Symmetric Ciphers

Segurança Informática em Redes e Sistemas
2024/25

David R. Matos, Ricardo Chaves

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

Roadmap

- Introduction
- Symmetric Ciphers
- Hash Functions
- Message Integrity Codes

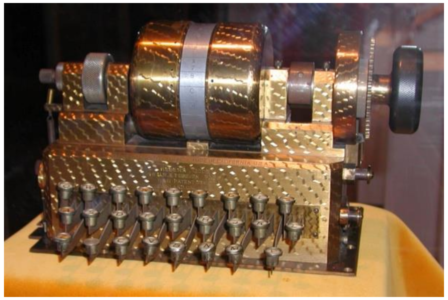
Ciphers: terminology

- Cipher
  - Specific cryptographic technique
- Cipher Procedure
  - Cipher: plaintext → cryptogram (aka ciphertext)
  - Decipher: cryptogram → plaintext

  - Algorithm: data transformation procedure
  - Key: algorithm parameter

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

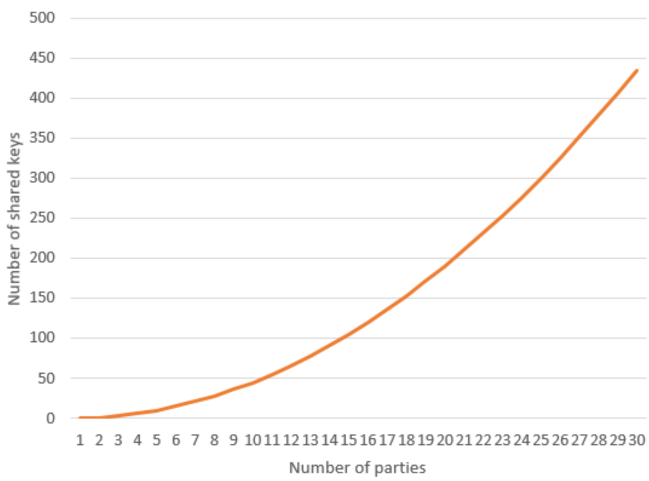
- Introduction
- **Symmetric Ciphers**
- Hash Functions
- Message Integrity Codes

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 x (N-1)/2 keys
- Problems
  - Key distribution

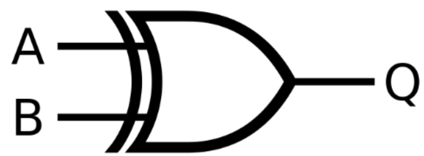


Perfectly Secure Cipher: One-Time Pad

- 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:

    $C_1 = P_1 \oplus K$ and $C_2 = P_2 \oplus K$

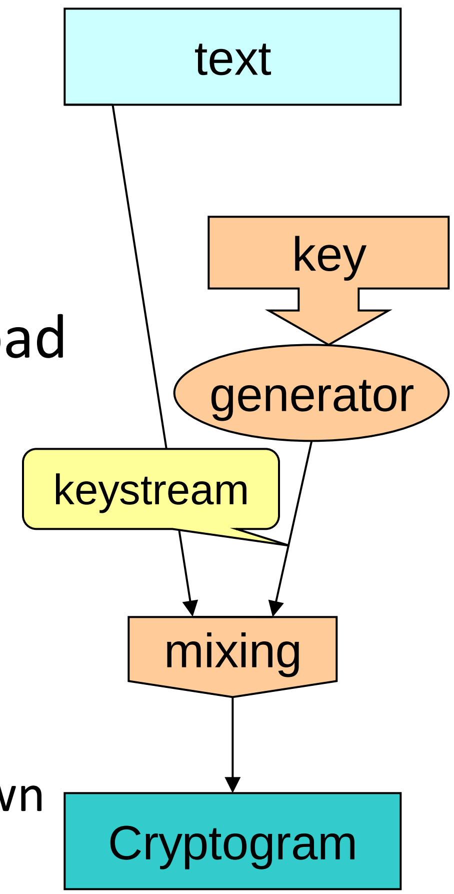
    $$C_1 \oplus C_2 = P_1 \oplus K \oplus P_2 \oplus K$$
    $$= P_1 \oplus P_2$$

- 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

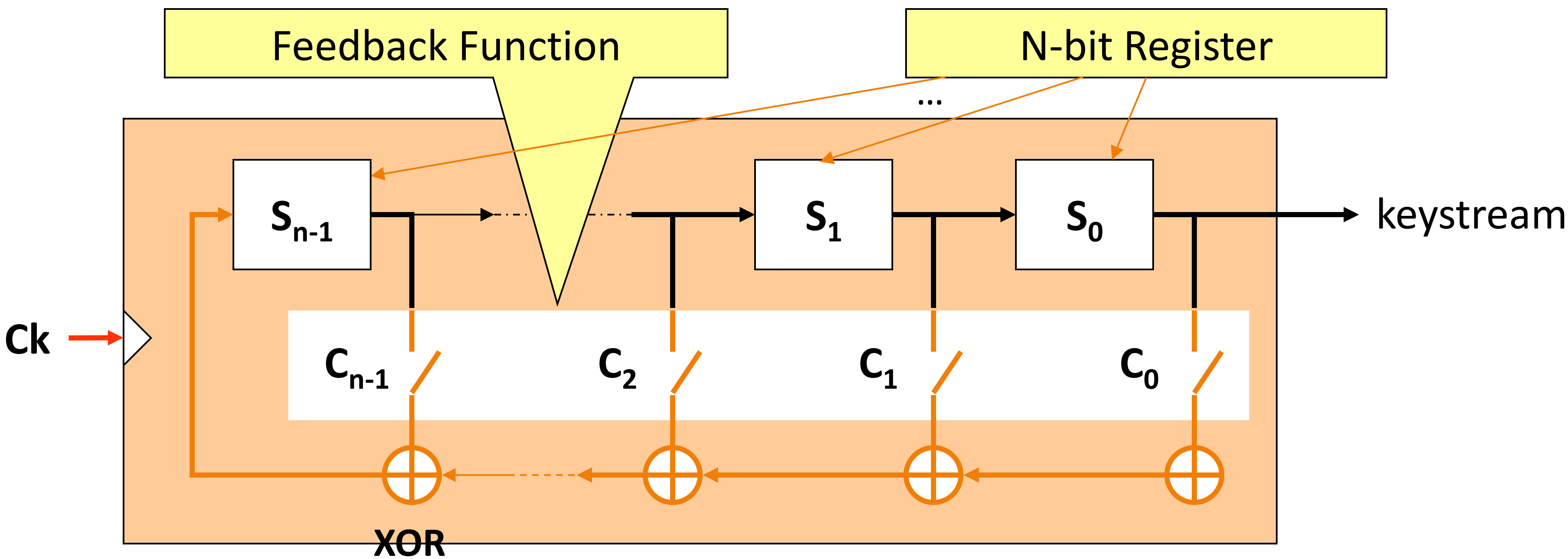


Symmetric stream ciphers

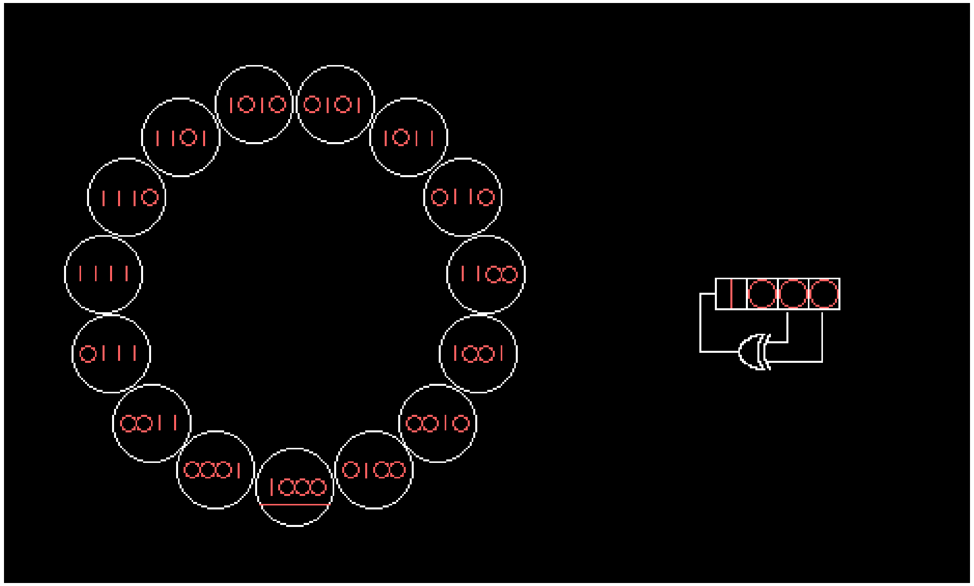
- Used approximations
  - Secure pseudo-random generators
    - Based on LFSRs (Linear Feedback Shift Registers) → next
    - Based on block ciphers → later
    - Other approximations (nonlinear functions, etc.)
  - Usually without self-synchronization
    - Receiver must know when encrypted data begins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

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
  - $S_0, \ldots, S_{n-1}$ = **register**'s bits; $C_0, \ldots, C_{n-1}$ = **coefficients** of the function
  - Max. period of the cycle is 2^n-1



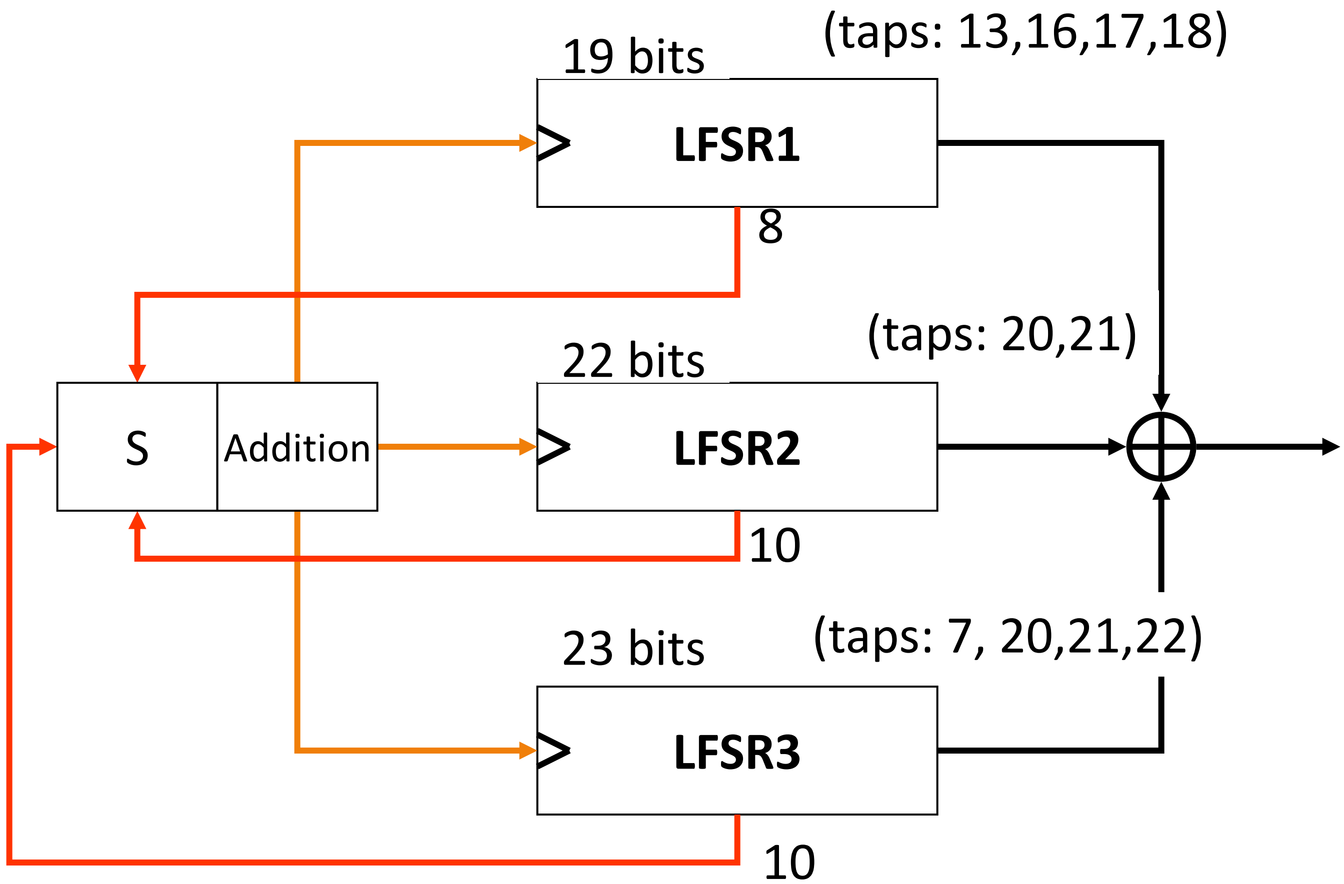
4 bit LFSR example



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 (IDEA, Blowfish, CAST, RC5, etc.)

DES Avalanche

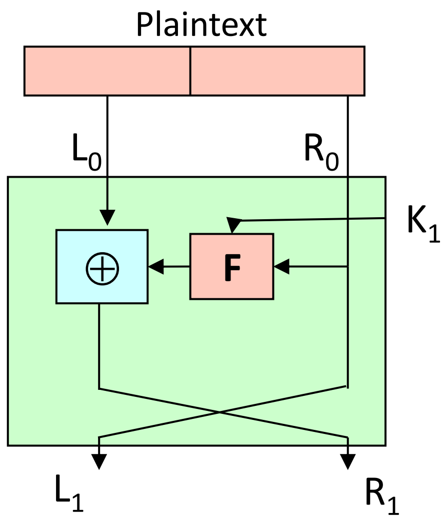
```
Input:     ...............................................................*     1
Permuted:  .......................................*........................     1
Round 1:   .......*........................................................     1
Round 2:   .*..*...*.....*........................*........................     5
Round 3:   .*..*.*.**..*.*.*.*....**.....**.*..*...*.....*.................    18
Round 4:   ..*.*****.*.*****.*.*......*.....*..*.*.**..*.*.*.*....**.....**    28
Round 5:   *...**..*.*...*.*.*.*...*.***..*..*.*****.*.*****.*.*......*....    29
Round 6:   ...*..**.....*.*..**.*.**...*..**...**..*.*...*.*.*.*...*.***..*    26
Round 7:   *****...***....**...*..*.*..*......*..**.....*.*..**.*.**...*..*
Round 8:   *.*.*.*.**.....*.*.*...**.*...*******...***....**...*..*.*..*...
Round 9:   ***.*.***...**.*.****.....**.*..*.*.*.*.**.....*.*.*...**.*...**
Round 10:  *.*..*.*.**.*..*.**.***.**.*...****.*.***...**.*.****.....**.*..
Round 11:  ..******......*..******....*....*.*..*.*.**.*..*.**.***.**.*...*
Round 12:  *..***....*...*.*.*.***...****....******......*..******....*....
Round 13:  **..*....*..******...*........*.*..***....*...*.*.*.***...****..
Round 14:  *.**.*....*.*....**.*...*..**.****..*....*..******...*........*.
Round 15:  **.*....*.*.*...*.**.*..*.*.**.**.**.*....*.*....**.*...*..**.**
Round 16:  .*..*.*..*..*.**....**..*..*..****.*....*.*.*...*.**.*..*.*.**.*
Output:    ..*..**.*.*...*....***..***.**.*...*..*..*.*.*.**.*....*.*.*.**.  ~50%
```

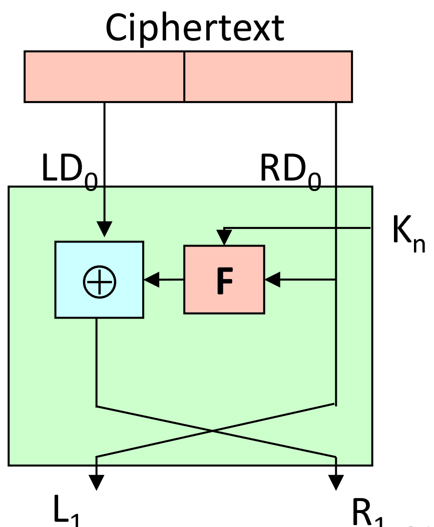
* is a changed bit that propagates changes

Feistel Network

- Complex function most commonly used in block cipher algorithms

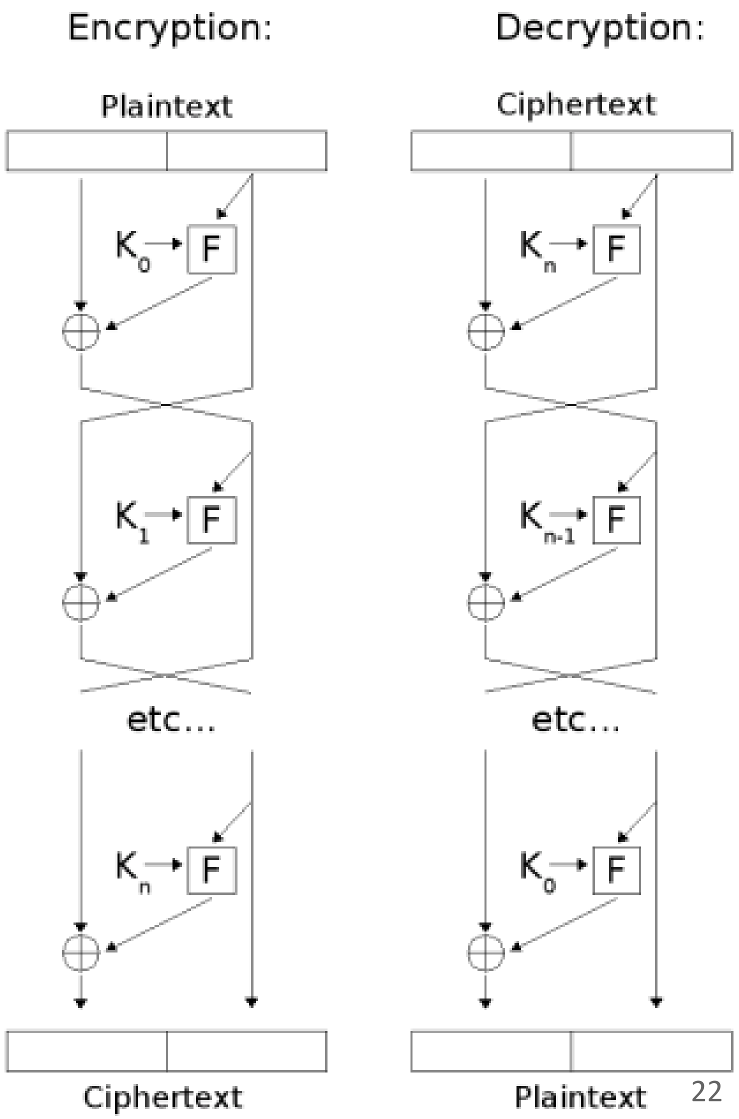
- Applies a round function (F) over multiple rounds
  - F can be a pseudo-random generator
  - Each round uses a different round key (K_i)
  - Round keys are obtained from the key (K)
  - Text is split in left (L) and right (R) parts





Feistel Network

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



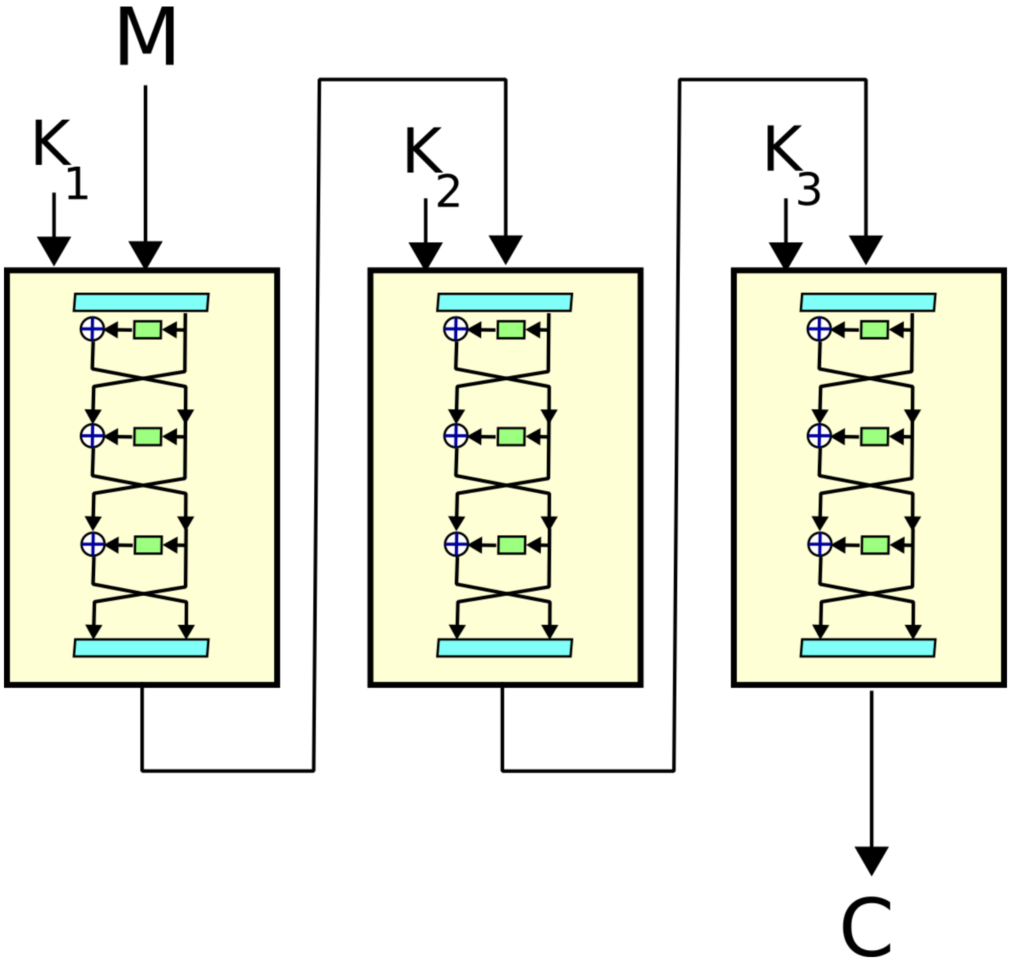
DES Algorithm

- DES:
  - 64-bit blocks
  - Key is 56 bits
- Possible key combinations
  - 2^{56} = 7.2 x 1016 = 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
    - $C_i = E_{K1}(D_{K2}(E_{K3}(P_i)))$
    - $P_i = D_{K3}(E_{K2}(D_{K1}(C_i))$
    - Typically, $K_1=K_3$ is used
    - Effective key size = 56 + 56 bits = 112 bits
- Key whitening (DESX)
    - DES-X$(P_i) = K_2 \oplus DES_K(K_1 \oplus P_i)$
      - $C_i = E_K(K_1 \oplus P_i) \oplus K_2$
      - $P_i = K_1 \oplus D_K(K_2 \oplus C_i)$

3DES – Triple DES / TDEA – Triple Data Encryption Algorithm



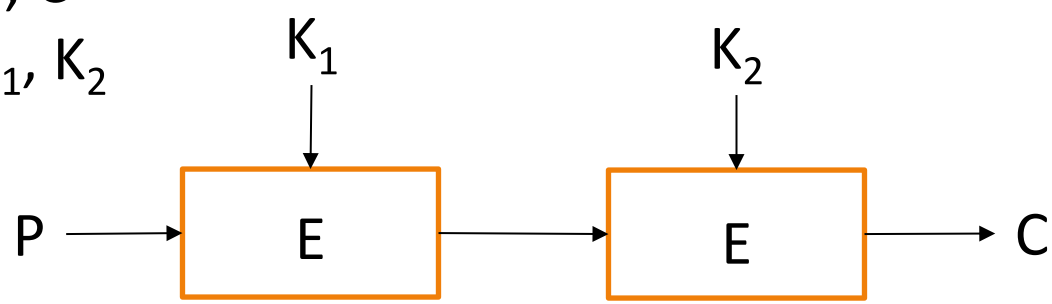
Meet-in-the-middle attack

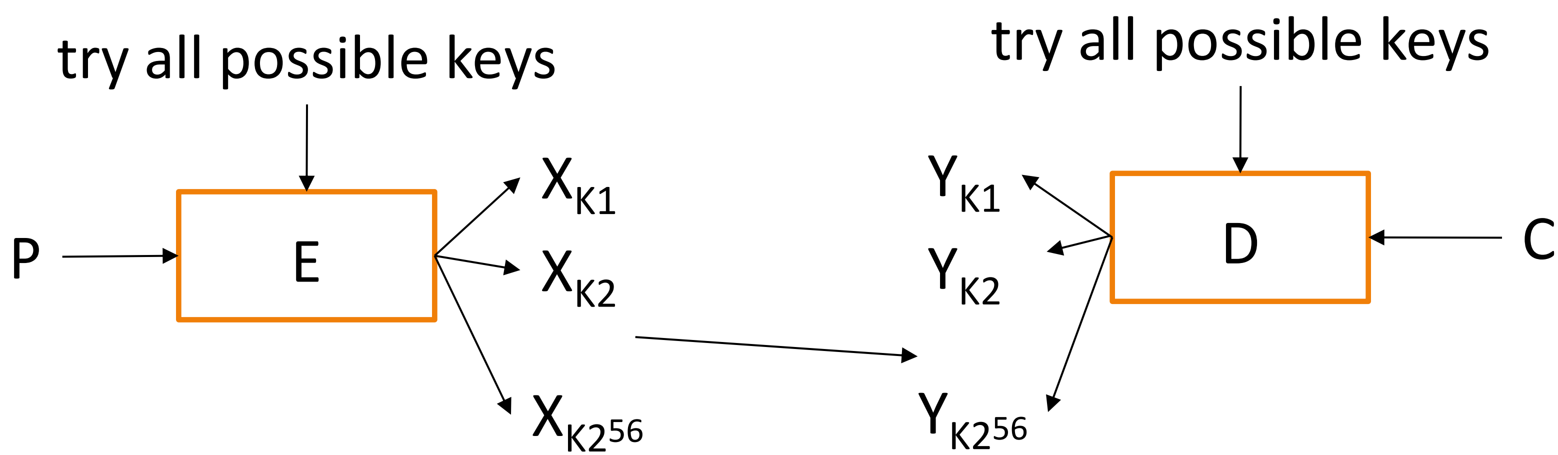
- Double DES
  - $C = E_{K2}(E_{K1}(P))$
  - Effective key size of Double DES?

    $= 2^{56} * 2^{56} = 2^{112}$ WRONG!

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

Attacker knows P, C
does not know K_1, K_2





One $X_{Ki} = Y_{Kj}$ 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
  - $C = E_{K2} (E_{K1} (P))$
  - Effective key size of Double DES?

    $= 2^{56} * 2^{56} = 2^{112}$ WRONG!

- Meet-in-the-Middle Attack
  - $C = E_{K2} (E_{K1} (P))$
  - $X = E_{K1} (P) = D_{K2} (C)$
  - Brute force attack (given one P/C pair):
    1. calculate EK1 (P) for all keys (2^{56} work)
    2. calculate DK2 (C) for all keys (2^{56} work)
    3. the match gives the keys
  - Total work $= 2 * 2^{56} = 2^{57}$

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)

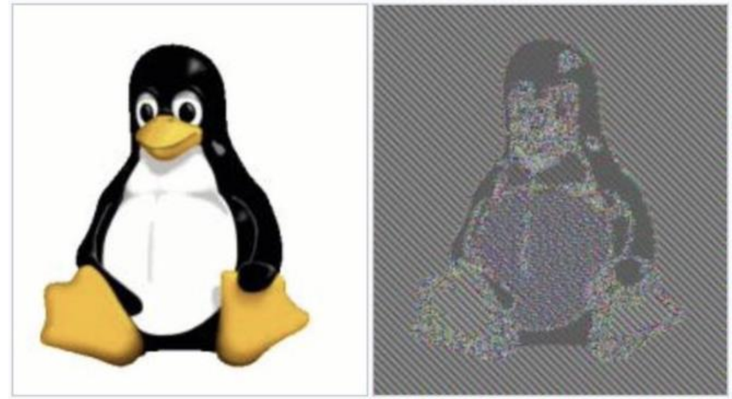
- Final sub-block processing
  - Alignment with padding
    - Augments the cryptogram by adding padding to the plaintext

Block cipher modes: ECB

ECB (Electronic Code Book)

$C_i = E_k(P_i)$

$P_i = D_k(C_i)$

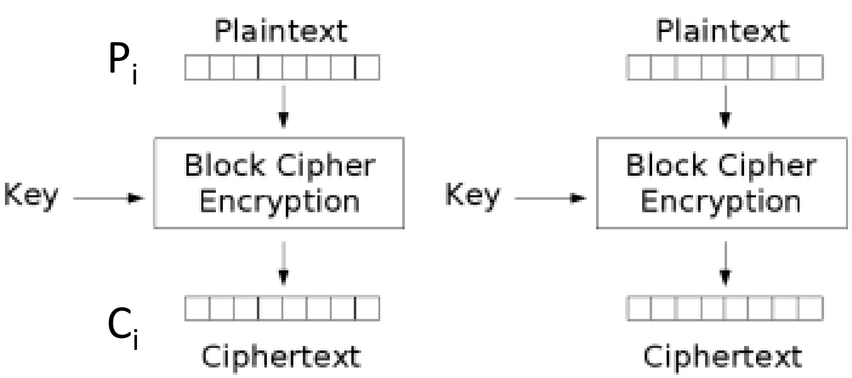


Original image

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



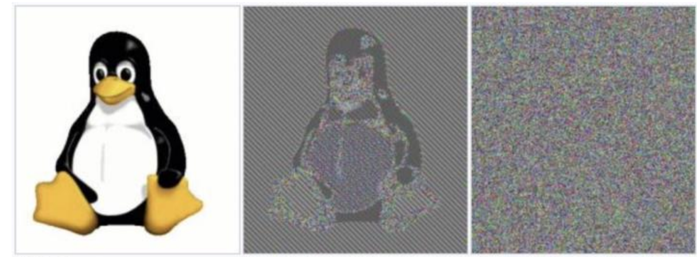
Block cipher modes: CBC

CBC (Cipher-Block Chaining)

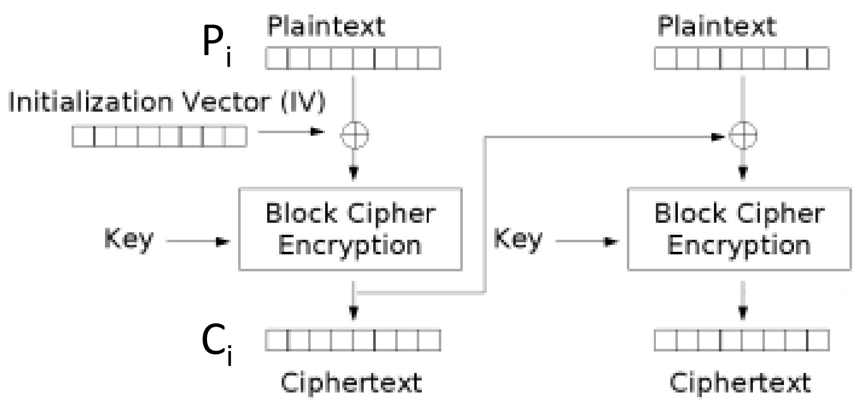
$C_i = E_k(P_i \oplus C_{i-1})$

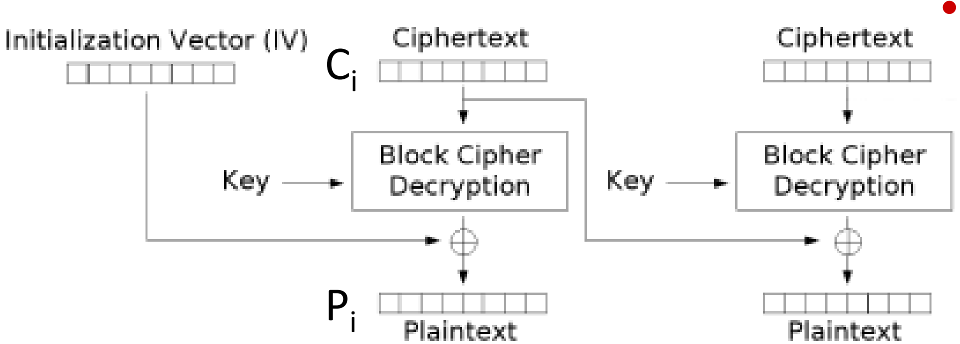
$P_i = C_{i-1} \oplus D_k(C_i)$

$C_0 = IV$



Original image | Encrypted using ECB mode | Modes other than ECB result in pseudo-randomness





- 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)

$C_i = P_i \oplus O_i$
$P_i = P_i \oplus O_i$

Transforms block cipher into stream cipher

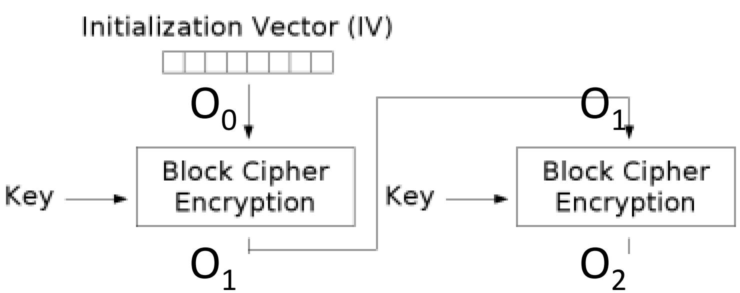
$O_i = E_k(O_{i-1})$
$O_0 = IV$

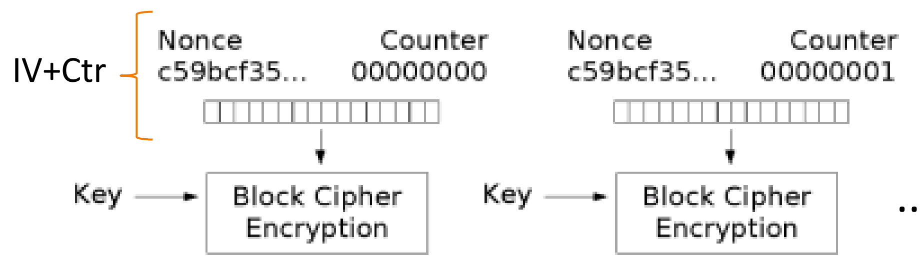
CTR (CounTer) mode *or* ICM (Integer Counter Mode)

$C_i = P_i \oplus E_k(IV+Ctr_i)$
$P_i = C_i \oplus E_k(IV+Ctr_i)$

Transforms block cipher into stream cipher

$Ctr_i = Ctr_i + 1$





Block cipher modes: OFB / CTR

OFB (Output FeedBack)

$C_i = P_i \oplus O_i$

$P_i = P_i \oplus O_i$

Transforms block cipher into stream cipher

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

$O_0 = IV$

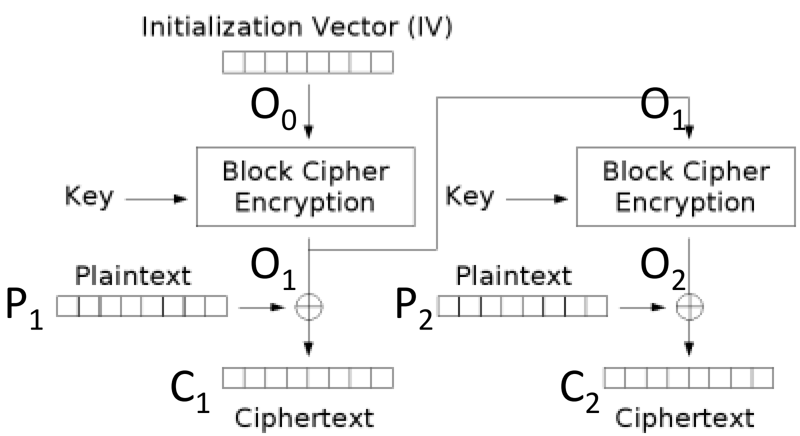
CTR (CounTer) mode *or* ICM (Integer Counter Mode)

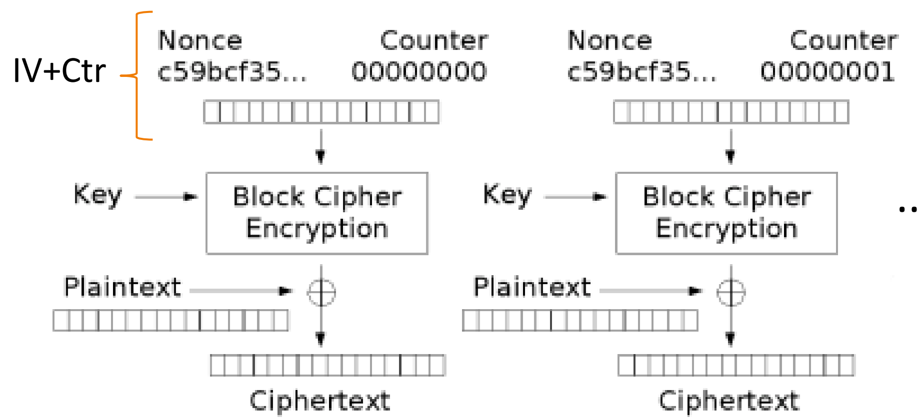
$C_i = P_i \oplus E_k(IV+Ctr_i)$

$P_i = C_i \oplus E_k(IV+Ctr_i)$

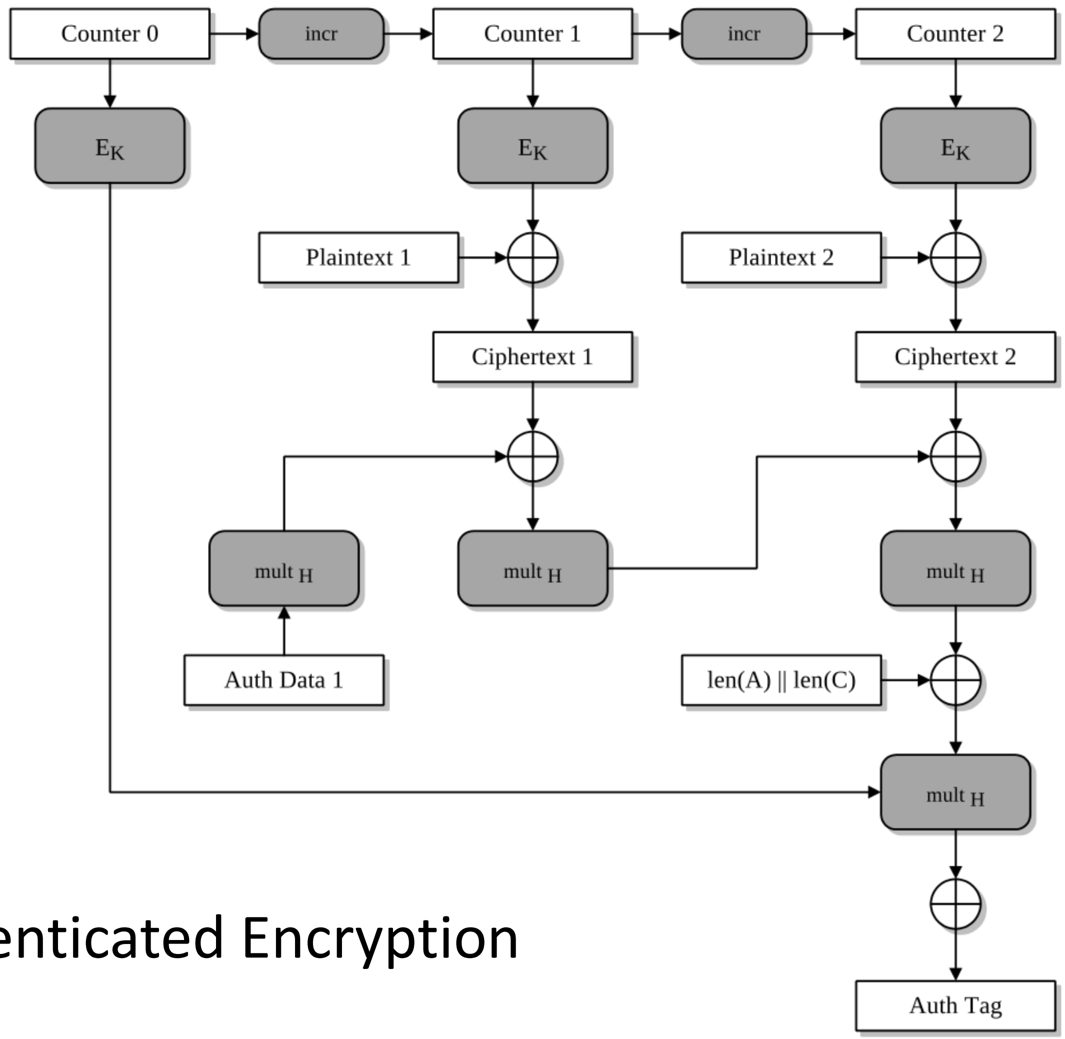
Transforms block cipher into stream cipher

$Ctr_i = Ctr_i + 1$





Block cipher modes: GCM
Galois Counter Mode



Provides Confidentiality

and Integrity

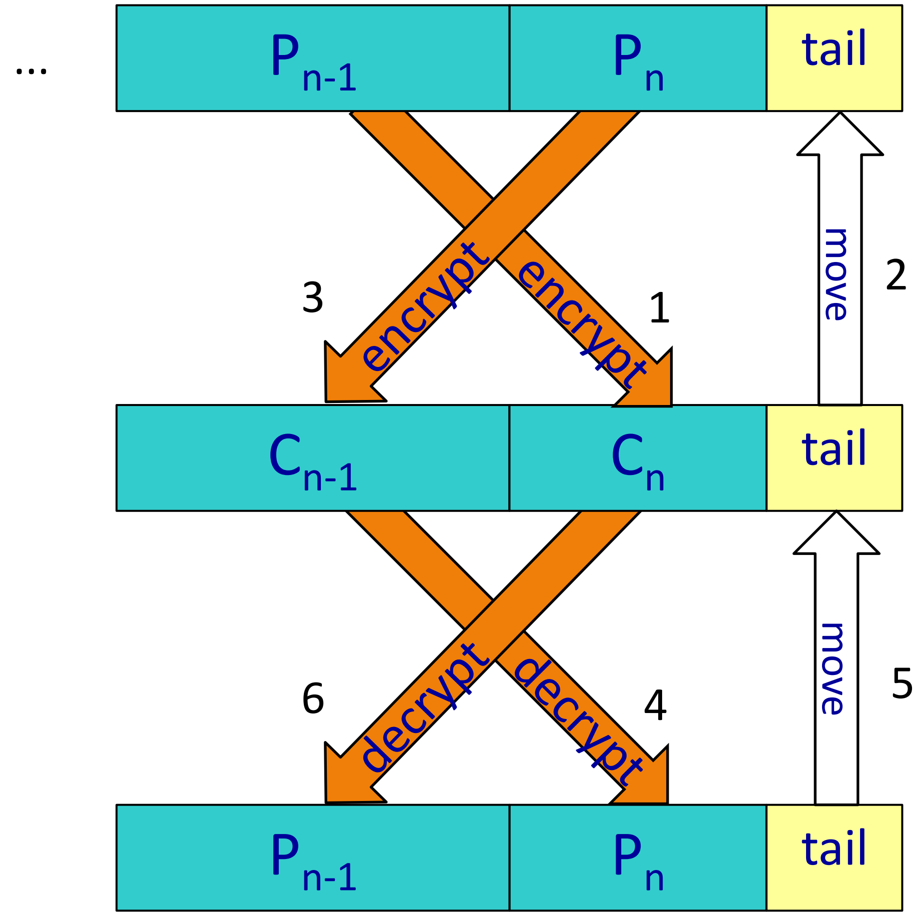
AE: Authenticated Encryption

Padding Schemes

- $M \bmod B \neq 0$ => **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!



AES – current encryption standard

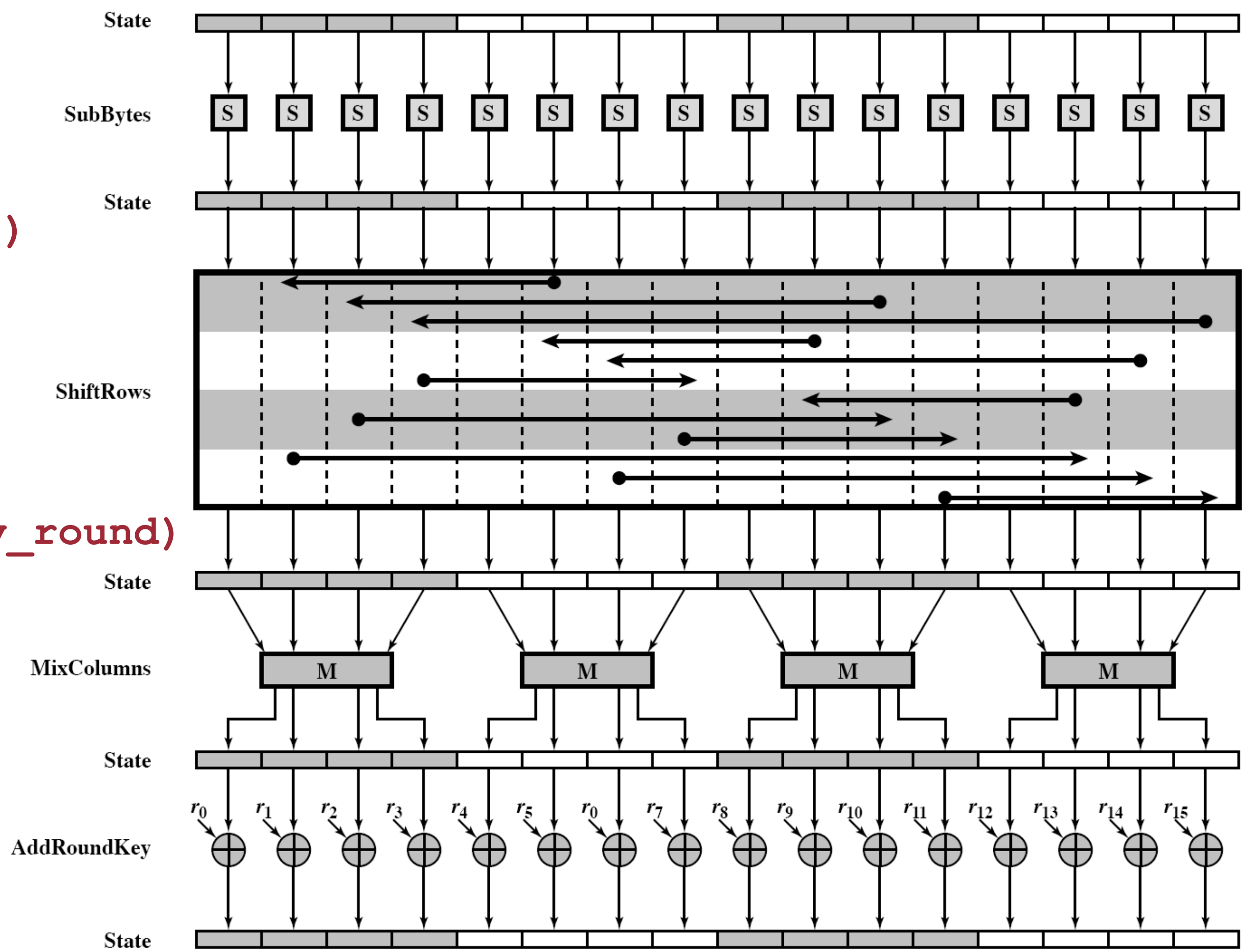
- AES (Advanced Encryption Standard) parameters:
  – Key size: 128, 192, 256 bits
  – Input block length: 128 bits

- 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

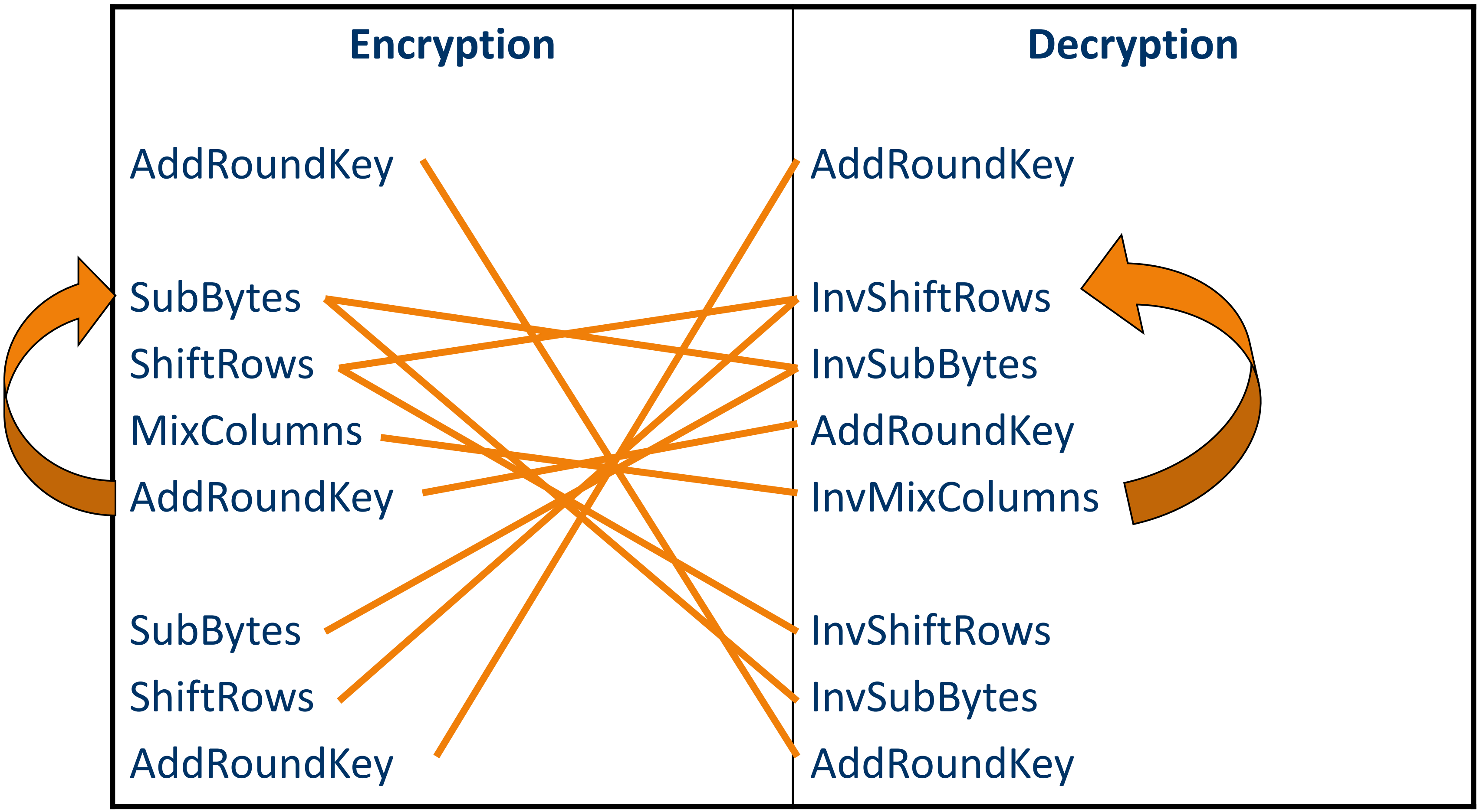
  – With fast XOR & table lookup implementations

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

Roadmap

- Introduction
- Symmetric Ciphers
- **Hash Functions**
- Message Integrity Codes

Hash functions / message digests

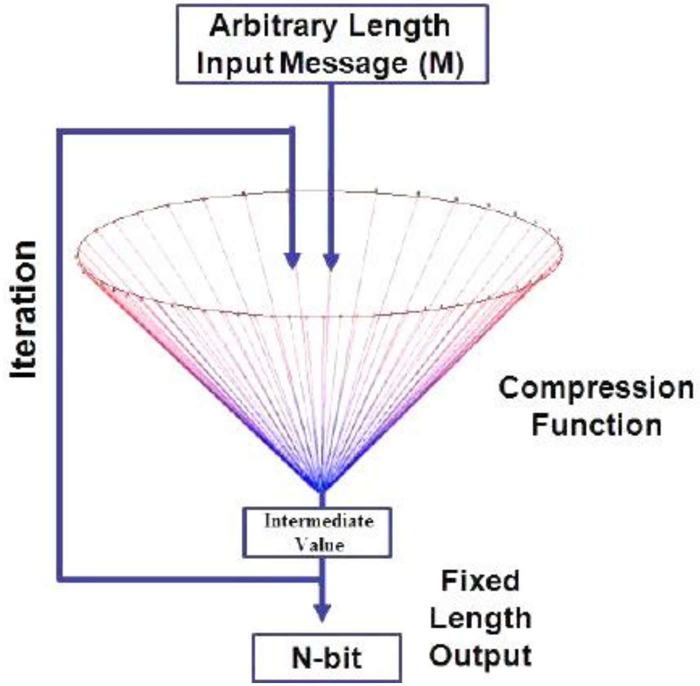
- Aka cryptographic hash functions or collision-resistant hash functions
  - Do not confuse with hash functions used in hash tables (that you learned in IAED)
- They are cryptographic, but not ciphers
  - not used for encryption

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

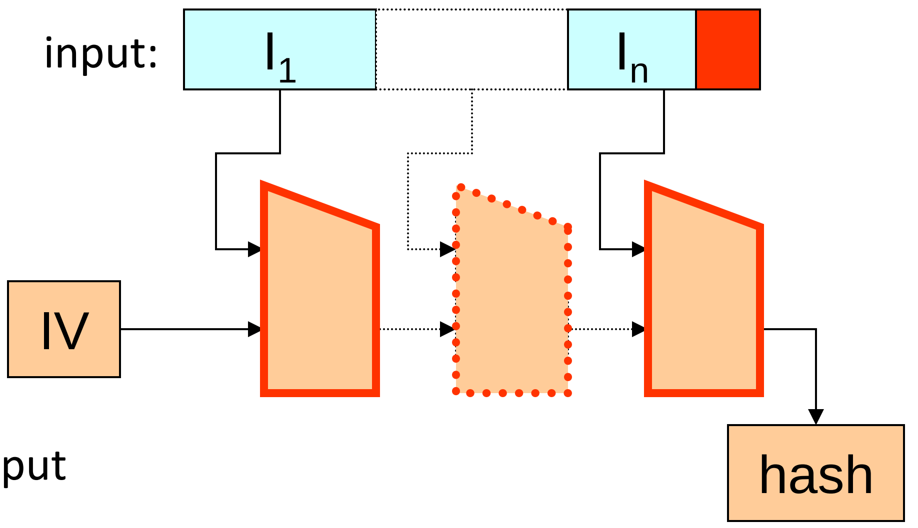
Hash functions

- 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

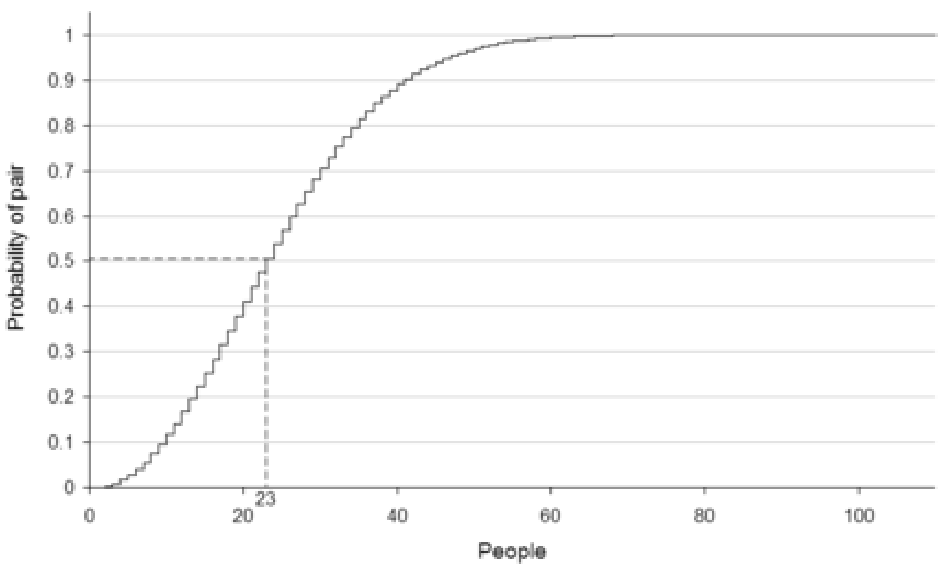


Hash functions

- 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 << 2^{64}
  - **SHA-1** (Secure Hash Algorithm, 160 bits) *Weak*
    - Collisions found in 2^{63} tries << 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



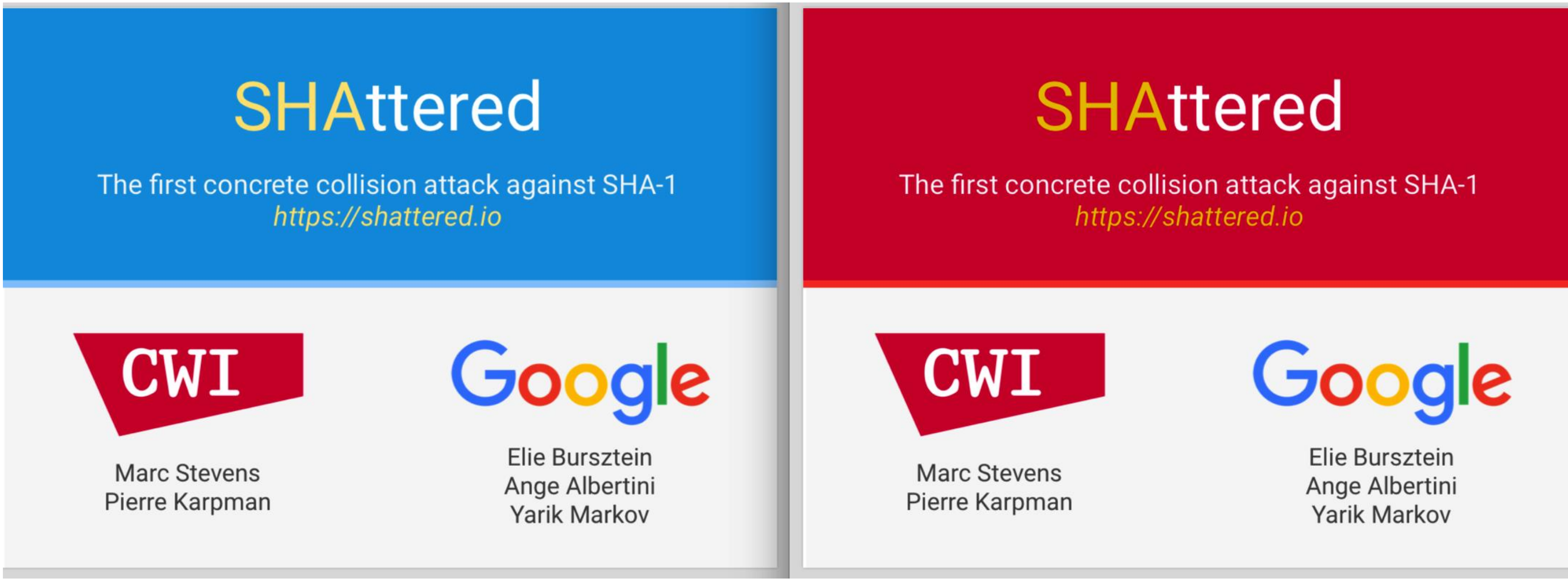
Birthday paradox

- Paradox
  - *P(me): probability n students having the same birthday as me ("collision")*
    - Probability A's birthday date is the same as mine = 1/365 ~= 0.003
    - Probability A's birthday date different from mine = 1 - 1/365 ~= 0.997
    - Probability A and B's dates different from mine = $(1 - 1/365)^2$ ~= 0.994
    - P(me) = $1-(1-\frac{1}{365})^n$
    - P(me) > 0,5 $\Leftrightarrow$ n $\geq$ 253
  - *P(pair): probability of pair with the same birthday*
    - P(pair) = $1-(1-\frac{1}{365})(1-\frac{2}{365})\ldots(1-\frac{n-1}{365})$
    - P(pair) > 0,5 $\Leftrightarrow$ n $\geq$ 23 !!!

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(collision) > 0.5 = $2^m/2$ = $\mathbf{2^{m-1}}$
  - 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(collision) > 0.5 ~= $\mathbf{2^{m/2}}$ tries (only)
- Cryptanalysis leads to even faster attacks

SHA-1 is no longer secure



Here are two different PDF files,
but with the same SHA-1 hash

https://shattered.io/

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

COMPUTER SECURITY

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

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 Integrity Codes**

Message Integrity Codes

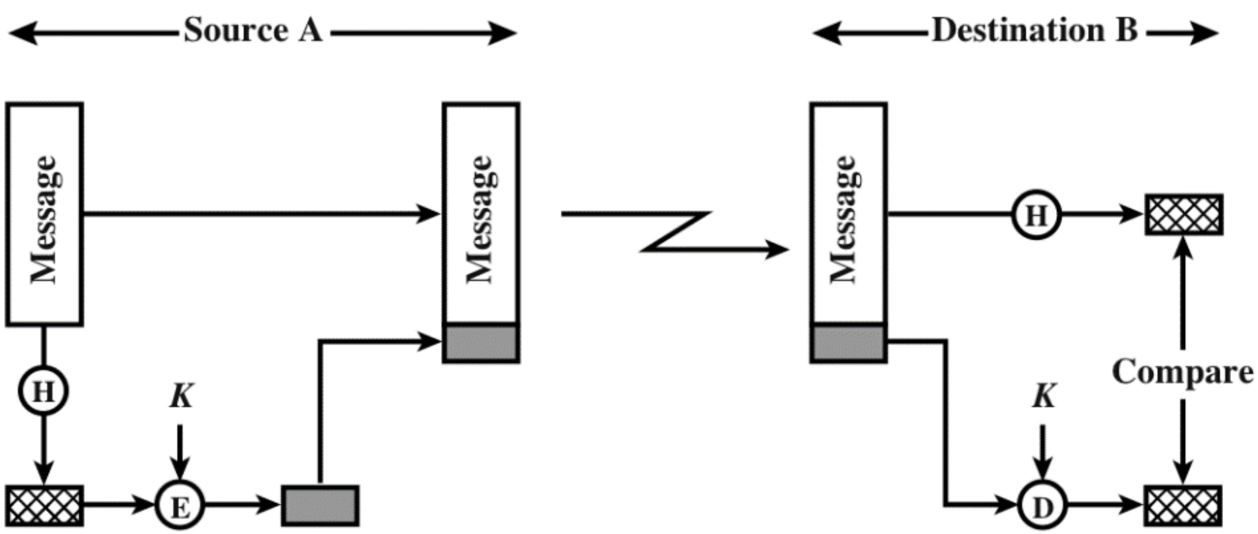
- Objective: detect changes to a message
  - Allows checking its integrity
  - With freshness, can provide authenticity
  - So, many times, called a MAC (Message Authentication Code)
- Assumption: sender and recipient have a shared secret key K
- Idea:
  - Send the message + MAC
  - If the message (or the MAC) is modified by an attacker, the recipient will be able to detect it
  - Attacker cannot create a valid MIC because he does not 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
- Implementation alternatives:
  - 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) – now using AES
    - Authenticated Encryption
      - Encrypts the data and generates an authentication Tag, e.g GCM
  - 3: Using a keyed-hash
    - 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 and encrypt

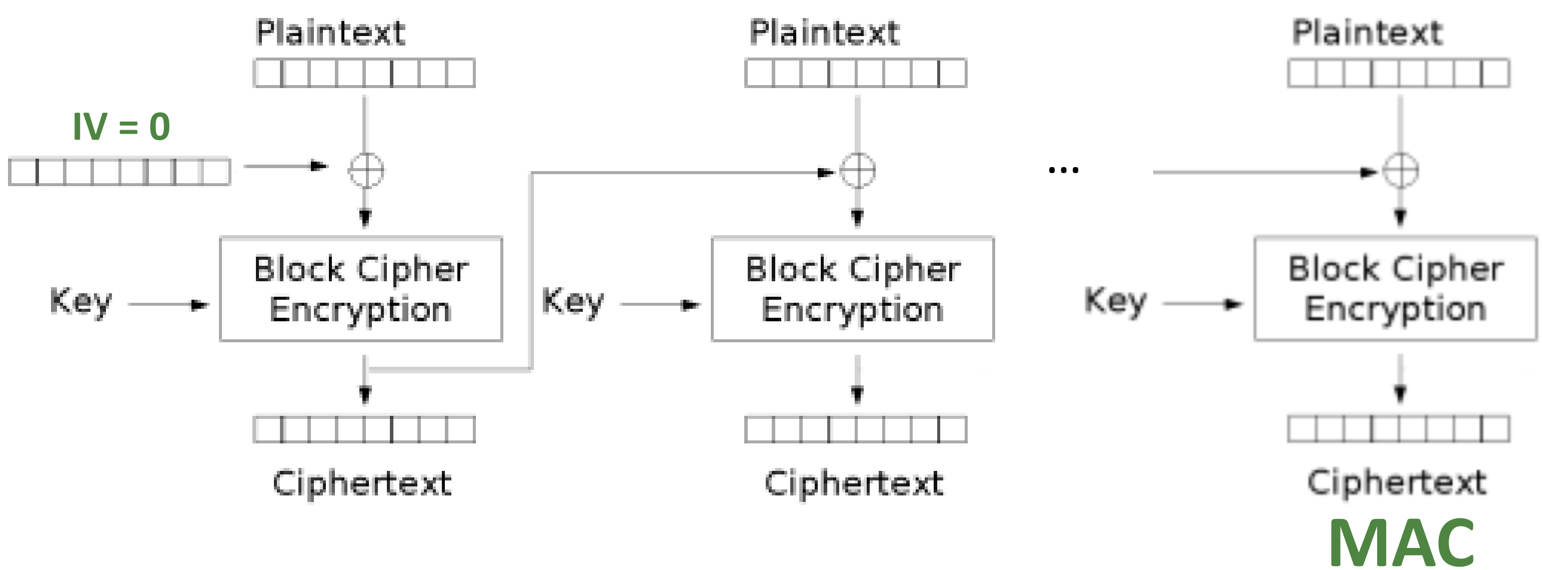
- Hash the message and encrypt the digest



If attacker modifies the message (or the hash), the hash calculated at the destination will not match the hash received

Message Authentication Code (MAC) 2: keyed function

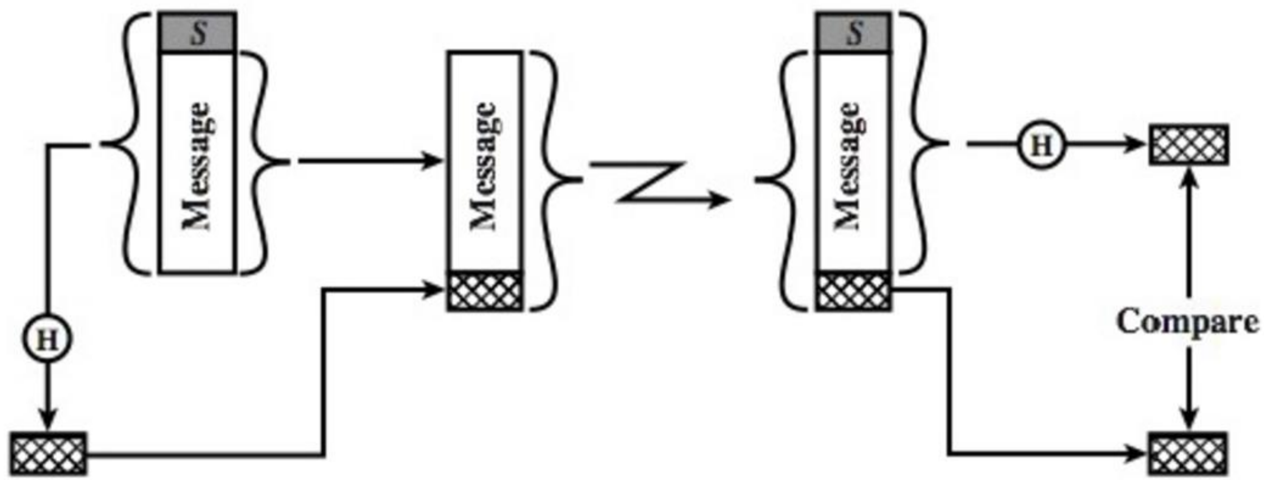
- CMAC (Cipher-based MAC)
  - Use a block cipher in CBC mode
  - MAC is the last block
  - Must be CBC for MAC to depend on the whole message



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

Message Authentication Code (MAC) 3: keyed hash

- Hash the message along with a shared key
  - Put secret in the beginning of message
  - Ad-hoc algorithm showing the basic idea:



Not recommended! Extension attacks are possible

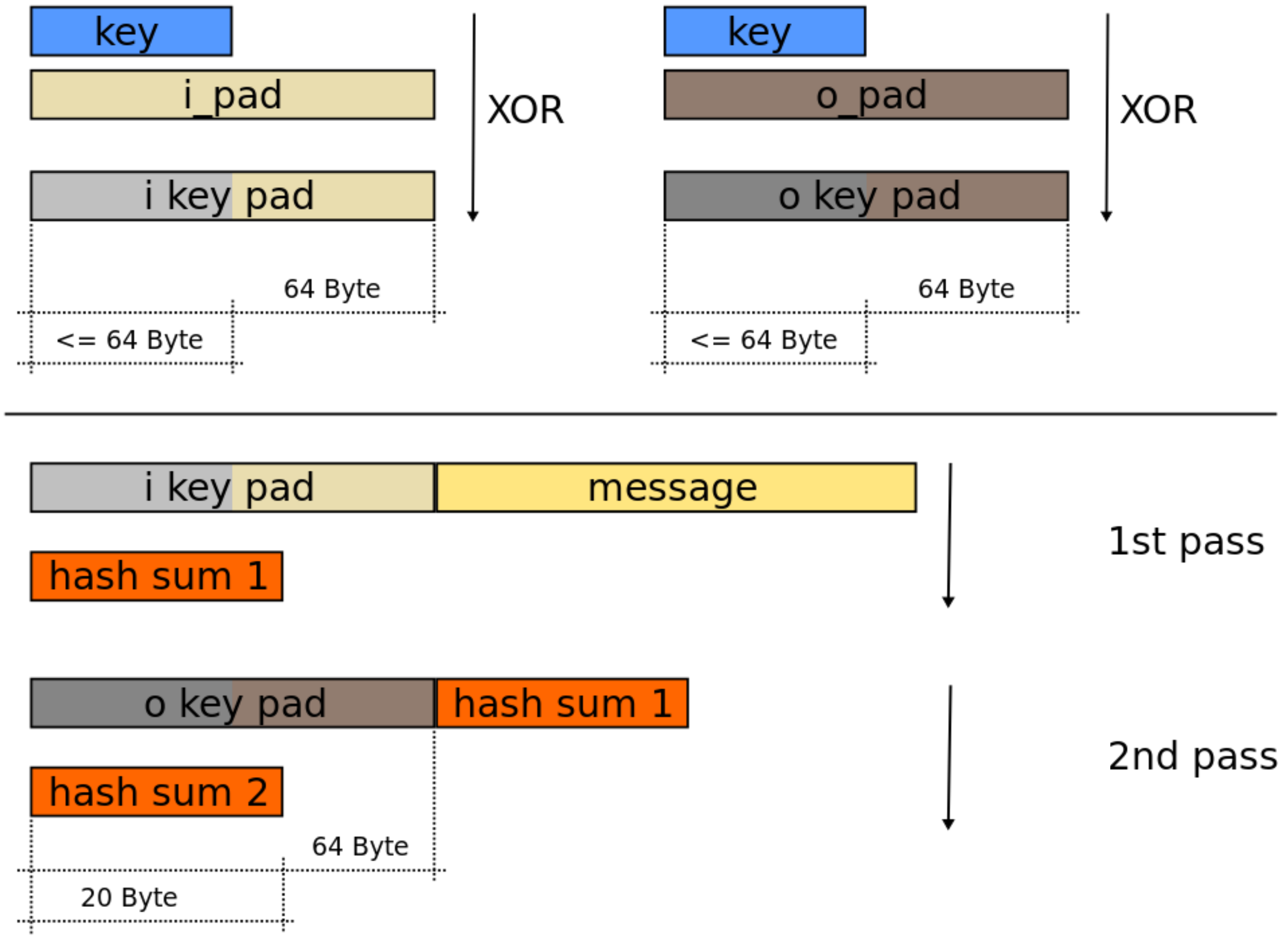
Extension attacks to keyed hash

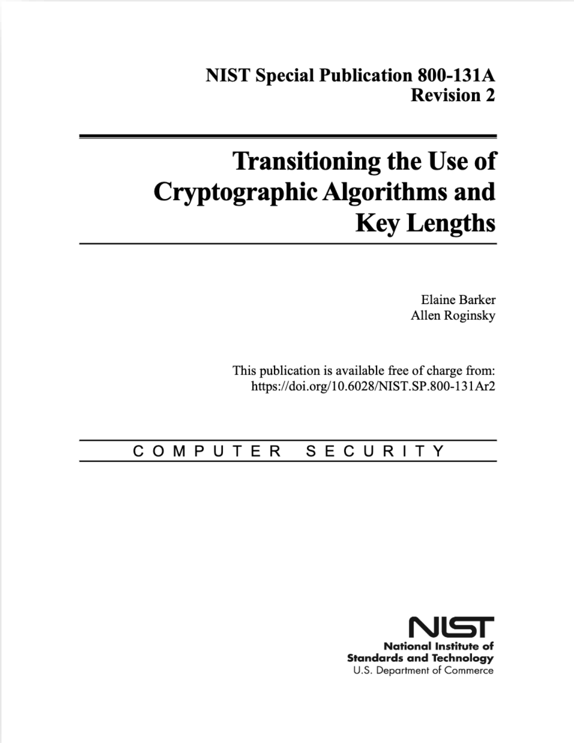
- Append additional data to the original message without knowing the secret key
  - Attacker intercepts a message M and its valid MAC
  - Without knowing the key, append extra data D to M, creating M' = M || D
  - Calculate a new MAC for M' using the intercepted MAC and **the structure of the hash function that continues from the previous value**
  - The recipient, unaware of the alteration, verifies M' with the provided MAC, which appears valid
  - This successfully deceives the recipient into accepting the tampered message \(M' \) as authentic

HMAC (Hash-based MAC)

- HMAC algorithm
  - FIPS Standard / RFC 2104: keyed hash MAC
  - Introduces the following technique to prevent extension attacks
  - $HMAC(m,k) = hash(k \oplus opad \,||\, hash(k \oplus ipad \,||\, m))$
    - ipad – inner padding – 0x36363636
    - opad – outer padding – 0x5c5c5c5c
    - HMAC is used in practice: SSL/TLS, WTLS, IPsec

HMAC processing steps




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

COMPUTER SECURITY

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

Table 9: Approval Status of MAC Algorithms

MAC Algorithm	Implementation Details	Status
HMAC Generation	Key lengths < 112 bits	Disallowed
	Key lengths ≥ 112 bits	Acceptable
HMAC Verification	Key lengths < 112 bits	Legacy use
	Key lengths ≥ 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 ≥ 112 bits	Acceptable

CMAC, GMAC, KMAC
são standards do NIST

Summary

- Introduction
- Symmetric Ciphers
- Hash Functions
- Message Integrity Codes