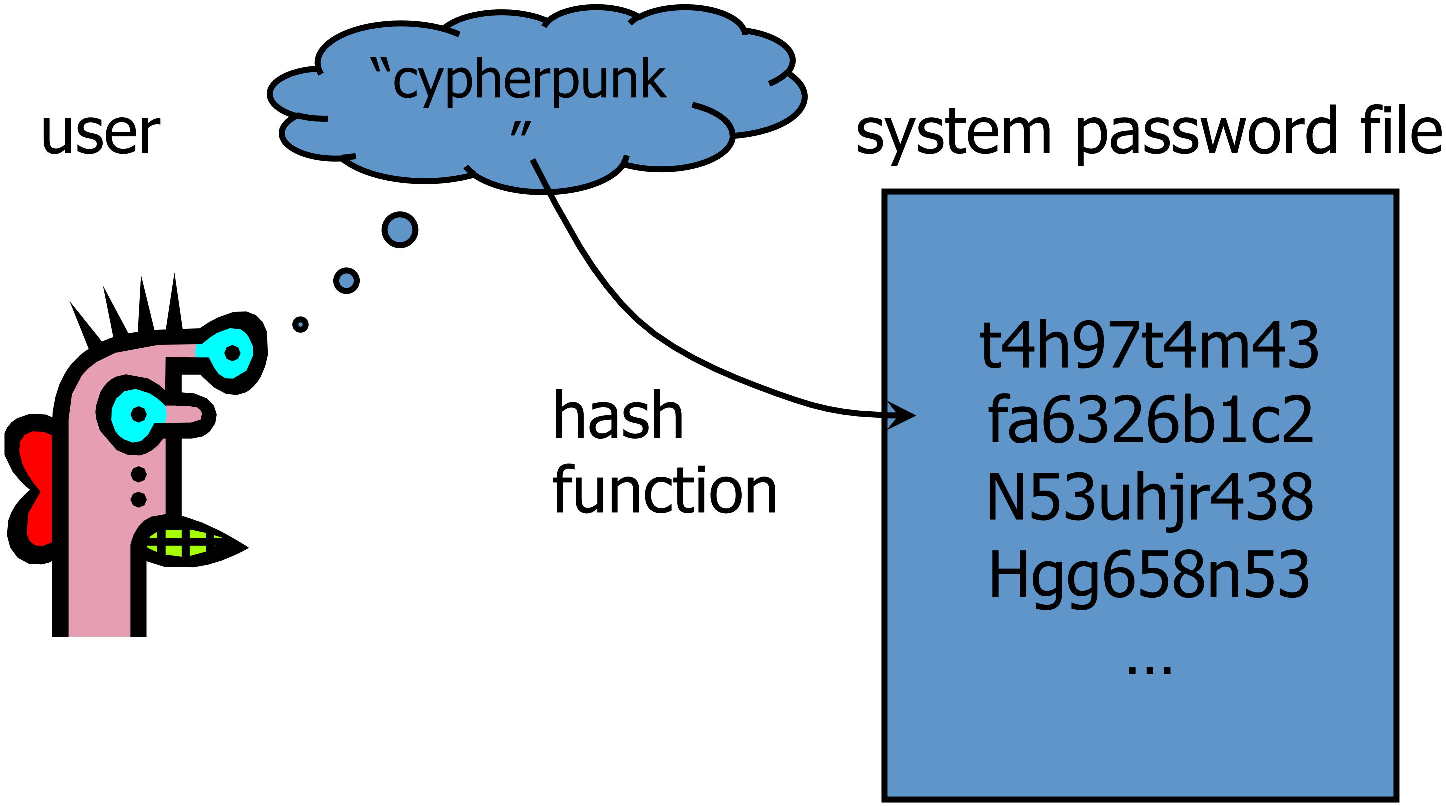
Lecture 3: User Authentication

Common User Authentication Methods

- Password-based Authentication
  - something the individual knows
- Token-based Authentication
  - something the individual has
- Biometric Authentication
  - something the individual is/does

UNIX-Style Passwords



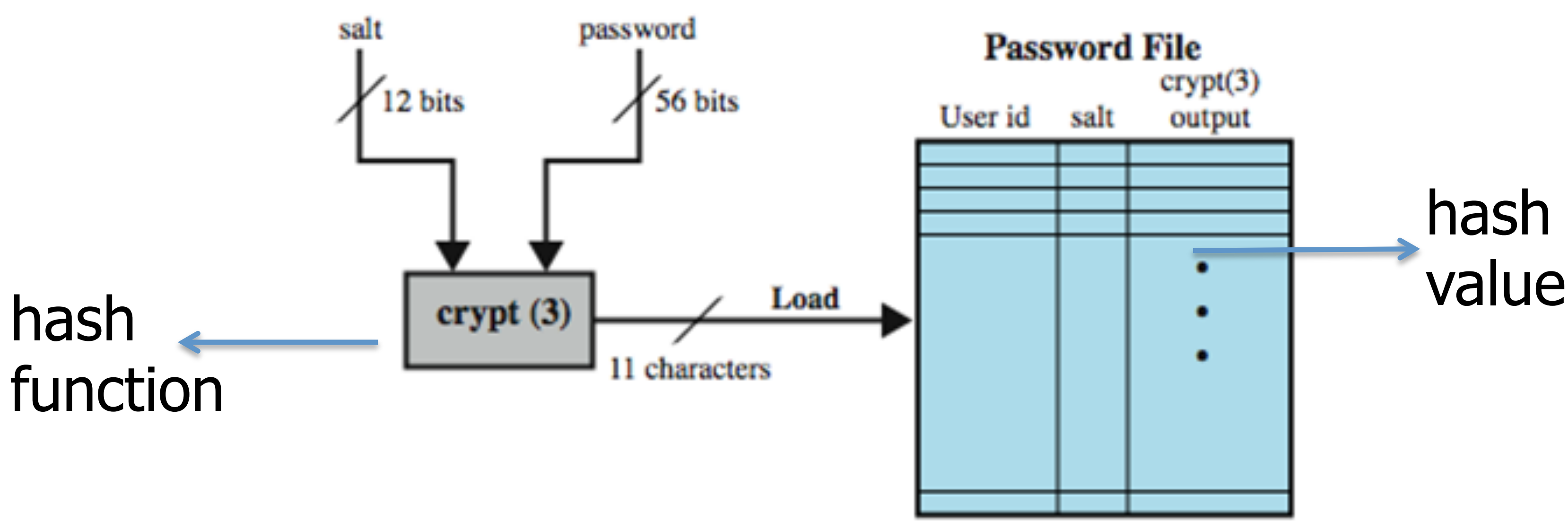
Hashed passwords are originally stored in a publicly accessible file /etc/passwd

Attacks to PWD Authentications

- Offline dictionary attack
  - Dictionary: a set of pwds that are commonly chosen
  - Dictionary attack is possible because many passwords come from a small dictionary
  - Attacker can compute H(password) for every password in the dictionary (**rainbow table**) and see if the result is in the password file
    - **Password file is sometimes available to the attacker**

Countermeasure—add salt

- A password is combined with a fixed-length **salt value**
- The hash value of the salted pwd is calculated and stored in the pwd file

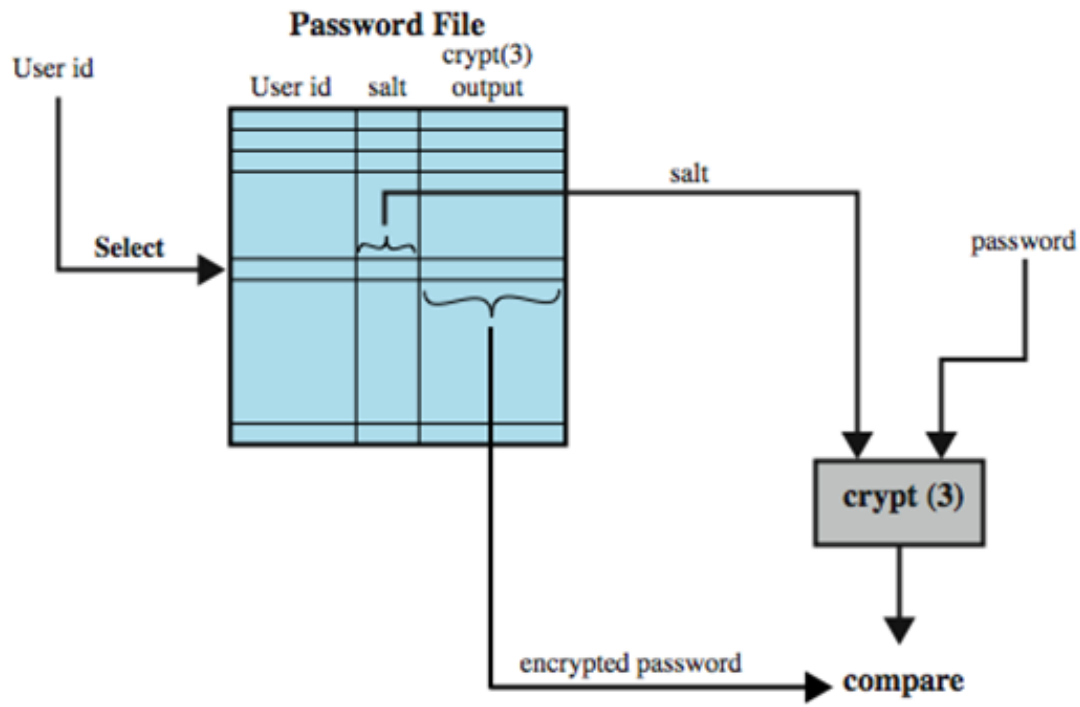


(a) Loading a new password

Why Can Salt Relieve Dictionary Attack?

- Even the attacker knows which user uses which salt (public/available) …

  - For each dictionary password, the attacker has to compute Hash(dictionary password||salt), compare it with every entry in pwd file, multiple users→multiple rainbow tables
  - without salt: the attacker only computes one rainbow table, compare it with every entry in pwd file
  - the efforts differ when the attacker try to compromise multiple pwds



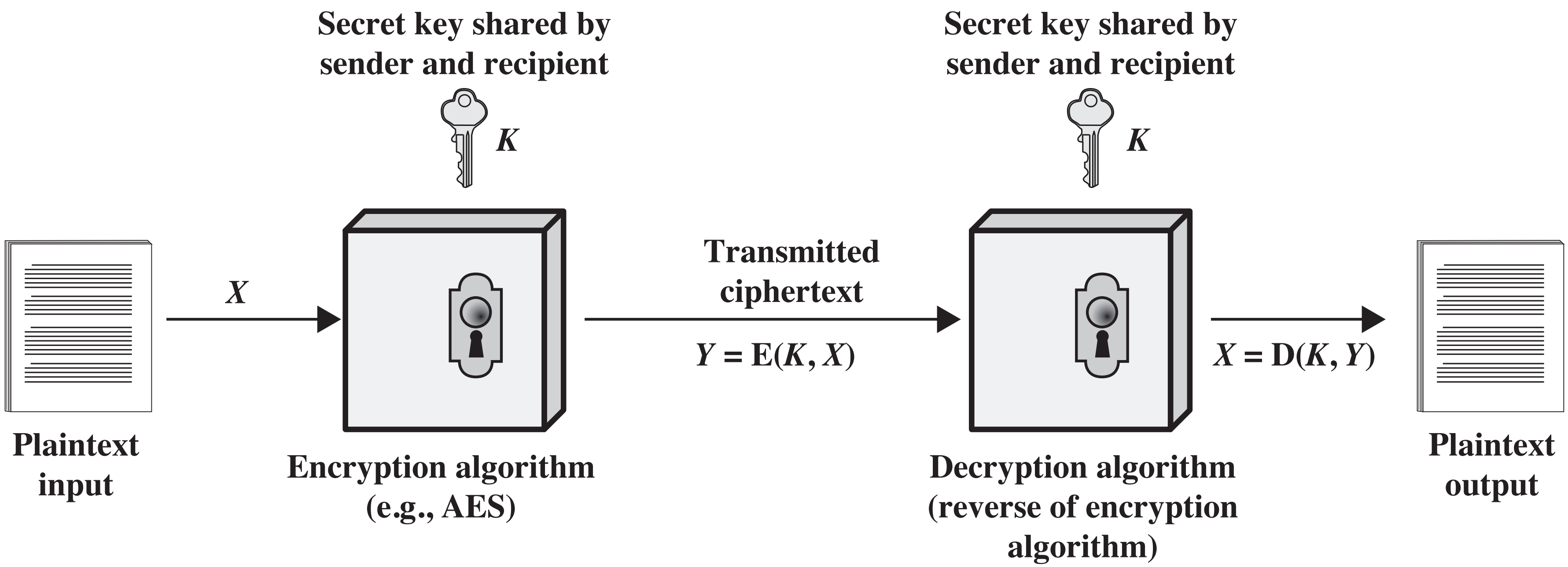
(b) Verifying a password

Questions

- Is it possible to thwart completely all password crackers by dramatically increasing the salt size to, say 24 bits or 48 bits?
  - the purpose of salt is to let each user have a unique salt, such that even though two users choose the same pwds, their hashed salted pwds are different (different rainbow tables—cannot reuse rainbow table)
  - if 12 bits can guarantee uniqueness of rainbow table for each user, there is no need to increase salt size

Lecture 5: Symmetric Encryption Techniques

Model of Symmetric Encryption



Cryptoanalysis and Brute-Force Attack

Cryptanalysis

- Attack relies on the nature of the algorithm plus some knowledge of the general characteristics of the plaintext
- Attack exploits the characteristics of the algorithm to attempt to deduce a specific plaintext or to deduce the key being used

Brute-force attack

- Attacker tries every possible key on a piece of ciphertext until an intelligible translation into plaintext is obtained
- On average, half of all possible keys must be tried to achieve success

Caesar Cipher Algorithm

- Can define transformation as:

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C

- Mathematically give each letter a number

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

- Algorithm can be expressed as:

$$c = \mathrm{E}(3, p) = (p + 3) \bmod (26)$$

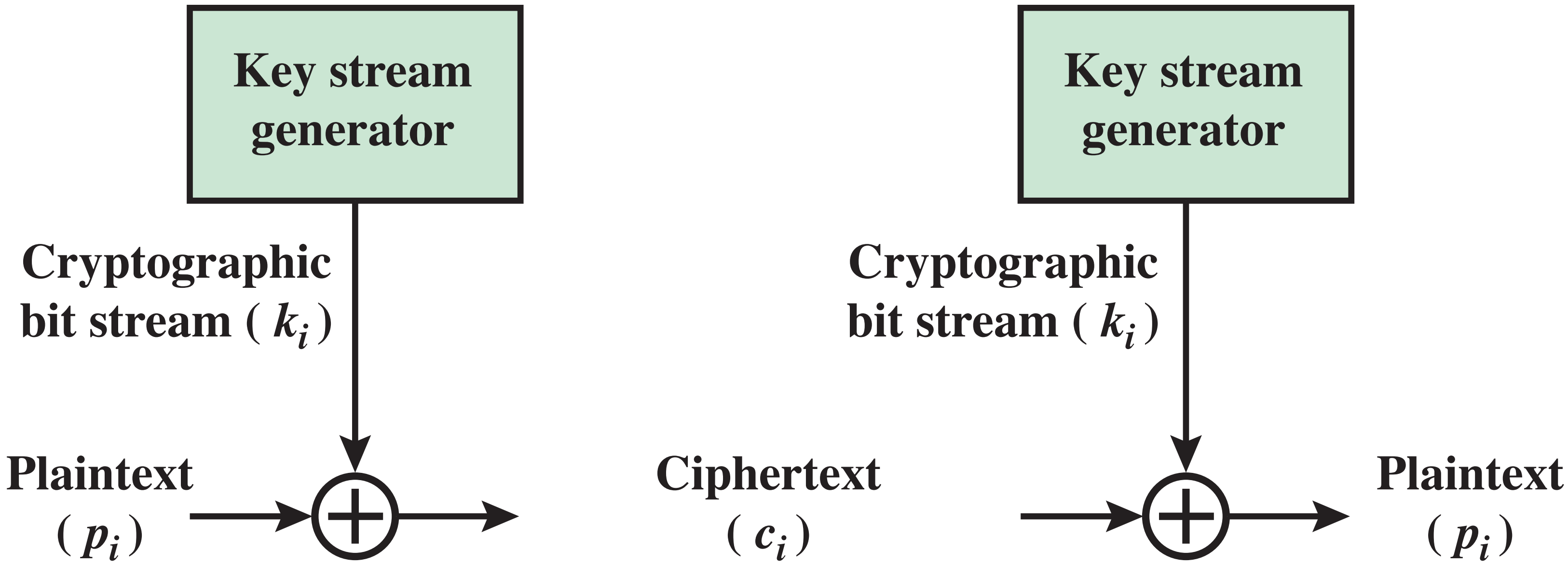
– A shift may be of any amount, so that the general Caesar algorithm is:

$$C = E(k, p) = (p + k) \bmod 26$$

- where k takes on a value in the range 1 to 25; the decryption algorithm is simply:

$$p = D(k, C) = (C - k) \bmod 26$$

Vernam Cipher



Vernam Cipher

```
SENDING
-------
message: 0 0 1 0 1 1 0 1 0 1 1 1 ...
pad:     1 0 0 1 1 1 0 0 1 0 1 1 ...
XOR      ---------------------------
cipher:  1 0 1 1 0 0 0 1 1 1 0 0 ...

RECEIVING
---------
cipher:  1 0 1 1 0 0 0 1 1 1 0 0 ...
pad:     1 0 0 1 1 1 0 0 1 0 1 1 ...
XOR      ---------------------------
message: 0 0 1 0 1 1 0 1 0 1 1 1 ...
```

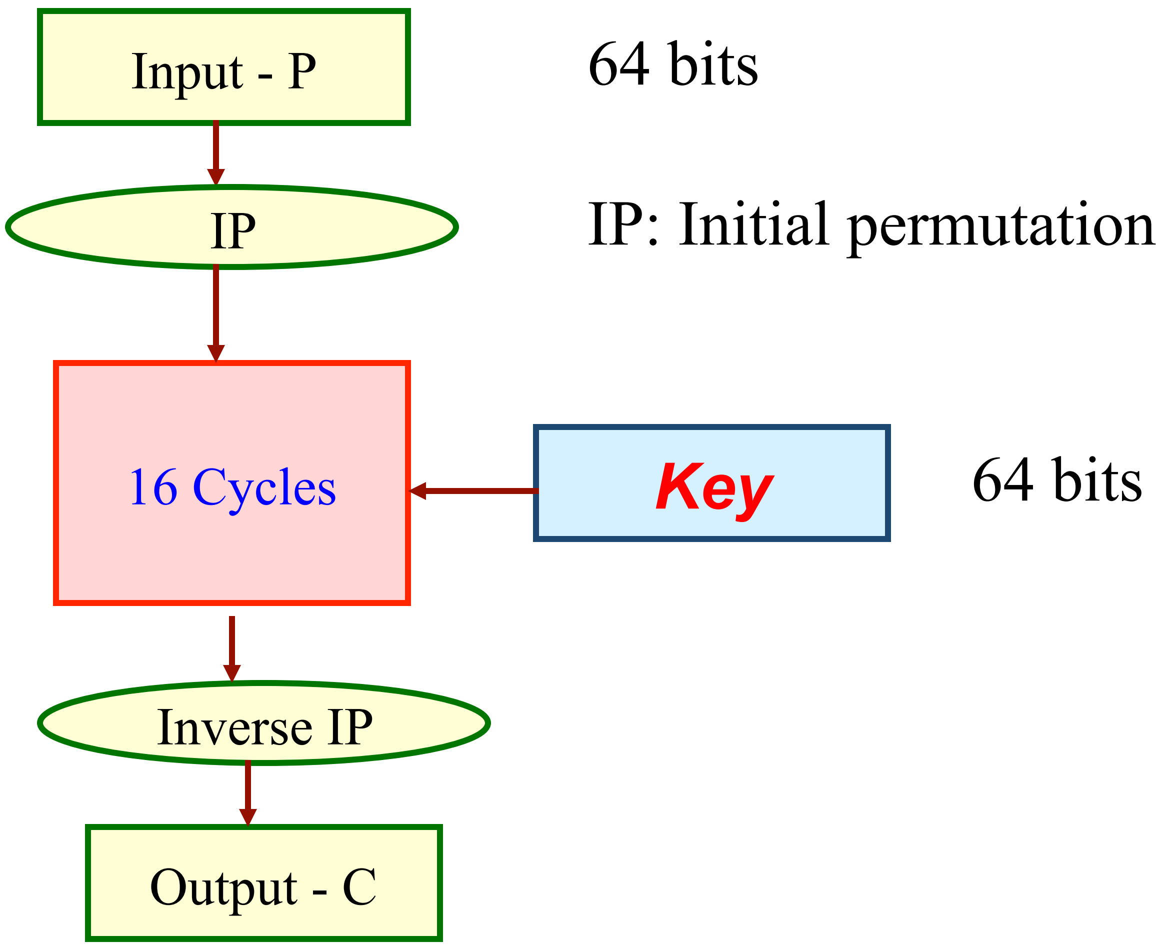
One-Time Pad

- Improvement to Vernam cipher proposed by an Army Signal Corp officer, Joseph Mauborgne
- Use a random key that is as long as the message so that the key need not be repeated
- Key is used to encrypt and decrypt a single message and then is discarded
- Each new message requires a new key of the same length as the new message
- Scheme is unbreakable (perfect security)

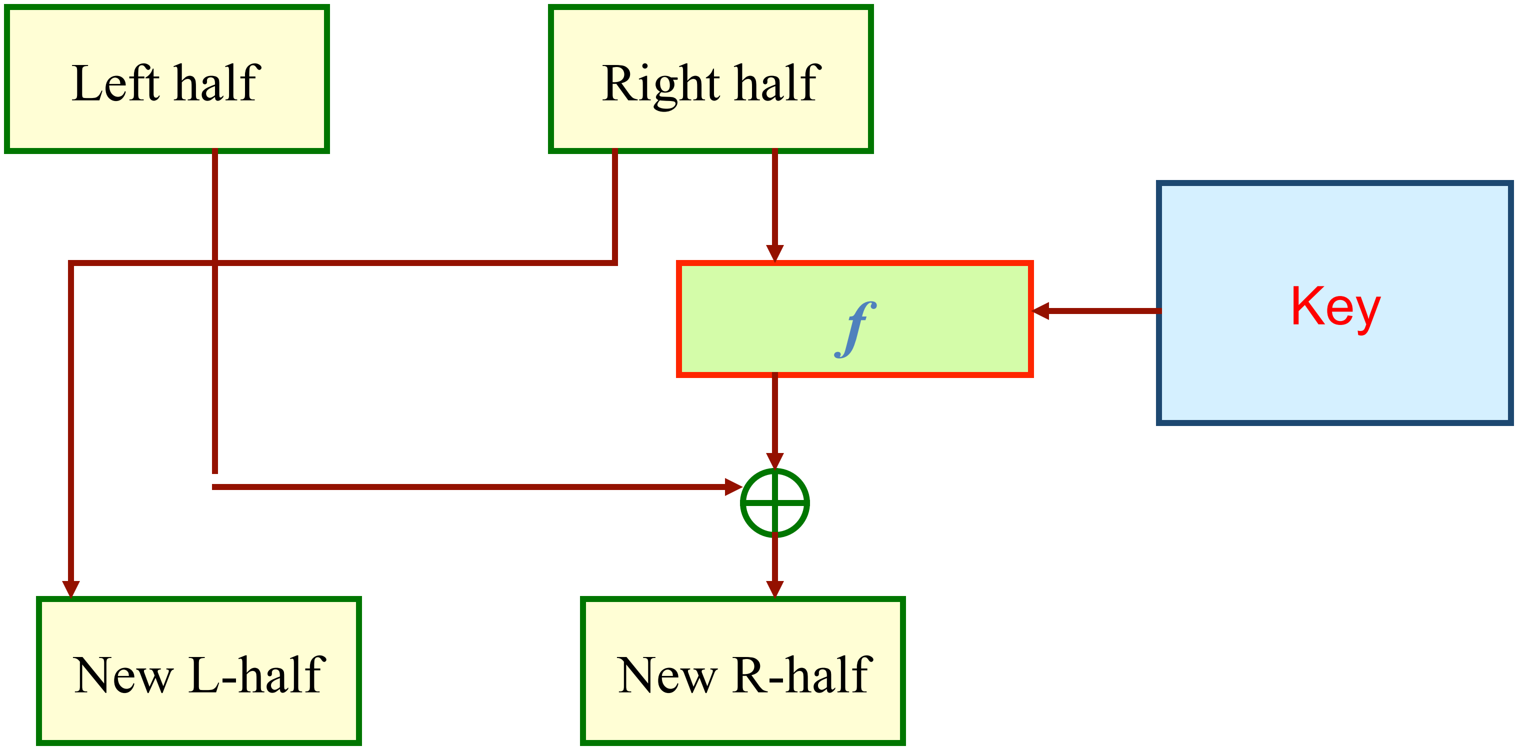
Lecture 6

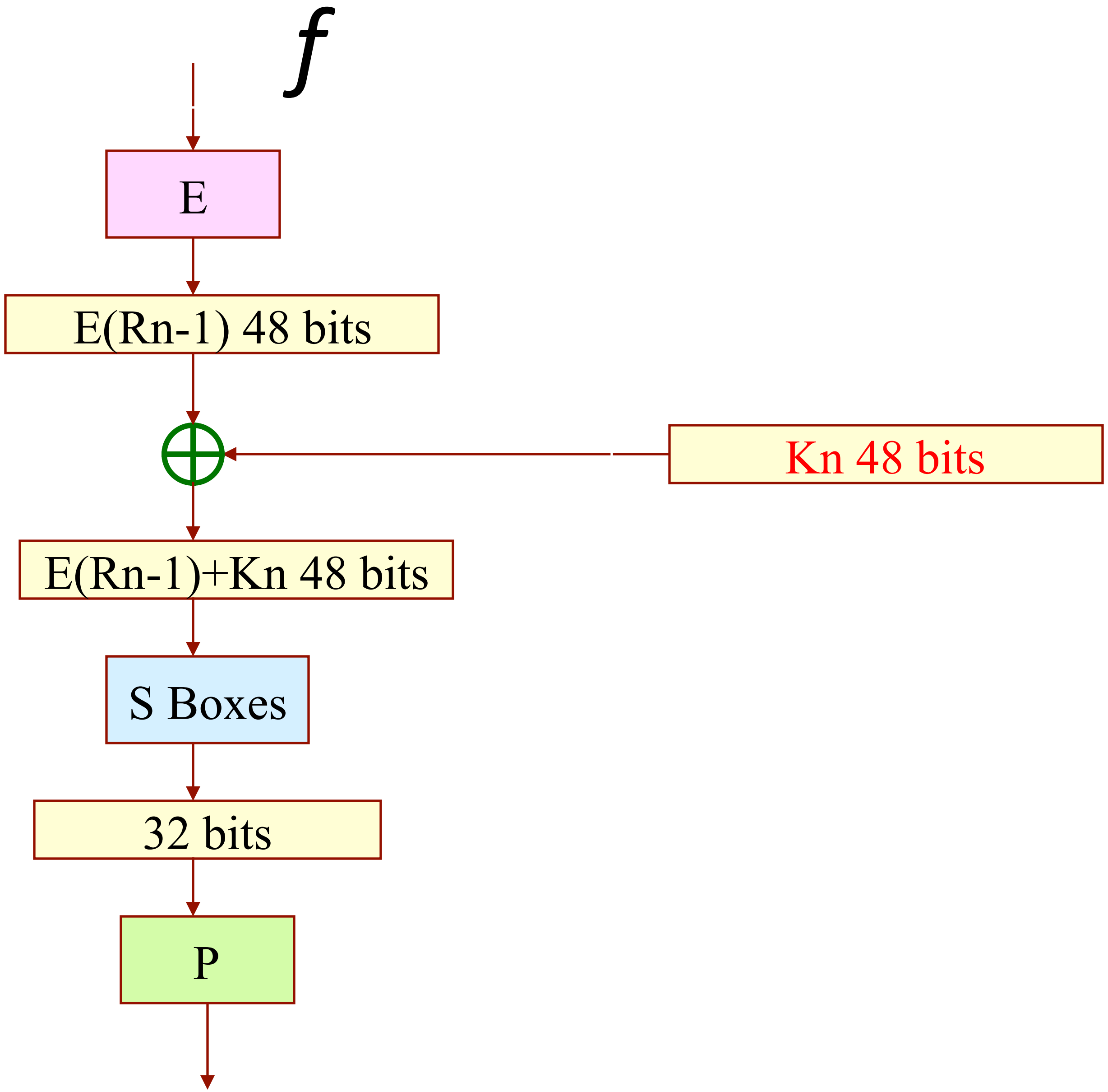
Data Encryption Standard (DES)

A High Level Description of DES



A Cycle in DES





f
E
E(Rn-1) 48 bits
Kn 48 bits
E(Rn-1)+Kn 48 bits
S Boxes
32 bits
P

Expansion Component

- **2.4 Expand each block R_{n-1} from 32 bits to 48 bits using a permuation table that repeats some of the bits in R_{n-1} .**

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

Substitution – S-Boxes

- $K_n + \mathrm{E}(R_{n-1}) = B_1B_2B_3B_4B_5B_6B_7B_8$

where each B_i is a group of six bits.

We now calculate

$$S_1(B_1)S_2(B_2)S_3(B_3)S_4(B_4)S_5(B_5)S_6(B_6)S_7(B_7)S_8(B_8)$$

where $S_i(B_i)$ referrers to the output of the i-th **S** box.

Substitution – S-Boxes (Cont.)

Box S1

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	9
2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

Finding S1(B1)

- The first and last bits of B represent in base 2 a number in the decimal range 0 to 3.
  - Let that number be i.
- The middle 4 bits of B represent in base 2 a number in the decimal range 0 to 15.
  - Let that number be j.
- Look up in the table the number in the i-th row and j-th column.

Lecture 7

Background Knowledge of DES

CS 450/650

Fundamentals of

Integrated Computer Security

Background Knowledge of DES

- Proposed the use of a cipher that alternates substitutions and permutations

Substitutions

- Each plaintext element or group of elements is uniquely replaced by a corresponding ciphertext element or group of elements

Permutation

- No elements are added or deleted or replaced in the sequence, rather the order in which the elements appear in the sequence is changed

Avalanche Effect

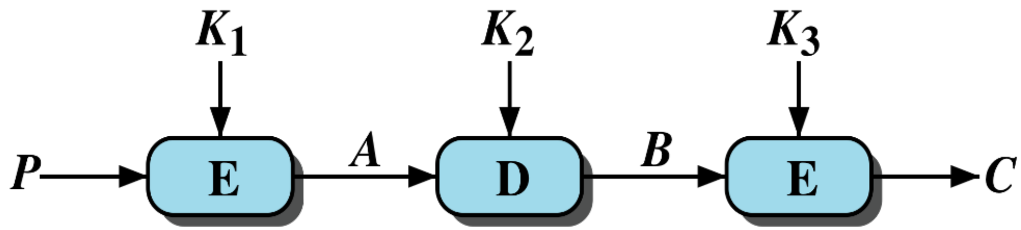
- Avalanche effect means a small change in the plaintext (or key) should create a significant change in the ciphertext.
- Avalanche effect is the prime design criteria for any block cipher—why?
  - If the change of one bit from the input leads to the change of only one bit of the output, then it is easy to guess to find the input
  - E(1011)=1110; E(1001)=?

Cracking DES

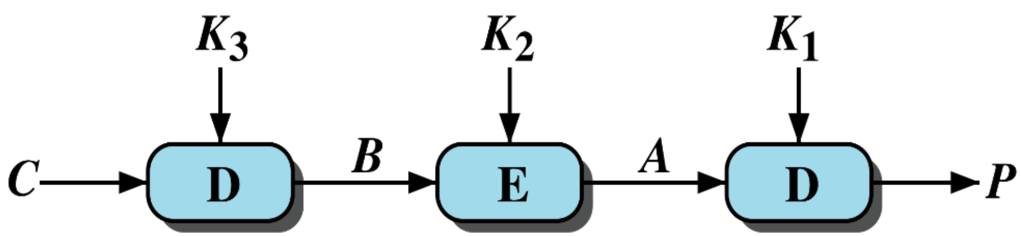
- Diffie and Hellman then outlined a "brute force" attack on DES
  - By "brute force" is meant that you try as many of the $\mathbf{2^{56}}$ (why?) possible keys to decrypt the ciphertext into a meaningful plaintext message
- cryptoanalysis—no good solution due to AE

Triple DES

- Triple-DES uses three keys and three executions of DES algorithm



(a) Encryption



(b) Decryption

Triple DES

- Keying options
  - Option 1: all three keys (K1, K2, K3) are independent: the strongest, with 3∗56=168 independent key bits
  - Option 2: K1 and K2 are independent, and K3=K1: provides less security with 2∗56=112 key bits, but stronger than pure DES
  - Option 3: all three keys are identical—equivalent of DES (why?)

Lecture 8
AES

CS 450/650

Fundamentals of
Integrated Computer Security

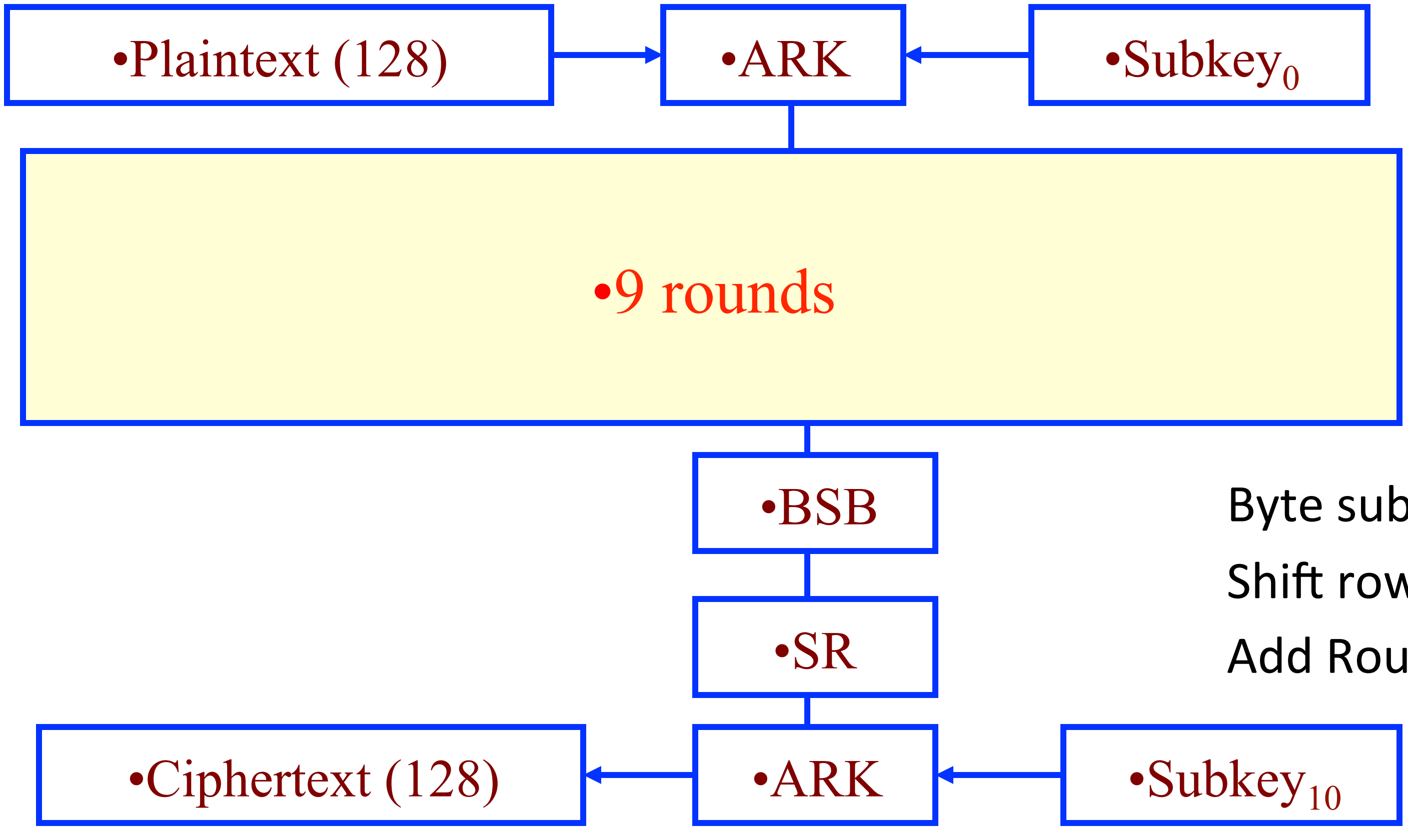
AES

- 10, 12, 14 rounds for 128, 192, 256 bit keys
  - Regular Rounds (9, 11, 13)
  - Final Round is different (10^{th}, 12^{th}, 14^{th})

- Each regular round consists of 4 steps
  - Byte substitution (BSB)
  - Shift row (SR)
  - Mix column (MC)
  - Add Round key (ARK)

AES Overview

128-bit AES



Byte substitution (BSB)

Shift row (SR)

Add Round key (ARK)

Four Operations

1. *Byte Substitution*
   – predefined substitution table s[i,j] → s' [i,j]
2. *Shift Row*
   – left circular shift
3. *Mix Columns*
   – 4 elements in each column are multiplied by a polynomial
4. *Add Round Key*
   – Key is derived and added to each column

Four Operations

1. *Byte Substitution*
   – predefined substitution table s[i,j] → s' [i,j]
2. *Shift Row*
   – left circular shift
3. *Mix Columns*
   – 4 elements in each column are multiplied by a polynomial
4. *Add Round Key*
   – Key is derived and added to each column

Substitution table

	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	BE	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	84	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

Exercise

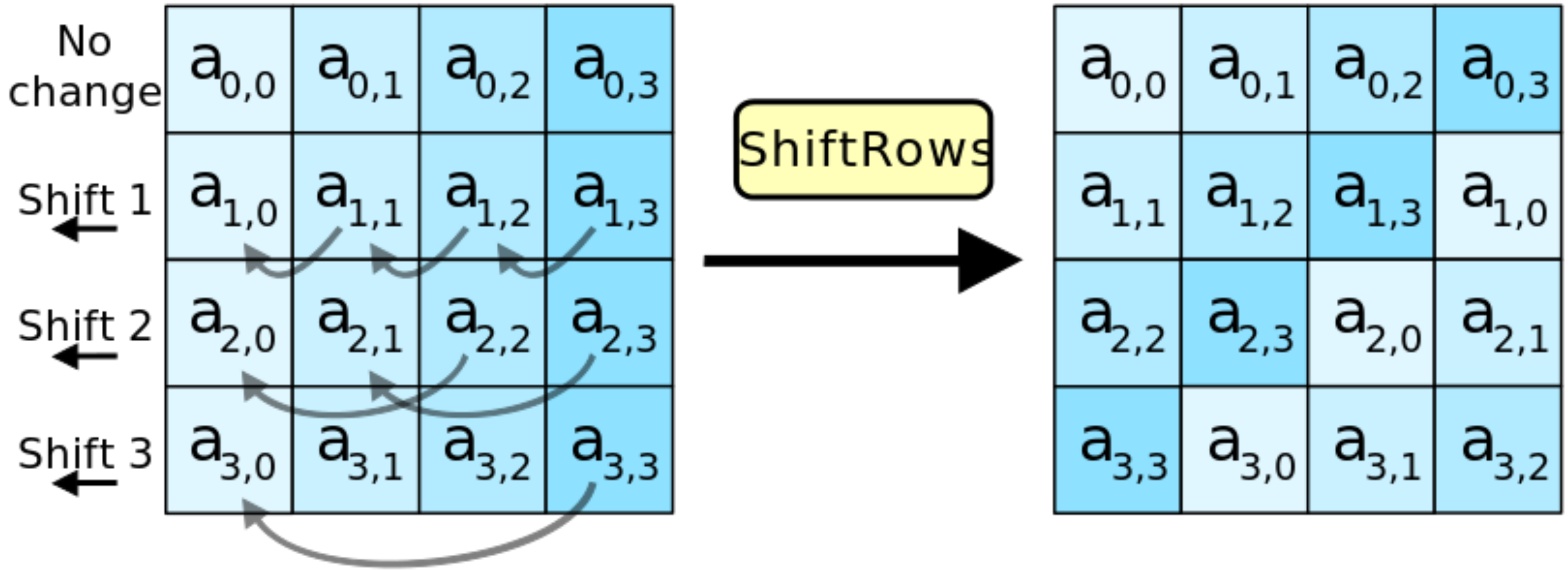
- Using the table, find the substitution of

6b, ff, 6e, 09

Four Operations

1. *Byte Substitution*
   – predefined substitution table s[i,j] → s' [i,j]
2. *Shift Row*
   – left circular shift
3. *Mix Columns*
   – 4 elements in each column are multiplied by a polynomial
4. *Add Round Key*
   – Key is derived and added to each column

Shift Row (128-bit)



Four Operations

1. *Byte Substitution*
   – predefined substitution table s[i,j] → s' [i,j]
2. *Shift Row*
   – left circular shift
3. *Mix Columns*
   – 4 elements in each column are multiplied by a polynomial
4. *Add Round Key*
   – Key is derived and added to each column

Mix Column

$$\begin{bmatrix} S'_{0,i} \\ S'_{1,i} \\ S'_{2,i} \\ S'_{3,i} \end{bmatrix} = \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} * \begin{bmatrix} S_{0,i} \\ S_{1,i} \\ S_{2,i} \\ S_{3,i} \end{bmatrix}$$

$i=0\ldots3$

- Multiplying by 1 → no change
- Multiplying by 2 → shift left one bit
- Multiplying by 3 → shift left one bit and XOR with original value

Exercise

$$\begin{bmatrix} S'_{0,I} \\ S'_{1,I} \\ S'_{2,I} \\ S'_{3,i} \end{bmatrix} = \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} * \begin{bmatrix} e5 \\ a8 \\ 6f \\ 33 \end{bmatrix}$$

Four Operations

1. *Byte Substitution*
   – predefined substitution table s[i,j] → s' [i,j]
2. *Shift Row*
   – left circular shift
3. *Mix Columns*
   – 4 elements in each column are multiplied by a polynomial
4. *Add Round Key*
   – Enc key is derived and added to each column

Add Key

b0	b4	b8	b12
b1	b5	b9	b13
b2	b6	b10	b14
b3	b7	b11	b15

k0	k4	k8	k12
k1	k5	k9	k13
k2	k6	k10	k14
k3	k7	k11	k15

$$b'_x = b_x \text{ XOR } k_x$$

Example

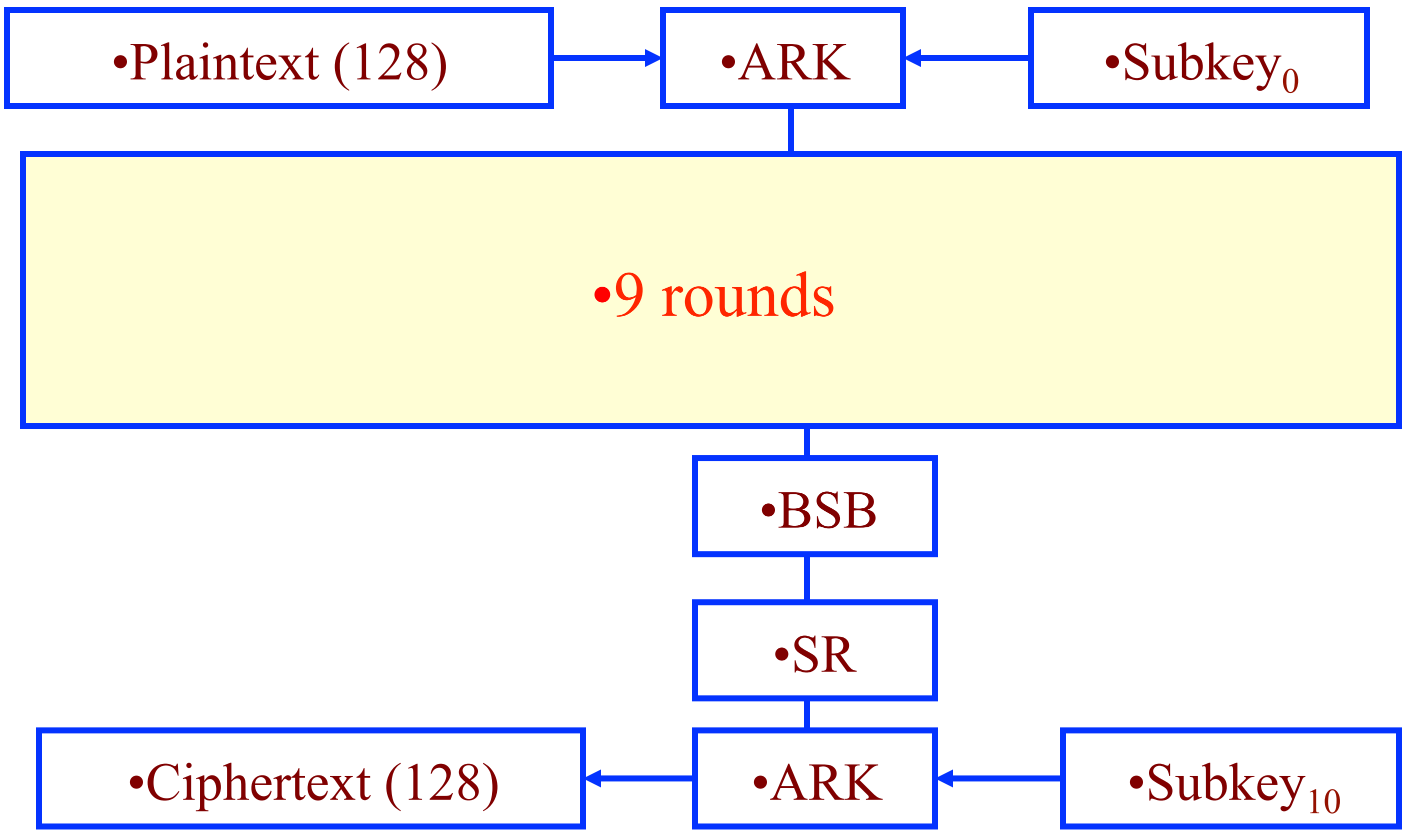
k = 1f 34 0c da 5a 29 bb 71 6e a3 90 f1 47 d6 8b 12

B = e5 a8 6f 33 0a 52 31 9c c2 75 f8 1e b0 46 de 3a

B’= fa 9c 63 9e 50 7b 8a ed ac d6 68 ef f7 90 55 28

AES Overview

128-bit AES



Lecture 9

Arithmetic Fundamentals for RSA

CS 450/650

Fundamentals of

Integrated Computer Security

Modular Arithmetic

- If *a* is an integer and *n* is a positive integer, we define *a* mod *n* to be the remainder when *a* is divided by *n*; the integer *n* is called the modulus
- Thus, for any integer *a*
  - $a = qn + r, \qquad 0 <= r < n; q = [a/n]$
  - $a \bmod n = r$
- Example: 1) 11 mod 7 and 2) −11 mod 7

Modular Arithmetic

- $[(a \bmod n) + (b \bmod n)] \bmod n = (a + b) \bmod n$
  - $[(11 \bmod 10) + (12 \bmod 10)] \bmod 10 = (11 + 12) \bmod 10$
- $[(a \bmod n) - (b \bmod n)] \bmod n = (a - b) \bmod n$
  - $[(11 \bmod 10) - (12 \bmod 10)] \bmod 10 = (11 - 12) \bmod 10$
- $[(a \bmod n) \times (b \bmod n)] \bmod n = (a \times b) \bmod n$
  - $[(11 \bmod 10) \times (12 \bmod 10)] \bmod 10 = (11 \times 12) \bmod 10$

Modular Arithmetic

- for integer $n > 1$, if $a \equiv b \bmod n$ and $c \equiv d \bmod n$, then $a \pm c \equiv b \pm d \bmod n$ and $ac \equiv bd \bmod n$
- for integer $n > 1$ and d != 0, if $ad \equiv bd \bmod n$ then $a \equiv b \bmod [n/gcd(d,n)]$

Lecture 10

Rivest-Shamir-Adelman (RSA)

CS 450/650

Fundamentals of

Integrated Computer Security

RSA

- To encrypt message M compute
  - $c = m^e \bmod n$

- To decrypt ciphertext c compute
  - $m = c^d \bmod n$

Key Choice

- Let p and q be two large prime numbers
- Let n = pq
- Compute $\phi(n) = \phi(p)\phi(q) = (p - 1)(q - 1)$, where ϕ is Euler's totient function. This value is kept private.
  - Euler's totient function of n: counts the positive integers less than or equal to *n* that are relatively prime to *n*
  - Euler's totient function is a multiplicative function, meaning that if two numbers *m* and *n* are coprime, then $\phi(mn) = \phi(m)\,\phi(n)$

Key Choice

- Choose an integer e such that $1 < e < \phi(n)$ and $\gcd(e, \phi(n)) = 1$; i.e., e and $\phi(n)$ are coprime.
  - e is released as the encryption key.
- Determine d as $d \equiv e^{-1} \pmod{\phi(n)}$; or solve for d given $d \cdot e \equiv 1 \pmod{\phi(n)}$
  - d is the decryption key
- (e, d) is the RSA key pair

Example

- Select primes p=11, q=3
- $n = p * q = 11*3 = 33$
- Compute $\phi(33) = \phi(11)\phi(3) = (11-1)*(3-1)=20$
- Choose $e = 3$

Example (cont.)

- Compute d such that

$$e * d \mod \phi(n) = 1$$

$$3 * d \mod 20 = 1$$

$$d = 7$$

Public key = (n, e) = (33, 3)

Private key = (d) = (7)

Example (cont.)

- Now say we want to encrypt message m = 5
- $c = m^e \bmod n = 5^3 \bmod 33 = 125 \bmod 33 = 26$
  - Hence the ciphertext c = 26
- To check decryption, we compute

$$m = c^d \bmod n = 26^7 \bmod 33 = 5$$

Lecture 10

Digital Signatures

CS 450/650

Fundamentals of

Integrated Computer Security

Digital Signatures

- A digital signature can be interpreted as indicating the signer's agreement with the contents of an electronic document
  - Similar to handwritten signatures on physical documents, but in digital format

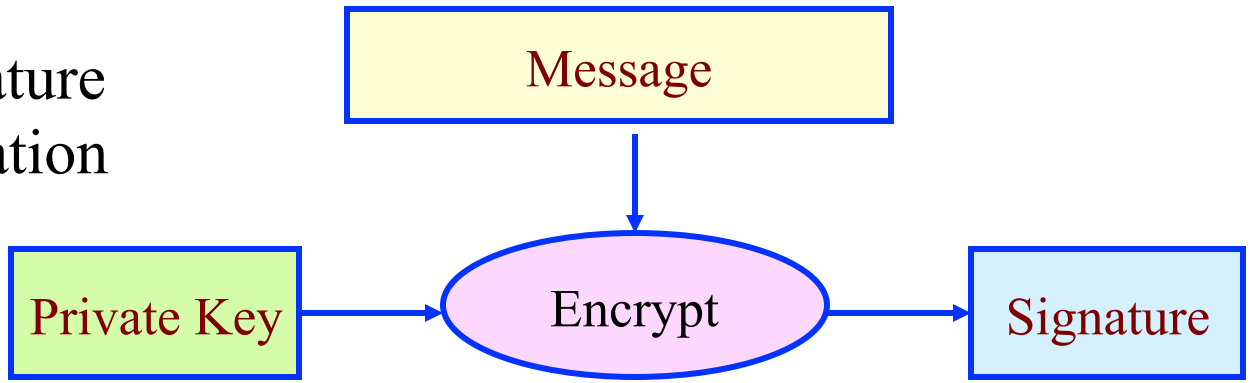
Digital Signature Using RSA

- The RSA public-key cryptosystem can be used to create a digital signature for a message m
  - Asymmetric Cryptographic techniques are well suited for creating digital signatures

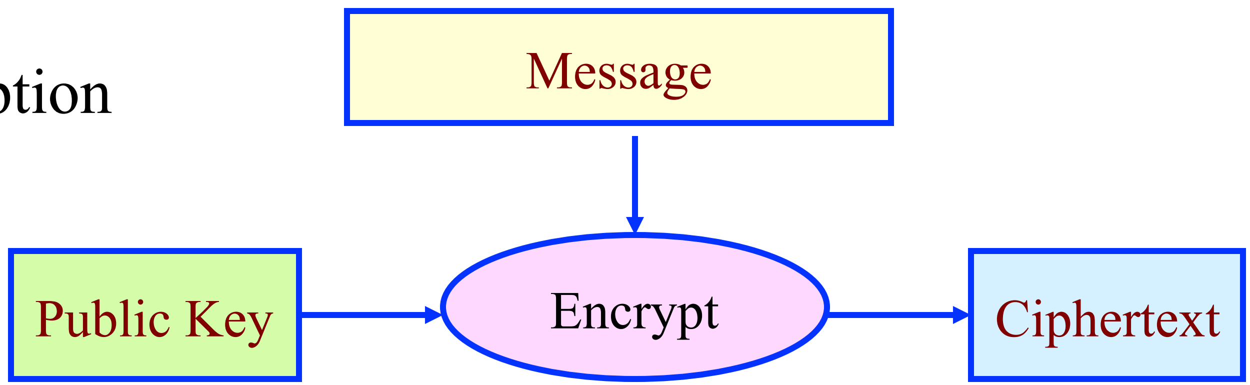
- RSA cryptosystem
  - $c = M^e \bmod n$
  - $M = c^d \bmod n$

Signature Generation (Signer)

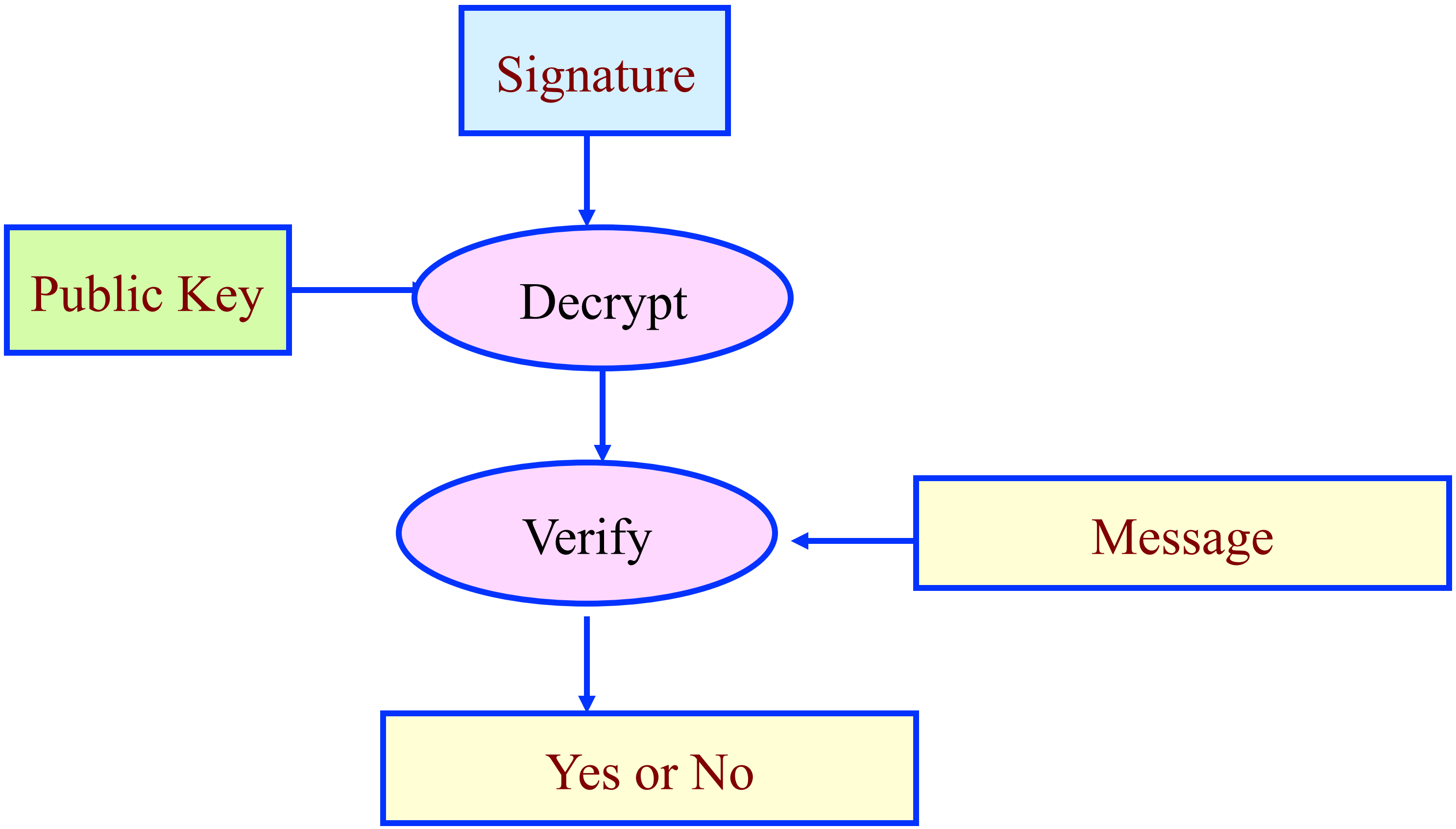
- Signature generation



- Data encryption



Signature Verification



Lecture 12
Cryptographic Hash Functions

CS 450/650

Fundamentals of

Integrated Computer Security

Basic Knowledge

- Hash functions are important cryptographic primitive and are widely used in security protocols
- Compute digest of a message which is a short, fixed-length bit-string
  - Finger print of a message, i.e., unique representation of a message
- Does not have key

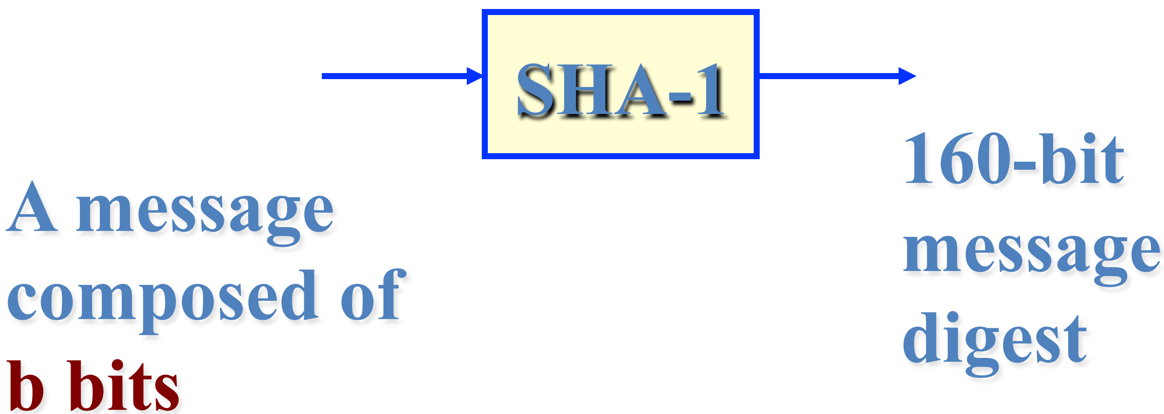
Security Requirements of Hash Functions

- One-wayness
  - Given M, it is easy to compute h
  - Given any h, it is hard to find any M, such that H(M) = h
- Collision-resistant
  - Given M1, it is **difficult** to find M2, such that H(M1) = H(M2)

Lecture 13 Secure Hash Algorithm (SHA)

Secure Hash Algorithm (SHA)

- Input: 0-2^{64} bits
  - 2^{30} bits~ 1G bits
- Output: 160 bits, contant



Preprocess-- Padding

- *Padding* → the total length of a padded message is multiple of 512

Padding (cont.)

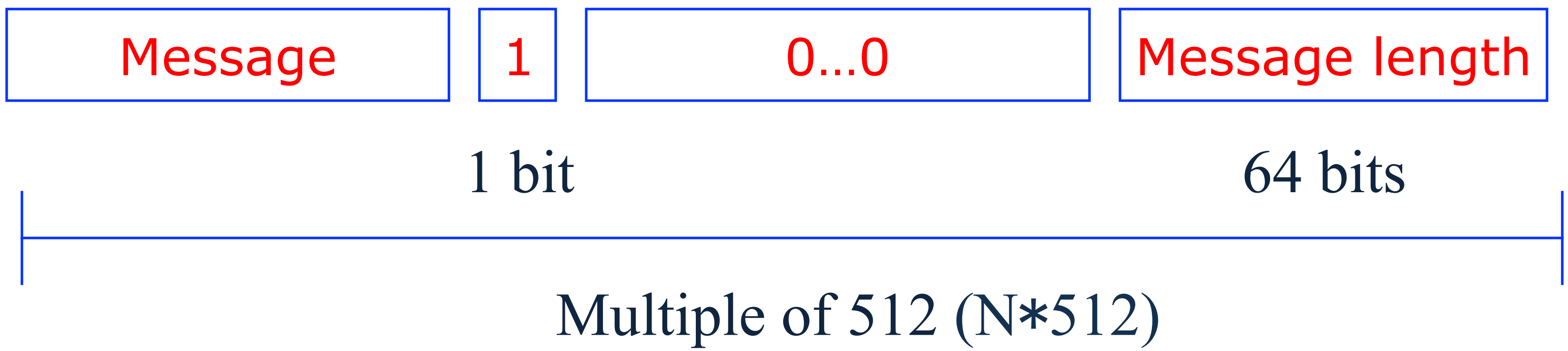
Message	1	0…0	Message length
	1 bit		64 bits

Multiple of 512 (N*512)

- Padding is done by appending to the input
  - A single bit, 1
  - Enough additional bits, all 0, to make **the final block** exactly 512 bits long
  - A 64-bit integer representing the length of the original message in bits

Example

- M = 01100010 11001010 1001 (20 bits)



- How many 0's?
- Representation of "Message length"?

Example

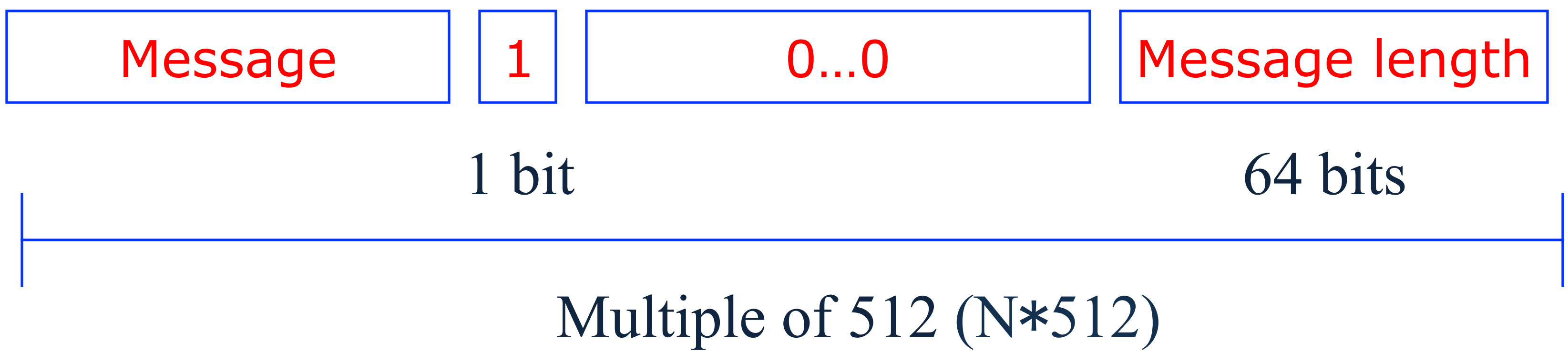
- M = 01100010 11001010 1001 (20 bits)
- Padding is done by appending to the input
  - A single bit, 1
  - 427 0s=512-1-64-20
  - A 64-bit integer representing 20

- Pad(M) = 01100010 11001010 10011000 … 00010100
- Length of Pad(M): 512 bits (N=1)

Example 2

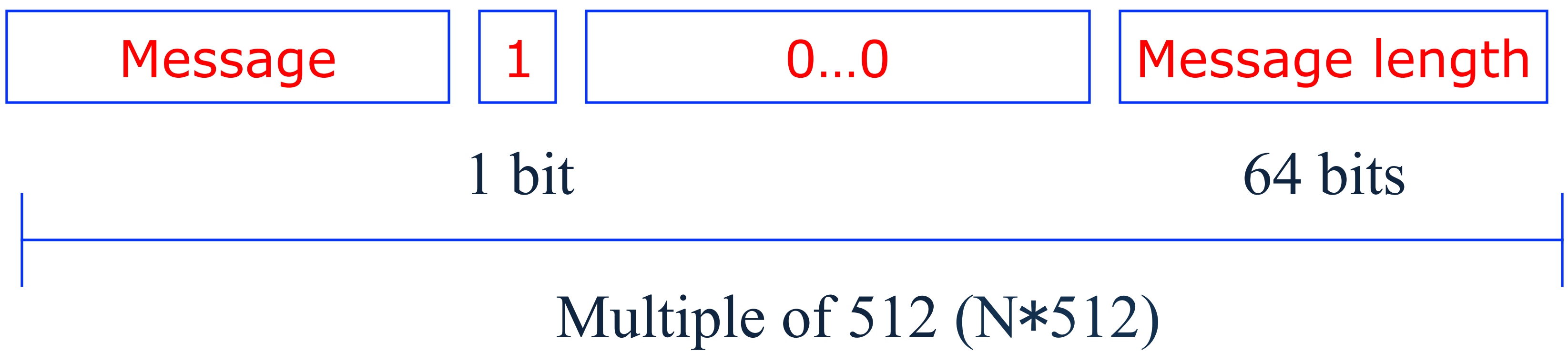
- Length of M = 500 bits

- How many blocks? (N=?)



Example 2

- Length of M = 500 bits→N=2
- How many 0’s?
- “Message length”?



Example 2

- Length of M = 500 bits

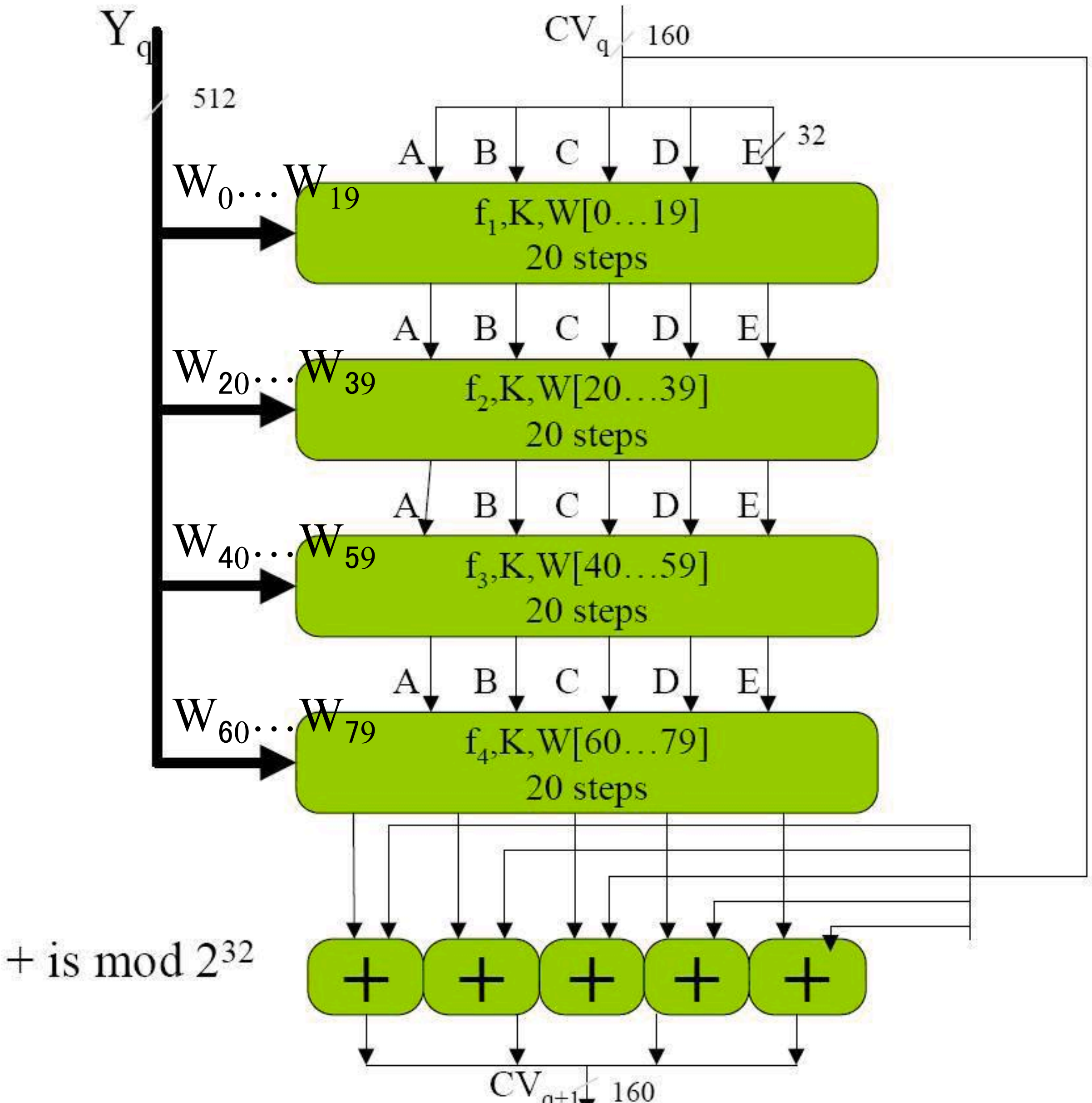
- Padding is done by appending to the input:
  - A single bit, 1
  - 459 0s=1024-500-1-64
  - A 64-bit integer representing 500

- Length of Pad(M) = 1024 bits

Step 1 -- Dividing Pad(M)

- Pad (M) = $B_1, B_2, B_3, \ldots, B_n$
- Each B_i denote a 512-bit block
- Each B_i is divided into 16 32-bit words
  - $W_0, W_1, \ldots, W_{15}$

SHA-1 operation over one block



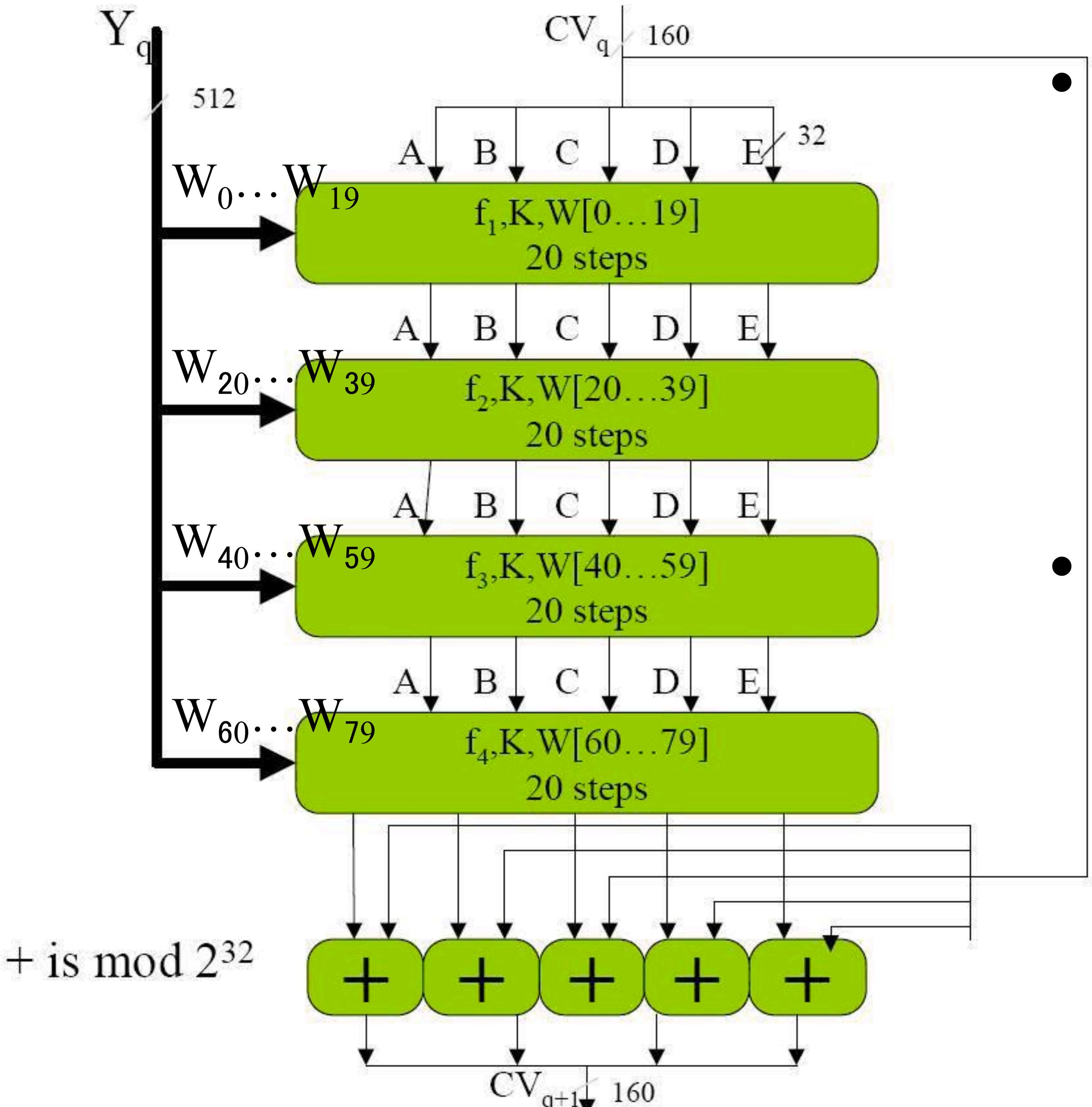
Step 2 – Compute W_{16} – W_{79}

- To Compute word W_j ($16 <= j <= 79$)

  – $W_j = (W_{j-3} \text{ XOR } W_{j-8} \text{ XOR } W_{j-14} \text{ XOR } W_{j-16}) <<< 1$

    - W_{j-3}, W_{j-8}, W_{j-14} , W_{j-16} are XORed
    - The result is circularly left shifted one bit

SHA-1 operation over one block

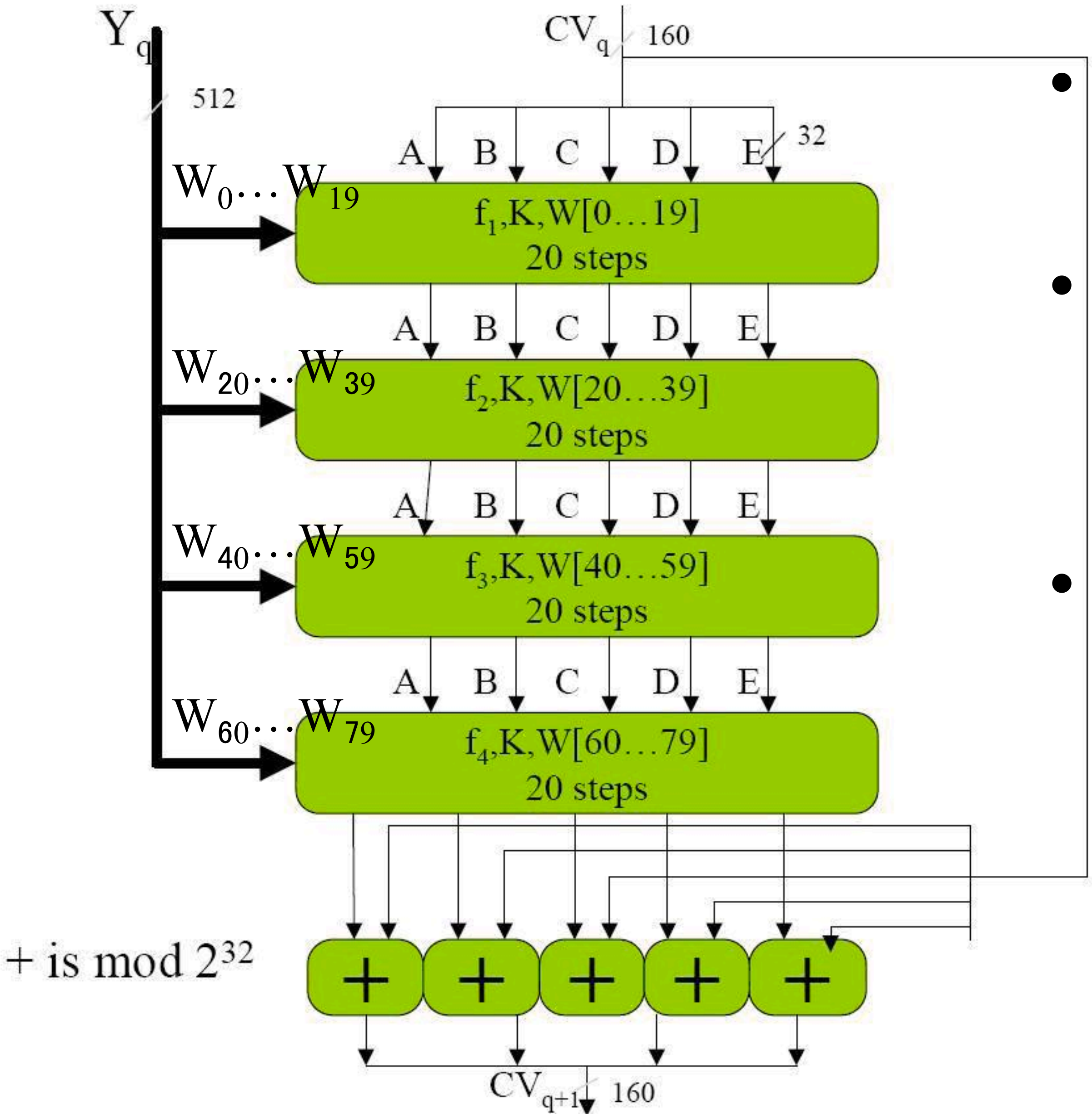


- The output of last block operation (CV_q) is the input of this block operation (q+1)
- How to obtain CV_0(A--E)

Step 3 Initialization

- **A** = CV_0 (0)= 67452301
- **B** = CV_0 (1) = EFCDAB89
- **C** = CV_0 (2) = 98BADCFE
- **D** = CV_0 (3) = 10325476
- **E** = CV_0 (4) = C3D2E1F0

SHA-1 operation over one block

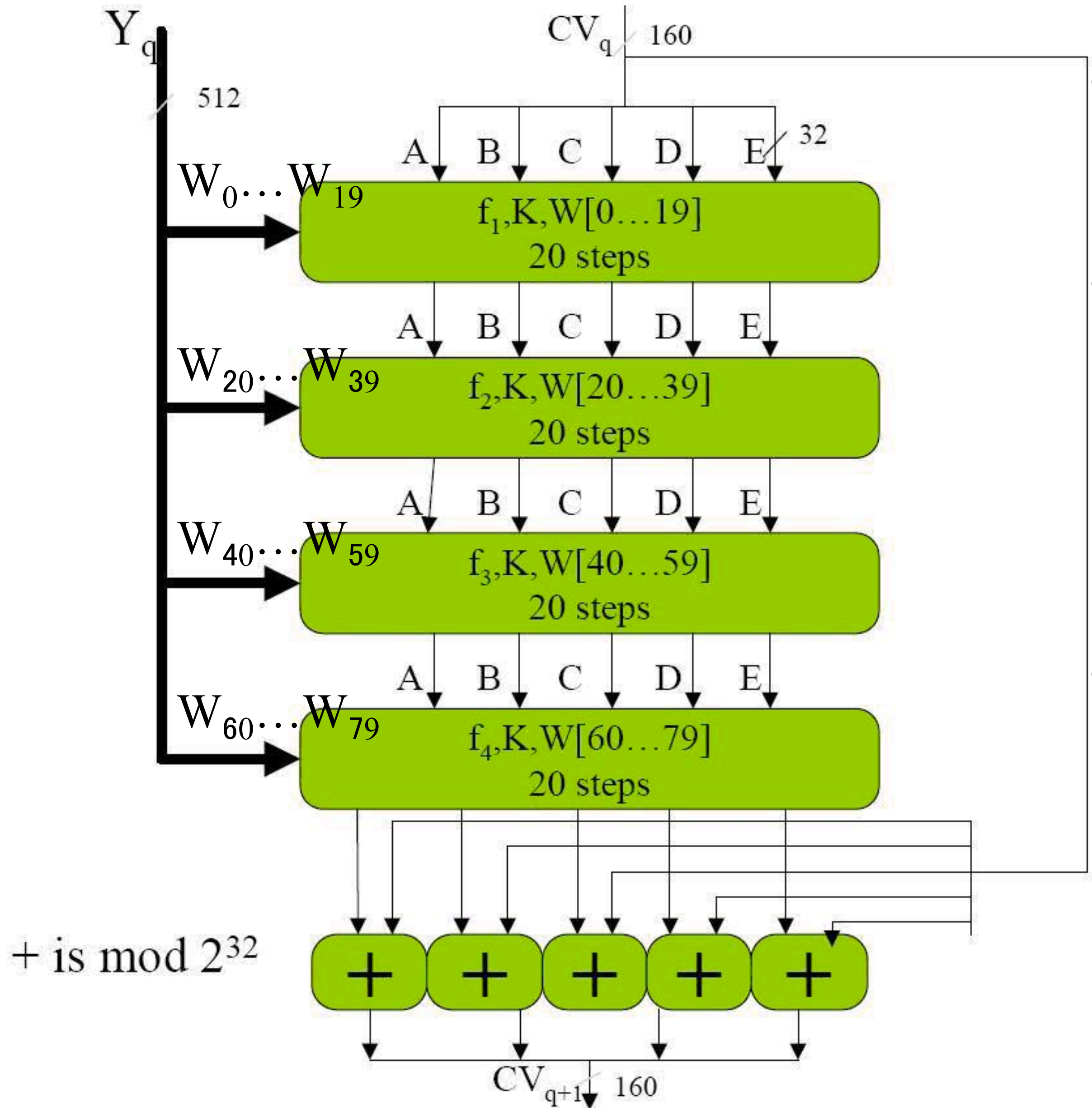


- 4 stage, each with 20 steps
- In each stage t, there is a stage-dependent K_t
- K_t's are constant values

Step 3 Initialization

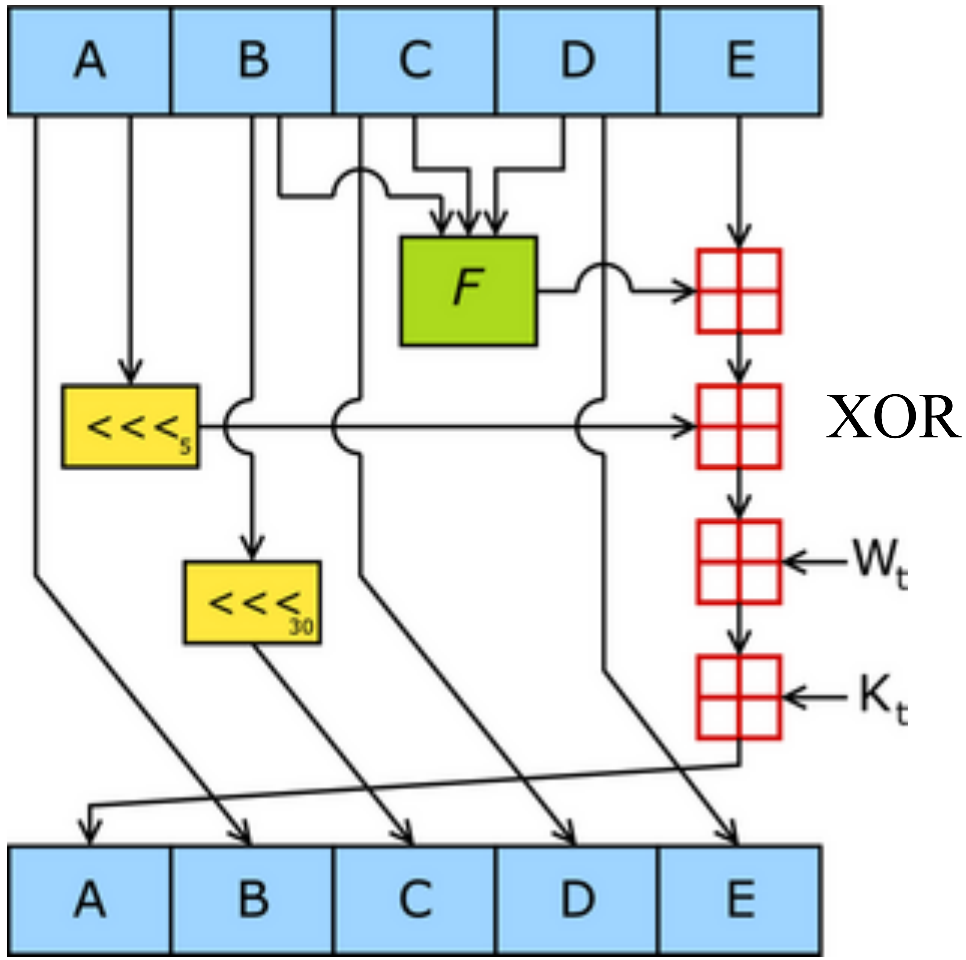
- K_0= 5A827999
- K_1= 6ED9EBA1
- K_2= 8F1BBCDC
- K_3= CA62C1D6

SHA-1 operation over one block



- Input for step i: W_i, (ABCDE)
- $f_{t:}$ some internal function, different for each stage
- A-E: output from last step

Details of One Step (step 4 loop)

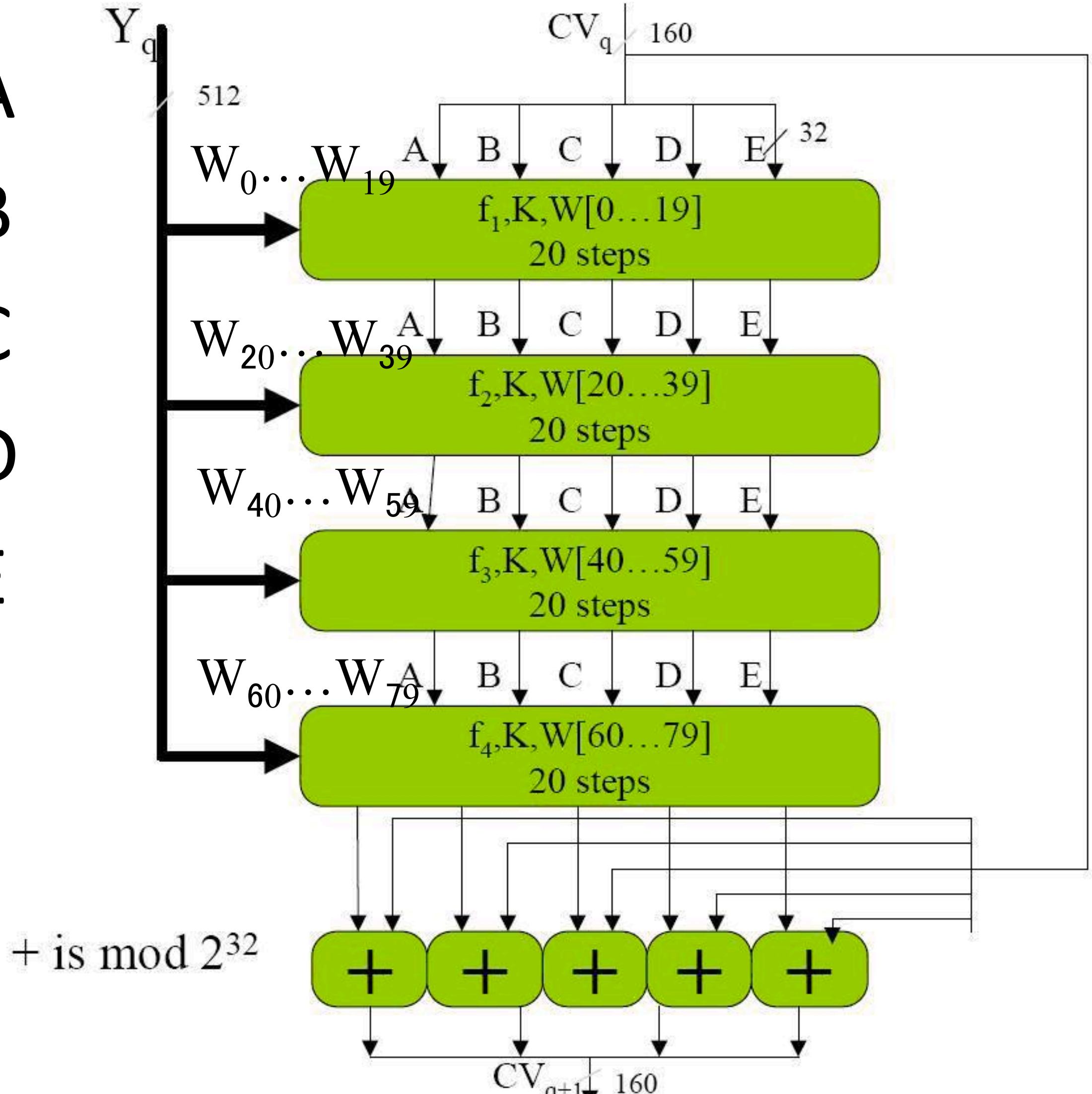


Four functions

- For $j = 0 \dots 19$
  - $f_j(B,C,D) = (B \text{ AND } C) \text{ OR } (B \text{ AND } D) \text{ OR } (C \text{ AND } D)$
- For $j = 20 \dots 39$
  - $f_j(B,C,D) = (B \text{ XOR } C \text{ XOR } D)$
- For $j = 40 \dots 59$
  - $f_j(B,C,D) = (B \text{ AND } C) \text{ OR } ((\text{NOT } B) \text{ AND } D)$
- For $j = 60 \dots 79$
  - $f_j(B,C,D) = (B \text{ XOR } C \text{ XOR } D)$

Step 5 – Final

- $CV_{q+1}(0) = CV_q(0) + A$
- $CV_{q+1}(1) = CV_q(1) + B$
- $CV_{q+1}(2) = CV_q(2) + C$
- $CV_{q+1}(3) = CV_q(3) + D$
- $CV_{q+1}(4) = CV_q(4) + E$



Done

- Once these steps have been performed on each 512-bit block (B_1, B_2, …, B_n) of the padded message,
  - the 160-bit message digest is given by

$$CV_n\ (0)\ CV_n\ (1)\ Cn_1\ (2)\ CV_n\ (3)\ CV_n\ (4)$$

Lecture 14 Key Exchange

A Scenario

- Alice and Bob want to communicate with each other with symmetric encryption.

- How to distribute the shared secret key between Alice and Bob?
  - Secret key=encryption key=decryption key

Diffie-Hellman Key Exchange

- public-key distribution scheme
  - cannot be used to exchange an arbitrary message
  - rather it can establish a common key
- **security relies on the difficulty of computing discrete logarithms**
  - Recall RSA

Diffie-Hellman Key Exchange

- Alice sends $A = g^a \bmod p$ to Bob
- Bob sends $B = g^b \bmod p$ to Alice
- shared session key for users is K_{AB}:
  - $K_{AB} = g^{ab} \bmod p$

    $= A^b \bmod p$ (which Bob can compute)

    $= B^a \bmod p$ (which Alice can compute)

Diffie-Hellman Key Exchange

- Once Alice and Bob obtain K_{AB}:
  - They can use it as the shared secret key for symmetric encryption direclty

Diffie-Hellman Key Exchange

- g can be small
  - 2 or 5 is common
- a, b, p should be large
- attacker needs a or b to obtain the session key
  - $A = g^a \bmod p$ and $B = g^b \bmod p$
  - must solve discrete logarithm (not logarithm)

Diffie-Hellman Example

- Alice & Bob who wish to establish a shared secret key
  - agree on prime p=353 and g=3
- select random secret keys:
  - A chooses a=97, B chooses b=233
- compute respective public keys:
  - $A=3^{97}$ mod 353 = 40 (Alice)
  - $B=3^{233}$ mod 353 = 248 (Bob)
- compute shared session key as:
  - $K_{AB} = B^{a}$ mod 353 = 248^{97} = 160 (Alice)
  - $K_{AB} = A^{b}$ mod 353 = 40^{233} = 160 (Bob)