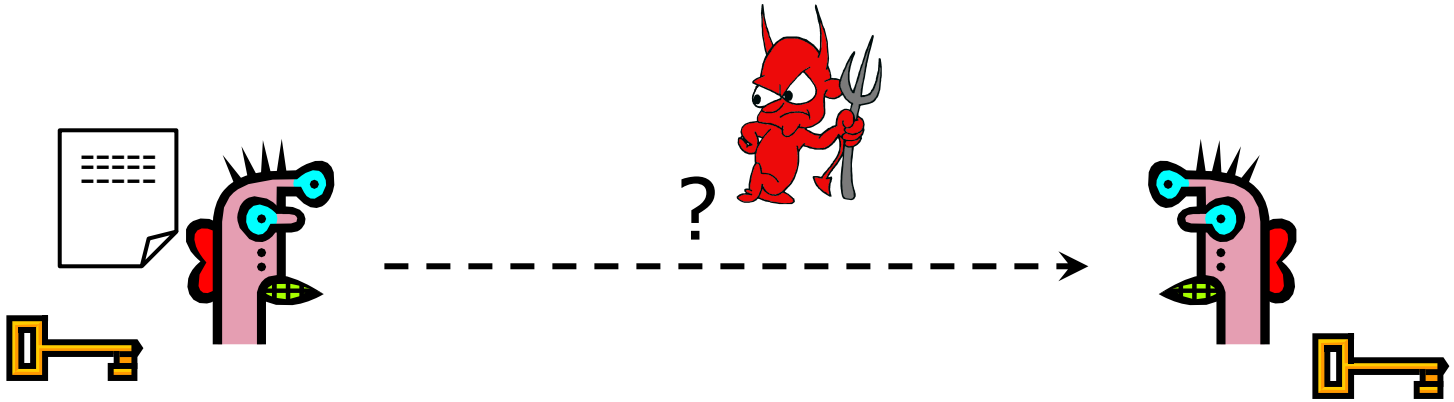
Network Security

H B ACHARYA

Day 2

Encryption Schemes

Basic Problem



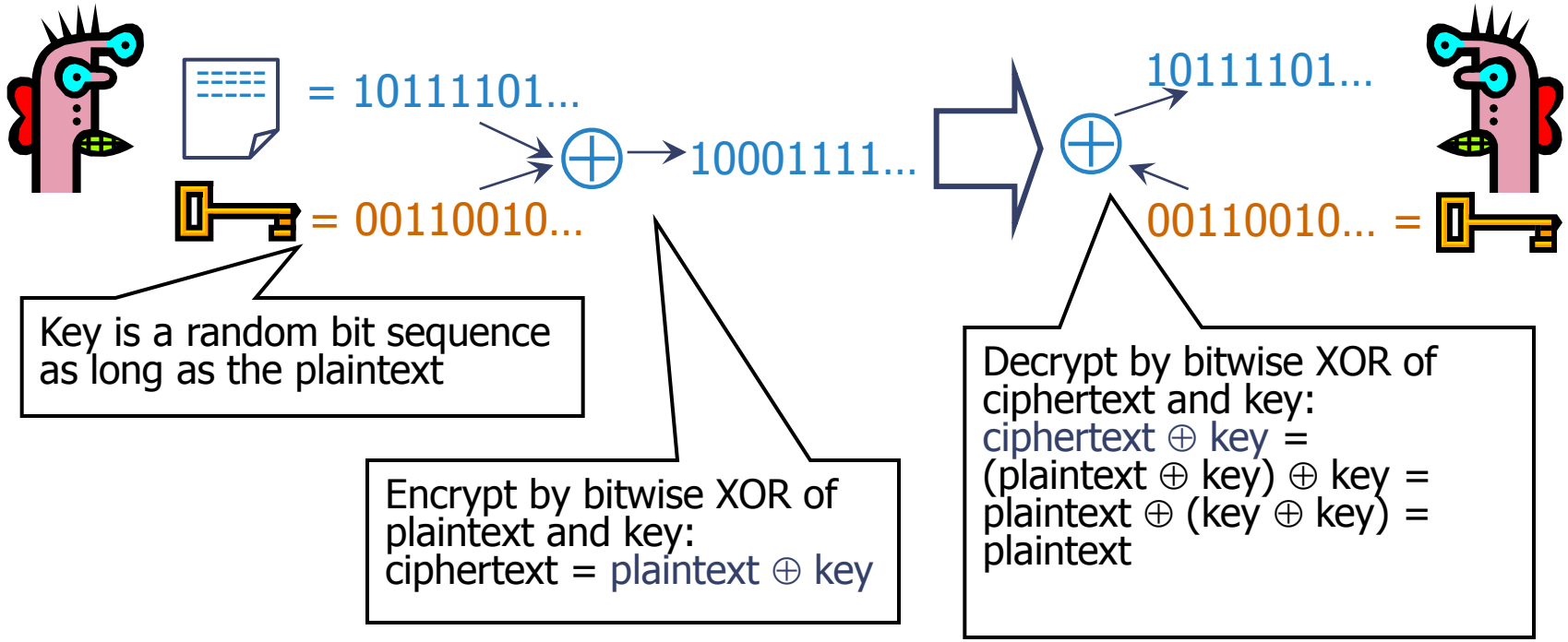
Given: both parties already know the same secret

How is this achieved in practice?

Goal: send a message confidentially

Any communication system that aims to guarantee confidentiality must solve this problem

One-Time Pad (Vernam Cipher)



Cipher achieves perfect secrecy if and only if there are as many possible keys as possible plaintexts, and every key is equally likely (Claude Shannon, 1949)

Advantages of One-Time Pad

Easy to compute

- Encryption and decryption are the same operation
- Bitwise XOR is very cheap to compute

As secure as theoretically possible

- Given a ciphertext, all plaintexts are equally likely, regardless of attacker's computational resources
- …if and only if the key sequence is truly random
  - True randomness is expensive to obtain in large quantities
- …if and only if each key is as long as the plaintext
  - But how do the sender and the receiver communicate the key to each other? Where do they store the key?

Problems with One-Time Pad

Key must be as long as the plaintext

- Impractical in most realistic scenarios
- Still used for diplomatic and intelligence traffic

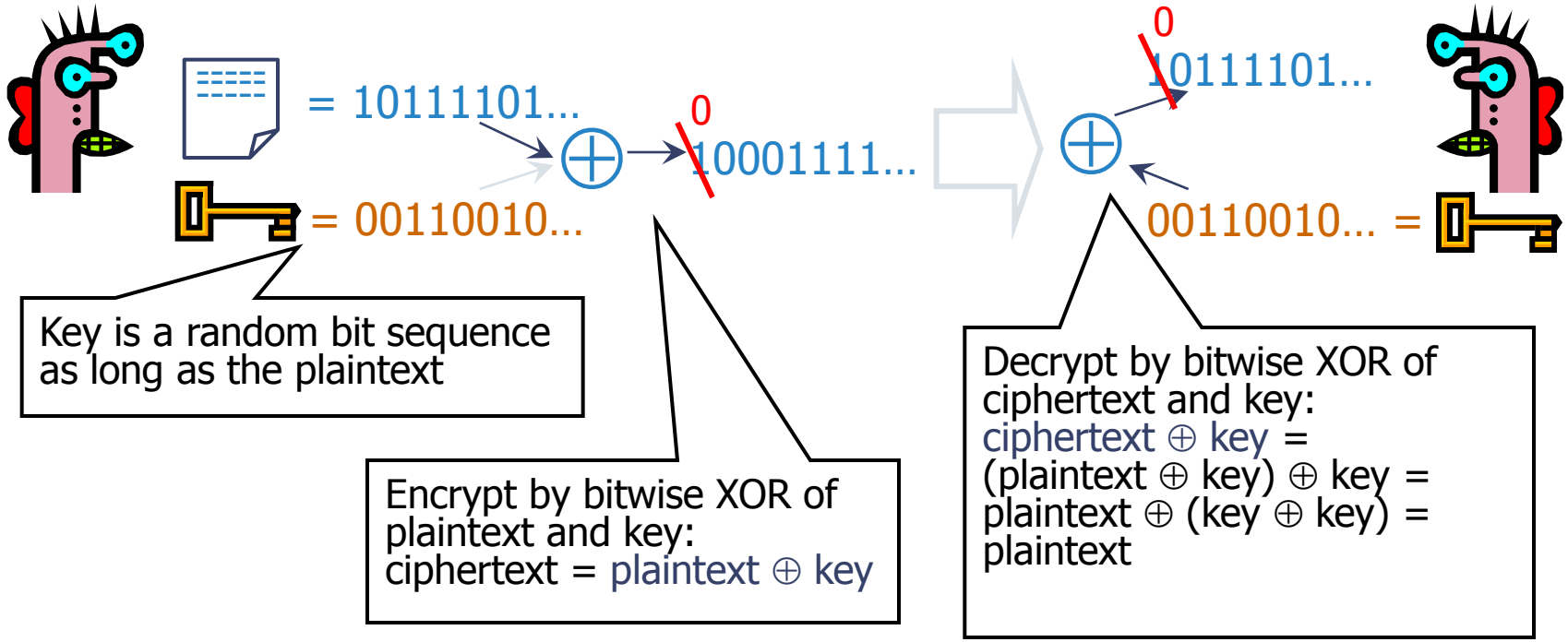
Does not guarantee integrity

- One-time pad only guarantees confidentiality
- Attacker cannot recover plaintext, but can easily change it to something else

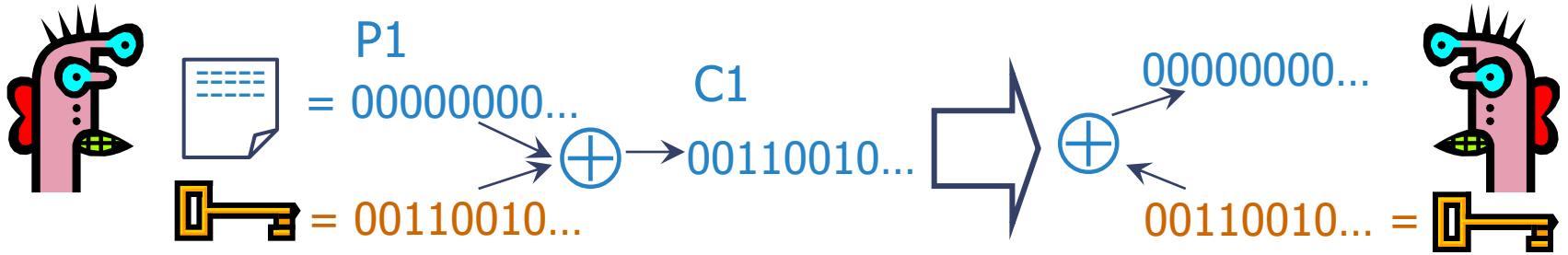
Insecure if keys are reused

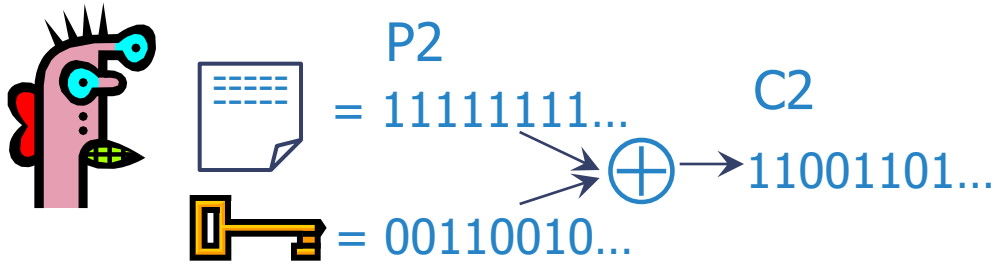
- Attacker can obtain XOR of plaintexts

No Integrity



Dangers of Reuse

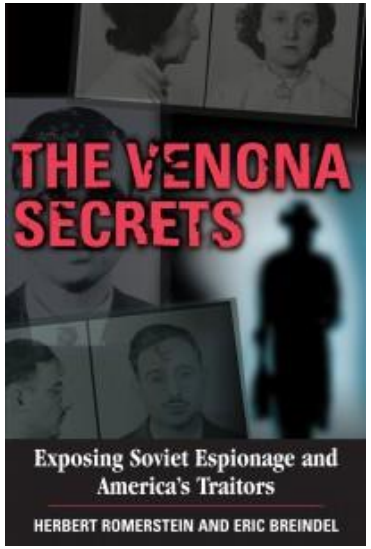




Learn relationship between plaintexts

C1⊕C2 = (P1⊕K)⊕(P2⊕K) =

(P1⊕P2)⊕(K⊕K) = P1⊕P2



Reducing Key Size

What to do when it is infeasible to pre-share huge random keys?

Use special cryptographic primitives:

block ciphers, stream ciphers

- Single key can be re-used (with some restrictions)
- Not as theoretically secure as one-time pad

Block Ciphers

Operates on a single chunk (“block”) of plaintext

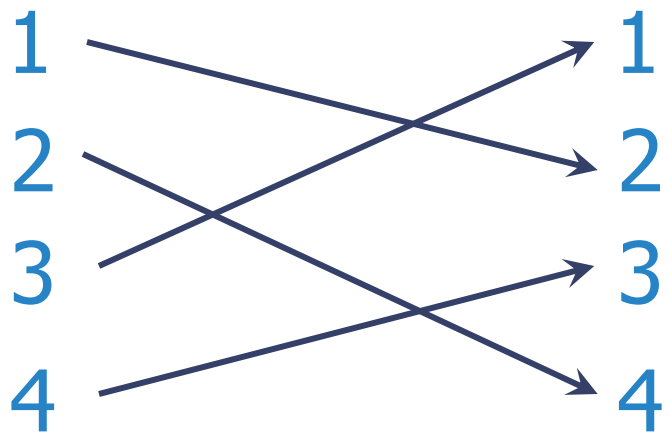
- For example, 64 bits for DES, 128 bits for AES
- Same key is reused for each block (can use short keys)

Result should look like a random permutation

Not impossible to break, just very expensive

- If there is no more efficient algorithm (unproven assumption!), can only break the cipher by brute-force, try-every-possible-key search
- Time and cost of breaking the cipher exceed the value and/or useful lifetime of protected information

Permutation



CODE becomes DCEO

For N-bit input, N! possible permutations

Idea: split plaintext into blocks, for each block use secret key to pick a permutation, rinse and repeat

- Without the key, permutation should “look random”

A Bit of Block Cipher History

Playfair and variants (from 1854 until WWII)

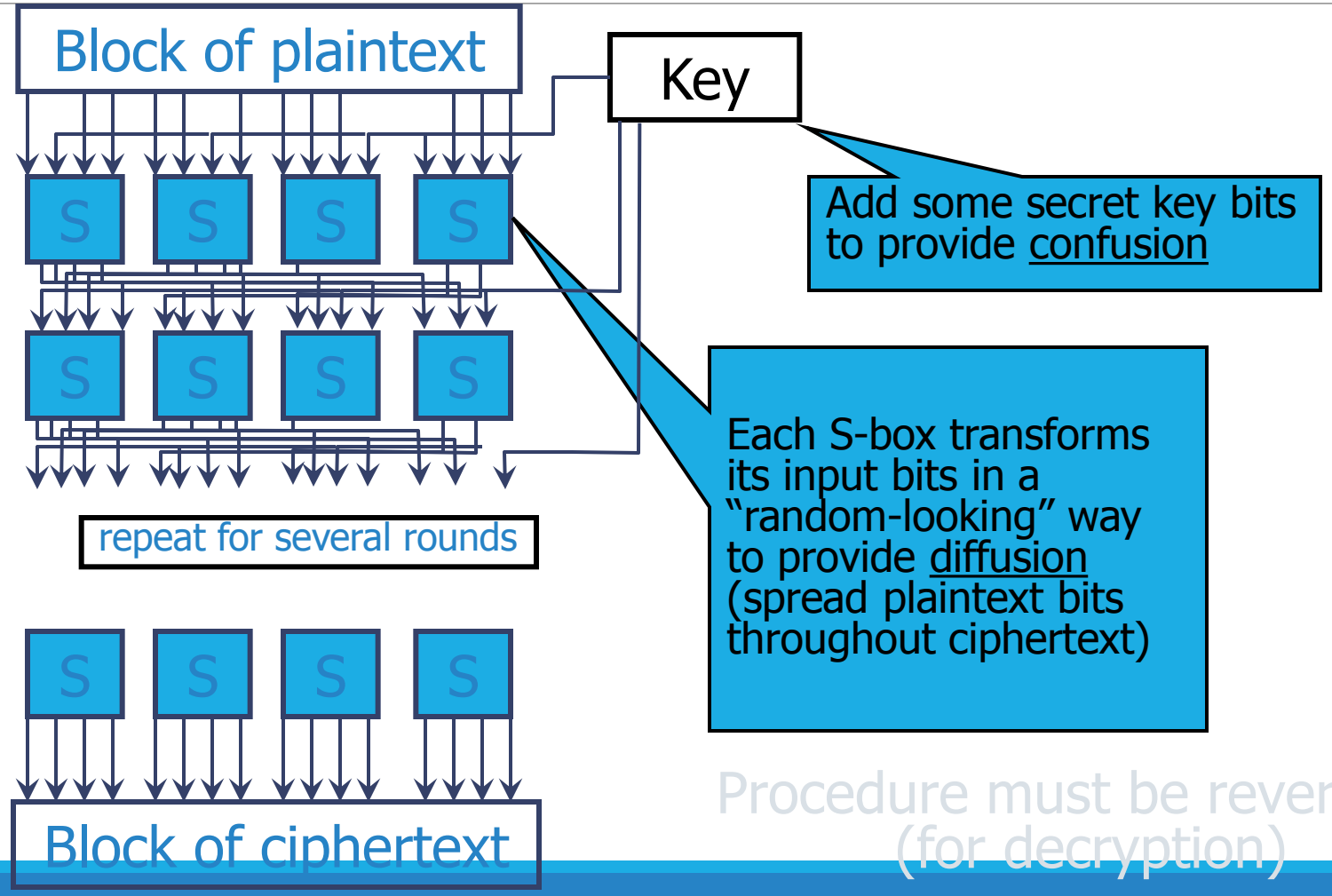
Feistel structure

- “Ladder” structure: split input in half, put one half through the round and XOR with the other half
- After 3 random rounds, ciphertext indistinguishable from a random permutation

DES: Data Encryption Standard

- Invented by IBM, issued as federal standard in 1977
- 64-bit blocks, 56-bit key + 8 bits for parity
- Very widely used (usually as 3DES) until recently
  - 3DES: DES + inverse DES + DES (with 2 or 3 different keys)

DES Operation (Simplified)



Block vs Stream Ciphers

block ciphers process messages in blocks, each of which is then en/decrypted

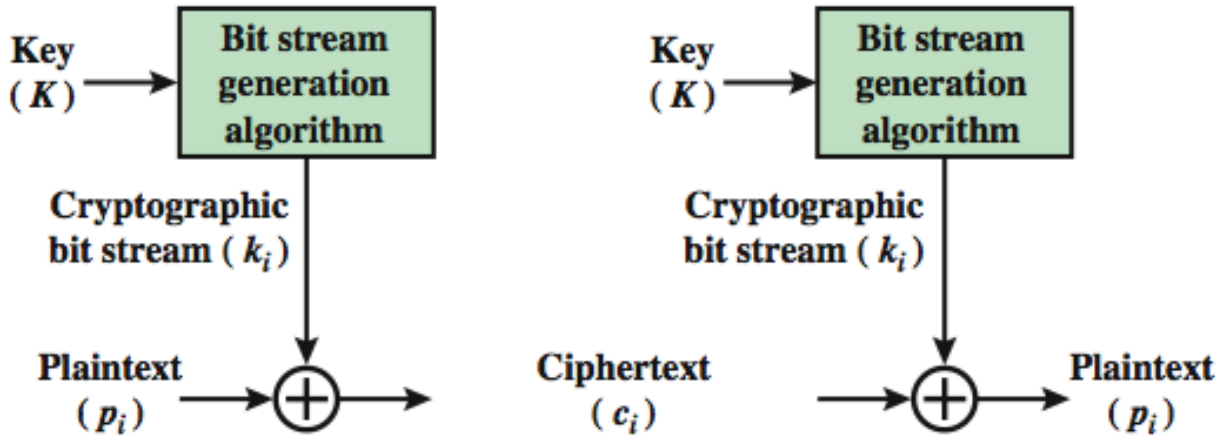
like a substitution on very big characters

- 64-bits or more

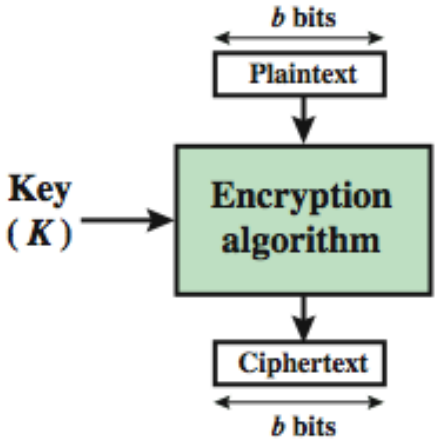
stream ciphers process messages a bit or byte at a time when en/decrypting

many current ciphers are block ciphers

- better analysed
- broader range of applications



(a) Stream Cipher Using Algorithmic Bit Stream Generator



(b) Block Cipher

Block Cipher Principles

Most symmetric block ciphers based on Feistel Cipher Structure

- decrypt ciphertext to recover messages efficiently

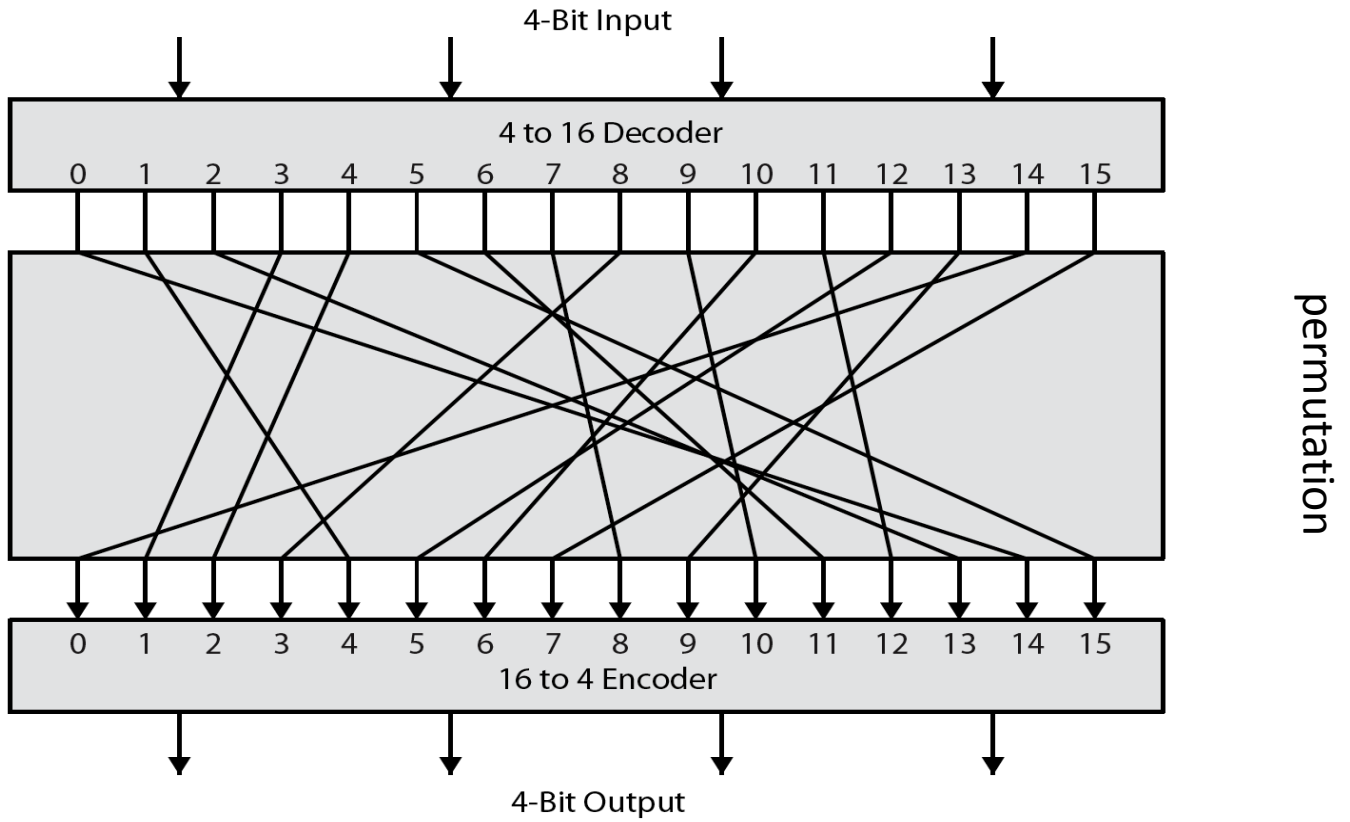
Block ciphers look like an extremely large substitution

- 2^{64} entries for a 64-bit block

instead create from smaller building blocks

using idea of a product cipher

Ideal Block Cipher



Claude Shannon and Substitution-Permutation Ciphers

Claude Shannon introduced idea of substitution-permutation (S-P) networks in 1949 paper

form basis of modern block ciphers

S-P nets are based on the two primitive cryptographic operations seen before:

- substitution (S-box)
- permutation (P-box)

provide confusion & diffusion of message & key

Confusion and Diffusion

cipher needs to completely obscure statistical properties of original message

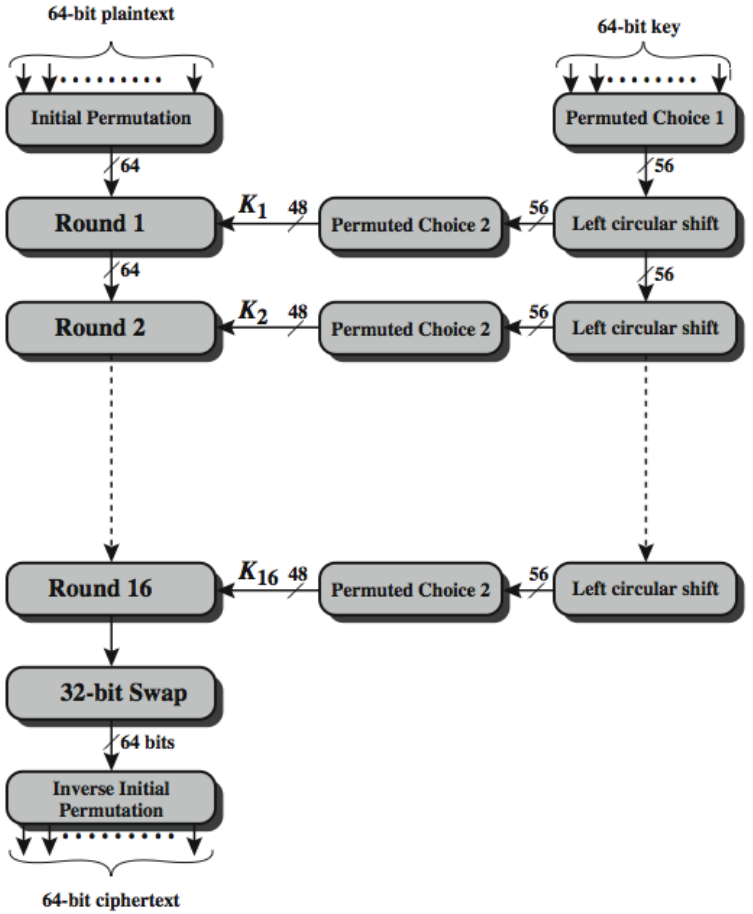
a one-time pad does this

more practically Shannon suggested combining S & P elements to obtain:

diffusion – dissipates statistical structure of plaintext over bulk of ciphertext

confusion – makes relationship between ciphertext and key as complex as possible

DES Encryption Overview



Initial Permutation IP

first step of the data computation

IP reorders the input data bits

even bits to LH half, odd bits to RH half

quite regular in structure (easy in h/w)

no cryptographic value

example:

IP(675a6967 5e5a6b5a) = (ffb2194d 004df6fb)

DES Round Structure

uses two 32-bit L & R halves

as for any Feistel cipher can describe as:

Li = Ri–1

Ri = Li–1 ⊕ F(Ri–1, Ki)

F takes 32-bit R half and 48-bit subkey:

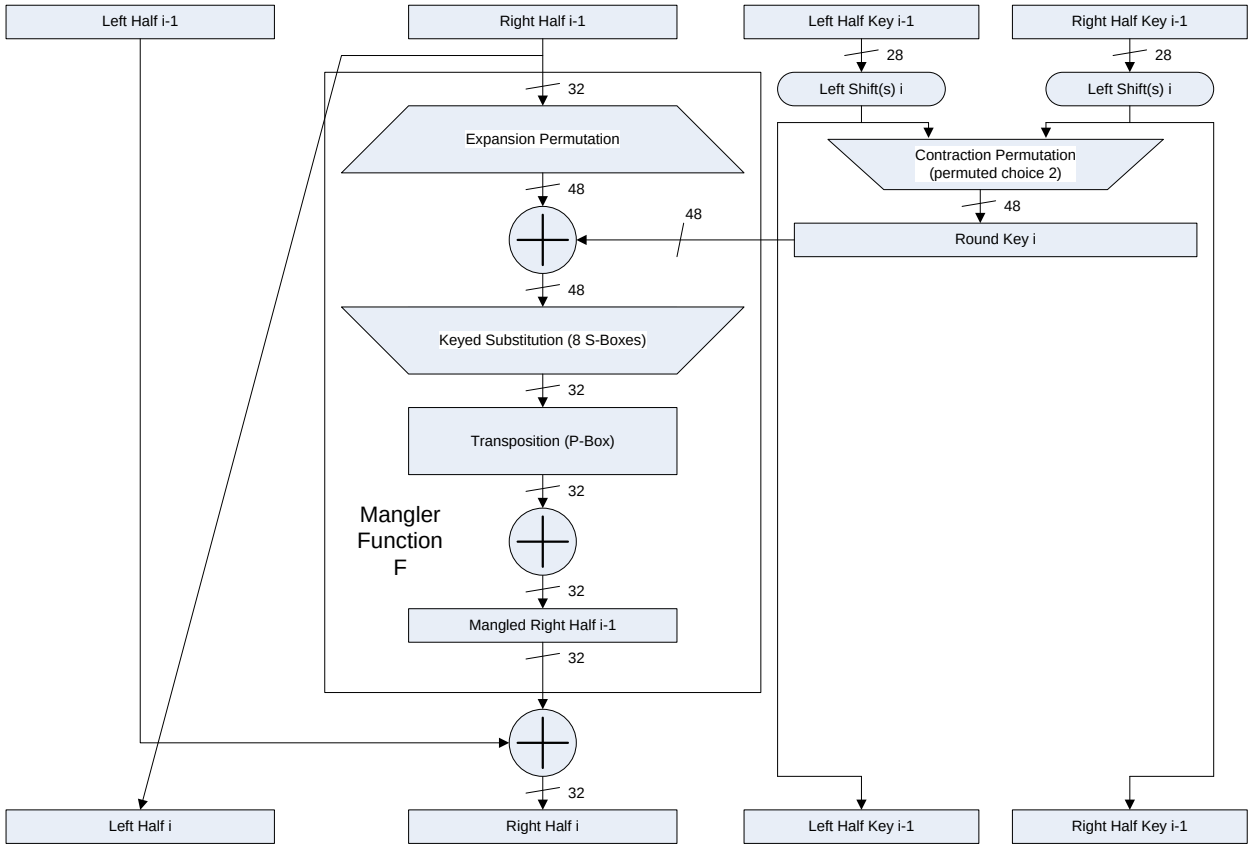
expands R to 48-bits using perm E

adds to subkey using XOR

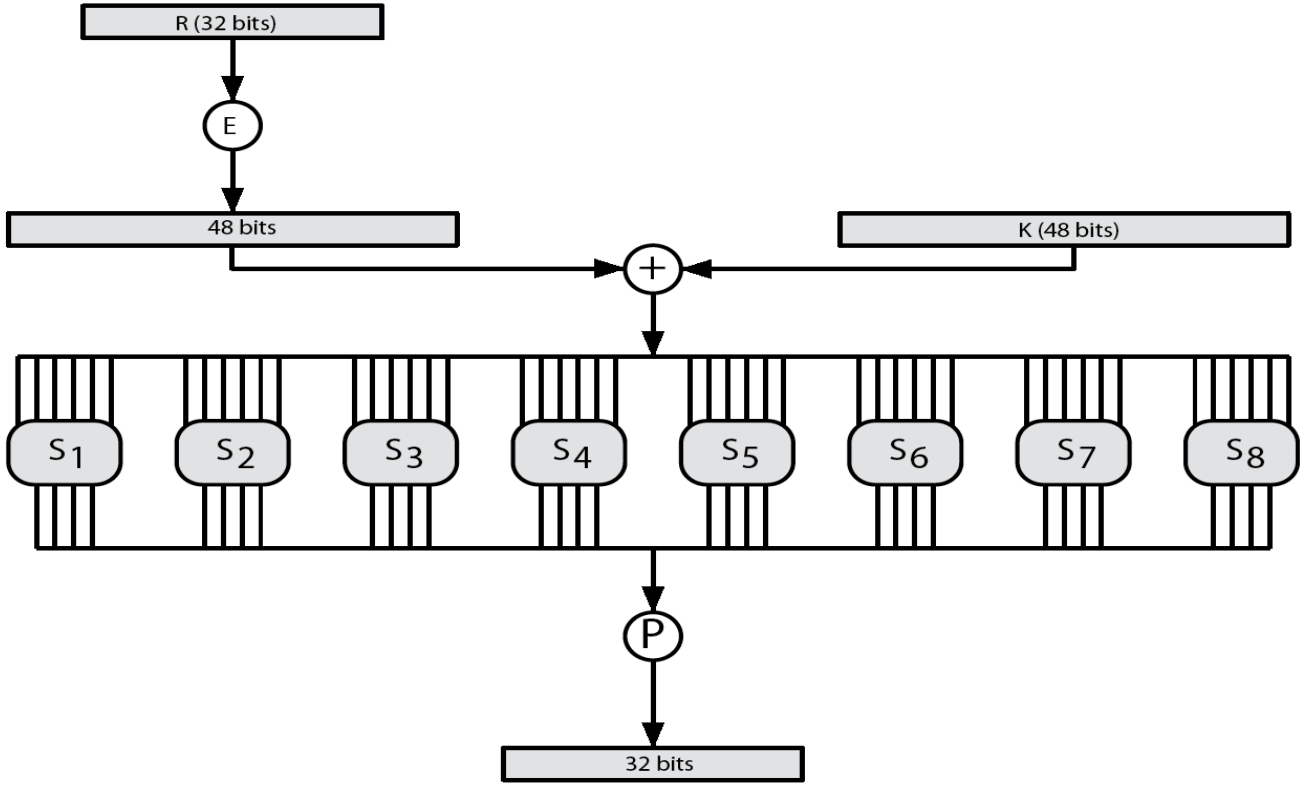
passes through 8 S-boxes to get 32-bit result

finally permutes using 32-bit perm P

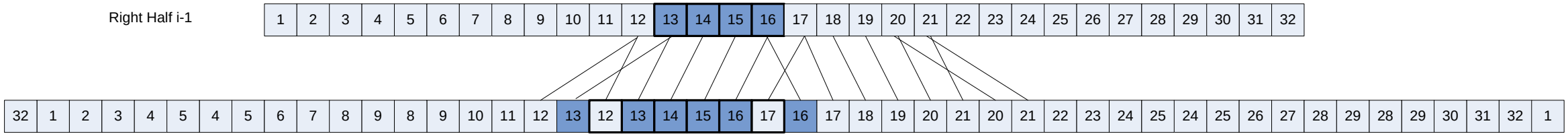
DES Round Structure



DES Round Structure



DES Expansion Permutation



R half expanded to same length as 48-bit subkey

consider R as 8 nybbles (4 bits each)

expansion permutation

copies each nybble into the middle of a 6-bit block

copies the end bits of the two adjacent nybbles into the two end bits of the 6-bit block

Substitution Boxes S

have eight S-boxes which map 6 to 4 bits

each S-box is actually 4 little 4 bit boxes

outer bits 1 & 6 (row bits) select one row of 4

inner bits 2-5 (col bits) are substituted

result is 8 lots of 4 bits, or 32 bits

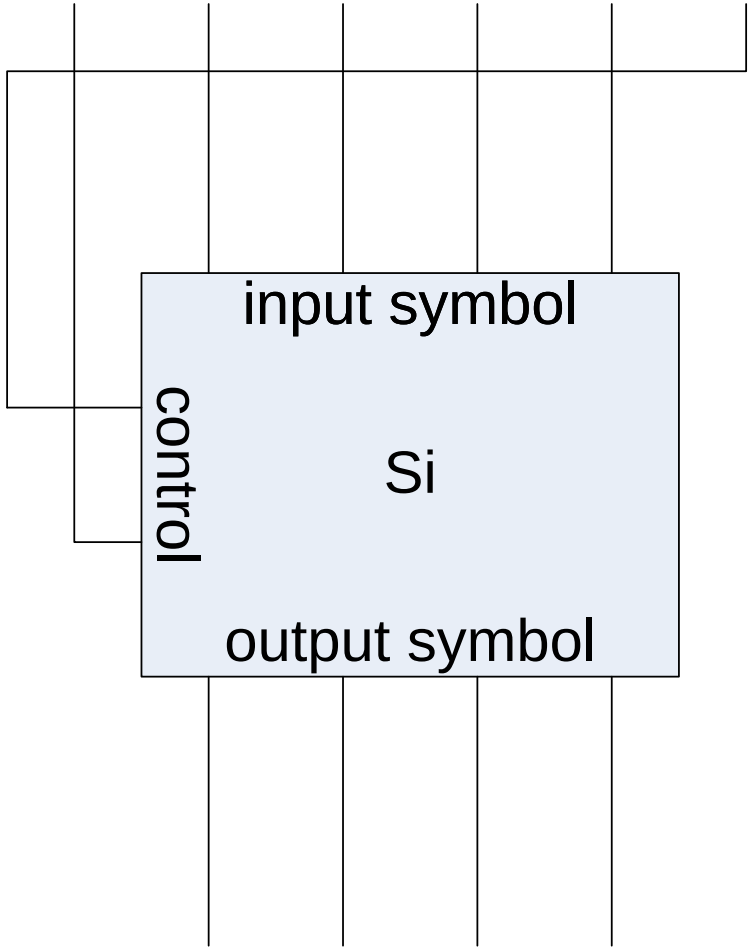
row selection depends on both data & key

feature known as autoclaving (autokeying)

example:

S(18 09 12 3d 11 17 38 39) = 5fd25e03

Substitution Boxes S

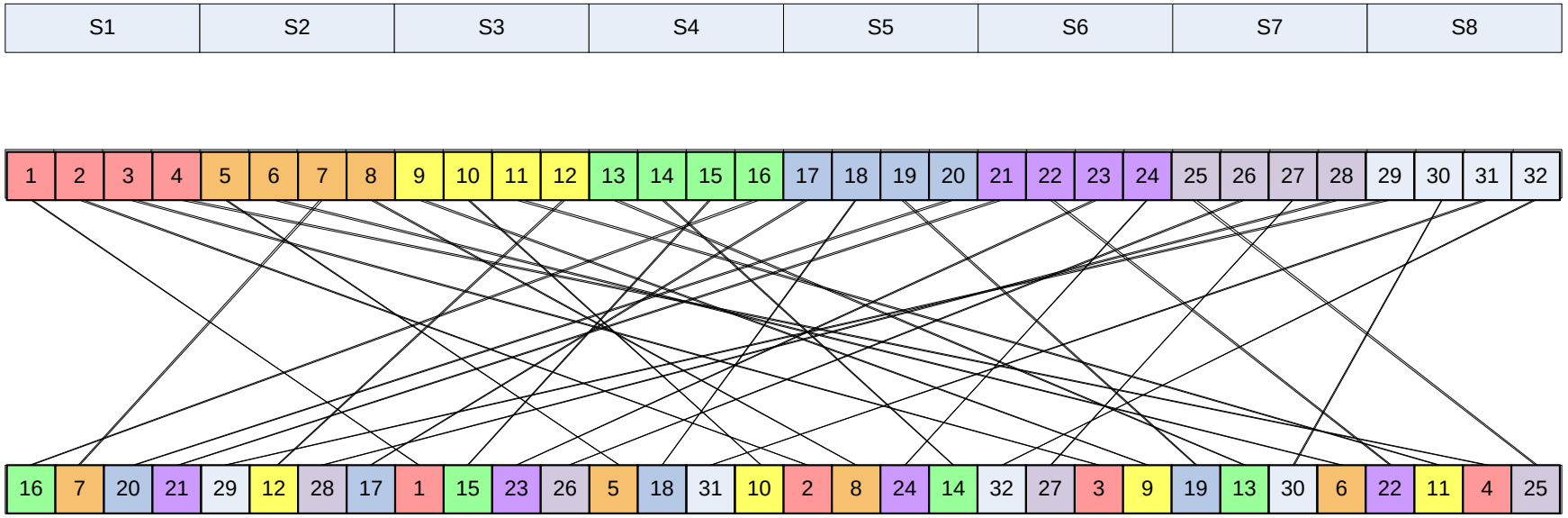


each of the eight s-boxes is different

each s-box reduces 6 bits to 4 bits

so the 8 s-boxes implement the 48-bit to 32-bit contraction substitution

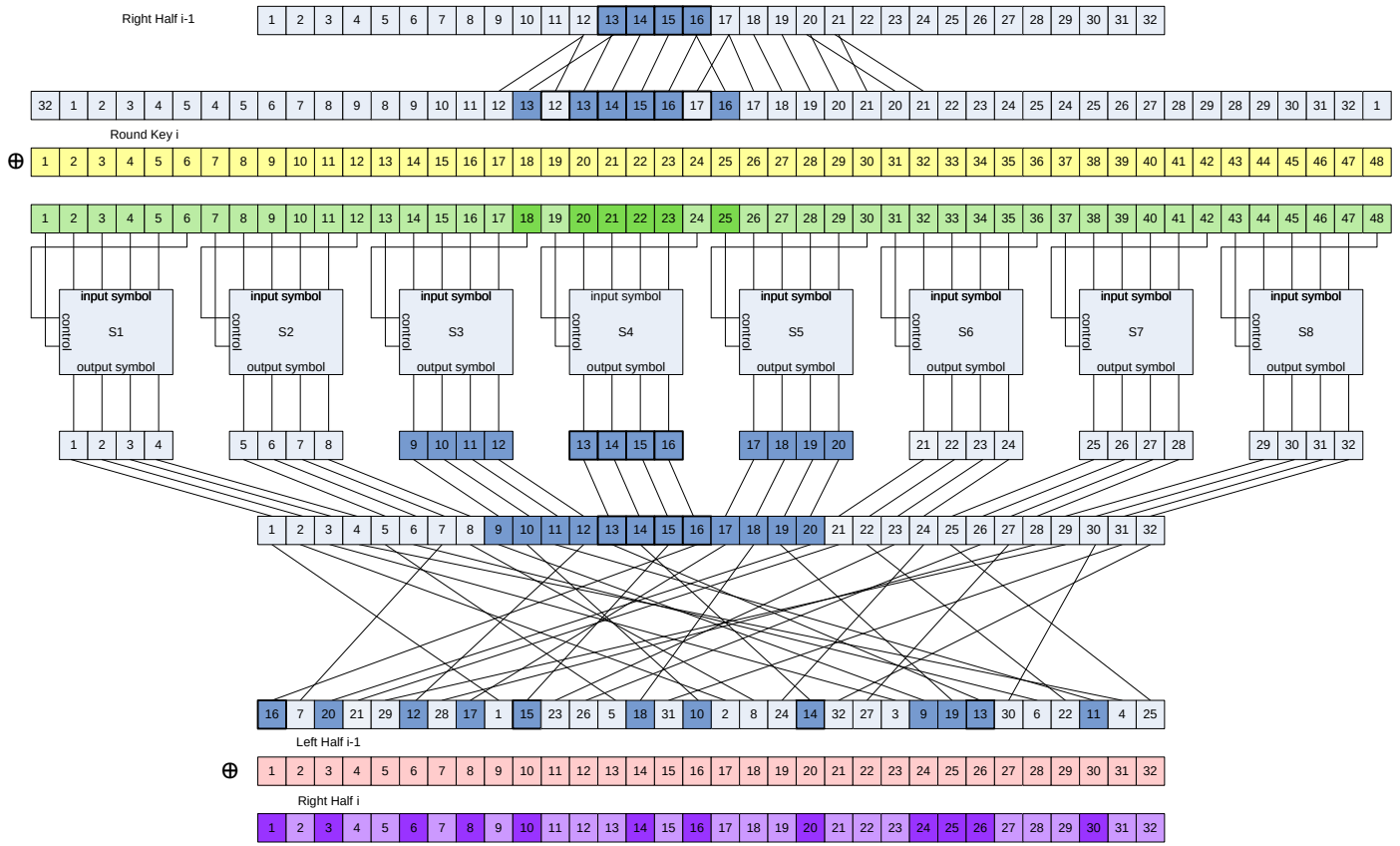
Permutation Box P



P-box at end of each round

Increases diffusion/avalanche effect

DES Round in Full



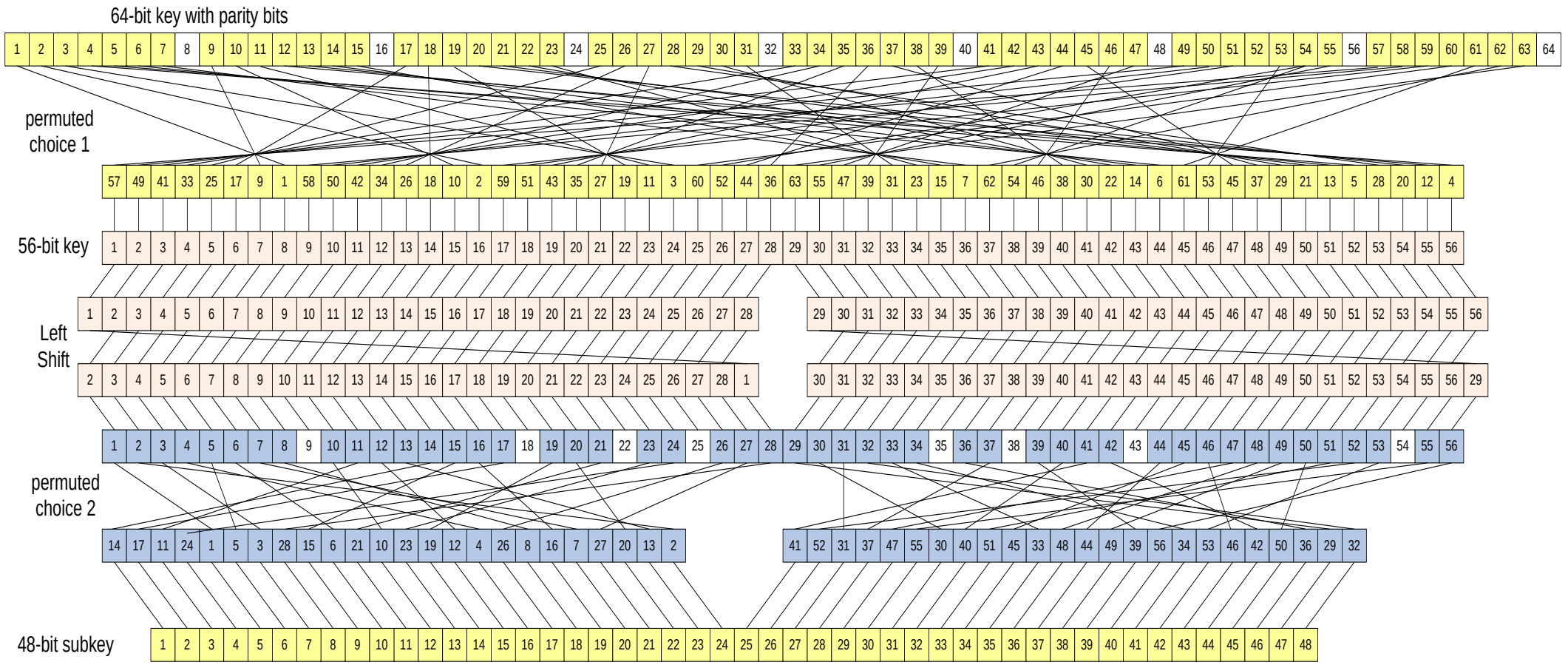
DES Key Schedule

forms subkeys used in each round

- initial permutation of the key (PC1) which selects 56-bits in two 28-bit halves
- 16 stages consisting of:
  - rotating each half separately either 1 or 2 places depending on the key rotation schedule K
  - selecting 24-bits from each half & permuting them by PC2 for use in round function F

note practical use issues in h/w vs s/w

DES Key Schedule



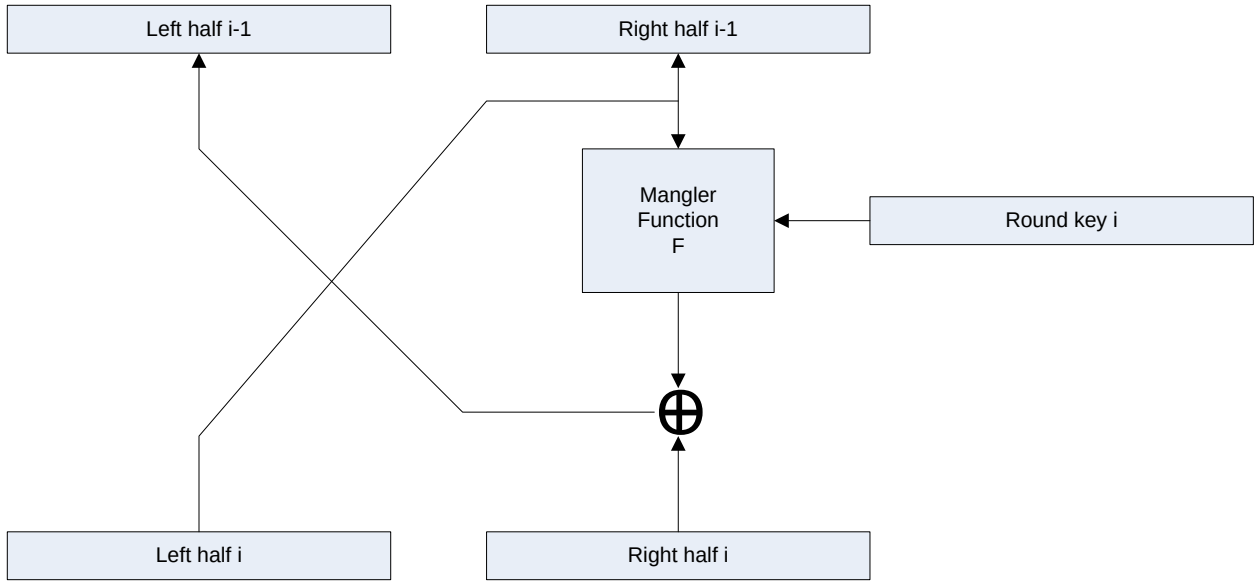
DES Decryption

decrypt must unwind steps of data computation

with Feistel design, do encryption steps again using subkeys in reverse order (SK16 … SK1)

- IP undoes final FP step of encryption
- 1st round with SK16 undoes 16th encrypt round
- ….
- 16th round with SK1 undoes 1st encrypt round
- then final FP undoes initial encryption IP
- thus recovering original data value

DES Round Decryption



Decryption

DES Example

Round	K_i	L_i	R_i
IP		5a005a00	3cf03c0f
1	1e030f03080d2930	3cf03c0f	bad22845
2	0a31293432242318	bad22845	99e9b723
3	23072318201d0c1d	99e9b723	0bae3b9e
4	05261d3824311a20	0bae3b9e	42415649
5	3325340136002c25	42415649	18b3fa41
6	123a2d0d04262a1c	18b3fa41	9616fe23
7	021f120b1c130611	9616fe23	67117cf2
8	1c10372a2832002b	67117cf2	c11bfc09
9	04292a380c341f03	c11bfc09	887fbc6c
10	2703212607280403	887fbc6c	600f7e8b
11	2826390c31261504	600f7e8b	f596506e
12	12071c241a0a0f08	f596506e	738538b8
13	300935393c0d100b	738538b8	c6a62c4e
14	311e09231321182a	c6a62c4e	56b0bd75
15	283d3e0227072528	56b0bd75	75e8fd8f
16	2921080b13143025	75e8fd8f	25896490
IP^{-1}		da02ce3a	89ecac3b

Avalanche Effect

key desirable property of encryption alg

where a change of one input or key bit results in changing approx half output bits

making attempts to “home-in” by guessing keys impossible

DES exhibits strong avalanche

Avalanche in DES

Round		δ
	02468aceeca86420 12468aceeca86420	1
1	3cf03c0fbad22845 3cf03c0fbad32845	1
2	bad2284599e9b723 bad3284539a9b7a3	5
3	99e9b7230bae3b9e 39a9b7a3171cb8b3	18
4	0bae3b9e42415649 171cb8b3ccaca55e	34
5	4241564918b3fa41 ccaca55ed16c3653	37
6	18b3fa419616fe23 d16c3653cf402c68	33
7	9616fe2367117cf2 cf402c682b2cefbc	32
8	67117cf2c11bfc09 2b2cefbc99f91153	33

Round		δ
9	c11bfc09887fbc6c 99f911532eed7d94	32
10	887fbc6c600f7e8b 2eed7d94d0f23094	34
11	600f7e8bf596506e d0f23094455da9c4	37
12	f596506e738538b8 455da9c47f6e3cf3	31
13	738538b8c6a62c4e 7f6e3cf34bc1a8d9	29
14	c6a62c4e56b0bd75 4bc1a8d91e07d409	33
15	56b0bd7575e8fd8f 1e07d4091ce2e6dc	31
16	75e8fd8f25896490 1ce2e6dc365e5f59	32
IP^{-1}	da02ce3a89ecac3b 057cde97d7683f2a	32

Strength of DES – Key Size

56-bit keys have 256 = 7.2 x 1016 values

brute force search looks hard

recent advances have shown is possible

- in 1997 on Internet in a few months
- in 1998 on dedicated h/w (EFF) in a few days
- in 1999 above combined in 22hrs!

still must be able to recognize plaintext

must now consider alternatives to DES

Strength of DES – Analytic Attacks

now have several analytic attacks on DES

these utilise some deep structure of the cipher

- by gathering information about encryptions
- can eventually recover some/all of the sub-key bits
- if necessary then exhaustively search for the rest

generally these are statistical attacks

- differential cryptanalysis
- linear cryptanalysis
- related key attacks

Strength of DES – Timing Attacks

attacks actual implementation of cipher

use knowledge of consequences of implementation to derive information about some/all subkey bits

specifically use fact that calculations can take varying times depending on the value of the inputs to it

particularly problematic on smartcards

Differential Cryptanalysis

one of the most significant recent (public) advances in cryptanalysis

known by NSA in 70's cf DES design

Murphy, Biham & Shamir published in 90’s

powerful method to analyse block ciphers

used to analyse most current block ciphers with varying degrees of success

DES reasonably resistant to it, cf Lucifer

Differential Cryptanalysis

a statistical attack against Feistel ciphers

uses cipher structure not previously used

design of S-P networks has output of function f influenced by both input & key

hence cannot trace values back through cipher without knowing value of the key

differential cryptanalysis “eliminates” key by using differenced input

Differential Cryptanalysis Compares Pairs of Encryptions

differential cryptanalysis compares two related pairs of encryptions

with a known difference in the input

searching for a known difference in output

when same subkeys are used

Differential Cryptanalysis Compares Pairs of Encryptions

Let m_{i-1} be the left half of the input to round i, and m_i be the right half

$$\begin{aligned}\Delta m_{i+1} &= m_{i+1} \oplus m'_{i+1} \\ &= [m_{i-1} \oplus \mathrm{f}(m_i, K_i)] \oplus [m'_{i-1} \oplus \mathrm{f}(m'_i, K_i)] \\ &= \Delta m_{i-1} \oplus [\mathrm{f}(m_i, K_i) \oplus \mathrm{f}(m'_i, K_i)]\end{aligned}$$

$f(m_i,K_i) = P(S(K_i \text{ XOR } E(m_i)))$, where P is the PBox transposition, S is the Sbox substitution, and E is the expansion perm.

Differential Cryptanalysis Takes Advantage of Linearity

- $f(m_i,K_i) = P(S(K_i \text{ XOR } E(m_i)))$,
  where P is the PBox transposition, S is the Sbox substitution, and E is the expansion perm.

- So $E(m_i) \text{ XOR } E(m'_i) = E(m_i \text{ XOR } m'_i)$,
  the expansion permutation preserves differences (E is linear)

- XOR with K_i also preserves differences

- E changes the input in a known way, so difference changes in known way

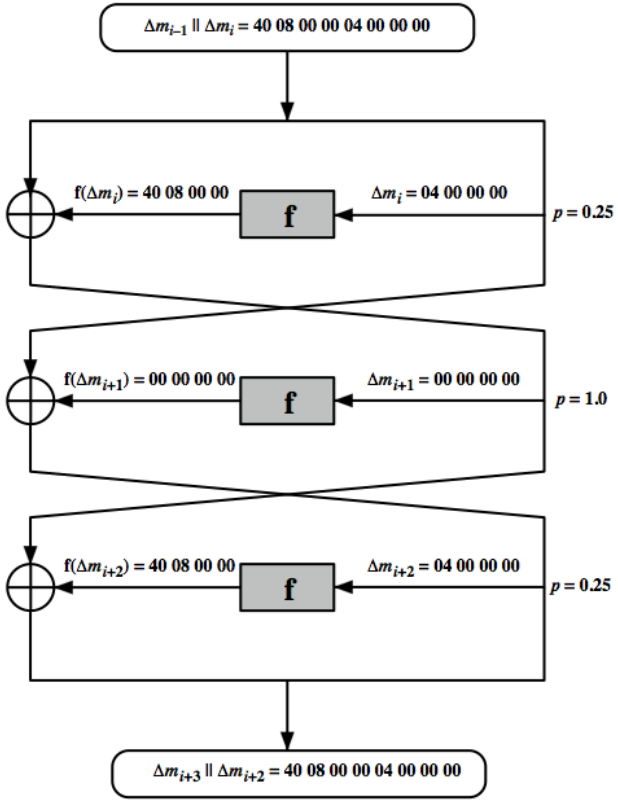
Differential Cryptanalysis Takes Advantage of Non-Uniformity

- For all pairs of inputs with the same difference, compute differences in output
- Build a table with Δx as row index and Δy as column index
  - frequency in cells T(Δx,Δy = # times inputs x and x’ have outputs y and y’
  - with Δx = x XOR x’ and Δy = y XOR y’
- Rows have non-uniformity, so some output differences are more likely than others for a given input difference

Differential Cryptanalysis

- have some input difference giving some output difference with probability p
- if find instances of some higher probability input / output difference occurring
- can infer subkey that was used in round
- then must iterate process over many rounds (with decreasing probabilities)

Differential Cryptanalysis



Differential Cryptanalysis

- perform attack by repeatedly encrypting plaintext pairs with known input XOR until obtain desired output XOR
- when found
  - if intermediate rounds match required XOR have a **right pair**
  - if not then have a **wrong pair**, relative ratio is S/N for attack
- can then deduce keys values for the rounds
  - right pairs suggest same key bits
  - wrong pairs give random values
- for large numbers of rounds, probability is so low that more pairs are required than exist with 64-bit inputs
- Biham and Shamir have shown how a 13-round iterated characteristic can break the full 16-round DES

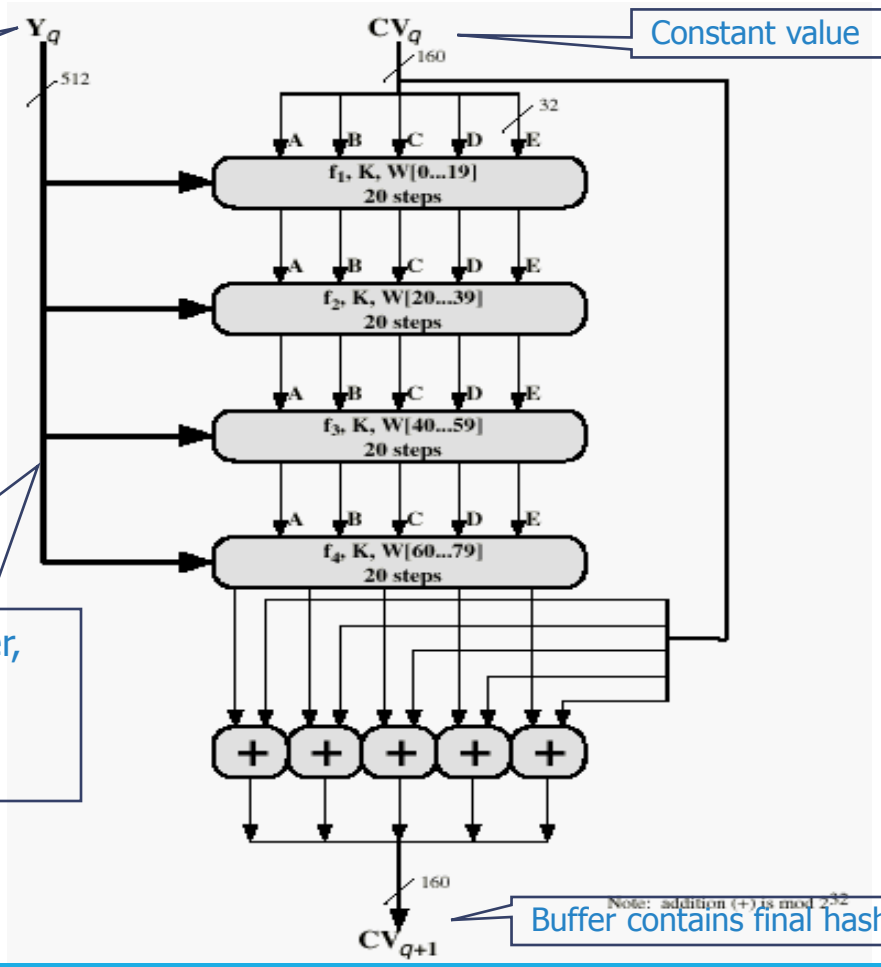
Linear Cryptanalysis

- find linear approximations with prob p != ½
  - $P[i_1,i_2,...,i_a] \oplus C[j_1,j_2,...,j_b] = K[k_1,k_2,...,k_c]$
  - where i_a,j_b,k_c are bit locations in P,C,K
- gives linear equation for key bits
- get one key bit using max likelihood alg
- using a large number of trial encryptions
- effectiveness given by: $|p-1/2|$

DES Design Criteria

- as reported by Coppersmith in [COPP94]
- 7 criteria for S-boxes provide for
  - non-linearity
  - resistance to differential cryptanalysis
  - good confusion
- 3 criteria for permutation P provide for
  - increased diffusion

Remember SHA-1?



Current **message block**

Constant value

Very similar to a block cipher, with message itself used as the key for each round

Buffer contains final hash value

Advanced Encryption Standard (AES)

US federal standard as of 2001

Based on the Rijndael algorithm

128-bit blocks, keys can be 128, 192 or 256 bits

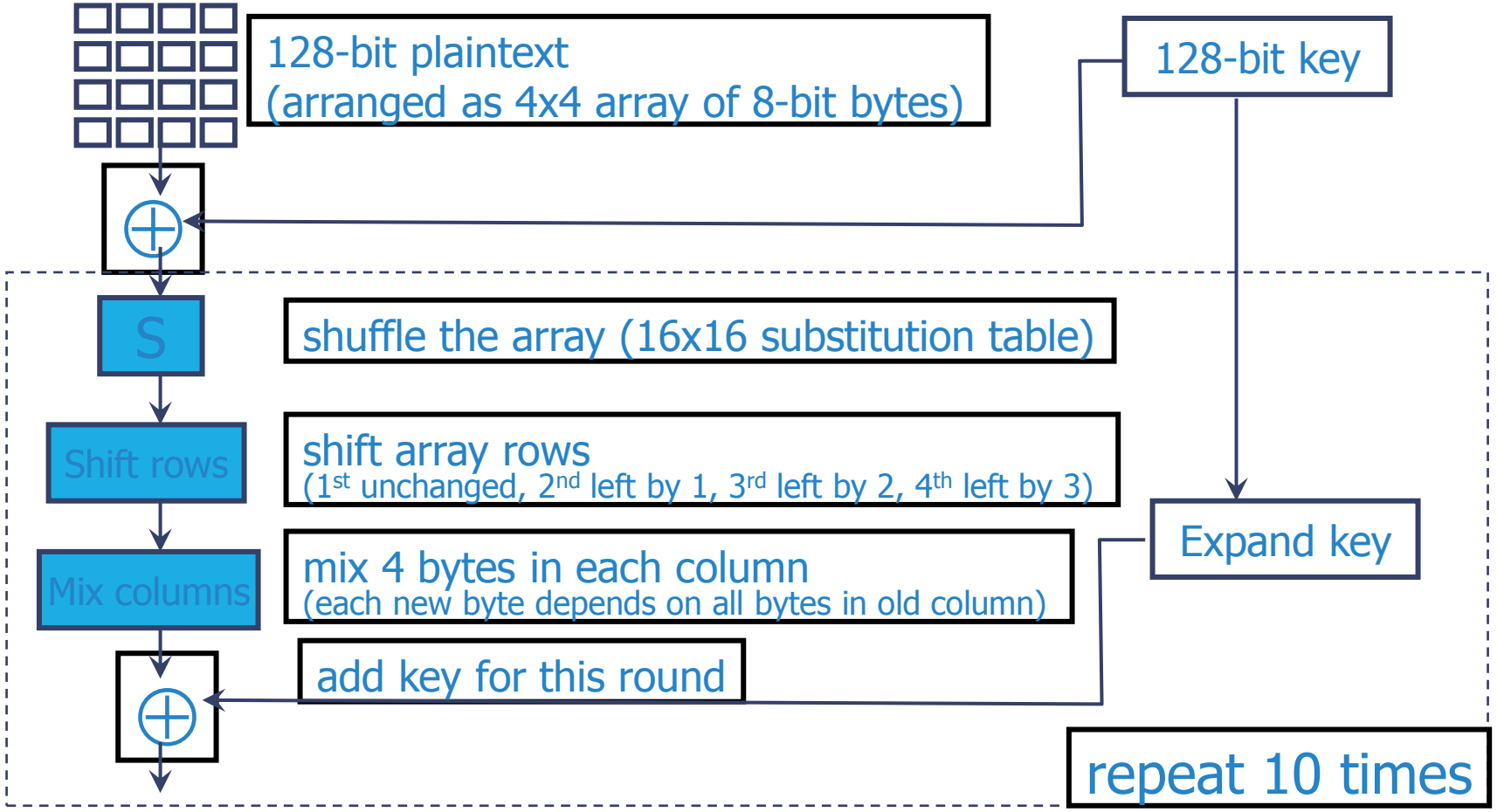
Unlike DES, does not use Feistel structure

- The entire block is processed during each round

Design uses some clever math

- See section 8.5 of the textbook for a concise summary

Basic Structure of Rijndael



AES Origins

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 in 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

The AES Cipher - Rijndael

designed by Rijmen-Daemen in Belgium

has 128/192/256 bit keys, 128 bit data

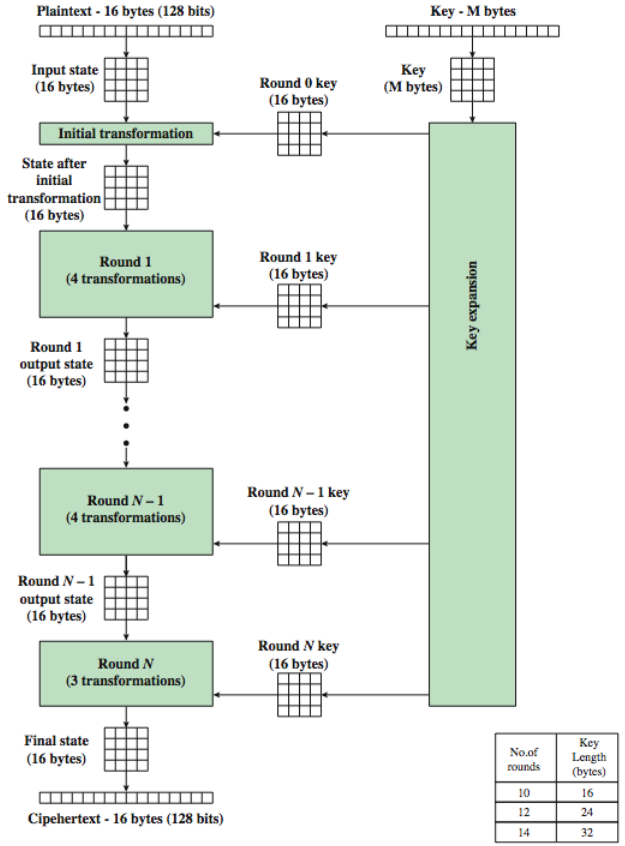
an **iterative** rather than **Feistel** cipher

- processes data as block of 4 columns of 4 bytes
- operates on entire data block in every round

designed to have:

- resistance against known attacks
- speed and code compactness on many CPUs
- design simplicity

AES Encryption Process

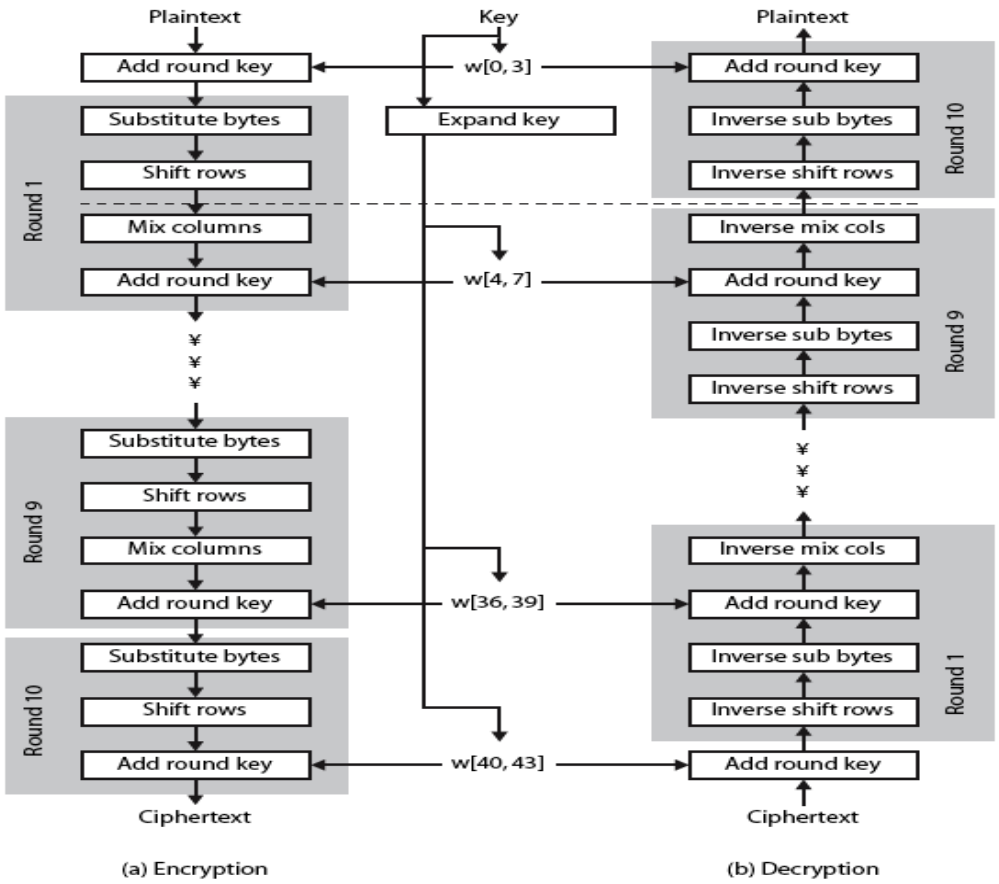


No.of rounds	Key Length (bytes)
10	16
12	24
14	32

AES Structure

- data block of 4 columns of 4 bytes is state
- key is expanded to array of words
- has 9/11/13 rounds in which state undergoes:
  - byte substitution (1 S-box used on every byte)
  - shift rows (permute bytes between groups/columns)
  - mix columns (subs using matrix multiply of groups)
  - 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

AES Structure



Some Comments on AES

1. an **iterative** rather than **Feistel** cipher
2. key expanded into array of 32-bit words
    1. four words form round key in each round
3. 4 different stages are used as shown
4. has a simple structure
5. only AddRoundKey uses key
6. AddRoundKey a form of Vernam cipher
7. each stage is easily reversible
8. decryption uses keys in reverse order
9. decryption does recover plaintext
10. final round has only 3 stages

Substitute Bytes

a simple substitution of each byte

uses one table of 16x16 bytes containing a permutation of all 256 8-bit values

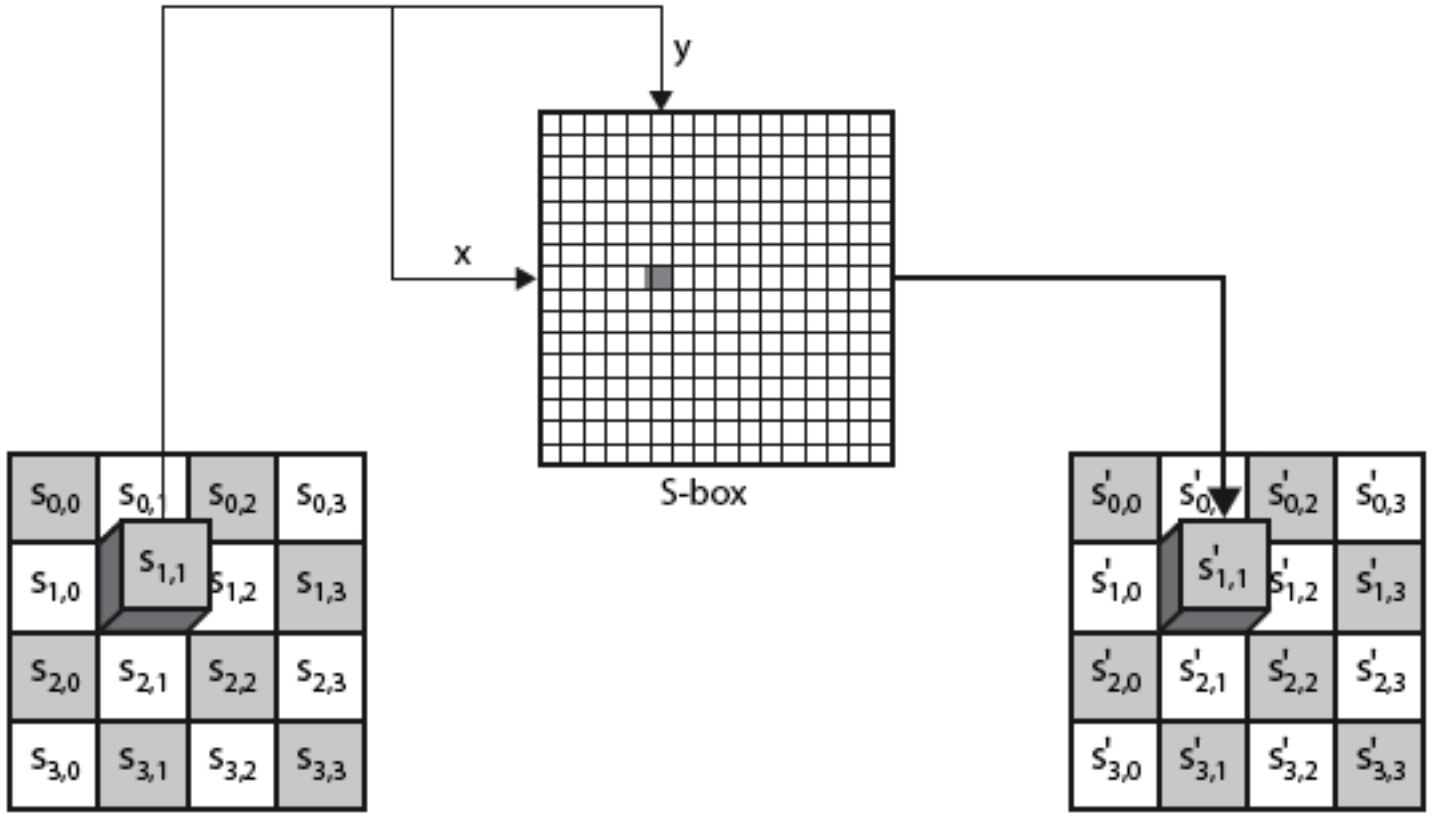
each byte of state is replaced by byte indexed by row (left 4-bits) & column (right 4-bits)

- eg. byte {95} is replaced by byte in row 9 column 5
- which has value {2A}

S-box constructed using defined transformation of values in $GF(2^8)$

designed to be resistant to all known attacks

Substitute Bytes



Substitute Bytes Example

EA	04	65	85
83	45	5D	96
5C	33	98	B0
F0	2D	AD	C5

→

87	F2	4D	97
EC	6E	4C	90
4A	C3	46	E7
8C	D8	95	A6

Shift Rows

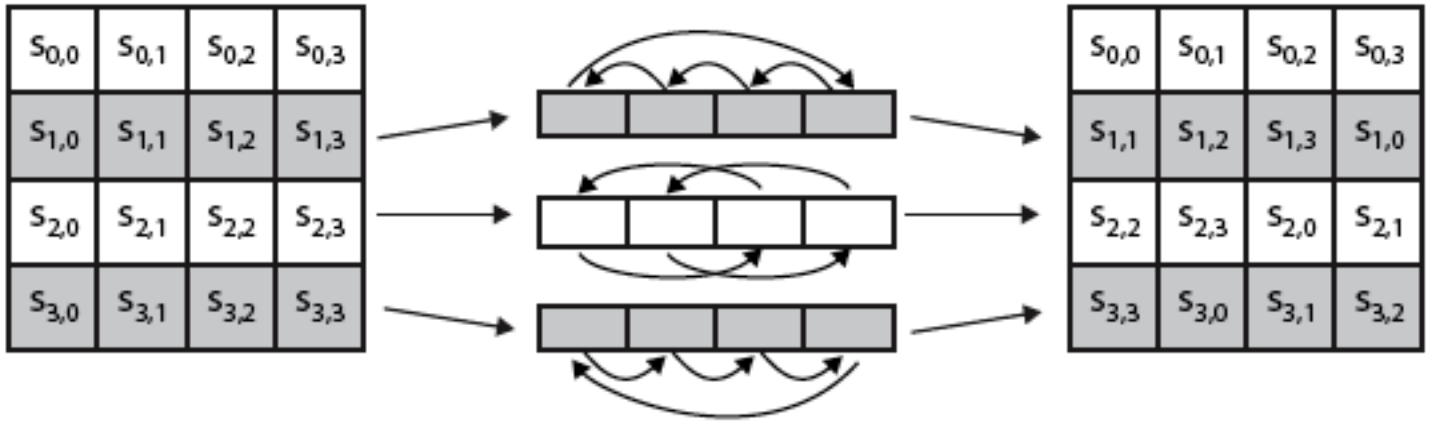
a circular byte shift in each each

- 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

Shift Rows



87	F2	4D	97
EC	6E	4C	90
4A	C3	46	E7
8C	D8	95	A6

→

87	F2	4D	97
6E	4C	90	EC
46	E7	4A	C3
A6	8C	D8	95

Mix Columns

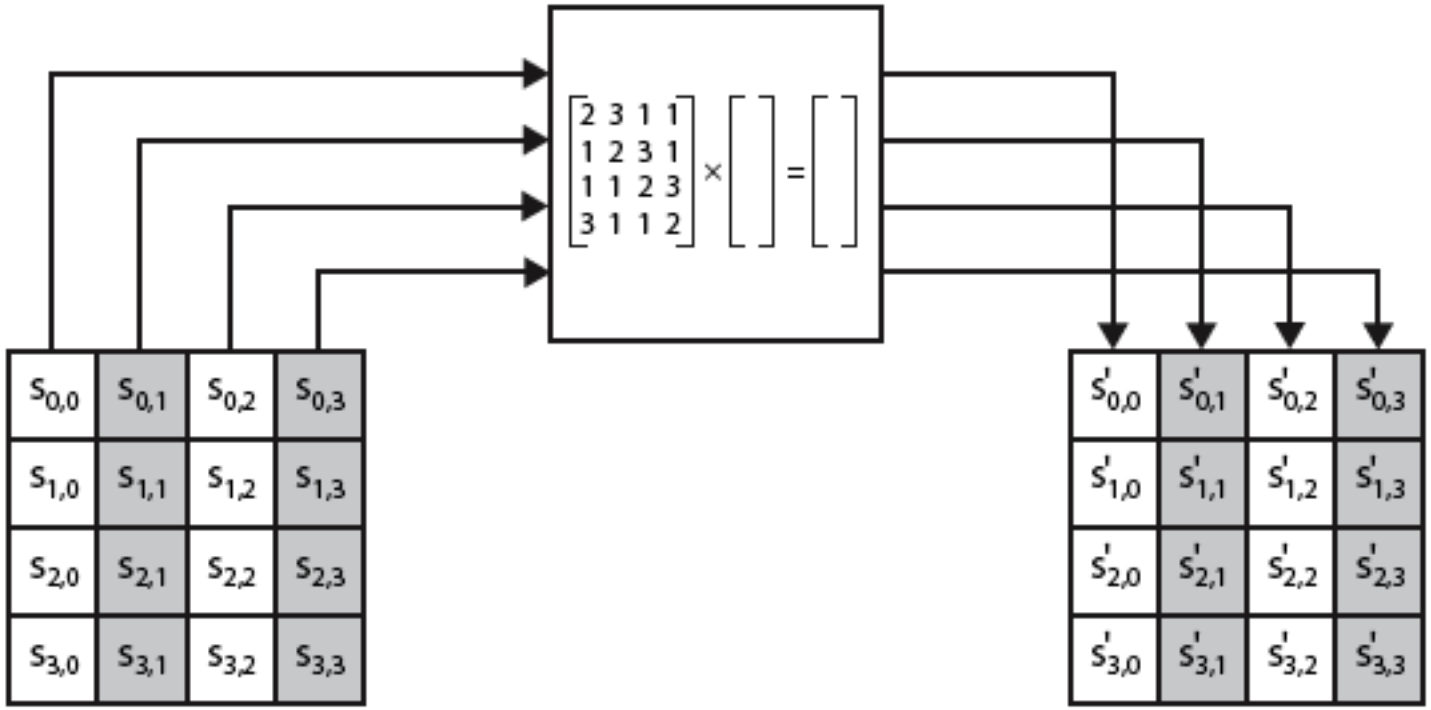
each column is processed separately

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) =$x^8+x^4+x^3+x+1$

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

Mix Columns


2 3 1 1
1 2 3 1
1 1 2 3
3 1 1 2
×
=
$s_{0,0}$ $s_{0,1}$ $s_{0,2}$ $s_{0,3}$
$s_{1,0}$ $s_{1,1}$ $s_{1,2}$ $s_{1,3}$
$s_{2,0}$ $s_{2,1}$ $s_{2,2}$ $s_{2,3}$
$s_{3,0}$ $s_{3,1}$ $s_{3,2}$ $s_{3,3}$
$s'_{0,0}$ $s'_{0,1}$ $s'_{0,2}$ $s'_{0,3}$
$s'_{1,0}$ $s'_{1,1}$ $s'_{1,2}$ $s'_{1,3}$
$s'_{2,0}$ $s'_{2,1}$ $s'_{2,2}$ $s'_{2,3}$
$s'_{3,0}$ $s'_{3,1}$ $s'_{3,2}$ $s'_{3,3}$

Mix Columns Example

87	F2	4D	97
6E	4C	90	EC
46	E7	4A	C3
A6	8C	D8	95

→

47	40	A3	4C
37	D4	70	9F
94	E4	3A	42
ED	A5	A6	BC

$$(\{02\} \bullet \{87\}) \oplus (\{03\} \bullet \{6E\}) \oplus \{46\} \oplus \{A6\} = \{47\}$$

$$\{87\} \oplus (\{02\} \bullet \{6E\}) \oplus (\{03\} \bullet \{46\}) \oplus \{A6\} = \{37\}$$

$$\{87\} \oplus \{6E\} \oplus (\{02\} \bullet \{46\}) \oplus (\{03\} \bullet \{A6\}) = \{94\}$$

$$(\{03\} \bullet \{87\}) \oplus \{6E\} \oplus \{46\} \oplus (\{02\} \bullet \{A6\}) = \{ED\}$$

AES Arithmetic

uses arithmetic in the finite field GF(2^8)

with irreducible polynomial

$m(x) = x^8 + x^4 + x^3 + x + 1$

which is (100011011) or {11b}

e.g.

{02} • {87} mod {11b} = (1 0000 1110) mod {11b}

= (1 0000 1110) xor (1 0001 1011) = (0001 0101)

Mix Columns

can express each col as 4 equations
- to derive each new byte in col

decryption requires use of inverse matrix
- with larger coefficients, hence a little harder

have an alternate characterization
- each column a 4-term polynomial
- with coefficients in GF(2^8)
- and polynomials multiplied modulo (x^4+1)

coefficients based on linear code with maximal distance between codewords

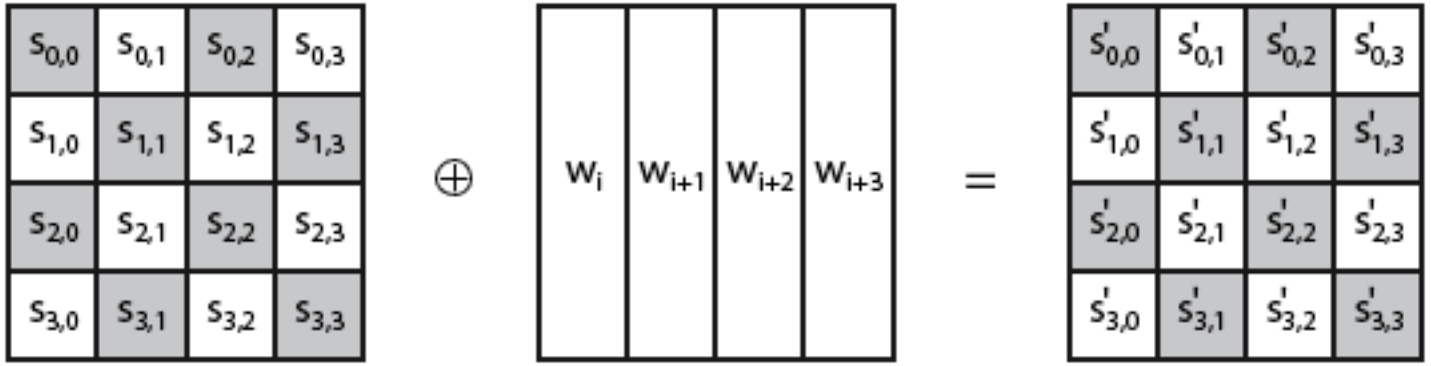
Add Round Key

- XOR state with 128-bits of the round key
- again processed by column (though effectively a series of byte operations)

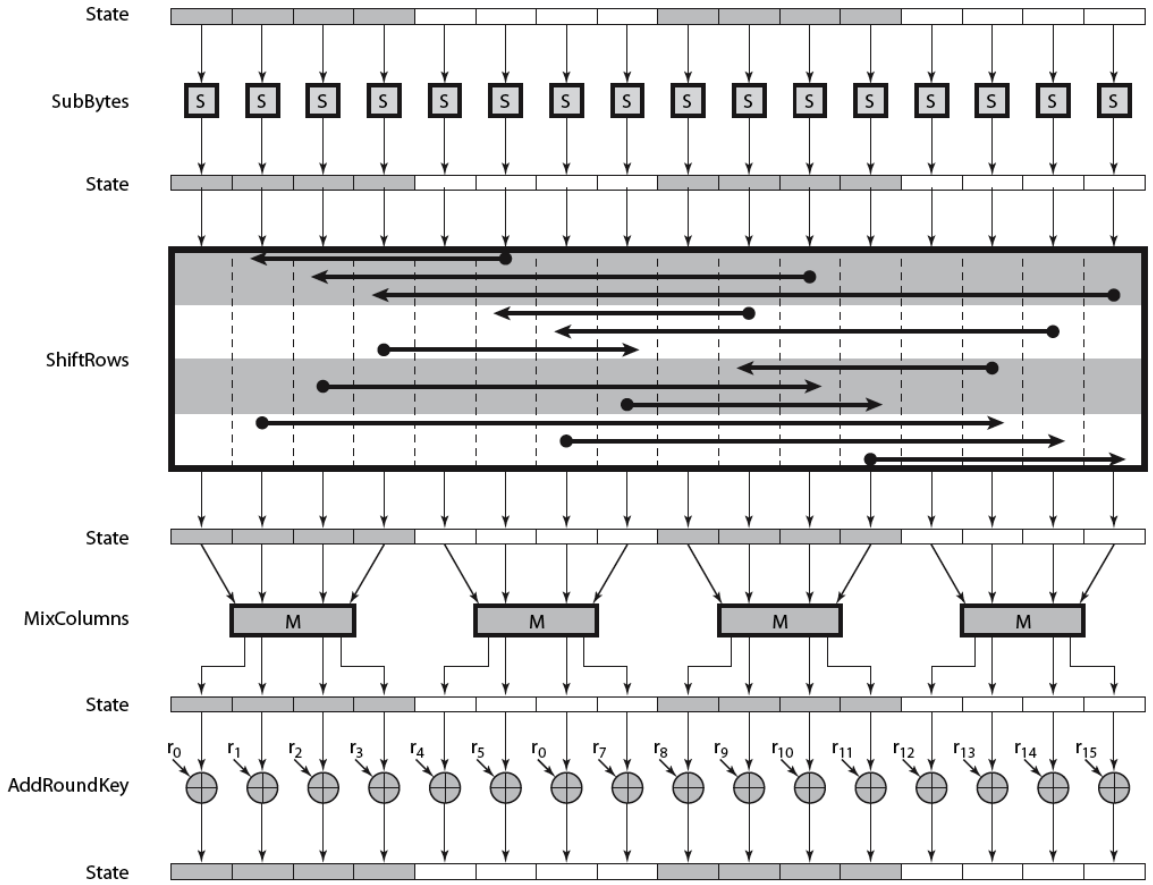
- inverse for decryption identical
  - since XOR own inverse, with reversed keys

- designed to be as simple as possible
  - a form of Vernam cipher on expanded key
  - requires other stages for complexity / security

Add Round Key



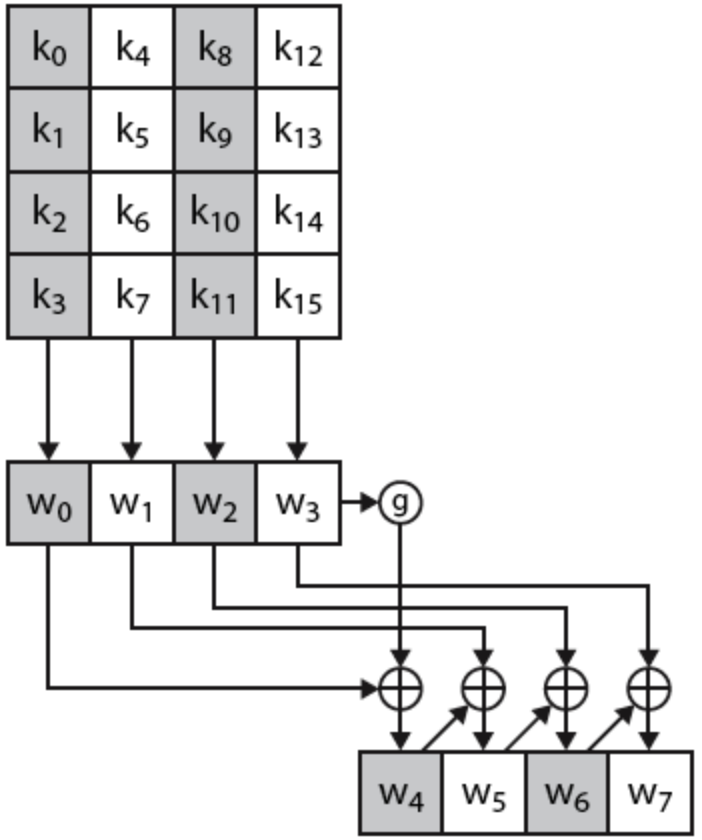
AES Round


State
SubBytes
S
State
ShiftRows
State
MixColumns
M
State
AddRoundKey
r0 r1 r2 r3 r4 r5 r0 r7 r8 r9 r10 r11 r12 r13 r14 r15
State

AES Key Expansion

- takes 128-bit (16-byte) key and expands into array of 44/52/60 32-bit words
- start by copying key into first 4 words
- then loop creating words that depend on values in previous & 4 places back
  - in 3 of 4 cases just XOR these together
  - 1st word in 4 has rotate + S-box + XOR round constant on previous, before XOR 4th back

AES Key Expansion



Key Expansion Rationale

designed to resist known attacks

design criteria included

- knowing part key insufficient to find many more
- invertible transformation
- fast on wide range of CPU's
- use round constants to break symmetry
- diffuse key bits into round keys
- enough non-linearity to hinder analysis
- simplicity of description

AES Example Key Expansion

Key Words	Auxiliary Function
w0 = 0f 15 71 c9 w1 = 47 d9 e8 59 w2 = 0c b7 ad w3 = af 7f 67 98	RotWord(w3)= 7f 67 98 af = x1 SubWord(x1)= d2 85 46 79 = y1 Rcon(1)= 01 00 00 00 y1 ⊕ Rcon(1)= d3 85 46 79 = z1
w4 = w0 ⊕ z1 = dc 90 37 b0 w5 = w4 ⊕ w1 = 9b 49 df e9 w6 = w5 ⊕ w2 = 97 fe 72 3f w7 = w6 ⊕ w3 = 38 81 15 a7	RotWord(w7)= 81 15 a7 38 = x2 SubWord(x4)= 0c 59 5c 07 = y2 Rcon(2)= 02 00 00 00 y2 ⊕ Rcon(2)= 0e 59 5c 07 = z2
w8 = w4 ⊕ z2 = d2 c9 6b b7 w9 = w8 ⊕ w5 = 49 80 b4 5e w10 = w9 ⊕ w6 = de 7e c6 61 w11 = w10 ⊕ w7 = e6 ff d3 c6	RotWord(w11)= ff d3 c6 e6 = x3 SubWord(x2)= 16 66 b4 8e = y3 Rcon(3)= 04 00 00 00 y3 ⊕ Rcon(3)= 12 66 b4 8e = z3
w12 = w8 ⊕ z3 = c0 af df 39 w13 = w12 ⊕ w9 = 89 2f 6b 67 w14 = w13 ⊕ w10 = 57 51 ad 06 w15 = w14 ⊕ w11 = b1 ae 7e c0	RotWord(w15)= ae 7e c0 b1 = x4 SubWord(x3)= e4 f3 ba c8 = y4 Rcon(4)= 08 00 00 00 y4 ⊕ Rcon(4)= ec f3 ba c8 = 4
w16 = w12 ⊕ z4 = 2c 5c 65 f1 w17 = w16 ⊕ w13 = a5 73 0e 96 w18 = w17 ⊕ w14 = f2 22 a3 90 w19 = w18 ⊕ w15 = 43 8c dd 50	RotWord(w19)= 8c dd 50 43 = x5 SubWord(x4)= 64 c1 53 1a = y5 Rcon(5)= 10 00 00 00 y5 ⊕ Rcon(5)= 74 c1 53 1a = z5
w20 = w16 ⊕ z5 = 58 9d 36 eb w21 = w20 ⊕ w17 = fd ee 38 7d w22 = w21 ⊕ w18 = 0f cc 9b ed w23 = w22 ⊕ w19 = 4c 40 46 bd	RotWord(w23)= 40 46 bd 4c = x6 SubWord(x5)= 09 5a 7a 29 = y6 Rcon(6)= 20 00 00 00 y6 ⊕ Rcon(6)= 29 5a 7a 29 = z6
w24 = w20 ⊕ z6 = 71 c7 4c c2 w25 = w24 ⊕ w21 = 8c 29 74 bf w26 = w25 ⊕ w22 = 83 e5 ef 52 w27 = w26 ⊕ w23 = cf a5 a9 ef	RotWord(w27)= a5 a9 ef cf = x7 SubWord(x6)= 06 d3 df 8a = y7 Rcon(7)= 40 00 00 00 y7 ⊕ Rcon(7)= 46 d3 df 8a = z7
w28 = w24 ⊕ z7 = 37 14 93 48 w29 = w28 ⊕ w25 = bb 3d e7 f7 w30 = w29 ⊕ w26 = 38 d8 08 a5 w31 = w30 ⊕ w27 = f7 7d a1 4a	RotWord(w31)= 7d a1 4a f7 = x8 SubWord(x7)= ff 32 d6 68 = y8 Rcon(8)= 80 00 00 00 y8 ⊕ Rcon(8)= 7f 32 d6 68 = z8
w32 = w28 ⊕ z8 = 48 26 45 20 w33 = w32 ⊕ w29 = f3 1b a2 d7 w34 = w33 ⊕ w30 = cb c3 aa 72 w35 = w34 ⊕ w32 = 3c be 0b 38	RotWord(w35)= be 0b 38 3c = x9 SubWord(x8)= ae 2b 07 eb = y9 Rcon(9)= 1B 00 00 00 y9 ⊕ Rcon(9)= b5 2b 07 eb = z9
w36 = w32 ⊕ z9 = fd 0d 42 cb w37 = w36 ⊕ w33 = 0e 16 e0 1c w38 = w37 ⊕ w34 = c5 d5 4a 6e w39 = w38 ⊕ w35 = f9 6b 41 56	RotWord(w39)= 6b 41 56 f9 = x10 SubWord(x9)= 7f 83 b1 99 = y10 Rcon(10)= 36 00 00 00 y10 ⊕ Rcon(10)= 49 83 b1 99 = z10
w40 = w36 ⊕ z10 = b4 8e f3 52 w41 = w40 ⊕ w37 = ba 98 13 4e w42 = w41 ⊕ w38 = 7f 4d 59 20 w43 = w42 ⊕ w39 = 86 26 18 76	

AES Example Encryption

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

AES Example Avalanche

Round		Number of bits that differ
	0123456789abcdeffedcba9876543210 0023456789abcdeffedcba9876543210	1
0	0e3634aece7225b6f26b174ed92b5588 0f3634aece7225b6f26b174ed92b5588	1
1	657470750fc7ff3fc0e8e8ca4dd02a9c c4a9ad090fc7ff3fc0e8e8ca4dd02a9c	20
2	5c7bb49a6b72349b05a2317ff46d1294 fe2ae569f7ee8bb8c1f5a2bb37ef53d5	58
3	7115262448dc747e5cdac7227da9bd9c ec093dfb7c45343d689017507d485e62	59
4	f867aee8b437a5210c24c1974cffeabc 43efdb697244df808e8d9364ee0ae6f5	61
5	721eb200ba06206dcbd4bce704fa654e 7b28a5d5ed643287e006c099bb375302	68
6	0ad9d85689f9f77bc1c5f71185e5fb14 3bc2d8b6798d8ac4fe36a1d891ac181a	64
7	db18a8ffa16d30d5f88b08d777ba4eaa 9fb8b5452023c70280e5c4bb9e555a4b	67
8	f91b4fbfe934c9bf8f2f85812b084989 20264e1126b219aef7feb3f9b2d6de40	65
9	cca104a13e678500ff59025f3bafaa34 b56a0341b2290ba7dfdfbddcd8578205	61
10	ff0b844a0853bf7c6934ab4364148fb9 612b89398d0600cde116227ce72433f0	58

AES Decryption

AES decryption is not identical to encryption since steps done in reverse

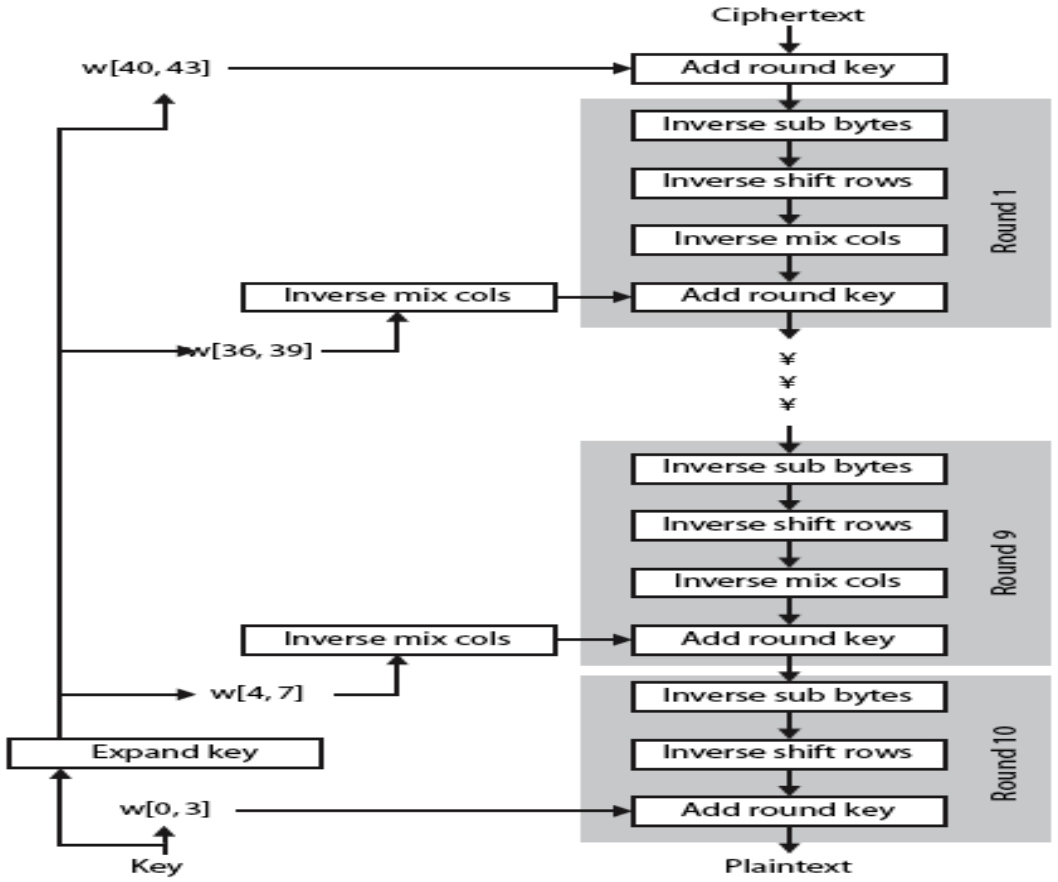
but can define an equivalent inverse cipher with steps as for encryption

- but using inverses of each step
- with a different key schedule

works since result is unchanged when

- swap byte substitution & shift rows
- swap mix columns & add (tweaked) round key

AES Decryption



Implementation Aspects

can efficiently implement on 8-bit CPU

- byte substitution works on bytes using a table of 256 entries
- shift rows is simple byte shift
- add round key works on byte XOR's
- mix columns requires matrix multiply in GF(2^8)
  - which works on byte values, can be simplified to use table lookups & byte XOR's

Implementation Aspects

- can efficiently implement on 32-bit CPU
  - redefine steps to use 32-bit words
  - can precompute 4 tables of 256-words
  - then each column in each round can be computed using 4 table lookups + 4 XORs
  - at a cost of 4Kb to store tables
- designers believe this very efficient implementation was a key factor in its selection as the AES cipher

Encrypting a Large Message

So, we've got a good block cipher, but our plaintext is larger than 128-bit block size

Electronic Code Book (ECB) mode

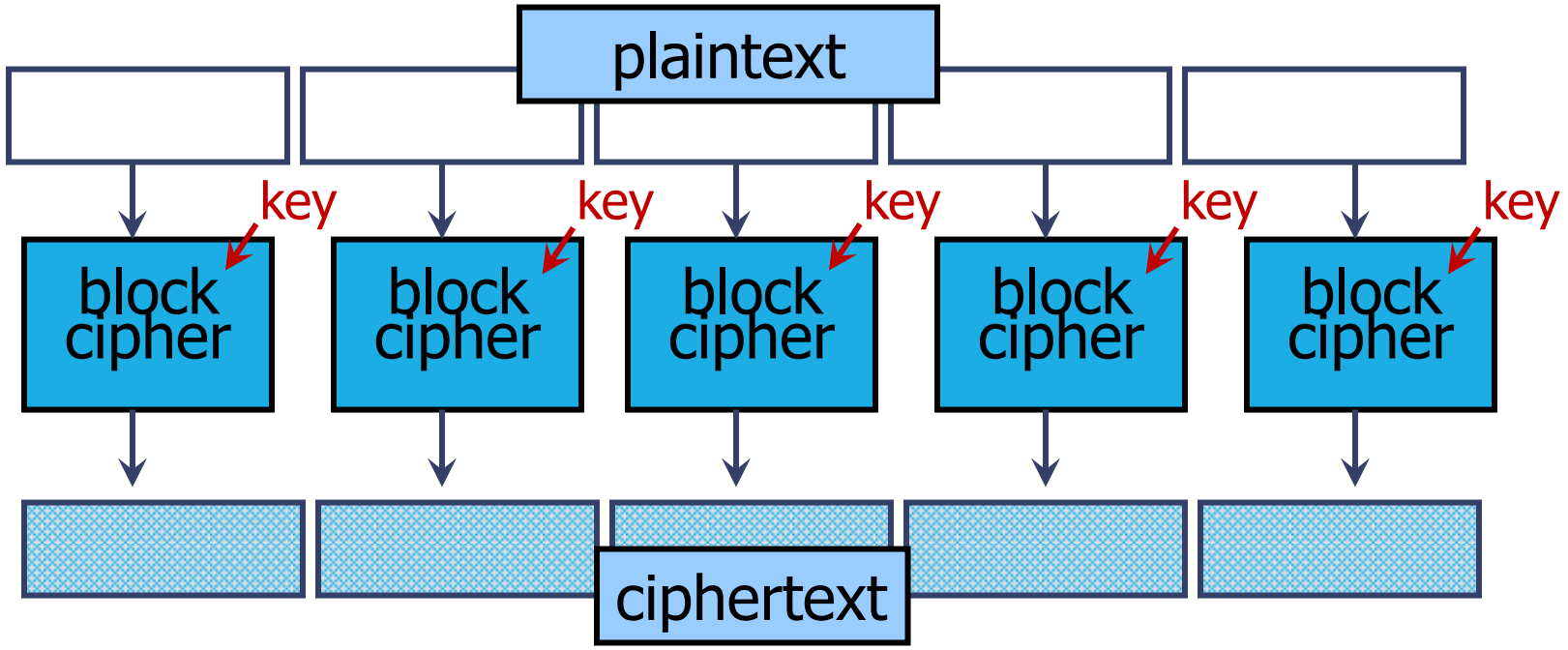
- Split plaintext into blocks, encrypt each one separately using the block cipher

Cipher Block Chaining (CBC) mode

- Split plaintext into blocks, XOR each block with the result of encrypting previous blocks

Also various counter modes, feedback modes, etc.

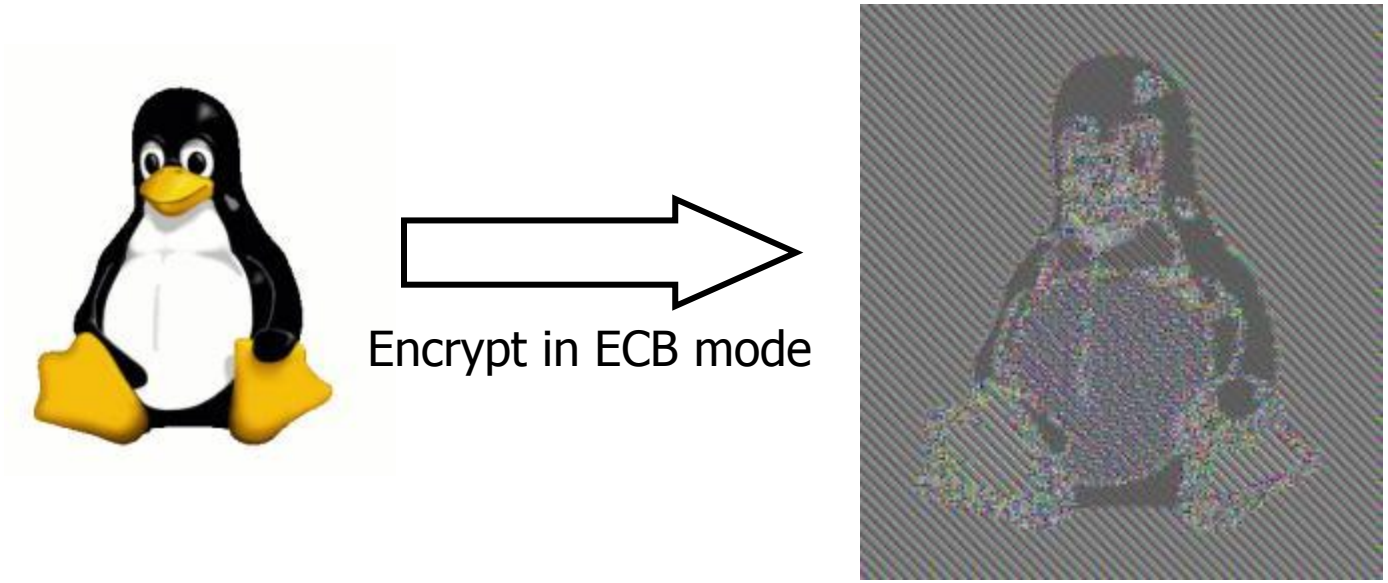
ECB Mode



Identical blocks of plaintext produce identical blocks of ciphertext

No integrity checks: can mix and match blocks

Information Leakage in ECB Mode



[Wikipedia]

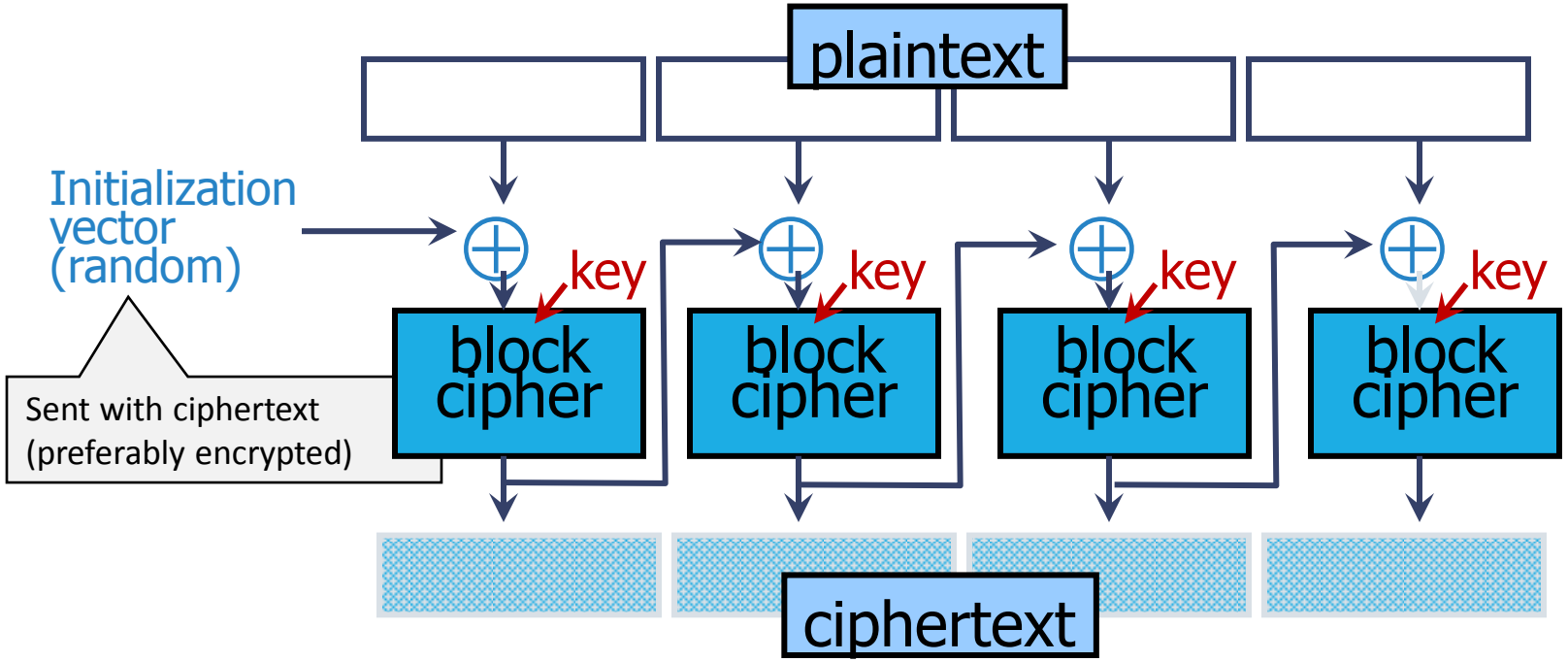
Adobe Passwords Stolen (2013)

153 million account passwords

- 56 million of them unique

Encrypted using 3DES in ECB mode rather than hashed

CBC Mode: Encryption

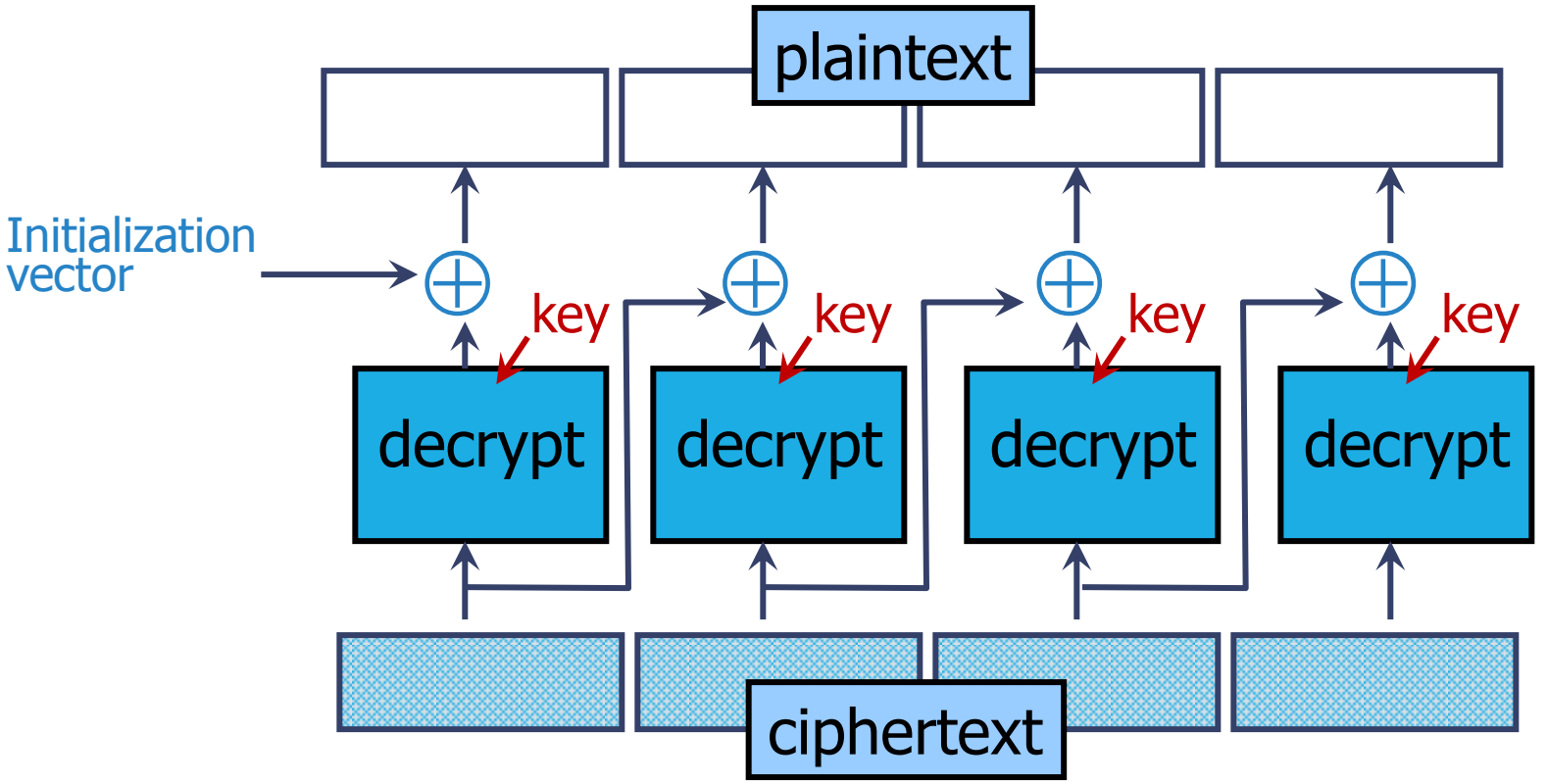


Identical blocks of plaintext encrypted differently

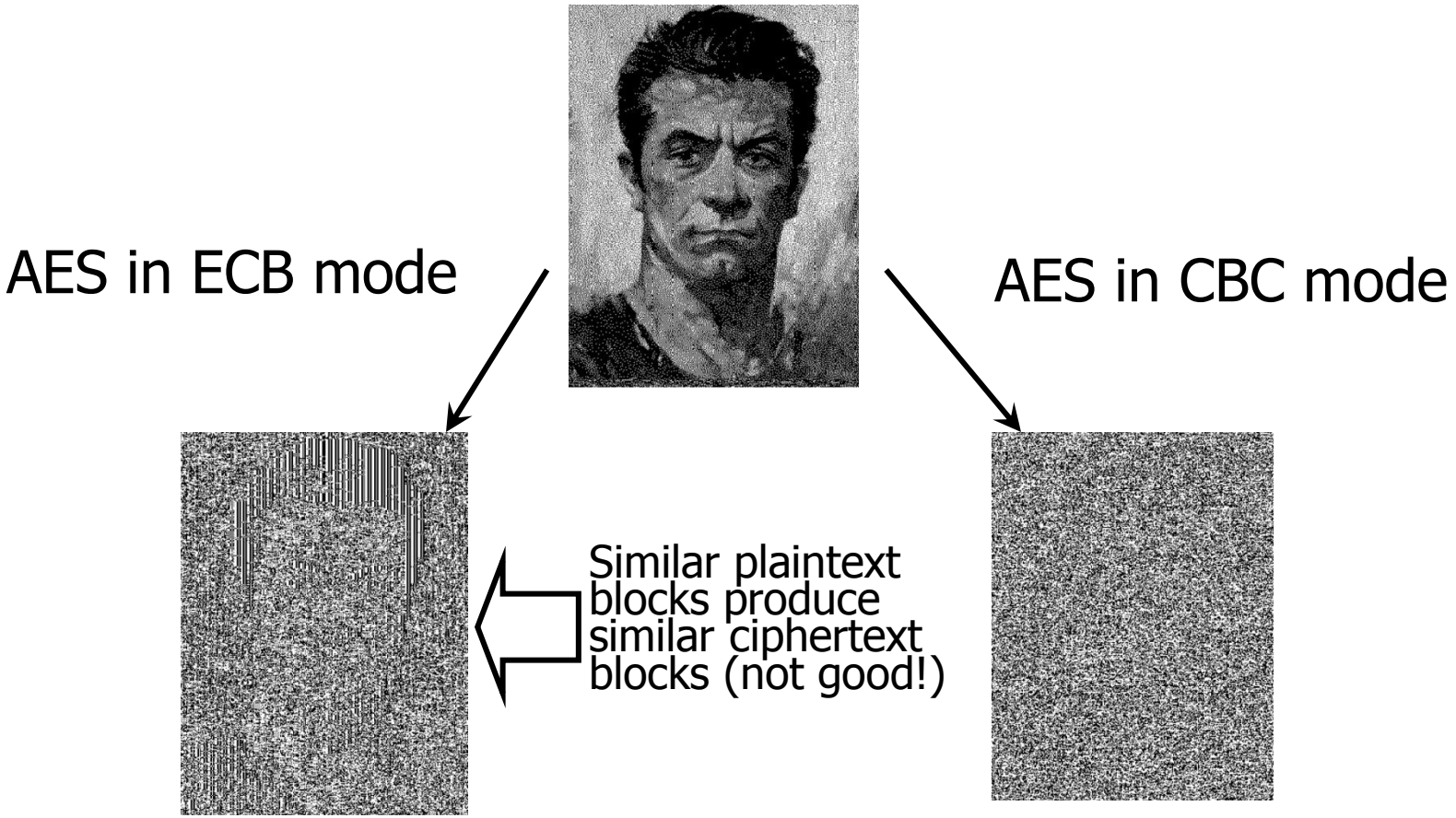
Last cipherblock depends on entire plaintext

- Still does not guarantee integrity

CBC Mode: Decryption



ECB vs. CBC


AES in ECB mode
AES in CBC mode
Similar plaintext blocks produce similar ciphertext blocks (not good!)

Choosing the Initialization Vector

Key used only once

- No IV needed (can use IV=0)

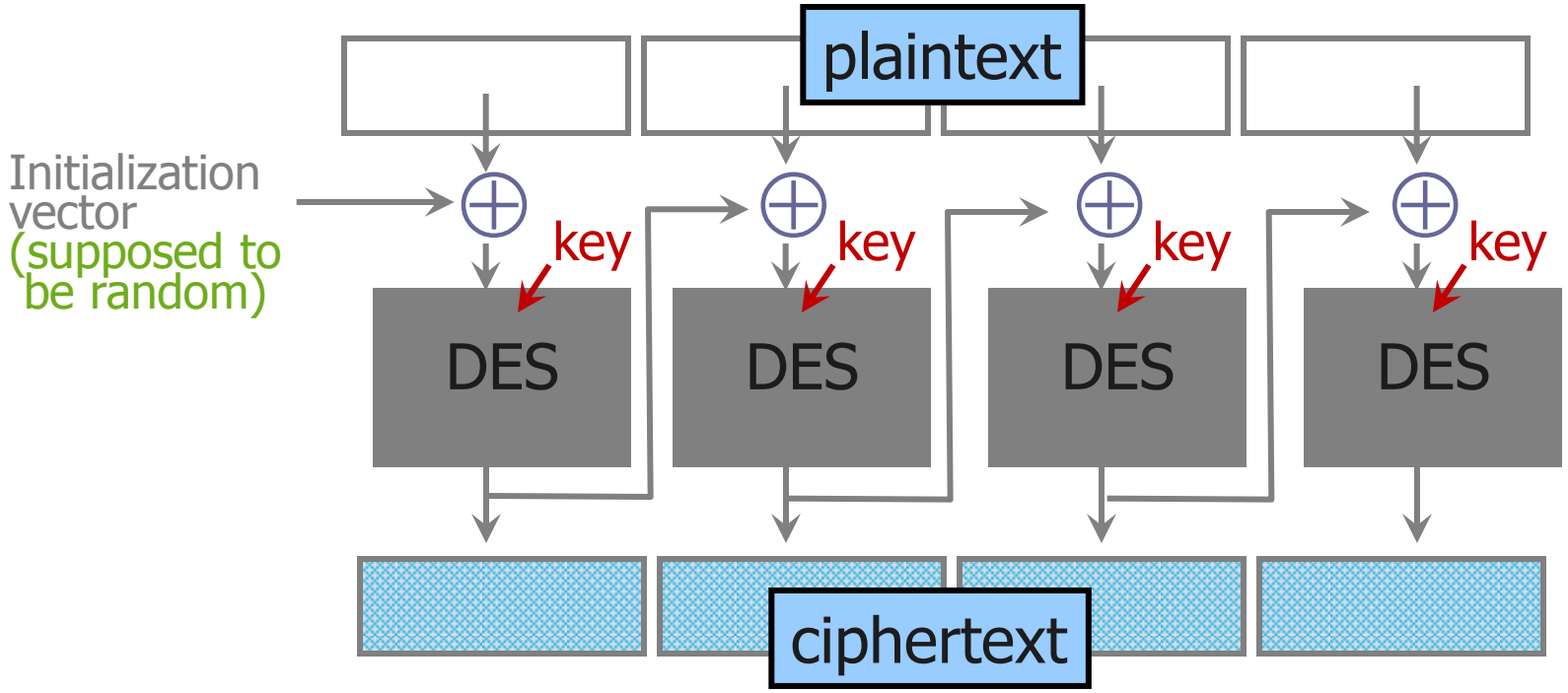
Key used multiple times

- Best: fresh, random IV for every message
- Can also use unique IV (eg, counter), but then the first step in CBC mode must be IV' ← E(k, IV)
  - Example: Windows BitLocker
  - May not need to transmit IV with the ciphertext

Multi-use key, unique messages

- Synthetic IV: IV ← F(k', message)
  - F is a cryptographically secure keyed pseudorandom function

CBC and Electronic Voting

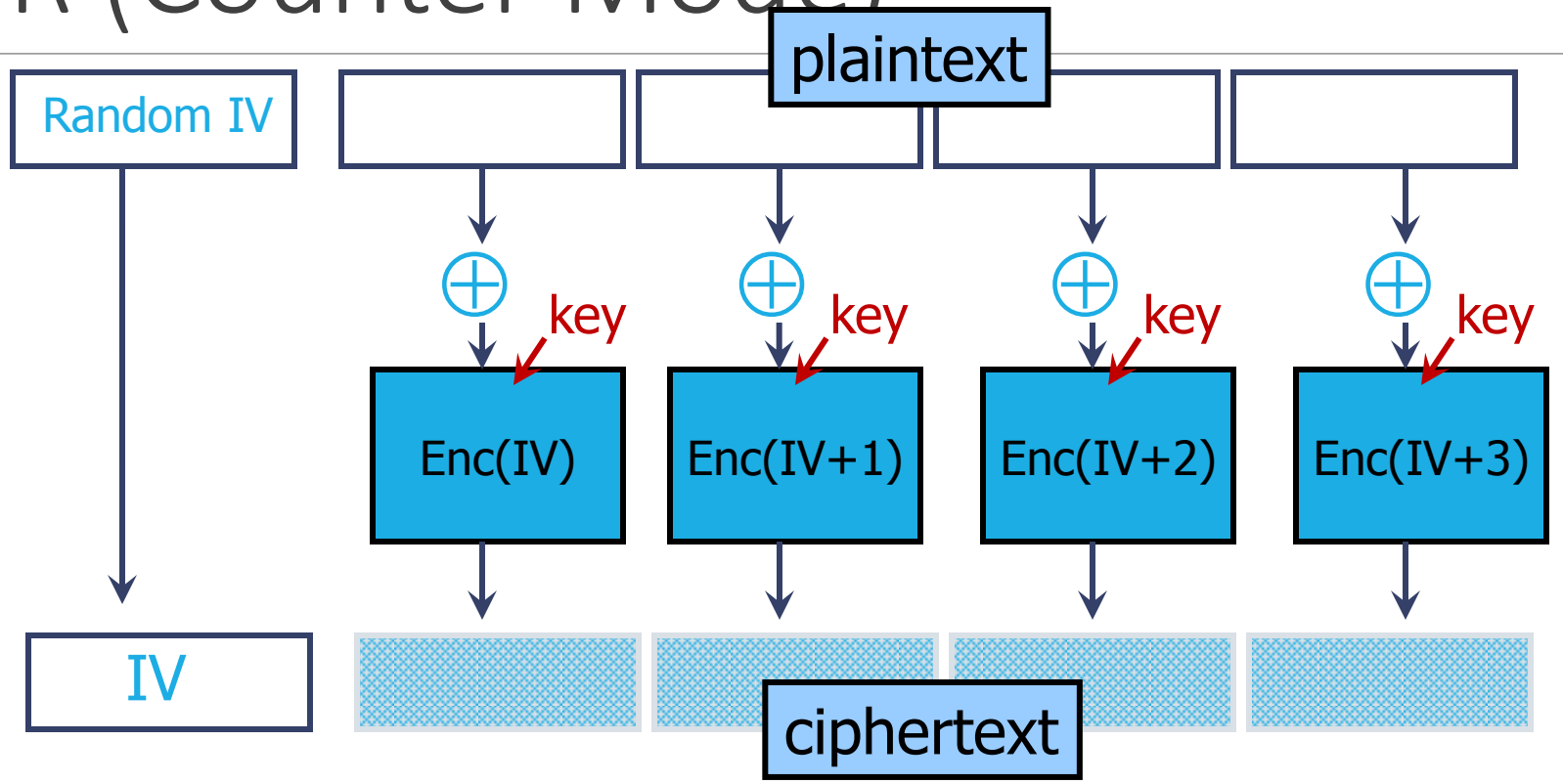


[Kohno, Stubblefield, Rubin, Wallach]

Found in the source code for Diebold voting machines:

```
DesCBCEncrypt((des_c_block*)tmp, (des_c_block*)record.m_Data,
              totalSize, DESKEY, NULL, DES_ENCRYPT)
```

CTR (Counter Mode)



Still does not guarantee integrity

Fragile if counter repeats

When Is a Cipher "Secure"?

Hard to recover plaintext from ciphertext?

- What if attacker learns only some bits of the plaintext? Some function of the bits? Some partial information about the plaintext?

Fixed mapping from plaintexts to ciphertexts?

- What if attacker sees two identical ciphertexts and infers that the corresponding plaintexts are identical?
- What if attacker guesses the plaintext – can he verify his guess?
- Implication: encryption must be randomized or stateful

How Can a Cipher Be Attacked?

Attackers knows ciphertext and encryption algthm

- What else does the attacker know? Depends on the application in which the cipher is used!

Known-plaintext attack (stronger)

- Knows some plaintext-ciphertext pairs

Chosen-plaintext attack (even stronger)

- Can obtain ciphertext for any plaintext of his choice

Chosen-ciphertext attack (very strong)

- Can decrypt any ciphertext except the target
- Sometimes very realistic

Known-Plaintext Attack

[From "The Art of Intrusion"]

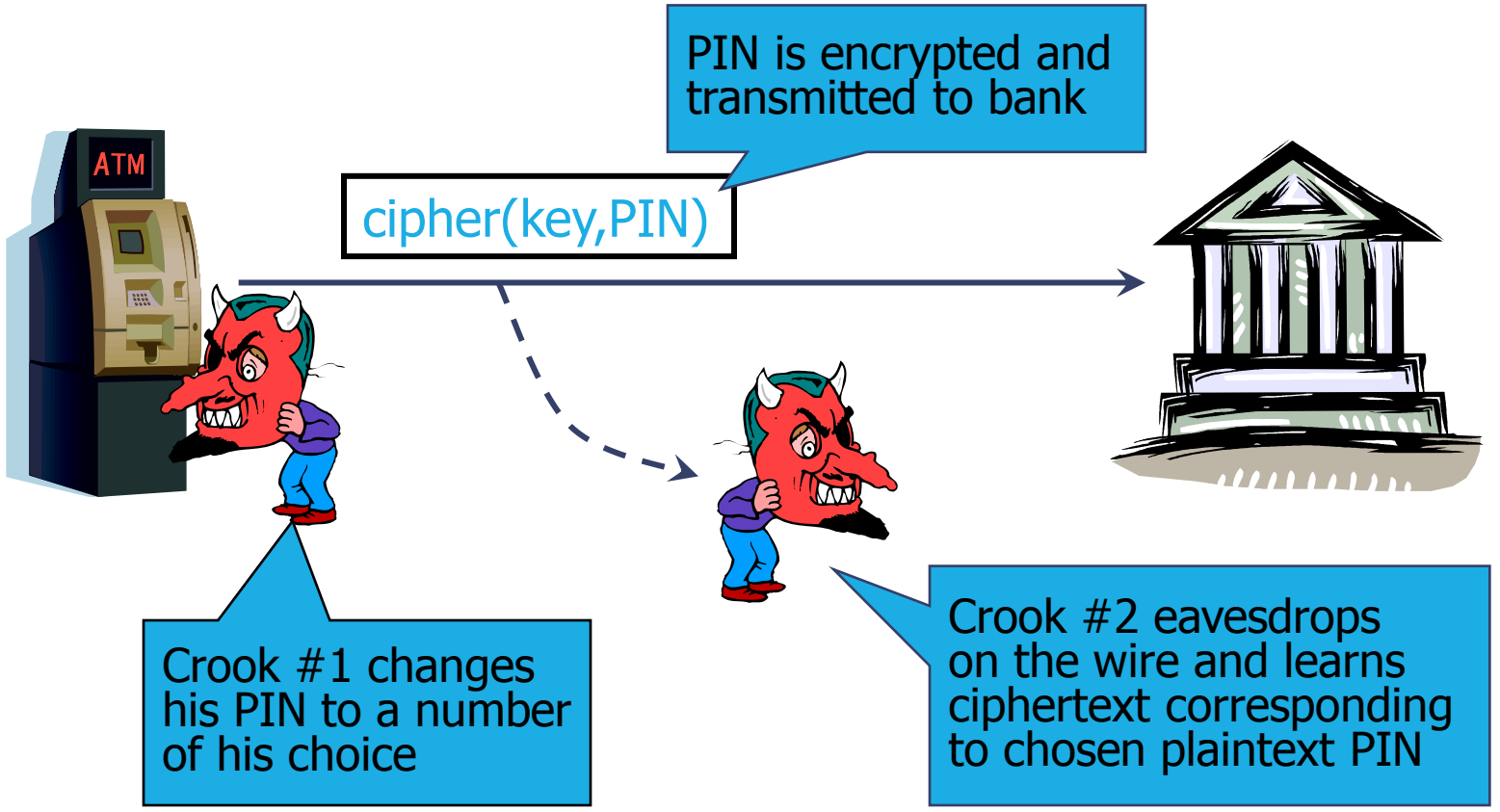
Extracting password from an encrypted PKZIP file …

"… I opened the ZIP file and found a `logo.tif' file, so I went to their main Web site and looked at all the files named `logo.tif.' I downloaded them and zipped them all up and found one that matched the same checksum as the one in the protected ZIP file"

With known plaintext, PkCrack took 5 minutes to extract the key

- Biham-Kocher attack on PKZIP stream cipher

Chosen-Plaintext Attack



… repeat for any PIN value

Very Informal Intuition

Minimum security requirement for a modern encryption scheme

Security against chosen-plaintext attack

- Ciphertext leaks no information about the plaintext
- Even if the attacker correctly guesses the plaintext, he cannot verify his guess
- Every ciphertext is unique, encrypting same message twice produces completely different ciphertexts

Security against chosen-ciphertext attack

- Integrity protection – it is not possible to change the plaintext by modifying the ciphertext

The Chosen-Plaintext Game

Attacker does not know the key

He chooses as many plaintexts as he wants, and receives the corresponding ciphertexts

When ready, he picks two plaintexts M_0 and M_1

- He is even allowed to pick plaintexts for which he previously learned ciphertexts!

He receives either a ciphertext of M_0, or a ciphertext of M_1

He wins if he guesses correctly which one it is

Meaning of “Leaks No Information”

Idea: given a ciphertext, attacker should not be able to learn even a single bit of useful information about the plaintext

0 or 1

Let Enc(M_0,M_1,b) be a “magic box” that returns encrypted M_b

- Given two plaintexts, the box always returns the ciphertext of the left plaintext or right plaintext
- Attacker can use this box to obtain the ciphertext of any plaintext M by submitting M_0=M_1=M, or he can try to learn even more by submitting $M_0 \neq M_1$

Attacker’s goal is to learn just this one bit b

Chosen-Plaintext Security

Consider two experiments (A is the attacker)

Experiment 0

A interacts with Enc(-,-,0)

and outputs his guess of bit b

Experiment 1

A interacts with Enc(-,-,1)

and outputs his guess of bit b

- Identical except for the value of the secret bit
- b is attacker's guess of the secret bit

Attacker's advantage is defined as

$$| \text{Prob}(\text{A outputs 1 in Exp0}) - \text{Prob}(\text{A outputs 1 in Exp1})) |$$

Encryption scheme is chosen-plaintext secure if this advantage is negligible for any efficient A

Simple Example

Any deterministic, stateless symmetric encryption scheme is insecure

- Attacker can easily distinguish encryptions of different plaintexts from encryptions of identical plaintexts
- This includes ECB mode of common block ciphers!

Attacker A interacts with Enc(-,-,b)

Let X,Y be any two different plaintexts

$C_1 \leftarrow$ Enc(X,X,b); $C_2 \leftarrow$ Enc(X,Y,b);

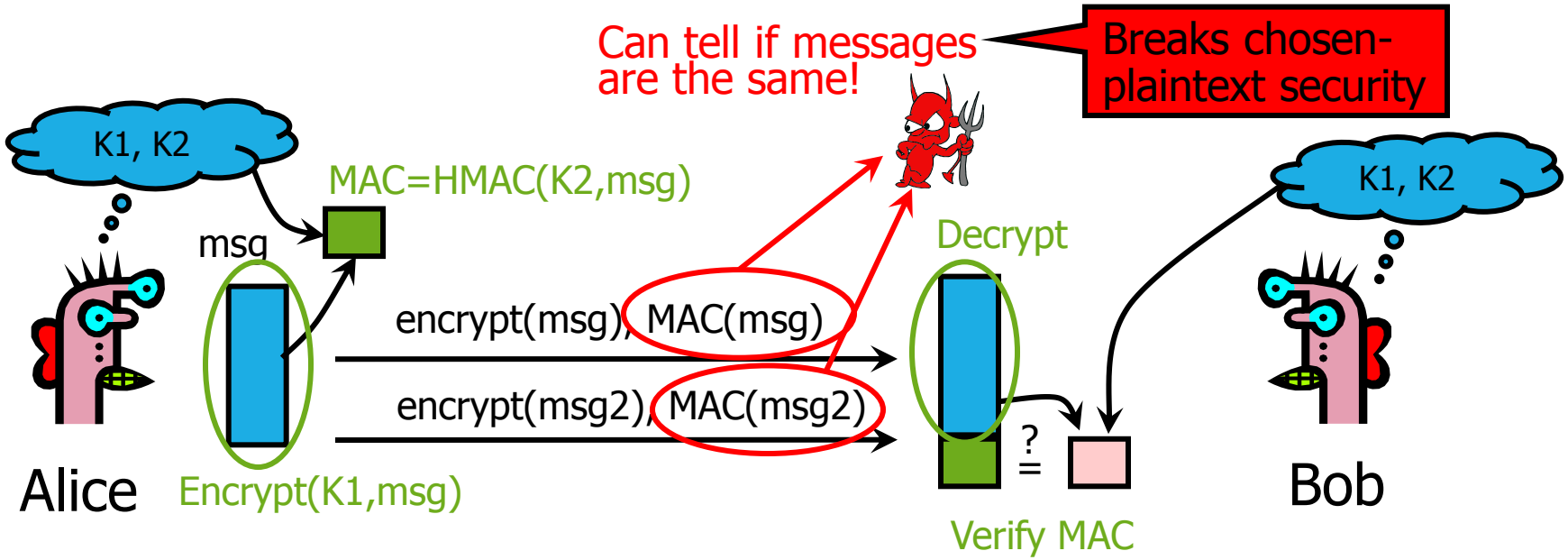
If $C_1=C_2$ then b=0 else b=1

The advantage of this attacker A is 1

Prob(A outputs 1 if b=0)=0 Prob(A outputs 1 if b=1)=1

Encrypt + MAC

Goal: confidentiality + integrity + authentication



MAC is deterministic: messages are equal ⇒ their MACs are equal

Solution: Encrypt, then MAC (or MAC, then encrypt)

Block Cipher Operation

Multiple Encryption & DES

- clear a replacement for DES was needed
  - theoretical attacks that can break it
  - demonstrated exhaustive key search attacks
- AES is a new cipher alternative
- prior to this alternative was to use multiple encryption with DES implementations
- Triple-DES is the chosen form

Why not Double-DES?

- could use 2 DES encrypts on each block
  - $C = E_{K2}(E_{K1}(P))$
- concern at time of reduction to single stage
- "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
  - Requires…

known plaintext

Triple-DES with Two-Keys

- hence must use 3 encryptions
  - would seem to need 3 distinct keys
- but can use 2 keys with E-D-E sequence
  - $C = E_{K1}(D_{K2}(E_{K1}(P)))$
  - n.b. encrypt & decrypt equivalent in security
  - if $K1=K2$ then can work with single DES
- standardized in ANSI X9.17 & ISO8732
- no current known practical attacks
  - several proposed impractical attacks might become basis of future attacks

Triple-DES with Three-Keys

- although are no practical attacks on two-key Triple-DES 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, e.g., PGP, S/MIME

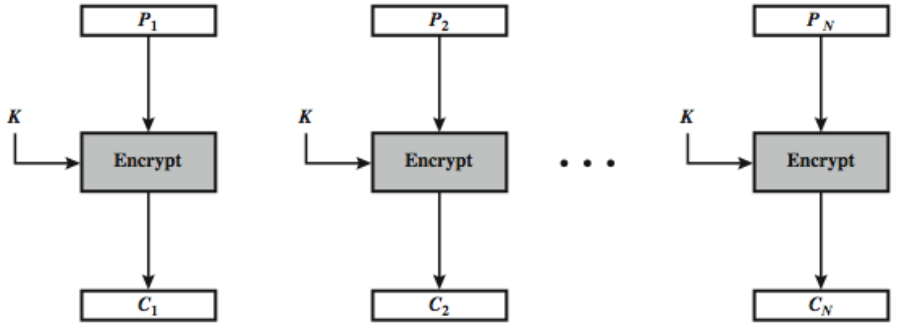
Modes of Operation

- block ciphers encrypt fixed size blocks
  - e.g., DES encrypts 64-bit blocks
- need some way to en/decrypt arbitrary amounts of data in practice
- NIST SP 800-38A defines 5 modes
- have **block** and **stream** modes
- to cover a wide variety of applications
- can be used with any block cipher

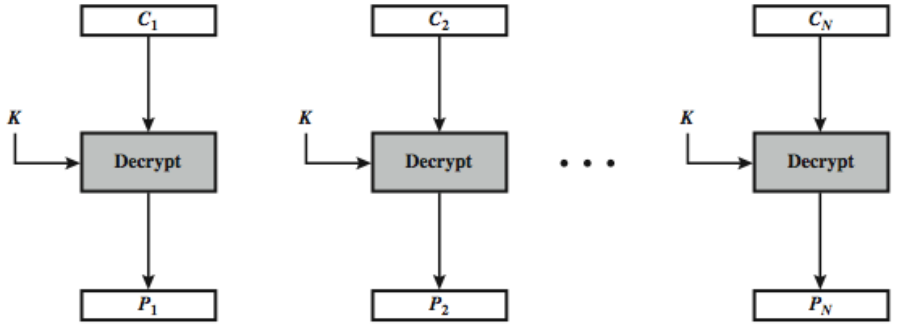
Electronic Codebook Book (ECB)

- message is broken into independent blocks that are encrypted
- each block is a value which is substituted, like a codebook, hence name
- each block is encoded independently of the other blocks
  - $C_i = E_K(P_i)$
- uses: secure transmission of single values

Electronic Codebook Book (ECB)



(a) Encryption



(b) Decryption

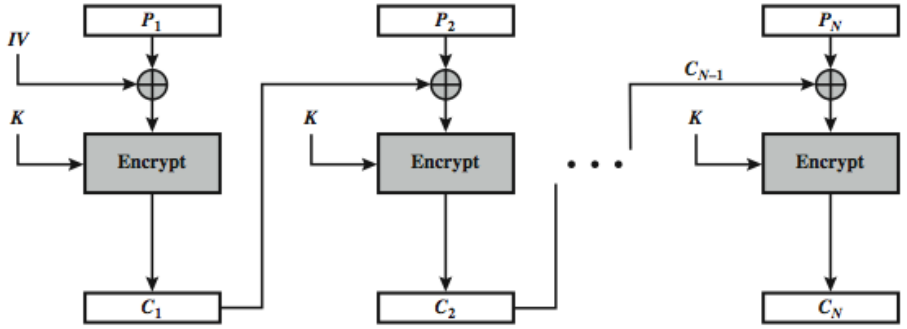
Advantages and Limitations of ECB

- message repetitions may show in ciphertext
  - if aligned with message block
  - particularly with data such graphics
  - or with messages that change very little, which become a code-book analysis problem
- weakness is due to the encrypted message blocks being independent
- vulnerable to cut-and-paste attacks
- main use is sending a few blocks of data

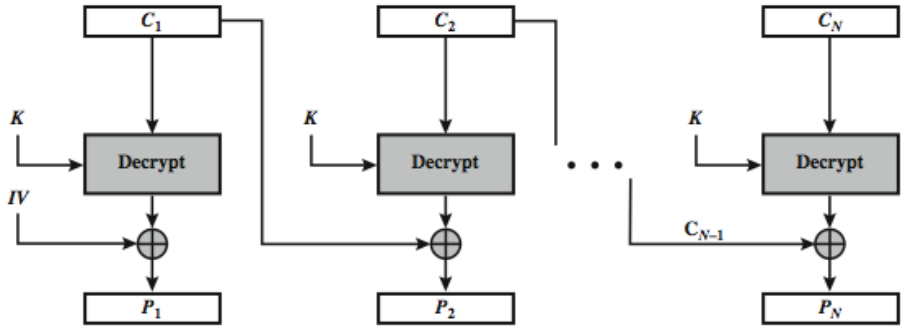
Cipher Block Chaining (CBC)

- message is broken into blocks
- linked together in encryption operation
- each previous cipher block is chained with current plaintext block, hence name
- use Initial Vector (IV) to start process
  - $C_i = E_K(P_i \text{ XOR } C_{i-1})$
  - $C_{-1} = IV$
- IV prevents same P from making same C
- uses: bulk data encryption, authentication

Cipher Block Chaining (CBC)



(a) Encryption



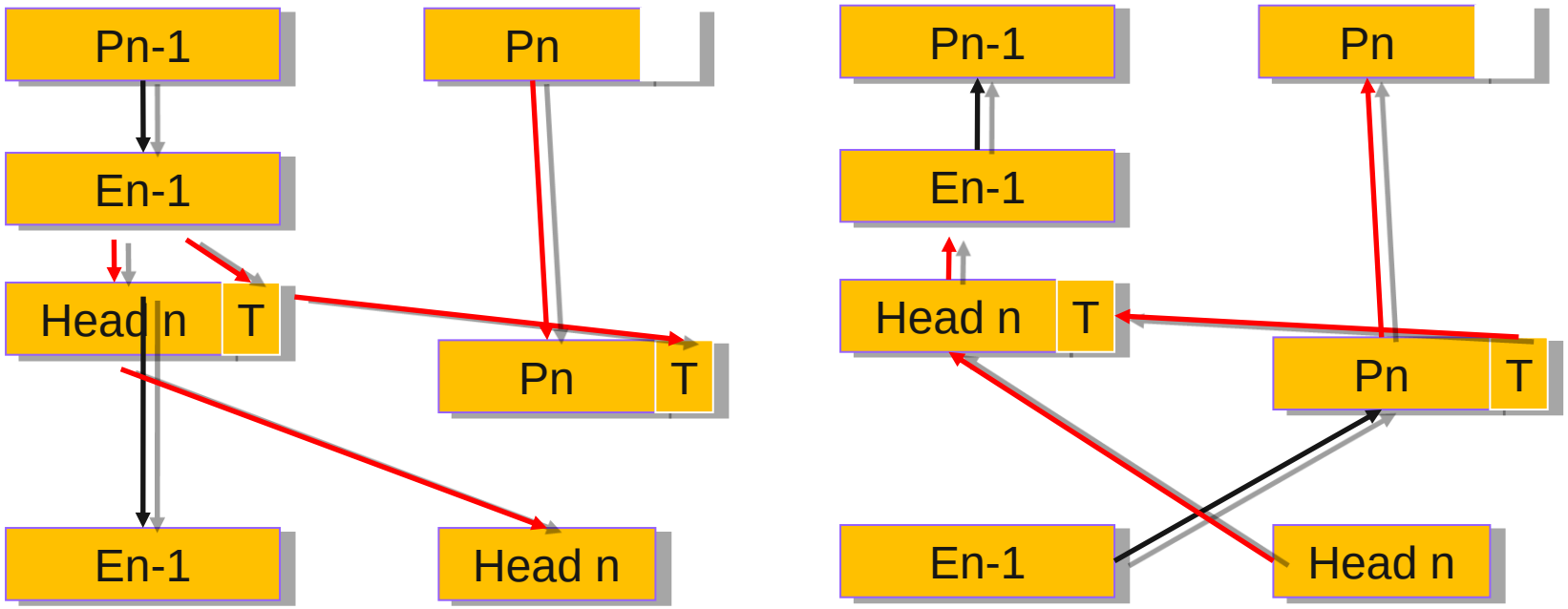
(b) Decryption

Message 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
    - e.g., nulls
  - or pad last block along with count of pad size
    - e.g., [b1 b2 b3 0 0 0 0 5]
    - means have 3 data bytes, then 5 bytes pad+count
  - this may require an extra entire block over those in message
- there are other, more esoteric modes, which avoid the need for an extra block

Ciphertext Stealing

- Use to make ciphertext length same as plaintext length
- Requires more than one block of ptxt



Advantages and Limitations of CBC

- a ciphertext block depends on **all** blocks before it
- any change to a block affects all following ciphertext blocks... avalanche effect
- need **Initialization Vector** (IV)
  - which must be known to sender & receiver
  - if sent in clear, attacker can change bits of first block, by changing corresponding bits of IV
  - hence IV must either be a fixed value (as in EFTPOS)
  - or derived in way hard to manipulate
  - or sent encrypted in ECB mode before rest of message
  - or message integrity must be checked otherwise

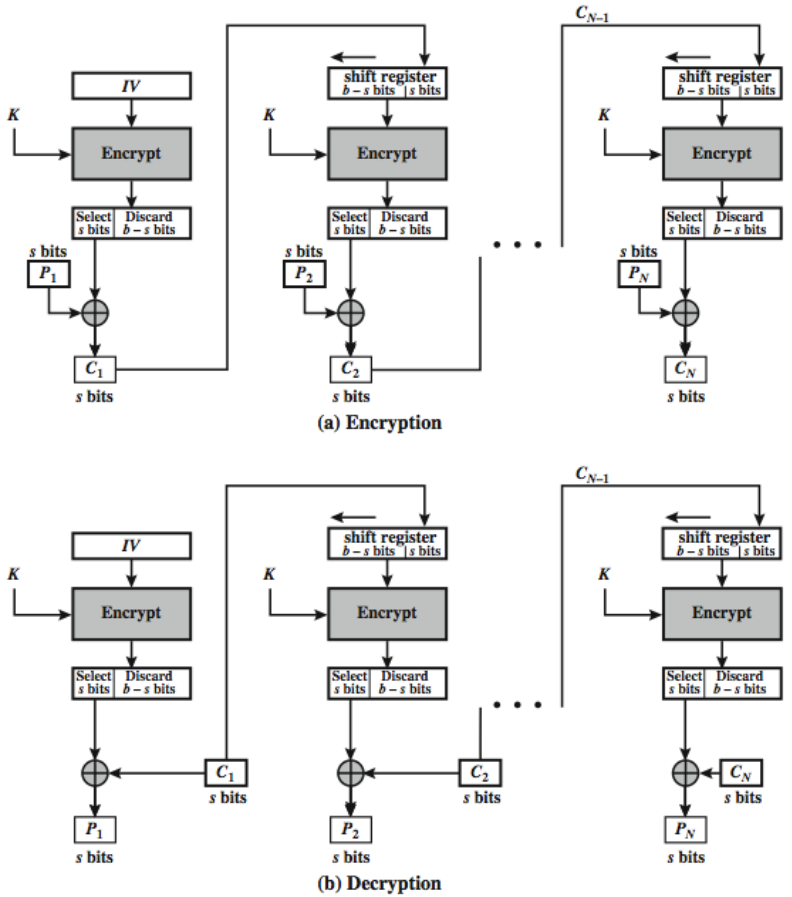
Stream Modes of Operation

- block modes encrypt entire block
- may need to operate on smaller units
  - real time data
- convert block cipher into stream cipher
  - cipher feedback (CFB) mode
  - output feedback (OFB) mode
  - counter (CTR) mode
- use block cipher as some form of **pseudo-random number** generator... Vernam cipher

Cipher FeedBack (CFB)

- message is treated as a stream of bits
- added to the output of the block cipher
- result is feed back for next stage (hence name)
- standard allows any number of bits (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)
  - $C_i = P_i \text{ XOR } E_K(C_{i-1})$
  - $C_{-1} = IV$
- uses: stream data encryption, authentication

s-bit Cipher FeedBack (CFB-s)



Advantages and Limitations of CFB

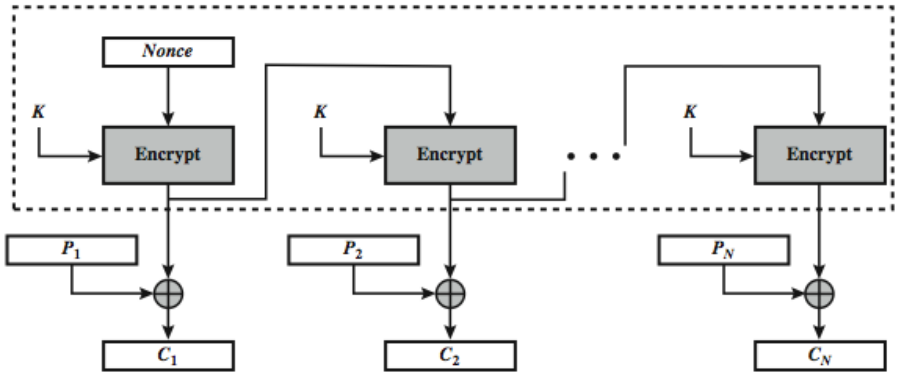
- most common stream mode
- appropriate when data arrives in bits/bytes
- limitation is need to stall while do block encryption after every s-bits
- note that the block cipher is used in **encryption** mode at **both** ends (XOR)
- errors propagate for several blocks after the error ... how many?

Output FeedBack (OFB)

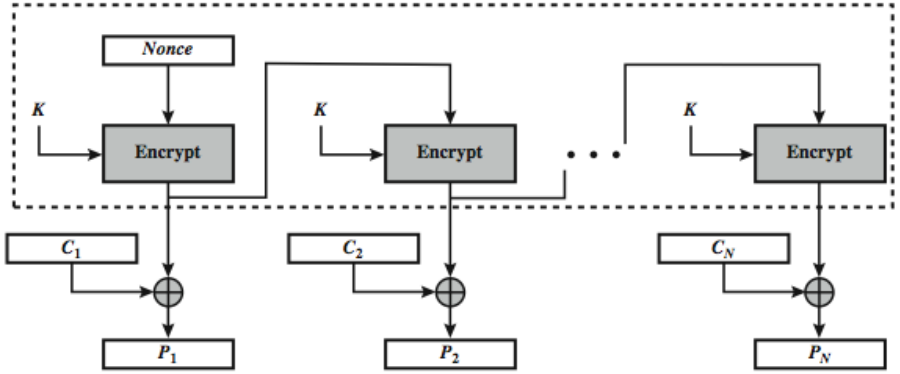
- message is treated as a stream of bits
- output of cipher is added to message
- output is then feed back (hence name)
  - $O_i = E_K(O_{i-1})$
  - $C_i = P_i \text{ XOR } O_i$
  - $O_{-1} = IV$
- feedback is independent of message
- can be computed in advance
- uses: stream encryption on noisy channels

  Why noisy channels?

Output FeedBack (OFB)



(a) Encryption



(b) Decryption

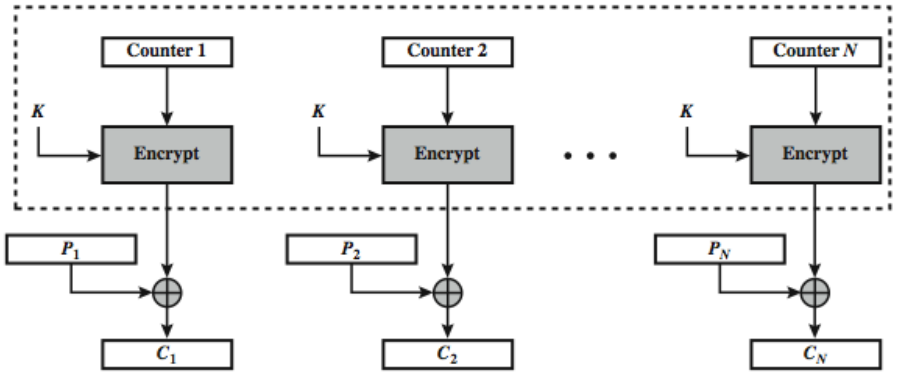
Advantages and Limitations of OFB

- needs an IV which is unique for each use
  - if ever reuse attacker can recover outputs...
  - OTP
- can pre-compute
- bit errors do not propagate
- more vulnerable to message stream modification...
  - change arbitrary bits by changing ciphertext
- sender & receiver must remain in sync
- only use with full block feedback
  - subsequent research has shown that only **full block feedback** (ie CFB-64 or CFB-128) should ever be used

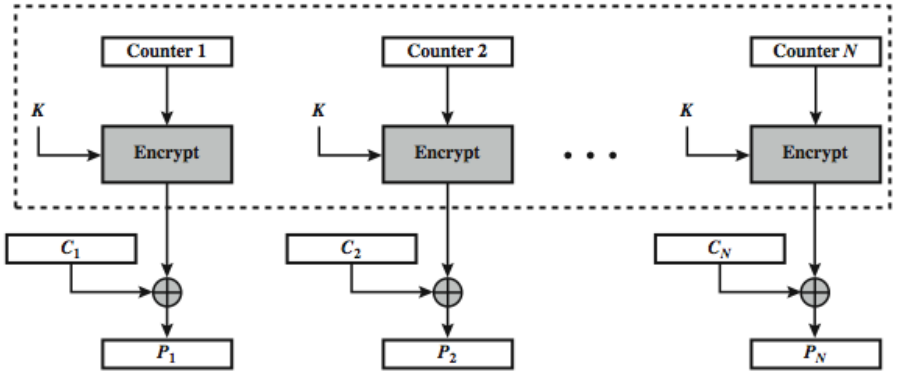
Counter (CTR)

- a “new” mode, though proposed early on
- similar to OFB but encrypts counter value rather than any feedback value
  - $O_i = E_K(i)$
  - $C_i = P_i \text{ XOR } O_i$
- must have a different key & counter value for every plaintext block (never reused)
  - again, OTP issue
- uses: high-speed network encryptions

Counter (CTR)



(a) Encryption

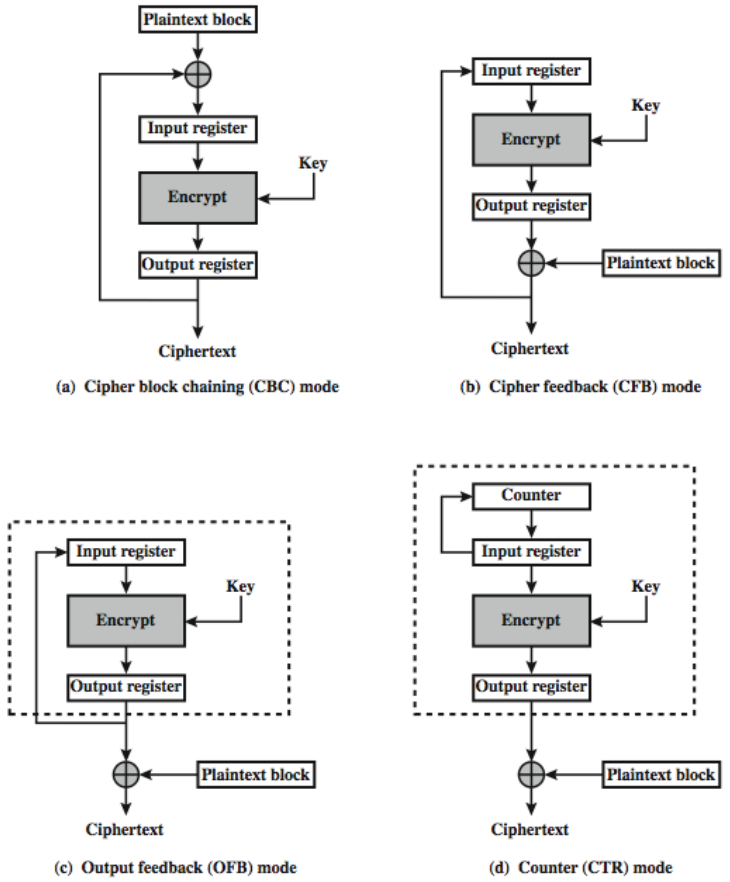


(b) Decryption

Advantages and Limitations of CTR

- efficiency
  - can do parallel encryptions in h/w or s/w
  - can preprocess in advance of need
  - good for bursty high speed links
- random access to encrypted data blocks
- provable security (good as other modes)
- never have cycle less than 2^b
- but must ensure never reuse key/counter values, otherwise could break (cf OFB)

Feedback Characteristics



XTS-AES Mode

- need mode for block oriented storage
  - No extra room in sector – data only
  - Disk addressed by sector number
  - Encryption can only take key externally
  - Encryption can also use sector#, block#
- Access to any sector should be independent of other sectors
- Must prevent attack that copies sector to unused sector, then requests decryption

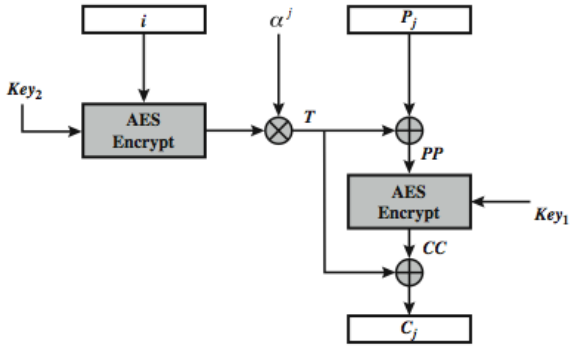
XTS-AES Mode

- new mode, for block oriented storage use
  - in IEEE Std 1619-2007
- concept of tweakable block cipher
- different requirements to transmitted data
- uses AES twice for each block
  - $T_j = E_{K2}(i) \text{ XOR } \alpha^j$
  - $C_j = E_{K1}(P_j \text{ XOR } T_j) \text{ XOR } T_j$
  - where i is tweak & j is sector no
- each sector may have multiple blocks

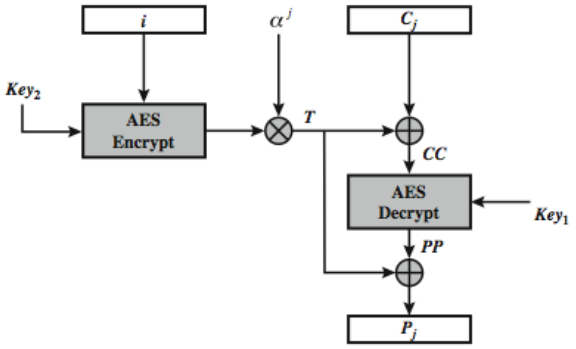
XTS-AES Mode per block

Key whitening applied by XOR
With “tweak” that depends on
- sector
- block
- second key

Makes attacks more difficult
Makes operations depend on
data location

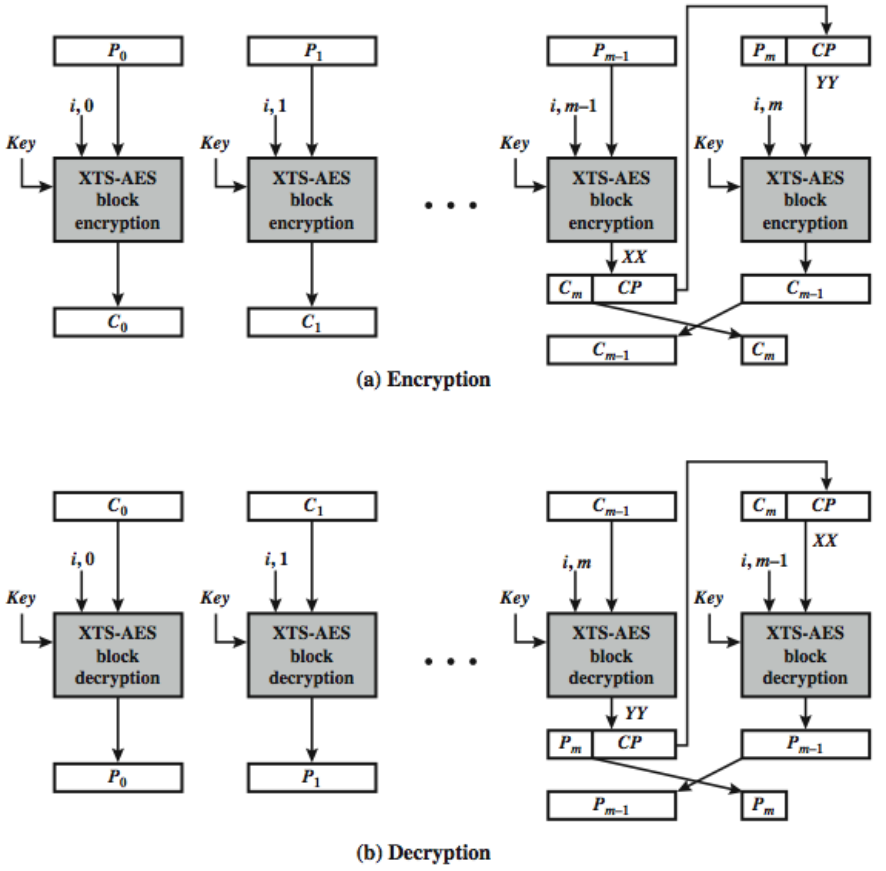


(a) Encryption



(b) Decryption

XTS-AES Mode Overview



Advantages and Limitations of XTS-AES

- efficiency
  - can do parallel encryptions in h/w or s/w
  - random access to encrypted data blocks
- has both nonce & counter
- addresses security concerns related to stored data

Stream Ciphers and Random Number Generation

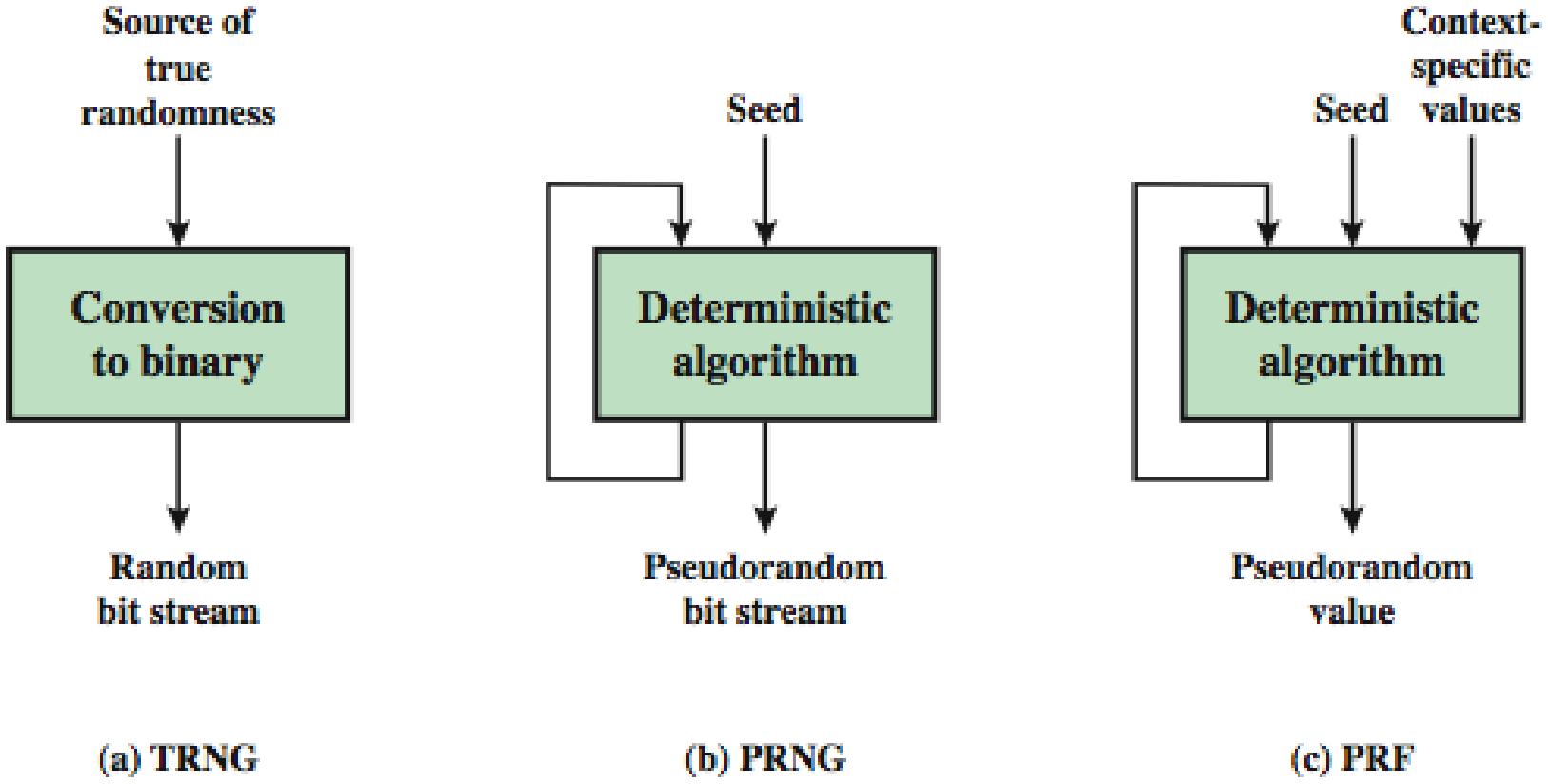
Random Numbers

- many uses of **random numbers** in cryptography
  - nonces in authentication protocols to prevent replay
  - session keys
  - public key generation
  - keystream for a one-time pad
- in all cases its critical that these values be
  - statistically random, uniform distribution, independent
  - unpredictability of future values from previous values
- true random numbers provide this
- care needed with generated random numbers

Pseudorandom Number Generators (PRNGs)

- often use deterministic algorithmic techniques to create "random numbers"
  - although are not truly random
  - can pass many tests of "randomness"
- known as "pseudorandom numbers"
- created by "Pseudorandom Number Generators (PRNGs)"

Random & Pseudorandom Number Generators



PRNG Requirements

- randomness
  - uniformity, scalability, consistency
- unpredictability
  - forward & backward unpredictability
  - use same tests to check
- characteristics of the seed
  - secure
  - if known adversary can determine output
  - so must be random or pseudorandom number

Linear Congruential Generator

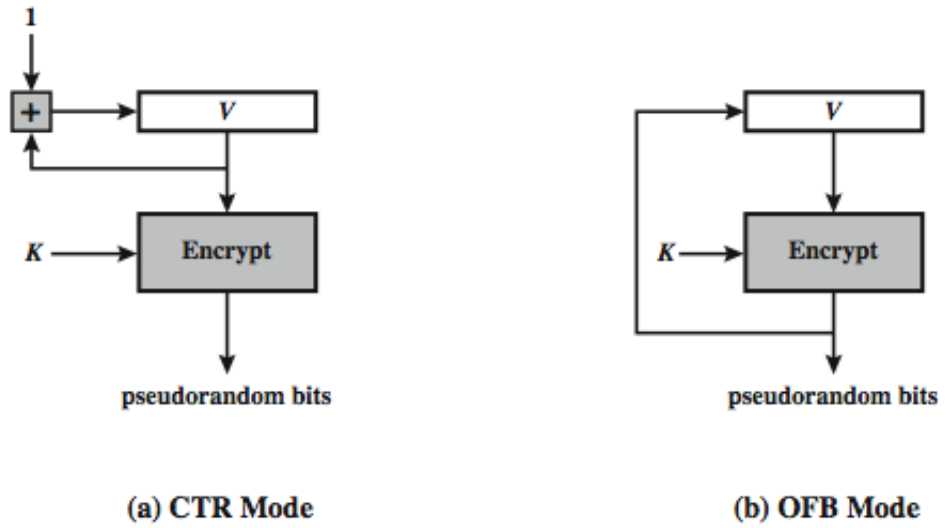
- common iterative technique using:
  - $X_{n+1} = (aX_n + c) \bmod m$
- given suitable values of parameters can produce a long random-like sequence
- suitable criteria to have are:
  - function generates a full-period
  - generated sequence should appear random
  - efficient implementation with 32-bit arithmetic
- note that an attacker can reconstruct sequence given a small number of values
- have possibilities for making this harder

Blum Blum Shub Generator

- based on public key algorithms
- use least significant bit from iterative equation:
  - $x_i = x_{i-1}^2 \bmod n$
  - where $n=p.q$, and primes $p, q=3 \bmod 4$
- unpredictable, passes **next-bit** test
- security rests on difficulty of factoring N
- is unpredictable given any run of bits
- slow, since very large numbers must be used
- too slow for cipher use, good for key generation

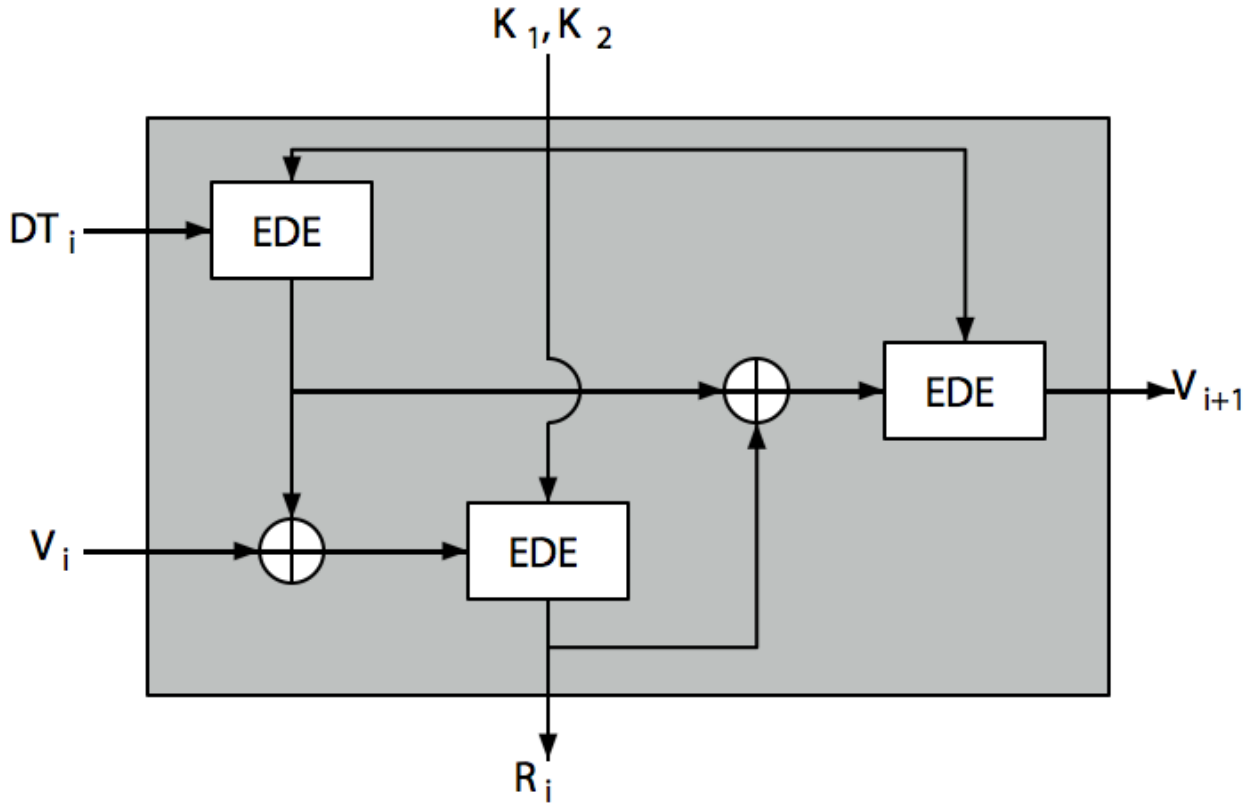
Using Block Ciphers as PRNGs

- for cryptographic applications, can use a block cipher to generate random numbers
- often for creating session keys from master key
- CTR
  - $X_i = E_K[V_i]$
- OFB
  - $X_i = E_K[X_{i-1}]$



(a) CTR Mode (b) OFB Mode

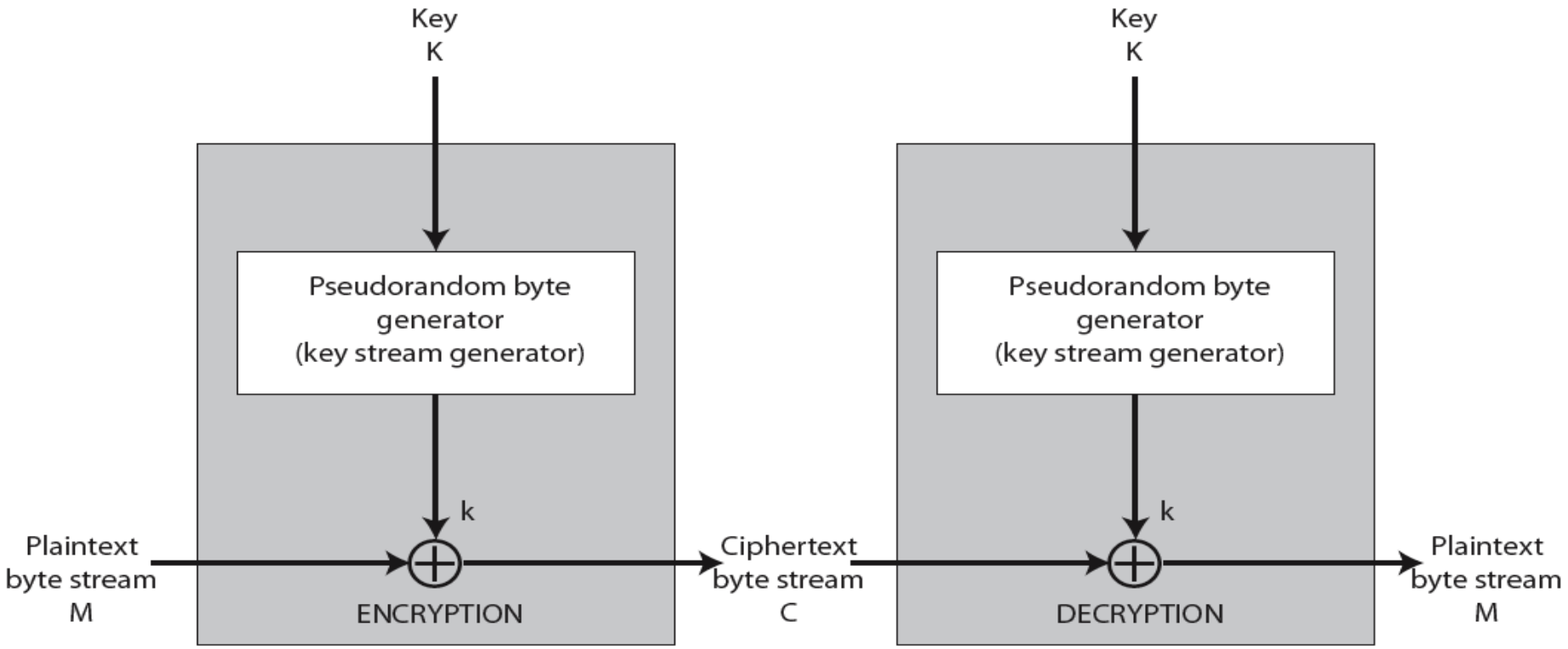
ANSI X9.17 PRG



Stream Ciphers

- process message bit by bit (as a stream)
- have a pseudo random **keystream**
- combined (XOR) with plaintext bit by bit
- randomness of **stream key** completely destroys statistically properties in message
  - $C_i = M_i \text{ XOR } StreamKey_i$
- but must never reuse stream key
  - otherwise can recover messages (cf book cipher)

Stream Cipher Structure



Stream Cipher Properties

- 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

- Proprietary cipher, owned by RSA DSI, Ron Rivest design
- variable key size, byte-oriented stream cipher
- widely used (web SSL/TLS, wireless WEP/WPA)
- key forms random permutation of all 8-bit values
- uses that permutation to scramble input info processed a byte at a time

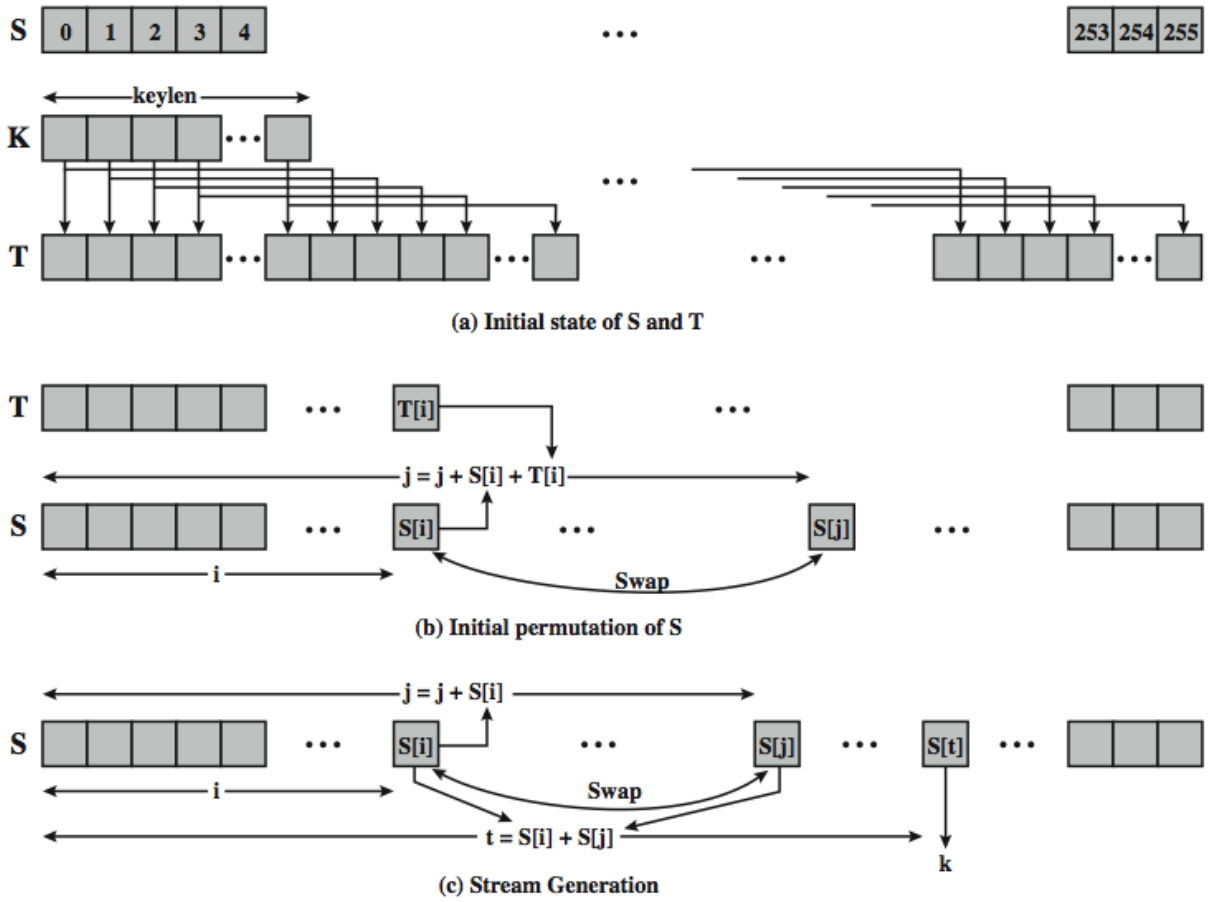
RC4 Key Schedule

- starts with an array S of numbers: 0..255
- use key to well and truly shuffle
- S forms **internal state** of the cipher
  - `for i = 0 to 255 do`
    - `S[i] = i`
    - `T[i] = K[i mod keylen])`
  - `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 Overview



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 rather than RC4 itself

Natural Random Noise

- best source is natural randomness in real world
- find a regular but random event and monitor
- do generally need special h/w to do this
  - eg. radiation counters, radio noise, audio noise, thermal noise in diodes, leaky capacitors, mercury discharge tubes etc
- starting to see such h/w in new CPU's
- problems of **bias** or uneven distribution in signal
  - have to compensate for this when sample, often by passing bits through a hash function
  - best to only use a few noisiest bits from each sample
  - RFC4086 recommends using multiple sources + hash

Published Sources

- a few published collections of random numbers
- Rand Co, in 1955, published 1 million numbers
  - generated using an electronic roulette wheel
  - has been used in some cipher designs cf Khafre
- earlier Tippett in 1927 published a collection
- issues are that:
  - these are limited
  - too well-known for most uses

Public Key Cryptography and RSA

Private-Key Cryptography

- traditional **private/secret/single key** cryptography uses **one** key
- shared by both sender and receiver

- If this key is disclosed, communications are compromised

- **Symmetric**: parties have exact same powers
  - Receiver can forge a message & claim it was sent by sender

Public-Key Cryptography

- probably most significant advance in the 3000 year history of cryptography
- uses **two** keys – a public & a private key
- **asymmetric** since parties are **not** equal
- uses clever application of number theoretic concepts to function
- complements **rather than** replaces private key crypto

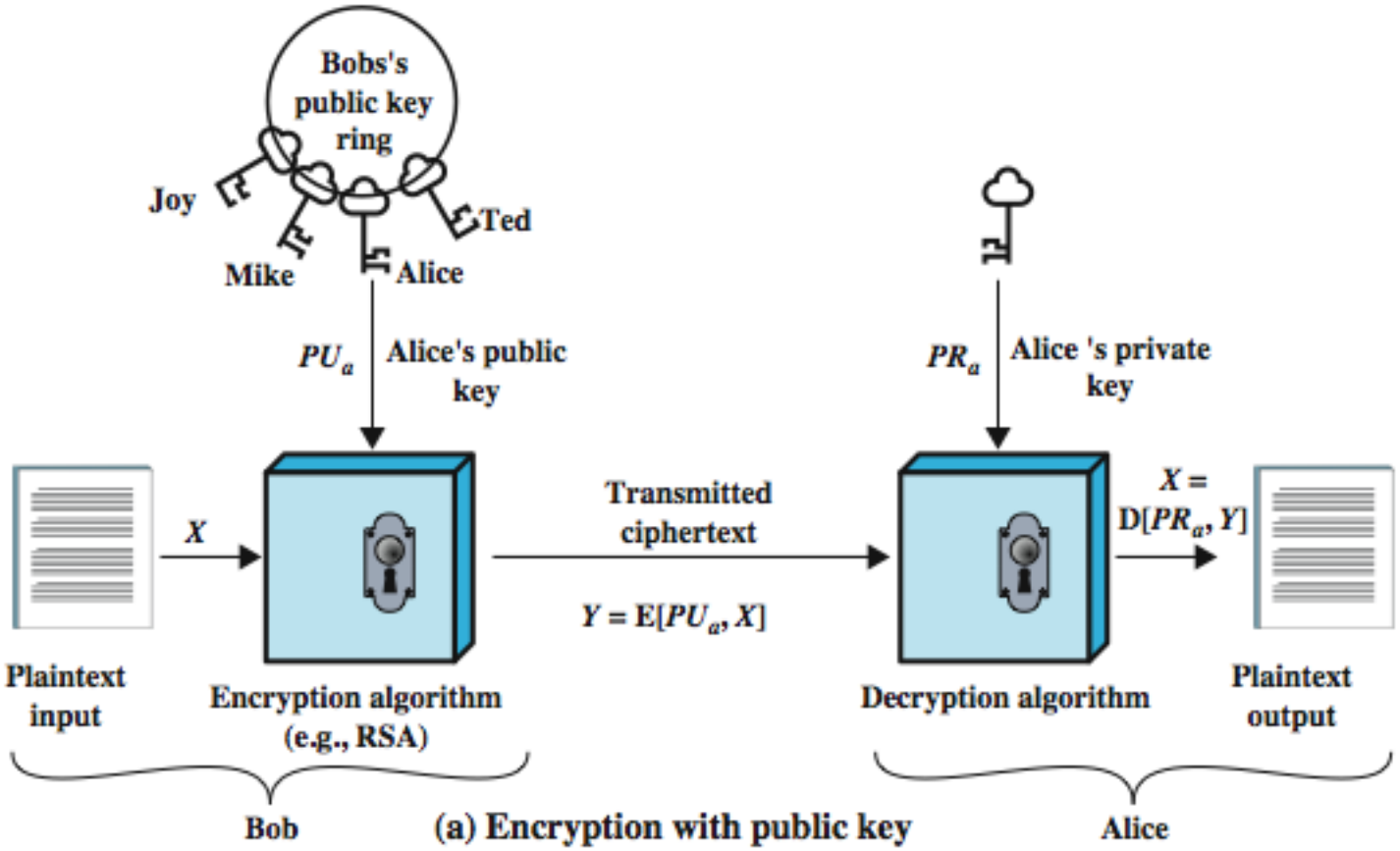
Why Public-Key Cryptography?

- developed to address two key issues:
  - **key distribution** – how to have secure communications in general without having to trust a KDC with your key
  - **digital signatures** – how to verify a message comes intact from the claimed sender
- public invention due to Whitfield Diffie & Martin Hellman at Stanford Uni in 1976
  - known earlier in classified community

Public-Key Cryptography

- **public-key/two-key/asymmetric** cryptography involves the use of **two** keys:
  - a **public-key**, which may be known by anybody, and can be used to **encrypt messages**, and **verify signatures**
  - a related **private-key**, known only to the recipient, used to **decrypt messages**, and **sign** (create) **signatures**
- **infeasible to determine private key from public**
- is **asymmetric** because
  - those who encrypt messages or verify signatures **cannot** decrypt messages or create signatures

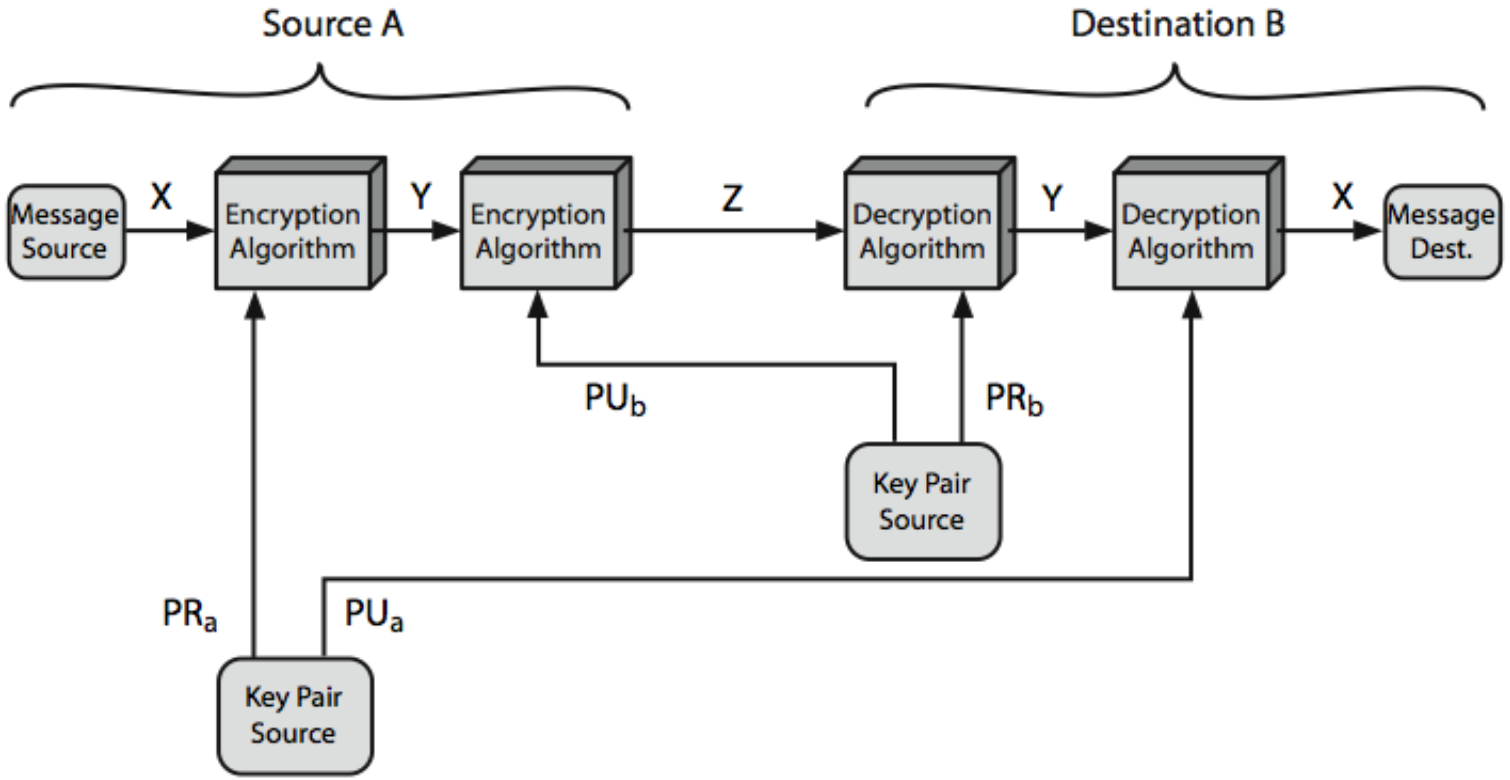
Public-Key Cryptography



Symmetric vs Public-Key

Conventional Encryption	Public-Key Encryption
Needed to Work: 1. The same algorithm with the same key is used for encryption and decryption. 2. The sender and receiver must share the algorithm and the key. *Needed for Security:* 1. The key must be kept secret. 2. It must be impossible or at least impractical to decipher a message if no other information is available. 3. Knowledge of the algorithm plus samples of ciphertext must be insufficient to determine the key.	*Needed to Work:* 1. One algorithm is used for encryption and decryption with a pair of keys, one for encryption and one for decryption. 2. The sender and receiver must each have one of the matched pair of keys (not the same one). *Needed for Security:* 1. One of the two keys must be kept secret. 2. It must be impossible or at least impractical to decipher a message if no other information is available. 3. Knowledge of the algorithm plus one of the keys plus samples of ciphertext must be insufficient to determine the other key.

Public-Key Cryptosystems



Public-Key Applications

- can classify uses into 3 categories:
  - **encryption/decryption** (provide secrecy)
  - **digital signatures** (provide authentication)
  - **key exchange** (of session keys)
- some algorithms are suitable for all uses, others are specific to one

Algorithm	Encryption/Decryption	Digital Signature	Key Exchange
RSA	Yes	Yes	Yes
Elliptic Curve	Yes	Yes	Yes
Diffie-Hellman	No	No	Yes
DSS	No	Yes	No

Public-Key Requirements

- Public-Key algorithms rely on two keys where:
  - it is computationally infeasible to find decryption key knowing only algorithm & encryption key
  - it is computationally easy to en/decrypt messages when the relevant (en/decrypt) key is known
  - either of the two related keys can be used for encryption, with the other used for decryption (for some algorithms)
- these are formidable requirements which only a few algorithms have satisfied

Public-Key Requirements

- need a trapdoor one-way function
- one-way function has
  - $Y = f(X)$ easy
  - $X = f^{-1}(Y)$ infeasible
- a trap-door one-way function has
  - $Y = f_k(X)$ easy, if k and X are known
  - $X = f_k^{-1}(Y)$ easy, if k and Y are known
  - $X = f_k^{-1}(Y)$ infeasible, if Y known but k not known
- a practical public-key scheme depends on a suitable trap-door one-way function

Security of Public Key Schemes

- like private key schemes brute force **exhaustive search** attack is always theoretically possible
- but keys used are too large (>512bits)
- security relies on a **large enough** difference in difficulty between **easy** (en/decrypt) and **hard** (cryptanalyse) problems
- more generally the **hard** problem is known, but is made hard enough to be impractical to break
- requires the use of **very large numbers**
- hence is **slow** compared to private key schemes

RSA

- by Rivest, Shamir & Adleman of MIT in 1977
- best known & widely used public-key scheme
- based on exponentiation in a finite (Galois) field over integers modulo a prime
  - nb. exponentiation takes $O((\log n)^3)$ operations (easy)
- uses large integers (eg. 1024 bits)
- security due to cost of factoring large numbers
  - nb. factorization takes $O(e^{\log n \log \log n})$ operations (hard)

RSA En/decryption

- to encrypt a message M the sender:
  - obtains **public key** of recipient $PU=\{e,n\}$
  - computes: $C = M^e \bmod n$, where $0 \leq M < n$
- to decrypt the ciphertext C the owner:
  - uses their private key $PR=\{d,n\}$
  - computes: $M = C^d \bmod n$
- note that the message M must be smaller than the modulus n (block if needed)

RSA Key Setup

- each user generates a public/private key pair by:
- selecting two large primes at random: p, q
- computing their system modulus $n=p.q$
  - note $\varnothing(n)=(p-1)(q-1)$
- selecting at random the encryption key e
  - where $1<e<\varnothing(n), \gcd(e,\varnothing(n))=1$
- solve following equation to find decryption key d
  - $e.d=1 \bmod \varnothing(n)$ and $0 \leq d \leq n$
- publish their public encryption key: PU={e,n}
- keep secret private decryption key: PR={d,n}

Why RSA Works

- because of Euler's Theorem:
  - $a^{\phi(n)} \bmod n = 1$ where $\gcd(a,n)=1$
- in RSA have:
  - $n=p.q$
  - $\phi(n)=(p-1)(q-1)$
  - carefully chose e & d to be inverses $\bmod \phi(n)$
  - hence $e.d=1+k.\phi(n)$ for some k
- hence :

$$C^d = M^{e.d} = M^{1+k.\phi(n)} = M^1.(M^{\phi(n)})^k$$
$$= M^1.(1)^k = M^1 = M \bmod n$$

RSA Example - Key Setup

1. Select primes: $p=17$ & $q=11$
2. Calculate $n = pq = 17 \times 11 = 187$
3. Calculate $\phi(n) = (p-1)(q-1) = 16 \times 10 = 160$
4. Select e: $\gcd(e,160)=1$; choose $e=7$
5. Determine d: $de=1 \mod 160$ and $d < 160$ Value is $d=23$ since $23 \times 7 = 161 = 10 \times 160 + 1$
6. Publish public key $PU=\{7,187\}$
7. Keep secret private key $PR=\{23,187\}$

RSA Example - En/Decryption

- sample RSA encryption/decryption is:
- given message $M = 88$ (nb. $88<187$)
- encryption:
  - $C = 88^7 \bmod 187 = 11$
- decryption:
  - $M = 11^{23} \bmod 187 = 88$

Exponentiation

- can use the Square and Multiply Algorithm
- a fast, efficient algorithm for exponentiation
- concept is based on repeatedly squaring base
- and multiplying in the ones that are needed to compute the result
- look at binary representation of exponent
- only takes $O(\log_2 n)$ multiples for number n
  - eg. $7^5 = 7^4.7^1 = 3.7 = 10 \bmod 11$
  - eg. $3^{129} = 3^{128}.3^1 = 5.3 = 4 \bmod 11$

Exponentiation

```
c = 0; f = 1
for i = k downto 0
    do c = 2 x c
       f = (f x f) mod n
    if b_i == 1 then
       c = c + 1
     f = (f x a) mod n
return f
```

Efficient Encryption

- encryption uses exponentiation to power e
- hence if e small, this will be faster
  - often choose e=65537 (2^{16}-1)
  - also see choices of e=3 or e=17
- but if e too small (e.g. e=3) can attack
  - using Chinese remainder theorem & 3 messages with different modulii
- if e fixed must ensure $\gcd(e,\phi(n))=1$
  - ie reject any p or q not relatively prime to e

Efficient Decryption

- decryption uses exponentiation to power d
  - this is likely large, insecure if not
- can use the Chinese Remainder Theorem (CRT) to compute mod p & q separately. then combine to get desired answer
  - approx 4 times faster than doing directly
- only owner of private key who knows values of p & q can use this technique

RSA Key Generation

- users of RSA must:
  - determine two primes at random - p, q
  - select either e or d and compute the other
- primes p, q must not be easily derived from modulus $n=p.q$
  - means must be sufficiently large
  - typically guess and use probabilistic test
- exponents e, d are inverses, so use Inverse algorithm to compute the other

RSA Security

- possible approaches to attacking RSA are:
  - brute force key search - infeasible given size of numbers
  - mathematical attacks - based on difficulty of computing ø(n), by factoring modulus n
  - timing attacks - on running of decryption
  - chosen ciphertext attacks - given properties of RSA

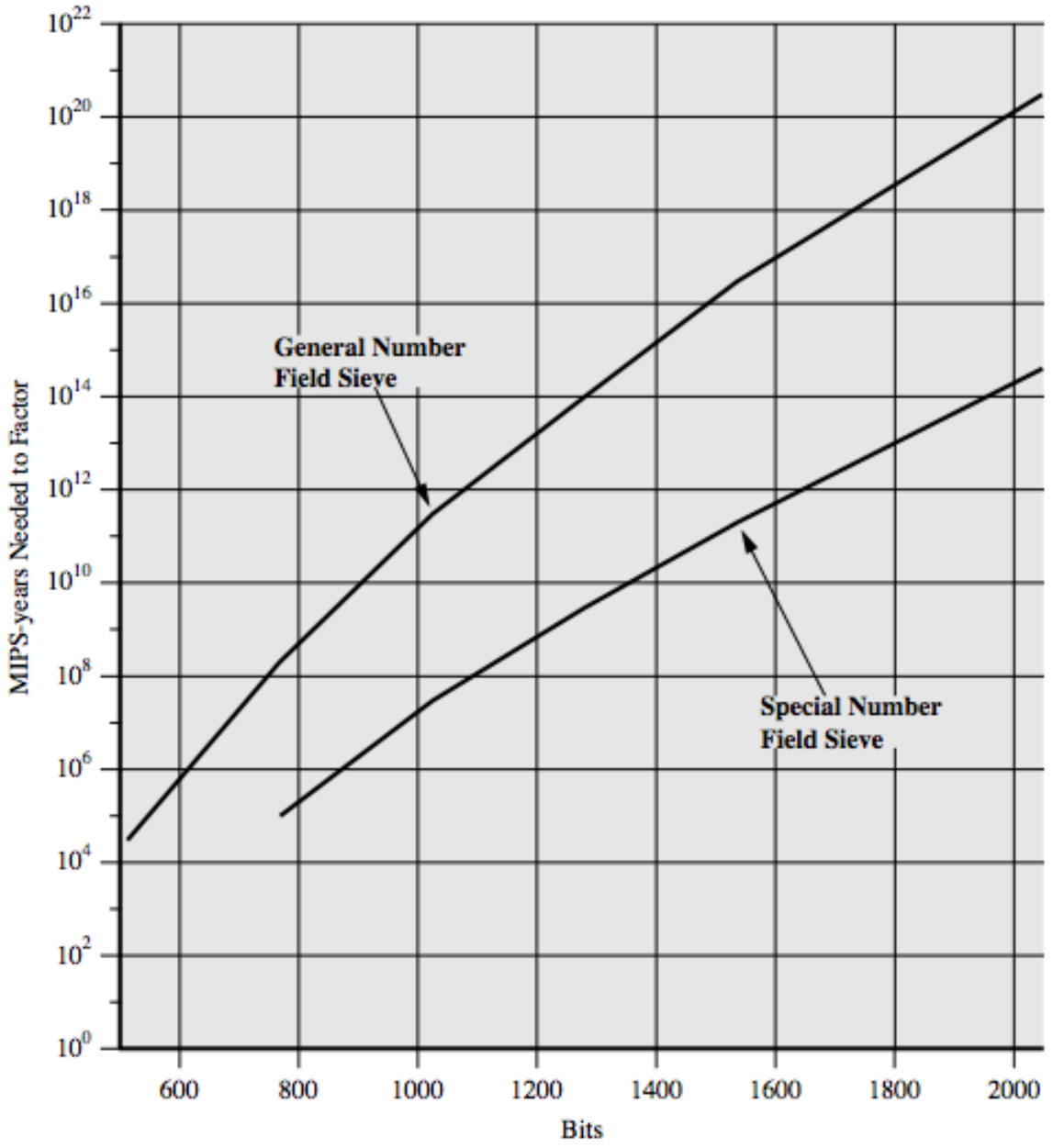
Factoring Problem

- mathematical approach takes 3 forms:
  - factor $n=p.q$, hence compute $\varnothing(n)$ and then d
  - determine $\varnothing(n)$ directly and compute d
  - find d directly
- currently believe all equivalent to factoring
  - have seen slow improvements over the years
    - as of May-05 best is 200 decimal digits (663) bit with LS
  - biggest improvement comes from improved algorithm
    - cf QS to GHFS to LS
  - currently assume 1024-2048 bit RSA is secure
    - ensure p, q of similar size and matching other constraints

Progress in Factoring

Number of Decimal Digits	Approximate Number of Bits	Date Achieved	MIPS-years	Algorithm
100	332	April 1991	7	quadratic sieve
110	365	April 1992	75	quadratic sieve
120	398	June 1993	830	quadratic sieve
129	428	April 1994	5000	quadratic sieve
130	431	April 1996	1000	generalized number field sieve
140	465	February 1999	2000	generalized number field sieve
155	512	August 1999	8000	generalized number field sieve
160	530	April 2003	—	Lattice sieve
174	576	December 2003	—	Lattice sieve
200	663	May 2005	—	Lattice sieve

Progress in Factoring



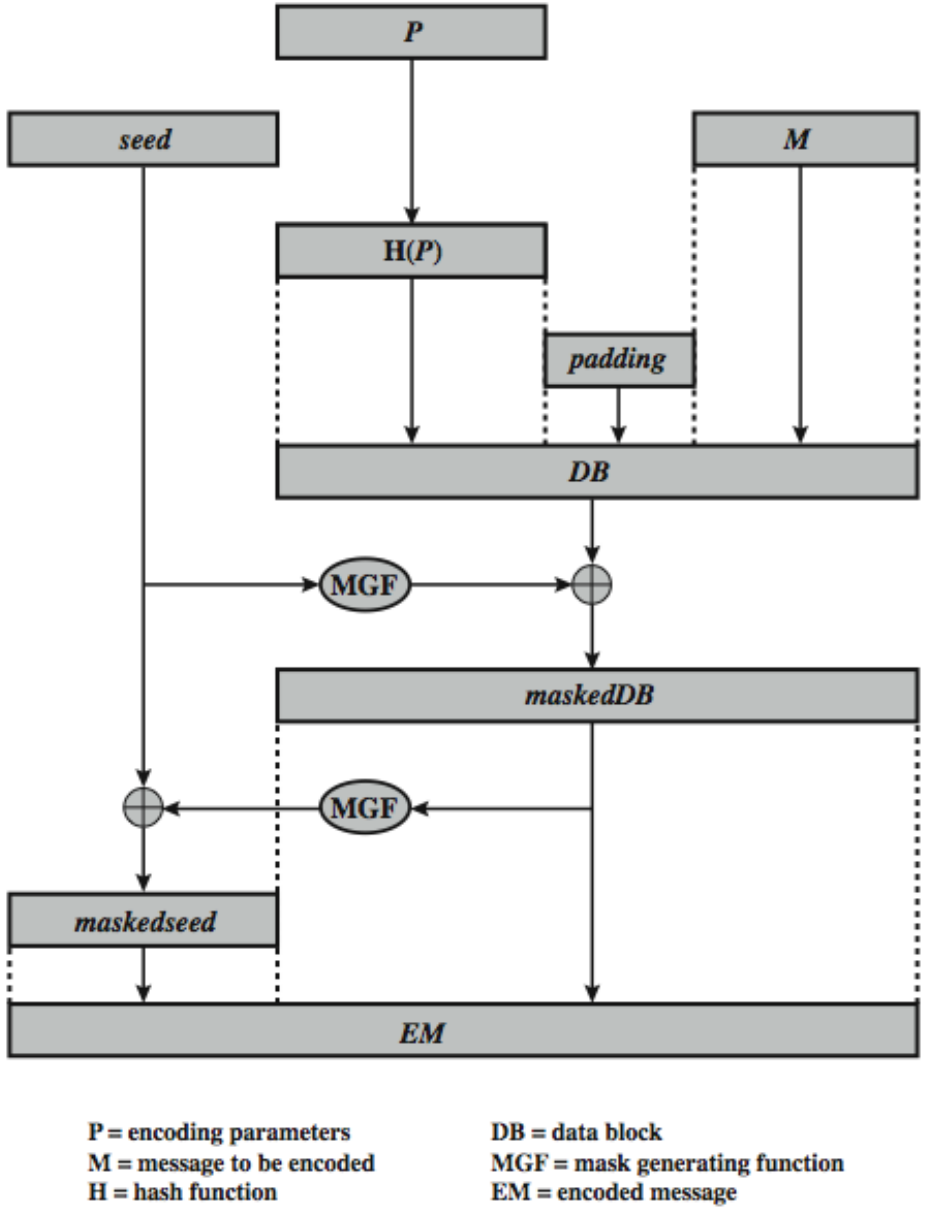
Timing Attacks

- Paul Kocher, mid-1990's : exploit timing variations in operations
  - eg. multiplying by small vs large number
  - or IF's varying which instructions executed
- infer operand size based on time taken
- For RSA: exploit time taken in exponentiation
- countermeasures
  - use constant exponentiation time
  - add random delays
  - blind values used in calculations

Chosen Ciphertext Attacks

- RSA is vulnerable to a Chosen Ciphertext Attack (CCA)
- attackers chooses ciphertexts & gets decrypted plaintext back
- choose ciphertext to exploit properties of RSA to provide info to help cryptanalysis
- can counter with random pad of plaintext
- or use Optimal Asymmetric Encryption Padding (OASP)

Optimal Asymmetric Encryption Padding (OASP)



Summary

- have considered:
  - principles of public-key cryptography
  - RSA algorithm, implementation, security

Other Public Key Cryptosystems

Diffie-Hellman Key Exchange

- first public-key type scheme proposed
- by Diffie & Hellman in 1976 along with the exposition of public key concepts
  - note: now know that Williamson (UK CESG) secretly proposed the concept in 1970
- is a practical method for public exchange of a secret key
- used in a number of commercial products

Diffie-Hellman Key Exchange

- a public-key distribution scheme
  - cannot be used to exchange an arbitrary message
  - rather it can establish a common key
  - known only to the two participants
- value of key depends on the participants (and their private and public key information)
- based on exponentiation in a finite (Galois) field (modulo a prime or a polynomial) - easy
- security relies on the difficulty of computing discrete logarithms (similar to factoring) – hard

Diffie-Hellman Setup

- all users agree on global parameters:
  - large prime integer or polynomial q
  - a being a primitive root mod q
- each user (eg. A) generates their key
  - chooses a secret key (number): $x_A < q$
  - compute their **public key**: $y_A = a^{x_A} \bmod q$
- each user makes public that key y_A

Diffie-Hellman Key Exchange

- shared session key for users A & B is K_{AB}:
  - $K_{AB} = a^{x_A \cdot x_B} \bmod q$
  - $= y_A^{x_B} \bmod q$ (which **B** can compute)
  - $= y_B^{x_A} \bmod q$ (which **A** can compute)
- K_{AB} is used as session key in private-key encryption scheme between Alice and Bob
- if Alice and Bob subsequently communicate, they will have the **same** key as before, unless they choose new public-keys
- attacker needs an x, must solve discrete log

Diffie-Hellman Example

- users Alice & Bob who wish to swap keys:
- agree on prime $q=353$ and $a=3$
- select random secret keys:
  - A chooses $x_A=97$, B chooses $x_B=233$
- compute respective public keys:
  - $y_A=3^{97} \mod 353 = 40$ (Alice)
  - $y_B=3^{233} \mod 353 = 248$ (Bob)
- compute shared session key as:
  - $K_{AB}= y_B^{x_A} \mod 353 = 248^{97} = 160$ (Alice)
  - $K_{AB}= y_A^{x_B} \mod 353 = 40^{233} = 160$ (Bob)

Key Exchange Protocols

- users could create random private/public D-H keys each time they communicate
- users could create a known private/public D-H key and publish in a directory, then consulted and used to securely communicate with them
- both of these are vulnerable to a Man-in-the-Middle Attack
- authentication of the keys is needed

Man-in-the-Middle Attack

1. Darth prepares by creating two private / public keys
2. Alice transmits her public key to Bob
3. Darth intercepts this and transmits his first public key to Bob. Darth also calculates a shared key with Alice
4. Bob receives the public key and calculates the shared key (with Darth instead of Alice)
5. Bob transmits his public key to Alice
6. Darth intercepts this and transmits his second public key to Alice. Darth calculates a shared key with Bob
7. Alice receives the key and calculates the shared key (with Darth instead of Bob)

➢ Darth can then intercept, decrypt, re-encrypt, forward all messages between Alice & Bob

ElGamal Cryptography

- public-key cryptosystem related to D-H
- uses exponentiation in a finite field
- with security based difficulty of computing discrete logarithms, as in D-H
- each user (eg. A) generates their key
  - chooses a secret key (number): $1 < x_A < q-1$
  - compute their **public key**: $y_A = a^{x_A} \bmod q$

ElGamal Message Exchange

- Bob encrypts a message to send to A computing
  - represent message M in range $0 <= M <= q-1$
    - longer messages must be sent as blocks
  - chose random integer k with $1 <= k <= q-1$
  - compute one-time key $K = y_A^k \bmod q$
  - encrypt M as a pair of integers (C_1, C_2) where
    - $C_1 = a^k \bmod q$; $C_2 = KM \bmod q$
- A then recovers message by
  - recovering key K as $K = C_1^{xA} \bmod q$
  - computing M as $M = C_2 \ K^{-1} \bmod q$
- a unique k must be used each time
  - otherwise result is insecure

ElGamal Example

- use field GF(19) $q=19$ and $a=10$
- Alice computes her key:
  - A chooses $x_A=5$ & computes $y_A=10^5 \bmod 19 = 3$
- Bob send message $m=17$ as $(11,5)$ by
  - chosing random $k=6$
  - computing $K = y_A^k \bmod q = 3^6 \bmod 19 = 7$
  - computing $C_1 = a^k \bmod q = 10^6 \bmod 19 = 11;$
  - $C_2 = KM \bmod q = 7.17 \bmod 19 = 5$
- Alice recovers original message by computing:
  - recover $K = C_1^{xA} \bmod q = 11^5 \bmod 19 = 7$
  - compute inverse $K^{-1} = 7^{-1} = 11$
  - recover $M = C_2\ K^{-1} \bmod q = 5.11 \bmod 19 = 17$

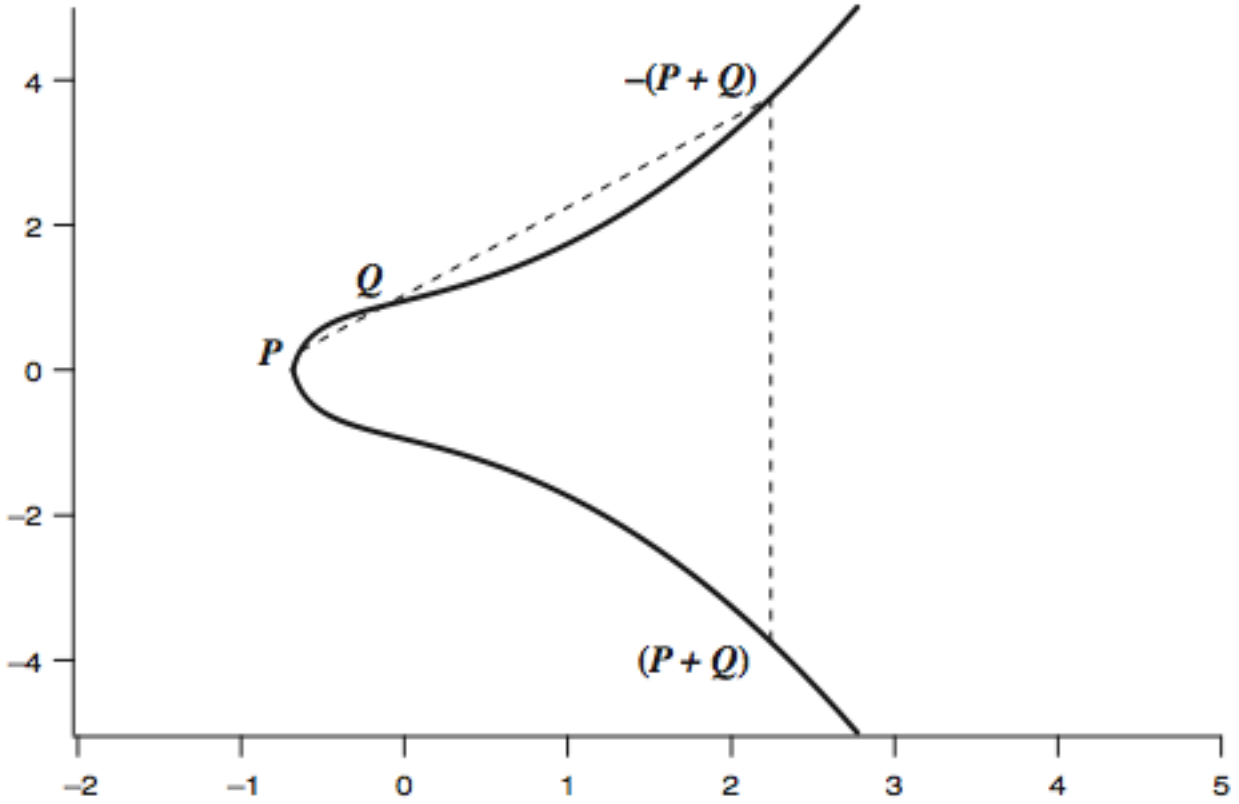
Elliptic Curve Cryptography

- majority of public-key crypto (RSA, D-H) use either integer or polynomial arithmetic with very large numbers/polynomials
- imposes a significant load in storing and processing keys and messages
- an alternative is to use elliptic curves
- offers same security with smaller bit sizes
- newer, but not as well analysed

Real Elliptic Curves

- an elliptic curve is defined by an equation in two variables x & y, with coefficients
- consider a cubic elliptic curve of form
  - $y^2 = x^3 + ax + b$
  - where x,y,a,b are all real numbers
  - also define zero point O
- consider set of points E(a,b) that satisfy
- have addition operation for elliptic curve
  - geometrically sum of P+Q is reflection of the intersection R

Real Elliptic Curve Example



(b) $y^2 = x^3 + x + 1$

Finite Elliptic Curves

- Elliptic curve cryptography uses curves whose variables & coefficients are finite
- have two families commonly used:
  - prime curves $E_p(a,b)$ defined over Z_p
    - use integers modulo a prime
    - best in software
  - binary curves $E_{2m}(a,b)$ defined over GF(2^n)
    - use polynomials with binary coefficients
    - best in hardware

Elliptic Curve Cryptography

- ECC addition is analog of modulo multiply
- ECC repeated addition is analog of modulo exponentiation
- need “hard” problem equiv to discrete log
  - $Q=kP$, where Q,P belong to a prime curve
  - is “easy” to compute Q given k,P
  - but “hard” to find k given Q,P
  - known as the elliptic curve logarithm problem
- Certicom example: $E_{23}(9,17)$

ECC Diffie-Hellman

- can do key exchange analogous to D-H
- users select a suitable curve $E_q(a,b)$
- select base point $G=(x_1, y_1)$
  - with large order n s.t. $nG=O$
- A & B select private keys $n_A<n$, $n_B<n$
- compute public keys: $P_A=n_AG$, $P_B=n_BG$
- compute shared key: $K=n_AP_B$, $K=n_BP_A$
  - same since $K=n_An_BG$
- attacker would need to find k, hard

ECC Encryption/Decryption

- several alternatives, will consider simplest
- must first encode any message M as a point on the elliptic curve P_m
- select suitable curve & point G as in D-H
- each user chooses private key $n_A < n$
- and computes public key $P_A = n_A G$
- to encrypt P_m : $C_m = \{kG, \ P_m + kP_b\}$, k random
- decrypt C_m compute:
  - $P_m + kP_b - n_B(kG) = P_m + k(n_B G) - n_B(kG) = P_m$

ECC Security

- relies on elliptic curve logarithm problem
- fastest method is “Pollard rho method”
- compared to factoring, can use much smaller key sizes than with RSA etc
- for equivalent key lengths computations are roughly equivalent
- hence for similar security ECC offers significant computational advantages

Comparable Key Sizes for Equivalent Security

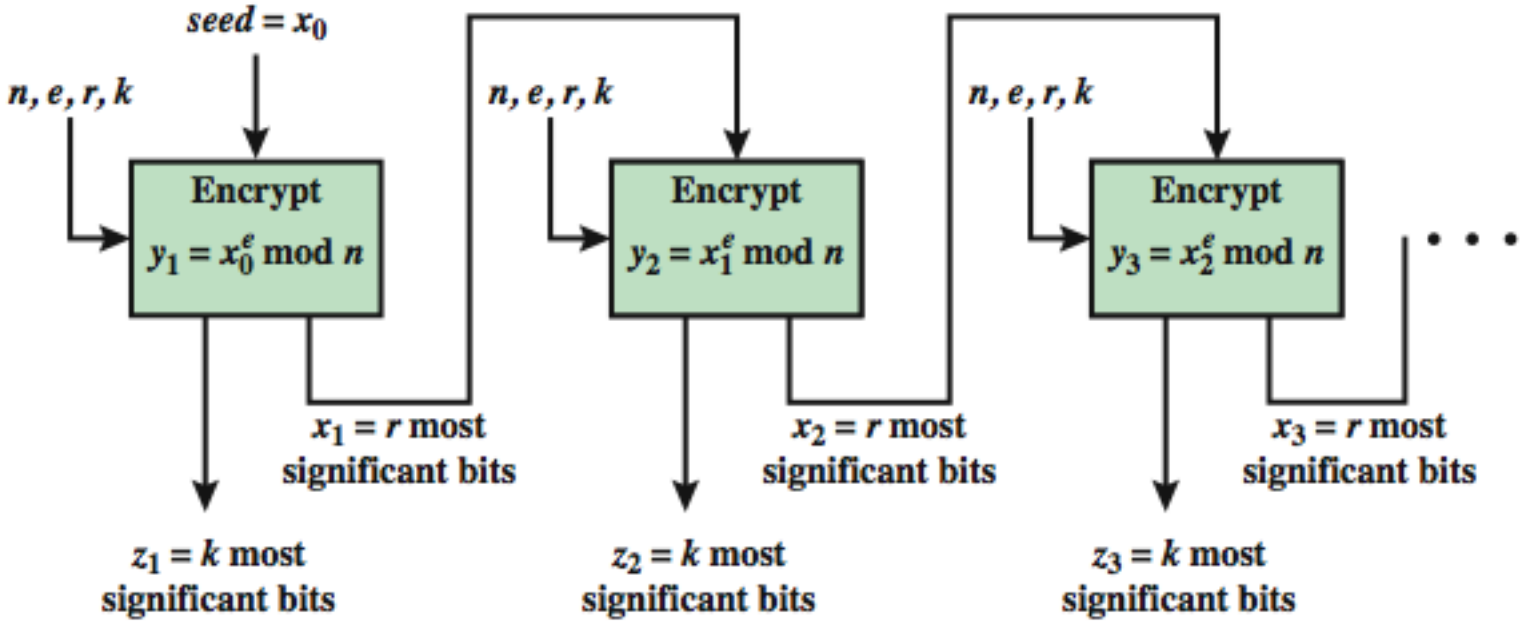
Symmetric scheme (key size in bits)	ECC-based scheme (size of *n* in bits)	RSA/DSA (modulus size in bits)
56	112	512
80	160	1024
112	224	2048
128	256	3072
192	384	7680
256	512	15360

Pseudorandom Number Generation (PRNG) based on Asymmetric Ciphers

- asymmetric encryption algorithm produce apparently random output
- hence can be used to build a pseudorandom number generator (PRNG)
- much slower than symmetric algorithms
- hence only use to generate a short pseudorandom bit sequence (eg. key)

PRNG based on RSA

- have Micali-Schnorr PRNG using RSA
  - in ANSI X9.82 and ISO 18031

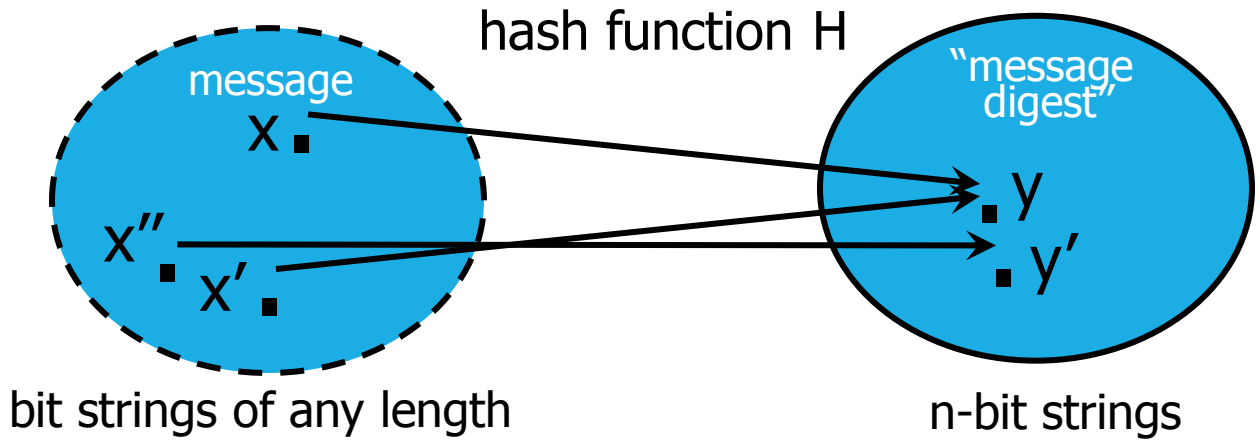


Summary

- have considered:
  - Diffie-Hellman key exchange
  - ElGamal cryptography
  - Elliptic Curve cryptography
  - Pseudorandom Number Generation (PRNG) based on Asymmetric Ciphers (RSA & ECC)

Cryptographic Hash Functions

Hash Functions: Main Idea



Hash function H is a lossy compression function

- Collision: H(x)=H(x') for some inputs x≠x'

H(x) should look "random"

- Every bit (almost) equally likely to be 0 or 1

A cryptographic hash function must have certain properties

One-Way

Intuition: hash should be hard to invert

- "Preimage resistance"
- Given a random, it should be hard to find any x such that h(x)=y
  - y is an n-bit string randomly chosen from the output space of the hash function, ie, y=h(x') for some x'

How hard?

- Brute-force: try every possible x, see if h(x)=y
- SHA-1 (a common hash function) has 160-bit output
  - Suppose we have hardware that can do 2^{30} trials a pop
  - Assuming 2^{34} trials per second, can do 2^{89} trials per year
  - Will take 2^{71} years to invert SHA-1 on a random image

Birthday Paradox

T people

Suppose each birthday is a random number taken from K days (K=365) – how many possibilities?

- K^T - samples with replacement

How many possibilities that are all different?

- $(K)_T = K(K-1)\ldots(K-T+1)$ - samples without replacement

Probability of no repetition?

- $(K)_T/K^T \approx 1 - T(T-1)/2K$

Probability of repetition?

- $O(T^2)$

Collision Resistance

Should be hard to find $x \neq x'$ such that $h(x)=h(x')$

Birthday paradox

- Let T be the number of values x, x', x''… we need to look at before finding the first pair $x \neq x'$ s.t. $h(x)=h(x')$
- Assuming h is random, what is the probability that we find a repetition after looking at T values? $O(T^2)$
- Total number of pairs? $O(2^n)$
  - n = number of bits in the output of hash function
- Conclusion: $T \approx O(2^{n/2})$

Brute-force collision search is $O(2^{n/2})$, <u>not</u> $O(2^n)$

- For SHA-1, this means $O(2^{80})$ vs. $O(2^{160})$

One-Way vs. Collision Resistance

One-wayness does not imply collision resistance

- Suppose g() is one-way
- Define h(x) as g(x') where x' is x except the last bit
  - h is one-way (cannot invert h without inverting g)
  - Collisions for h are easy to find: for any x, h(x0)=h(x1)

Collision resistance does not imply one-wayness

- Suppose g() is collision-resistant
- Define h(x) to be 0x if x is (n-1)-bit long, else 1g(x)
  - Collisions for h are hard to find: if y starts with 0, then there are no collisions; if y starts with 1, then must find collisions in g
  - h is not one way: half of all y's (those whose first bit is 0) are easy to invert (how?), thus random y is invertible with prob. 1/2

Weak Collision Resistance

Given a randomly chosen x, hard to find x' such that h(x)=h(x')

- Attacker must find collision for a specific x… by contrast, to break collision resistance, enough to find any collision
- Brute-force attack requires $O(2^n)$ time

Weak collision resistance does not imply collision resistance (why?)

Hashing vs. Encryption

Hashing is one-way. There is no "un-hashing"!

- A ciphertext can be decrypted with a decryption key… hashes have no equivalent of "decryption"

Hash(x) looks "random", but can be compared for equality with Hash(x')

- Hash the same input twice → same hash value
- Encrypt the same input twice → different ciphertexts

Cryptographic hashes are also known as "cryptographic checksums" or "message digests"

Application: Password Hashing

Instead of user password, store hash(password)

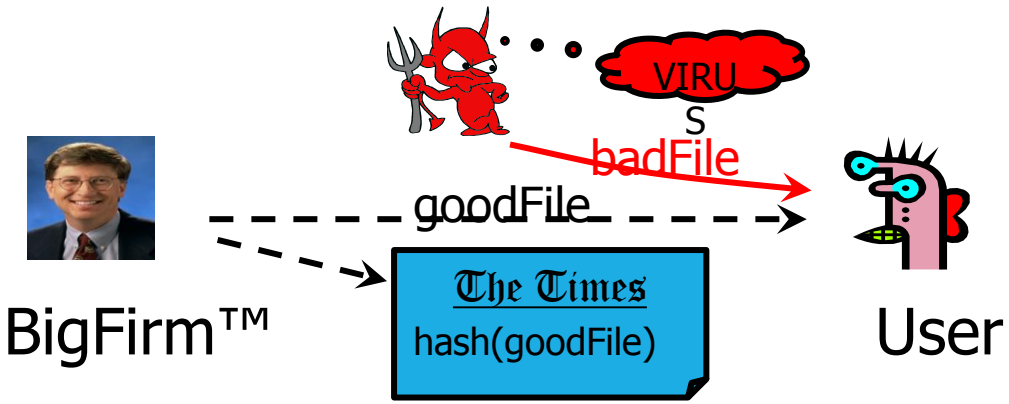
When user enters a password, compute its hash and compare with the entry in the password file

- System does not store actual passwords!
- Cannot go from hash to password!

Why is hashing better than encryption here?

Does hashing protect weak, easily guessable passwords?

Application: Software Integrity



Software manufacturer wants to ensure that the executable file

is received by users without modification…

Sends out the file to users and publishes its hash in the NY Times

The goal is integrity, not secrecy

Idea: given goodFile and hash(goodFile),

very hard to find badFile such that hash(goodFile)=hash(badFile)

Which Property Is Needed?

Passwords stored as hash(password)

- One-wayness: hard to recover entire password
- Passwords are not random and thus guessable

Integrity of software distribution

- Weak collision resistance?
- But software images are not random… maybe need full collision resistance

Auctions: to bid B, send H(B), later reveal B

- One-wayness… but does not protect B from guessing
- Collision resistance: bidder should not be able to find two bids B and B' such that H(B)=H(B')

Common Hash Functions

MD5

- Completely broken by now

RIPEMD-160

- 160-bit variant of MD-5

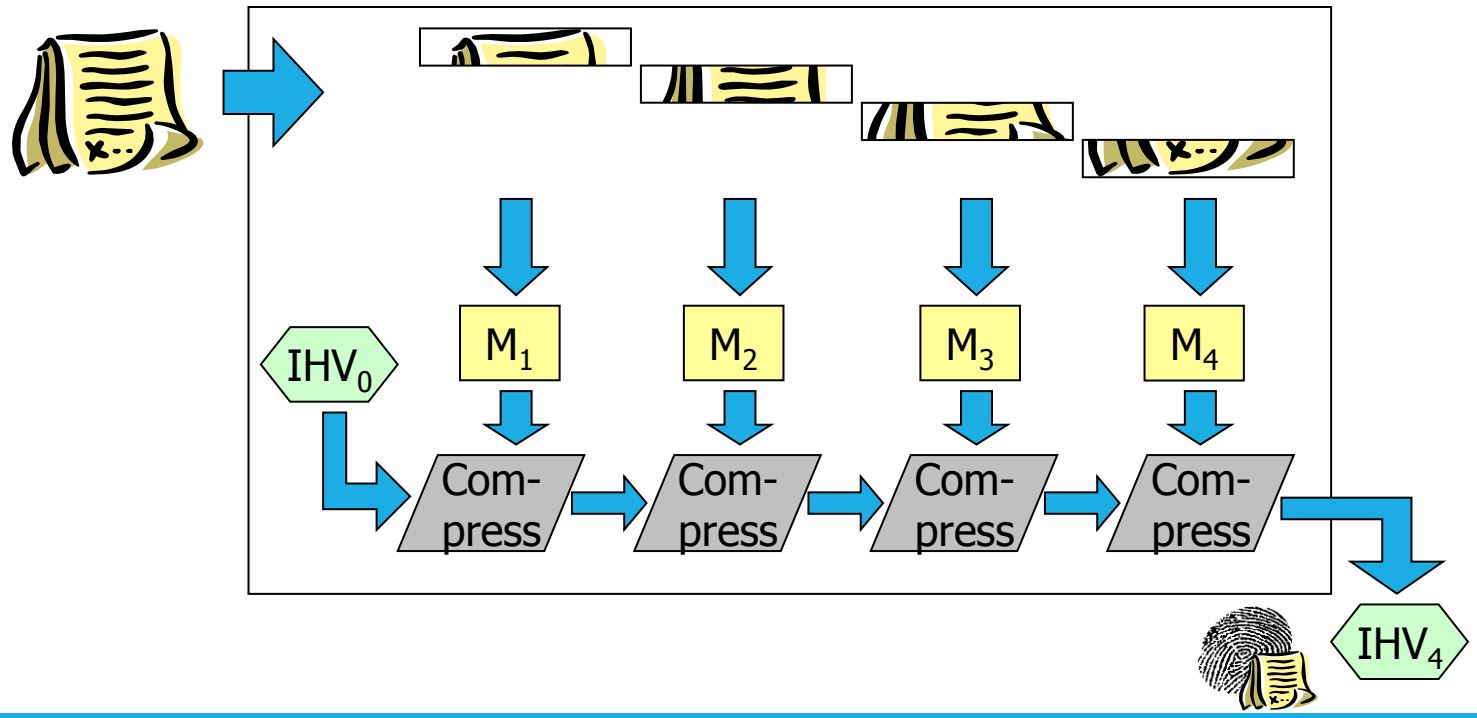
SHA-1 (Secure Hash Algorithm)

- Widely used
- US government (NIST) standard as of 1993-95
  - Also the hash algorithm for Digital Signature Standard (DSS)

Overview of MD5

Designed in 1991 by Ron Rivest

Iterative design using compression function



History of MD5 Collisions

2004: first collision attack

- The only difference between colliding messages is 128 random-looking bytes

2007: chosen-prefix collisions

- For any prefix, can find colliding messages that have this prefix and differ up to 716 random-looking bytes

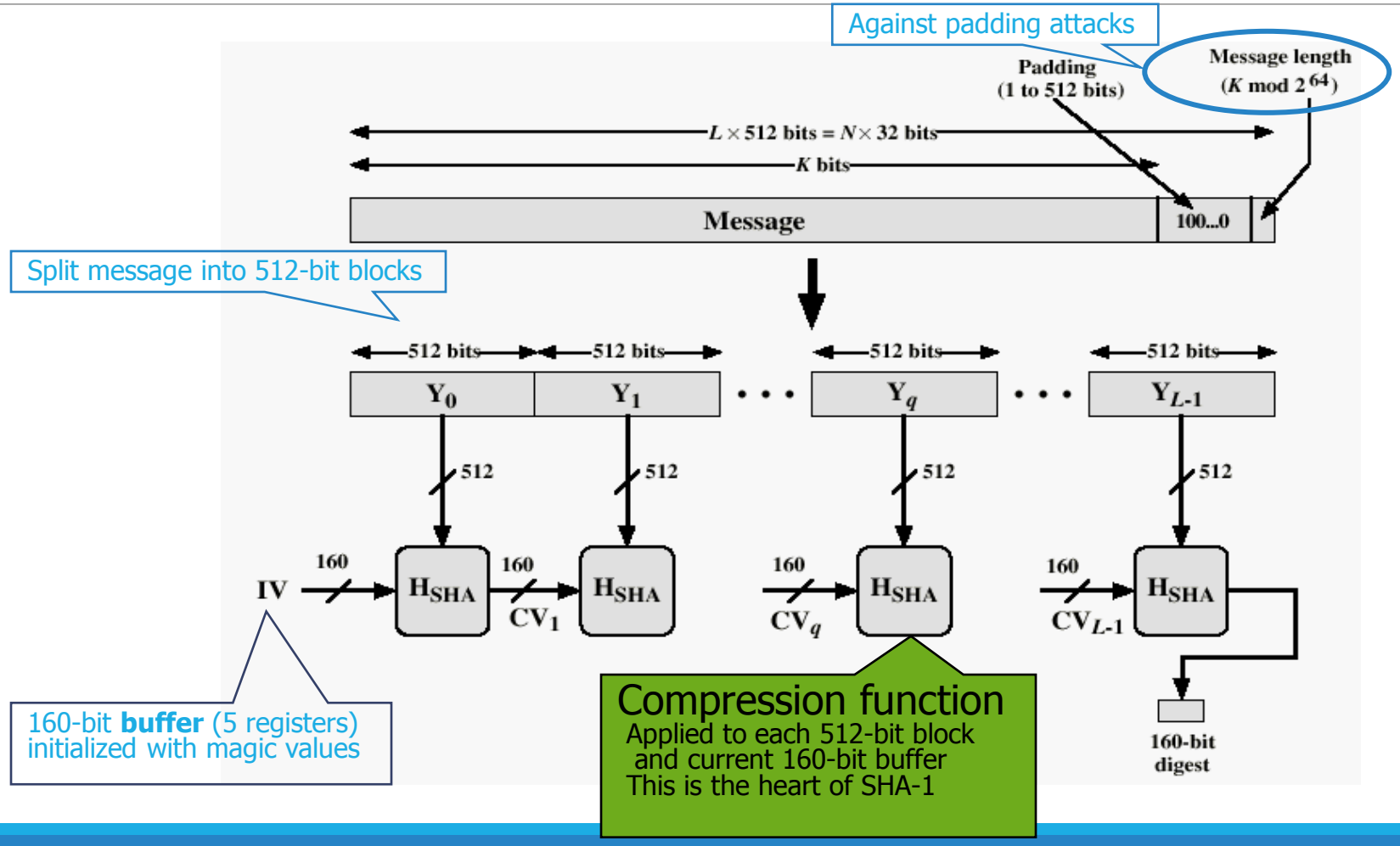
2008: rogue SSL certificates

- Talk about this in more detail when discussing PKI

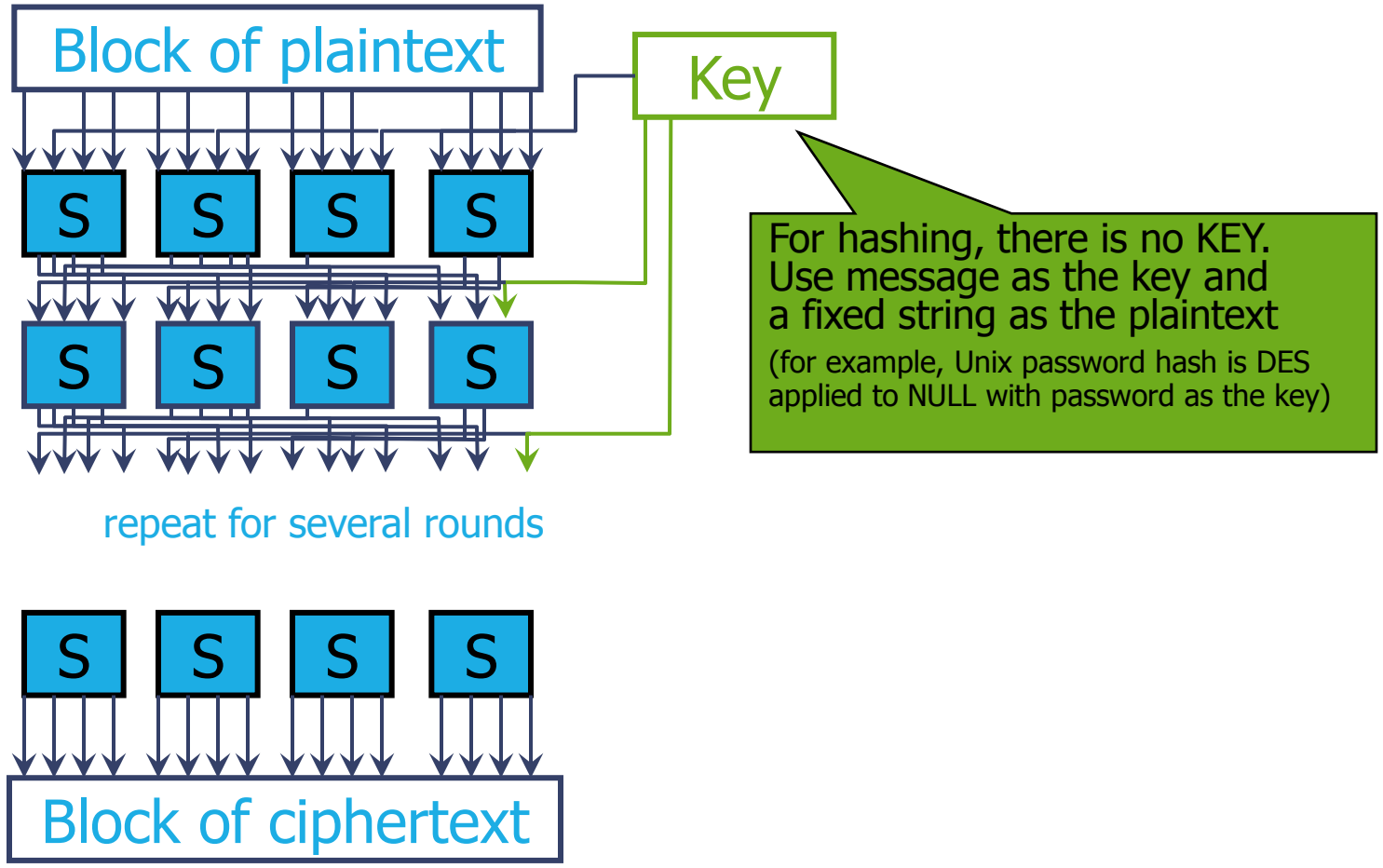
2012: MD5 collisions used in cyberwarfare

- Flame malware uses an MD5 prefix collision to fake a Microsoft digital code signature

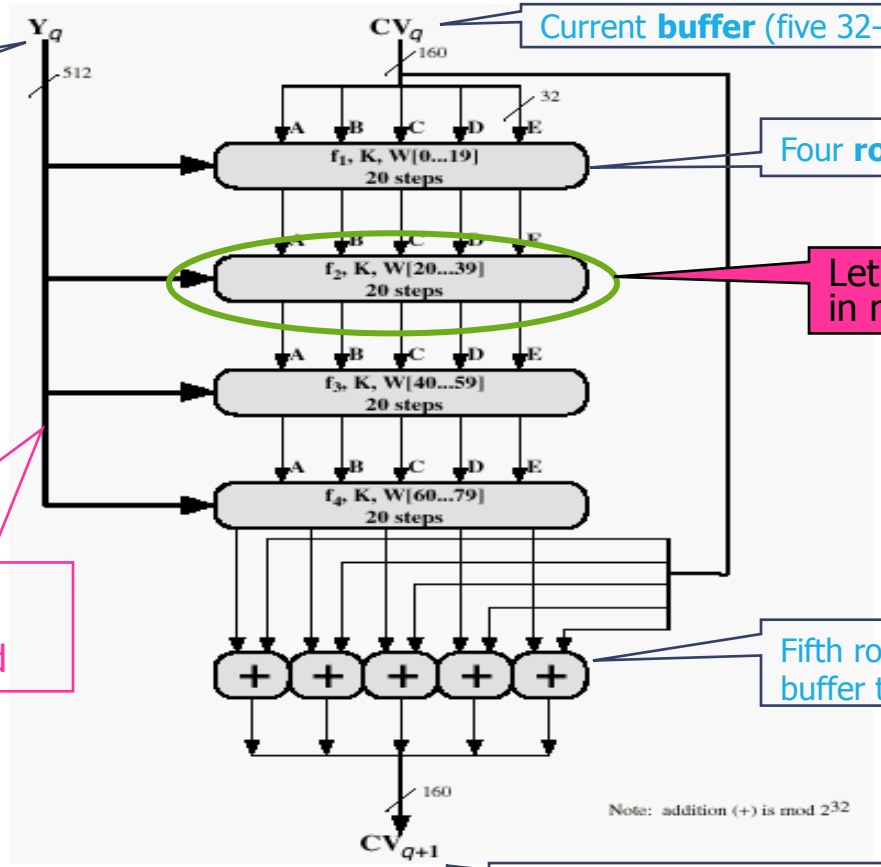
Basic Structure of SHA-1



Block Ciphers



SHA-1 Compression Function



Current **message block**

Current **buffer** (five 32-bit registers A,B,C,D,E)

Four **rounds**, 20 steps in each

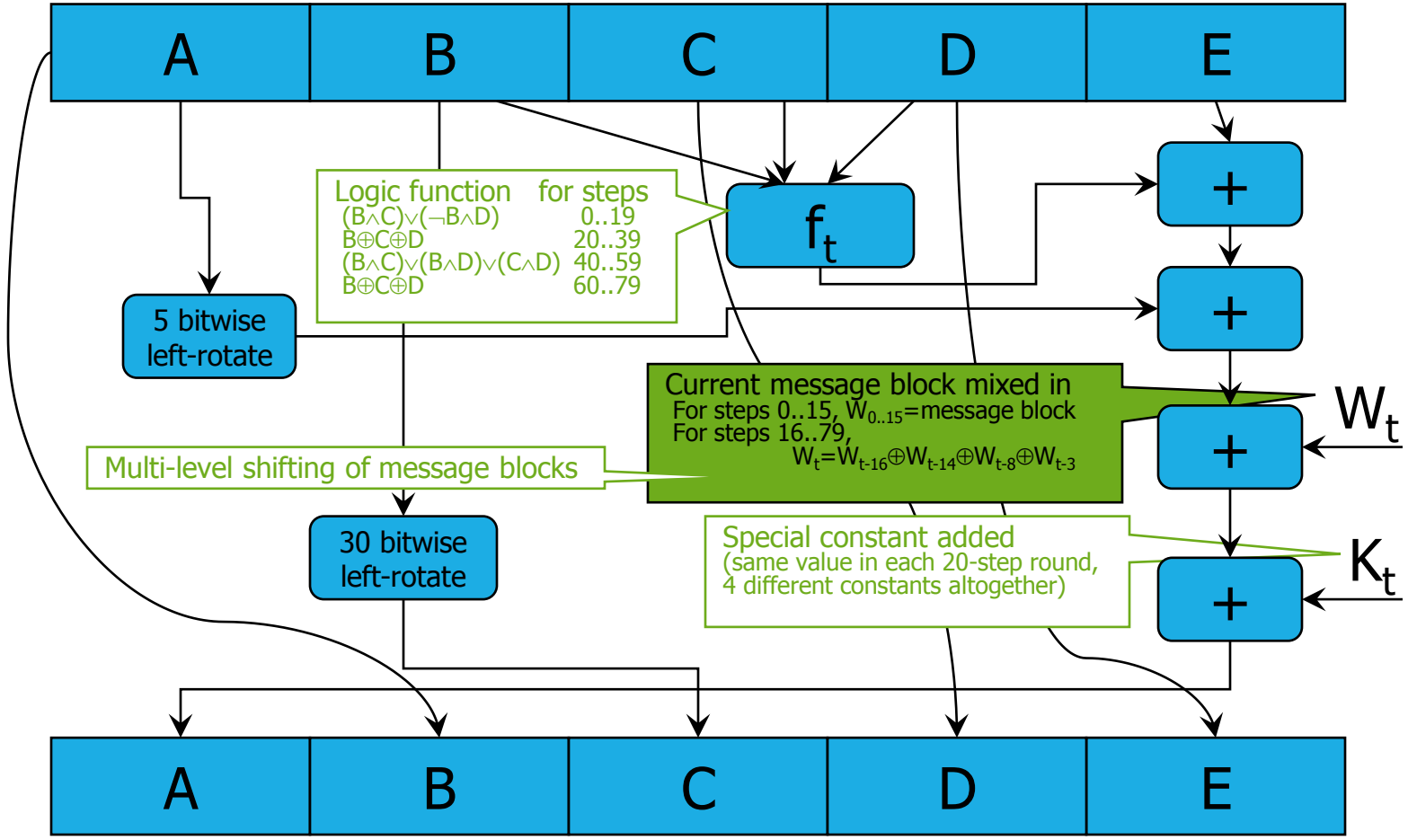
Let's look at each step in more detail...

Similar to a block cipher, with message itself used as the key for each round

Fifth round adds the original buffer to the result of 4 rounds

Buffer contains final hash value

One Step of SHA-1 (80 steps total)



How Strong Is SHA-1?

Every bit of output depends on every bit of input

- Very important property for collision-resistance

Brute-force inversion requires 2^{160} ops, birthday attack on collision resistance requires 2^{80} ops

Some weaknesses discovered in 2005

- Collisions can be found in 2^{63} ops

NIST Competition

A public competition to develop a new cryptographic hash algorithm

- Organized by NIST (read: NSA)

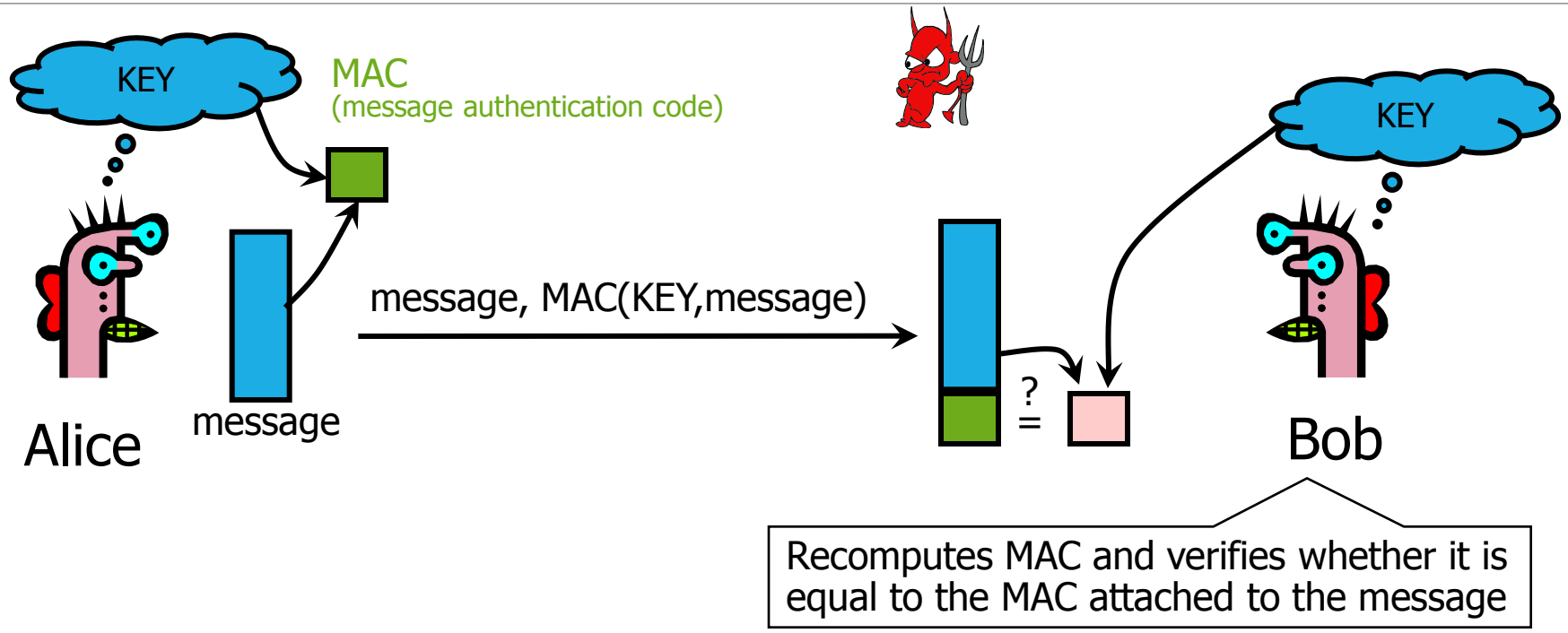
64 entries into the competition (Oct 2008)

5 finalists in 3rd round (Dec 2010)

Winner: Keccak (Oct 2012)

- Will be standardized as SHA-3

Integrity and Authentication



Integrity and authentication: only someone who knows KEY can compute correct MAC for a given message

HMAC

Construct MAC from a cryptographic hash function

- Invented by Bellare, Canetti, and Krawczyk (1996)
- Used in SSL/TLS, mandatory for IPsec

Why not encryption?

- Hashing is faster than encryption
- Library code for hash functions widely available
- Can easily replace one hash function with another
- There used to be US export restrictions on encryption

Structure of HMAC