Computer & Information Security (372-1-460-1)

Cryptographic Algorithms

Dept. of Software and Information Systems Engineering, Ben-Gurion University

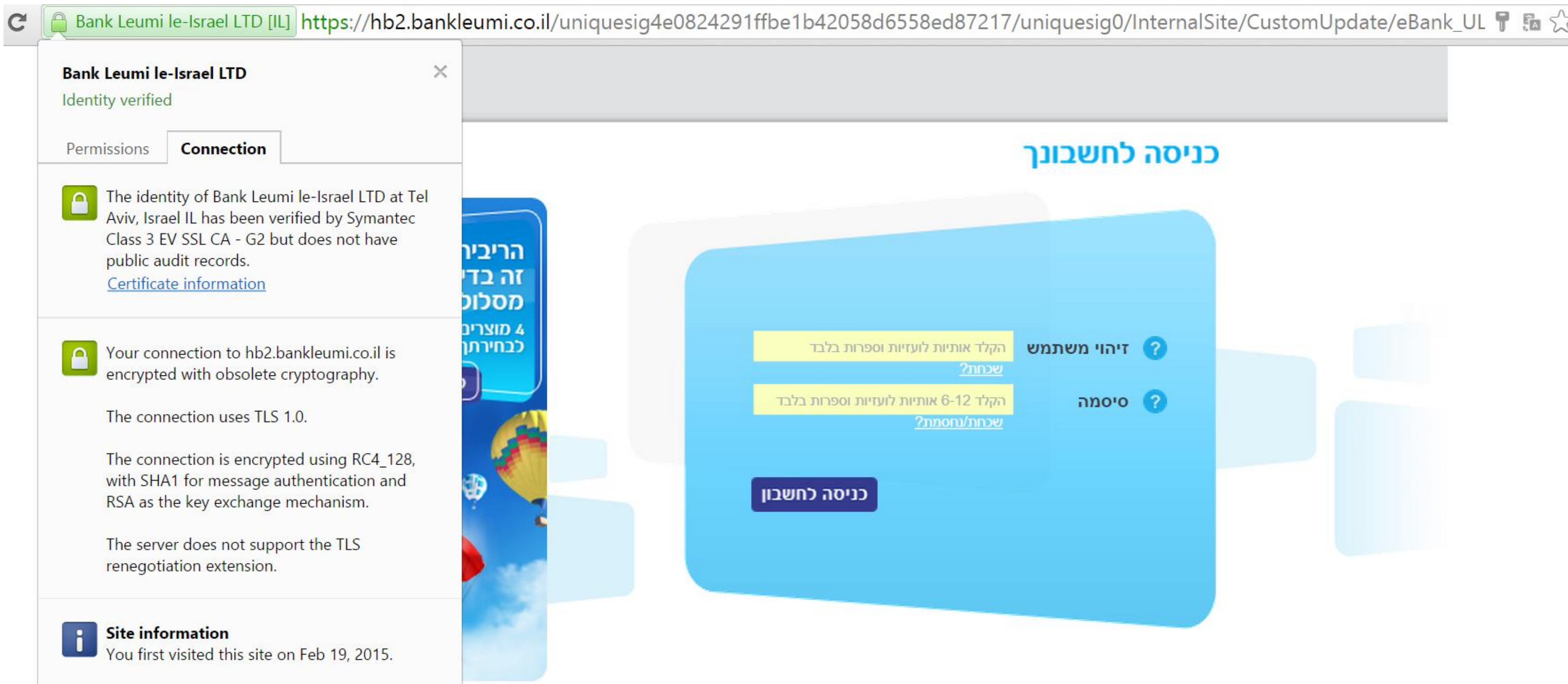
Prof. Yuval Elovici, Dr. Asaf Shabtai

{elovici, shabtaia}@bgu.ac.il

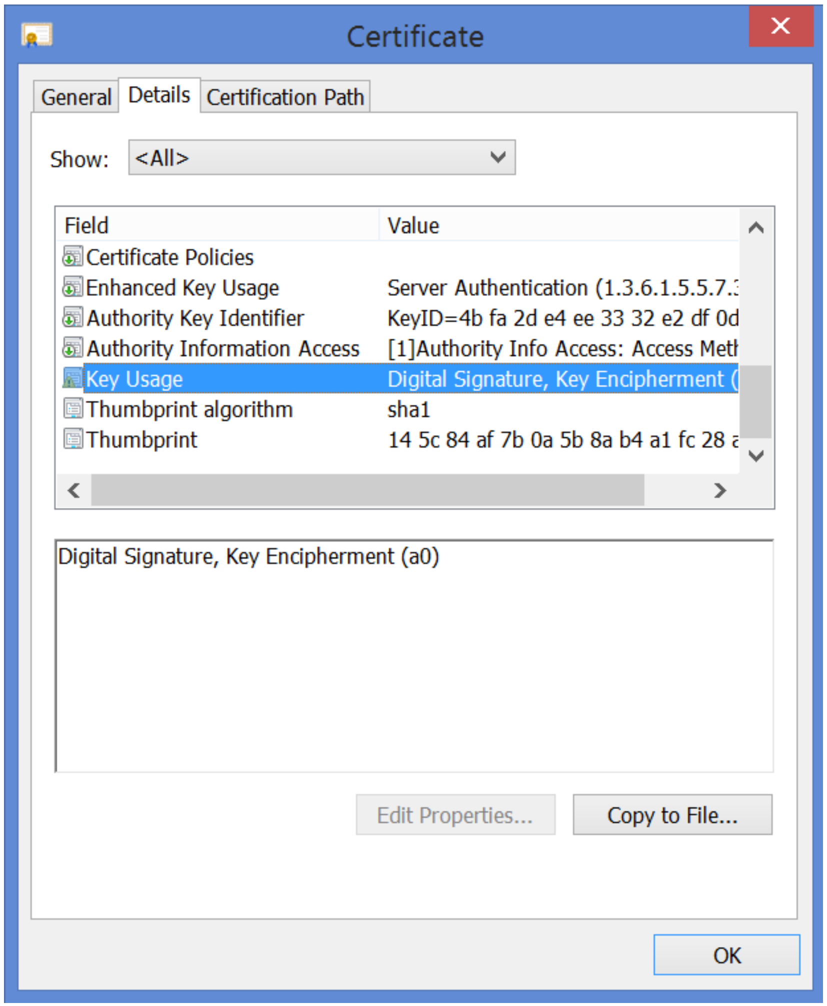
Spring, 2018

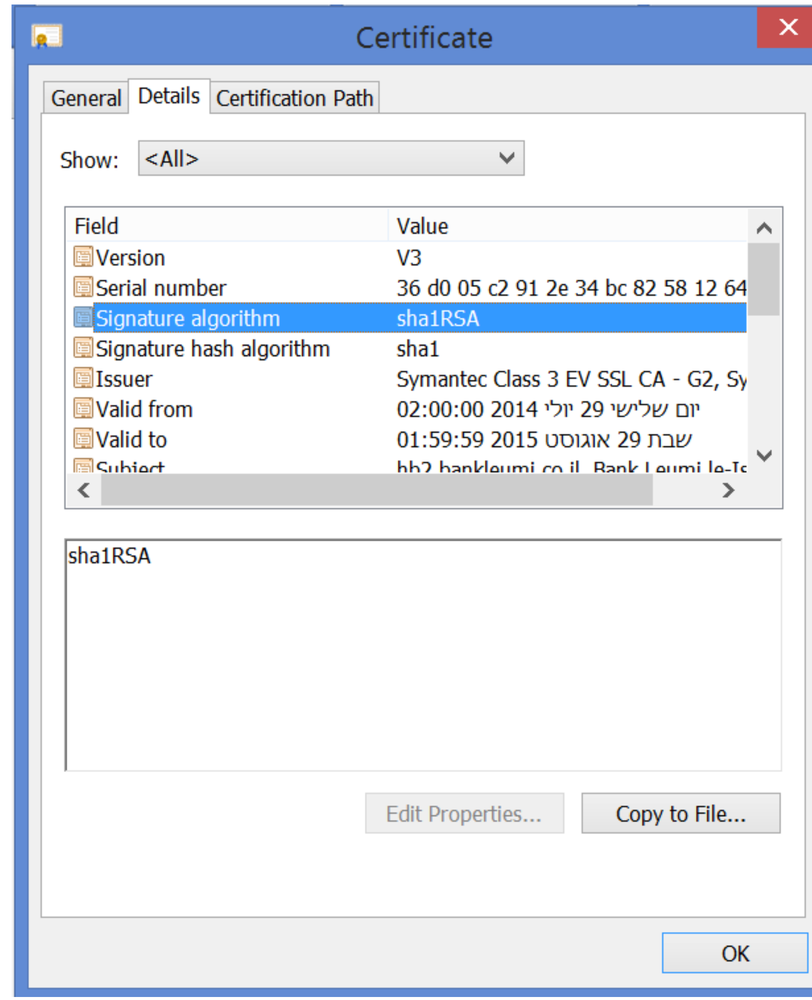
Applications of encryption

Applications of encryption

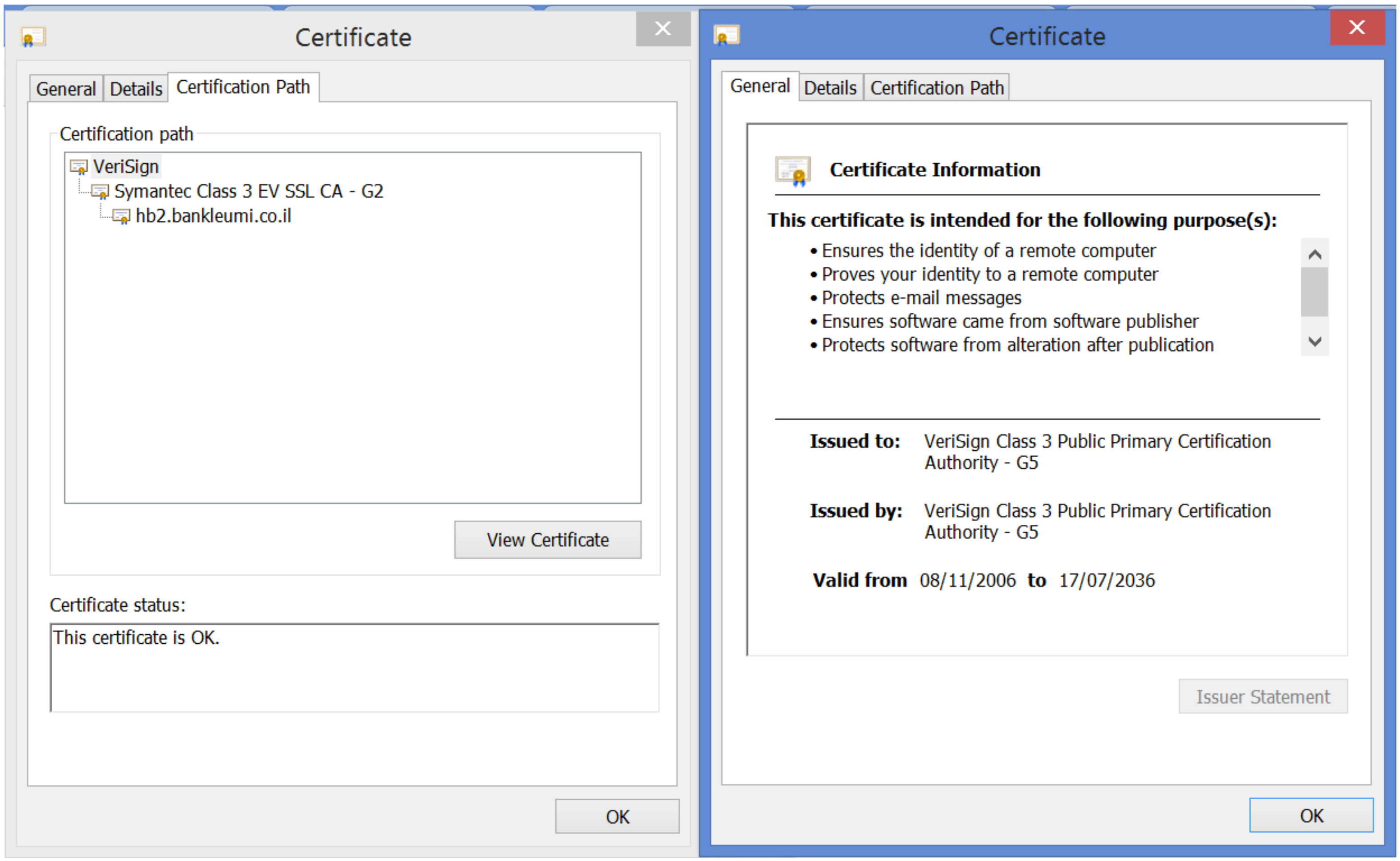


Applications of encryption

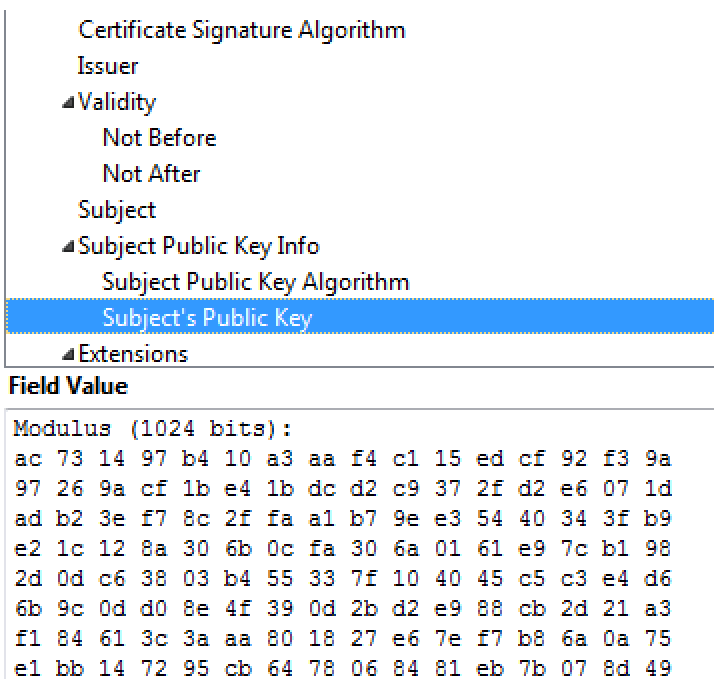


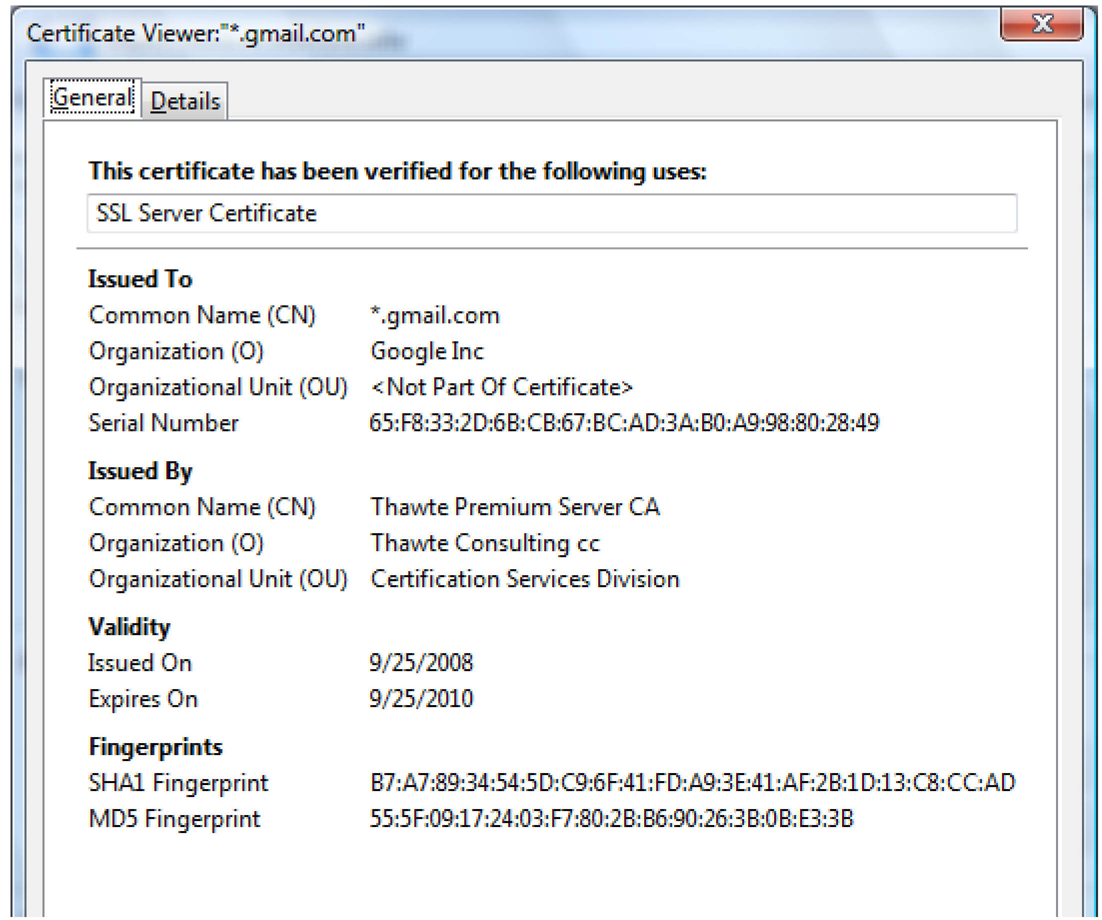


Applications of encryption

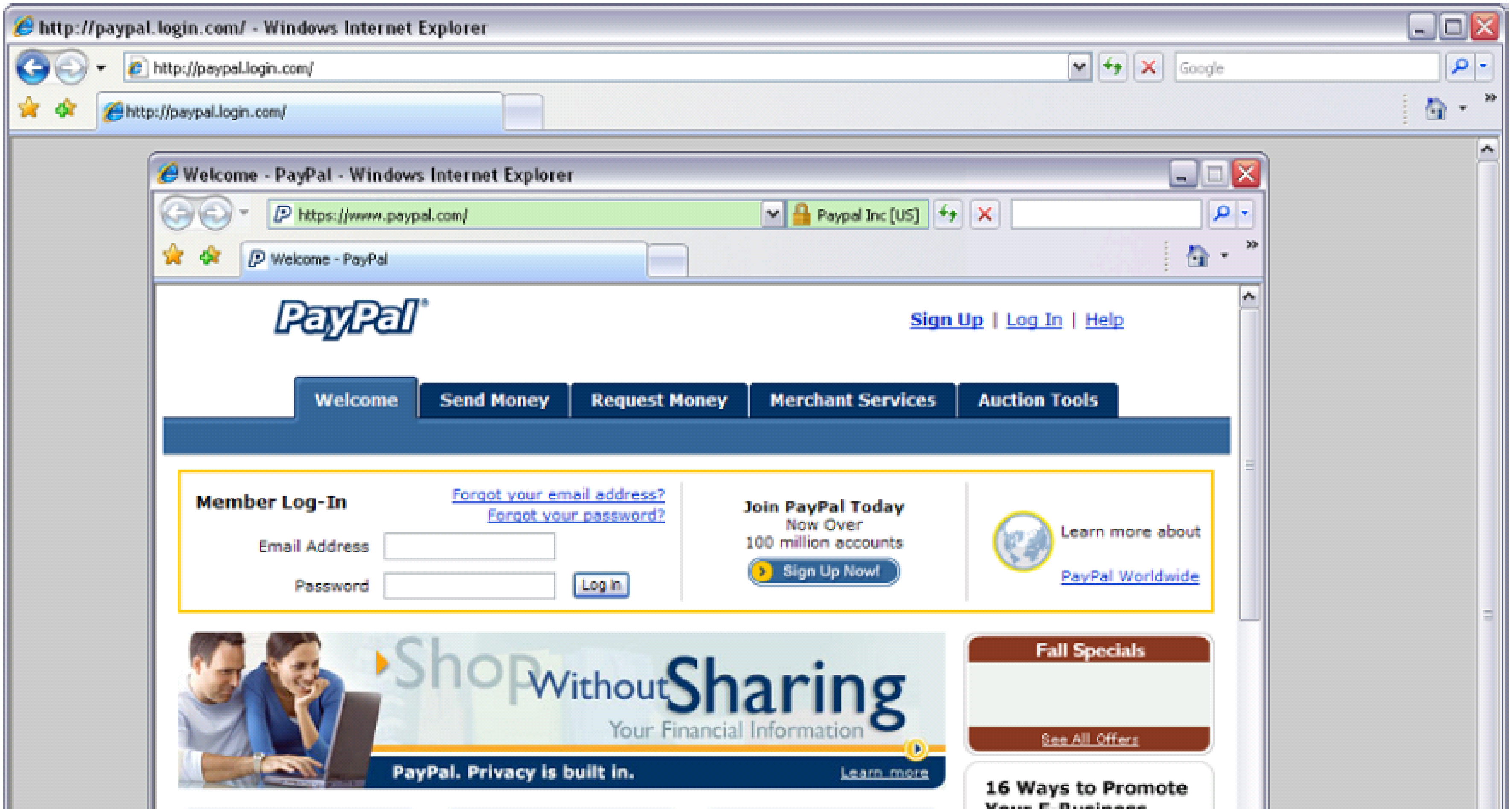


Applications of encryption



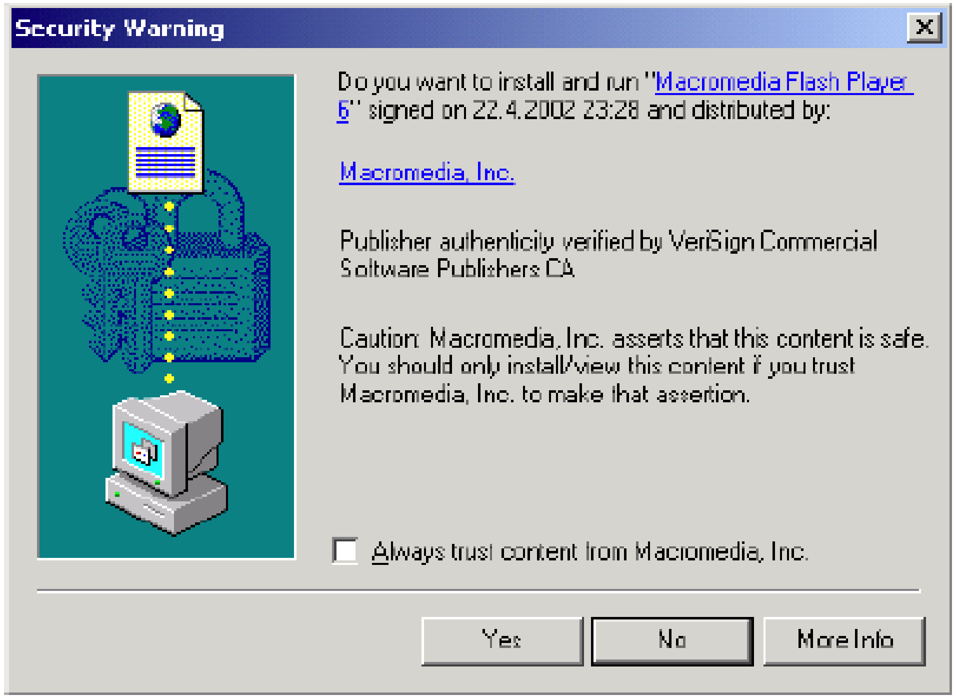


A general UI attack: picture-in-picture



Using Digital signatures for code signing

– Provide assurance of the authenticity and integrity of software codes.
– The executable files, or possibly the entire installation package of a program, are wrapped with a digitally signed envelope, which allows the end user to verify the signature before installing the software.



Cryptography classification

- operations used
  - Substitution
  - transposition
- keys used
  - symmetric
  - asymmetric
- how plaintext is processed
  - block cipher – processes input one block of elements at a time
    - Difficult to design: must resist subtle attacks such as differential attacks, linear attacks, brute-force, …
  - stream cipher – processes the input elements continuously

Cryptanalysis [595]

type of attack	known to cryptanalyst	
Ciphertext only	•Encryption algorithm •Ciphertext to be decoded	- Least info (very hard) = Brute Force on all possible keys - Statistical tests + prior knowledge about plaintext's language (English, Java), only weak encryption algorithms fails
Known plaintext	•Encryption algorithm •Ciphertext to be decoded •One or more plaintext-ciphertext pairs formed with the secret key	- More info (still hard) – helps deducing the encryption key - E.g., Email message has known format - Certain key words inside specific locations in the file such as: "Bank Account", "Date" - Probable-Word-attack
Chosen plaintext	•Encryption algorithm •Ciphertext to be decoded •Plaintext message chosen by cryptanalyst, together with its corresponding ciphertext generated with the secret key	- The attacker has an access to the encryption system - Then can encrypt specific patterns that will help him to reveal the structure of the key
Chosen ciphertext	•Encryption algorithm •Ciphertext to be decoded •Purported ciphertext chosen by cryptanalyst, together with its corresponding decrypted plaintext generated with the secret key	- Less common - The attacker has an access to the encryption system - Then can decrypt chosen ciphertext patterns that will help him to reveal the structure of the key
Chosen text	•Encryption algorithm •Ciphertext to be decoded •Plaintext message chosen by cryptanalyst, together with its corresponding ciphertext generated with the secret key •Purported ciphertext chosen by cryptanalyst, together with its corresponding decrypted plaintext generated with the secret key	- Less common - Combines both Chosen plaintext and Chosen ciphertext

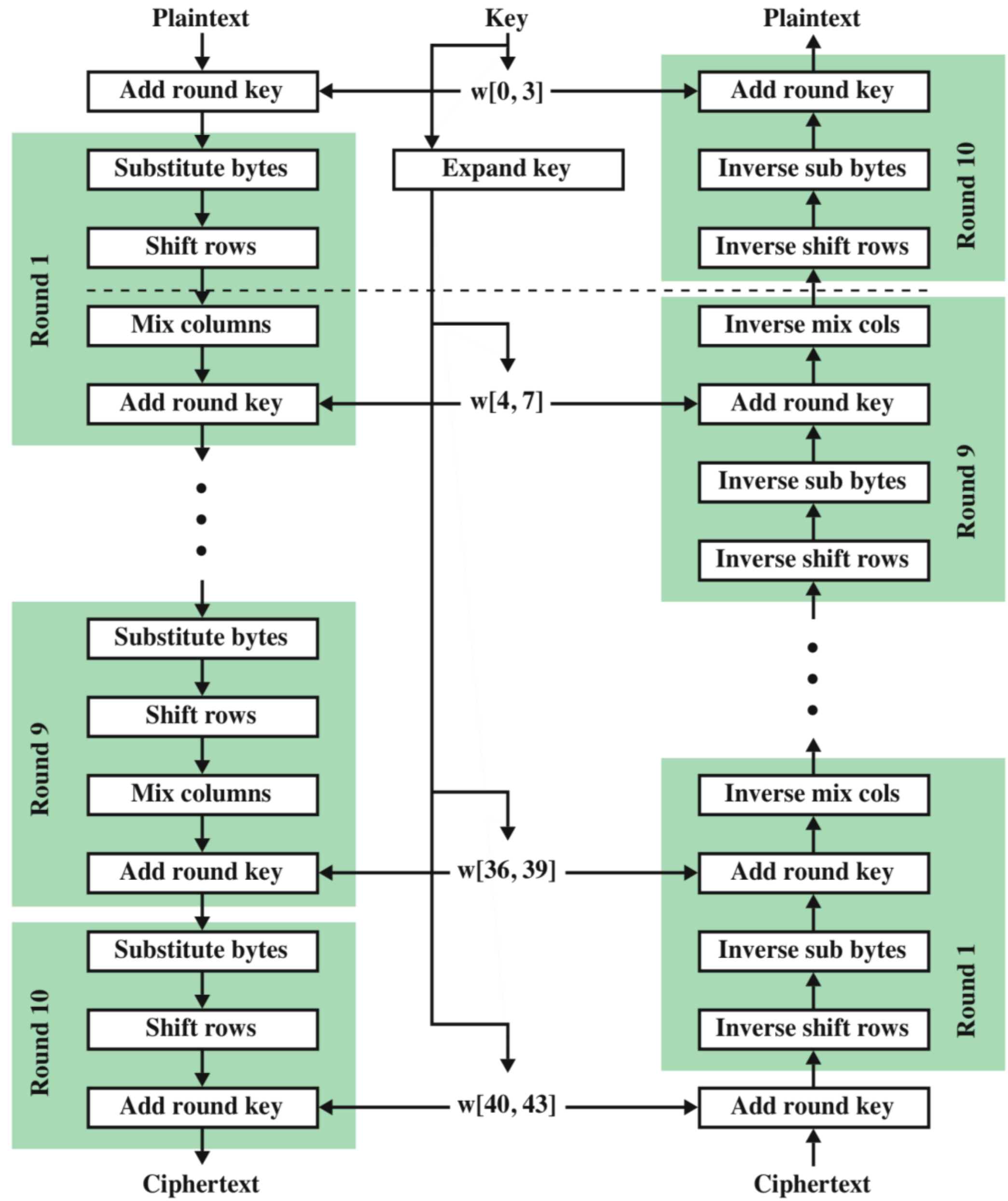
Computationally secure encryption schemes

- encryption is computationally secure if
  - cost of breaking cipher exceeds value of information
  - time required to break cipher exceeds the useful lifetime of the information
- usually very difficult to estimate the amount of effort required to break
- can estimate time/cost of a brute-force attack
  - on average, half of the keys must be tried

Advanced Encryption Standard (AES)

- AES is more secure and efficient than DES/3DES
- block length of 128 bits
- key length that can be 128 /192/ 256 bits
- not a Feistel structure
- applies 10 rounds
- four stages
  - substitute bytes (SubBytes)
  - shift rows
  - mix columns
  - add round key (the only stage that uses the key)

Advanced Encryption Standard (AES)



(a) Encryption

(b) Decryption

Advanced Encryption Standard (AES)

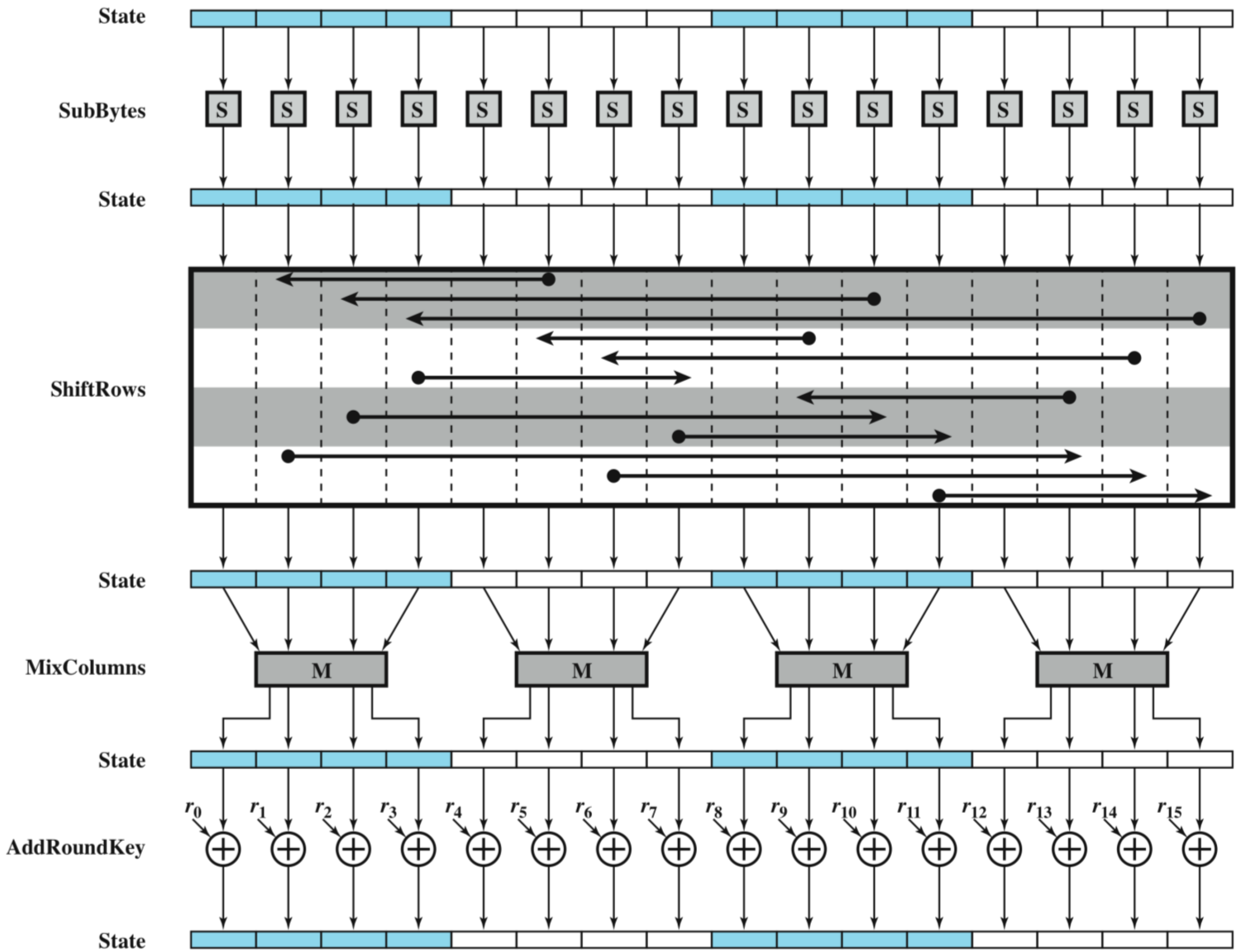
- Plaintext and Keys (128 bit) represented by 4X4 matrix:
  - each cell contains 1 byte (8 bits)
  - bits are ordered by columns from the leftmost to the rightmost

- 128 bit: 11101010100000111010111001111000000000100……11000101

  (E A) (8 3) (5 c) (F 0) (0 4) (C 5)

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

AES Round Structure

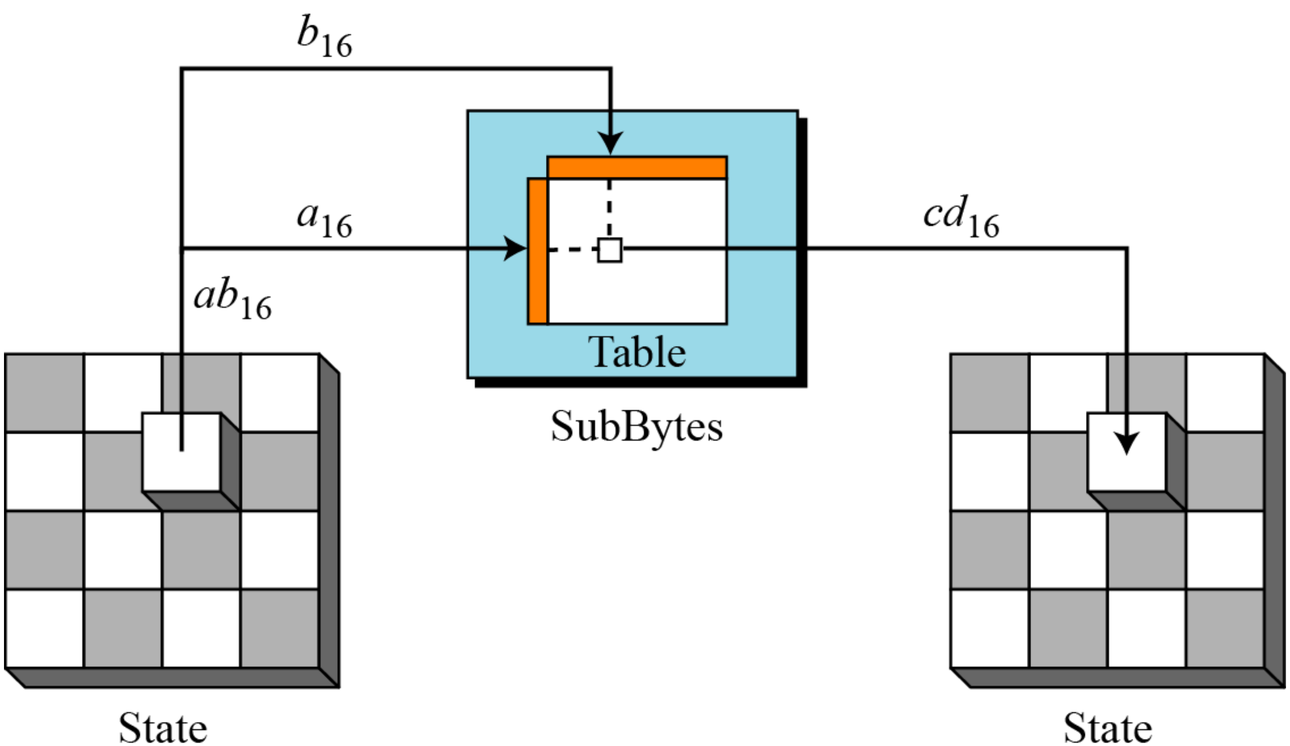


Substitute Bytes

- a simple table lookup in S-box
  - a 16×16 matrix of byte values
  - a permutation of all possible 256 8-bit values
  - maps each byte to a new value
    - e.g. {95} maps to {2A}
- constructed using finite field properties
  - designed to be resistant to known cryptanalytic attacks
- decrypt uses inverse of S-box

Substitute Bytes

- ab_{16} = 11101010
  (E A)



S-box (Encryption)

		y															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
x	0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
	1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
	2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
	3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
	4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
	5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
	6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
	7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
	8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
	9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
	A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
	B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
	C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
	D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
	E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
	F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

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

→

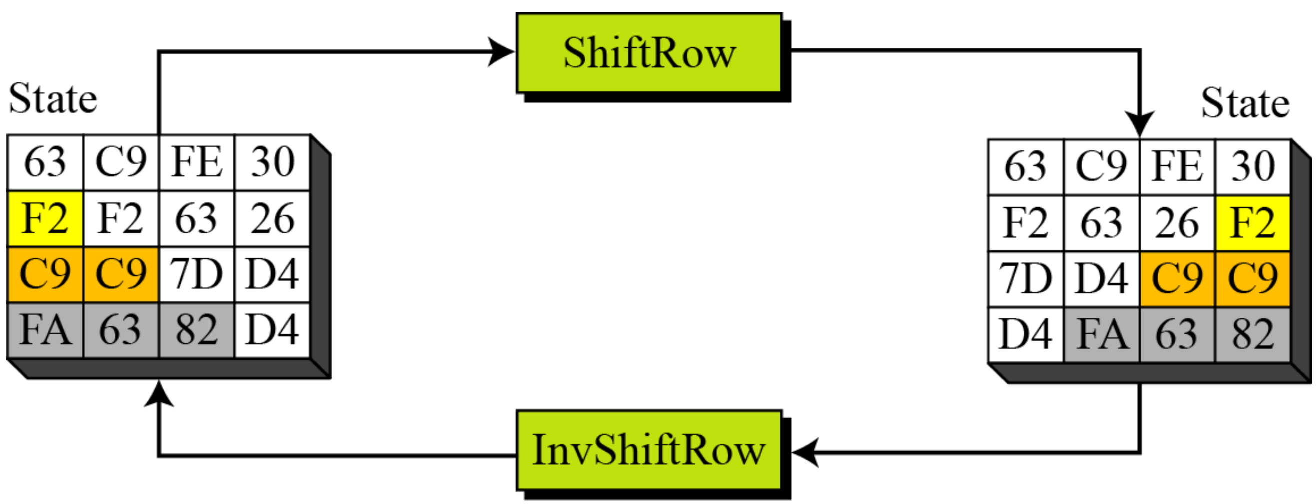
87	F2	4D	97
EC	6E	4C	90
4A	C3	46	E7
8C	D8	95	A6

Inverse S-box (Decryption)

		y															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
x	0	52	09	6A	D5	30	36	A5	38	BF	40	A3	9E	81	F3	D7	FB
	1	7C	E3	39	82	9B	2F	FF	87	34	8E	43	44	C4	DE	E9	CB
	2	54	7B	94	32	A6	C2	23	3D	EE	4C	95	0B	42	FA	C3	4E
	3	08	2E	A1	66	28	D9	24	B2	76	5B	A2	49	6D	8B	D1	25
	4	72	F8	F6	64	86	68	98	16	D4	A4	5C	CC	5D	65	B6	92
	5	6C	70	48	50	FD	ED	B9	DA	5E	15	46	57	A7	8D	9D	84
	6	90	D8	AB	00	8C	BC	D3	0A	F7	E4	58	05	B8	B3	45	06
	7	D0	2C	1E	8F	CA	3F	0F	02	C1	AF	BD	03	01	13	8A	6B
	8	3A	91	11	41	4F	67	DC	EA	97	F2	CF	CE	F0	B4	E6	73
	9	96	AC	74	22	E7	AD	35	85	E2	F9	37	E8	1C	75	DF	6E
	A	47	F1	1A	71	1D	29	C5	89	6F	B7	62	0E	AA	18	BE	1B
	B	FC	56	3E	4B	C6	D2	79	20	9A	DB	C0	FE	78	CD	5A	FA
	C	1F	DD	A8	33	88	07	C7	31	B1	12	10	59	27	80	EC	5F
	D	60	51	7F	A9	19	B5	4A	0D	2D	E5	7A	9F	93	C9	9C	EF
	E	A0	E0	3B	4D	AE	2A	F5	B0	C8	EB	BB	3C	83	53	99	61
	F	17	2B	04	7E	BA	77	D6	26	E1	69	14	63	55	21	0C	7D

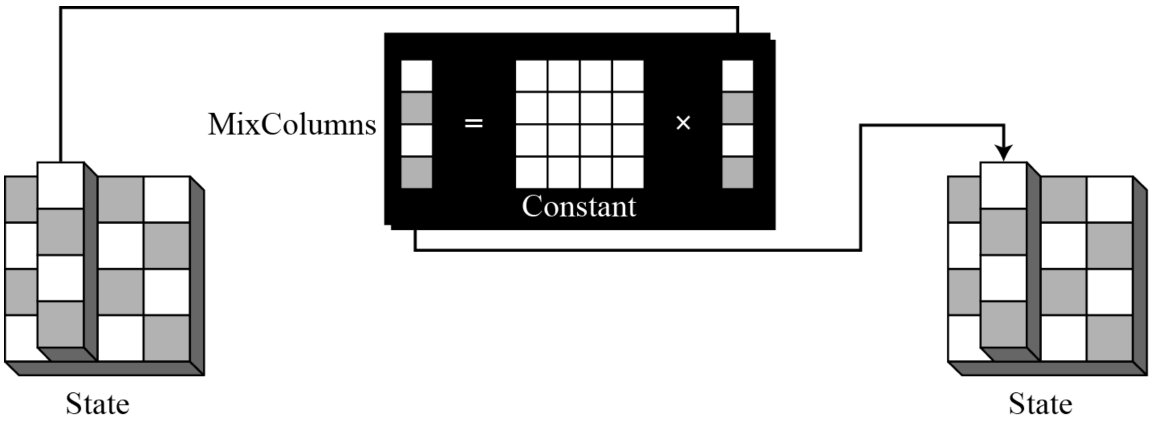
Shift Rows

- on encryption: shift each row of State by 0,1,2,3 bytes respectively
- ensures that each column is now spread over four columns
- decrypt does reverse



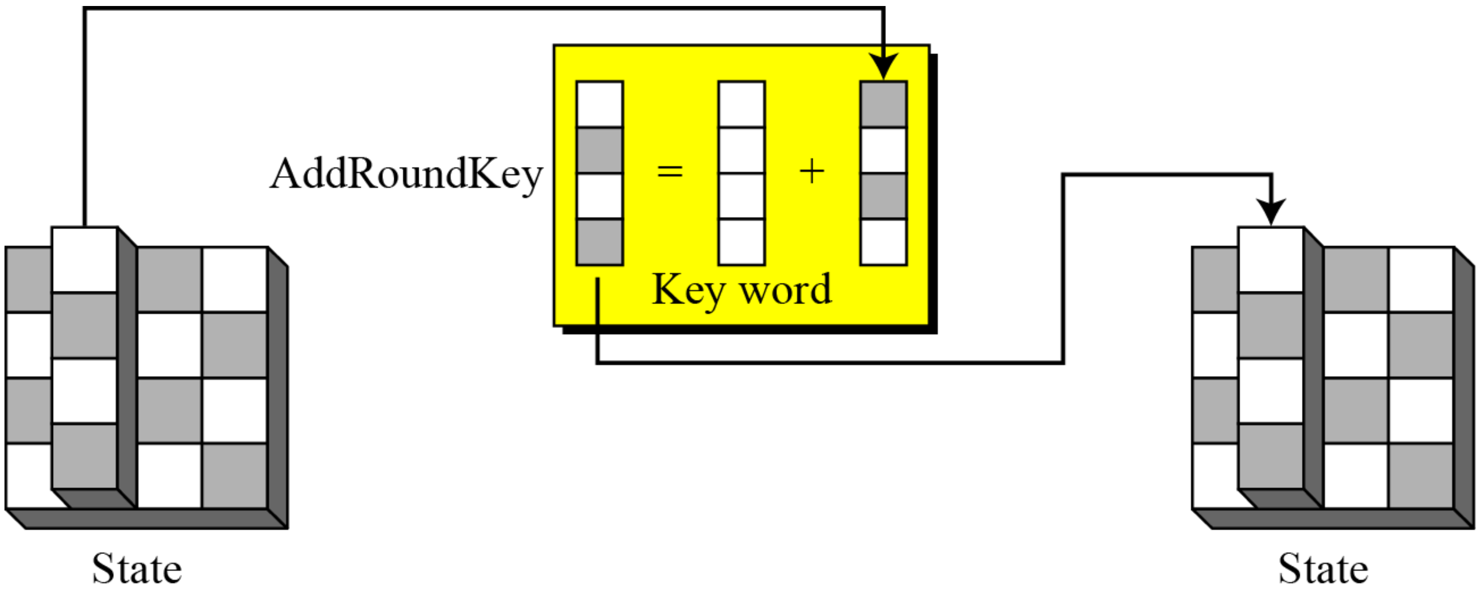
Mix Column

- mix columns
  - operates on each column individually
  - mapping each byte to a new value that is a function of all four bytes in the column
  - use of equations over finite fields
  - to provide good mixing of bytes in column



Add round key

- simply XOR State with bits of expanded key
- security from complexity of round key expansion and other stages of AES



Key-Expansion and creation of Round-Key

- Using the symmetric key 128 bit (16 bytes)
  - create expended-Key of 176 bytes
  - from which create 11 different round keys
  - each key is 16 bytes = 4 words of 4 bytes
  - the first key in is the original key
  - using a complex finite field algorithm, each added Word in the expanded key depends on the two previous words w[i-1] and w[i-4]

Key-Expansion and creation of Round-Key

Original Key
16 byte Key (4 Words)

Word0	AC	19	28	57
Word1	77	FA	D1	5C
Word2	66	DC	29	00
Word3	ED	A5	A6	BC

Expanded Key
176 byte Key (44 Words)

Word0	AC	19	28	57	Original Key
Word1	77	FA	D1	5C	
Word2	66	DC	29	00	
Word3	ED	A5	A6	BC	
Word4 =	F(word3,word,0)				
Word5 =	F(word4,word,1)				
Word6 =	F(word5,word,2)				
Word7 =	F(word6,word,3)				
……					
….					
…..					
……					
Word40					
Word41					
Word42					
Word43					

Add round key

- simply XOR State with bits of expanded key
- security from complexity of round key expansion and other stages of AES

$$
\begin{array}{|c|c|c|c|}\hline 47 & 40 & A3 & 4C \\ \hline 37 & D4 & 70 & 9F \\ \hline 94 & E4 & 3A & 42 \\ \hline ED & A5 & A6 & BC \\ \hline \end{array}
\oplus
\begin{array}{|c|c|c|c|}\hline AC & 19 & 28 & 57 \\ \hline 77 & FA & D1 & 5C \\ \hline 66 & DC & 29 & 00 \\ \hline ED & A5 & A6 & BC \\ \hline \end{array}
=
\begin{array}{|c|c|c|c|}\hline EB & 59 & 8B & 1B \\ \hline 40 & 2E & A1 & C3 \\ \hline F2 & 38 & 13 & 42 \\ \hline 1E & 84 & E7 & D2 \\ \hline \end{array}
$$

AES steps

Stage 1: SubBytes

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

Stage 2: ShiftRows

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

Stage 3: MixColumns

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

Stage 4: AddRoundKey

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

$\oplus$

AC	19	28	57
77	FA	D1	5C
66	DC	29	00
ED	A5	A6	BC

=

EB	59	8B	1B
40	2E	A1	C3
F2	38	13	42
1E	84	E7	D2

RSA Public-Key Encryption

- by Rivest, Shamir & Adleman of MIT in 1977
- best known and widely used public-key algorithm
- uses exponentiation of integers modulo a prime

- encrypt: $C = M^e \bmod n$

- *decrypt:* $M = C^d \bmod n = (M^e)^d \bmod n = M$

- both sender and receiver know values of n and e
- only receiver knows value of d
- public-key encryption algorithm with
  – public key $PU = \{e, n\}$ and private key $PR = \{d, n\}$

RSA Algorithm [637\667]

Key Generation	
Select p, q	p and q both prime, $p \neq q$
Calculate $n = p \times q$	
Calculate $\phi(n) = (p-1)(q-1)$	
Select integer e	$\gcd(\phi(n), e) = 1;\ 1 < e < \phi(n)$
Calculate d	$de \bmod \phi(n) = 1$
Public key	$KU = \{e, n\}$
Private key	$KR = \{d, n\}$

Encryption	
Plaintext:	$M < n$
Ciphertext:	$C = M^e \pmod{n}$

Decryption	
Ciphertext:	C
Plaintext:	$M = C^d \pmod{n}$

Figure 21.5 The RSA Algorithm

public key $PU = \{e, n\}$

private key $PR = \{d, n\}$

$\Phi(n)$ – Euler totient function

counts the positive integers less than or equal to n that are relatively prime to n

$\Phi(9) = 6$

1,2,4,5,7,8 ?

$\Phi(7) = 6$

RSA Example - Key Generation

Key Generation	
Select p, q	p and q both prime, $p \neq q$
Calculate $n = p \times q$	
Calculate $\phi(n) = (p-1)(q-1)$	
Select integer e	$\gcd(\phi(n), e) = 1;\ 1 < e < \phi(n)$
Calculate d	$de \bmod \phi(n) = 1$
Public key	$KU = \{e, n\}$
Private key	$KR = \{d, n\}$

1. Select two prime numbers, $p = 17$ and $q = 11$.
2. Calculate $n = pq = 17 \times 11 = 187$.
3. Calculate $\phi(n) = (p - 1)(q - 1) = 16 \times 10 = 160$.
4. Select e such that e is relatively prime to $\phi(n) = 160$ and less than $\phi(n)$; we choose $e = 7$.
5. Determine d such that $de \bmod 160 = 1$ and $d < 160$. The correct value is $d = 23$, because $23 \times 7 = 161 = (1 \times 160) + 1$.

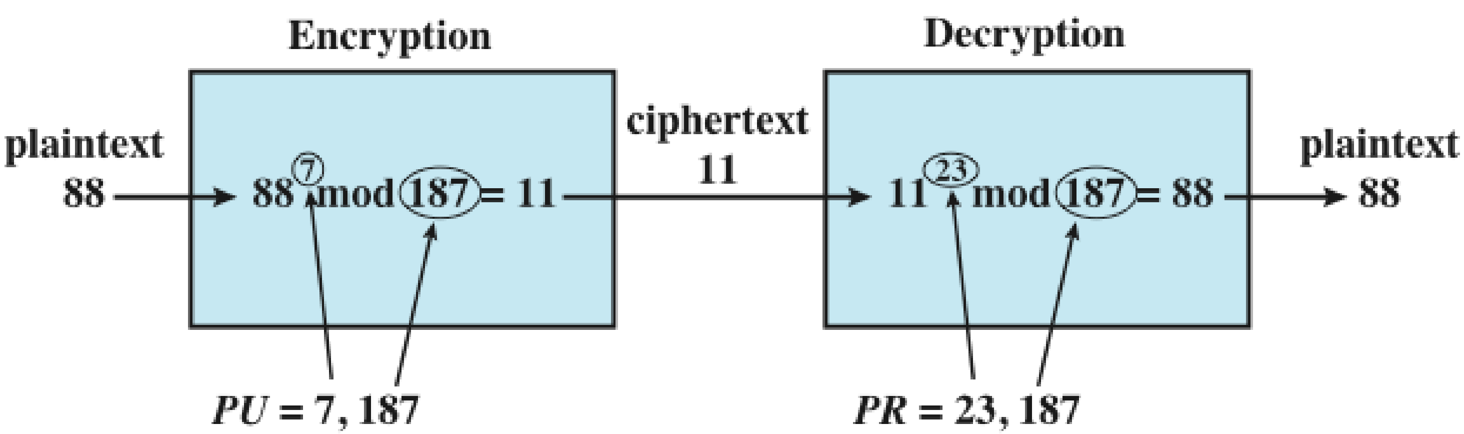
The resulting keys are public key $PU = \{7, 187\}$ and private key $PR = \{23, 187\}$.

RSA Example - Encryption\Decryption

For encrypting Message M with plaintext = 88

Encryption	
Plaintext:	$M < n$
Ciphertext:	$C = M^e \pmod{n}$

Decryption	
Ciphertext:	C
Plaintext:	$M = C^d \pmod{n}$



public key $PU = \{e=7, n=187\}$ $PR = \{d=23, n=187\}$ private key

Attacks on RSA

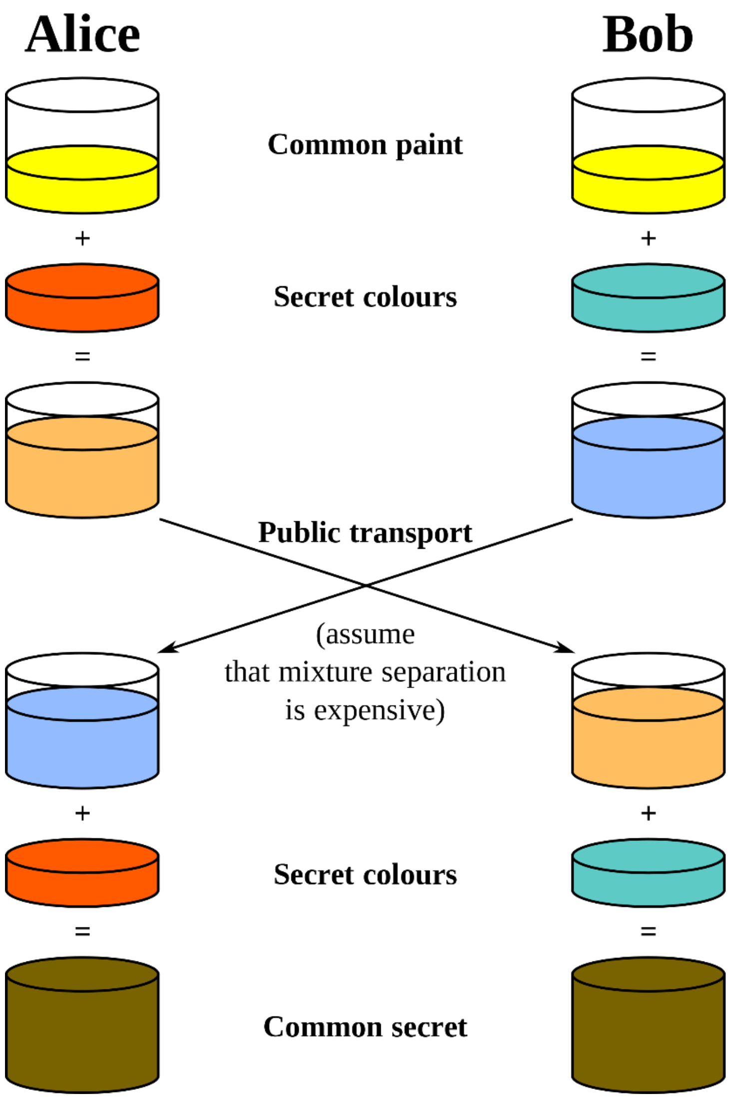
- brute force
  - trying all possible private keys
  - use larger key, but then the process is slower
- mathematical attacks (factoring n)
  - n = pXq → finding p,q enables to find Φ(n) and finally d
  - currently 1024-2048-bit keys seem secure
  - the threat still exists regarding to larger keys:
    - increasing computing power
    - refinement of factoring algorithms (QS, GNFS, SNFS)

Attacks on RSA

- timing attacks (on implementation)
  - Paul Kocher: possible to determine private key according to the time takes to decrypt message
  - use to prevent: constant time, random delays, blinding (multiply by random numbers)
- chosen ciphertext attacks (on RSA props)

Diffie-Hellman Key exchange

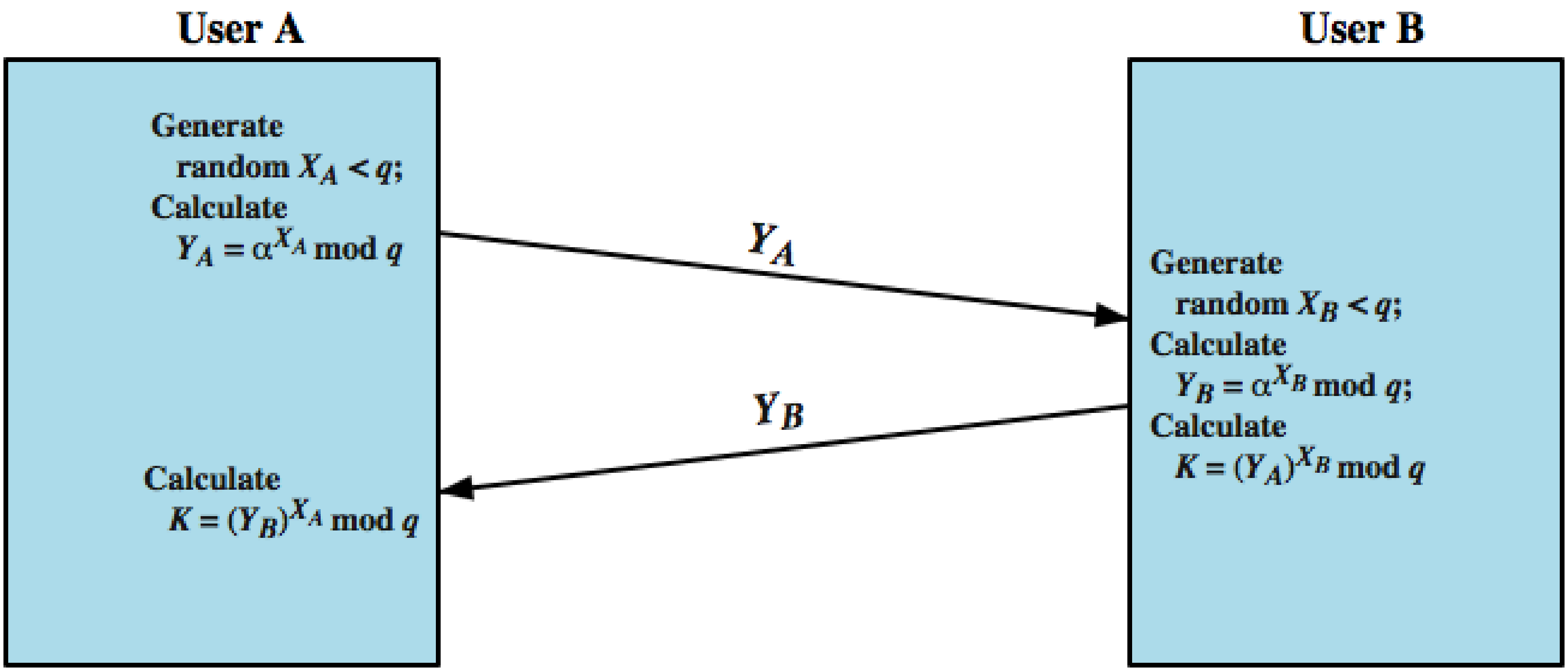
- first published public-key algorithm by Diffie and Hellman in 1976
- used in a number of commercial products
- practical method to exchange a secret key securely that can then be used for subsequent encryption of messages
- security relies on difficulty of computing discrete logarithms


Alice
Bob
Common paint
+
Secret colours
=
Public transport
(assume
that mixture separation
is expensive)
+
Secret colours
=
Common secret

Primitive root

- Let p be a prime
- Then b is a primitive root of p if the powers of b: 1, b, b^2, b^3, ... include all of the residue classes mod p (except 0)
- If p=7, then 3 is a primitive root for p
  1, 3, 9, 27, 81, 243 mod 7 =
  1, 3, 2, 6, 4, 5

Key Exchange Protocols



Diffie-Hellman: key exchange algorithm

Global Public Elements	
q	Prime number
α	$\alpha < q$ and α a primitive root of q

User A Key Generation	
Select private X_A	$X_A < q$
Calculate public Y_A	$Y_A = \alpha^{X_A} \bmod q$

User B Key Generation	
Select private X_B	$X_B < q$
Calculate public Y_B	$Y_B = \alpha^{X_B} \bmod q$

Generation of Secret Key by User A

$K = (Y_B)^{X_A} \bmod q$

Generation of Secret Key by User B

$K = (Y_A)^{X_B} \bmod q$

Figure 21.7 **The Diffie-Hellman Key Exchange Algorithm**

Diffie-Hellman: Proof

$$
\begin{aligned}
K &= (Y_B)^{X_A} \bmod q \\
&= (\alpha^{X_B} \bmod q)^{X_A} \bmod q \\
&= (\alpha^{X_B})^{X_B} \bmod q \\
&= \alpha^{X_B\, X_A} \bmod q \\
&= (\alpha^{X_A})^{X_B} \bmod q \\
&= (\alpha^{X_A} \bmod q)^{X_B} \bmod q \\
&= (Y_A)^{X_B} \bmod q
\end{aligned}
$$

Diffie-Hellman Example

- have
  - prime number $q = 353$
  - primitive root $\alpha = 3$
  - $X_A = 97$
  - $X_B = 233$

User A Key Generation	
Select private X_A	$X_A < q$
Calculate public Y_A	$Y_A = \alpha^{X_A} \bmod q$

User B Key Generation	
Select private X_B	$X_B < q$
Calculate public Y_B	$Y_B = \alpha^{X_B} \bmod q$

- A and B each compute their public keys
  - A computes $Y_A = 3^{97} \bmod 353 = 40$
  - B computes $Y_B = 3^{233} \bmod 353 = 248$
- then exchange and compute secret key:
  - for A: $K = (Y_B)^{XA} \bmod 353 = 248^{97} \bmod 353 = 160$
  - for B: $K = (Y_A)^{XB} \bmod 353 = 40^{233} \bmod 353 = 160$

$$K = (Y_B)^{X_A} \bmod q$$

$$K = (Y_A)^{X_B} \bmod q$$

- attacker must solve:
  - $3^{Xa} \bmod 353 = 40$, with brute force which is feasible (97)
  - desired answer is 97, then compute key as B does
  - impractical to do it for large prime numbers (α, q)
  - since it is hard to calculate discrete logarithms

$$X_B = \text{dlog}_{\alpha,q}(Y_B)$$

Man-in-the-Middle Attack

- attack is:
  1. Darth generates private keys X_{D1} & X_{D2}, and their public keys Y_{D1} & Y_{D2}
  2. Alice transmits Y_A to Bob
  3. Darth intercepts Y_A and transmits Y_{D1} to Bob. Darth also calculates K2
  4. Bob receives Y_{D1} and calculates K1
  5. Bob transmits X_A to Alice
  6. Darth intercepts X_A and transmits Y_{D2} to Alice. Darth calculates K1
  7. Alice receives Y_{D2} and calculates K2
- all subsequent communications compromised

Man-in-the-Middle Attack

- attack is:
  1. A → E alpha^x mod p
  2. B → E alpha^y mod p
  3. E → B alpha^z mod p
  4. E → A alpha^ze mod p
  5. Darth generates private keys X_{D1} & X_{D2}, and their public keys Y_{D1} & Y_{D2}
  6. Alice transmits Y_A to Bob
  7. Darth intercepts Y_A and transmits Y_{D1} to Bob. Darth also calculates K2
  8. Bob receives Y_{D1} and calculates K1
  9. Bob transmits X_A to Alice
  10. Darth intercepts X_A and transmits Y_{D2} to Alice. Darth calculates K1
  11. Alice receives Y_{D2} and calculates K2
- all subsequent communications compromised

Simple Hash Functions

- a one-way or secure hash function used in:
  - message authentication
  - source authentication (digital signatures)
- all hash functions process input a block at a time in an iterative fashion
- one of simplest hash functions is the bit-by-bit exclusive-OR (XOR) of each block

$$C_i = b_{i1} \oplus b_{i2} \oplus \ldots \oplus b_{im}$$

  - effective data integrity check on random data
  - less effective on more predictable data
  - virtually useless for data security

Secure Hash Functions [657]

	Bit 1	Bit 2	• • •	Bit n
Block 1	b_{11}	b_{21}		b_{n1}
Block 2	b_{12}	b_{22}		b_{n2}
	⋮	⋮	⋮	⋮
Block m	b_{1m}	b_{2m}		b_{nm}
Hash code	C_1	C_2		C_n

Characteristics

- Given M (plaintext), it is easy to compute h (hash code)
- Given h, it is hard to compute M such that H(M) = h
- Given M, it is hard to find another message, M', such that H(M) = H(M')
- Collision-resistance: it is hard to find two random messages, M and M', such that H(M) = H(M')

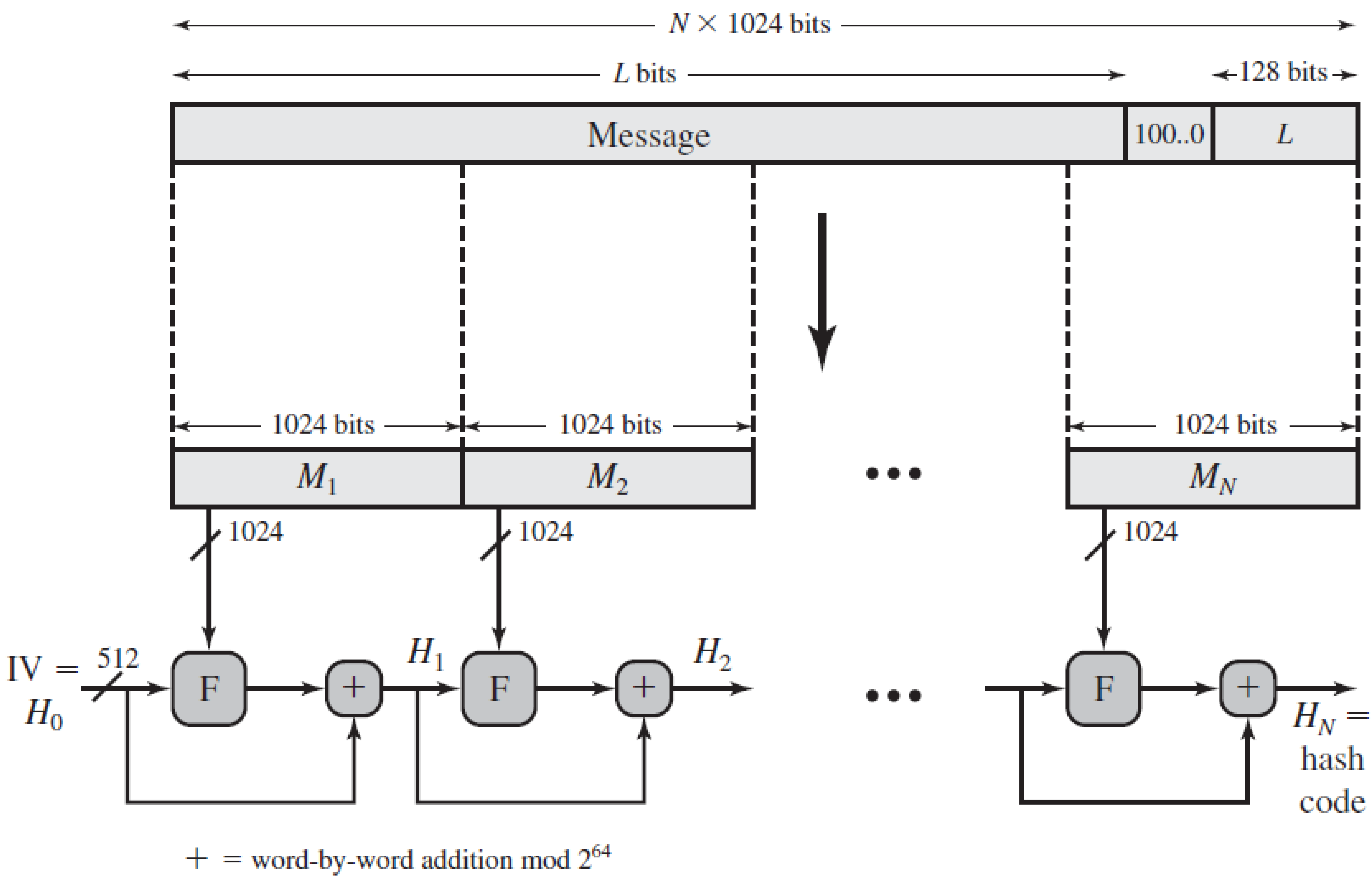
Secure Hash Algorithm (SHA)

- SHA was originally developed by NIST
- published as FIPS 180 in 1993 (federal information processing standard)
- was revised in 1995 as SHA-1
  - produces 160-bit hash values
- NIST issued revised FIPS 180-2 in 2002
  - adds 3 additional versions of SHA: SHA-256, SHA-384, SHA-512
  - with 256/384/512-bit hash values
  - same basic structure as SHA-1 but greater security

SHA-512 Steps

- Step 1: Append padding bits
  - 896 mod 1024
  - Padding always added (1-1024 bit)
- Step 2: Append length
  - 128 bits that represent the length of the message before padding
- Step 3: Initialize hash buffer (IV)
  - A 512 bit buffer of 8 X 64 bit registers (a,b,c,d,e,f,g,h)
- Step 4: Process message in 1024-bit (128-word) blocks
  - F = The heart module of SHA-512 and is being done 80 rounds
- Step 5: Output
  - A 512 bit message digest

SHA-512 Structure



SHA-512
Initial Values

- initialise 8 (512-bit) buffer (A,B,C,D,E,F,G,H) to

6a09e667f3bcc908
Bb67ae8584caa73b
3c6ef372fe94f82b
a54ff53a5f1d36f1
510e527fade682d1
9b05688c2b3e6c1f
1f83d9abfb41bd6b
5be0cd19137e2179

SHA-512 Round

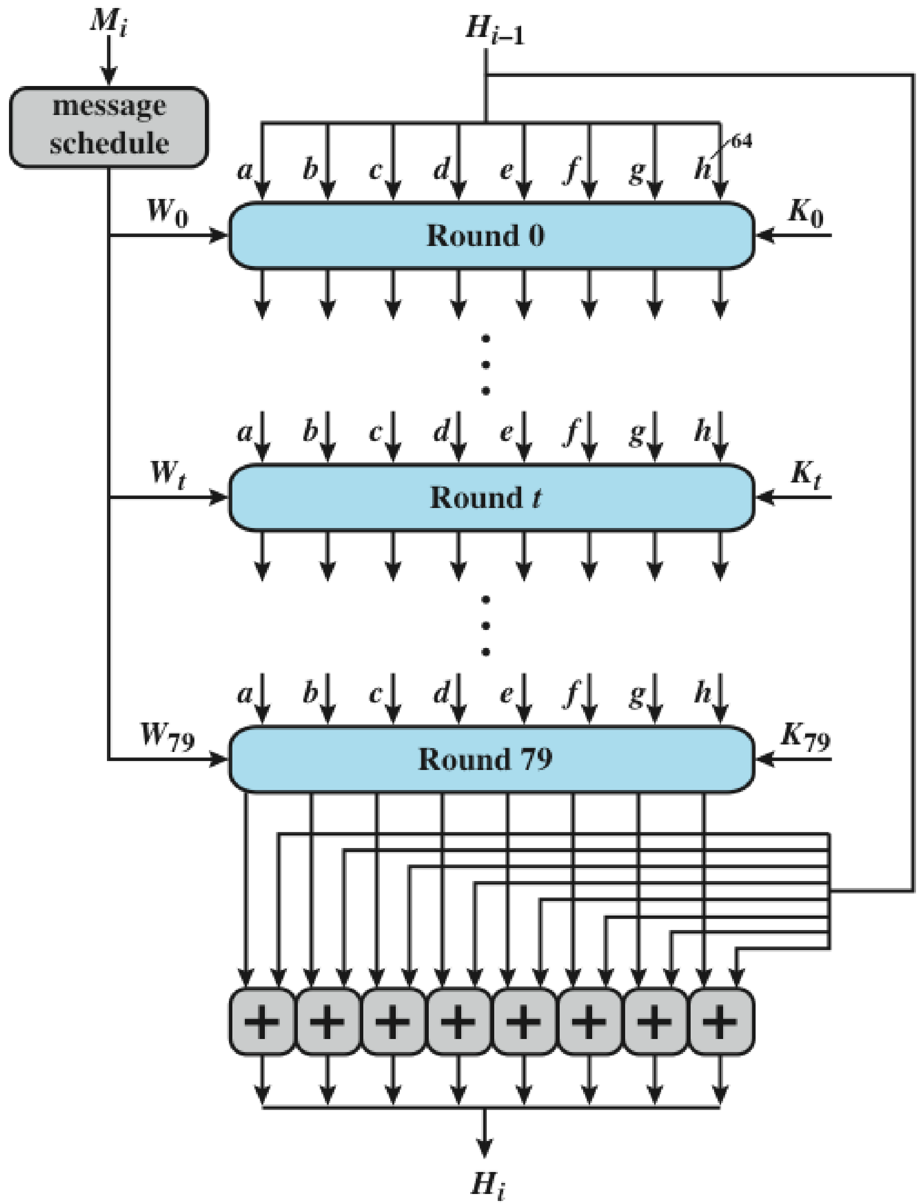
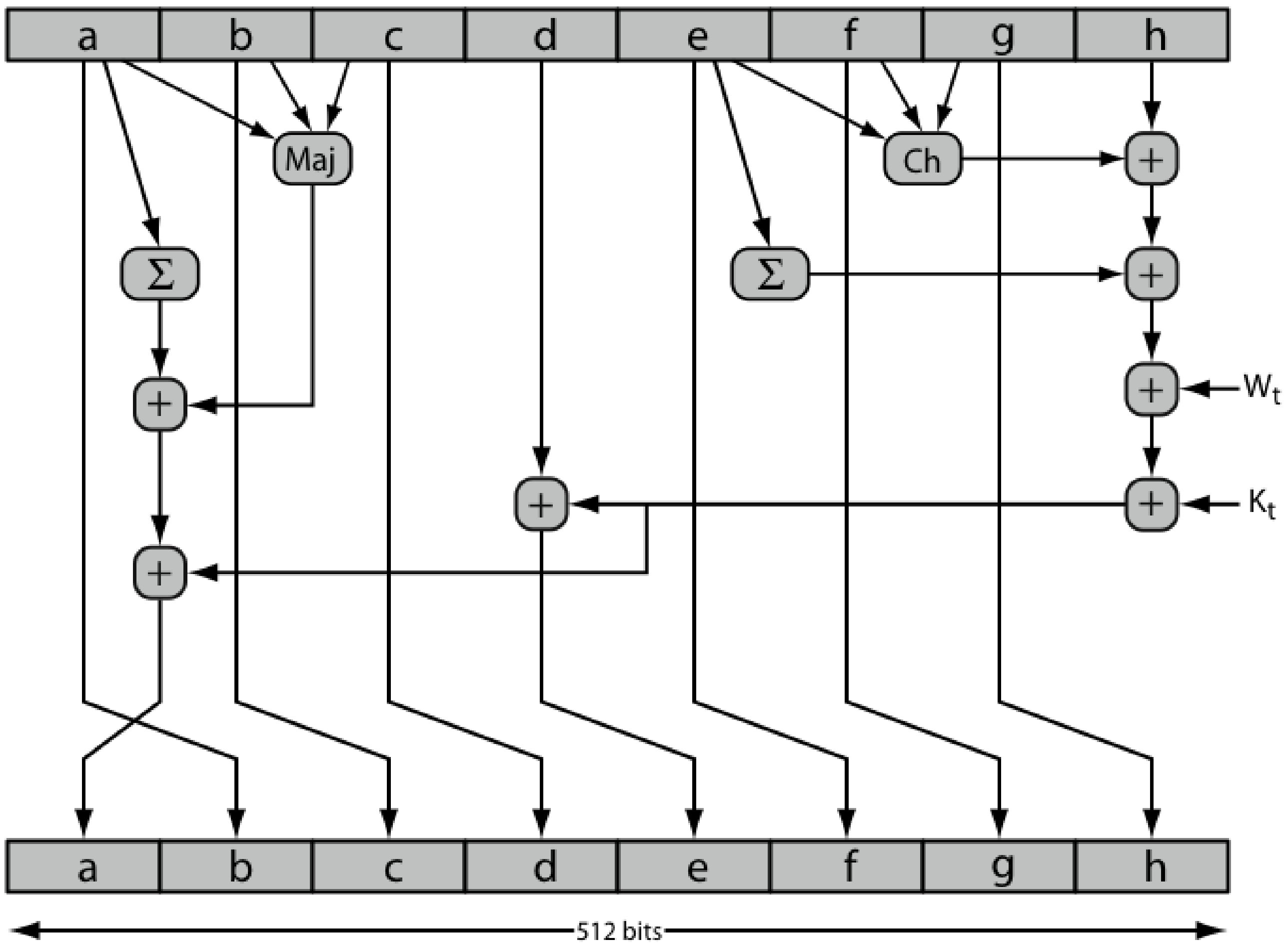


Figure 21.3 SHA-512 Processing of a Single 1024-Bit Block

SHA-512
Message Scheduling

$$W_t = \begin{cases} M_t\,, & 0 \le t \le 15 \\ f_6(W_{t\text{-}2}) \oplus W_{t\text{-}7} \oplus f_5(W_{t\text{-}15}) \oplus W_{t\text{-}16}, & 16 \le t \le 79 \end{cases}$$

SHA-512 Round Function



80 Constants (K_t)

428a2f98d728ae22	7137449123ef65cd	b5c0fbcfec4d3b2f	e9b5dba58189dbbc
3956c25bf348b538	59f111f1b605d019	923f82a4af194f9b	ab1c5ed5da6d8118
d807aa98a3030242	12835b0145706fbe	243185be4ee4b28c	550c7dc3d5ffb4e2
72be5d74f27b896f	80deb1fe3b1696b1	9bdc06a725c71235	c19bf174cf692694
e49b69c19ef14ad2	efbe4786384f25e3	0fc19dc68b8cd5b5	240ca1cc77ac9c65
2de92c6f592b0275	4a7484aa6ea6e483	5cb0a9dcbd41fbd4	76f988da831153b5
983e5152ee66dfab	a831c66d2db43210	b00327c898fb213f	bf597fc7beef0ee4
c6e00bf33da88fc2	d5a79147930aa725	06ca6351e003826f	142929670a0e6e70
27b70a8546d22ffc	2e1b21385c26c926	4d2c6dfc5ac42aed	53380d139d95b3df
650a73548baf63de	766a0abb3c77b2a8	81c2c92e47edaee6	92722c851482353b

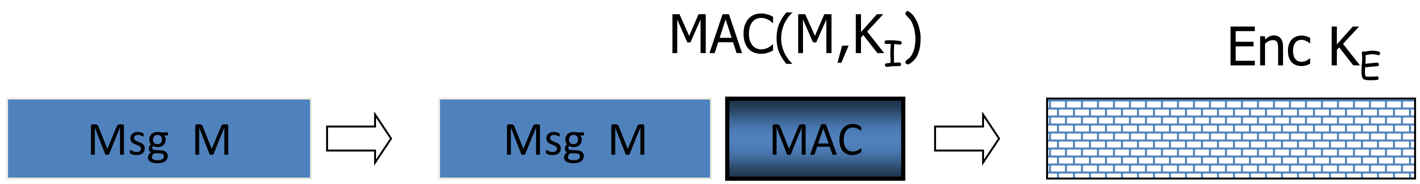
80 Constants (K_t)

a2bfe8a14cf10364	a81a664bbc423001	c24b8b70d0f89791	c76c51a30654be30
d192e819d6ef5218	d69906245565a910	F40e35855771202a	106aa07032bbd1b8
19a4c116b8d2d0c8	1e376c085141ab53	2748774cdf8eeb99	34b0bcb5e19b48a8
391c0cb3c5c95a63	4ed8aa4ae3418acb	5b9cca4f7763e373	682e6ff3d6b2b8a3
748f82ee5defb2fc	78a5636f43172f60	84c87814a1f0ab7	8cc702081a6439ec
90befffa23631e28	a4506cebde82bde9	bef9a3f7b2c67915	c67178f2e372532b
ca273eceea26619c	d186b8c721c0c207	eada7dd6cde0eb1e	F57d4f7fee6ed178
06f067aa72176fba	0a637dc5a2c898a6	113f9804bef90dae	1b710b35131c471b
28db77f523047d84	32caab7b40c72493	3c9ebe0a15c9bebc	431d67c49c100d4c
4cc5d4becb3e42b6	597f299cfc657e2a	5fcb6fab3ad6faec	6c44198c4a475817

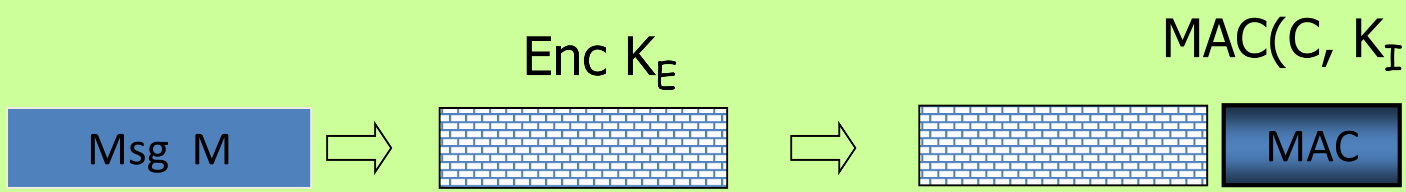
Combining MAC and Encryption

Encryption key K_E MAC key = K_I

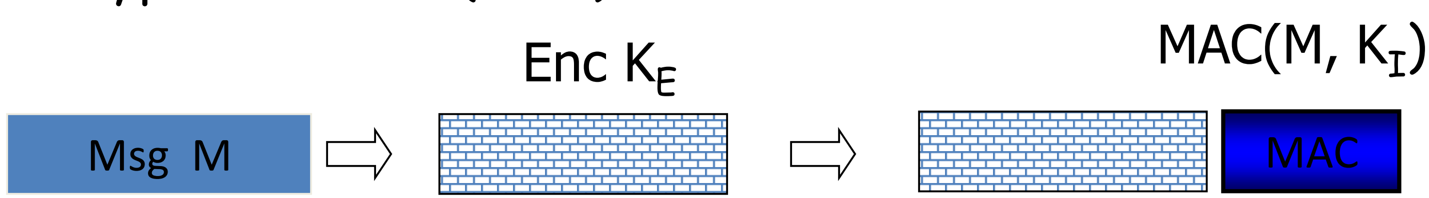
Option 1: MAC-then-Encrypt (SSL)



Option 2: Encrypt-then-MAC (IPsec)



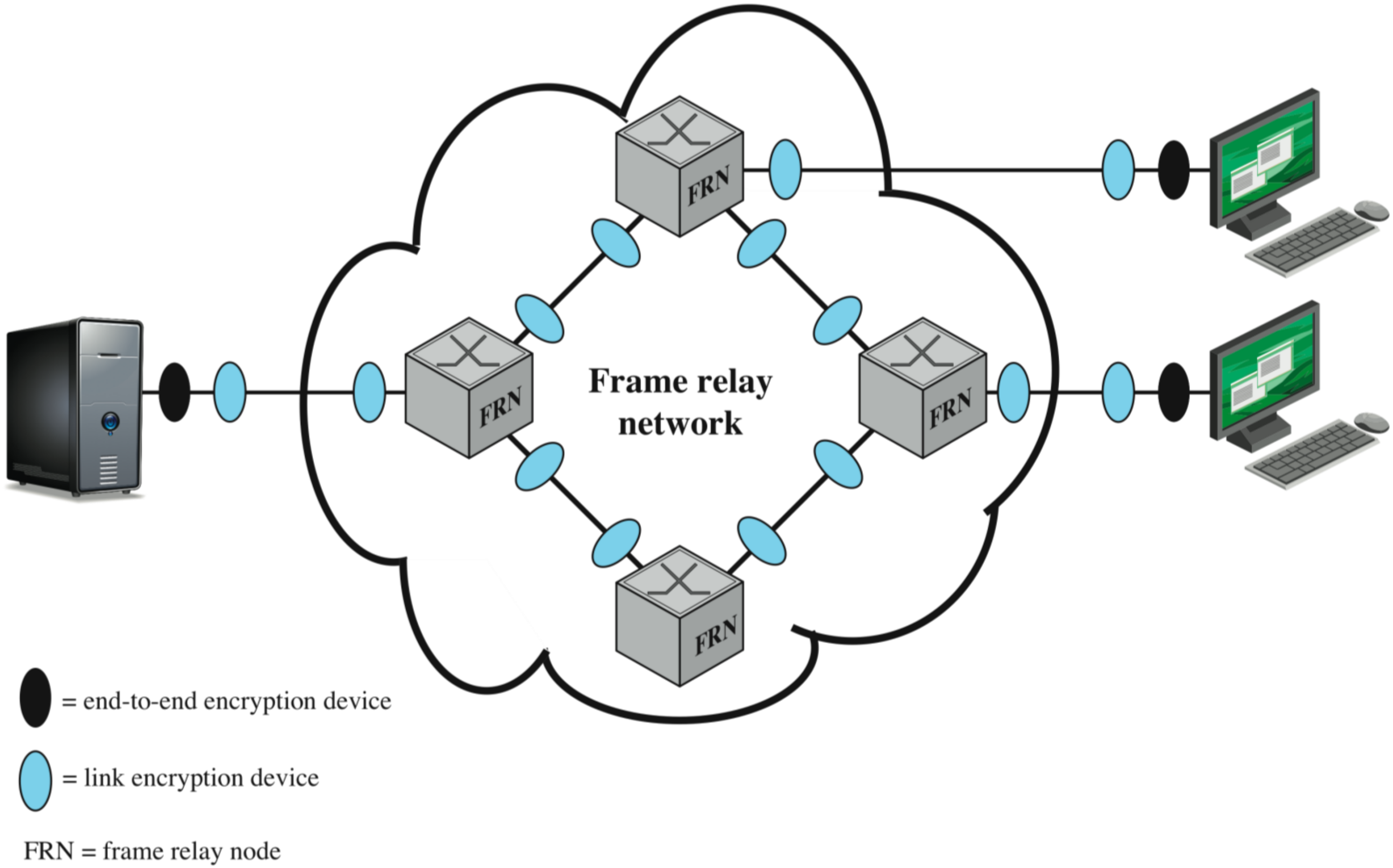
Option 3: Encrypt-and-MAC (SSH)



Applications of asymmetric cryptography

- Session set up
- Non-interactive applications (e.g. sending emails Email)
- Encrypting file systems
- Key escrow: data recovery

Location of Encryption



Location of Encryption

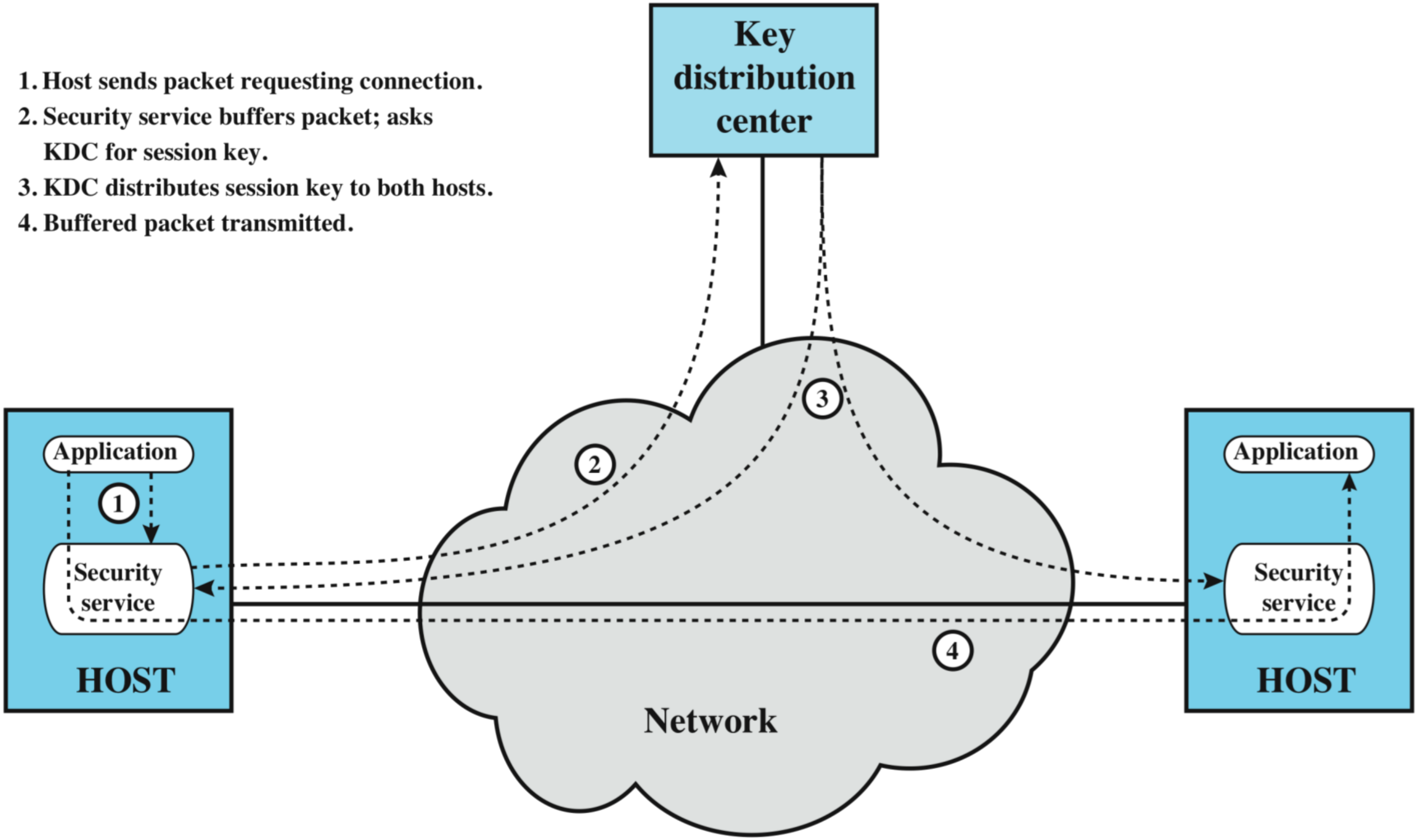
- Link Encryption
  - Every vulnerable link is equipped on both end with encryption device
  - Requires lots of encryption devices
  - The message and its header as well are encrypted thus must be decrypted in every switch – so the switch will know how to route it next
  - The message is vulnerable at every switch
- End-to-End Encryption
  - The message is encrypted only at end points
  - The header is not encrypted – allows the switch to rout it without decrypt
  - The header (traffic patterns) is not secured

Location of Encryption

- Solution: combine between Link and end-to-end encryption
  - Encrypt the message using end-to-end encryption
  - Then encrypt the encrypted messages + the header using the link Encryption
  - The entire message is secured, except the time that the header is decrypted and vulnerable at the switch's memory

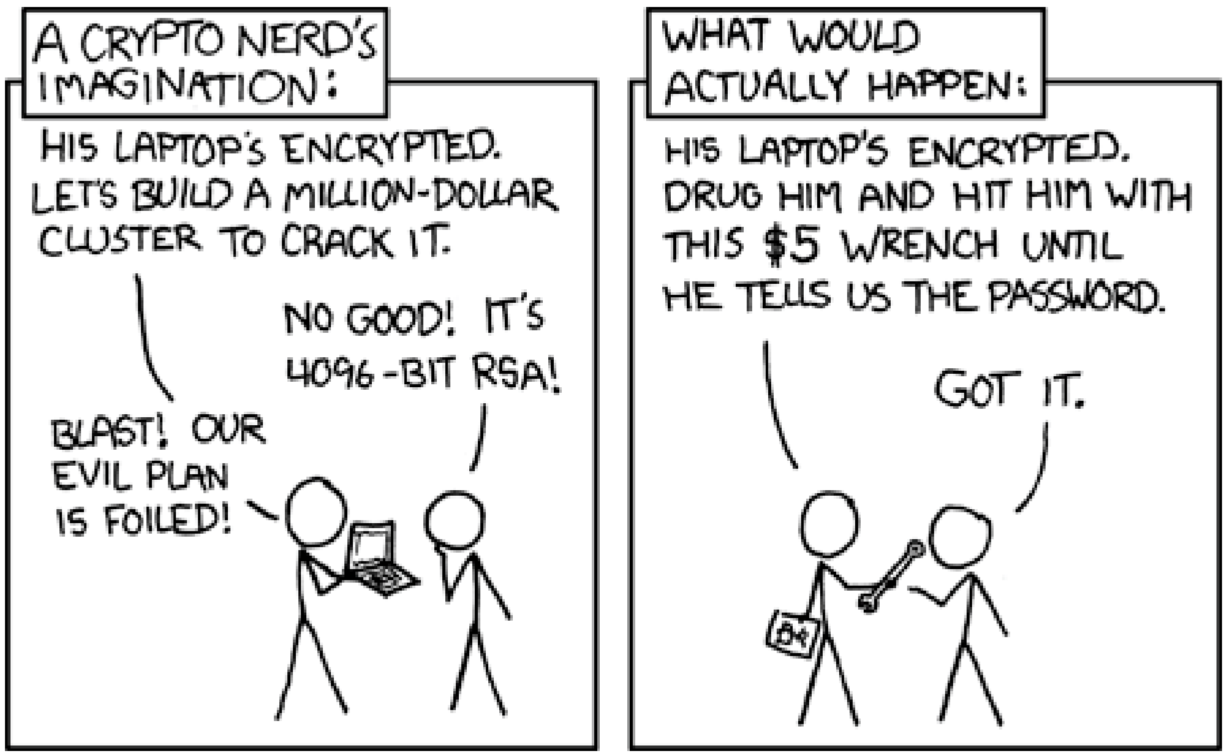
Key Distribution

1. Host sends packet requesting connection.
2. Security service buffers packet; asks KDC for session key.
3. KDC distributes session key to both hosts.
4. Buffered packet transmitted.



Limitations of cryptography

- People make other mistakes; crypto doesn't solve them
- Misuse of cryptography is fatal for security (e.g., WEP)



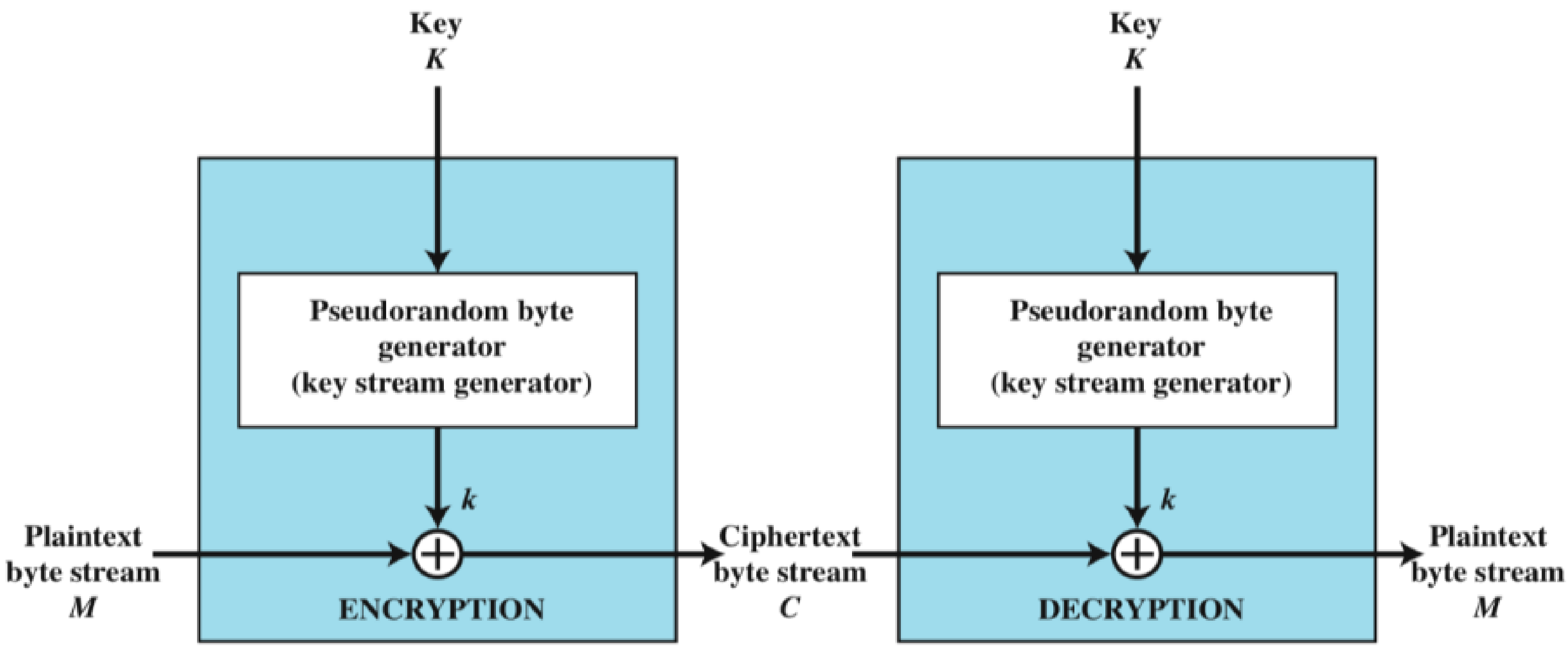
Stream Ciphers

- processes input elements continuously
- key input to a pseudorandom bit generator
  - produces stream of random like numbers using the key
  - unpredictable without knowing input key
  - XOR keystream output with plaintext bytes
- are faster and use far less code than Block-Cyphers

Stream Ciphers

- design considerations:
  - encryption sequence should have a large period - since it eventually repeats
  - keystream approximates random number properties 1s ~= 0s
  - uses a sufficiently long key to protect against brute force attack

Stream Ciphers



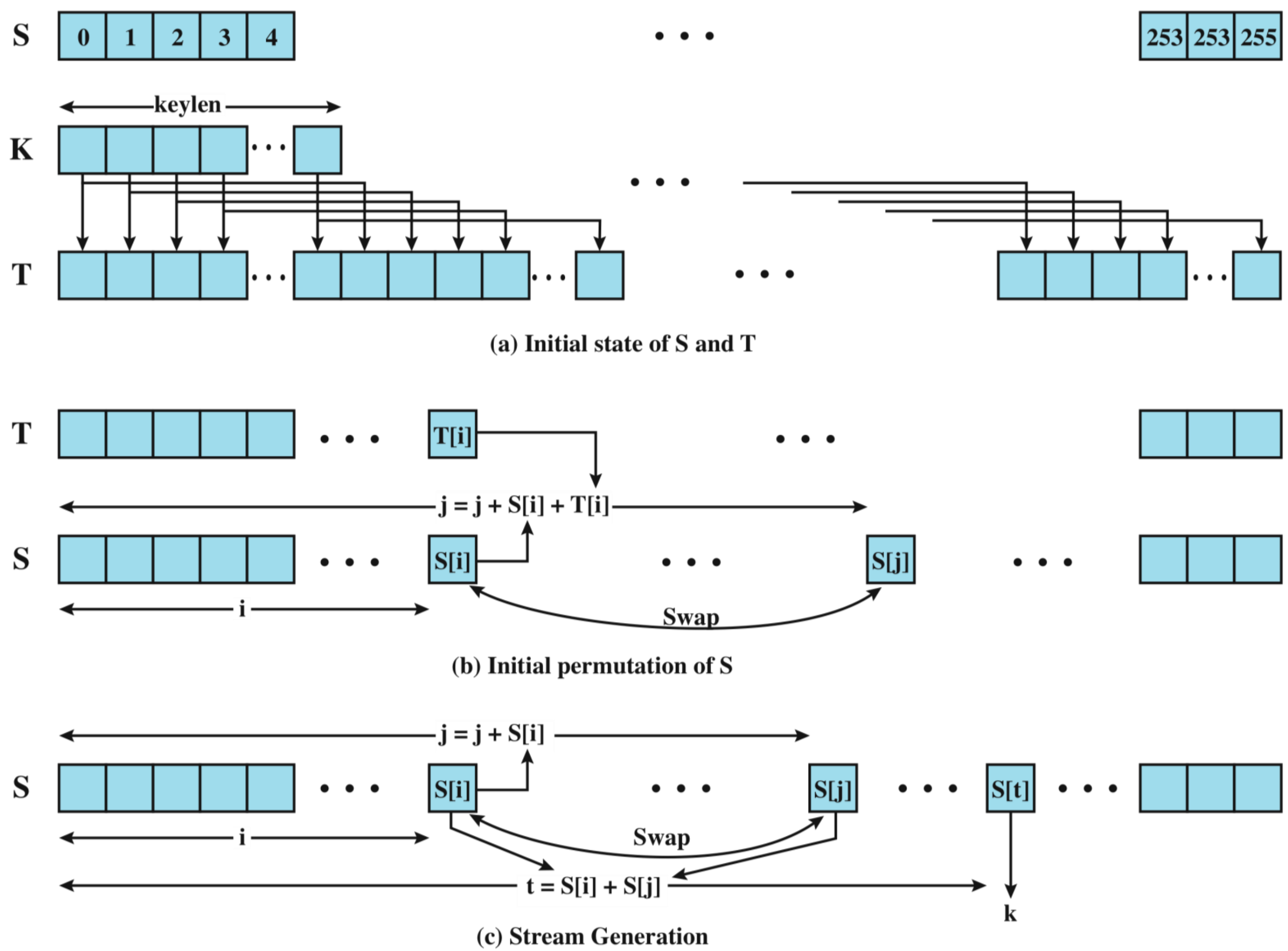
The RC4 Algorithm

- Designed in 1987 by Ron Rivest for RSA Security
- Stream cipher with byte-oriented operations
- Based on the use of a random permutation
- Can be expected to run very quickly in software
- Used in the SSL/TLS standards, WEP (Wired Equivalent Privacy) and WPA (WiFi Protected Access) protocol
- In September 1994 was anonymously posted on the Internet

RC4 Description

- Three main parts:
  - initialization of State Vector with the Symmetric Key
  - initial permutation = KSA (Key Scheduling Algorithm)
  - stream generation = PRGA (Pseudo Random Generation algorithm)
- Notation:
  - S = {0, 1, 2, … n-1} is the initial permutation
  - l = length of key

The RC4 Algorithm



RC4: Initialization of State Vector

- Two vectors of bytes:
  - $S[0],\ S[1],\ S[2],\ \ldots,\ S[255]$
  - $T[0],\ T[1],\ T[2],\ \ldots,\ T[255]$
- Key: variable length, from 1 to 256 bytes
- Initialization:

1. $S[i] \leftarrow i,\quad \text{for } 0 \le i \le 255$
2. $T[i] \leftarrow K[i \bmod \text{key-length}],\quad \text{for } 0 \le i \le 255$

   (i.e., fill up $T[0..255]$ with the key K repeatedly.)

RC4: Initial Permutation (KSA)

- Initial Permutation of S:

$$j \leftarrow 0$$

$$\text{for } \; i \; \leftarrow 0 \text{ to } 255 \text{ do}$$

$$j \; \leftarrow (j \; + \; S[i] \; + \; T[i] \,) \; \bmod \; 256$$

$$\text{Swap } \; S[i], \, S[\, j]$$

- This part of RC4 is generally known as the Key Scheduling Algorithm (KSA).
- After KSA, the input key and the temporary vector T will no longer be used.

RC4: Key Stream Generation

- Key stream generation:

$$
\begin{aligned}
& i, j \leftarrow 0 \\
& \text{while } (\text{true}) \\
& \qquad i \leftarrow (i + 1) \bmod 256 \\
& \qquad j \leftarrow (j + S[i]) \bmod 256 \\
& \qquad \text{Swap } S[i], S[j] \\
& \qquad t \leftarrow (S[i] + S[j]) \bmod 256 \\
& \qquad k \leftarrow S[t] \\
& \qquad \text{output } k
\end{aligned}
$$

RC4 Example

- Simple 4-byte example
- S = {0, 1, 2, 3}
- K = {1, 7, 1, 7}
- Set i = j = 0

KSA

- First Iteration (i = 0, j = 0, S = {0, 1, 2, 3}):
- j = (j + S[i] + K[i]) = (0 + 0 + 1) = 1 (1mod 4)
- Swap S[i] with S[j]: Swap S[0] with S[1]: S = {1, 0, 2, 3}

- Second Iteration (i = 1, j = 1, S = {1, 0, 2, 3}):
- j = (j + S[i] + K[i]) = (1 + 0 + 7) = 0 (8mod 4)
- Swap S[i] with S[j]: S = {0, 1, 2, 3}

- K = {1, 7, 1, 7}

KSA

Third Iteration ($i = 2, j = 0, S = \{0, 1, 2, 3\}$):
$j = (j + S[\,i\,] + K[\,i\,]) = (0 + 2 + 1) = 3$ (3mod 4)
Swap $S[\,i\,]$ with $S[\,j\,]$: $S = \{0, 1, 3, 2\}$

Fourth Iteration ($i = 3, j = 3, S = \{0, 1, 3, 2\}$):
$j = (j + S[\,i\,] + K[\,i\,]) = (3 + 2 + 7) = 0$ (12 mod 4)
Swap $S[\,i\,]$ with $S[\,j\,]$: $S = \{2, 1, 3, 0\}$

$K = \{1, 7, 1, 7\}$

PRGA (Pseudo Random Generation algorithm)

- Reset i = j = 0, Recall S = {2, 1, 3, 0}
- i = i + 1 = 1 (1 mod 4)
- j = j + S[i] = 0 + 1 = 1 (1 mod 4)
- Swap S[i] and S[j]: S = {2, 1, 3, 0}
- t= (S[i] + S[j]) mod 4 = 1+1=2 (2 mod 4)
- Output k = S[t] = S[2] = 3

The RC4 Algorithm

- Does not use IV (nonce)
- Same key on the same plaintext will result in the same cypher
- Weakness in the random number generator
- WEP was hacked in 2007

Risks in using stream ciphers

"Two time pad" is insecure:

$$\begin{cases} C_1 \leftarrow m_1 \oplus PRG(k) \\ C_2 \leftarrow m_2 \oplus PRG(k) \end{cases}$$

Eavesdropper does:

$$C_1 \oplus C_2 \quad \rightarrow \quad m_1 \oplus m_2$$

Enough redundant information in English that:

$$m_1 \oplus m_2 \rightarrow \quad m_1 , m_2$$

Risks in using stream ciphers

- **Short Cycle Length** key-streams generated by pseudorandom generators are cyclic. True random are unbreakable.
- **Correlation Attack** statistical analyses where parts of the contents of the two messages could be identified as equal → leads to the key, or parts of the key.

Risks in using stream ciphers

- **Substitution Attack** type of man-in-the-middle attack: In structure messages specific part my be substituted → cause confusion or misbehavior of the system even if the information is protected by a strong stream cipher.
- **Reused-Key Attack** Attack known from Wired Equivalent Privacy (WEP) : Example: long term key plus 24 bits changing as IV: Chance of finding reused key is high: Breaking the system in short time is likely.

Against which attacks protects symmetric and asymmetric encryption?

Basic Attacks on a telematics system

- Confidentiality (interception of the message)
- Integrity (Change of Content)
- Non repudiation of origin
- Non repudiation of receipt.
- Insertion (of a own message)
- Replay (of an intercepted message)
- Deletion (remove a message)
- Masquerade (deception on sending person)

What is the big difference between symmetric and asymmetric encryption?

What must a program do when a secure long message must be encrypted (assumed hybrid encryption: symmetric encryption, asymmetric key exchange), when sending to 3 partners? (please avoid multiple encryption of the document?

Against which attacks protects symmetric and asymmetric encryption?

Confidentiality only, with both of it

Basic Attacks on a telematics system

- ❑ Confidentiality (interception of the message)
- ❑ Integrity (Change of Content)
- ❑ Non repudiation of origin
- ❑ Non repudiation of receipt.
- ❑ Insertion (of a own message)
- ❑ Replay (of an intercepted message)
- ❑ Deletion (remove a message)
- ❑ Masquerade (deception on sending person)

What is the big difference between symmetric and asymmetric encryption?

Symmetric is thousand times faster and able to encrypt large messages
Asymmetric is “easier” for key management

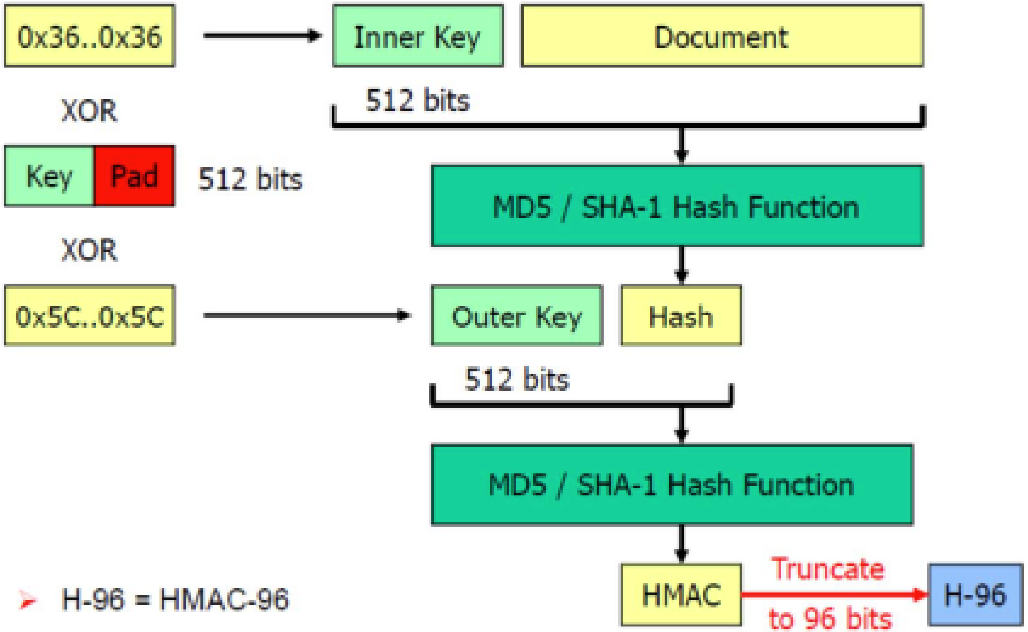
What must a program do when a secure long message must be encrypted (assumed hybrid encryption: symmetric encryption, asymmetric key exchange), when sending to 3 partners? (please avoid multiple encryption of the document?

The symmetric key must be encrypted three times!

Against which attacks protects this HMAC? (pre-shared and asymmetric key exchange)

Basic Attacks on a telematics system

- ❑ Confidentiality (interception of the message)
- ❑ Integrity (Change of Content)
- ❑ Non repudiation of origin
- ❑ Non repudiation of receipt.
- ❑ Insertion (of a own message)
- ❑ Replay (of an intercepted message)
- ❑ Deletion (remove a message)
- ❑ Masquerade (deception on sending person)

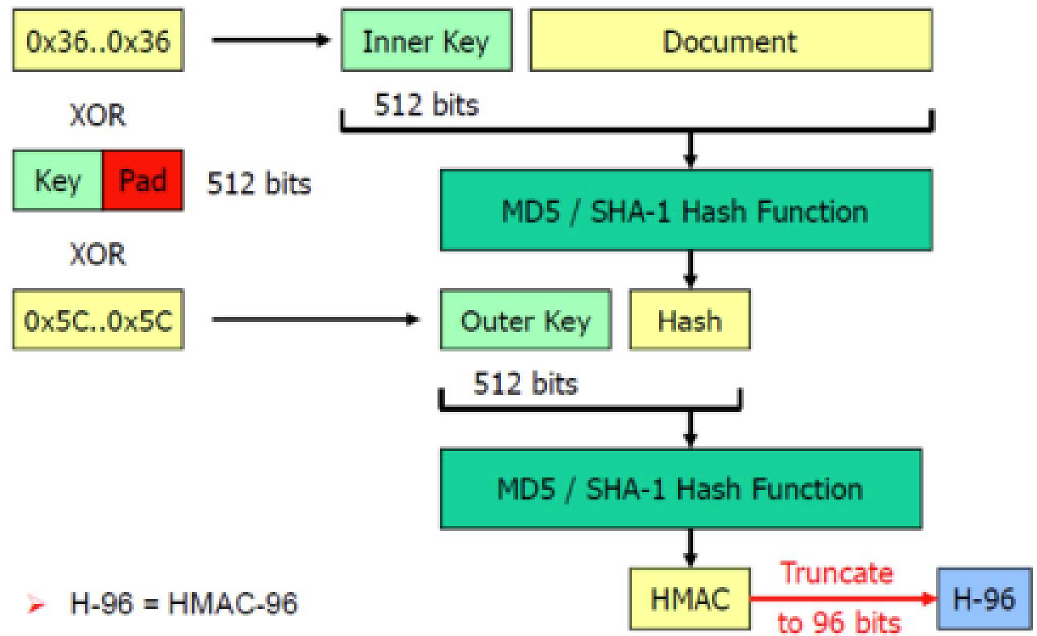


- H-96 = HMAC-96

Against which attacks protects this HMAC?
(pre-shared and asymmetric key exchange)

Basic Attacks on a telematics system

- ❑ Confidentiality (interception of the message)
- ❑ Integrity (Change of Content)
- ❑ Non repudiation of origin
- ❑ Non repudiation of receipt.
- ❑ Insertion (of a own message)
- ❑ Replay (of an intercepted message)
- ❑ Deletion (remove a message)
- ❑ Masquerade (deception on sending person)



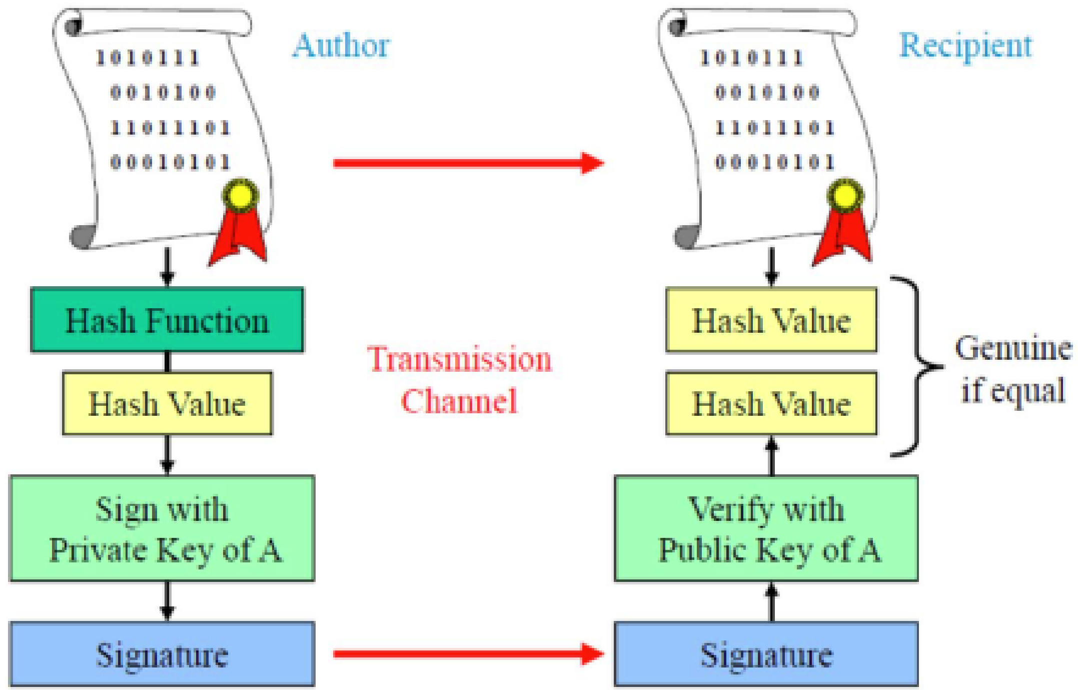
- H-96 = HMAC-96

Integrity!
If the key are exchanged asymmetrically, it remains with integrity, why?

Against which attacks protects this protocol? (pre-shared and asymmetric key exchange)

Basic Attacks on a telematics system

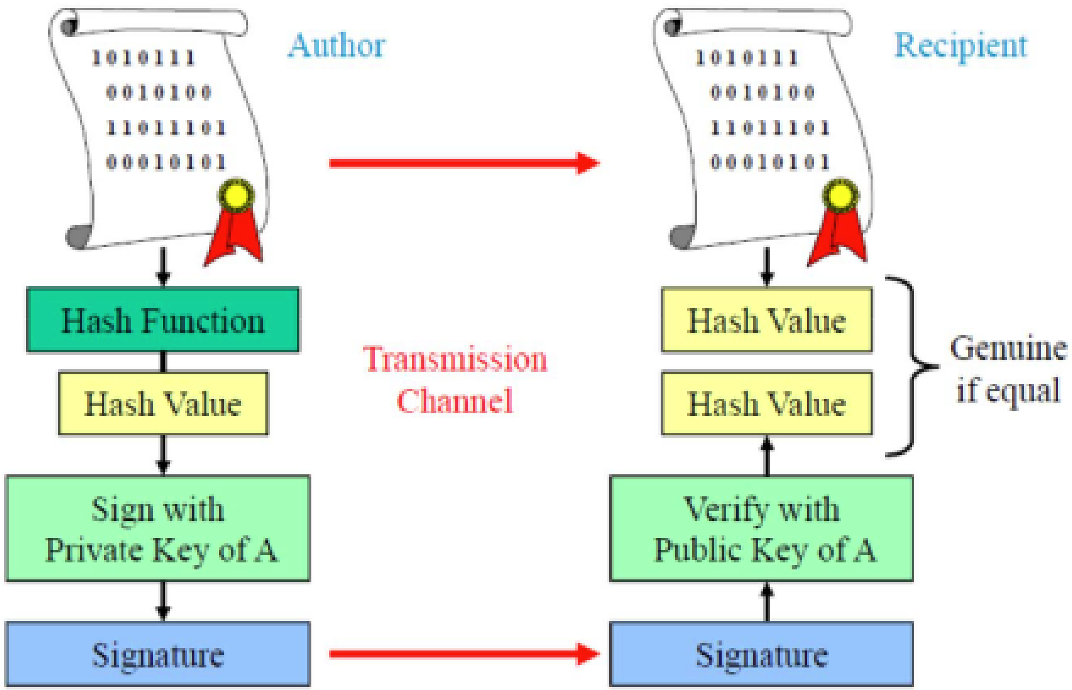
- ❑ Confidentiality (interception of the message)
- ❑ Integrity (Change of Content)
- ❑ Non repudiation of origin
- ❑ Non repudiation of receipt.
- ❑ Insertion (of a own message)
- ❑ Replay (of an intercepted message)
- ❑ Deletion (remove a message)
- ❑ Masquerade (deception on sending person)



Against which attacks protects this protocol? (pre-shared and asymmetric key exchange)

Basic Attacks on a telematics system

- ❑ Confidentiality (interception of the message)
- ❑ Integrity (Change of Content)
- ❑ Non repudiation of origin
- ❑ Non repudiation of receipt.
- ❑ Insertion (of a own message)
- ❑ Replay (of an intercepted message)
- ❑ Deletion (remove a message)
- ❑ Masquerade (deception on sending person)



Three attacks:
Integrity, insertion of an own message, non repudiation of author.

How to secure insertion, replay and deletion?

Basic Attacks on a telematics system

- ❑ Confidentiality (interception of the message)
- ❑ Integrity (Change of Content)
- ❑ Non repudiation of origin
- ❑ Non repudiation of receipt.
- ❑ Insertion (of a own message)
- ❑ Replay (of an intercepted message)
- ❑ Deletion (remove a message)
- ❑ Masquerade (deception on sending person)

Remaining attacks need secure protocols to defend against.

How to react on missing number, double number or out of sequence?

Attention: restart is an option to hide deletion! Industry examples.

How to secure insertion, replay and deletion?

Basic Attacks on a telematics system

- Confidentiality (interception of the message)
- Integrity (Change of Content)
- Non repudiation of origin
- Non repudiation of receipt.
- Insertion (of a own message)
- Replay (of an intercepted message)
- Deletion (remove a message)
- Masquerade (deception on sending person)

Remaining attacks need secure protocols to defend against.

With sequence numbers Insertion and replay is defended (MAC protected).
Replay can be defended by time/date stamps (MAC protected).

How to react on missing number, double number or out of sequence?

Deletion and replay will be detected with the next arriving message.
Out of sequence will require a restart.

Attention: restart is an option to hide deletion! Industry examples.