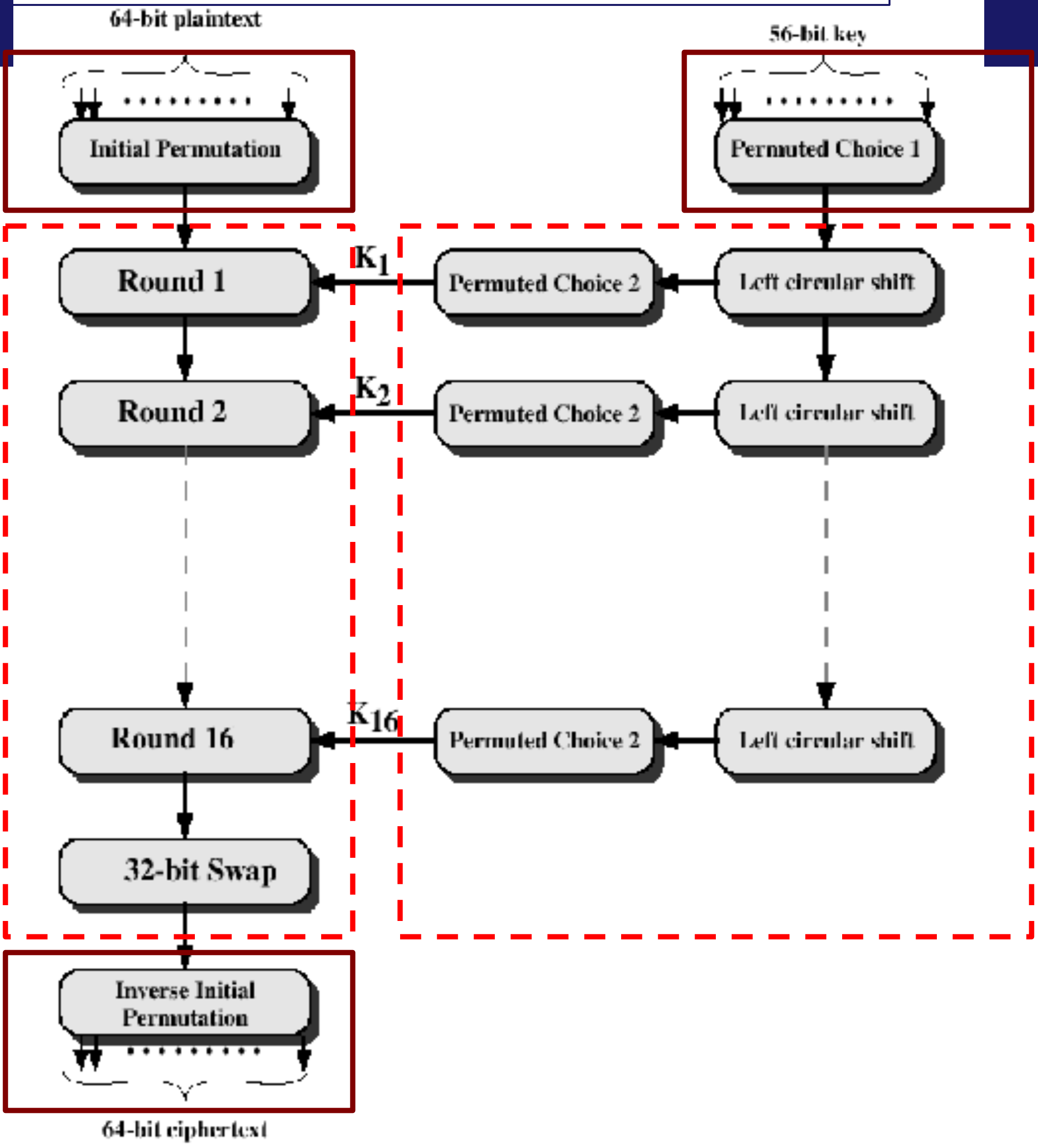
DES Algorithm

Base: Feistel structure:
The overall processing
at each iteration (round):

$L_i = R_{i-1}$

$R_i = L_{i-1} \quad F(R_{i-1}, K_i)$

- Security concerns about:
  - >The algorithm and the key length (56-bits)
  - > Weak, semi-weak or potentially weak keys
  - > 1998, EFF, DES cracker machine (3 days)
  - > 10 hours (machine Doing 10^6 decryptions/microsec



Initial and final permutations

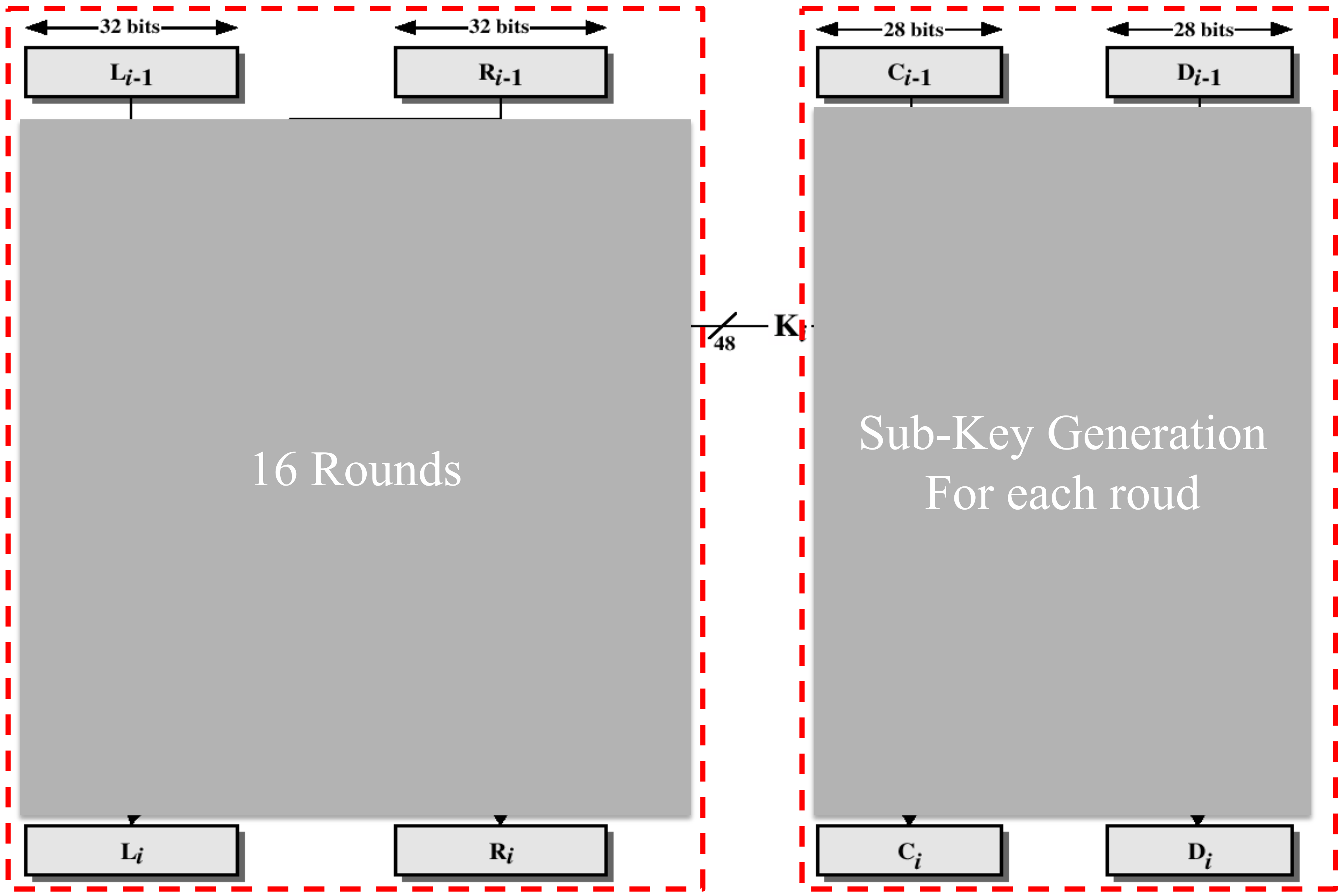
(a) Initial Permutation (IP)

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

(b) Inverse Initial Permutation (IP^{-1})

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

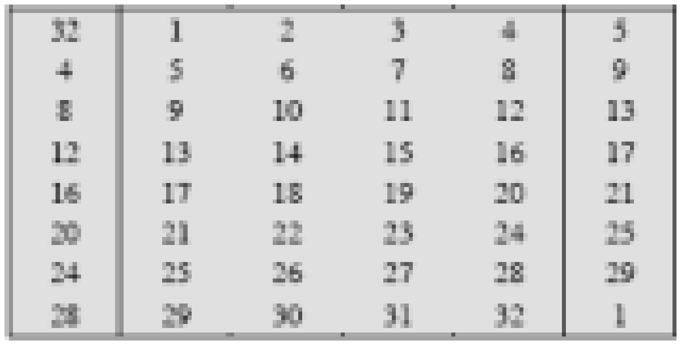
```
Ex: Init Permut(675a6967 5e5a6b5a) = (ffb2194d 004df6fb)
0110011101011010011010010110011101011110010110100110101101011010
1111111110110010…..etc
```



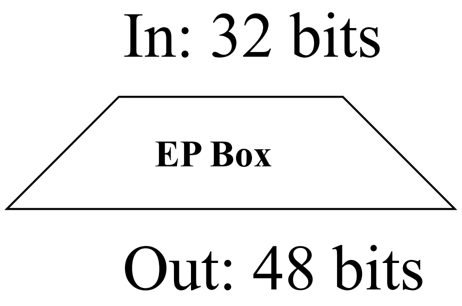
Single Round of DES Algorithm

Expansion/Permutation (EP box) and Pbox

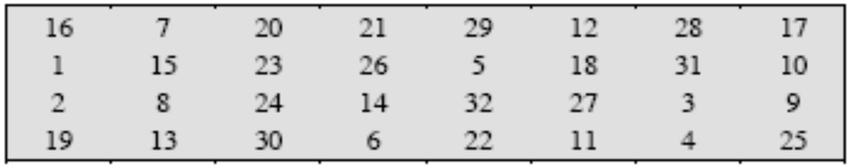
(c) Expansion Permutation (E)



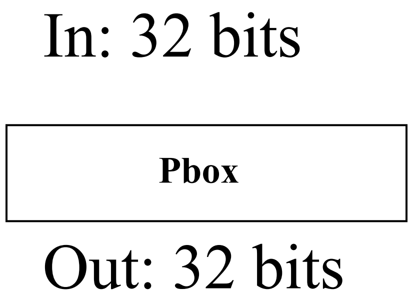
32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1



(d) Permutation Function (P)



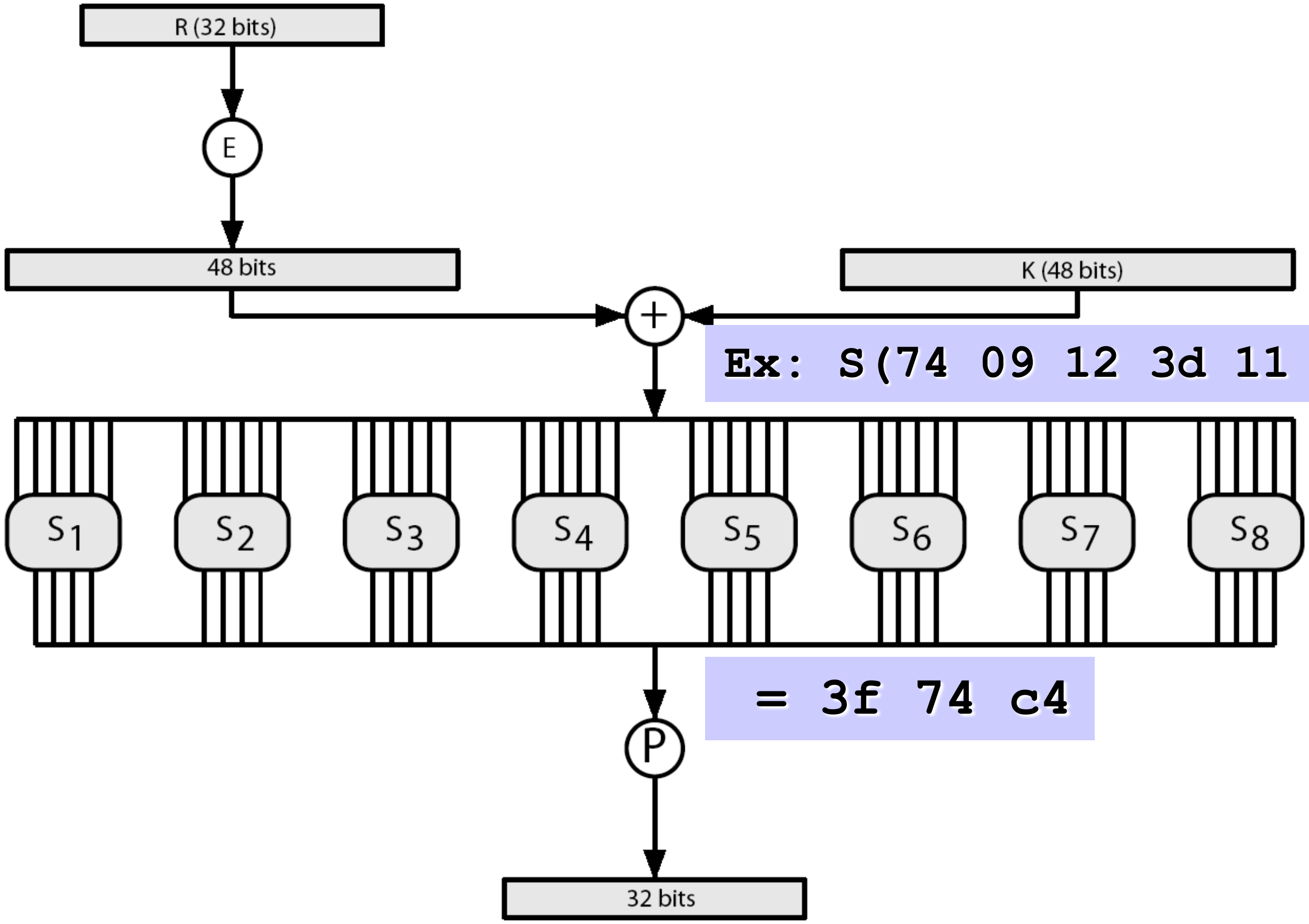
16	7	20	21	29	12	28	17
1	15	23	26	5	18	31	10
2	8	24	14	32	27	3	9
19	13	30	6	22	11	4	25



Substitution Boxes (for 16 rounds)

Ex:
011101

0011

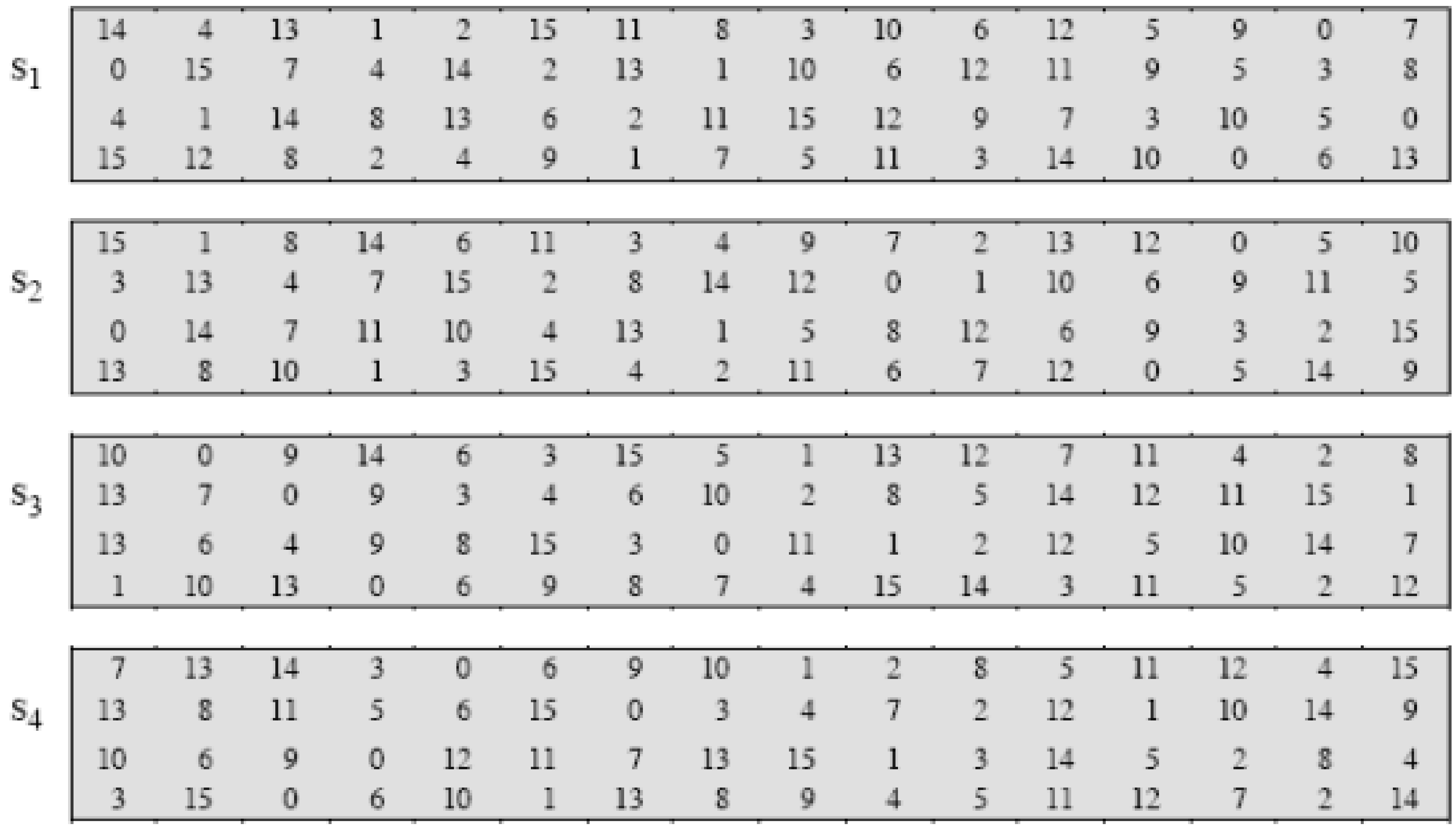


```
Ex: S(74 09 12 3d 11 17 )
```

```
= 3f 74 c4
```

S boxes (S1 to S4)

Valores de entrada: **0 (000000) a 63 (111111)**



S_1	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
S_2	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
S_3	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
S_4	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

Ex:
011101
$\downarrow$
0011

Note: values in each sBOX: 0 to 15 (0000 a 1111)
Ex: 8º sexteto MS **011101** → **01** (row 1) **1110** (column 14)
Then**: 011101** → 3, then output is **0011**

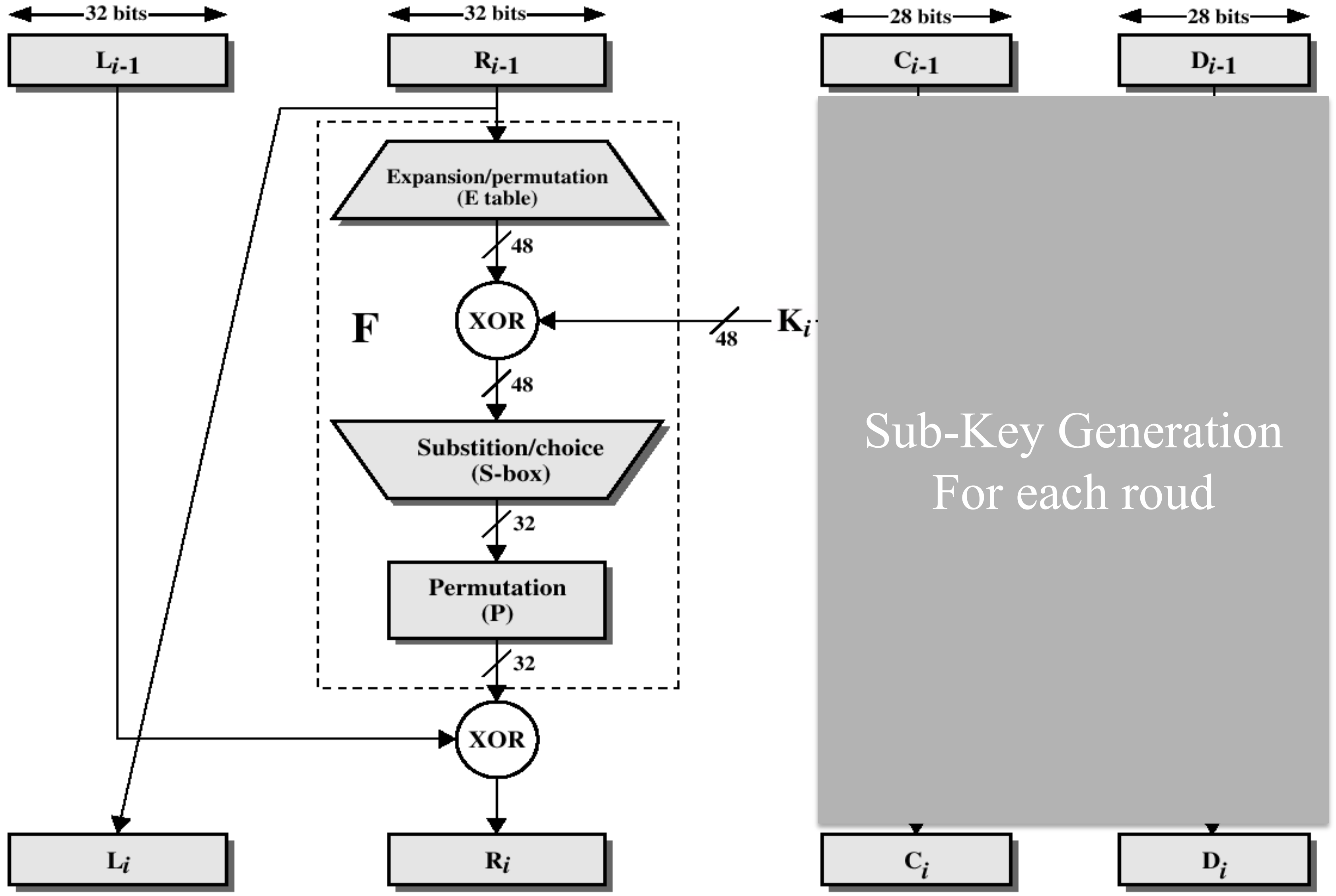
S boxes (S5 a S8)

S_5	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

S_6	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

S_7	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

S_8	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



Single Round of DES Algorithm

Sub-key generation

- A sub-key is generated in each round
- PC1: two halfs (28 bits) obtained by permutation
- In each round, each half is rotated for the next round

Input Key (56 bits)

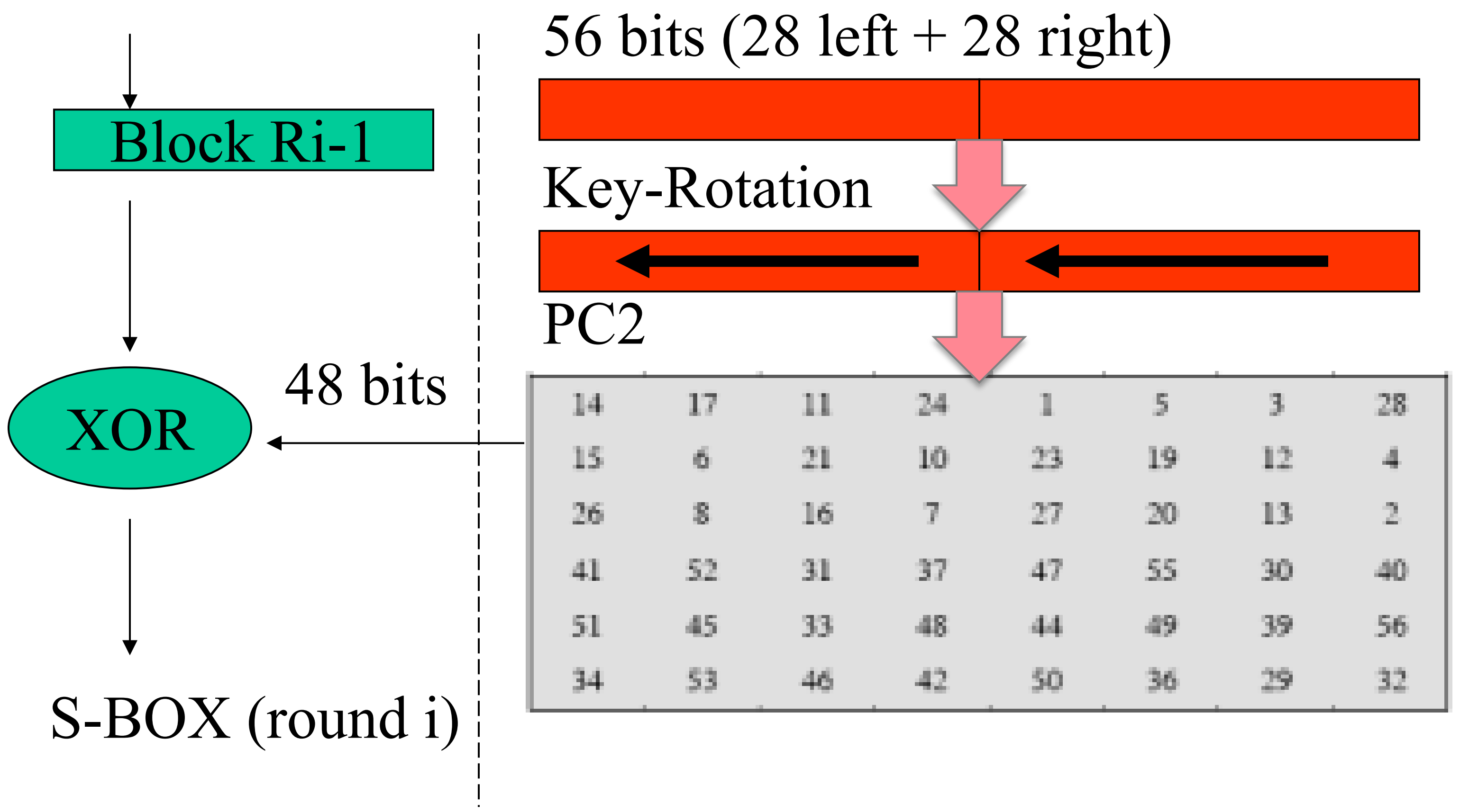
1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56
57	58	59	60	61	62	63	64

2 Initial Sub-Keys (28 bits)
(Initial Permutation PC1)

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

- 16 rotations (one per round): a specific rotation is defined for each round
- PC2: In each round, a permutation with reduction (56 > 48 bits)

PC2 sub-key rescheduling



14	17	11	24	1	5	3	28
15	6	21	10	23	19	12	4
26	8	16	7	27	20	13	2
41	52	31	37	47	55	30	40
51	45	33	48	44	49	39	56
34	53	46	42	50	36	29	32

Key-Rotation (for the next round)

Round number	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Bits rotated	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

Interesting problem … (Key Generation Quality)

- Know that the better is to start from randomly generated keys … (for any algorithm)
- But suppose that randomly we obtain the following keys:

```
11111111111111111111111111111111111111111111111111111111
00000000000000000000000000000000000000000000000000000000

11111111111111111111111111110000000000000000000000000000
00000000000000000000000000001111111111111111111111111111

Etc .. Other regularities (0-1 transitions) reducing the
entropy of the subkey-scheduling transformation
```

- Better: discarding of weak, semi-weak or possibly-weak keys
- So … We could have Key-PRGs and possible Weak-Keys Avoidance

  Is it relevant ?

DES Strengths and Security Analysis

- DES Standard (NIST DEA, FIPS46, 1977)

- From the algorithm viewpoint: probably one of the more studied algorithms in cryptanalysis – no one so far succeeded in discovering fatal weaknesses in DES (at least no one has credible publicly published acknowledged such a discovery)

- Form the keysize and blocksize viewpoints: 56 bits is to short !
  - Only possible 7.2 x 10^{16} keys
  - Brute-force attacks using one single computer (single PC):
    - 1 encrtption-decyprion / nanosecond => 2^{55} nanoseconds, 1125 years to break
  - But using 10^4 (parallelized) x PC computing power => 1 hour max. to break it for sure
  - EFF DES Cracking Contest (Special cracker machine) in 1998: US$ 250 K, 3 Days !
  - Off-the-Shelf Processors / Processor Cards
    - 10^9 DES operations/second currently in Multicore HW Processors
    - 10^{13} DES operations is reasonably expectable in the short term !

Symmetric Block Encryption Algorithms

Multiple Encryption and Onion-Encryption Constructions

DES and Multiple Encryption Methods

- Clear a replacement for DES was needed (short key size)
  - Theoretical attacks that can break it
  - Demonstrated exhaustive key search attacks
  - Cryptanalysis: demonstrated vulnerabilities
  - Weak, Semi-Weak and Potentially Weak keys

- AES Context: AES as the new cipher alternative
  - The "standard" symmetric encryption for the XXI century
  - Promoted by different organizations: NIST, ANSI, etc…

- Prior to new alternatives people start using multiple encryption with DES implementations
- Triple-DES is the chosen form
  - Compatibility, during the progressive adoption of AES

Double-DES ? No thanks...

- Could use 2 DES encrypts on each block

  $C = [E_{K2} (E_{K1}(P)]$

- Issue of reduction to single stage with single key (1992)

- ... and have "meet-in-the-middle" attack
  - Works whenever use a cipher twice
  - Since $X = E_{K1}(P) = D_{K2}(C)$
  - Attack by encrypting P with all keys and store
  - Then decrypt C with keys and match X value
  - Can show takes $O(2^{56})$ steps

Triple-DES with Two-Keys (DES-EDE-2)

- hence must use 3 encryptions
  - would seem to need 3 distinct keys

- but can use 2 keys with E-D-E sequence

  $C = E_{K1}(D_{K2}(E_{K1}(P)))$
  - Encrypt & decrypt equivalent in security
  - If K1=K2 then can work with single DES (compatibility)

- standardized in ANSI X9.17 & ISO8732
- Effective key length of 112 bits
- no current known practical attacks
  - But the use of 3 keys is better (improvement of the key-length effect)

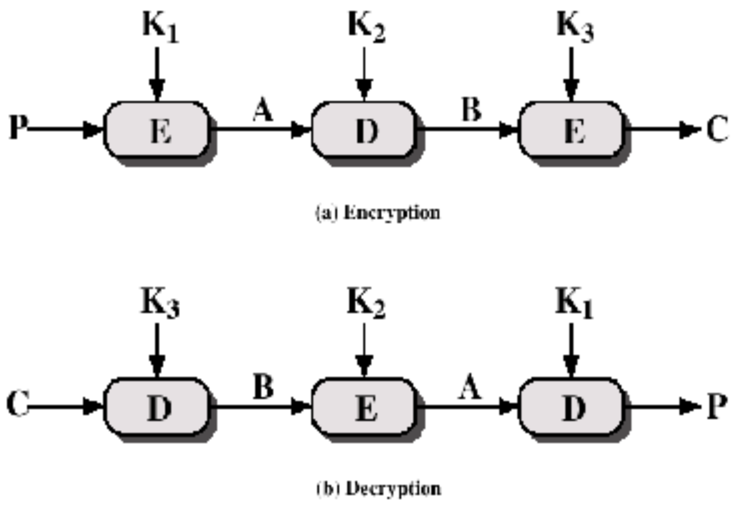
Triple DES with 3 keys (DES-EDE-3)

- Although are no practical attacks on two-key TripleDES have some indications
- can use Triple-DES with Three-Keys to avoid even these

  $C = [\ E_{K3}\ [\ D_{K2}\ (E_{K1}(P)\ ]\ ]$
- has been adopted by some Internet applications, eg PGP, S/MIME

- Effective key lenght: 168 bits

Paranoid ? Use N times DEA with N different keys
Problem ?



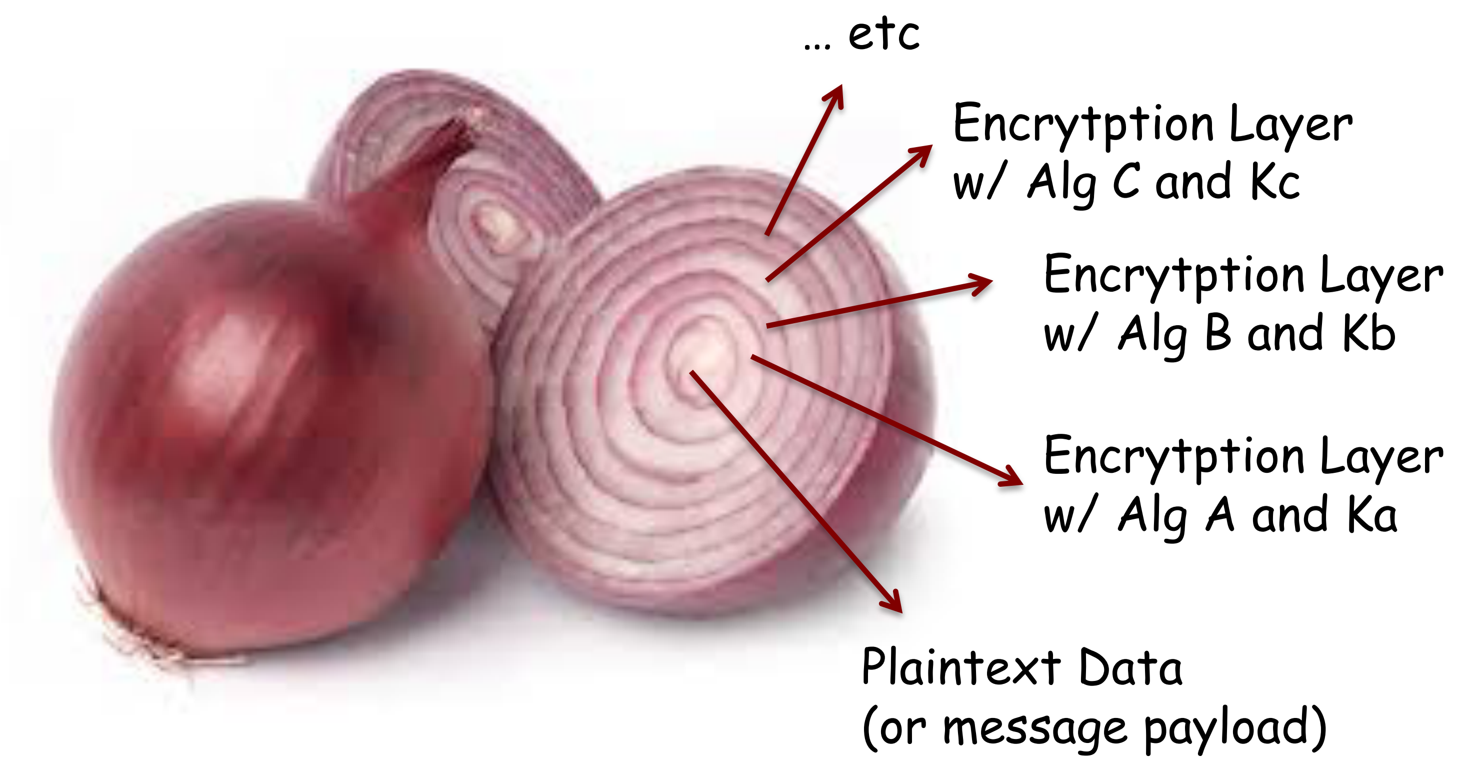
Note:
Triple or N-encryption-decryption can be applied to any block cipher algorithm

Other multiple encryption schemes
can be used in general for any Sym. Block Algorithm

- TEMK – Triple Encryption with Minimum Key
  Constants T1, T2 and T3 and Two initial Keys: K1, K2 to derive three keys: KX1, KX2, KX3 for triple encryption
  KXi = D(K2, E(K1,Ti))
- Use of Triple encryption modes (ex., Inner CBC, Outer CBC)
- Triple encryption with prefix-pads (pad size < block size)
- Doubling Block Length Space
- Ntuple Encryption: ex., n=5, 3 keys,
  C=Ek1(Dk2(Ek3(Dk4(Ek5(P)))))
- Witening technique: C= K3 XOR E(K2, (P XOR K1))
- Cascading multiple block ciphers: it is at least as hard to break as any of its component ciphers
- Combination technique
  - Random bit-string R with the same size as M
  - C = E(K1, R) XOR E(K2, (M XOR R))

Onion-Encryption Constructions or Schemes

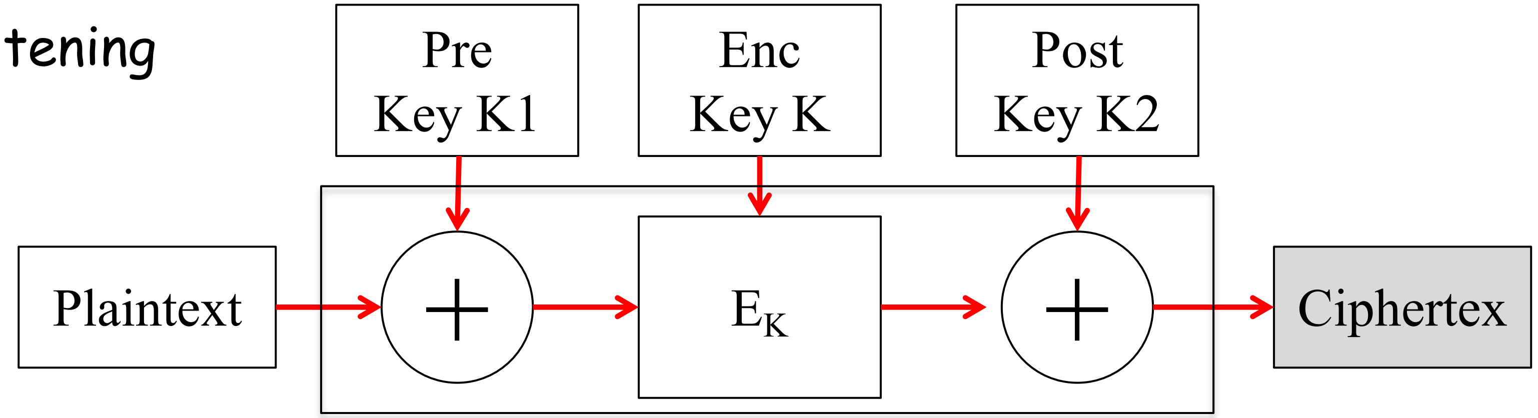
- More robustness combining different encryption algorithms in a multiple-encryption processing approch


... etc
Encrytption Layer w/ Alg C and Kc
Encrytption Layer w/ Alg B and Kb
Encrytption Layer w/ Alg A and Ka
Plaintext Data (or message payload)

Problems ?

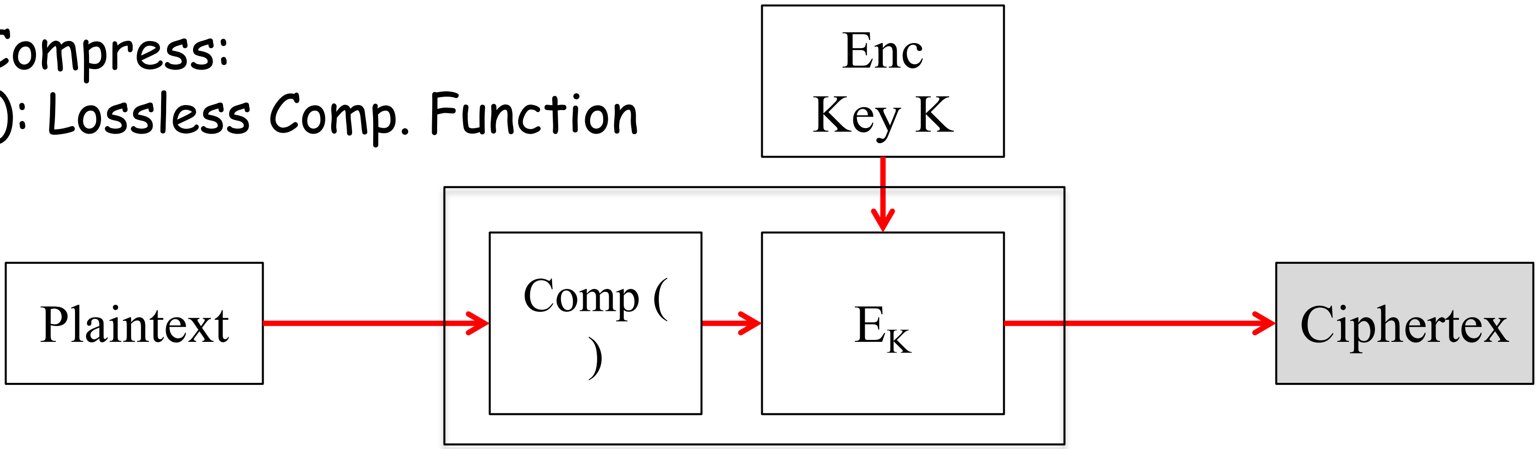
Other usual techniques: Whitening or Pre-Compressing

Whitening



Ci = K2 XOR E_k (Pi XOR K1)

Pre-Compress:
Com(): Lossless Comp. Function



Ci = E_k (Comp(Pi))

Symmetric Block Encryption Algorithms

Symmetric Block Encryption Algorithms

Other Symmetric Block Ciphers

- **International Data Encryption Algorithm (IDEA)**
  - 128-bit key
  - Used in PGP
- **Blowfish**
  - Easy to implement
  - High execution speed , Run in less than 5K of memory
- **Cast-128**
  - Key size from 40 to 128 bits
  - The round function differs from round to round
- **RC5**
  - Suitable for hardware and software
  - Fast, simple, Low memory requirement
  - Adaptable to processors of different word lengths
  - Variable number of rounds, Variable-length key
  - High security with Data-dependent rotations

Symmetric block cyphers and characteristics

Alg.	Key size	Rounds	Structure Operations	Applications (standadization)
DES	56	16	XOR, S-boxes, P-Boxes EP Boxes, Rotations, ...	SET, Kerberos
3DES	112/168	48	idem	Financial standards E-commerce, PGP S/MIME, SSL
IDEA	128	8	XOR, addition mod 2^16 ,Multiplication mod 2^16	PGP, SSL
TEA	128	32	XOR + shift	
Blowfish	< 448	16	XOR + Variable S-boxes	
RC5	< 2048	< 255	Addition, subtraction XOR and rotation operations	SSL
Cast-128	40-128	16	Addition, Subtraction, XOR, Rotation and S-boxes	PGP
AES	128, 192, 256	10,12,14	S boxes, Matrix SR-MC, XOR	Many: promoted as a standard after 2000

AES (Advanced Encryption Standard)

Origins and characteristics

- clear a replacement for DES was needed
  - have theoretical attacks that can break it
  - have demonstrated exhaustive key search attacks
- can use Triple-DES – but slow, has small blocks
- US NIST issued call for ciphers in 1997
- 15 candidates accepted (Jun/98), 5 were shortlisted in Aug/99
- Rijndael was selected as the AES in Oct-2000
- issued as FIPS PUB 197 standard in Nov-2001
- designed by Rijmen-Daemen in Belgium
- an **iterative** rather than **feistel** cipher
  - processes data as block of 4 columns of 4 bytes
  - operates on entire data block in every round
- designed to be:
  - resistant against known attacks
  - speed and code compactness on many CPUs
  - Simple to analyze

AES Requirements

NIST
Established the following reference criteria for proposals:

- Secret key / symmetric block cipher
- 128-bit data, 128/192/256-bit keys
- stronger & faster than Triple-DES
- active life of 20-30 years (+ archival use)
- provide full specification & design details
- both C & Java implementations
- NIST have released all submissions & unclassified analyses

AES structure

Not a Feistel structure …

- Data block of 4 columns of 4 bytes is state (128 bits)
- Key is expanded to array of words (44 words of 32 bits)

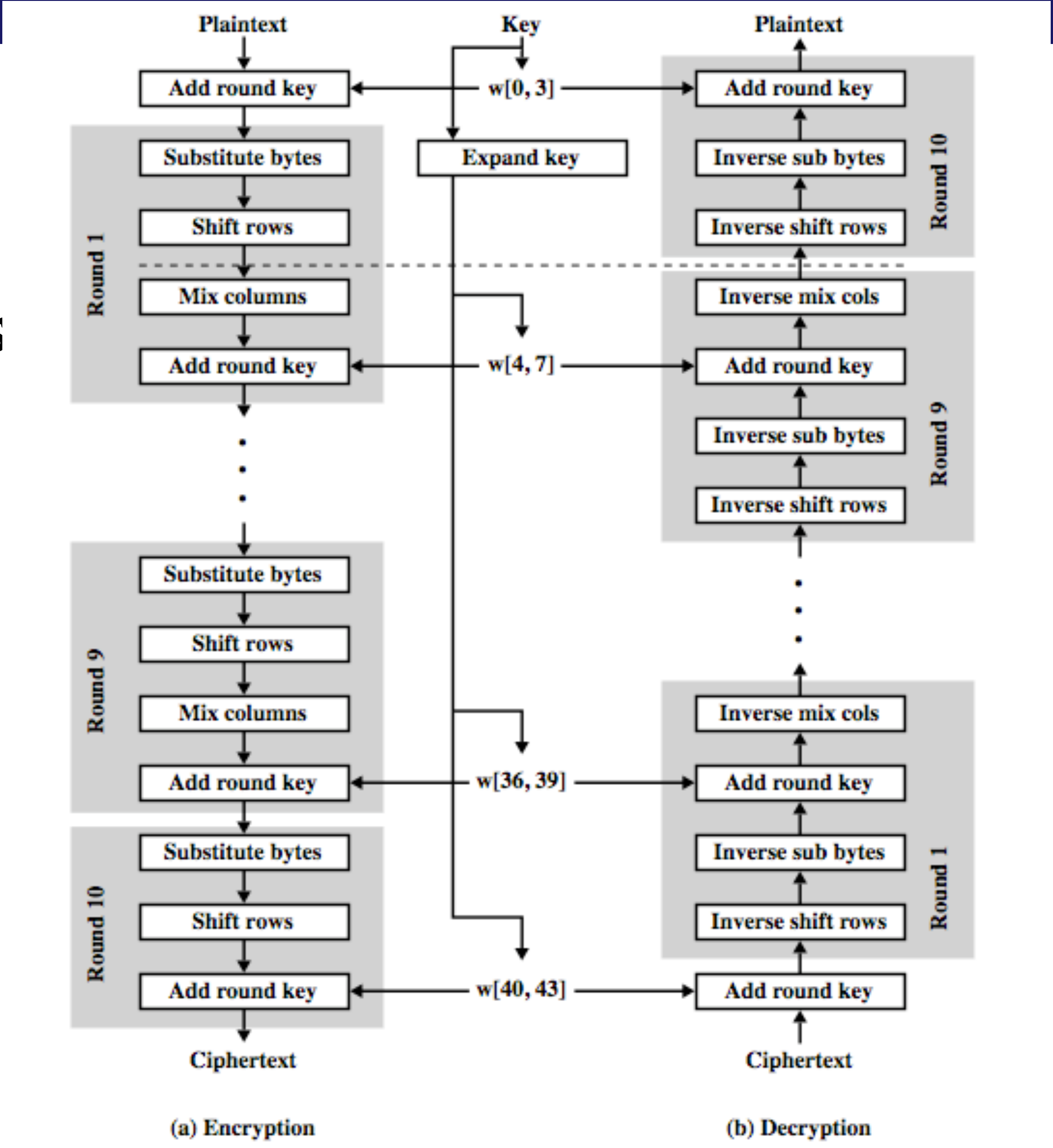
- Has 9/11/13 rounds in which state undergoes:
  - **I** : **byte substitution** (1 S-box used on every byte)
  - **II** : **shift rows** (permute bytes between groups/columns)
  - **III** : **mix columns** (subs using matrix multipy of groups)
  - **IV** : **add round key** (XOR state with key material)
  - view as alternating XOR key & scramble data bytes

- Initial XOR key material & incomplete last round
- With fast XOR & table lookup implementation

Advanced Encryption Algorithm (AES)

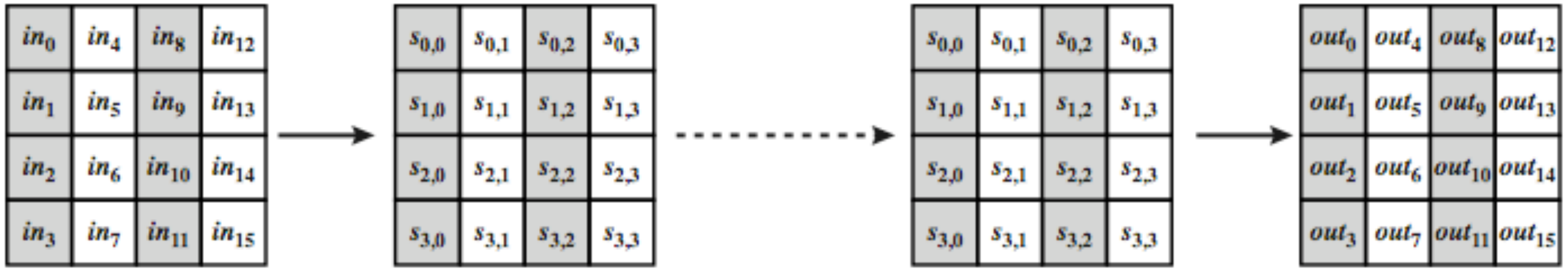
- Internal structure
- Ex., 10 rounds for 128 bit keys
- Successive processing of block-state arrays, combined with the expanded key

128 bit sub-key in each round



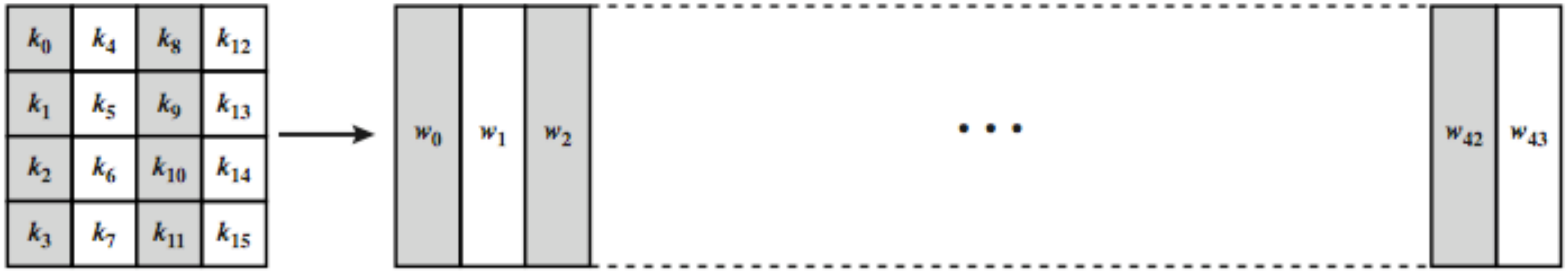
Input/Output State Array and Key Expansion

16 rounds of successive transformation of a 128 bit blocks (state arrays), organized as a square matrix of bytes (16x16 bits, ordered by column)



in_0 in_4 in_8 in_{12}
in_1 in_5 in_9 in_{13}
in_2 in_6 in_{10} in_{14}
in_3 in_7 in_{11} in_{15}
$s_{0,0}$ $s_{0,1}$ $s_{0,2}$ $s_{0,3}$
$s_{1,0}$ $s_{1,1}$ $s_{1,2}$ $s_{1,3}$
$s_{2,0}$ $s_{2,1}$ $s_{2,2}$ $s_{2,3}$
$s_{3,0}$ $s_{3,1}$ $s_{3,2}$ $s_{3,3}$
out_0 out_4 out_8 out_{12}
out_1 out_5 out_9 out_{13}
out_2 out_6 out_{10} out_{14}
out_3 out_7 out_{11} out_{15}

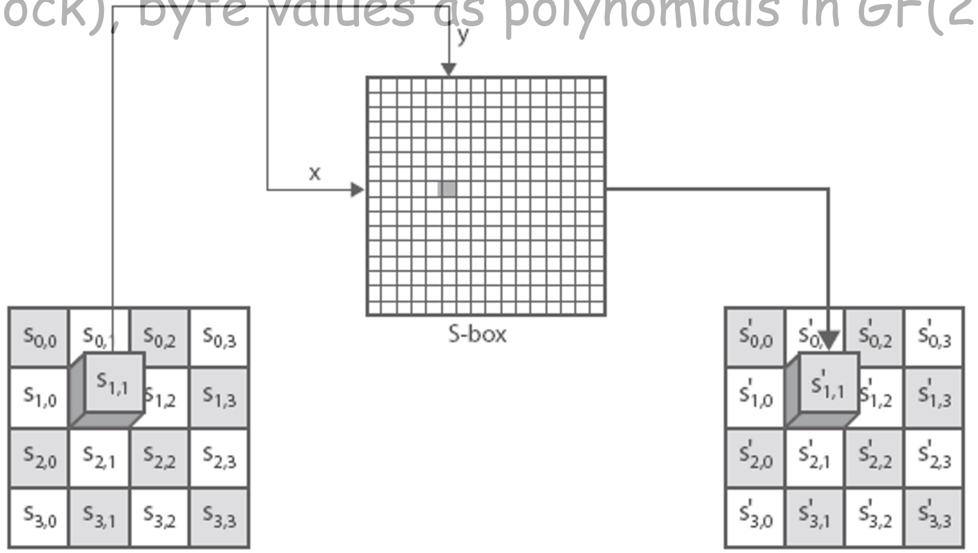
Key expanded from initial size (as a square matrix of bytes) to 44 columns (44 words, 4 bytes each)



k_0 k_4 k_8 k_{12}
k_1 k_5 k_9 k_{13}
k_2 k_6 k_{10} k_{14}
k_3 k_7 k_{11} k_{15}
w_0 w_1 w_2 • • • w_{42} w_{43}

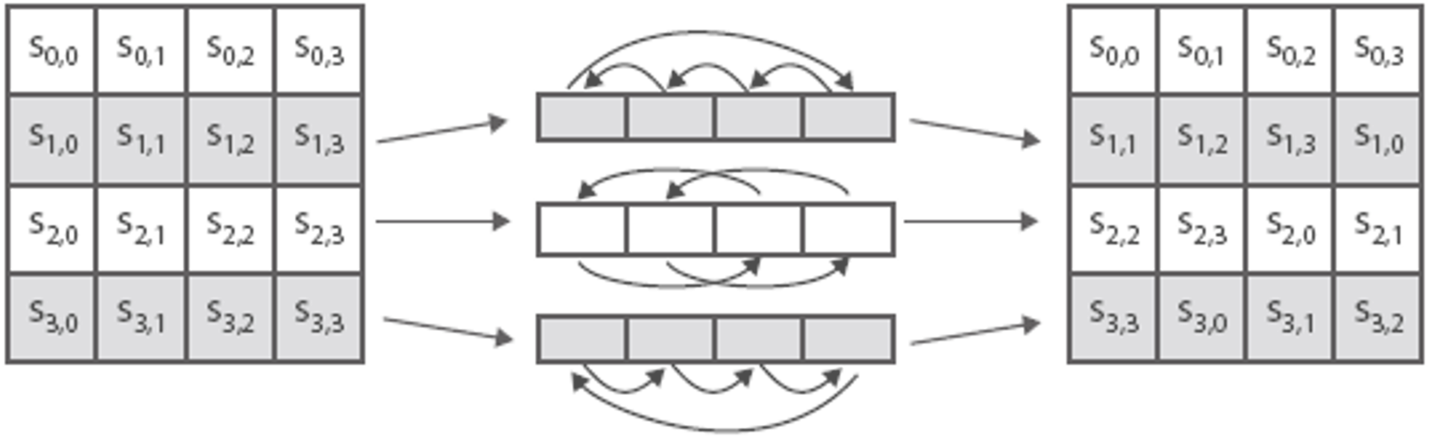
Byte Substitution

- A simple substitution of each byte (defined)
- Uses one table of 16x16 bytes containing a permutation of all 256 8-bit values
- Each byte of state is replaced by byte indexed by row (left 4-bits) & column (right 4-bits)
  - eg. byte {95} is replaced by byte in row 9 column 5
  - which has value {2A}
- S-box constructed using defined transformation of values (byte-to-byte substitution of the block), byte values as polynomials in GF(2^8)
- designed to be resistant to all known attacks


y
x
S-box
$s_{0,0}$ $s_{0,1}$ $s_{0,2}$ $s_{0,3}$
$s_{1,0}$ $s_{1,1}$ $s_{1,2}$ $s_{1,3}$
$s_{2,0}$ $s_{2,1}$ $s_{2,2}$ $s_{2,3}$
$s_{3,0}$ $s_{3,1}$ $s_{3,2}$ $s_{3,3}$
$s'_{0,0}$ $s'_{0,1}$ $s'_{0,2}$ $s'_{0,3}$
$s'_{1,0}$ $s'_{1,1}$ $s'_{1,2}$ $s'_{1,3}$
$s'_{2,0}$ $s'_{2,1}$ $s'_{2,2}$ $s'_{2,3}$
$s'_{3,0}$ $s'_{3,1}$ $s'_{3,2}$ $s'_{3,3}$

Shift Rows

- a circular byte shift in each each
  - 1st row is unchanged
  - 2nd row does 1 byte circular shift to left
  - 3rd row does 2 byte circular shift to left
  - 4th row does 3 byte circular shift to left
- decrypt inverts using shifts to right
- since state is processed by columns, this step permutes bytes between the columns

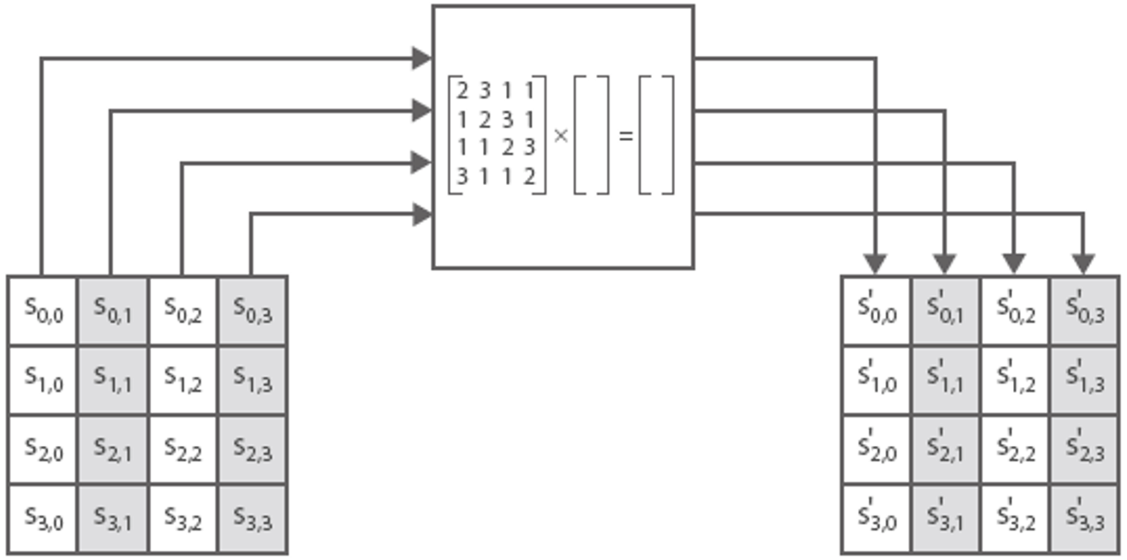


$s_{0,0}$ $s_{0,1}$ $s_{0,2}$ $s_{0,3}$
$s_{1,0}$ $s_{1,1}$ $s_{1,2}$ $s_{1,3}$
$s_{2,0}$ $s_{2,1}$ $s_{2,2}$ $s_{2,3}$
$s_{3,0}$ $s_{3,1}$ $s_{3,2}$ $s_{3,3}$
$s_{0,0}$ $s_{0,1}$ $s_{0,2}$ $s_{0,3}$
$s_{1,1}$ $s_{1,2}$ $s_{1,3}$ $s_{1,0}$
$s_{2,2}$ $s_{2,3}$ $s_{2,0}$ $s_{2,1}$
$s_{3,3}$ $s_{3,0}$ $s_{3,1}$ $s_{3,2}$

Mix Columns

- each column is processed separately / each byte is replaced by a value dependent on all 4 bytes in the column
- effectively a matrix multiplication in GF(2^8) using prime poly m(x) =$x^8+x^4+x^3+x+1$

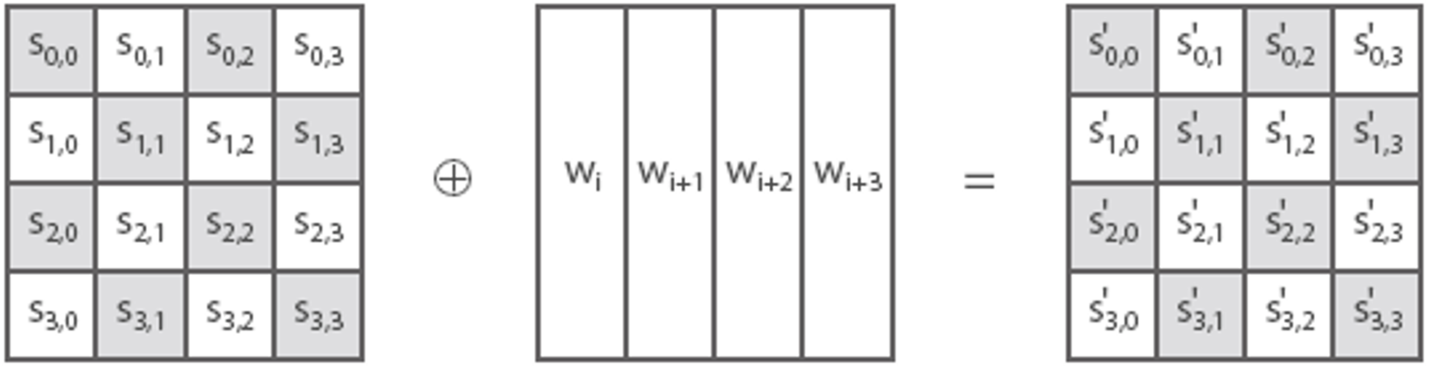
$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} = \begin{bmatrix} s'_{0,0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\ s'_{1,0} & s'_{1,1} & s'_{1,2} & s'_{1,3} \\ s'_{2,0} & s'_{2,1} & s'_{2,2} & s'_{2,3} \\ s'_{3,0} & s'_{3,1} & s'_{3,2} & s'_{3,3} \end{bmatrix}$$

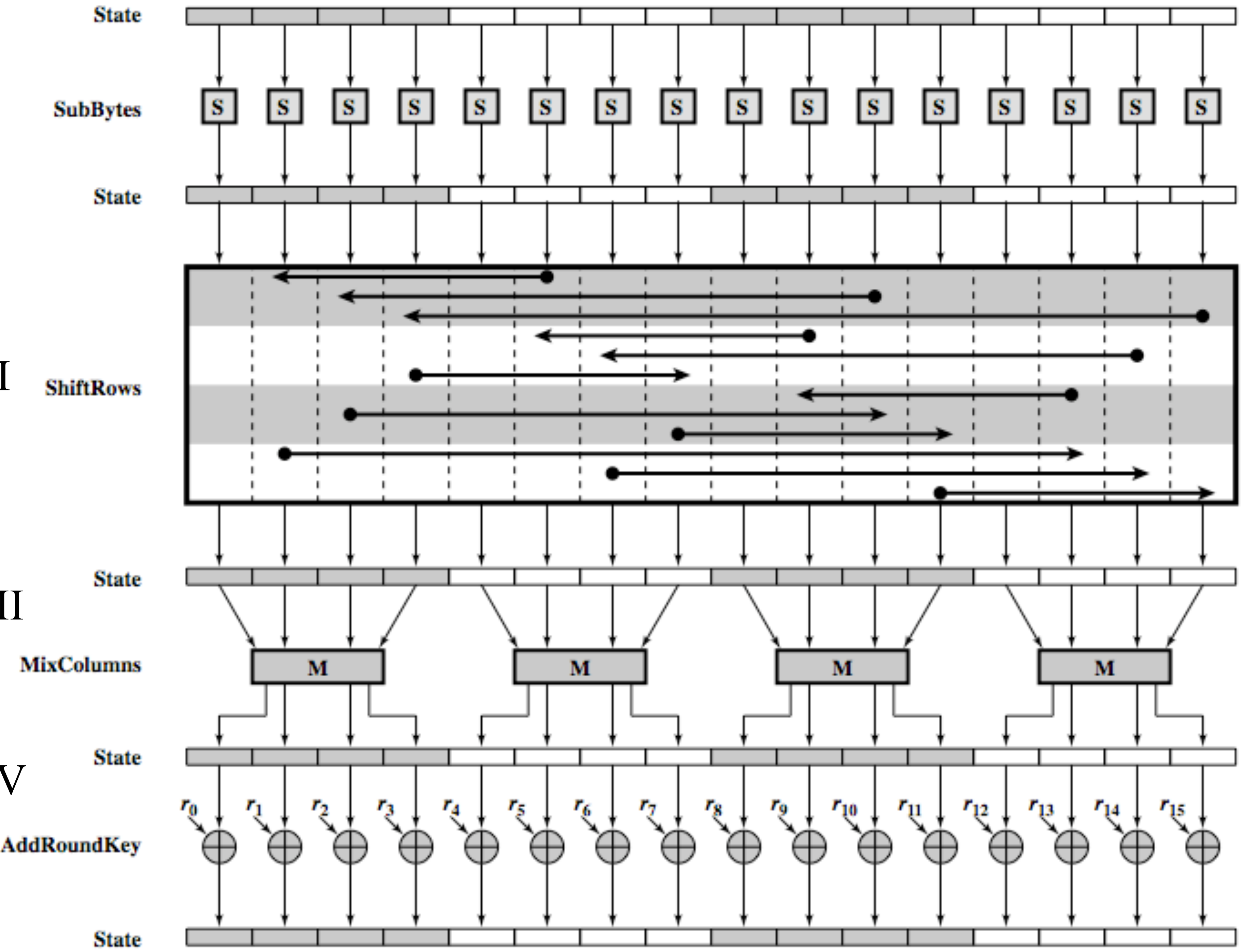


$\begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \times \begin{bmatrix} \\ \end{bmatrix} = \begin{bmatrix} \\ \end{bmatrix}$
$s_{0,0}$ $s_{0,1}$ $s_{0,2}$ $s_{0,3}$
$s_{1,0}$ $s_{1,1}$ $s_{1,2}$ $s_{1,3}$
$s_{2,0}$ $s_{2,1}$ $s_{2,2}$ $s_{2,3}$
$s_{3,0}$ $s_{3,1}$ $s_{3,2}$ $s_{3,3}$
$s'_{0,0}$ $s'_{0,1}$ $s'_{0,2}$ $s'_{0,3}$
$s'_{1,0}$ $s'_{1,1}$ $s'_{1,2}$ $s'_{1,3}$
$s'_{2,0}$ $s'_{2,1}$ $s'_{2,2}$ $s'_{2,3}$
$s'_{3,0}$ $s'_{3,1}$ $s'_{3,2}$ $s'_{3,3}$

Add Round Key

- XOR state with 128-bits of the round key
- again processed by column (though effectively a series of byte operations)
- inverse for decryption identical
  - since XOR own inverse, with reversed keys
- designed to be as simple as possible
  - a form of Vernam cipher on expanded key
  - requires other stages for complexity / security


$s_{0,0}$ $s_{0,1}$ $s_{0,2}$ $s_{0,3}$
$s_{1,0}$ $s_{1,1}$ $s_{1,2}$ $s_{1,3}$
$s_{2,0}$ $s_{2,1}$ $s_{2,2}$ $s_{2,3}$
$s_{3,0}$ $s_{3,1}$ $s_{3,2}$ $s_{3,3}$
$\oplus$
w_i w_{i+1} w_{i+2} w_{i+3}
$=$
$s'_{0,0}$ $s'_{0,1}$ $s'_{0,2}$ $s'_{0,3}$
$s'_{1,0}$ $s'_{1,1}$ $s'_{1,2}$ $s'_{1,3}$
$s'_{2,0}$ $s'_{2,1}$ $s'_{2,2}$ $s'_{2,3}$
$s'_{3,0}$ $s'_{3,1}$ $s'_{3,2}$ $s'_{3,3}$


State
SubBytes
S
State
I
ShiftRows
State
II
MixColumns
M
State
V
r_0 r_1 r_2 r_3 r_4 r_5 r_6 r_7 r_8 r_9 r_{10} r_{11} r_{12} r_{13} r_{14} r_{15}
AddRoundKey
State