Overview

- Modern symmetric-key cryptosystems:
  - The Data Encryption Standard (DES)
    - Block size = 64 bits
    - Key length = 56 bits
      - Adopted 1976
  - The Advanced Encryption Standard (AES)
    - Block sizes = 128, 192, or 256 bits
    - Key lengths = 128, 192, or 256 bits
      - Adopted 2000

DES - History

- 1973: the National Bureau of Standards (NBS) solicits proposals for a standard cryptographic algorithm which:
  - Provides a high level of security
  - Is completely specified and easy to understand
  - Is available royalty-free to the U.S. government and all other users
  - Be efficient and economically implementable on electronic devices

DES – History (cont)

- 1974:
  - A team of IBM cryptographers submits a Lucifer algorithm variant
  - NBS asks National Security Agency (NSA) for comments
  - NSA recommends that algorithm be adopted with several modifications:
    - Key size reduced from 128 to 56 bits
    - A few minor algorithm changes details

DES – History (cont)

- 1976:
  - NBS approves DES as a U.S. government standard for use on all unclassified communications
  - The standard to be reviewed every five years
- 1983: NBS recertifies DES
- 1987: NBS recertifies DES
- 1988: NBS becomes the National Institute of Standards and Technology (NIST)
- 1993: NIST recertifies DES
- 1998: NIST begins a competition to establish an Advanced Encryption Standard to replace DES

DES - Overview

- 1976, adopted by the U.S. government as a Federal Information Processing Standard (FIPS)
- Block cipher
  - Encrypts 64-bit plaintext blocks to generate 64-bit ciphertext blocks
- Symmetric key
  - Same algorithm and 56-bit key are used for encryption and decryption

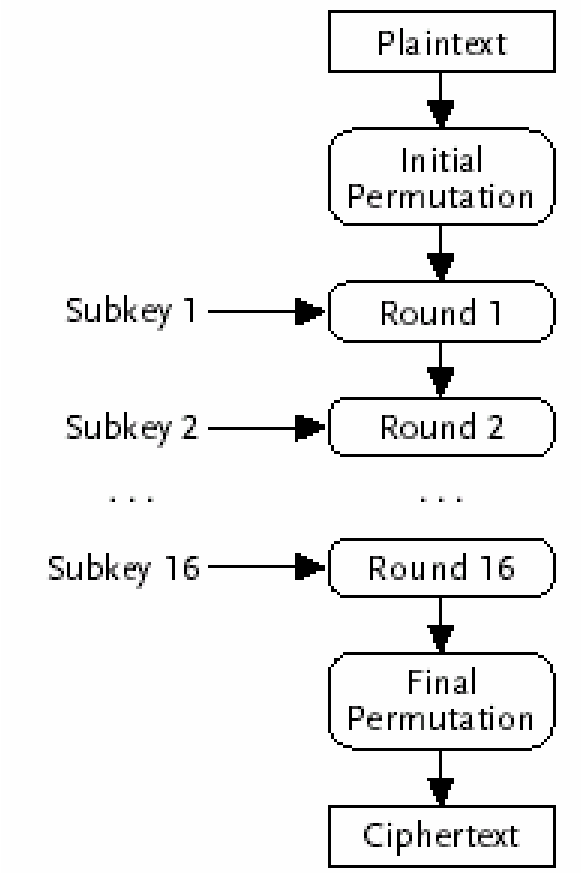
DES - Keys

- Key = any 56-bit pattern
- Keyspace contains 2^{56} elements
  - 72,057,594,037,927,936 different keys
- Exhaustive search at one trillion keys per second takes:
  - 2 hours
- A very small number of weak keys are known
  - Should be avoided

DES – The Algorithm

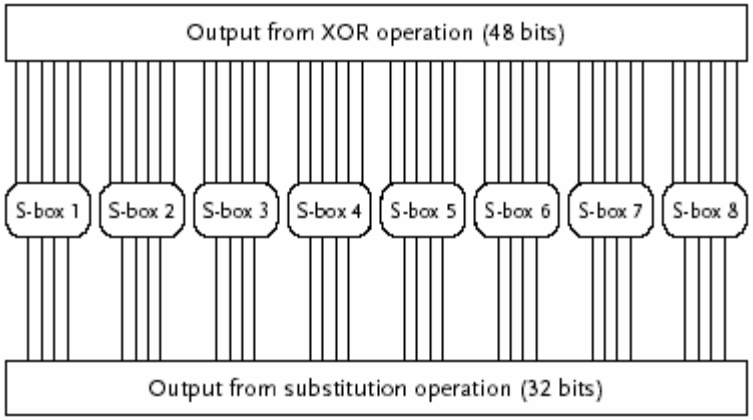
- Transforms a 64-bit plaintext block into a 64-bit ciphertext block
- Each 64-bit plaintext block goes through:
  - An **initial permutation**
  - 16 **rounds** of substitution and transposition operations
    - Influenced by a 48-bit **subkey** for each round, which is derived from the 56-bit DES key
  - **Final permutation**

DES – Algorithm Overview



DES – S-boxes

- **S-boxes** perform substitution operations
- There are 8 different S-boxes
- Each S-box takes 6 input bits and produces 4 output bits:



- Bits 1-6 are the input to S-box 1
- Bits 7-12 are the input to S-box 2, etc.

DES – P-box

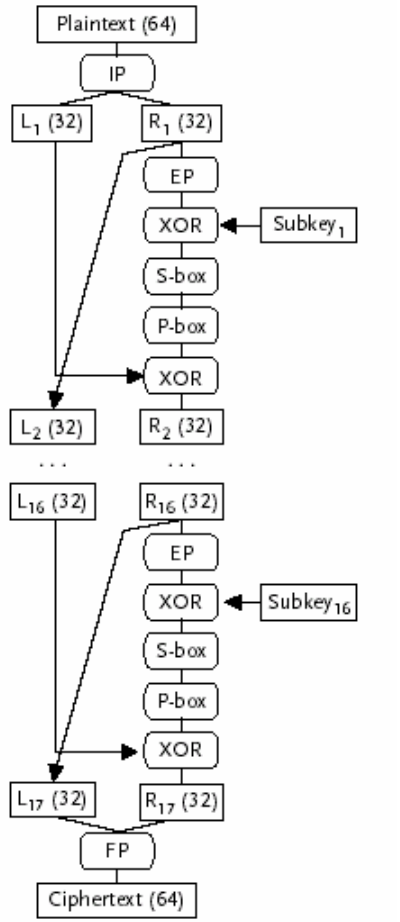
- The 32-bit output of the S-boxes is passed through a **P-box**
- The P-box permutes the bits into a new order:

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

  - The first output bit from the S-boxes is moved into position 16
  - The second bit is moved into position 7
  - The third bit is moved into position 20

    …
  - The thirty-second bit is moved into position 25

DES – Encryption Overview



DES - Decryption

- The same algorithm and key is used for decryption
- The subkeys are applied in the opposite order
  - Subkey 16 is used during the first round of decryption
  - Subkey 15 is used during the second round of decryption

    …
  - Subkey 1 is used during the 16th round of decryption

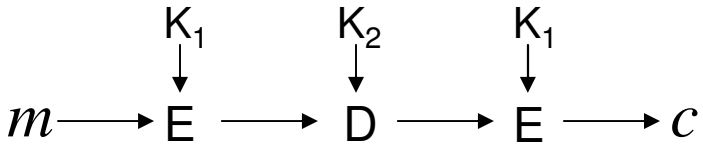
DES - Summary

- DES is still a widely used cryptosystem
- Increased computing power has weakened the protection offered by DES considerably:
  - 1998: the Electronic Frontier Foundation builds a $220,000, special-purpose machine that could recover the key for a message encrypted with DES in about four days
- DES helped focus and unify the public cryptographic research community
- NIST's 1998 call for an Advanced Encryption Standard to replace DES produced 15 promising candidate algorithms

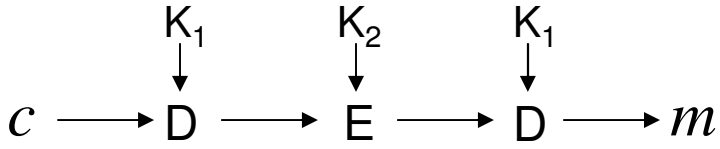
Multiple Encryption with DES

3DES:

- Define two key values K_1 and K_2.
- Each block is encrypted as: (the second pass encrypts with decryption)



- Decryption does the reverse:



See [Kaufman 2002] if you want to understand why the 3rd time is the charm.

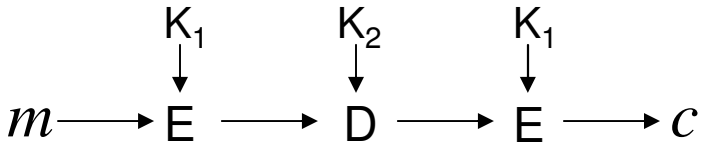
Note that encrypting twice with the same key is not much more than a single encryption (exhaustive search requires the same number of keys to be tested; it is true that each key has to be tested twice, but that isn't a big deal).

Also, encrypting twice with two keys is not as strong as encrypting once with a key twice as long. There exists a possible attack that breaks double-encryption DES in roughly twice the time for a brute-force attack on single-encryption DES.

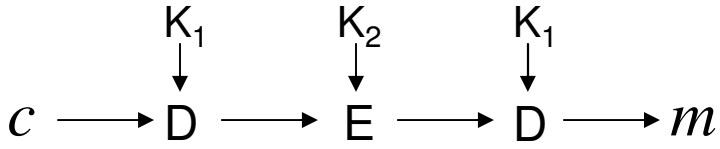
Multiple Encryption with DES

3DES:

- Define two key values K_1 and K_2.
- Each block is encrypted as: (the second pass encrypts with decryption)



- Decryption does the reverse:



See [Kaufman 2002] if you want to understand why the 3rd time is the charm.

Note that encrypting twice with the same key is not much more than a single encryption (exhaustive search requires the same number of keys to be tested; it is true that each key has to be tested twice, but that isn't a big deal).

Also, encrypting twice with two keys is not as strong as encrypting once with a key twice as long. There exists a possible attack that breaks double-encryption DES in roughly twice the time for a brute-force attack on single-encryption DES.

Block Cipher Modes

Using Block Ciphers: Padding

- If the plaintext length is not an exact multiple of the block size, the plaintext needs to be padded.
- Padding must be reversible.
- An erroneous padding must be treated as an authentication failure.

- Let *l(P)* be the length of the plaintext in bytes. Use one of the two following schemes:

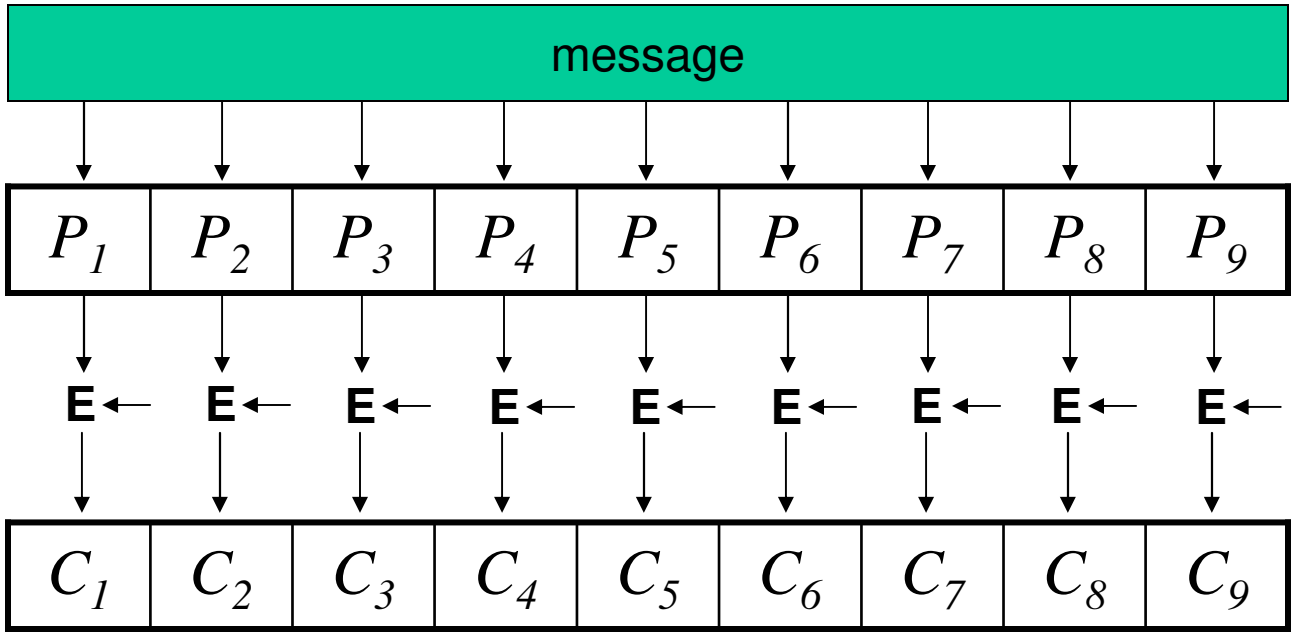
  1) Append a single byte with value 128, then as many 0 bytes to make *l(P)* a multiple of *b*.

  2) Let *n* be the number of padding bytes required. Pad *P* with *n* bytes, each with value *n*. Calculate *n* so that it satisfies:

$$n + l(P) = k * b, \text{ for some integer } k$$

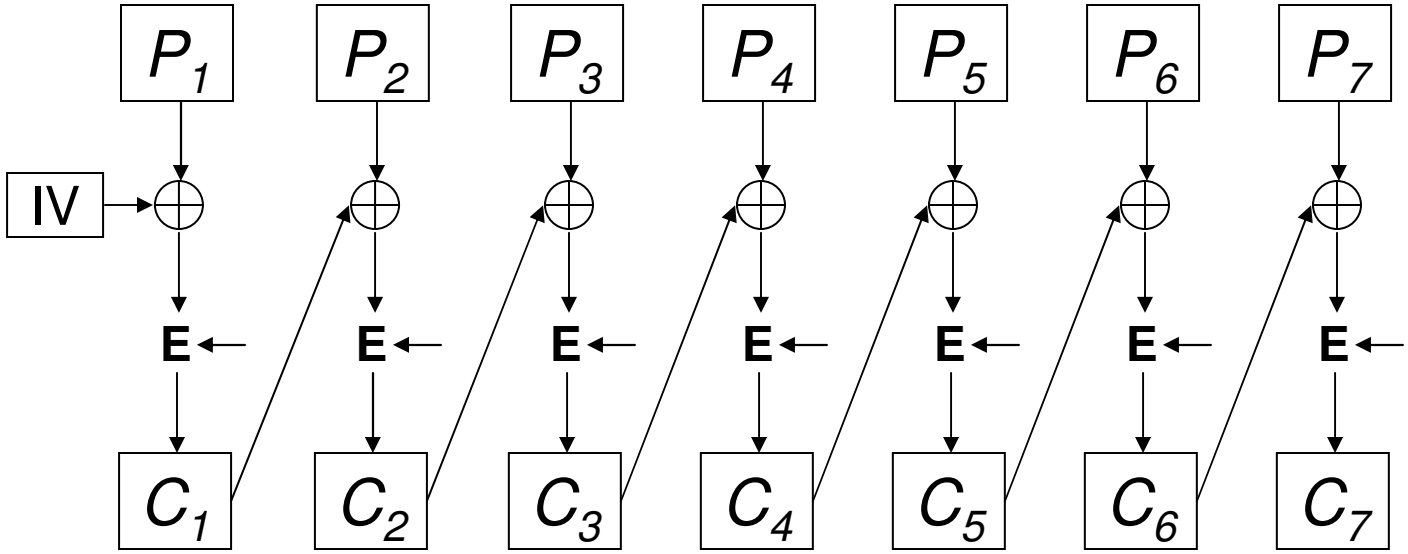
$$1 \le n \le b$$

Electronic Code Book (ECB)



- If any two blocks m_i and m_j are identical, the corresponding c_i and c_j will also be identical: one can learn the key from the repeated blocks.
- The blocks can be rearranged or tampered with.

Cipher Block Chaining (CBC)



- If all messages were to use the same IV, someone could figure out facts about the messages being encrypted.
- If IV is chosen randomly, then even if the same message is repeatedly encrypted, their corresponding ciphertext will be different each time.
- The message may still be modified in transit: the effect may be obvious to human eyes, but hard to spot by a program.

Cipher Block Chaining (CBC)

$$C_i = E(K, P_i \oplus C_{i-1}), \text{ for } i = 1,...,k$$

Each block of plaintext is “randomized” according to the previous ciphertext block. Identical plaintext blocks will most likely encrypt to different ciphertext blocks.

How do you choose C_0 (the initialization vector)? Two different messages starting with the same block, yield ciphertexts that start with the same block: this opens a breach for attackers. Possible solutions:

1) Use a message counter.

2) Use a random number: $C_0 = \text{random block value}$

$$C_i = E(K, P_i \oplus C_{i-1}), \text{ for } i = 1,...,k$$

3) Use a nonce: Each message is given a *nonce* (a unique *n*umber used only *once*). You have to be careful never to use the same nonce twice with the same key! Each message *P* is assigned a number (a counter that does not wrap around). This number is used to construct a nonce unique in the whole system. The nonce has the size of a block, and is next encrypted with *K* producing C_0. Send the message number in front of the ciphertext, so that the receiver can reconstruct the nonce. The receiver must accept any one message number only once!

Output Feedback Mode (OFB)

Stream cipher: message is XORed with the one-time pad generated by OFB.

$$K_0 = \mathrm{IV}$$

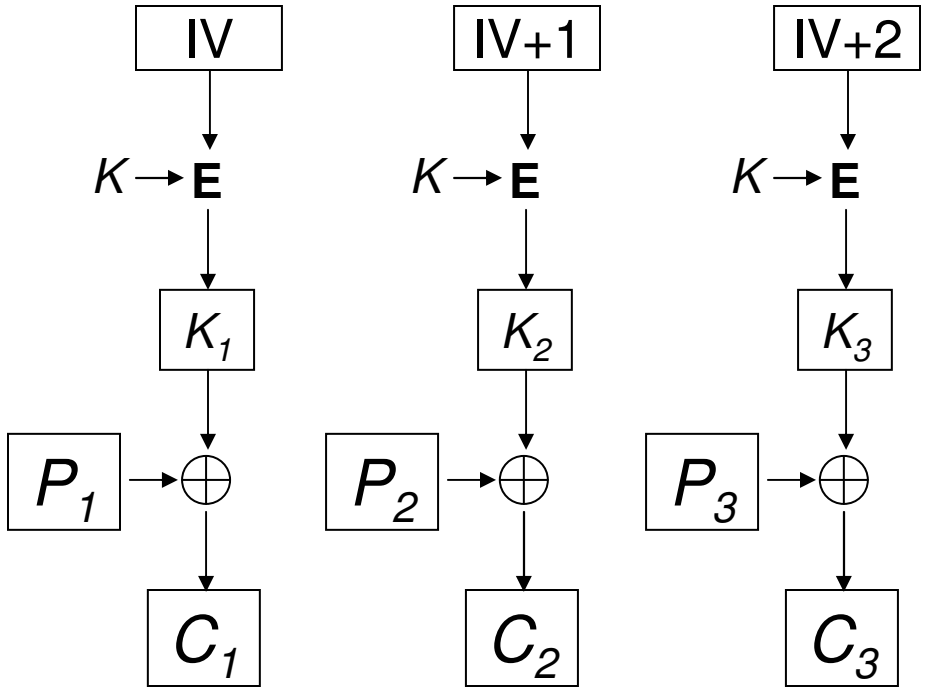
$$K_i = E(K, K_{i-1}) \text{ for } i = 1, \mathrm{K}\ , k$$

$$C_i = P_i \oplus K_i$$

- IV can be chosen randomly and sent with ciphertext or it can be generated from a nonce.
-There is no need for padding: you only send as many ciphertext bytes as there are plaintext bytes.
- The secrecy relies on the one-time pad really being used only once!
- If a lot of data is encrypted, it is possible that a sequence of key blocks could start repeating…

Counter Mode (CTR)

Stream cipher. Key stream is generated very simply using the IV as a starting point and adding to it a counter value (which represents the number of blocks processed).



$$K_i = E(K, \text{Nonce} \| i) \quad \text{for } i = 1, \text{K}\ , k$$

$$C_i = P_i \oplus K_i$$

AES - History

- 1997: to replace DES, NIST requests proposals for a new Advanced Encryption Standard (AES)
- NIST required that the algorithm be:
  - A symmetric-key cryptosystem
  - A block cipher
  - Capable of supporting a block size of 128 bits
  - Capable of supporting key lengths of 128, 192, and 256 bits
  - Available on a worldwide, non-exclusive, royalty-free basis
- Evaluation criteria:
  - Security - soundness of the mathematical basis and the results of analysis by the research community
  - Computational efficiency, memory requirements, flexibility, and simplicity

AES – Round 1 of the Competition

- NIST selects 15 submissions for evaluation:
  - CAST-256 (Entrust Technologies, Inc.)
  - Crypton (Future Systems, Inc.)
  - DEAL (Richard Outerbridge, Lars Knudsen)
  - DFC (Centre National pour la Recherche Scientifique—Ecole Normale Superieure)
  - E2 (Nippon Telegraph and Telephone Corporation)
  - Frog (TecApro Internacional S.A.)
  - HPC (Rich Schroeppel)
  - Loki97 (Lawrie Brown, Josef Pieprzyk, Jennifer Seberry)
  - Magenta (Deutsche Telekom AG)
  - MARS (IBM)
  - RC6 (RSA Laboratories)
  - Rijndael (Joan Daemen, Vincent Rijmen)
  - SAFER+ (Cylink Corporation)
  - Serpent (Ross Anderson, Eli Biham, Lars Knudsen)
  - Twofish (Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall, Niels Ferguson)

AES – Round 1 Results

- After eight months of analysis and public comment, NIST:
  - Eliminated DEAL, Frog, HPC, Loki97, and Magenta
    - Had what NIST considered major security flaws
    - Were among the slowest algorithms submitted
  - Eliminated Crypton, DFC, E2, and SAFER+
    - Had what NIST considered minor security flaws
    - Had unimpressive characteristics on the other evaluation criteria
  - Eliminated CAST-256
    - Had mediocre speed and large ROM requirements
- Five candidates, MARS, RC6, Rijndael, Serpent, and Twofish, advanced to the second round

AES – Results

- Analysis and public comment on the five finalists
- October 2000: NIST:
  - Eliminates MARS
    - High security margin
  - Eliminates RC6
    - Adequate security margin, fast encryption and decryption on 32-bit platforms
  - Eliminates Serpent
    - High security margin
  - Eliminates Twofish
    - High security margin
  - Selects Rijndael
    - Adequate security margin, fast encryption, decryption, and key setup speeds, low RAM and ROM requirements

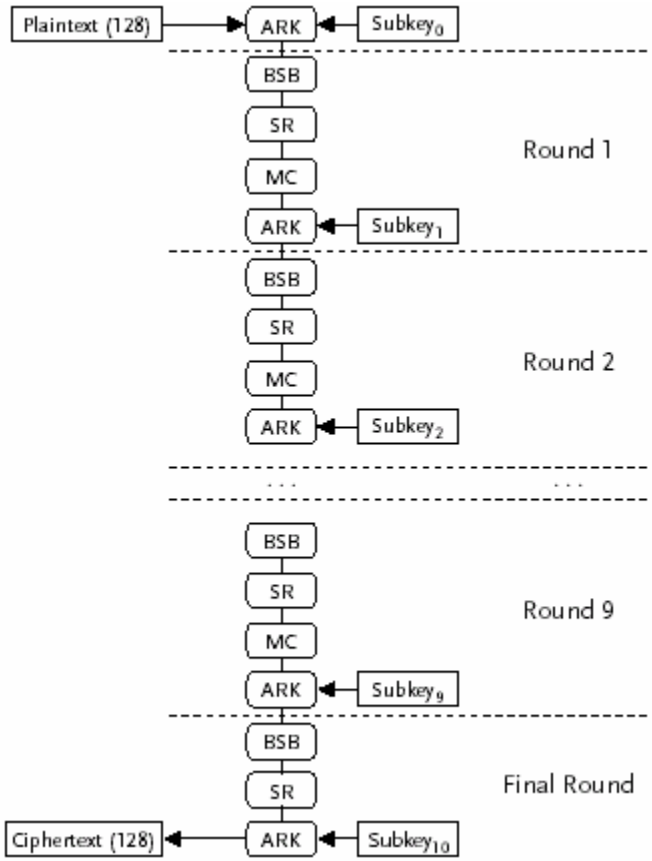
AES –Rijndael Algorithm

- Symmetric-key block cipher
  - Block sizes are 128, 192, or 256 bits
  - Key lengths are 128, 192, or 256 bits
- Performs several rounds of operations to transform each block of plaintext into a block of ciphertext
  - The number of rounds depends on the block size and the length of the key:
    - Nine regular rounds if both the block and key are 128 bits
    - Eleven regular rounds if either the block or key are 192 bits
    - Thirteen regular rounds if either the block or key is 256 bits
  - One, slightly different, final round is performed after the regular rounds

AES – The Rijndael Algorithm (cont)

- For a 128-bit block of plaintext and a 128-bit key the algorithm performs:
  - An initial AddRoundKey (ARK) operation
  - Nine regular rounds composed of four operations:
    - ByteSub (BSB)
    - ShiftRow (SR)
    - MixColumn (MC)
    - AddRoundKey (ARK)
  - One final (reduced) round composed of three operations:
    - ByteSub (BSB)
    - ShiftRow (SR)
    - AddRoundKey (ARK)

AES – Rijndael Overview

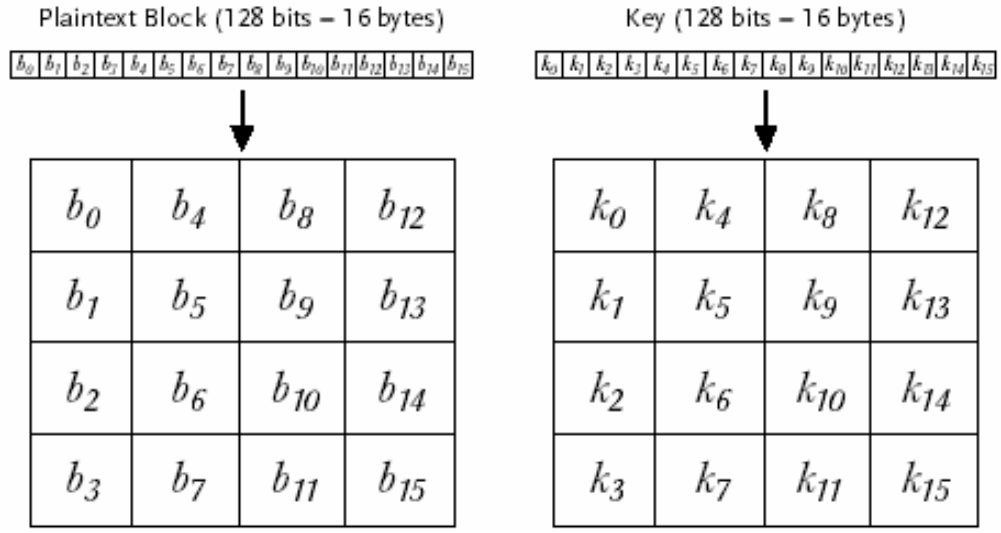


AES – Rijndael Keys

- Keys are expressed as 128-bit (or bigger) quantities
- Keyspace contains at least 2^{128} elements:
  - 340,282,366,920,938,463,463,374,607,431,768,211,456
- Exhaustive search at one trillion keys per second takes:
  - $1x10^{19}$ years (the universe is thought to be about $1x10^{10}$ years old)

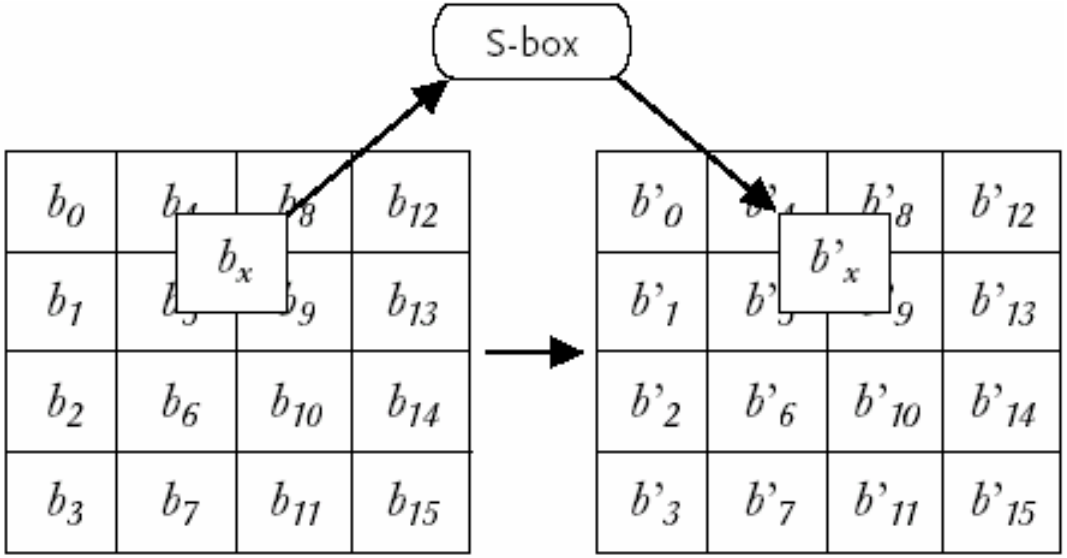
AES – Rijndael Keys (cont)

- Blocks and keys are represented as a two-dimensional array of bytes with four rows and four columns:
  - Block = 128 bits = 16 bytes = b_0 , b_1, . . ., b_{15}
  - Key = 128 bits = 16 bytes = k_0 , k_1, . . ., k_{15}



AES - The ByteSub Operation

- An S-box is applied to each of the 16 input bytes independently
- Each byte is replaced by the output of the S-box:



AES – The Rijndael S-box

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

AES – The Rijndael S-box (cont)

- The input to the S-box is one byte:
- Example 1:
  - b_0 = 01101011 (binary) = 6b (hex)
  - b'_0 = row 6, column b = 7f (hex) = 01111111 (binary)
- Example 2:
  - b_1 = 00001000 (binary) = 08 (hex)
  - b'_1 = row 0, column 8 = 30 (hex) = 00110000 (binary)
- Example 3:
  - b_2 = 11111001 (binary) = f9 (hex)
  - b'_2 = row f, column 9 = 99 (hex) = 10011001 (binary)

AES - The ShiftRow Operation

- Each row of the input is circularly left shifted:
  - First row by zero places
  - Second row by one place
  - Third row by two places
  - Fourth row by three places

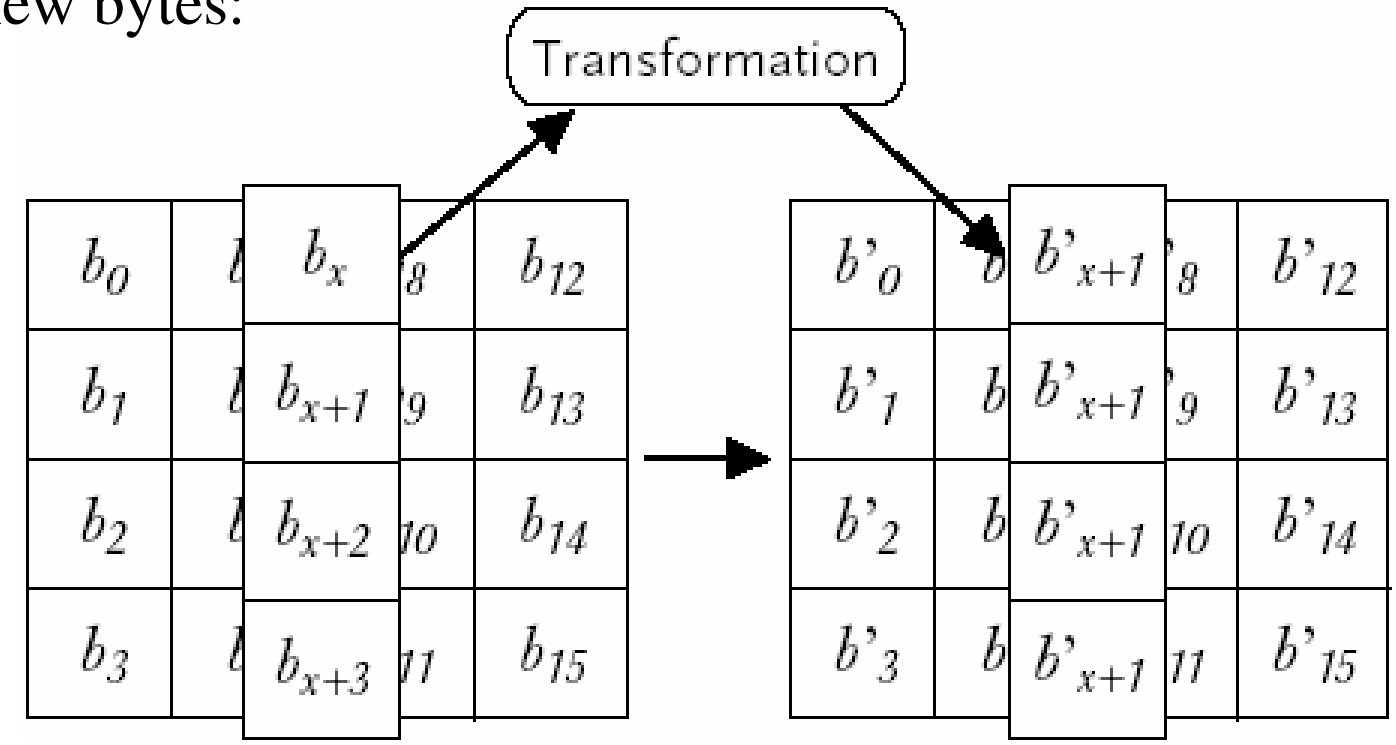
b_0	b_4	b_8	b_{12}
b_1	b_5	b_9	b_{13}
b_2	b_6	b_{10}	b_{14}
b_3	b_7	b_{11}	b_{15}

→

b_0	b_4	b_8	b_{12}
b_5	b_9	b_{13}	b_1
b_{10}	b_{14}	b_2	b_6
b_{15}	b_3	b_7	b_{11}

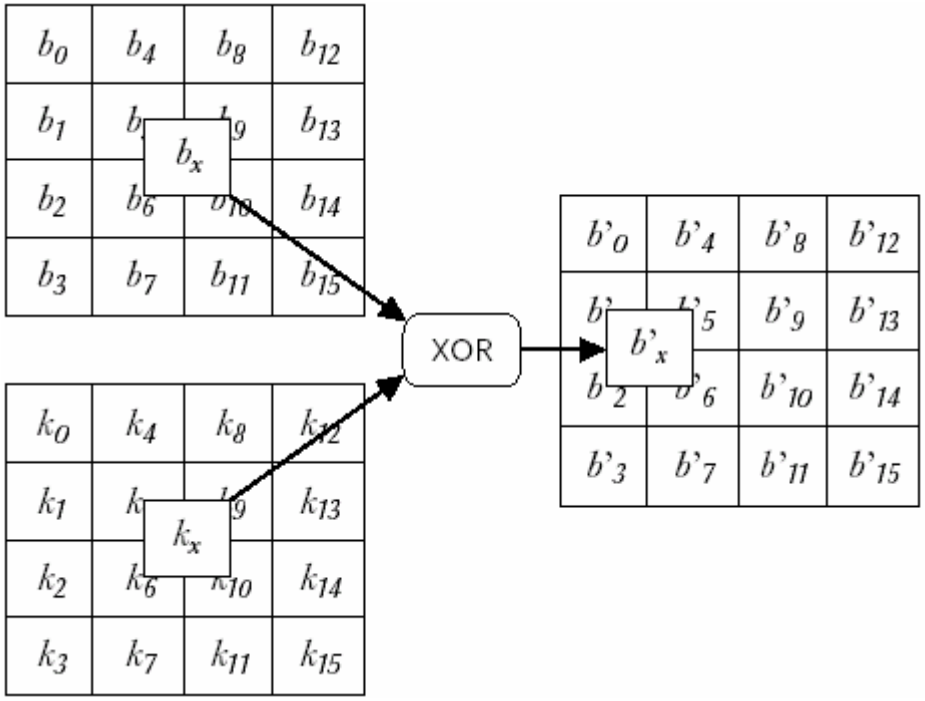
AES - The MixColumn Operation

- The four bytes in each input column are replaced with four new bytes:

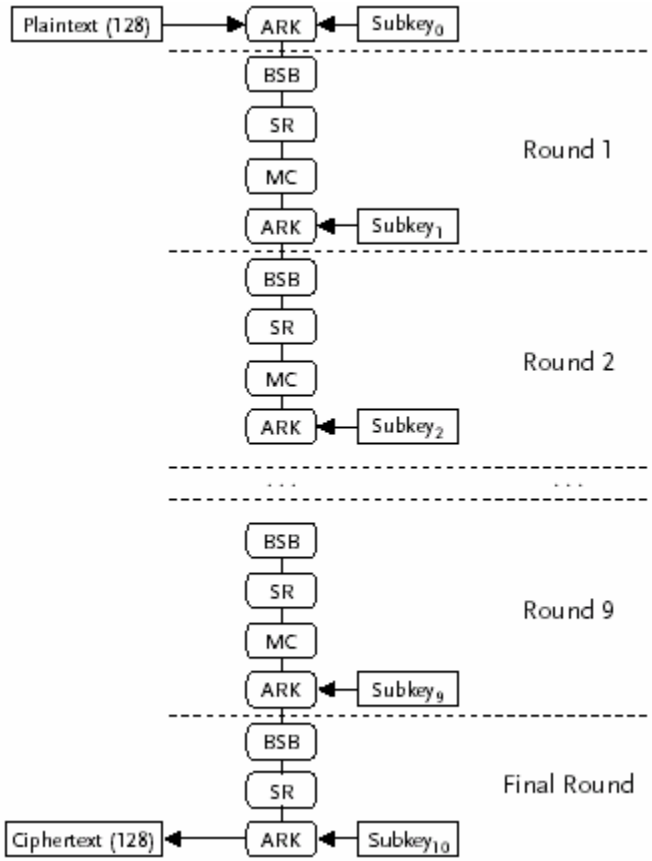


AES - The AddRoundKey Operation

- Each byte of the input block is XORed with the corresponding byte of the round subkey:



AES – Rijndael Overview



AES Summary

- The AES selection process served to raise public awareness of cryptography and its importance
- The AES algorithm is starting to be widely used
- The AES should offer useful cryptographic protection for at least the next few decades

Summary

- DES (56-bit keys): Each 64-bit block of plaintext goes through:
  - Initial permutation
  - Sixteen rounds of substitution and transposition operations
  - Final permutation
- AES (128-bit keys): Each 128-bit block of plaintext goes through:
  - An initial AddRoundKey (ARK) operation
  - Nine rounds of operations (BSB, SR, MC, ARK)
  - One final (reduced) round

Overview

- Cryptographic hash functions are functions that:
  - Map an arbitrary-length (but finite) input to a fixed-size output
  - Are one-way (hard to invert)
  - Are collision-resistant (difficult to find two values that produce the same output)
- Examples:
  - Message digest functions
    - Protect the integrity of data by creating a fingerprint of a digital document
  - Message Authentication Codes (MAC)
    - Protect both the integrity and authenticity of data by creating a fingerprint based on both the digital document and a secret key

Checksums vs. Message Digests

- Checksums:
  - Used to produce a compact representation of a message
  - If the message changes the checksum will probably not match
  - Good: accidental changes to a message can be detected
  - Bad: easy to purposely alter a message without changing the checksum
- Message digests:
  - Used to produce a compact representation (called the **fingerprint** or **digest**) of a message
  - If the message changes the digest will probably not match
  - Good: accidental changes to a message can be detected
  - Good: difficult to alter a message without changing the digest

Hash Functions

- Message digest functions are hash functions
  - A **hash function**, $H(M)=h$, takes an arbitrary-length input, M, and produces a fixed-length output, h
- Example hash function:
  - H = sum all the letters of an input word modulo 26
  - Input = a word
  - Output = a number between 0 and 25, inclusive
  - Example:
    - H(“Elvis”) = ((‘E’ + ‘L’ + ‘V’ + ‘I’ + ‘S’) mod 26)
    - H(“Elvis”) = ((5+12+22+9+19) mod 26)
    - H(“Elvis”) = (67 mod 26)
    - H(“Elvis”) = 15

Collisions

- For the hash function:
  - H = sum all the letters of an input word modulo 26
- There are more inputs (words) than possible outputs (numbers 0-25)
- Some different inputs must produce the same output
- A **collision** occurs when two different inputs produce the same output:
  - The values x and y are not the same, but $H(x)$ and $H(y)$ are the same

Collisions - Example

- *H*(“Jumpsuit”) = 25
  - (‘J’ + ‘U’ + ‘M’ + ‘P’ + ‘S’ + ‘U’ + ‘I’ + ‘T’) mod 26
  - (10+21+13+16+19+21+9+20) mod 26
  - 129 mod 26
  - 25
- *H*(“TCB”) = 25
  - (‘T’ + ‘C’ + ‘B’) mod 26
  - (20+3+2) mod 26
  - 25 mod 26
  - 25

Collision-Resistant Hash Functions

- Hash functions for which it is difficult to find collisions are called **collision-resistant**
- A collision-resistant hash function, $H(M)=h$:
  - For any message, *M1*
  - It is difficult to find another message, *M2* such that:
    - *M1* and *M2* are not the same
    - $H(M1)$ and $H(M2)$ are the same

One-Way Hash Functions

- A function, $H(M)=h$, is **one-way** if:
  - Forward direction: given M it is easy to compute h
  - Backward direction: given h it is difficult to compute M
- A one-way hash function:
  - Easy to compute the hash for a given message
  - Hard to determine what message produced a given hash value

Message Digest Functions

- **Message digest** functions are collision-resistant, one-way hash functions:
  - Given a message it is easy to compute its digest
  - Hard to find any message that produces a given digest (one-way)
  - Hard to find any two messages that have the same digest (collision-resistant)

Using Message Digest Functions

- Message digest functions protect data integrity:
  - A company makes some software available for download over the World Wide Web
  - Users want to be sure that they receive a copy that has not been tampered with
  - Solution:
    - The company creates a message digest
    - The digest is transmitted (securely) to users
    - Users compute their own digest for software they receive
    - If the digests match, the software probably has not been altered

Attacks on Message Digests

- Brute-force search for a collision:
  - Goal:
    - Find a message that produces a given digest, *d*
  - Assume:
    - The message digest function is “strong”
    - The message digest function creates *n*-bit digests
  - Approach:
    - Generate random messages and compute digests for them until one is found with digest *d*
    - Approximately 2^n random messages must be tried to find one that hashes to *d*

Attacks on Message Digests (cont)

- Birthday attack (based on the birthday paradox):
  - Goal:
    - Find any two messages that produce the same digest
  - Assume:
    - A “strong”mmessage digest function
    - A message digest function creates *n*-bit digests
  - Approach:
    - Generate random messages and compute digests for them until two are found that produce the same digest
    - Approximately $2^{n/2}$ random messages must be tried to find one that hashes to *d*

The Secure Hash Algorithm

- The Secure Hash Algorithm:
  - Federal Information Processing Standard (FIPS 180-1)
    - 1995 adopted by U.S. government
  - Based on MD4 message digest function
    - Created by Ron Rivest
    - Developed by NIST and the NSA
  - Input: a message of *b* bits
  - Output: a 160-bit message digest

SHA - Overview

- Pad the message
- Initialize constants
- For each 512-bit block (B_1, B_2, B_3, . . ., B_n):
  - Divide B_i into 16 32-bit words ($W_0 – W_{15}$)
  - Compute 64 new 32-bit words (W_{16}, W_{17}, . . ., W_{79})
  - Copy H_0 - H_4 into *A, B, C, D,* and *E*
  - For each W_j ($W_0 – W_{79}$) compute *TEMP* and update *A*-*E*
  - Update H_0 - H_4
- The 160-bit message digest is: $H_0\ H_1\ H_2\ H_3\ H_4$

Motivation for Message Authentication Codes

- Want to use a message digest function to protect files on our computer from viruses:
  - Calculate digests for important files and store them in a table
  - Recompute and check from time to time to verify that the files have not been modified
- Good: if a virus modifies a file the change will be detected since the digest of that file will be different
- Bad: the virus could just compute new digests for modified files and install them in the table

Message Authentication Codes

- **Message authentication code** (MAC): key-dependent message digest function
  - $MAC_K(M) = h$
  - Output, h, is a function of both the hash function and a key, K
- The MAC can only be created or verified by someone who knows K
- Can turn a one-way hash function into a MAC by encrypting the hash value with a symmetric-key cryptosystem

Using MAC

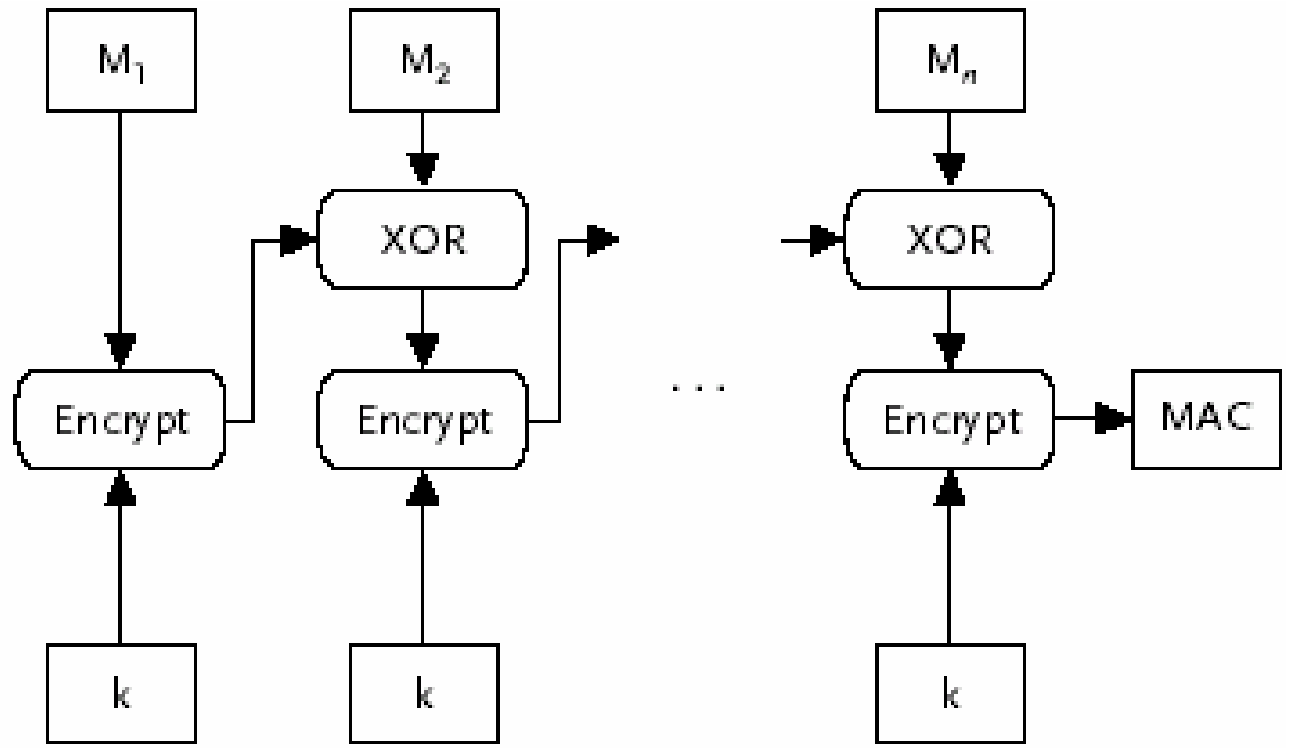
- MAC protects both data integrity and authenticity:
  - Want to use a MAC to protect files on our computer from viruses:
    - Calculate MAC values for important files and store them in a table
    - Recompute and check from time to time to verify that the files haven't been modified
  - Good: if a virus modifies a file the hash of that file will be different
  - Good: virus doesn't know the proper key so it can't install new MACs in the table to cover its tracks

Implementing a MAC

- Can use a block cipher algorithm:
  - Pad the message (if necessary) so that its length is a multiple of the cipher's block size
  - Divide the message into *n* blocks equal in length to the cipher's block size:
    - $m_1, m_2, \ldots, m_n$
  - Choose a key, k
  - Encrypt m_1 with k
  - XOR the result with m_2
  - Encrypt the result with k
  - XOR the result with m_3

    …

Implementing a MAC (cont)



Summary

- Message digest functions are collision-resistant, one-way hash functions
  - Collision-resistant: hard to find two values that produce the same output
  - One-way: hard to determine what input produced a given output
  - Protects the integrity of a digital document
- MAC
  - A message authentication code is a key-dependent message digest function
    - The output is a function of both the hash function and a secret key
    - The MAC can only be created or verified by someone who knows the key
  - Protects the integrity and authenticity of a digital document