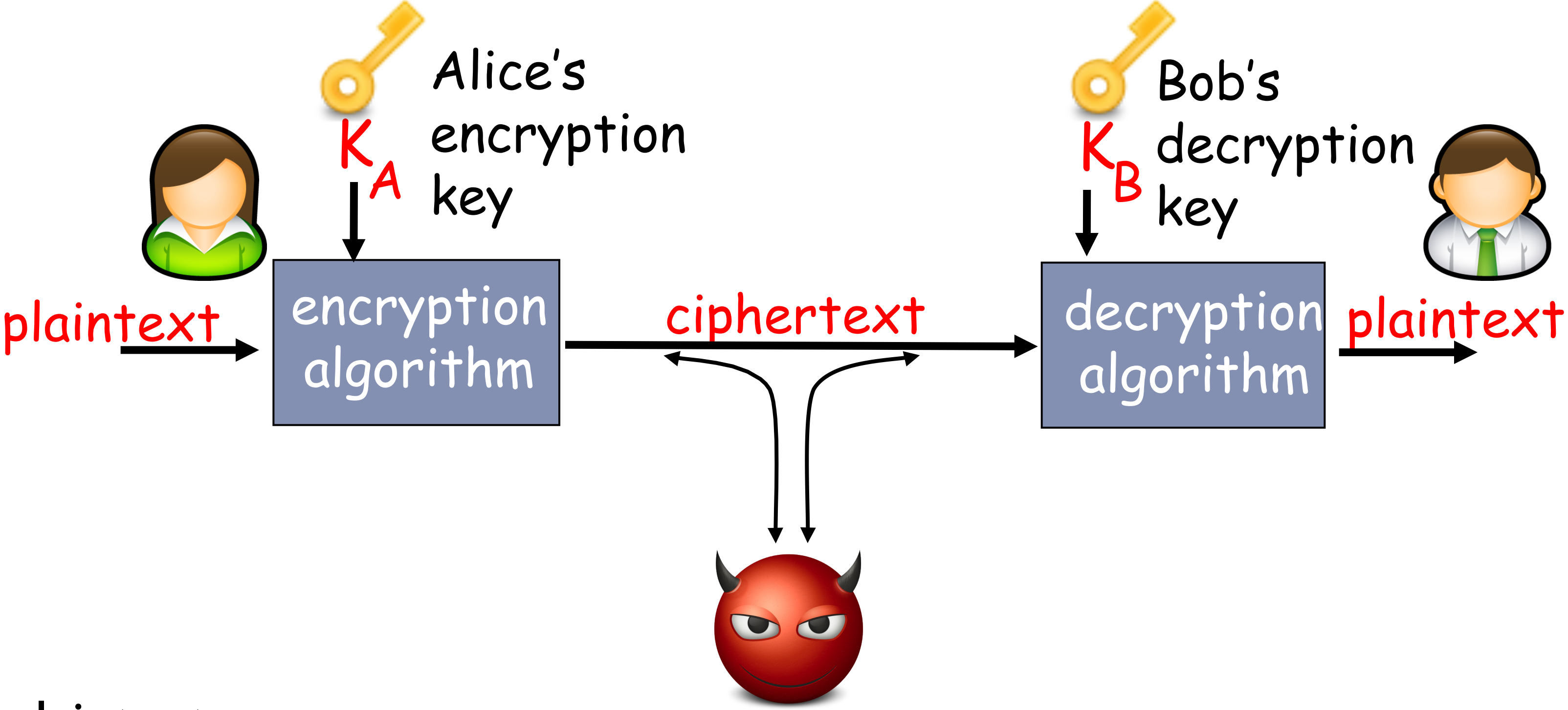
SYSTEM FAILURE
error 235553261...pending.....
fatal ER # 5444167QW32Z__wS e%$$

CSCI 4250/6250 – Fall 2015
Computer and Networks Security

INTRODUCTION TO CRYPTO
CHAPTER 8 (Goodrich)
CHAPTER 2-6 (Kaufman)
CHAPTER 8 (Kurose)

Slides adapted from Kurose et al., Goodrich et al., and Kaufman et al.

The language of cryptography



m plaintext message

$K_A(m)$ ciphertext, encrypted with key K_A

$m = K_B(K_A(m))$

Basics

- Alternative Notation
  - Secret key K
  - Encryption function $E_K(P)$
  - Decryption function $D_K(C)$
  - Plaintext length typically the same as ciphertext length
  - Encryption and decryption are permutation functions (bijections) on the set of all n-bit arrays
- Efficiency
  - functions E_K and D_K should have efficient algorithms
- Consistency
  - Decrypting the ciphertext yields the plaintext
  - $D_K(E_K(P)) = P$

Simple encryption scheme (Ceasar cipher)

substitution cipher: substituting one thing for another

- monoalphabetic cipher: substitute one letter for another

```
 plaintext:   abcdefghijklmnopqrstuvwxyz

ciphertext:   mnbvcxzasdfghjklpoiuytrewq
```

E.g.:

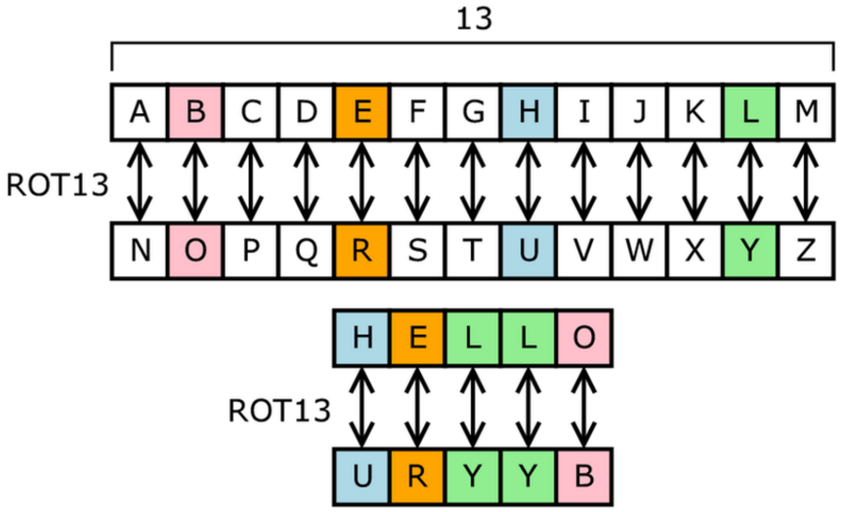
```
 Plaintext: bob. i love you. alice
ciphertext: nkn. s gktc wky. mgsbc
```

Key: the mapping from the set of 26 letters to the set of 26 letters

Substitution Ciphers

- Each letter is uniquely replaced by another.
- ROT13 examaple:
  - CIAO → PVNB

- One popular substitution "cipher" for some Internet posts is ROT13.



Public domain image from http://en.wikipedia.org/wiki/File:ROT13.png

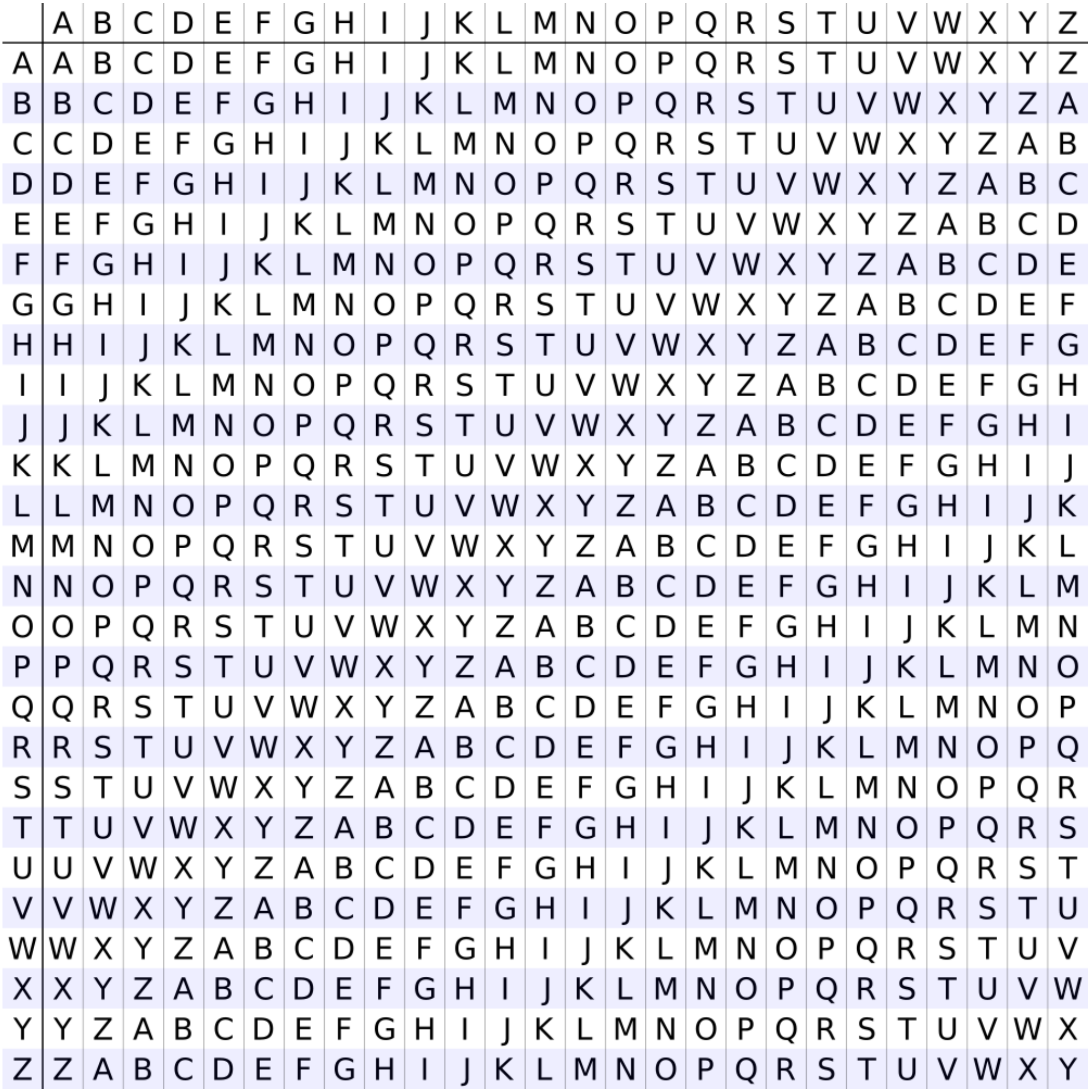
Polyalphabetic encryption

- n monoalphabetic cyphers, $M_1, M_2, \ldots, M_n$
- Cycling pattern:
  - e.g., n=4 $M_1,M_3,M_4,M_3,M_2; M_1,M_3,M_4,M_3,M_2;$
- For each new plaintext symbol, use subsequent monoalphabetic pattern in cyclic pattern
  - dog: d from M_1, o from M_3, g from M_4
- Key: the n ciphers and the cyclic pattern

- Example:
  - Vigenere cipher

Vigenere cipher

- Plaintext
  - ATTACKATDAWN
- Key
  - LEMON
- Keystream
  - LEMONLEMONLE…
- Ciphertext
  - LXFOPVEFRNHR



	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
A	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
B	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	A
C	C	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
D	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
E	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	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	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

Example from Wikipedia
http://en.wikipedia.org/wiki/Vigen%C3%A8re_cipher

Cryptography vs. Cryptanalysis

- Cryptographers invent new clever cryptographic schemes
    - Objective: make it infeasible to recover the plaintext
        - Computational difficulty: efficient to compute cipher-text, but hard to "reverse" without the key
- Cryptanalysis studies cryptographic schemes
    - Objective: try to find flaws in the schemes
        - E.g., recover some info about the plaintext, or recover the key
- **Fundamental Tenet of Cryptography**
    - "If lots of smart people have failed to solve a problem, then it probably won't be solved (soon)"

Breaking an encryption scheme

- Cipher-text only attack: Trudy has ciphertext that she can analyze
- Two approaches:
  - Search through all keys: must be able to differentiate resulting plaintext from gibberish
  - Statistical analysis

The crypto algorithms is typically public. Only thing that is assumed to be secret is the key.

- Known-plaintext attack: trudy has some plaintext corresponding to some ciphertext
  - eg, in monoalphabetic cipher, trudy determines pairings for a,l,i,c,e,b,o,b
- Chosen-plaintext attack: trudy can get the cyphertext for some chosen plaintext

Attacks

- Attacker may have

Harder

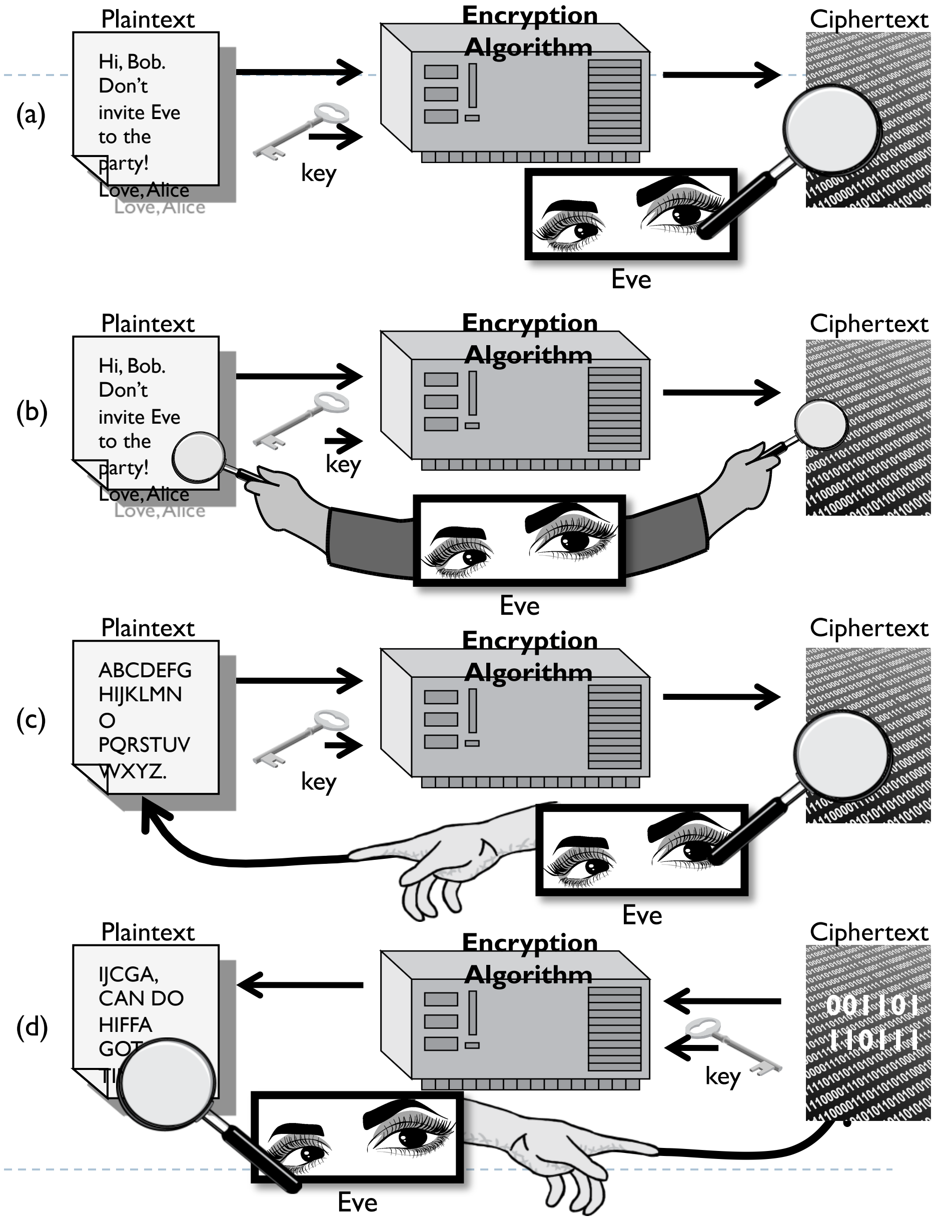
a) collection of ciphertexts (ciphertext only attack)

b) collection of plaintext/ ciphertext pairs (known plaintext attack)

c) collection of plaintext/ ciphertext pairs for plaintexts selected by the attacker (chosen plaintext attack)

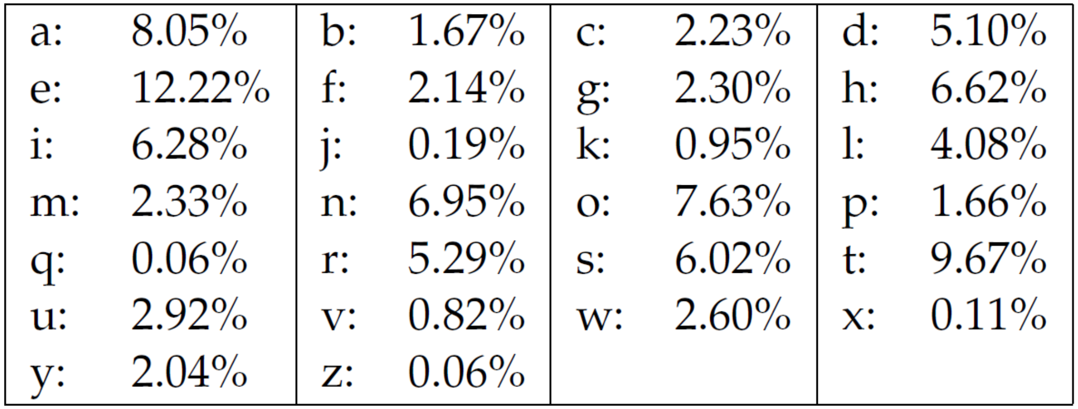
d) collection of plaintext/ ciphertext pairs for ciphertexts selected by the attacker (chosen ciphertext attack)

Easier



Frequency Analysis

- Letters in a natural language, like English, are not uniformly distributed.
- Knowledge of letter frequencies, including pairs and triples can be used in cryptologic attacks against substitution ciphers.



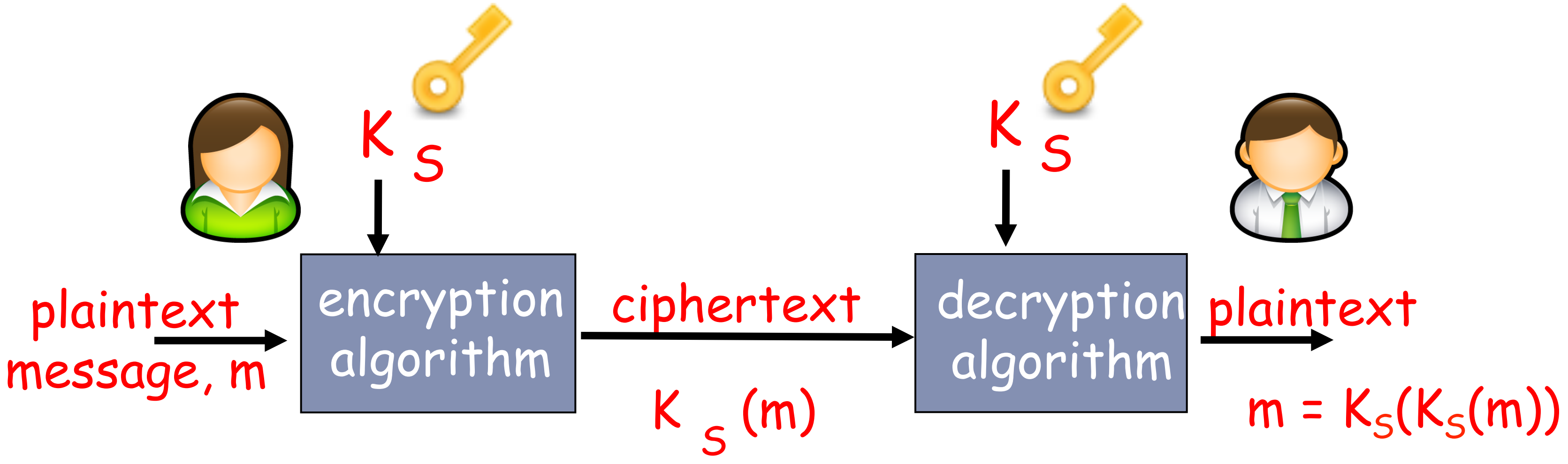
a:	8.05%	b:	1.67%	c:	2.23%	d:	5.10%
e:	12.22%	f:	2.14%	g:	2.30%	h:	6.62%
i:	6.28%	j:	0.19%	k:	0.95%	l:	4.08%
m:	2.33%	n:	6.95%	o:	7.63%	p:	1.66%
q:	0.06%	r:	5.29%	s:	6.02%	t:	9.67%
u:	2.92%	v:	0.82%	w:	2.60%	x:	0.11%
y:	2.04%	z:	0.06%				

8.1: Letter frequencies in the book *The Adventures of Tom Sawyer*, by Twain.

Types of Cryptography

- Crypto often uses keys:
  - Algorithm is known to everyone
  - Only "keys" are secret
- Public key cryptography
  - Involves the use of two keys
- Symmetric key cryptography
  - Involves the use of one key
- Hash functions
  - Involves the use of no keys
  - Nothing secret: How can this be useful?

Symmetric key cryptography



symmetric key crypto: Bob and Alice share same (symmetric) key: K_S

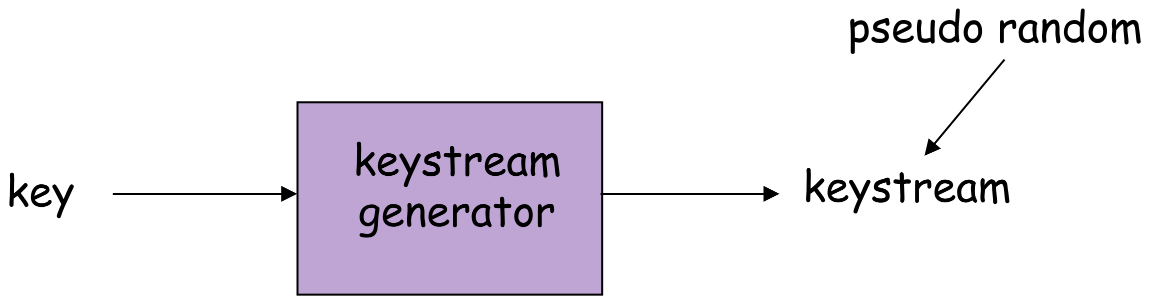
- e.g., key is knowing substitution pattern in mono alphabetic substitution cipher

Q: how do Bob and Alice agree on key value?

Two types of symmetric ciphers

- Stream ciphers
  - encrypt one bit at time
- Block ciphers
  - Break plaintext message in equal-size blocks
  - Encrypt each block as a unit

Stream Ciphers



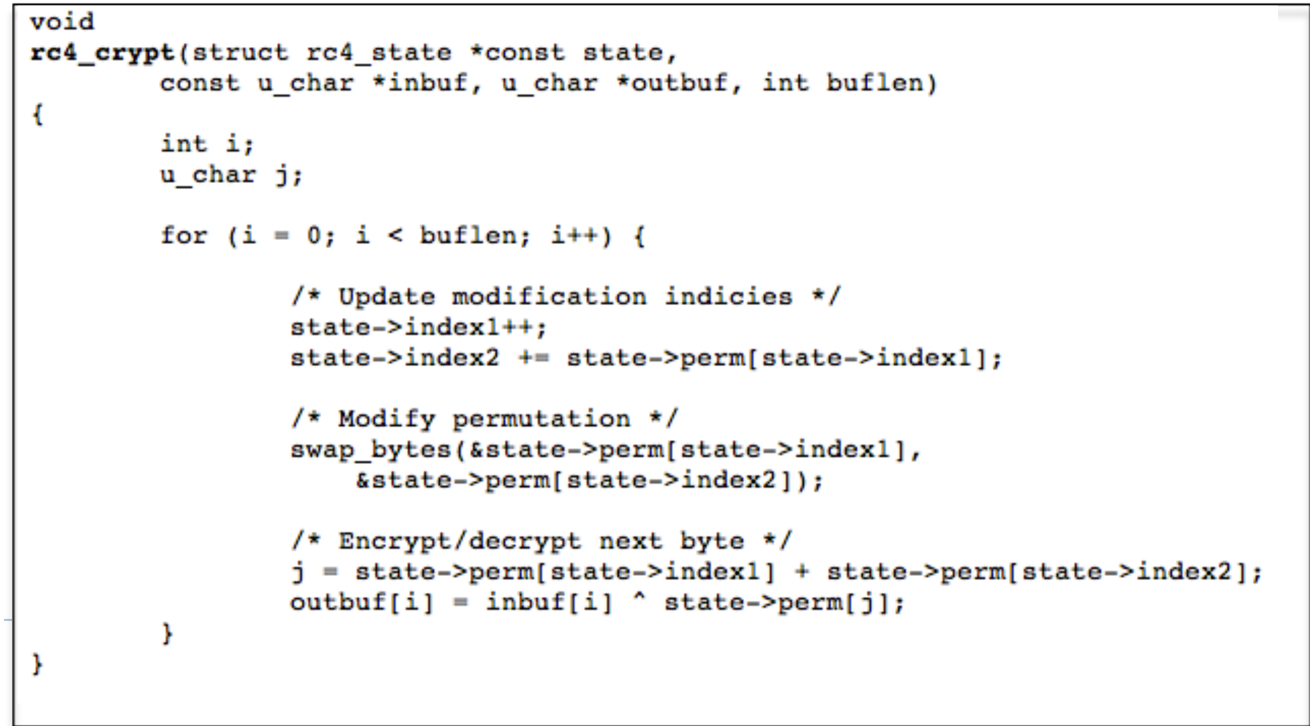
- Combine each bit of keystream with bit of plaintext to get bit of ciphertext
  - m(i) = ith bit of message
  - ks(i) = ith bit of keystream
  - c(i) = ith bit of ciphertext
  - c(i) = ks(i) ⊕ m(i)   (⊕ = exclusive or)
  - m(i) = ks(i) ⊕ c(i)
- Problem:
  - If attacker knows portion of plaintext P, she can replace it with desired malicious plaintext P’

RC4 Stream Cipher

- RC4 is a popular stream cipher
  - Extensively analyzed and considered good
  - Key can be from 1 to 256 bytes
  - Used in WEP for 802.11
  - Can be used in SSL

Use of RC4 in TLS is being phased out

https://tools.ietf.org/html/rfc7465



```
void
rc4_crypt(struct rc4_state *const state,
        const u_char *inbuf, u_char *outbuf, int buflen)
{
        int i;
        u_char j;

        for (i = 0; i < buflen; i++) {

                /* Update modification indicies */
                state->index1++;
                state->index2 += state->perm[state->index1];

                /* Modify permutation */
                swap_bytes(&state->perm[state->index1],
                    &state->perm[state->index2]);

                /* Encrypt/decrypt next byte */
                j = state->perm[state->index1] + state->perm[state->index2];
                outbuf[i] = inbuf[i] ^ state->perm[j];
        }
}
```

One-Time Pads

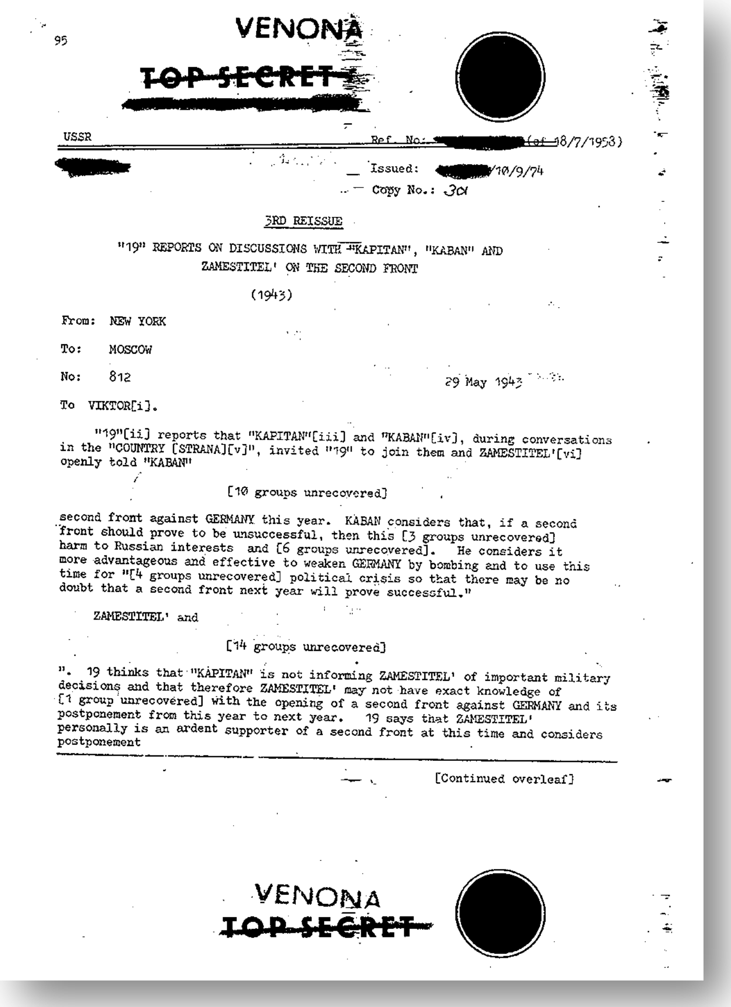
- There is one type of substitution cipher that is absolutely unbreakable.
  - The **one-time pad** was invented in 1917 by Joseph Mauborgne and Gilbert Vernam
  - We use a block of shift keys, $(k_1, k_2, \ldots, k_n)$, to encrypt a plaintext, M, of length n, with each shift key being chosen uniformly at random.
- Since each shift is random, every ciphertext is equally likely for any plaintext.

One-Time Pads

- Key is as long as the message to be sent
  - Stream of bits generated at random (not pseudo-random)
- Impossible to crack (perfect security?)
  - H(M) = H(M|C)
    - The ciphertext C provides no information about M
    - Given we only know C, every plaintext message is equally possible
  - Proven by Shannon
- Impractical
  - Keys need to be known to the receiver
  - Transferred through other means (e.g., paper)
  - Never reuse the same key

Weaknesses of the One-Time Pad

- In spite of their perfect security, one-time pads have some weaknesses
- The key has to be as long as the plaintext
- Keys can never be reused
  - Repeated use of one-time pads allowed the U.S. to break some of the communications of Soviet spies during the Cold War.



95

VENONA

~~TOP SECRET~~

[redacted]

[redacted]

USSR — Ref. No.: [redacted] ~~(of~~ 18/7/1953)

[redacted]

Issued: [redacted]/10/9/74

Copy No.: 30

3RD REISSUE

"19" REPORTS ON DISCUSSIONS WITH "KAPITAN", "KABAN" AND ZAMESTITEL' ON THE SECOND FRONT

(1943)

From: NEW YORK

To: MOSCOW

No: 812 — 29 May 1943

To VIKTOR[i].

"19"[ii] reports that "KAPITAN"[iii] and "KABAN"[iv], during conversations in the "COUNTRY [STRANA][v]", invited "19" to join them and ZAMESTITEL'[vi] openly told "KABAN"

[10 groups unrecovered]

second front against GERMANY this year. KABAN considers that, if a second front should prove to be unsuccessful, then this [3 groups unrecovered] harm to Russian interests and [6 groups unrecovered]. He considers it more advantageous and effective to weaken GERMANY by bombing and to use this time for "[4 groups unrecovered] political crisis so that there may be no doubt that a second front next year will prove successful."

ZAMESTITEL' and

[14 groups unrecovered]

". 19 thinks that "KAPITAN" is not informing ZAMESTITEL' of important military decisions and that therefore ZAMESTITEL' may not have exact knowledge of [1 group unrecovered] with the opening of a second front against GERMANY and its postponement from this year to next year. 19 says that ZAMESTITEL' personally is an ardent supporter of a second front at this time and considers postponement

[Continued overleaf]

VENONA

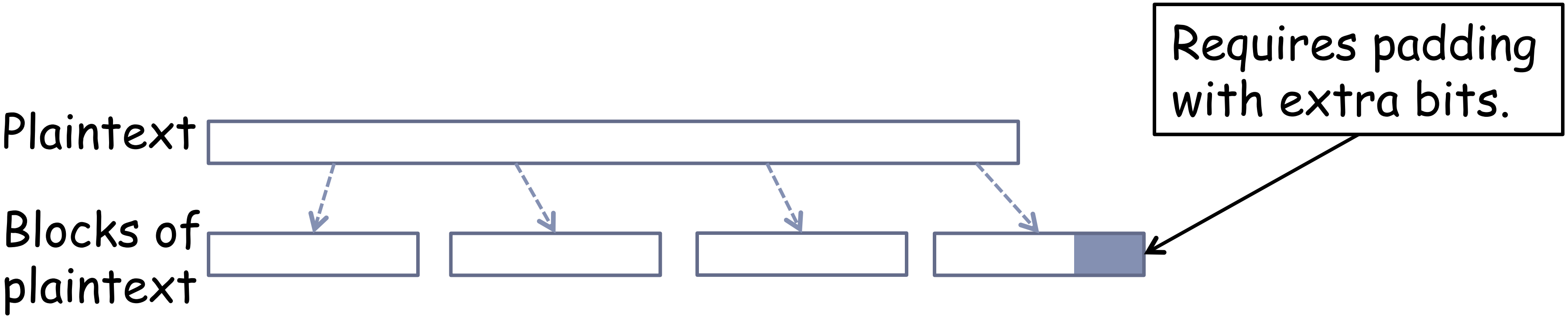
~~TOP SECRET~~

[redacted]

See graphical example at
https://www.khanacademy.org/computing/computer-science/cryptography/crypt/v/one-time-pad

Block Ciphers

- In a **block cipher:**
  - Plaintext and ciphertext have fixed length b (e.g., 128 bits)
  - A plaintext of length n is partitioned into a sequence of m **blocks**, P[0], …, P[m-1], where n ≤ bm < n + b
- Each message is divided into a sequence of blocks and encrypted or decrypted in terms of its blocks.



Padding

- Block ciphers require the length n of the plaintext to be a multiple of the block size b
- Padding the last block needs to be unambiguous (cannot just add zeroes)
- When the block size and plaintext length are a multiple of 8, a common padding method (PKCS#5) is a sequence of identical bytes, each indicating the length (in bytes) of the padding
- Example for b = 128 (16 bytes)
  - Plaintext: "Roberto" (7 bytes)
  - Padded plaintext: "Roberto999999999" (16 bytes), where 9 denotes the number and not the character
- We need to always pad the last block, which may consist only of padding (http://tools.ietf.org/html/rfc2898)

Block ciphers

- Message to be encrypted is processed in blocks of k bits (e.g., 64-bit blocks).
- 1-to-1 mapping is used to map k-bit block of plaintext to k-bit block of ciphertext

Example with k=3:

input	output
000	110
001	111
010	101
011	100

input	output
100	011
101	010
110	000
111	001

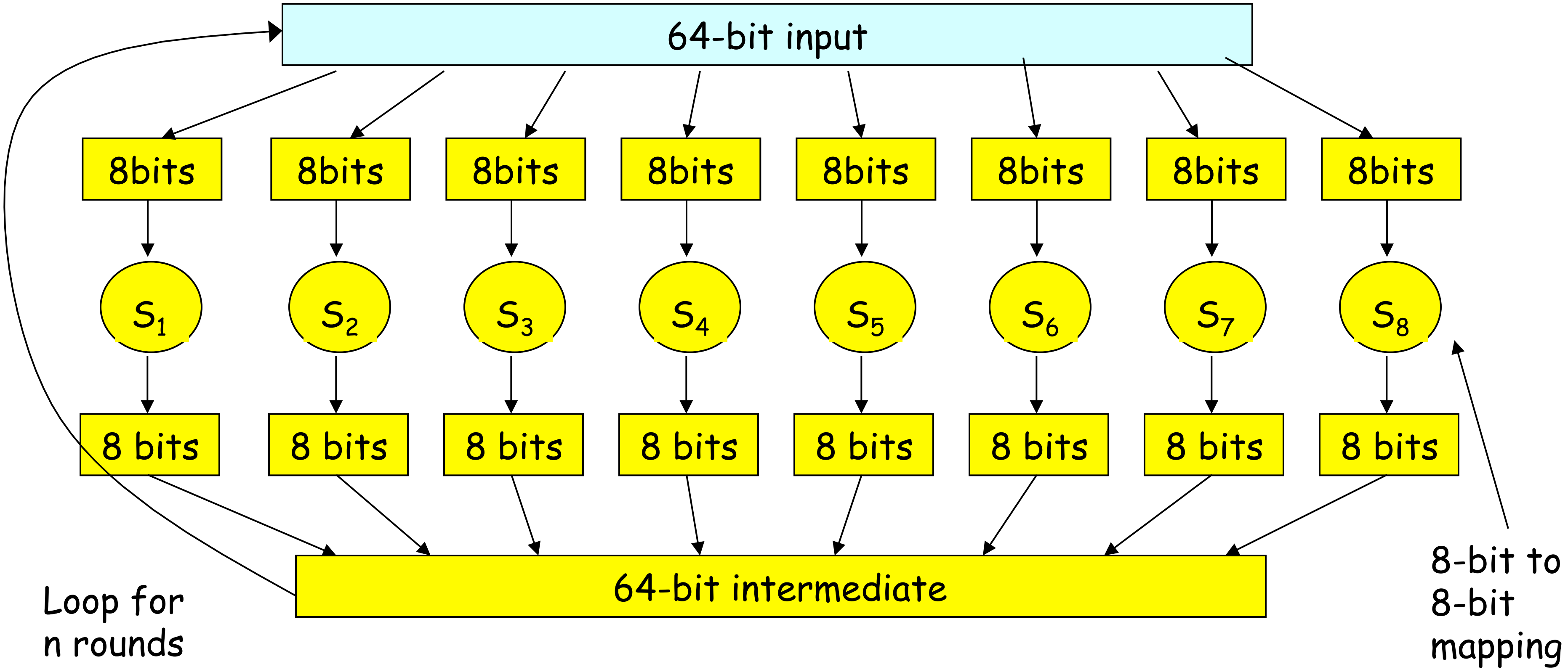
What is the ciphertext for 010110001111 ?

Block ciphers

- How many possible mappings are there for k=3?
  - How many 3-bit inputs?
  - How many permutations of the 3-bit inputs?
  - Answer: 40,320 ; not very many!
- In general, $2^k!$ mappings; huge for k=64
  - Hard to brute force!
- Storage Problem:
  - Table approach requires table with 2^{64} entries, each entry with 64 bits
  - It's like having a key that is 64 x 2^{64} bits long
- Table too big: instead use function that simulates a randomly permuted table

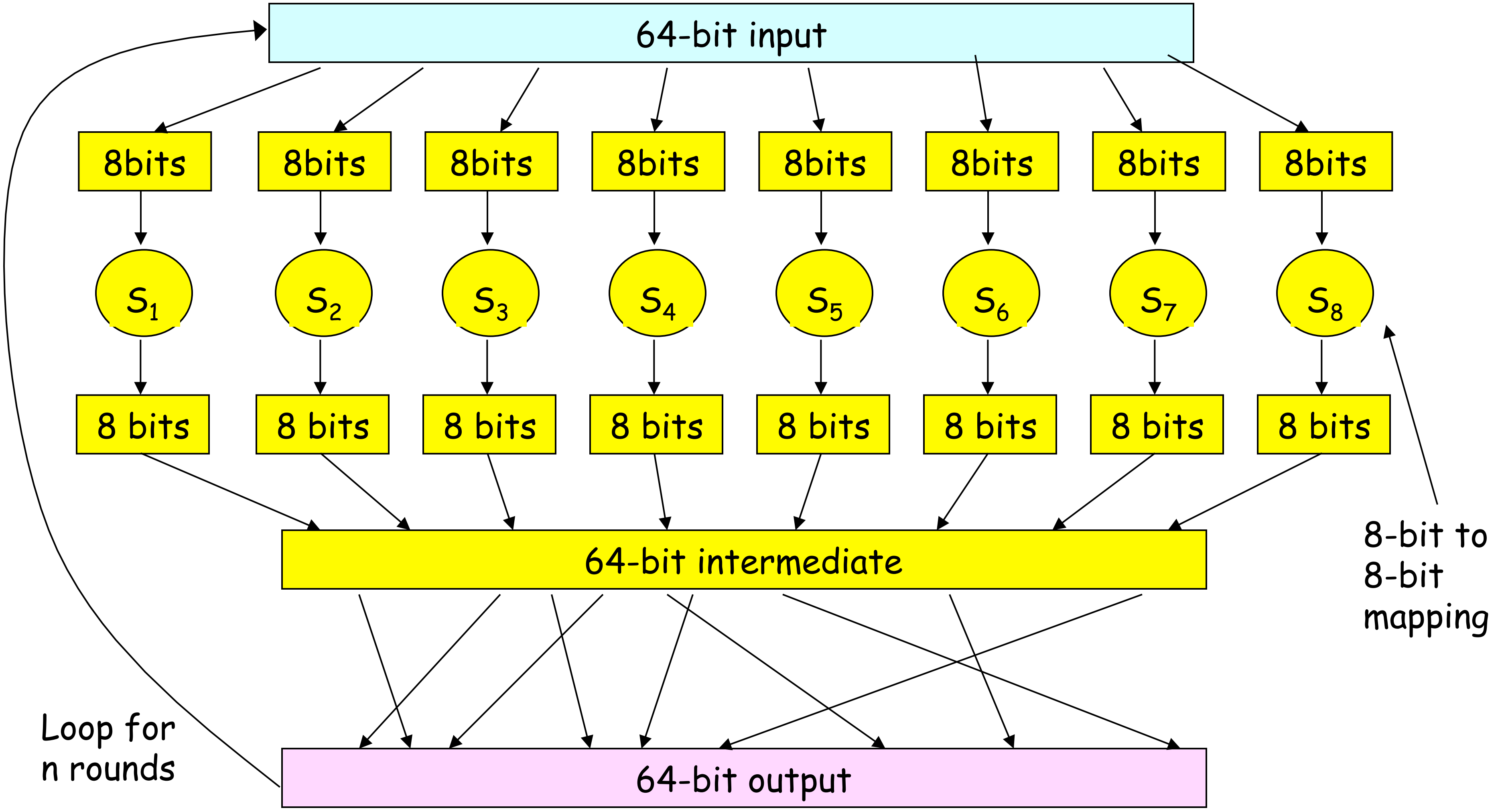
From Kaufman et al

Prototype function (Version 1)



From Kaufman et al

Prototype function (Version 2)



Why rounds?

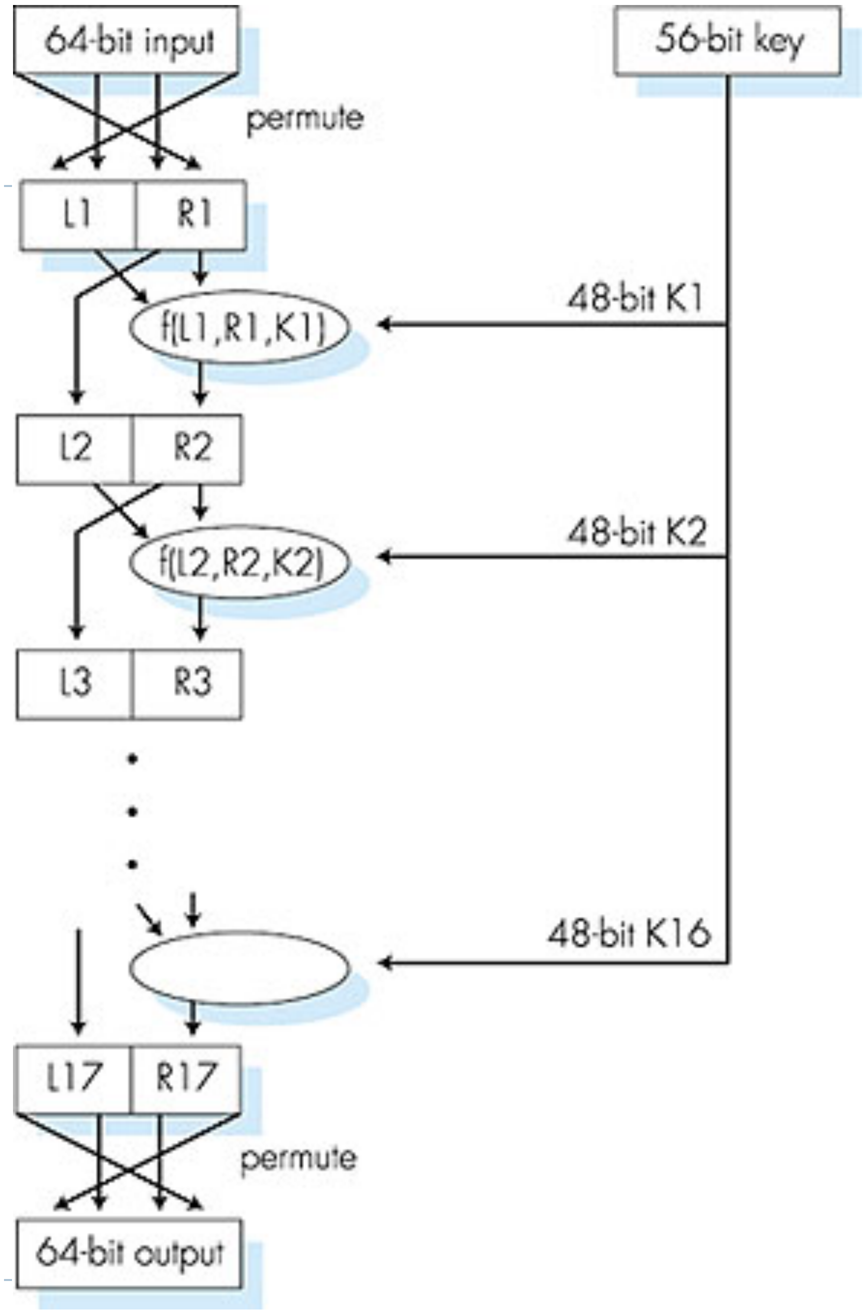
- If only a single round, then one bit of input affects at most 8 bits of output.
- In 2nd round, the 8 affected bits get scattered (via permutation) and inputted into multiple substitution boxes.
- How many rounds?
  - How many times do you need to shuffle cards
  - Becomes less efficient as n increases

Symmetric key crypto: DES

DES: Data Encryption Standard

- US encryption standard [NIST 1993]
- 56-bit symmetric key (64 – 8 parity bits)
- 64-bit plaintext input blocks
- Can be used in a cipher block chaining (CBC) setting to encrypt longer messages

Symmetric key crypto: DES



DES operation

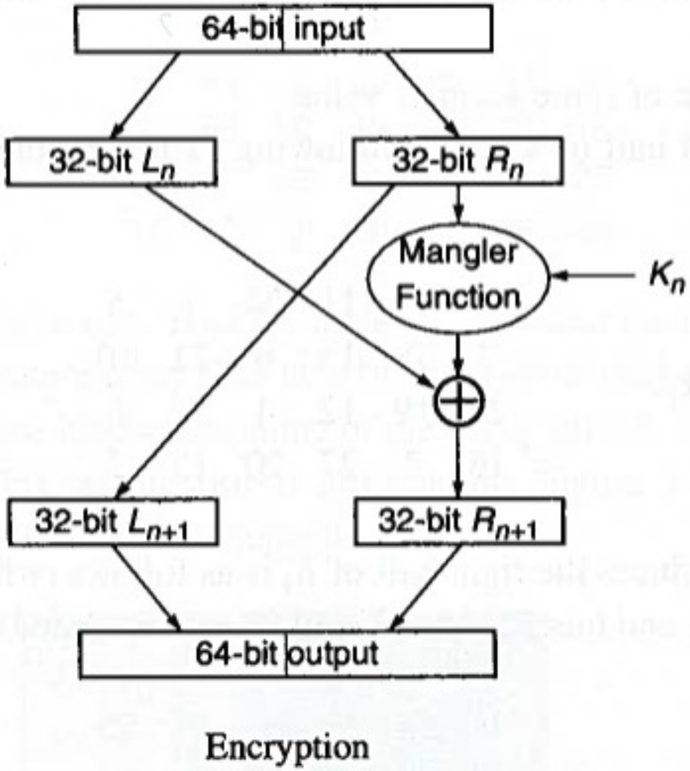
initial permutation

16 identical "rounds" of function application, each using different 48 bits of key

final permutation

DES Rounds

1-round Encryption and Decryption

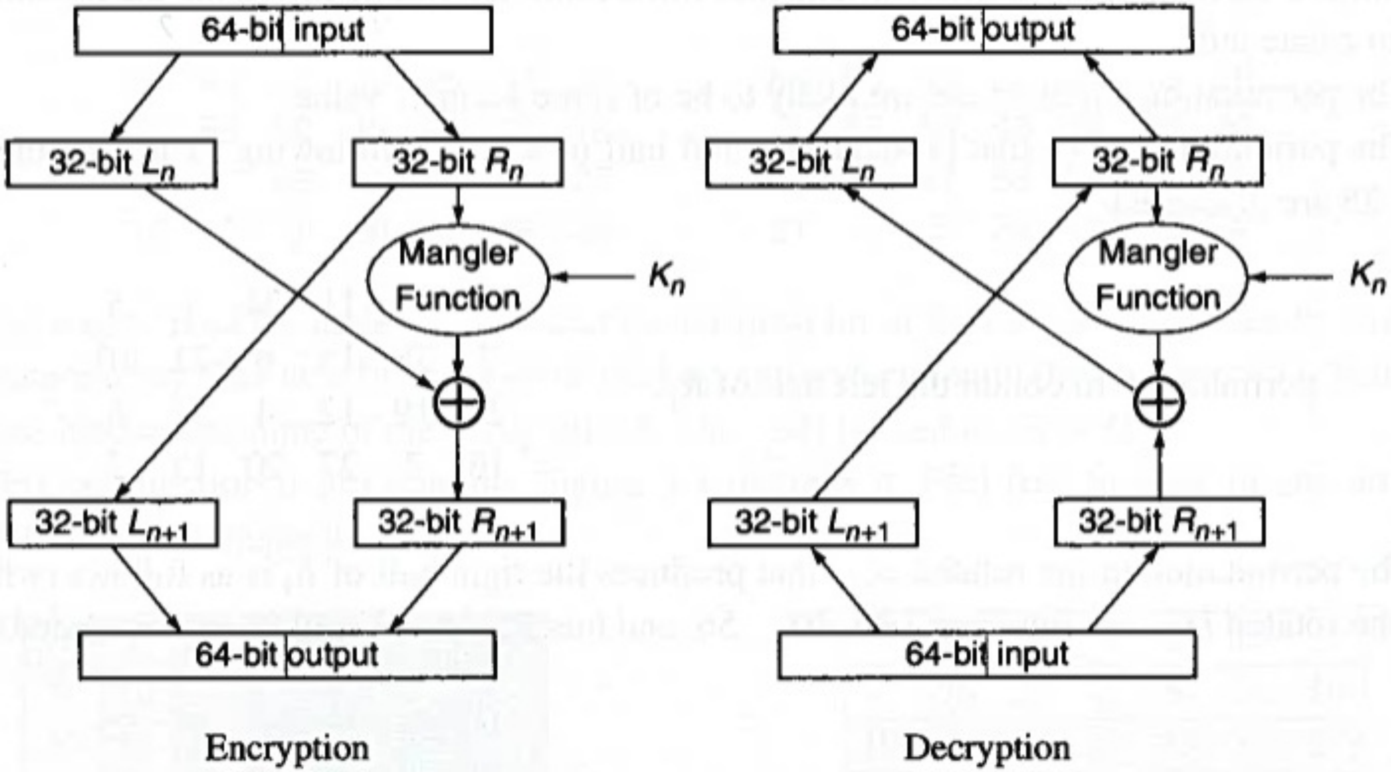


Encryption

See Kaufman et al. “Network Security, Private Communication in a Public World”

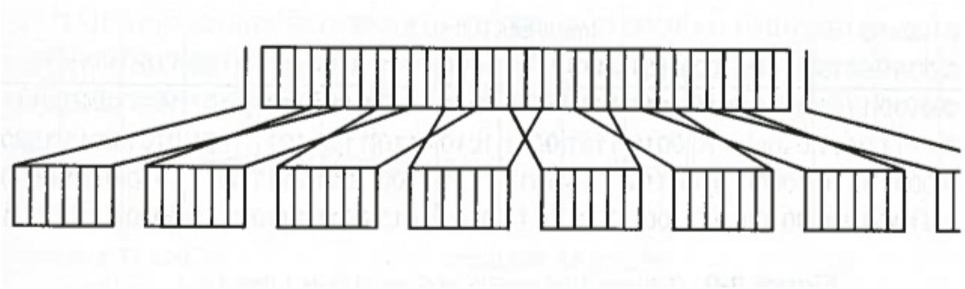
DES Rounds

1-round Encryption and Decryption

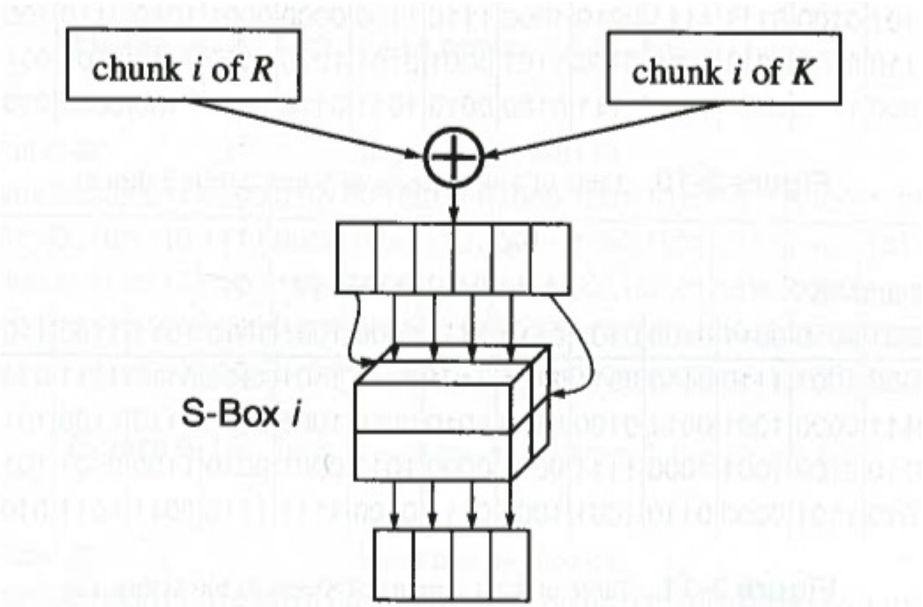


See Kaufman et al. “Network Security, Private Communication in a Public World”

DES Mangler Function



Expansion of R from 32 to 48 bits



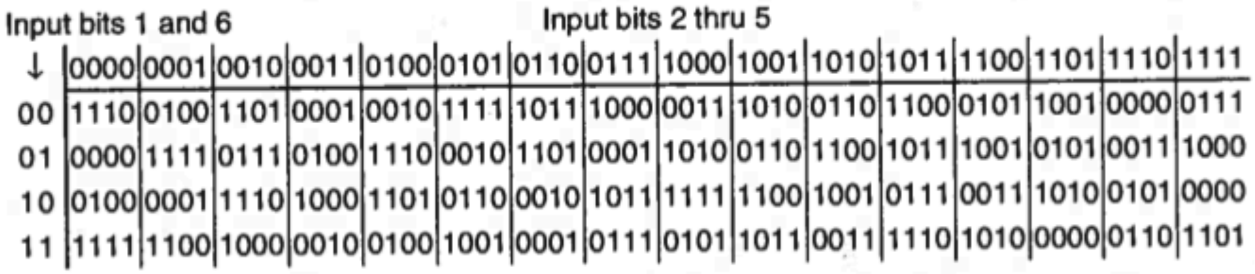
Expanded R and the Key are divided into eight 6-bit Chunks

Each 6-bit chunk is mapped into a 4-bit block

See Kaufman et al. "Network Security, Private Communication in a Public World"

How does the S-box look like?

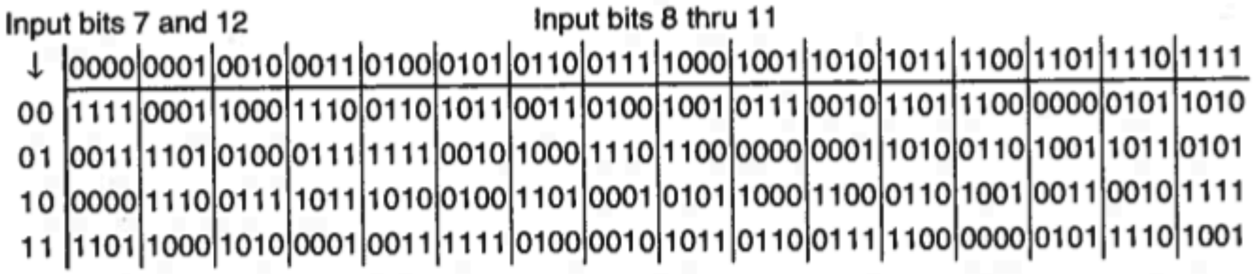
- There are 8 S-boxes (48/6)



Input bits 1 and 6 Input bits 2 thru 5

↓	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
00	1110	0100	1101	0001	0010	1111	1011	1000	0011	1010	0110	1100	0101	1001	0000	0111
01	0000	1111	0111	0100	1110	0010	1101	0001	1010	0110	1100	1011	1001	0101	0011	1000
10	0100	0001	1110	1000	1101	0110	0010	1011	1111	1100	1001	0111	0011	1010	0101	0000
11	1111	1100	1000	0010	0100	1001	0001	0111	0101	1011	0011	1110	1010	0000	0110	1101

Figure 3-9. Table of 4-bit outputs of S-box 1 (bits 1 thru 4)



Input bits 7 and 12 Input bits 8 thru 11

↓	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
00	1111	0001	1000	1110	0110	1011	0011	0100	1001	0111	0010	1101	1100	0000	0101	1010
01	0011	1101	0100	0111	1111	0010	1000	1110	1100	0000	0001	1010	0110	1001	1011	0101
10	0000	1110	0111	1011	1010	0100	1101	0001	0101	1000	1100	0110	1001	0011	0010	1111
11	1101	1000	1010	0001	0011	1111	0100	0010	1011	0110	0111	1100	0000	0101	1110	1001

Figure 3-10. Table of 4-bit outputs of S-box 2 (bits 5 thru 8)

Generating Per-Round Keys

- Start with 56-bit key (64 - 8 parity bits)
  - Why 56 bits? Unknown…
- First divide 56-bit key into two 28-bit chunks
- Rotate bits for 16 rounds…
  - Some rounds rotate only by one bit, others rotate by two bits

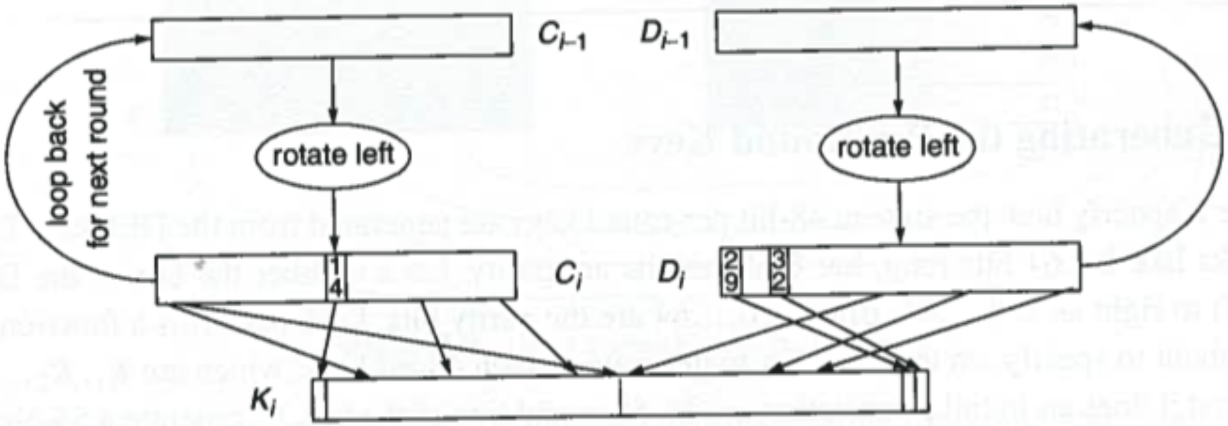
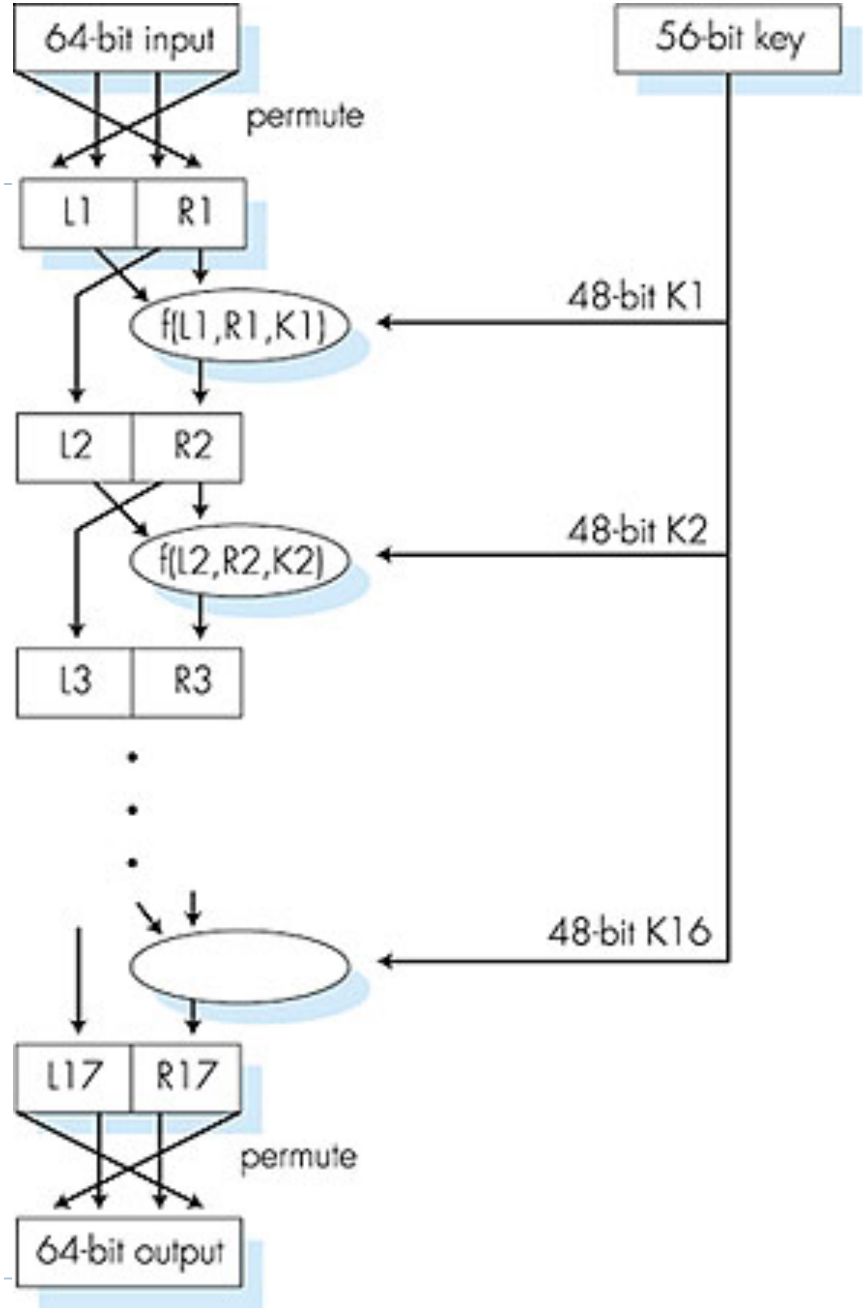


Figure 3-5. Round i for generating K_i

See Kaufman et al. "Network Security, Private Communication in a Public World"

Does DES work?



DES Security

- How secure is DES?
  - DES Challenge: 56-bit-key-encrypted phrase decrypted (brute force) in less than a day
  - No known good analytic attack
- making DES more secure:
  - 3DES: encrypt 3 times with 3 different keys (56*3=168 bits)

  (actually encrypt, decrypt, encrypt)
  - $c = K_c(K_b^{-1}(K_a(m)))$
  - $m = K_a^{-1}(K_b(K_c^{-1}(c)))$

3DES

- In practice only 2 keys are used
  - $c = K_a(K_b^{-1}(K_a(m)))$
  - $m = K_a^{-1}(K_b(K_a^{-1}(c)))$
  - It has been shown to be sufficiently secure
  - Avoids overhead of sending over 3 keys

- In DES we can *encrypt by decrypting* (???)
  - Using $c = K_a(K_b^{-1}(K_a(m)))$ allows for inter-operation with DES
  - Use Kb = Ka

- Why 3DES and not 120DES or 2DES?
  - 2DES has been proven not secure (takes only twice the time to brute-force a single-DES key)
  - 120DES would be very expensive from a computational point of view

NSA has some techniques in this area that we in the academic world do not. Certainly the fact that the NSA is pushing elliptic-curve cryptography is some indication that it can break them more easily.

The New York Times

WORLD | U.S. | N.Y. / REGION | BUS

POLITICS EDUCATION TEXAS

N.S.A. Foils Much Internet Encryption

By NICOLE PERLROTH, JEFF LARSON and SCOTT SHANE
Published: September 5, 2013 | 471 Comments

The National Security Agency is winning its long-running secret war on encryption, using supercomputers, technical trickery, court orders and behind-the-scenes persuasion to undermine the major tools protecting the privacy of everyday communications in the Internet age, according to newly disclosed documents.

Enlarge This Image

Associated Press

This undated photo released by the United States government shows the National Security Agency campus in

The agency has circumvented or cracked much of the encryption, or digital scrambling, that guards global commerce and banking systems, protects sensitive data like trade secrets and medical records, and automatically secures the e-mails, Web searches, Internet chats and phone calls of Americans and others around the world, the documents show.

FACEB
TWITTE
GOOG
SAVE
E-MAIL
SHARE
PRINT
SINGLE
REPRIN

Beginning in 2000, as encryption tools were gradually blanketing the Web, the N.S.A. invested billions of dollars in a clandestine campaign to preserve its ability to eavesdrop. Having lost a public battle in the 1990s to insert its own "back door" in all encryption, it set out to accomplish the same goal by stealth.

The agency, according to the documents and interviews with industry officials, deployed custom-built, superfast computers to break codes, and began collaborating with technology companies in the United States and abroad to build entry points into their products. The documents do not identify which companies have participated.

Simultaneously, the N.S.A. has been deliberately weakening the international encryption standards adopted by developers. One goal in the agency's 2013 budget request was to "influence policies, standards and specifications for commercial public key technologies," the most common encryption method.

Cryptographers have long suspected that the agency planted vulnerabilities in a standard adopted in 2006 by the National Institute of Standards and Technology and later by the International Organization for Standardization, which has 163 countries as members.

"For the past decade, N.S.A. has led an aggressive, multipronged effort to break widely used Internet encryption technologies," said a 2010 memo describing a briefing about N.S.A. accomplishments for employees of its British counterpart, Government Communications Headquarters, or GCHQ. "Cryptanalytic capabilities are now coming online. Vast amounts of encrypted Internet data which have up till now been discarded are now exploitable."

Crypto modes

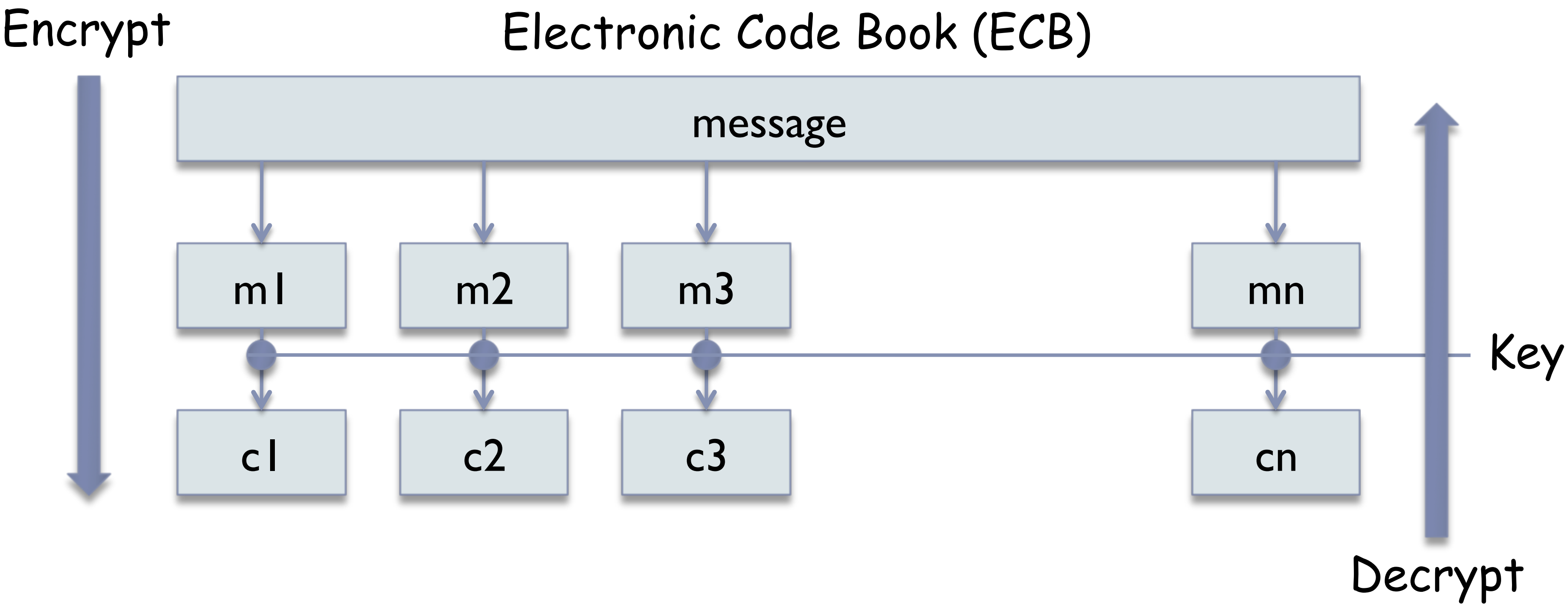
- Combining use of basic cipher for practical applications

- An application may need to
  - Be able to parallelize encryption and decryption
  - Preprocess as much as possible
  - Recover from bit errors/loss in the ciphertext
  - …

- Different modes provide different characteristics

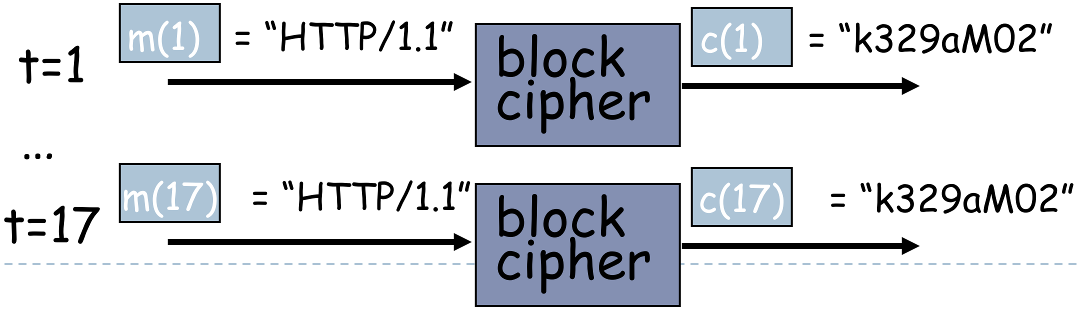
Encrypting a large message

- Why not just break message in 64-bit blocks, encrypt each block separately?



ECB

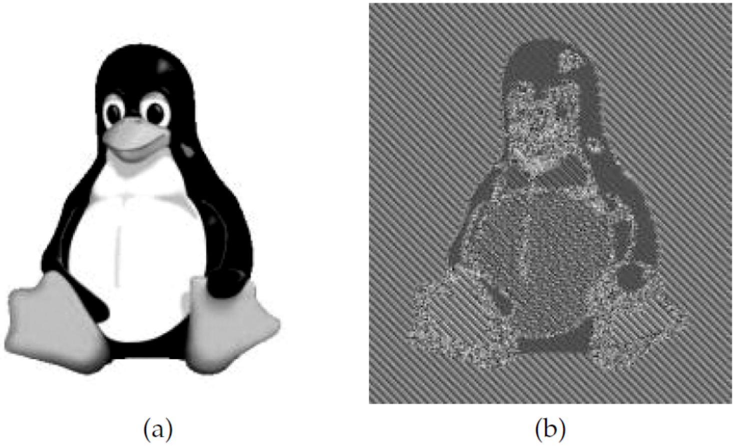
- Why not just break message in 64-bit blocks, encrypt each block separately?
  - The same plaintext always maps to the same ciphertext
    - in theory we can create a precomputed *code book* (one per key!)
  - Would be useful for random access files
    - ecryption and decryption trivially parallelizable
  - If same block of plaintext appears twice, will give same ciphertext
  - May facilitate cryptanalysis
    - Multiple messages that start with the same structure will give attacker a number of plaintext-ciphertext pairs to work with
  - we could swap things (e.g., swap salaries)



Strengths and Weaknesses of ECB

- Strengths:
  - Is very simple
  - Allows for parallel encryptions of the blocks of a plaintext
  - Can tolerate the loss or damage of a block

- Weakness:
  - Documents and images are not suitable for ECB encryption since patters in the plaintext are repeated in the ciphertext:


(a)

(b)

Figure 8.6: How ECB mode can leave identifiable patterns in a sequence of blocks: (a) An image of Tux the penguin, the Linux mascot. (b) An encryption of the Tux image using ECB mode. (The image in (a) is by Larry Ewing, lewing@isc.tamu.edu, using The Gimp; the image in (b) is by Dr. Juzam. Both are used with permission via attribution.)

Weaknesses of ECB

- Example: Assume attacker knows a block of plaintext and wants to modify or replace it

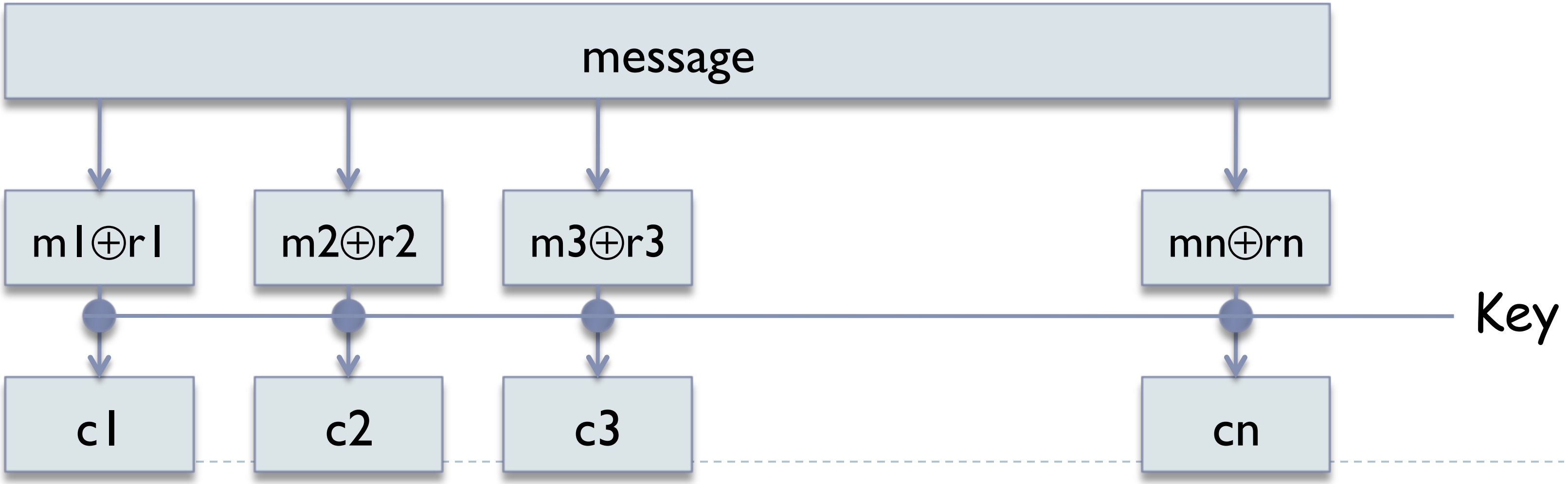
Jack Webb	$51,000	Jim Cook	$12,000
C1	C2	C3	C4

Jack Webb	$51,000	Jim Cook	$51,000
C1	C2	C3	C2

Encrypting a large message

- How about:
  - Generate random 64-bit number r(i) for each plaintext block m(i)
  - Calculate $c(i) = K_S(m(i) \oplus r(i))$
  - Transmit c(i), r(i), i=1,2,…
  - At receiver: $m(i) = K_S(c(i)) \oplus r(i)$
  - Problems:
    - inefficient, **need to send c(i) and r(i)**

Electronic Code Book (ECB)

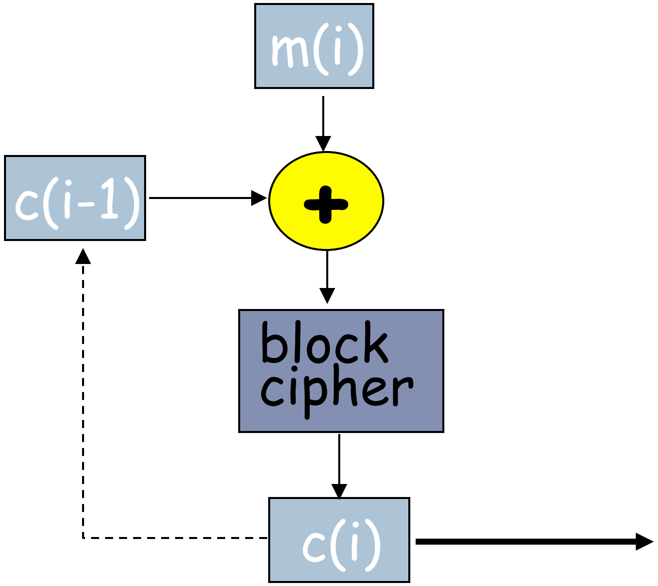


Cipher Block Chaining (CBC)

- CBC generates its own random numbers
  - Have encryption of current block depend on result of previous block
  - $c(i) = K_S(m(i) \oplus c(i-1))$
  - $m(i) = K_S(c(i)) \oplus c(i-1)$
- Forces same plaintext blocks to produce different ciphertext
- How do we encrypt first block?
  - Initialization vector (IV): random block = c(0)
  - IV does not have to be secret
- Change IV for each message (or session)
  - Guarantees that even if the same message is sent repeatedly, the ciphertext will be completely different each time

Cipher Block Chaining

- *cipher block chaining:* XOR ith input block, m(i), with previous block of cipher text, c(i-1)
  - c(0) transmitted to receiver in clear
  - what happens in "HTTP/1.1" scenario from above?



CBC

CBC Encryption

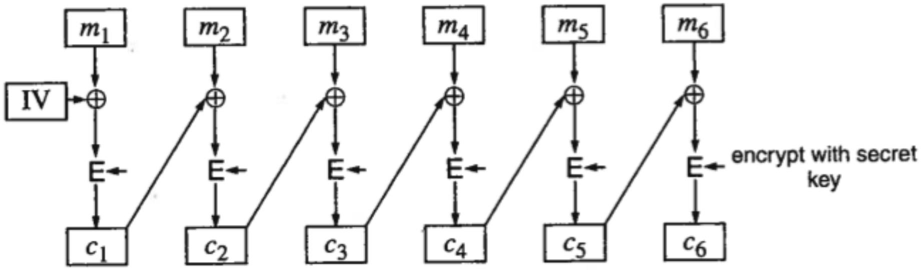


Figure 4-5. Cipher Block Chaining Encryption

See Kaufman et al. “Network Security, Private Communication in a Public World”

CBC

CBC Encryption

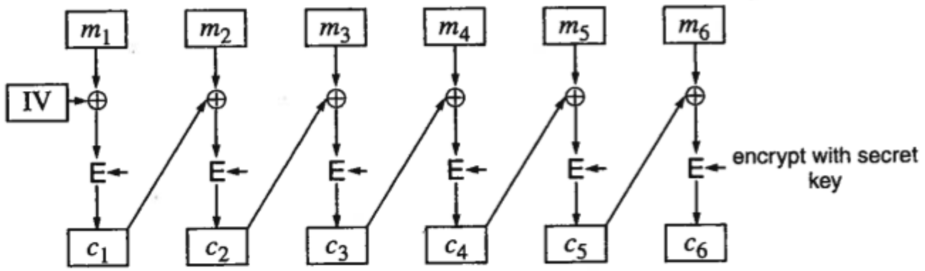


Figure 4-5. Cipher Block Chaining Encryption

CBC Decryption

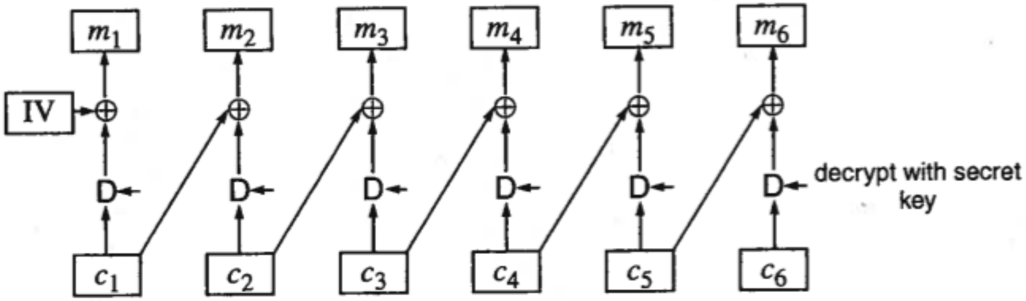


Figure 4-6. Cipher Block Chaining Decryption

See Kaufman et al. "Network Security, Private Communication in a Public World"

CBC: Threats

- CBC does not eliminate the possibility of somebody modifying the message *in transit*
- The attacker cannot swap blocks (e.g., to replace the IT guy's salary with the CEO salary), but can modify the ciphertext

- Example: Assume attacker knows a block of plaintext and wants to modify it

Jack Webb	IT Department	$51,000
Ci-1	Ci	Ci+1

- Changing Ci will modify M(i+1) in a predictable way
- However, Mi will be most likely garbled
  - The change may be noticeable or not, the attacker may decide to take his chances
- We may also need to "protect" IV, to avoid predictable changes to M1
  - Example: IV = timestamp; send E_k(IV)
- One possible defense
  - Attach one checksum block to the plaintext before encrypting
  - Changes in the plaintext will be detected with high probability

Strengths and Weaknesses of CBC

- Strengths:
  - Doesn't show patterns in the plaintext
  - Is the most common mode
  - Is fast and relatively simple

- Weaknesses:
  - CBC requires the reliable transmission of all the blocks sequentially
  - CBC is not suitable for applications that allow packet losses (e.g., music and video streaming)
  - Existence of Threats

Output Feedback Mode

- Use Block Cipher to generate key-stream (ks)
- $K(IV) = [b_0 \ldots b_n]$
- $K([b_0 \ldots b_n]) = b_{n+1} \ldots b_{2n}$
- etc.

- Advantage of OFB
  - If we need to perform per-packet encryption, we don't need to pad the payload
  - Keystream can be generated in advance, before message to be sent arrives
  - Destination knows IV and K, therefore can generate same keystream

- Ciphertext generated as usual
  - Encryption: $c = m \oplus ks$
  - Decryption: $m = c \oplus ks$

- Potential problem
  - If somebody knows a portion P or the plaintext, that can be replaced with another "malicious" portion P'

Output Feedback Mode (k-bits)

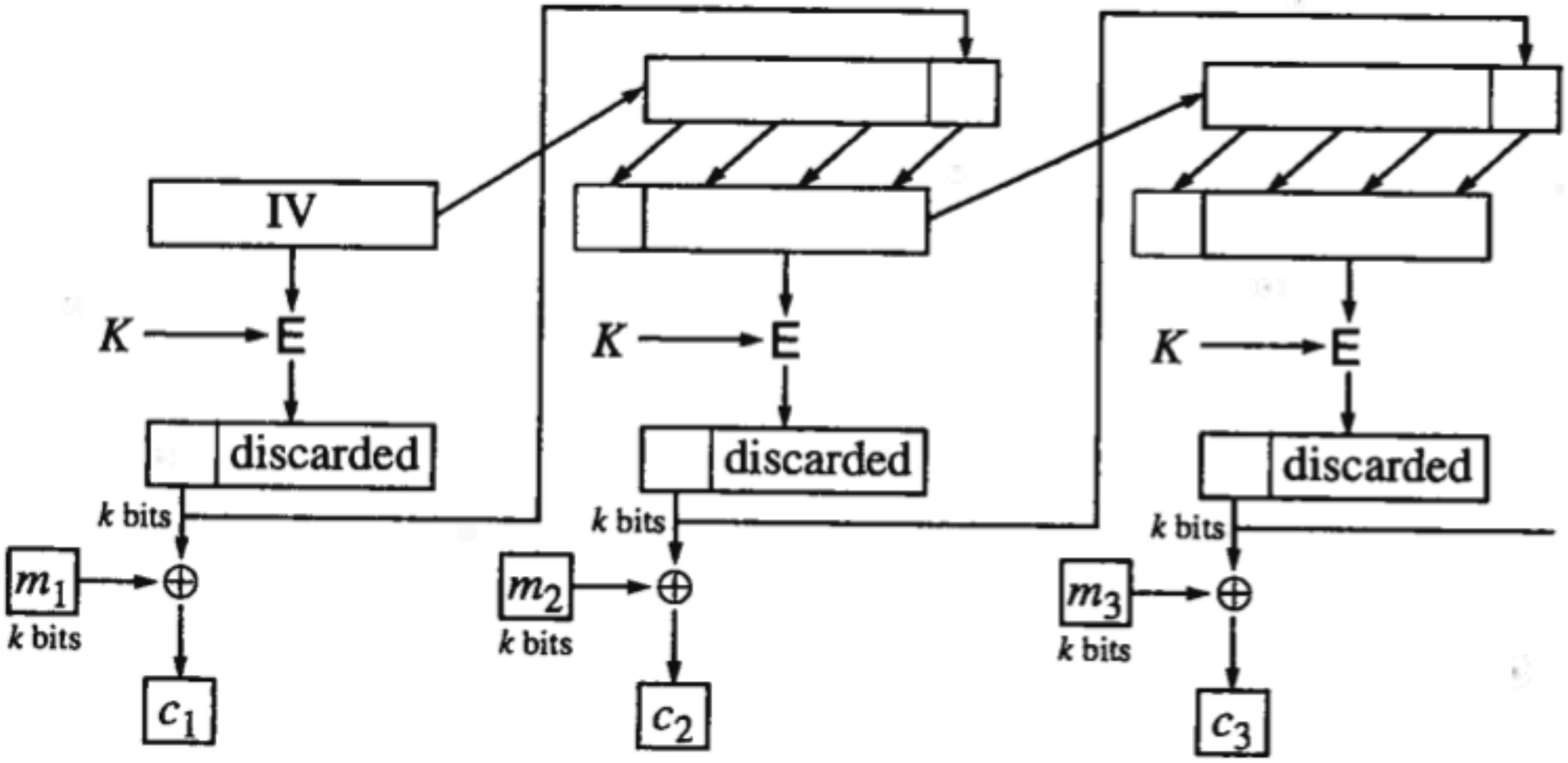


Figure 4-8. *k*-bit OFB

See Kaufman et al. "Network Security, Private Communication in a Public World"

Cipher Feedback Mode

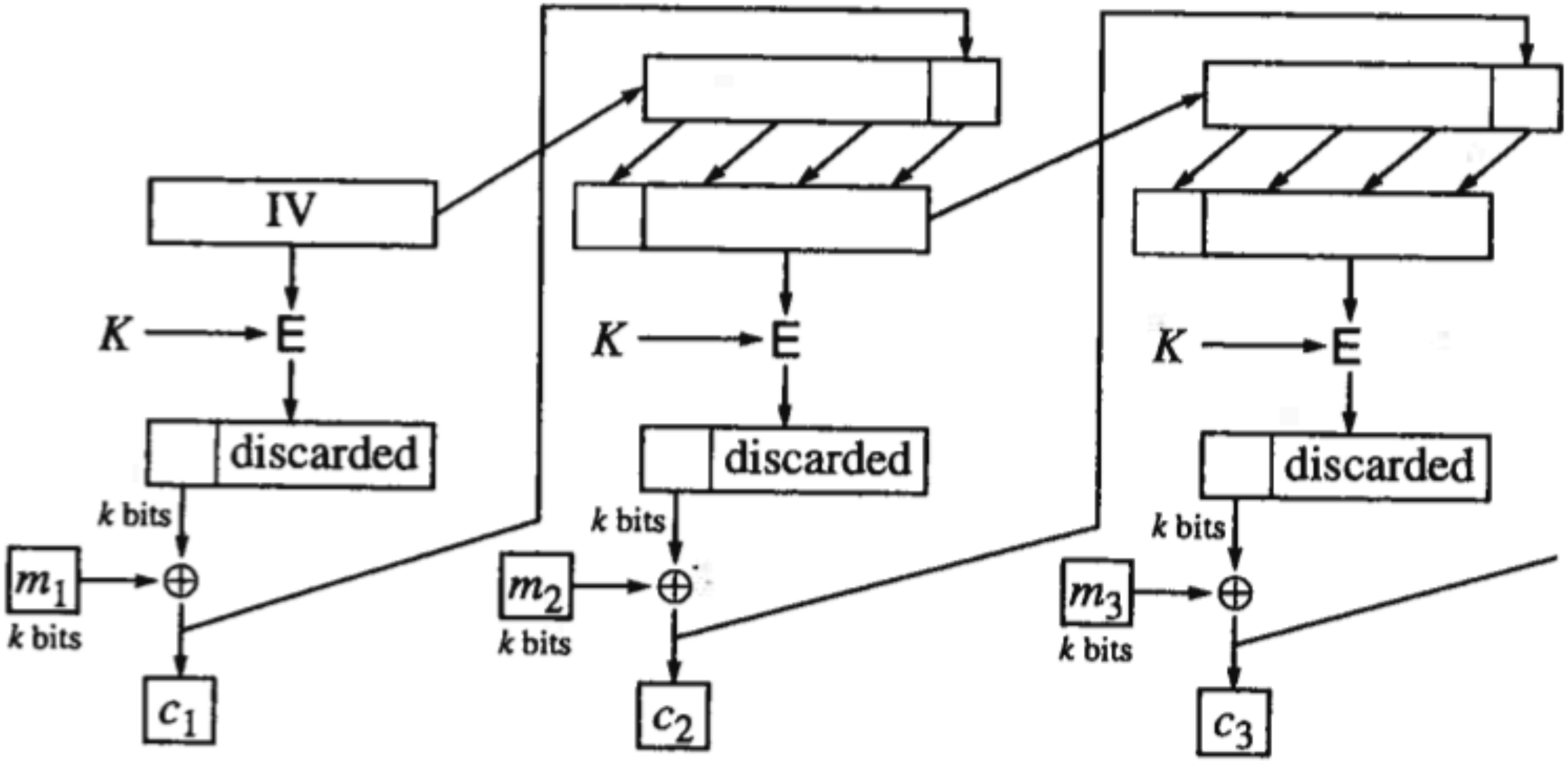


Figure 4-9. *k*-bit CFB

See Kaufman et al. “Network Security, Private Communication in a Public World”

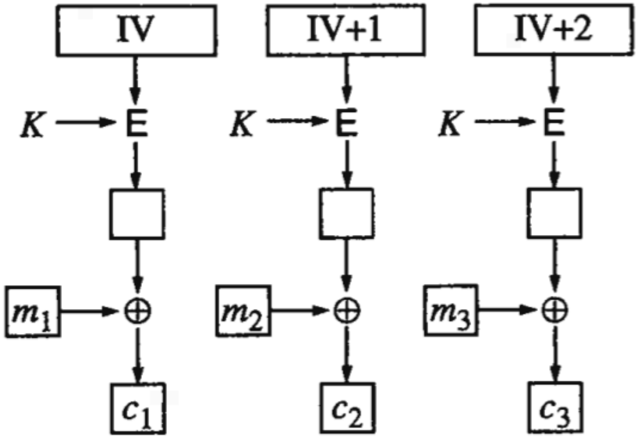
Cipher Feedback Mode (CFB)

- Keystream cannot be generated in advance
  - Need to wait for message to arrive

- Comparison with CBC and OFB
  - OFB: bit errors do not propagate beyond the current k-bit block
  - CBC/OFB: if bits of ciphertext lost in transmission, the entire rest of transmission is garbled
  - CFB: with 8-bit CFB, as long as the error is an integral number of bytes, things will re-sync. (1 bit error will affect 9 consecutive bytes)

Counter Mode (CTR)

- Similar to OFM
- Encrypts increments of IV to generate keystream
- Advantages:
  - Decryption can start anywhere, as long as you know the block number you are considering
  - Encryption/decryption can be trivially parallelized
  - Keystream can be preprocessed once IV is known
  - Useful in case of encrypted random access files, for example



See Kaufman et al. “Network Security, Private Communication in a Public World”

Summary

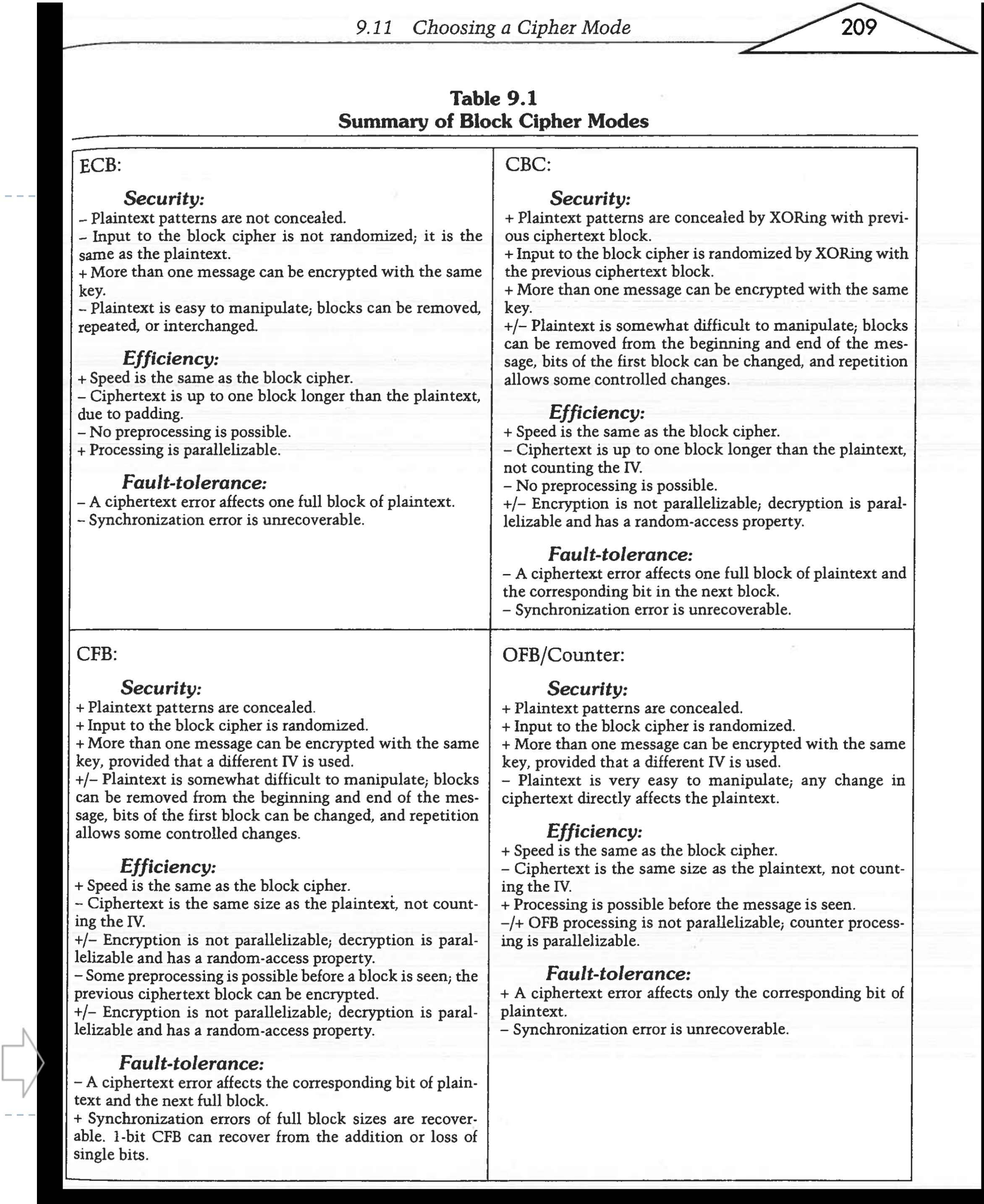


Table 9.1
Summary of Block Cipher Modes

ECB: ***Security:*** – Plaintext patterns are not concealed. – Input to the block cipher is not randomized; it is the same as the plaintext. + More than one message can be encrypted with the same key. – Plaintext is easy to manipulate; blocks can be removed, repeated, or interchanged. ***Efficiency:*** + Speed is the same as the block cipher. – Ciphertext is up to one block longer than the plaintext, due to padding. – No preprocessing is possible. + Processing is parallelizable. ***Fault-tolerance:*** – A ciphertext error affects one full block of plaintext. – Synchronization error is unrecoverable.	CBC: ***Security:*** + Plaintext patterns are concealed by XORing with previous ciphertext block. + Input to the block cipher is randomized by XORing with the previous ciphertext block. + More than one message can be encrypted with the same key. +/– Plaintext is somewhat difficult to manipulate; blocks can be removed from the beginning and end of the message, bits of the first block can be changed, and repetition allows some controlled changes. ***Efficiency:*** + Speed is the same as the block cipher. – Ciphertext is up to one block longer than the plaintext, not counting the IV. – No preprocessing is possible. +/– Encryption is not parallelizable; decryption is parallelizable and has a random-access property. ***Fault-tolerance:*** – A ciphertext error affects one full block of plaintext and the corresponding bit in the next block. – Synchronization error is unrecoverable.
CFB: ***Security:*** + Plaintext patterns are concealed. + Input to the block cipher is randomized. + More than one message can be encrypted with the same key, provided that a different IV is used. +/– Plaintext is somewhat difficult to manipulate; blocks can be removed from the beginning and end of the message, bits of the first block can be changed, and repetition allows some controlled changes. ***Efficiency:*** + Speed is the same as the block cipher. – Ciphertext is the same size as the plaintext, not counting the IV. +/– Encryption is not parallelizable; decryption is parallelizable and has a random-access property. – Some preprocessing is possible before a block is seen; the previous ciphertext block can be encrypted. +/– Encryption is not parallelizable; decryption is parallelizable and has a random-access property. ***Fault-tolerance:*** – A ciphertext error affects the corresponding bit of plaintext and the next full block. + Synchronization errors of full block sizes are recoverable. 1-bit CFB can recover from the addition or loss of single bits.	OFB/Counter: ***Security:*** + Plaintext patterns are concealed. + Input to the block cipher is randomized. + More than one message can be encrypted with the same key, provided that a different IV is used. – Plaintext is very easy to manipulate; any change in ciphertext directly affects the plaintext. ***Efficiency:*** + Speed is the same as the block cipher. – Ciphertext is the same size as the plaintext, not counting the IV. + Processing is possible before the message is seen. –/+ OFB processing is not parallelizable; counter processing is parallelizable. ***Fault-tolerance:*** + A ciphertext error affects only the corresponding bit of plaintext. – Synchronization error is unrecoverable.

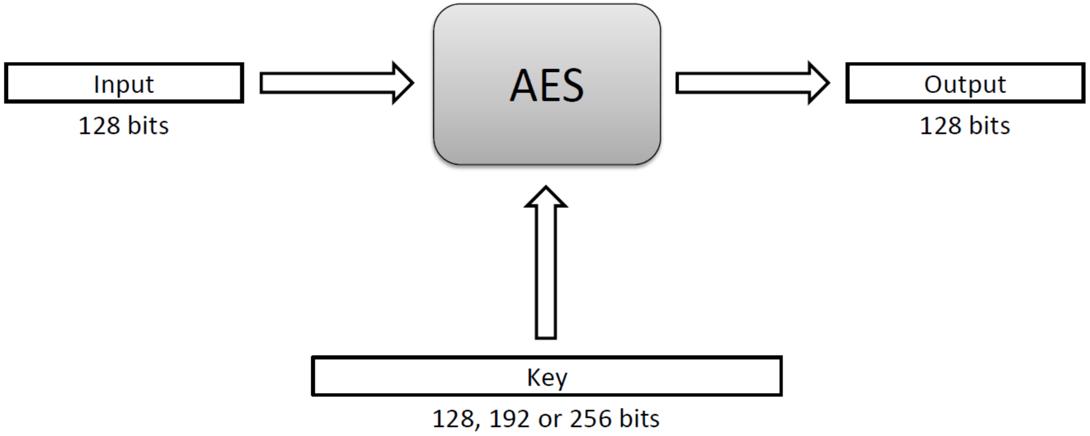
From “Applied Cryptography”, 2nd edition
Bruce Schneier
Wiley

AES: Advanced Encryption Standard

- In 1997, the U.S. National Institute for Standards and Technology (NIST) put out a public call for a replacement to DES.
- It narrowed down the list of submissions to five finalists, and ultimately chose an algorithm that is now known as the **Advanced Encryption Standard** (**AES**).
- new (Nov. 2001) symmetric-key NIST standard, replacing DES
  - **Nice mathematical justification for design choices**
- processes data in 128 bit blocks
- 128, 192, or 256 bit keys
- brute force decryption (try each key) taking 1 sec on DES, takes 149 trillion years for AES

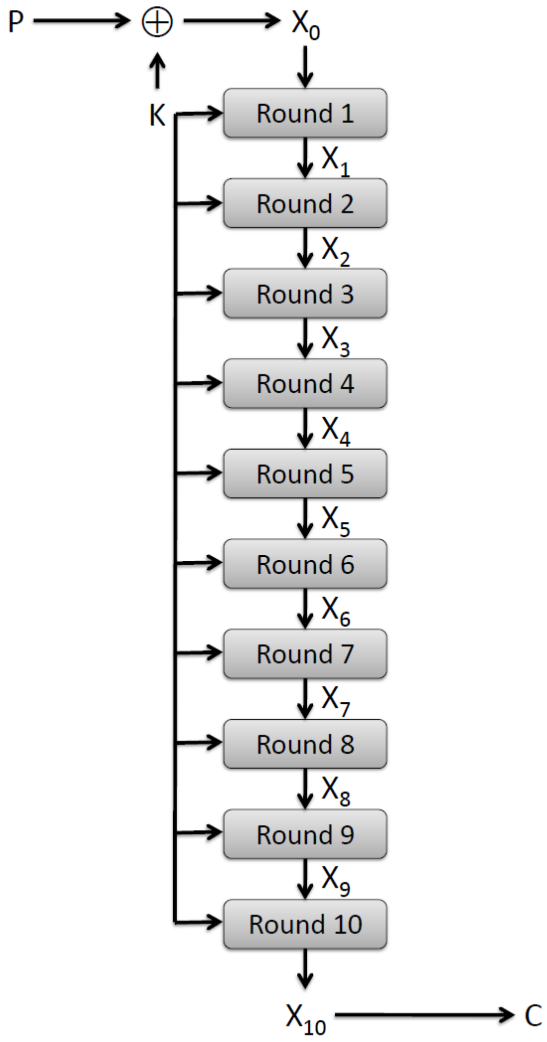
The Advanced Encryption Standard (AES)

- AES is a block cipher that operates on 128-bit blocks. It is designed to be used with keys that are 128, 192, or 256 bits long, yielding ciphers known as AES-128, AES-192, and AES-256.



AES Round Structure

- The 128-bit version of the AES encryption algorithm proceeds in ten rounds.
- Each round performs an invertible transformation on a 128-bit array, called **state**.
- The initial state X_0 is the XOR of the plaintext P with the key K:
- $X_0 = P \text{ XOR } K$.
- Round i (i = 1, …, 10) receives state X_{i-1} as input and produces state X_i.
- The ciphertext C is the output of the final round: $C = X_{10}$.



AES Rounds

- Each round is built from four basic steps:

1. **SubBytes step**: an S-box substitution step
2. **ShiftRows step**: a permutation step
3. **MixColumns step**: a matrix multiplication step
4. **AddRoundKey step**: an XOR step with a **round key** derived from the 128-bit encryption key

Key Exchange

- Enable Alice to communicate with Bob using shared key
  - The key cannot be transmitted in clear
  - It must be either encrypted when transmitted, or derived in a way that a third party cannot derive the same key
  - Alice and Bob may rely on a trusted third party, e.g., Cathy
  - The cryptosystem and protocols are publicly known

- First Attempt to Key Exchange
  - Alice and Cathy share a secret Ka
  - Cathy and Bob share a secret Kb

1. Alice >> Cathy : Ka(request for session key to Bob)
2. Cathy >> Alice : Ka(Ks) | Kb(Ks)
3. Alice >> Bob : Kb(Ks)
4. Alice can now privately send message M to Bob using Ks
   1. Alice >> Bob : Ks(M)

See Bishop "Introduction to Computer Security"

Key Exchange

- Problem: Replay Attack
  - Eve records (3) and Ks(M), which was sent by Alice to Bob
  - Eve >> Bob: Kb(Ks)
  - Eve >> Bob: Ks(M)
  - If M = "Deposit $500k in Roberto's account", we have a problem!

- Needham-Schroeder protocol
  1. Alice >> Cathy : "Alice" | "Bob" | Rand1
  2. Cathy >> Alice : Ka("Alice" | "Bob" | Rand1 | Ks | Kb("Alice" | Ks))
  3. Alice >> Bob : Kb("Alice" | Ks)
  4. Bob >> Alice : Ks(Rand2)
  5. Alice >> Bob : Ks(Rand2-1)

See Bishop "Introduction to Computer Security"

SYSTEM FAILURE
error 235553261...pending.....
fatal ER # 5444167QW32Z__wS e%$$

CSCI 4250/6250 – Fall 2015
Computer and Networks Security

INTRODUCTION TO CRYPTO
CHAPTER 8 (Goodrich)
CHAPTER 2-6 (Kaufman)
CHAPTER 8 (Kurose)

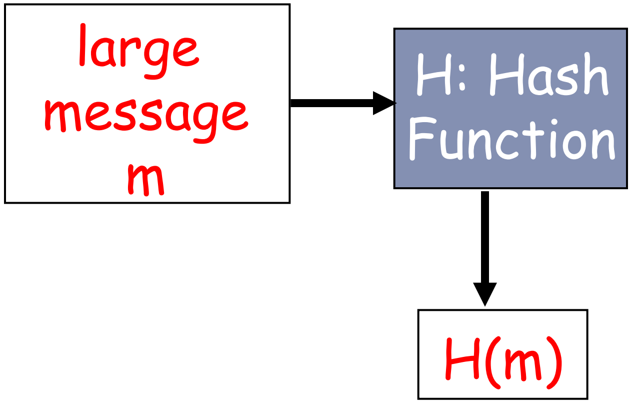
- Slides adapted from Kurose et al., Goodrich et al., and Kaufman et al.

Message Integrity

- Allows communicating parties to verify that received messages are authentic.
  - Content of message has not been altered
  - Source of message is who/what you think it is
  - Message has not been replayed
  - Sequence of messages is maintained
- Let's first talk about message digests

Message Digests

- Function H( ) that takes as input an arbitrary length message and outputs a fixed-length string: “message signature”
- Note that H( ) is a many-to-1 function
- H( ) is often called a “hash function”



- Desirable properties:
  - Easy to calculate
  - Irreversibility: Can’t determine m from H(m)
  - Collision resistance: Computationally difficult to produce m and m’ such that H(m) = H(m’)
  - ***Seemingly random output***

Internet checksum: poor message digest

Internet checksum has some properties of hash function:

- produces fixed length digest (16-bit sum) of input
- is many-to-one

- But given message with given hash value, it is easy to find another message with same hash value.
- Example: Simplified checksum: add 4-byte chunks at a time:

message	ASCII format
I O U 1	49 4F 55 31
0 0 . 9	30 30 2E 39
9 B O B	39 42 D2 42
	B2 C1 D2 AC

message	ASCII format
I O U 9	49 4F 55 39
0 0 . 1	30 30 2E 31
9 B O B	39 42 D2 42
	B2 C1 D2 AC

different messages but identical checksums!

Hash Functions

- A hash function h maps a plaintext x to a fixed-length value x = h(P) called hash value or digest of P
  - A collision is a pair of plaintexts P and Q that map to the same hash value, h(P) = h(Q)
  - Collisions are unavoidable
  - For efficiency, the computation of the hash function should take time proportional to the length of the input plaintext
- Example of application: Hash table
  - Search data structure based on storing items in locations associated with their hash value
  - Chaining deals with collisions
  - Domain of hash values proportional to the expected number of items to be stored
  - The hash function should spread plaintexts uniformly over the possible hash values to achieve constant expected search time

Cryptographic Hash Functions

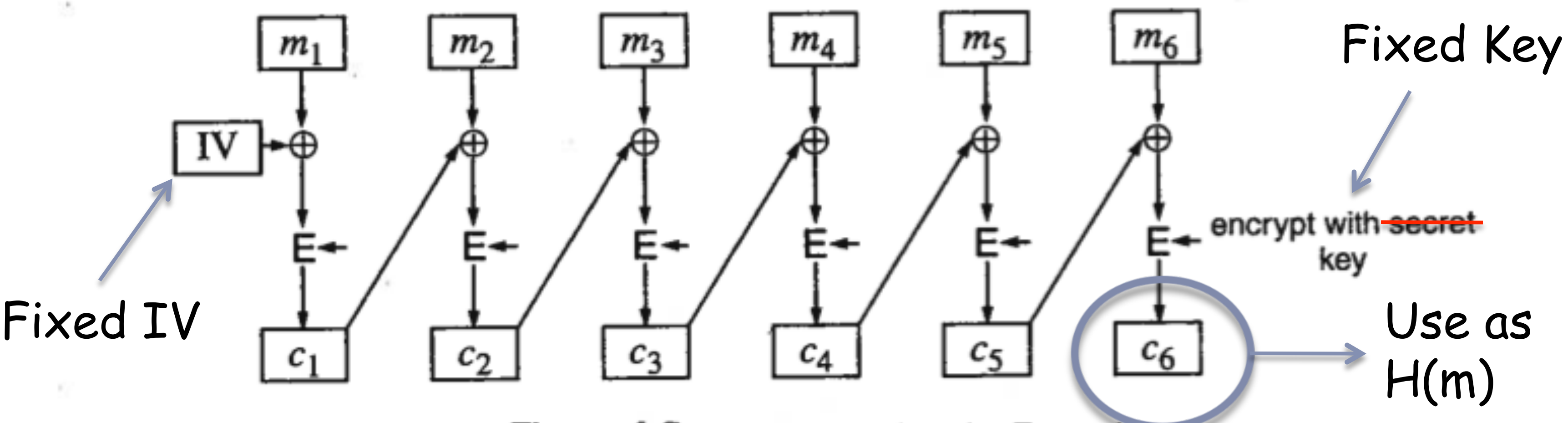
- A cryptographic hash function satisfies additional properties
  - Preimage resistance (aka one-way)
    - Given a hash value x, it is hard to find a plaintext P such that h(P) = x
  - Second preimage resistance (aka weak collision resistance)
    - Given a plaintext P, it is hard to find a plaintext Q such that h(Q) = h(P)
  - Collision resistance (aka strong collision resistance)
    - It is hard to find a pair of plaintexts P and Q such that h(Q) = h(P)
- Collision resistance implies second preimage resistance
- Hash values of at least 256 bits recommended to defend against brute-force attacks

How to build a Hash Function

- Can we use a block cipher + CBC?
- How?

How to build a Hash Function

- Can we use a block cipher + CBC?
- How?



Fixed Key

encrypt with ~~secret~~ key

Fixed IV

Use as H(m)

Figure 4-5. Cipher Block Chaining Encryption

- Problem
  - Not very efficient!

Hash Function Algorithms

- MD5 hash function widely used (RFC 1321)
  - computes 128-bit message digest in 4-step process.
- SHA-1 is also used.
  - US standard [NIST, FIPS PUB 180-1]
  - 160-bit message digest

Often, no good justification for design choices in Hash functions.

Message-Digest Algorithm 5 (MD5)

- Developed by Ron Rivest in 1991
- Uses 128-bit hash values
- Still widely used in legacy applications although considered insecure
- Various severe vulnerabilities discovered
- Chosen-prefix collisions attacks found by Marc Stevens, Arjen Lenstra and Benne de Weger
  - Start with two arbitrary plaintexts P and Q
  - One can compute suffixes S1 and S2 such that P||S1 and Q||S2 collide under MD5 by making 250 hash evaluations
  - Using this approach, a pair of different executable files or PDF documents with the same MD5 hash can be computed

Problems with MD5

- Hash collisions created this way are usually not directly applicable to attack widespread document formats or protocols.
- Attacks are possible by abusing dynamic constructs present in many formats
  - E.g., a malicious document would contain two different messages in the same document, but conditionally displays one or the other
- Computer programs have conditional constructs (if-then-else) that allow testing whether a location in the file has one value or another.
- Some document formats like PostScript, or macros in Microsoft Word, also have conditional constructs.

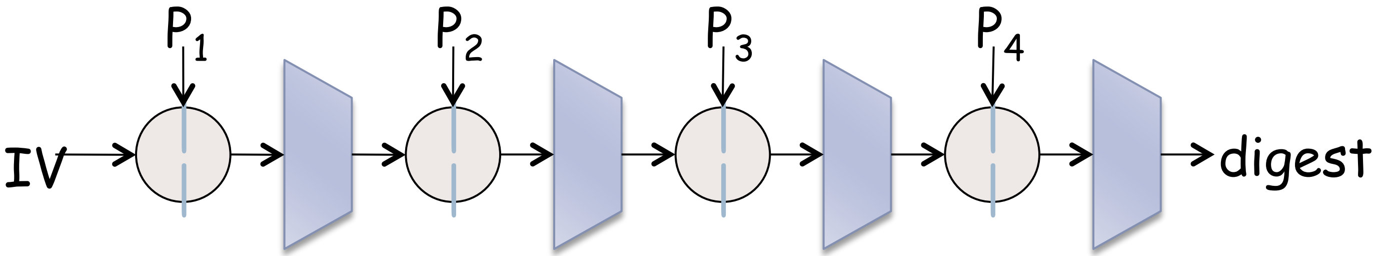
- Finding such colliding docs/programs may take just a few seconds on modern CPUs

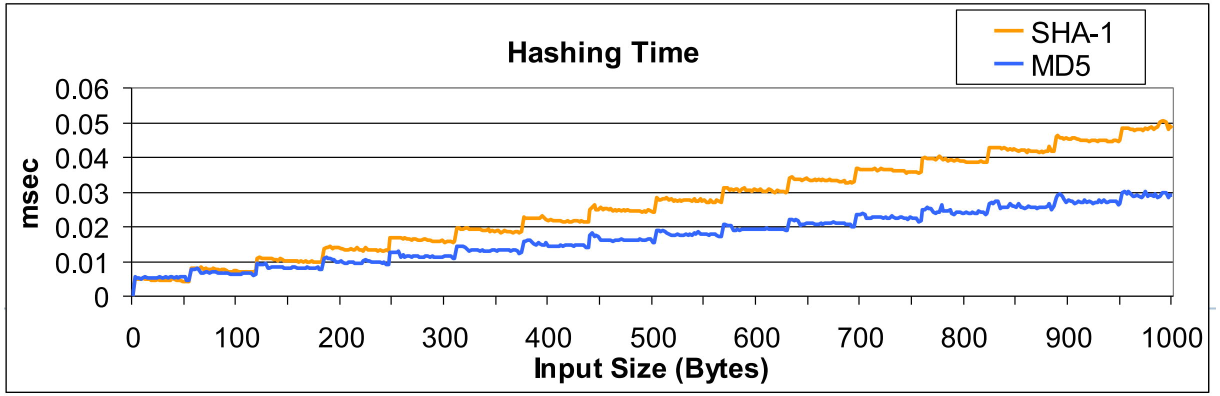
Secure Hash Algorithm (SHA)

- Developed by NSA and approved as a federal standard by NIST
- SHA-0 and SHA-1 (1993)
  - 160-bits
  - Considered insecure
  - Still found in legacy applications
  - Vulnerabilities less severe than those of MD5
- SHA-2 family (2002)
  - 256 bits (SHA-256) or 512 bits (SHA-512)
  - Still considered secure despite published attack techniques
- Public competition for SHA-3 announced in 2007

Iterated Hash Function

- A compression function works on input values of fixed length
  - Inputs: X,Y with len(X)=m, len(Y)=n; Output: Z with len(Z)=n
- An iterated hash function extends a compression function to inputs of arbitrary length
  - padding, initialization vector, and chain of compression functions
  - inherits collision resistance of compression function
- MD5 and SHA are iterated hash functions





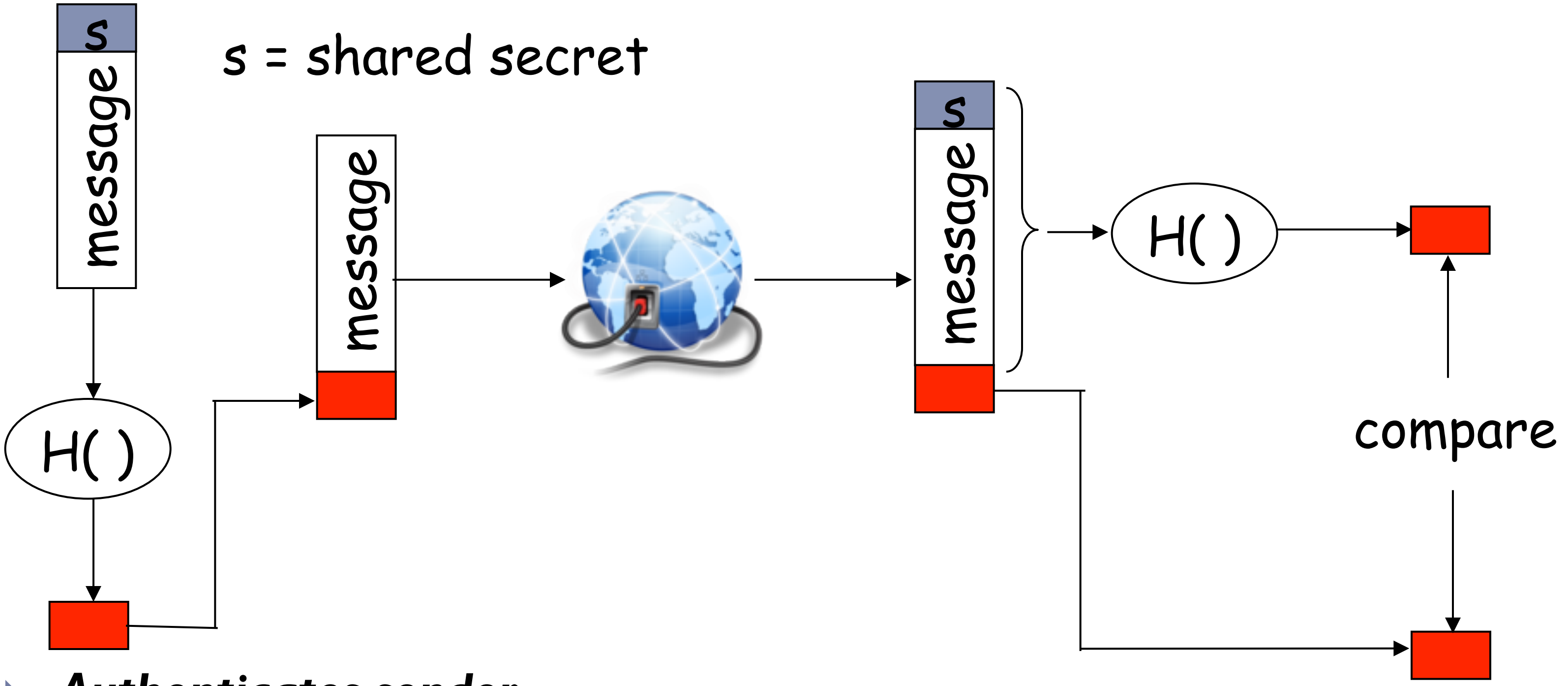
Question

- Assume we want to send a message
  - We are not concerned with confidentiality, only integrity

- What if we send
  - m' = m || MD5(m)
  - The receiver can extract m, compute MD5(m), and check if this matches the MD5 that was sent

- Does this guarantee integrity?

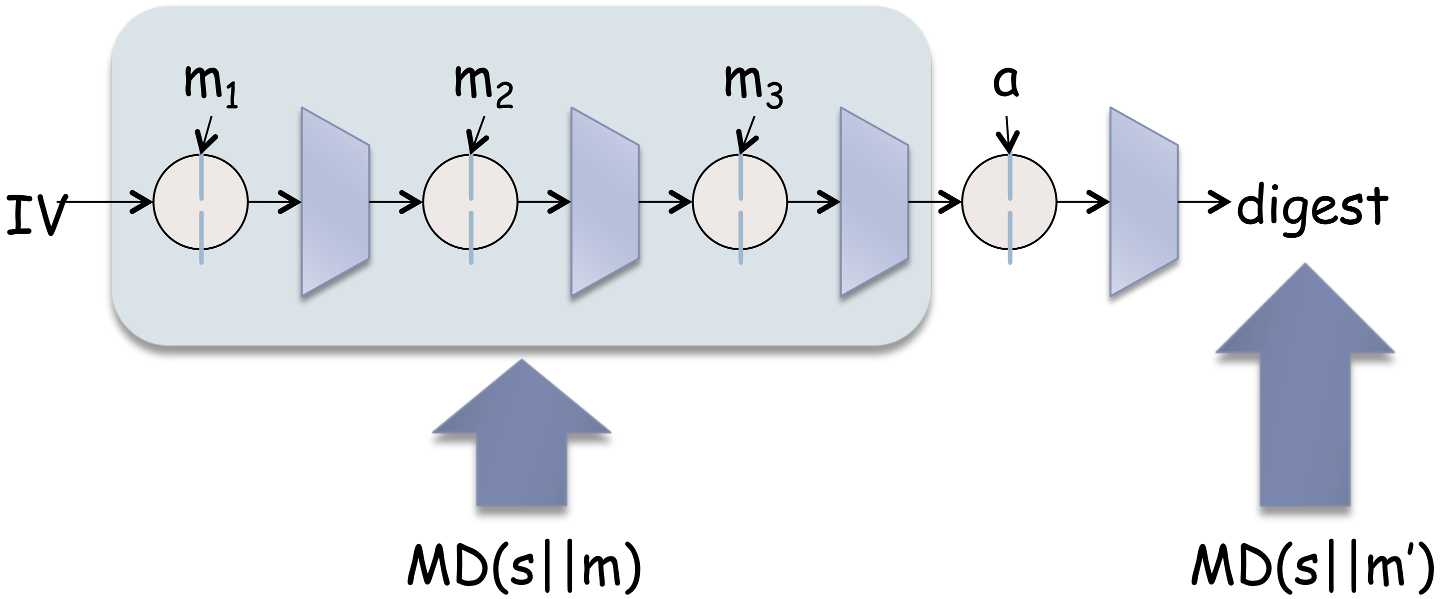
Message Authentication Code (MAC)



- ***Authenticates sender***
- ***Verifies message integrity***
- No encryption !
- Also called "keyed hash"
- Notation: MD_m = H(s||m) ; send m||MD_m
  - **Is this secure?** It seems like

Not so fast!

- Because most hash functions are iterated hash functions
  - Trudy knows the message m and MD(s||m)
  - She could append something to m to get m' = m||a, and use MD(s||m) to initialize the computation of MD(s||m')

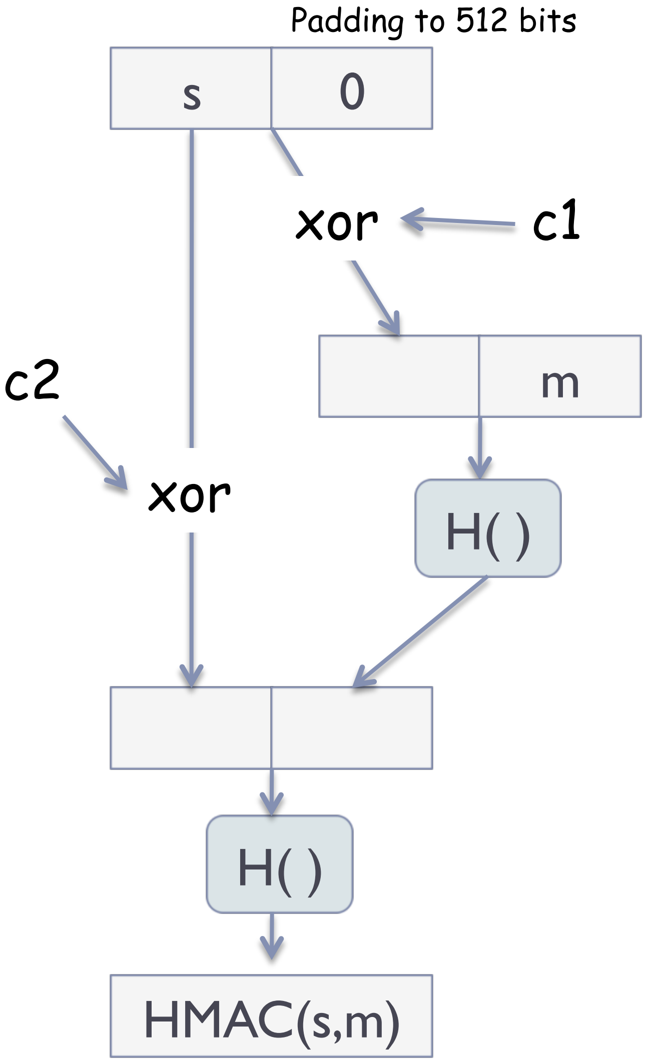


HMAC***

- Popular MAC standard
- Addresses some subtle flaws

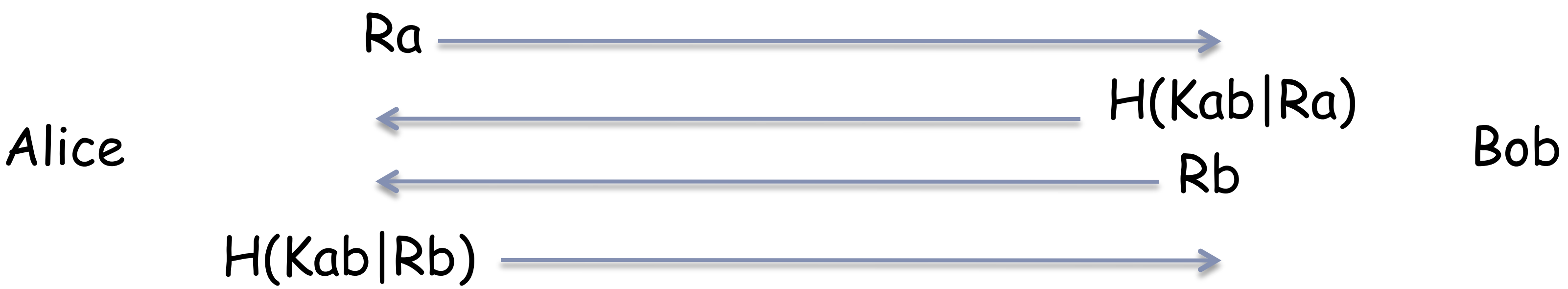
1. Concatenates secret to front of message.
2. Hashes concatenated message
3. Concatenates the secret to front of digest
4. Hashes the combination again.

HMAC(s,m) = H(s||H(s||M))



Other nifty things to do with a hash

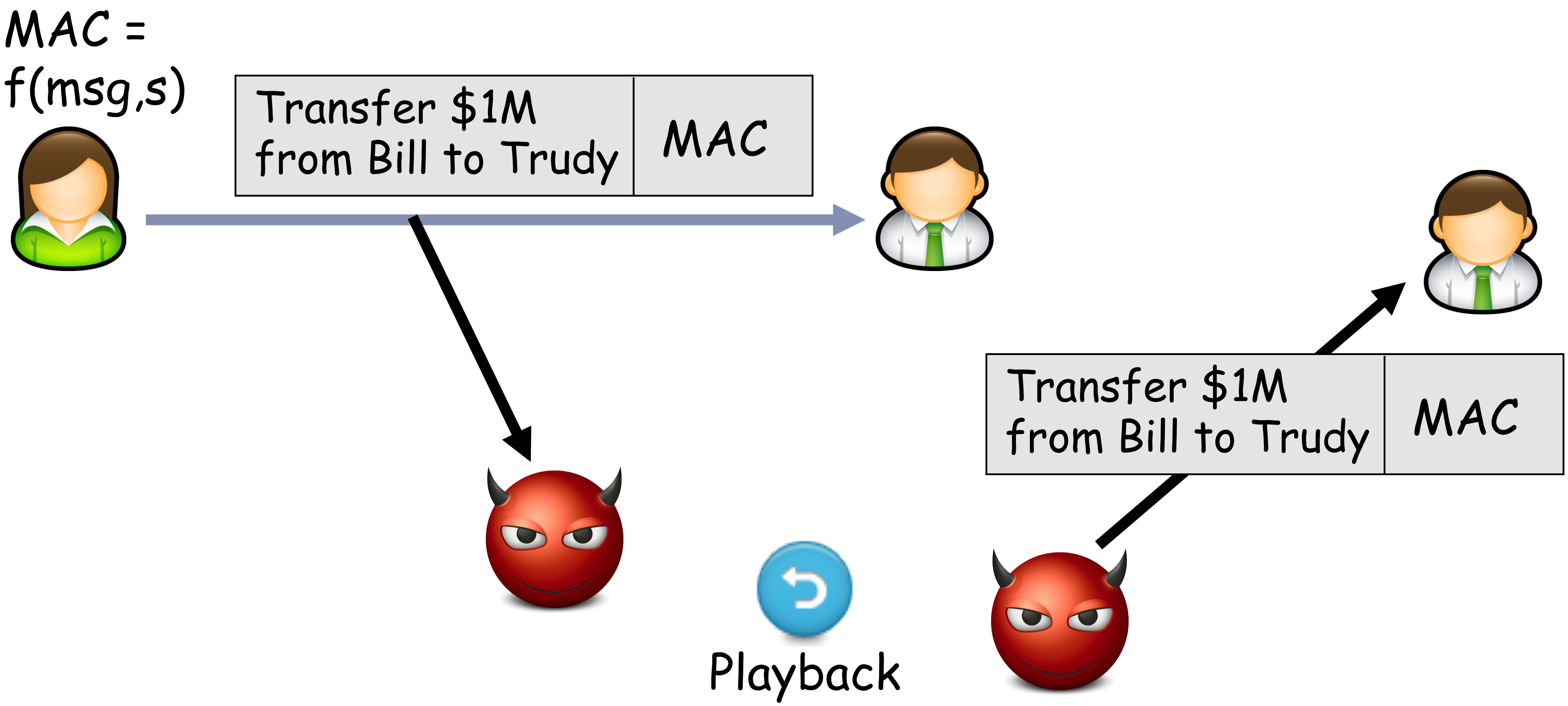
- Hashing passwords
- Document/Program fingerprint
- Authentication



- Encryption

b1 = H(Kab|IV) c1 = p1 xor b1
b2 = H(Kab|c1) c2 = p2 xor b2
b3 = H(Kab|c2) c3 = p3 xor b3
…

Playback attack



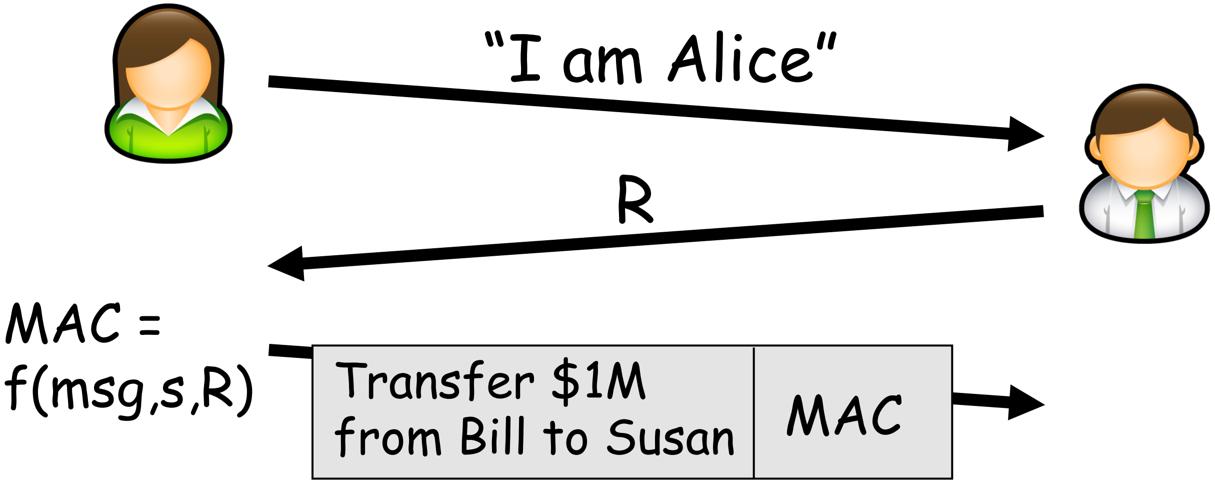
MAC = f(msg,s)

Transfer $1M from Bill to Trudy | MAC

Transfer $1M from Bill to Trudy | MAC

Playback

Defending against playback attack: nonce



"I am Alice"

R

MAC = f(msg,s,R)

Transfer $1M from Bill to Susan

MAC

SYSTEM FAILURE
error 235553261...pending.....
fatal ER # 5444167QW32Z__wS e%$$

CSCI 4250/6250 – Fall 2015
Computer and Networks Security

INTRODUCTION TO CRYPTO
CHAPTER 8 (Goodrich)
CHAPTER 2-6 (Kaufman)
CHAPTER 8 (Kurose)

Slides adapted from Kurose et al., Goodrich et al., and Kaufman et al.

Public Key Cryptography

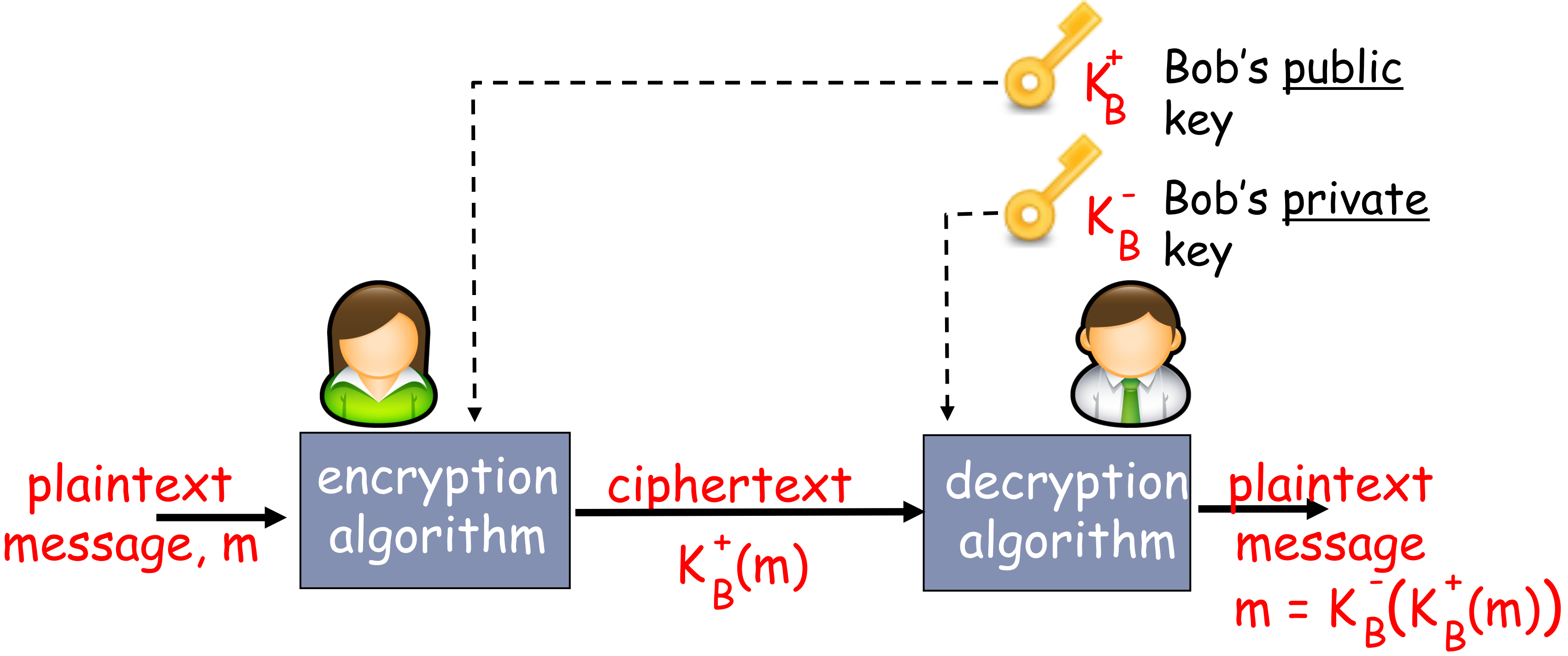
symmetric key crypto

- requires sender, receiver know shared secret key
- Q: how to agree on key in first place (particularly if never "met")?

public key cryptography

- ❐ radically different approach [Diffie-Hellman76, RSA78]
- ❐ sender, receiver do *not* share secret key
- ❐ *public* encryption key known to *all*
- ❐ *private* decryption key known only to receiver

Public key cryptography



Public key encryption algorithms

Requirements:

1. need $K_B^+(\,)$ and $K_B^-(\,)$ such that

$$K_B^-(K_B^+(m)) = m$$

2. given public key K_B^+, it should be impossible to compute private key K_B^-

RSA: Rivest, Shamir, Adelson algorithm

RSA: getting ready

- A message is a bit pattern.
- A bit pattern can be uniquely represented by an integer number.
- Thus encrypting a message is equivalent to encrypting a number.

Example

- m= 10010001
  - This message is uniquely represented by the decimal number 145.
  - To encrypt m, we encrypt the corresponding number, which gives a new number (the cyphertext).

RSA: Creating public/private key pair

1. Choose two large prime numbers $p, q.$
   (e.g., 1024 bits each, to avoid brute force given n)

2. Compute $n = pq, \quad z = (p-1)(q-1)$

3. Choose e (with $e<n$) that has no common factors with z. (e, z are "relatively prime").

4. Choose d (with d<n) so that $ed-1$ is divisible by $z.$
   (in other words: ed mod $z = 1$).

5. *Public* key is $\underbrace{(n,e)}_{K_B^+}$. *Private* key is $\underbrace{(n,d)}_{K_B^-}$.

RSA: Creating public/private key pair

1. Choose two large prime numbers *p, q.* (e.g., 1024 bits each, to avoid brute force given *n*)

2. Compute $n = pq$, $z = (p-1)(q-1)$

3. Choose *e* (*with e<n)* that has no common factors with z. (*e, z* are "relatively prime").

4. Choose *d* (with d<n) so that *ed-1* is divisible by *z.* (in other words: *ed* mod *z* = *1*).

5. *Public* key is $\underbrace{(n,e)}_{K_B^+}$. *Private* key is $\underbrace{(n,d)}_{K_B^-}$.

e can be relatively small

d should be large

RSA: Encryption, decryption

0. Given $\overset{\text{public}}{(n,e)}$ and $\overset{\text{private}}{(n,d)}$ as computed above

1. To encrypt message m ($<n$), compute

   $c = m^e \bmod n$

2. To decrypt received bit pattern, c, compute

   $m = c^d \bmod n$

Magic happens!

$$m = (\underbrace{m^e \bmod n}_{c})^d \bmod n$$

RSA example:

Bob chooses *p=5, q=7*. Then *n=35, z=24.*

e=5 (so *e, z* relatively prime).

d=29 (so *ed-1* exactly divisible by z)

ed-1 = 144, 144/24=6

Encrypting 8-bit messages.

	bit pattern	m	m^e	$c = m^e \bmod n$
encrypt:	0000l000	12	24832	17

	c	c^d	$m = c^d \bmod n$
decrypt:	17	481968572106750915091411825223071697	12

Prerequisite: modular arithmetic

- x mod n = remainder of x when divide by n
- Facts:

  [(a mod n) + (b mod n)] mod n = (a+b) mod n

  [(a mod n) - (b mod n)] mod n = (a-b) mod n

  [(a mod n) * (b mod n)] mod n = (a*b) mod n
- Thus

  $(a \bmod n)^d \bmod n = a^d \bmod n$
- Example: x=14, n=10, d=2:
  - $(x \bmod n)^d \bmod n = 4^2 \bmod 10 = 6$
  - $x^d = 14^2 = 196$ and $x^d \bmod 10 = 6$

Multiplicative Inverses (1)

- The residues modulo a positive integer n are the set

  $$Z_n = \{0, 1, 2, \ldots, (n - 1)\}$$

- Let x and y be two elements of Z_n such that

  $$xy \bmod n = 1$$

  We say that y is the multiplicative inverse of x in Z_n and we write $y = x^{-1}$

- Example:
  - Multiplicative inverses of the residues modulo 10

x	0	1	2	3	4	5	6	7	8	9
x^{-1}		1		7				3		9

Multiplicative Inverses (2)

Theorem

An element x of Z_n has a multiplicative inverse if and only if x and n are relatively prime

- Example
  - The elements of Z_{10} with a multiplicative inverse are $1, 3, 7, 9$

Corollary

If p is prime, every nonzero residue in Z_p has a multiplicative inverse

- Example:
  - Multiplicative inverses of the residues modulo 11

x	0	1	2	3	4	5	6	7	8	9	10
x^{-1}		1	6	4	3	9	2	8	7	5	10

Euler's Theorem

- The multiplicative group for Z_n, denoted with Z^*_n, is the subset of elements of Z_n relatively prime with n
- The totient function of n, denoted with $\phi(n)$, is the size of Z^*_n
- Example

  $Z^*_{10} = \{ 1, 3, 7, 9 \}$ $\qquad\qquad$ $\phi(10) = 4$

- If p is prime, we have

  $Z^*_p = \{1, 2, \ldots, (p - 1)\}$ $\qquad\qquad$ $\phi(p) = p - 1$

Euler's Theorem

For each element x of Z^*_n, we have $x^{\phi(n)} \bmod n = 1$

- Example ($n = 10$)

  $3^{\phi(10)} \bmod 10 = 3^4 \bmod 10 = 81 \bmod 10 = 1$

  $7^{\phi(10)} \bmod 10 = 7^4 \bmod 10 = 2401 \bmod 10 = 1$

  $9^{\phi(10)} \bmod 10 = 9^4 \bmod 10 = 6561 \bmod 10 = 1$

- Consequence
  - $x^y \bmod n = x^{y \bmod \phi(n)} \bmod n$

Why?

- Remember
  - $[(a \bmod n)(b \bmod n)] \bmod n = (ab) \bmod n$
  - $(a \bmod n)^d \bmod n = a^d \bmod n$

- Then
  - $x^y \bmod n = x^{(k\phi(n)+r)} \bmod n = x^{k\phi(n)} x^r \bmod n =$
    $[\underbrace{(x^{k\phi(n)} \bmod n)}_{=1 \text{ if } x \text{ in } Z^*_n}(x^r \bmod n)] \bmod n = x^{y \bmod \phi(n)} \bmod n$

Why does RSA work?

- Remember that
  - p and q are two large primes
  - $n = pq$; $z = (p-1)(q-1) = \phi(n)$
  - $ed \bmod z = 1$
- z is equal to the *totient* of n
  - the number of *numbers* < *n* that are relatively prime to n
- Fact: for any x and y, $x^y \bmod n = x^{(y \bmod z)} \bmod n$

- We need to show that $c^d \bmod n = m$, where $c = m^e \bmod n$

$$
\begin{aligned}
c^d \bmod n &= (m^e \bmod n)^d \bmod n \\
&= m^{ed} \bmod n \\
&= m^{(ed \bmod z)} \bmod n \\
&= m^1 \bmod n \\
&= m \quad \Longrightarrow \quad \text{(notice that m in [0, n-1])}
\end{aligned}
$$

RSA: another important property

The following property will be *very* useful later:

$$\underbrace{K_B^-(K_B^+(m))}_{\substack{\text{use public key}\\\text{first, followed}\\\text{by private key}}} = m = \underbrace{K_B^+(K_B^-(m))}_{\substack{\text{use private key}\\\text{first, followed}\\\text{by public key}}}$$

Result is the same!

Why $K_B^-(K_B^+(m)) = m = K_B^+(K_B^-(m))$?

Follows directly from modular arithmetic:

$$
\begin{aligned}
(m^e \bmod n)^d \bmod n &= m^{ed} \bmod n \\
&= m^{de} \bmod n \\
&= (m^d \bmod n)^e \bmod n
\end{aligned}
$$

Why is RSA Secure?

- Suppose you know Bob's public key (n,e). How hard is it to determine d?
- Essentially need to find factors of n without knowing the two factors p and q.
- Fact: factoring a big integer is hard
  - Even harder for large semiprime numbers (product of two large primes)

Algorithmic Issues

`https://tools.ietf.org/html/rfc3447`

- The implementation of the RSA cryptosystem requires various algorithms
- Overall
  - Representation of integers of arbitrarily large size and arithmetic operations on them
- Encryption
  - Modular power
- Decryption
  - Modular power

- Setup
  - Generation of random numbers with a given number of bits (to generate candidates p and q)
  - Primality testing (to check that candidates p and q are prime)
  - Computation of the GCD (to verify that e and $\phi(n)$ are relatively prime)
  - Computation of the multiplicative inverse (to compute d from e)

Session keys

- In practice RSA key between 1024 and 4096 bits (GPG)
  - 128 to 512 bytes
  - Effective msg length is less, due to padding
- Exponentiation is computationally intensive
- DES is at least 100 times faster than RSA

Session key, K_S

- Bob and Alice use RSA to exchange a symmetric key K_S
- Once both have K_S, they use symmetric key cryptography

Diffie-Hellman

- Public key cryptosystem
  - First known public key-based system
  - Useful to perform key exchange when communication channel is not private

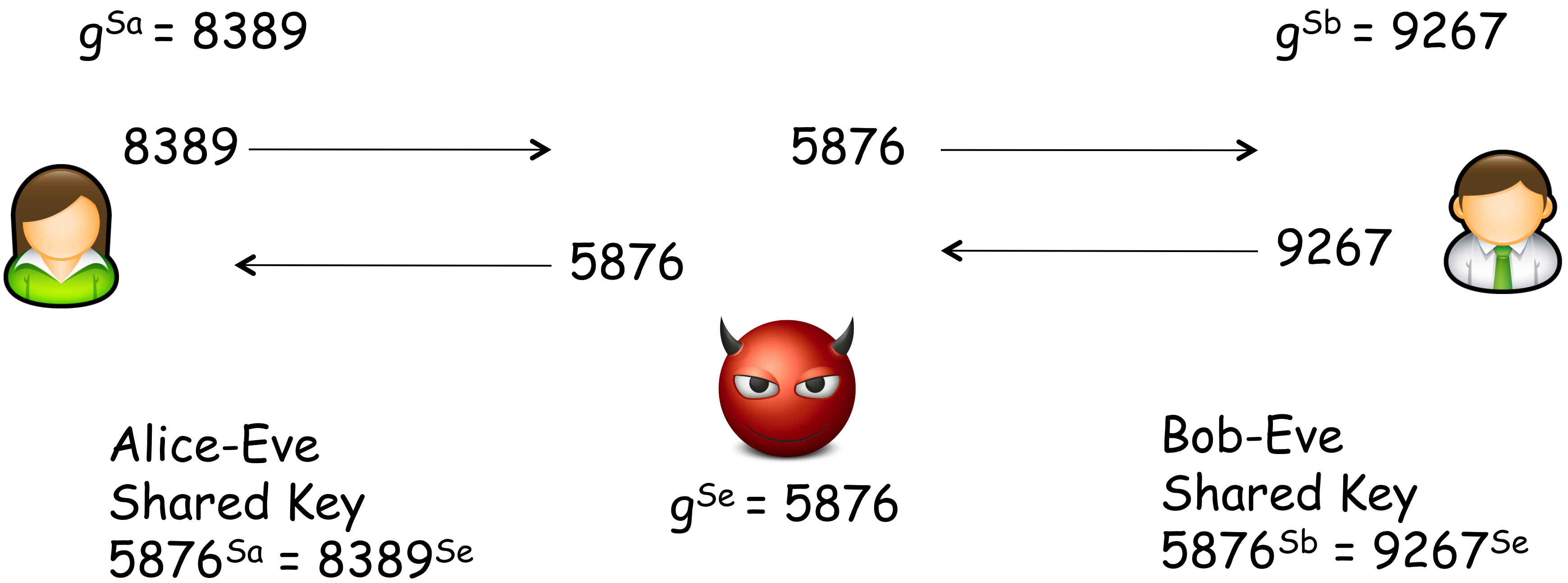
- Alice and Bob first agree on a large prime p and another number g < p (some subtle restrictions apply…), then
  1. g and p can be published (no need to keep them secret)
  2. Alice chooses a random number Sa, and Bob a rand num Sb
  3. Alice computes $Ta = g^{Sa} \bmod p$, Bob computes $Tb = g^{Sb} \bmod p$
  4. Alice and Bob exchange Ta and Tb (in public)
  5. Alice and Bob compute $Tb^{Sa} \bmod p$ and $Ta^{Sb} \bmod p$, respectively
  6. They will get the same number (the exchanged key)
     $Tb^{Sa} = g^{SbSa} \bmod p = g^{SaSb} \bmod p = Ta^{Sb}$

Diffie-Hellman

- Why is this secure?
  - Nobody else can calculate g^{SaSb}, even if they separately know Ta = g^{Sa} mod p and Tb = g^{Sb} mod p
  - To get Sa or Sb an attacker would need to compute discrete logarithms
    - □ E.g.: Sb = dlog(Tb | g, p)
    - Computing exponentials module a prime is easy
    - Discrete logarithms are very hard to compute
    - Mathematicians have not yet figured out how to do it efficiently

- Vulnerable to man-in-the-middle attack in certain scenarios
  - Alice and Bob do not authenticate each other
  - Attacker may intercept and replace Ta and Tb
  - To solve (or mitigate) problem, Ta and Tb should be stored in a secure repository of "public numbers"

DH – Man-in-the-Middle Attack



$g^{Sa} = 8389$

$g^{Sb} = 9267$

8389 →

5876 →

← 5876

← 9267

$g^{Se} = 5876$

Alice-Eve
Shared Key
$5876^{Sa} = 8389^{Se}$

Bob-Eve
Shared Key
$5876^{Sb} = 9267^{Se}$

Does it help if Alice and Bob try to verify their identity by sending each other a pre-shared password?

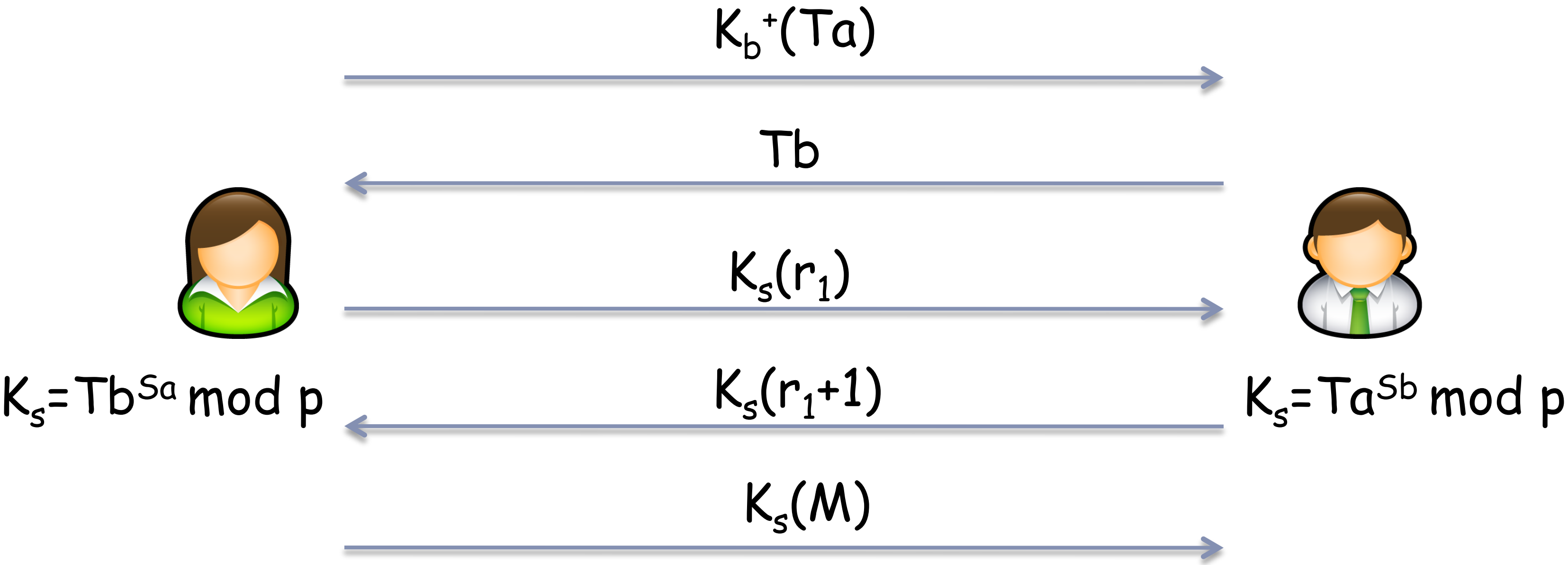
DH – Man-in-the-Middle Defense

- Published DH numbers
  - p and g are agreed upon
  - Each party chooses a fixed secret number Si and publishes her (Ti = g^{Si} mod p) in a reliable place
  - Assumption: the attacker cannot change/forge the published numbers

- Authenticated DH, examples
  - Alice can sign her Ta
  - Alice can encrypt her Ta with Bob's pub key
  - After DH, Alice sends Bob a hash HMAC(S|Ta), where S is a pre-shared secret (e.g., a password)

DH – Man-in-the-Middle Defense

- Bob is a server, and has a priv/pub key
- Alice knows (and trusts) Bob's pub key, K_b^+



(Eve can still pretend to be Alice)

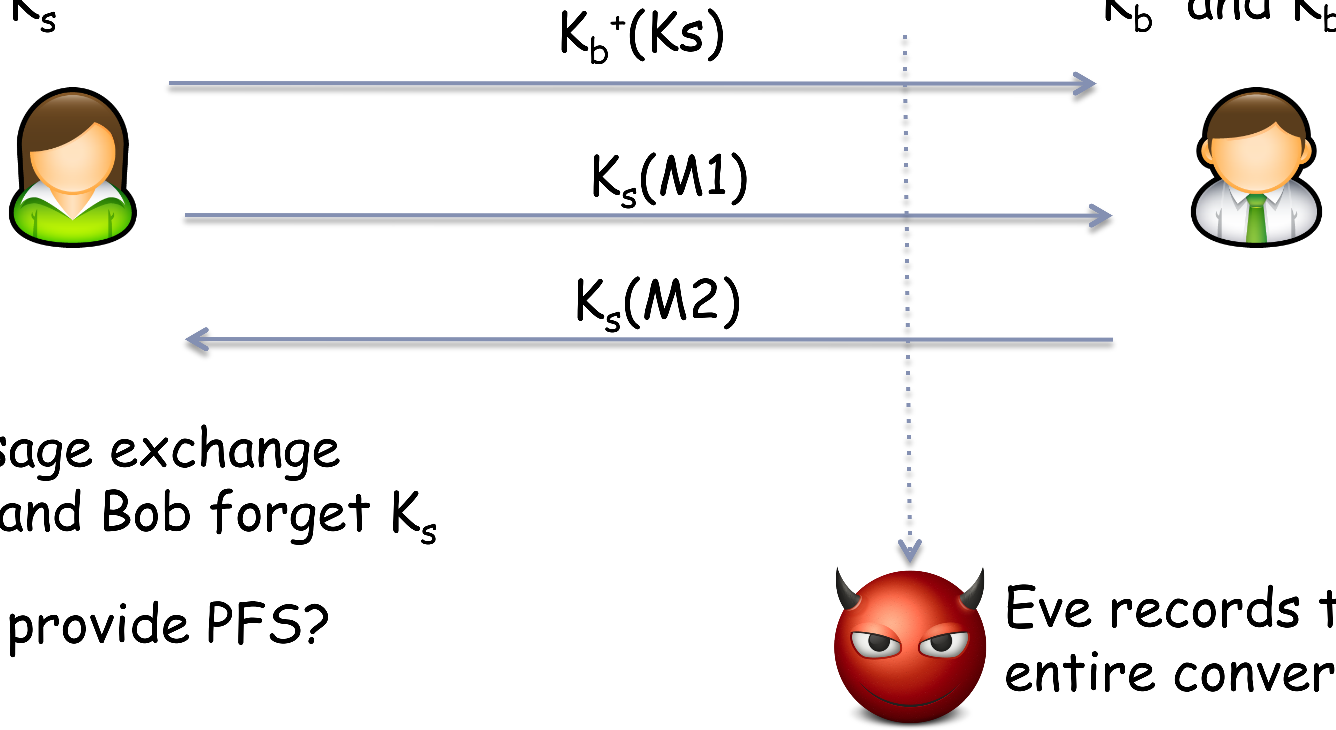
Perfect Forward Secrecy

- A protocol is said to have PFS if it is impossible for Trudy to decrypt a message m sent between Alice and Bob, even if Trudy, after m is sent, breaks into both Alice's and Bob's machines and steals their private keys
- This can be achieved by using session keys that
  - Are chosen independently from the private/public keys
  - Alice and Bob forget the session key as soon as the communication is over

Perfect Forward Secrecy

Alice chooses a random key K_s

Bob has his own K_b^+ and K_b^-



Alice

$K_b^+(Ks)$

$K_s(M1)$

$K_s(M2)$

Bob

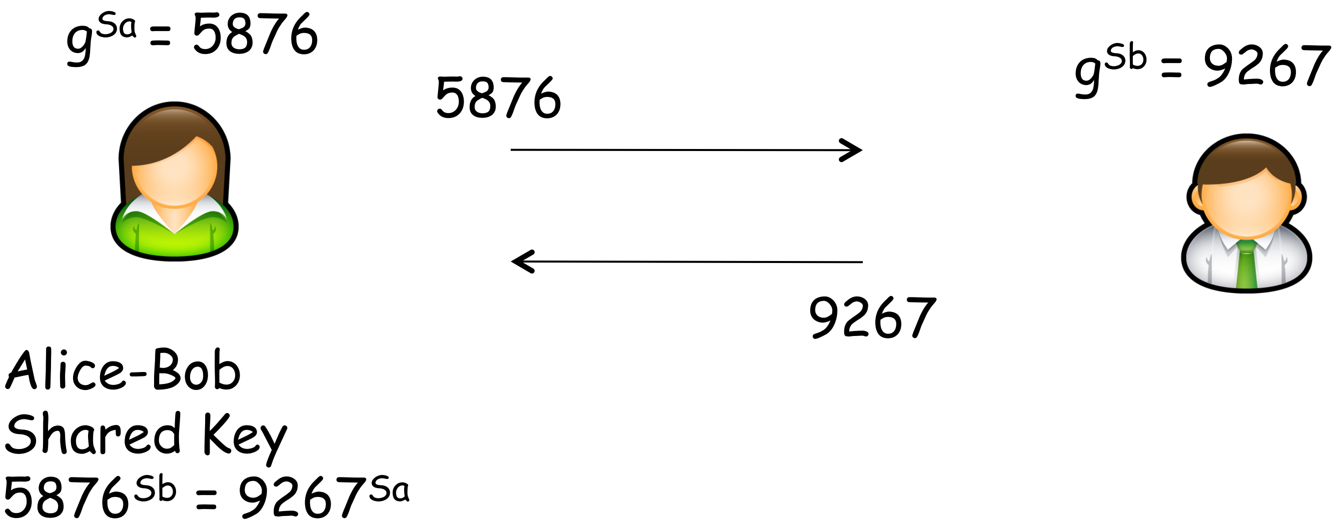
After message exchange both Alice and Bob forget K_s

Does this provide PFS?

Eve records the entire conversation

Perfect Forward Secrecy

- PFS can be attained using Diffie-Hellman
  - Why?



$g^{Sa} = 5876$

5876

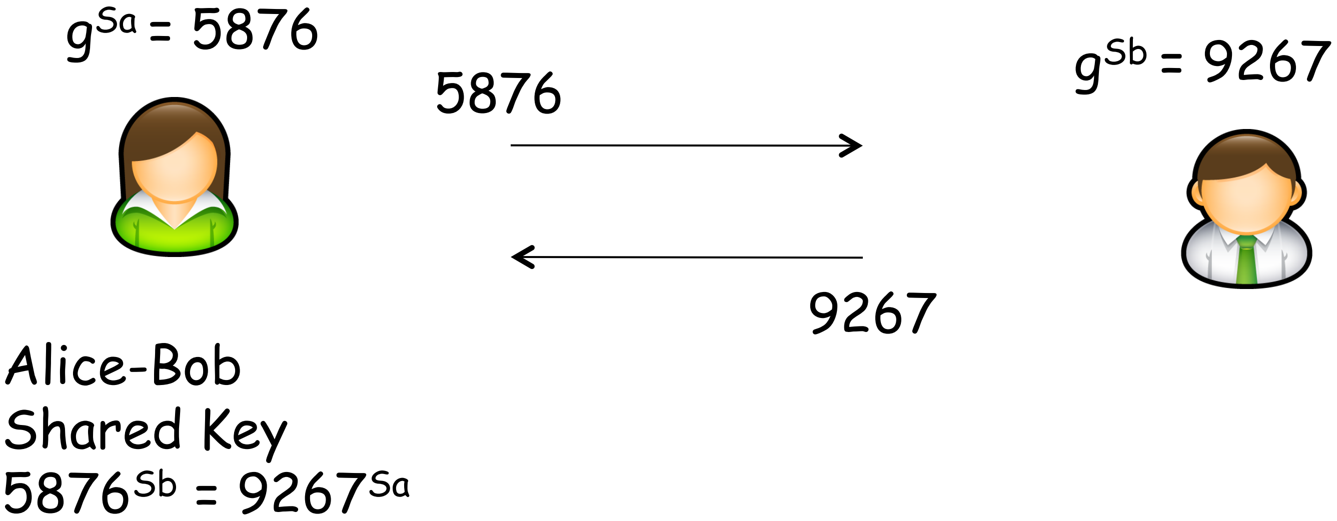
$g^{Sb} = 9267$

9267

Alice-Bob
Shared Key
$5876^{Sb} = 9267^{Sa}$

Perfect Forward Secrecy

- PFS can be attained using Diffie-Hellman
  - Alice and Bob forget their Sa and Sb after end of session



$g^{Sa} = 5876$

5876

$g^{Sb} = 9267$

9267

Alice-Bob
Shared Key
$5876^{Sb} = 9267^{Sa}$

Zero-Knowledge Proof Systems

- Used only for authentication
- Allows you to prove that you know a secret without actually revealing the secret
- E.g.: RSA is a zero-knowledge proof system
  - You can prove you know the "secret" associated with your public key without revealing your private key
- There exist ZKPSs that are much more efficient than RSA

Digital Signatures

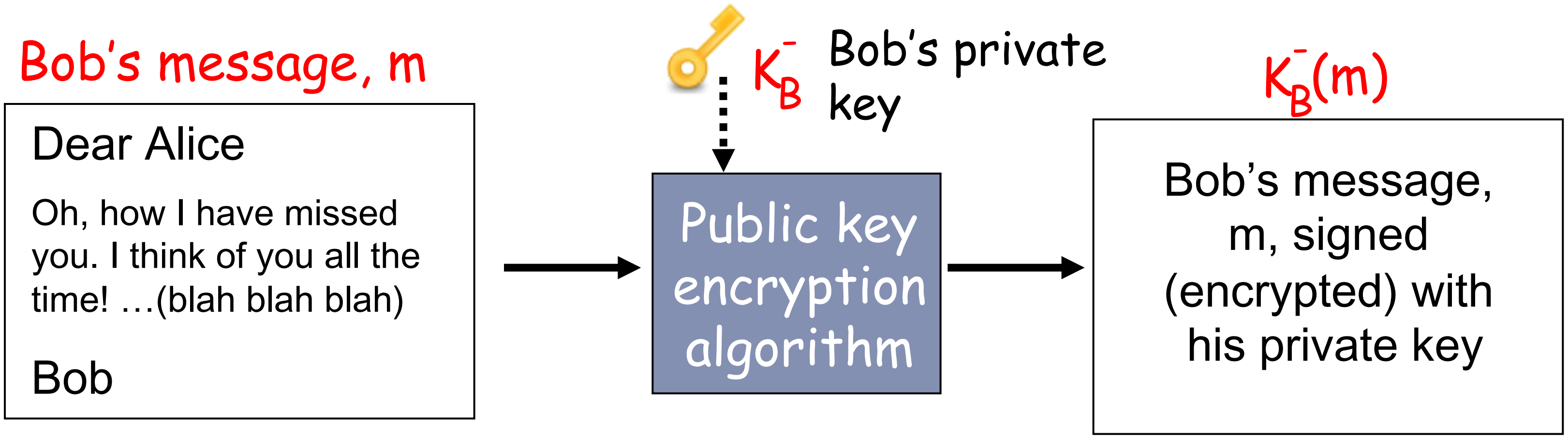
Cryptographic technique analogous to hand-written signatures.

- sender (Bob) digitally signs document, establishing he is document owner/creator.
- verifiable, nonforgeable: recipient (Alice) can prove to someone that Bob, and no one else (including Alice), must have signed document
- Goal is similar to that of a MAC, except now use public-key cryptography

Digital Signatures

Simple digital signature for message m:

- Bob signs m by encrypting with his private key K_B^-, creating "signed" message, $K_B^-(m)$



Dear Alice

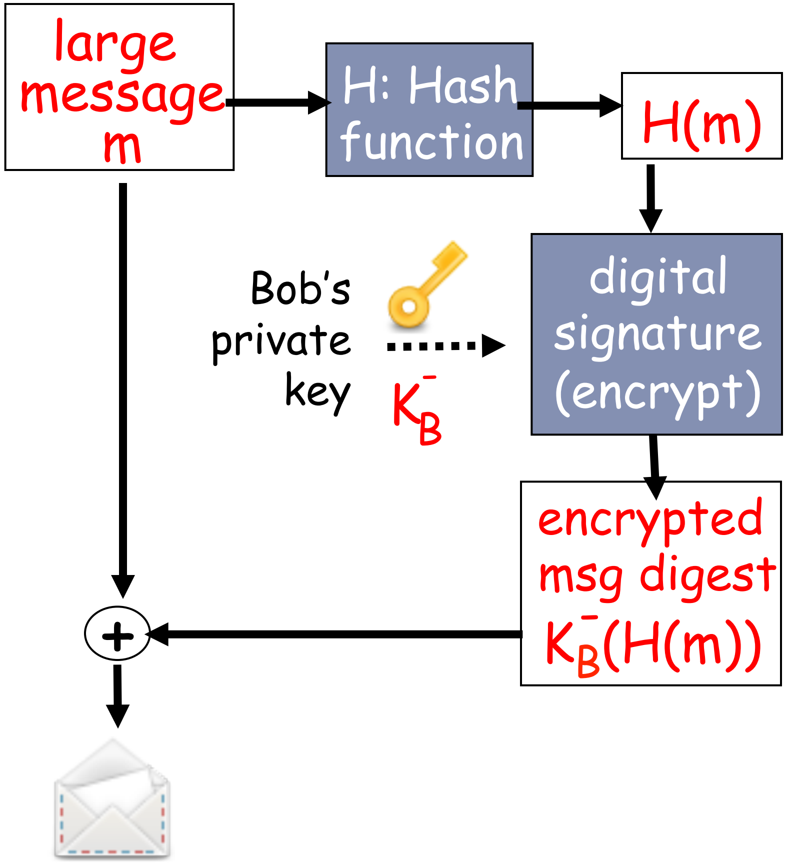
Oh, how I have missed you. I think of you all the time! …(blah blah blah)

Bob

Bob's message, m, signed (encrypted) with his private key

Digital signature = signed message digest

Bob sends digitally signed message:



large message m

H: Hash function

H(m)

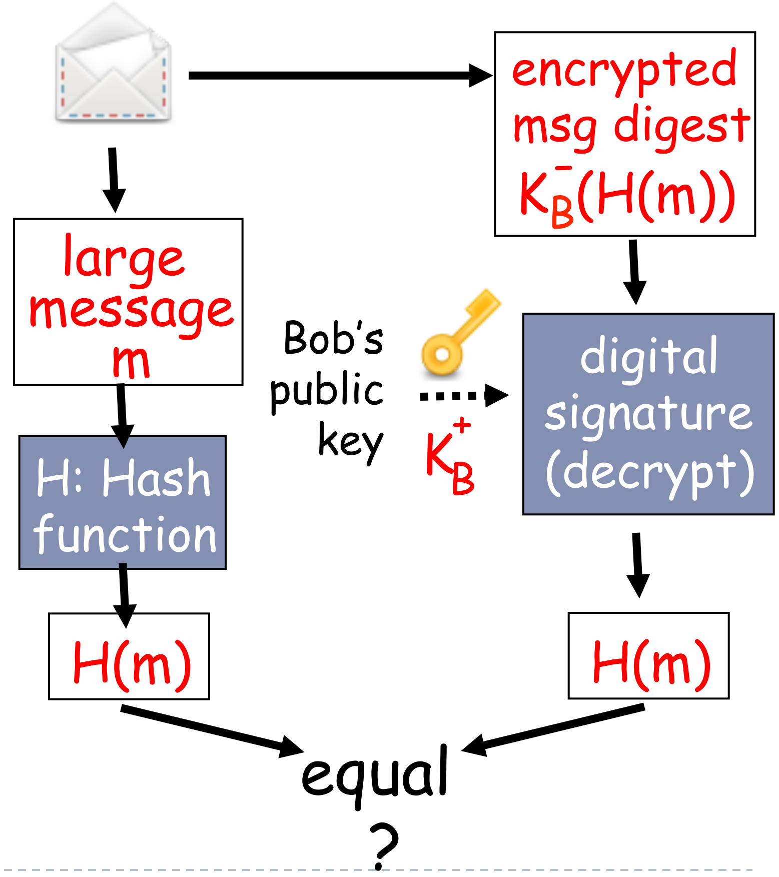
Bob's private key K_B^-

digital signature (encrypt)

encrypted msg digest $K_B^-(H(m))$

+

Alice verifies signature and integrity of digitally signed message:



encrypted msg digest $K_B^-(H(m))$

large message m

H: Hash function

Bob's public key K_B^+

digital signature (decrypt)

H(m)

H(m)

equal ?

Digital Signatures (more)

- Suppose Alice receives msg m, digital signature $K_B^-(m)$
- Alice verifies m signed by Bob by applying Bob's public key K_B to $K_B^+(m)$ then checks $K_B^+(K_B^-(m)) = m$.
- If $K_B^+(K_B^-(m)) = m$, whoever signed m must have used Bob's private key.

Alice thus verifies that:

- Bob signed m.
- No one else signed m.
- Bob signed m and not m'.

Non-repudiation:

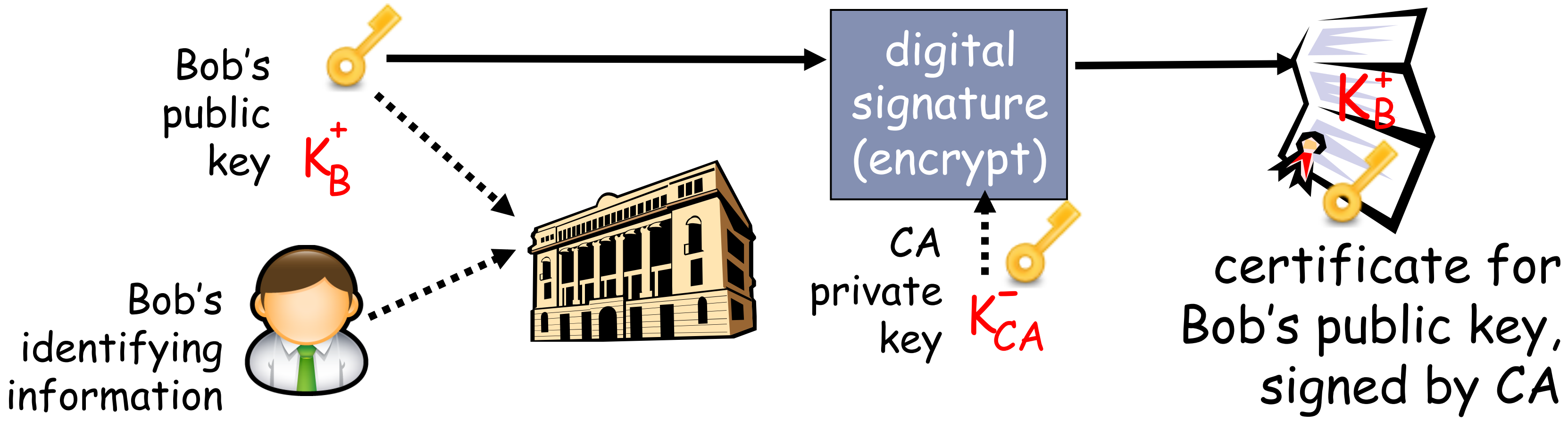
- ✓ Alice can take m, and signature $K_B^-(m)$ to court and prove that Bob signed m.

Public-key certification

- Motivation: Trudy plays pizza prank on Bob
  - Trudy creates e-mail order:
    Dear Pizza Store, Please deliver to me four pepperoni pizzas. Thank you, Bob
  - Trudy signs order with her private key
  - Trudy sends order to Pizza Store
  - Trudy sends to Pizza Store her public key, but says it's Bob's public key.
  - Pizza Store verifies signature; then delivers four pizzas to Bob.
  - Bob doesn't even like Pepperoni

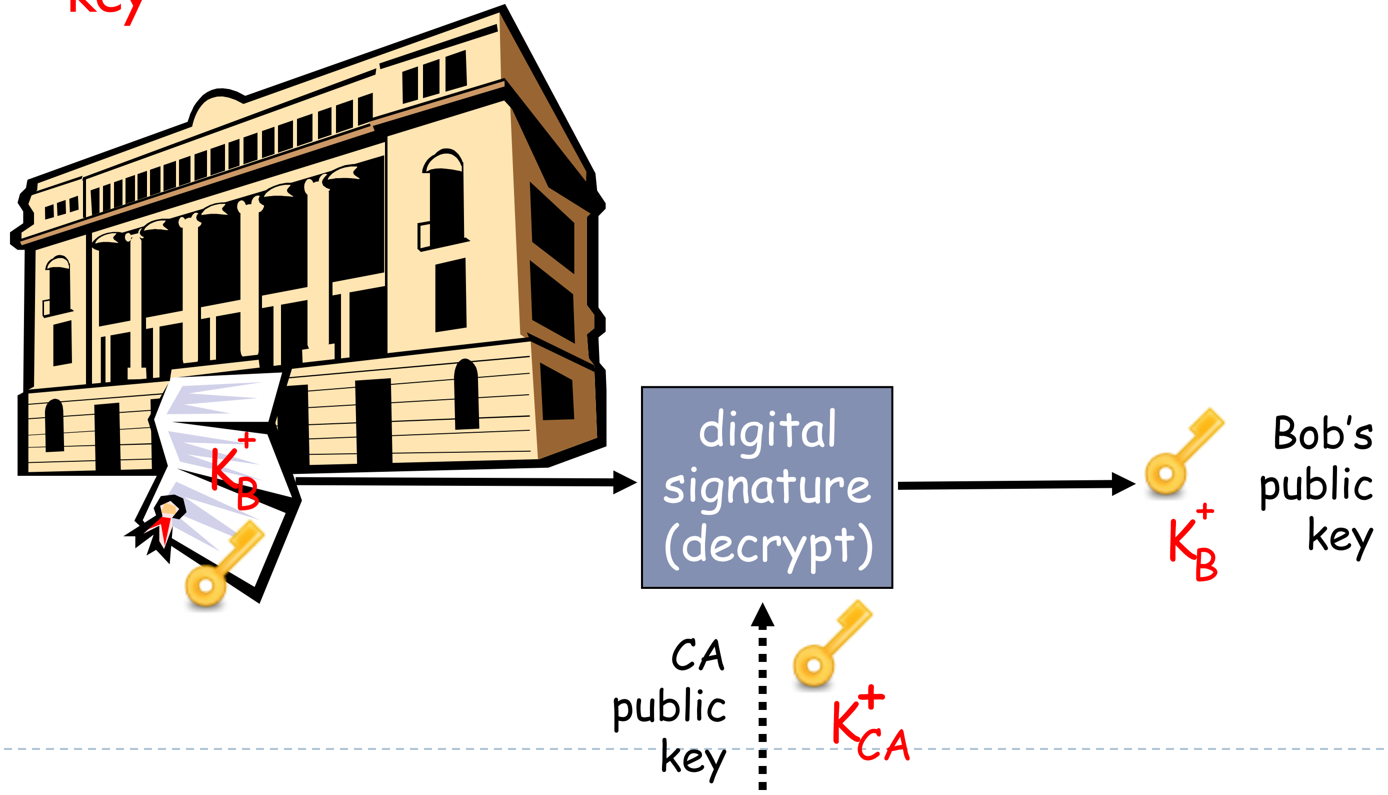
Certification Authorities

- Certification authority (CA): binds public key to particular entity, E.
- E (person, router) registers its public key with CA.
  - E provides "proof of identity" to CA.
  - CA creates certificate binding E to its public key.
  - certificate containing E's public key digitally signed by CA – CA says "this is E's public key"


Bob's public key
K_B^+
Bob's identifying information
digital signature (encrypt)
CA private key
K_{CA}^-
K_B^+
certificate for Bob's public key, signed by CA

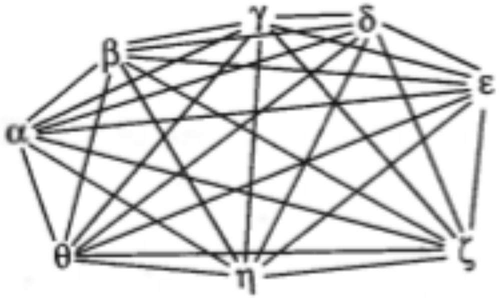
Certification Authorities

- When Alice wants Bob’s public key:
  - gets Bob’s certificate (Bob or elsewhere).
  - apply CA’s public key to Bob’s certificate, get Bob’s public key

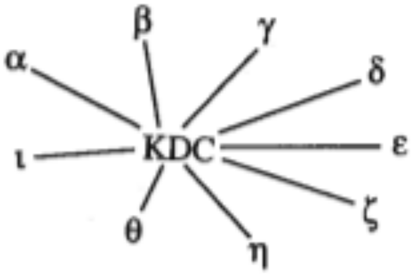


Alternative: symmetric crypto + KDC

- KDC = Key Distribution Center
  - Trusted Node
  - When Alice and Bob want to talk
    - Alice asks KDC for a symmetric session key to be shared with Bob
  - Reduces the number of keys that need to be distributed
    - If a new node joins the network, we need to generate *n* new keys
    - With KDC, only the new node and the KDC need to agree on a key



without KDC



with KDC

Key Exchange via KDC

- Needham-Schroeder protocol
  1. Alice >> KDC : "Alice" | "Bob" | Rand1
  2. KDC >> Alice : Ka("Alice" | "Bob" | Rand1 | Ks | Kb("Alice" | Ks))
  3. Alice >> Bob : Kb("Alice" | Ks)
  4. Bob >> Alice : Ks(Rand2)
  5. Alice >> Bob : Ks(Rand2-1)

See Bishop "Introduction to Computer Security"

KDC vs. CA

- KDC = Key Distribution Center
  - KDC can eavesdrop conversations
  - Single point of failure

- CA = Certification Authority
  - CA signs Alice's and Bob's pub keys
  - CA cannot decrypt communications between Alice and Bob
    - It does not have a copy of their private keys
    - If CA is compromised, attacker cannot gain access to the plaintext
  - Even if CA stops functioning, Alice and Bob can still communicate

Certificates: summary

- Primary standard X.509 (RFC 2459)
- Certificate contains:
  - Issuer name
  - Entity name, address, domain name, etc.
  - Entity's public key
  - Digital signature (signed with issuer's private key)
- Public-Key Infrastructure (PKI)
  - Certificates and certification authorities
  - Certificate Revocation List
  - Often considered "heavy"

Components of a PKI

- Certificates
- Repository from which certificates can be retrieved
- A method for revoking certificates
  - E.g., see https://wiki.mozilla.org/CA:ImprovingRevocation
- An "anchor of trust" (root certificate)
- A method for verifying a chain of certificates up to the anchor of trust

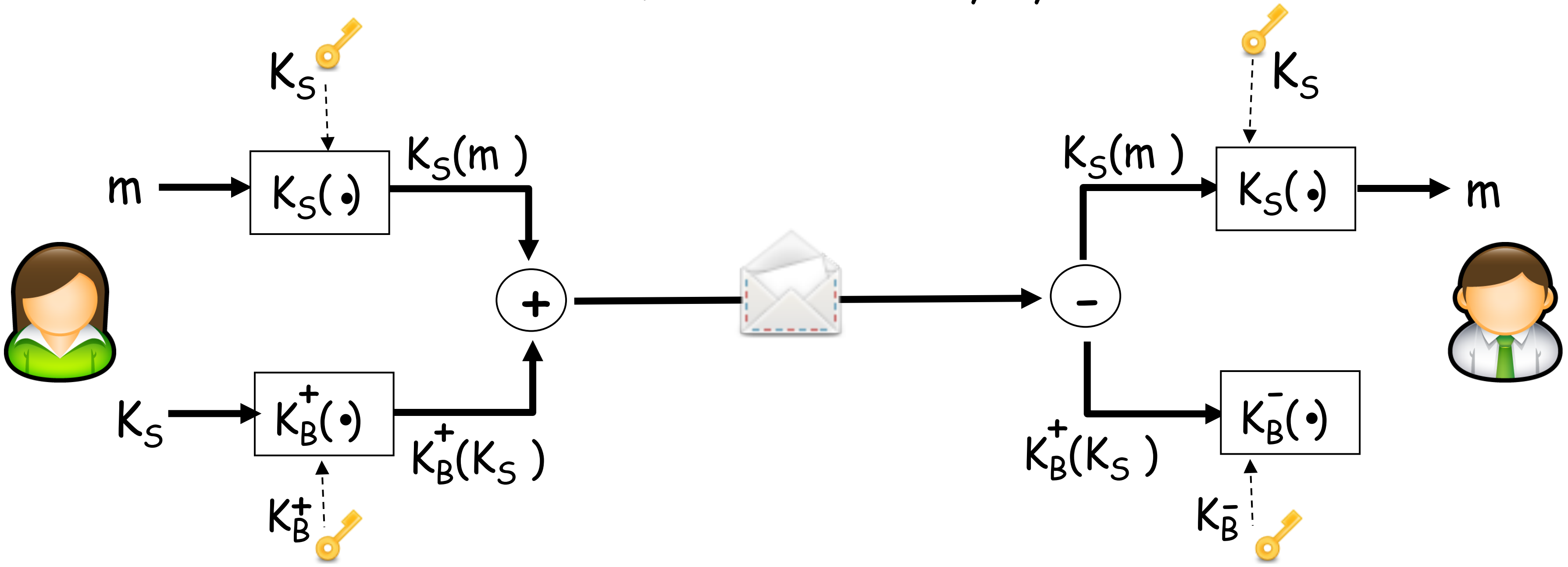
- Browser example:
  - Browsers ship with many trust anchors (i.e., public key of trusted CAs)
- Can we really trust the CAs?
  - http://www.comodo.com/Comodo-Fraud-Incident-2011-03-23.html
  - http://googleonlinesecurity.blogspot.com/2011/08/update-on-attempted-man-in-middle.html
  - It may be possible to trick users to add a trust anchor into the default set
  - The browser itself may be compromised an forced to add a malicious trust anchor

PKI problems

- https://www.eff.org/deeplinks/2011/09/post-mortem-iranian-diginotar-attack
- http://www.zdnet.com/article/trustwave-sold-root-certificate-for-surveillance/
- https://www.eff.org/observatory
  - https://www.eff.org/files/colour_map_of_cas.pdf

Secure e-mail

- Alice wants to send confidential e-mail, m, to Bob.

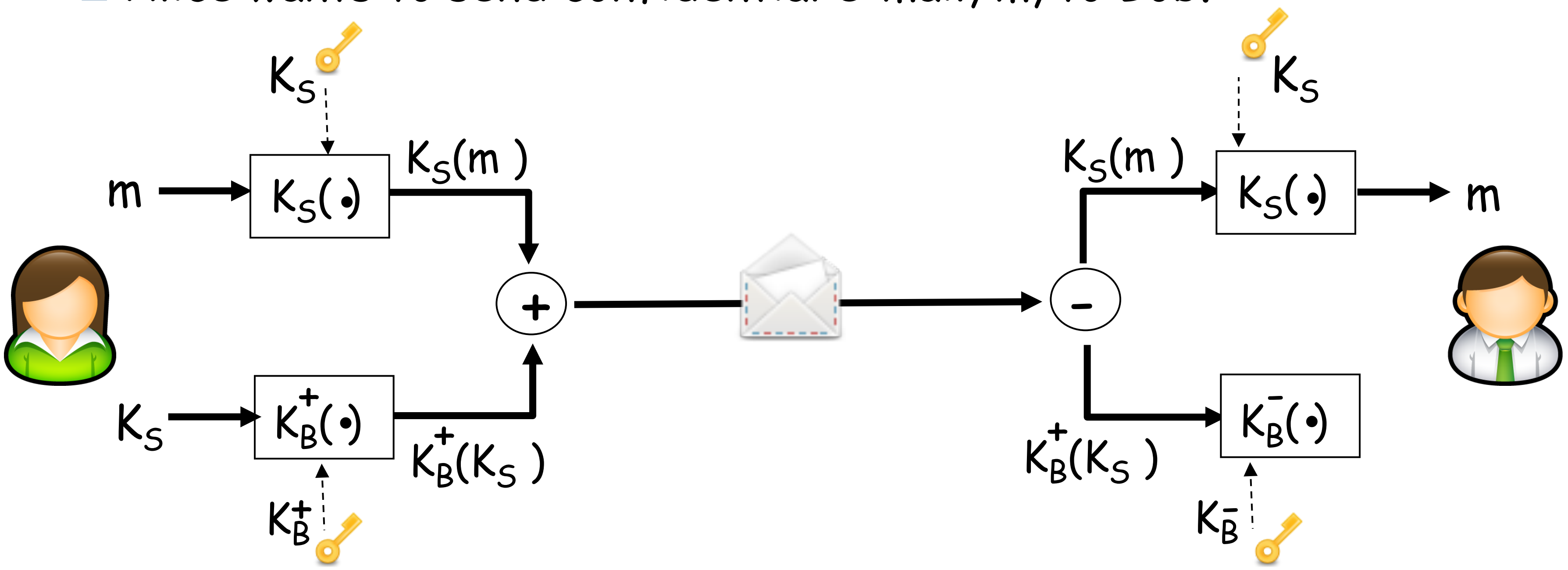


Alice:

- generates random *symmetric* private key, K_S.
- encrypts message with K_S (for efficiency)
- also encrypts K_S with Bob's public key.
- sends both $K_S(m)$ and $K_B(K_S)$ to Bob.

Secure e-mail

- Alice wants to send confidential e-mail, m, to Bob.

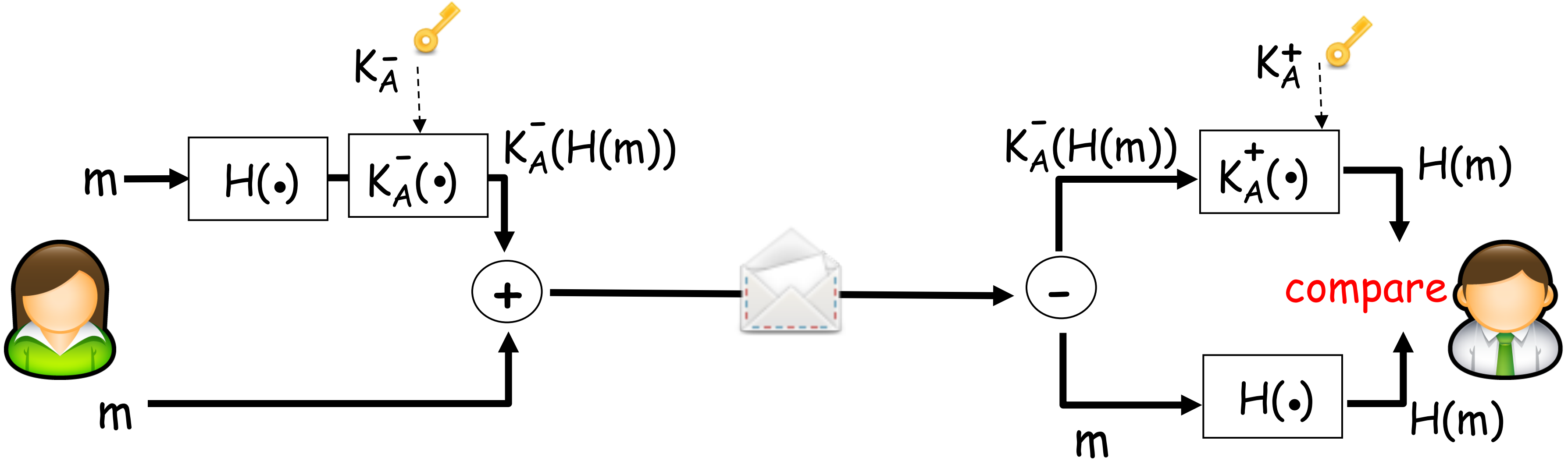


Bob:

- uses his private key to decrypt and recover K_S
- uses K_S to decrypt $K_S(m)$ to recover m

Secure e-mail (continued)

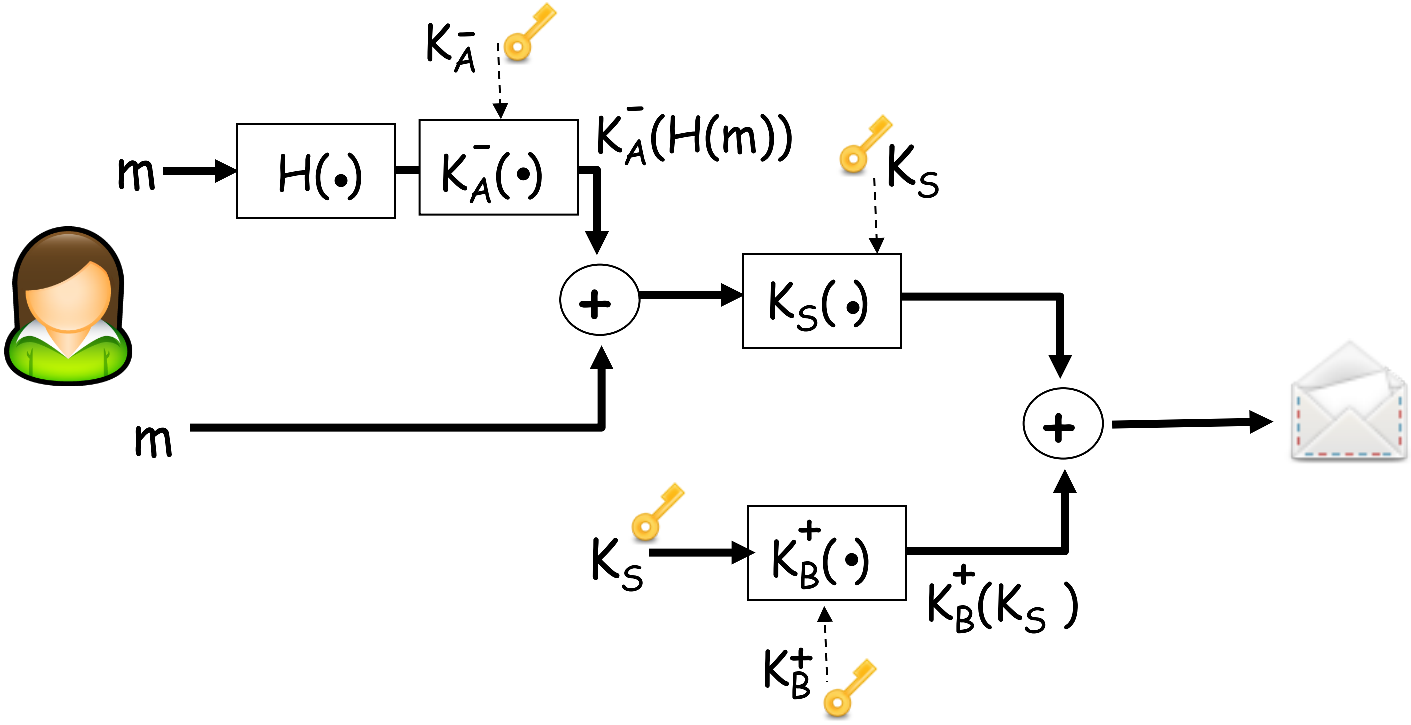
- Alice wants to provide sender authentication message integrity.



- Alice digitally signs message.
- sends both message (in the clear) and digital signature.

Secure e-mail (continued)

- Alice wants to provide secrecy, sender authentication, message integrity.

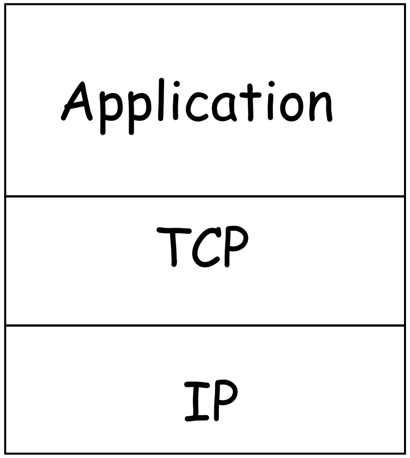


Alice uses three keys: her private key, Bob's public key, newly created symmetric key

SSL: Secure Sockets Layer

- Widely deployed security protocol
  - Supported by almost all browsers and web servers
  - https
  - Tens of billions $ spent per year over SSL
- Originally designed by Netscape in 1993
- Number of variations:
  - TLS: transport layer security, RFC 2246
- Provides
  - Confidentiality
  - Integrity
  - Authentication
- Original goals:
  - Had Web e-commerce transactions in mind
  - Encryption (especially credit-card numbers)
  - Web-server authentication
  - Optional client authentication
  - Minimum hassle in doing business with new merchant
- Available to all TCP applications
  - Secure socket interface

SSL and TCP/IP

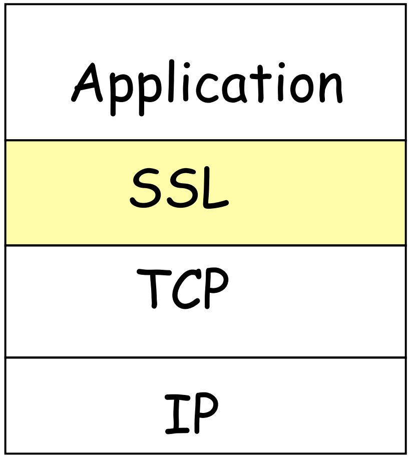


Application

TCP

IP

Normal Application



Application

SSL

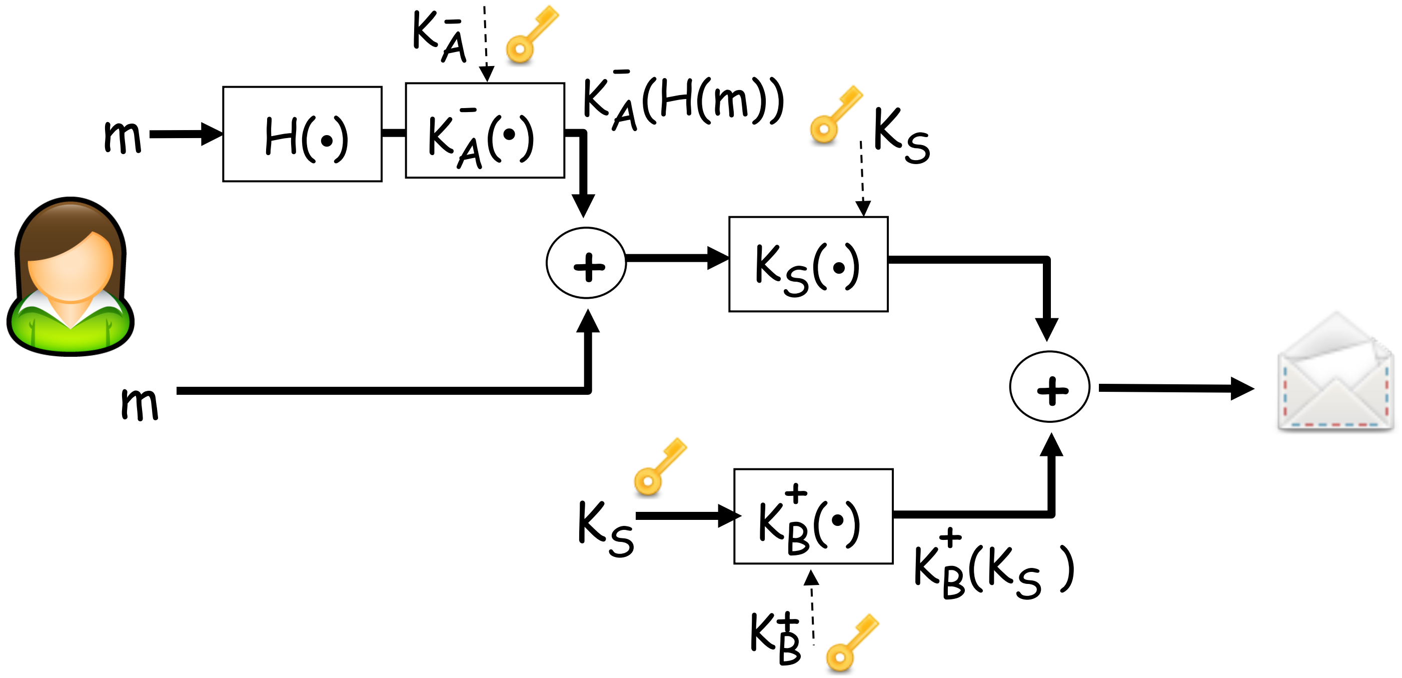
TCP

IP

Application with SSL

- SSL provides application programming interface (API) to applications
- C and Java SSL libraries/classes readily available

Could do something like PGP:

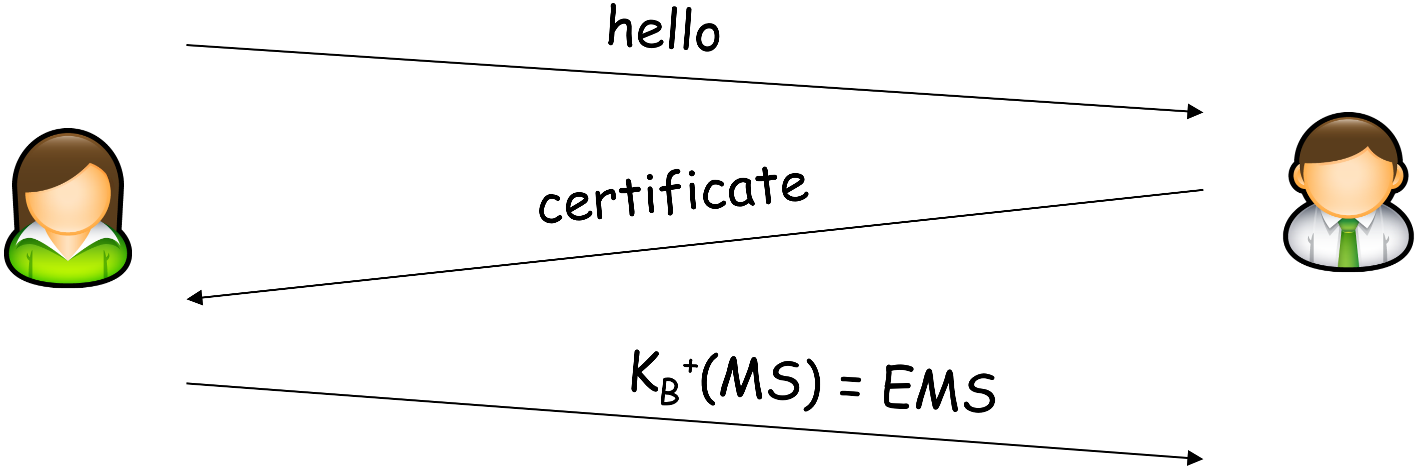


- But want to send byte streams & interactive data
- Want a set of secret keys for the entire connection
- Want certificate exchange part of protocol: handshake phase

SSL: a simple secure channel

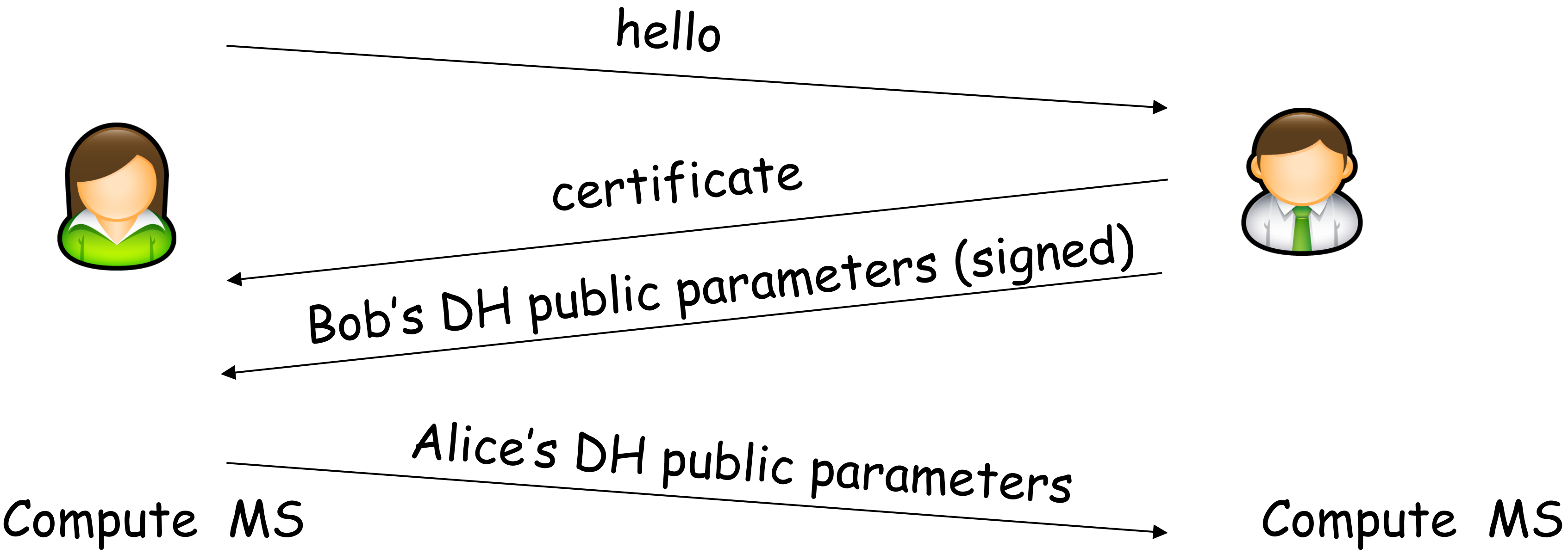
- Handshake: Alice and Bob use their certificates and private keys to authenticate each other and exchange shared secret
  - In most practical cases, only one-way authentication!
- Key Derivation: Alice and Bob use shared secret to derive set of keys
- Data Transfer: Data to be transferred is broken up into a series of records
- Connection Closure: Special messages to securely close connection

A simplified handshake using (RSA)



- MS = master secret
- EMS = encrypted master secret

A simplified handshake (Diffie-Hellman)


hello
certificate
Bob's DH public parameters (signed)
Alice's DH public parameters
Compute MS
Compute MS

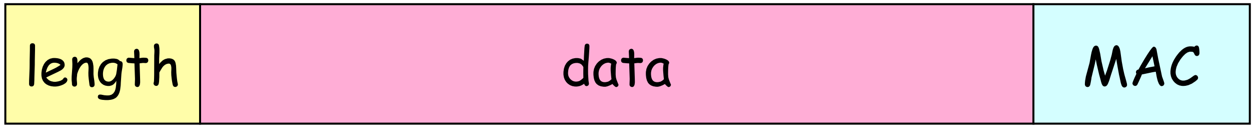
- MS = master secret

Key derivation

- Considered bad to use same key for more than one cryptographic operation
  - Use different keys for message authentication code (MAC) and encryption
- Four keys (both Alice and Bob will have all 4 keys):
  - K_c = encryption key for data sent from client to server
  - M_c = MAC key for data sent from client to server
  - K_s = encryption key for data sent from server to client
  - M_s = MAC key for data sent from server to client
- Keys derived from key derivation function (KDF)
  - Takes master secret and (possibly) some additional random data and creates the keys

Data Records

- Why not encrypt data in constant stream as we write it to TCP?
  - Where would we put the MAC? If at end, no message integrity until all data processed.
  - For example, with instant messaging, how can we do integrity check over all bytes sent before displaying?
- Instead, break stream in series of records
  - Each record carries a MAC
  - Receiver can act on each record as it arrives
- Issue: in record, receiver needs to distinguish MAC from data
  - Want to use variable-length records



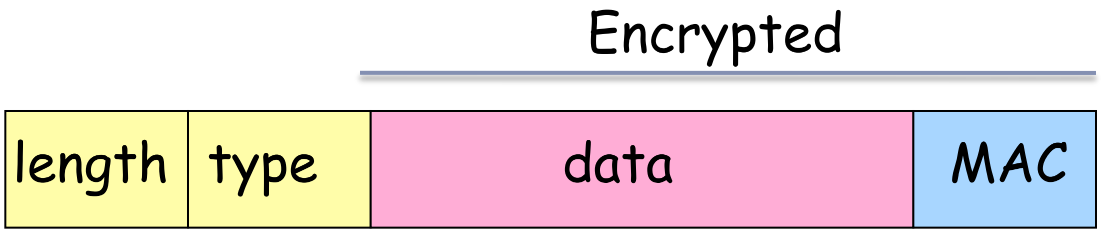
length data MAC

Sequence Numbers

- Attacker can capture and replay or re-order records
- Solution: put sequence number into MAC:
  - MAC = MAC(M_x, sequence||data)
  - Note: no sequence number field
- Attacker could still replay all of the records
  - Use random nonce

Control information

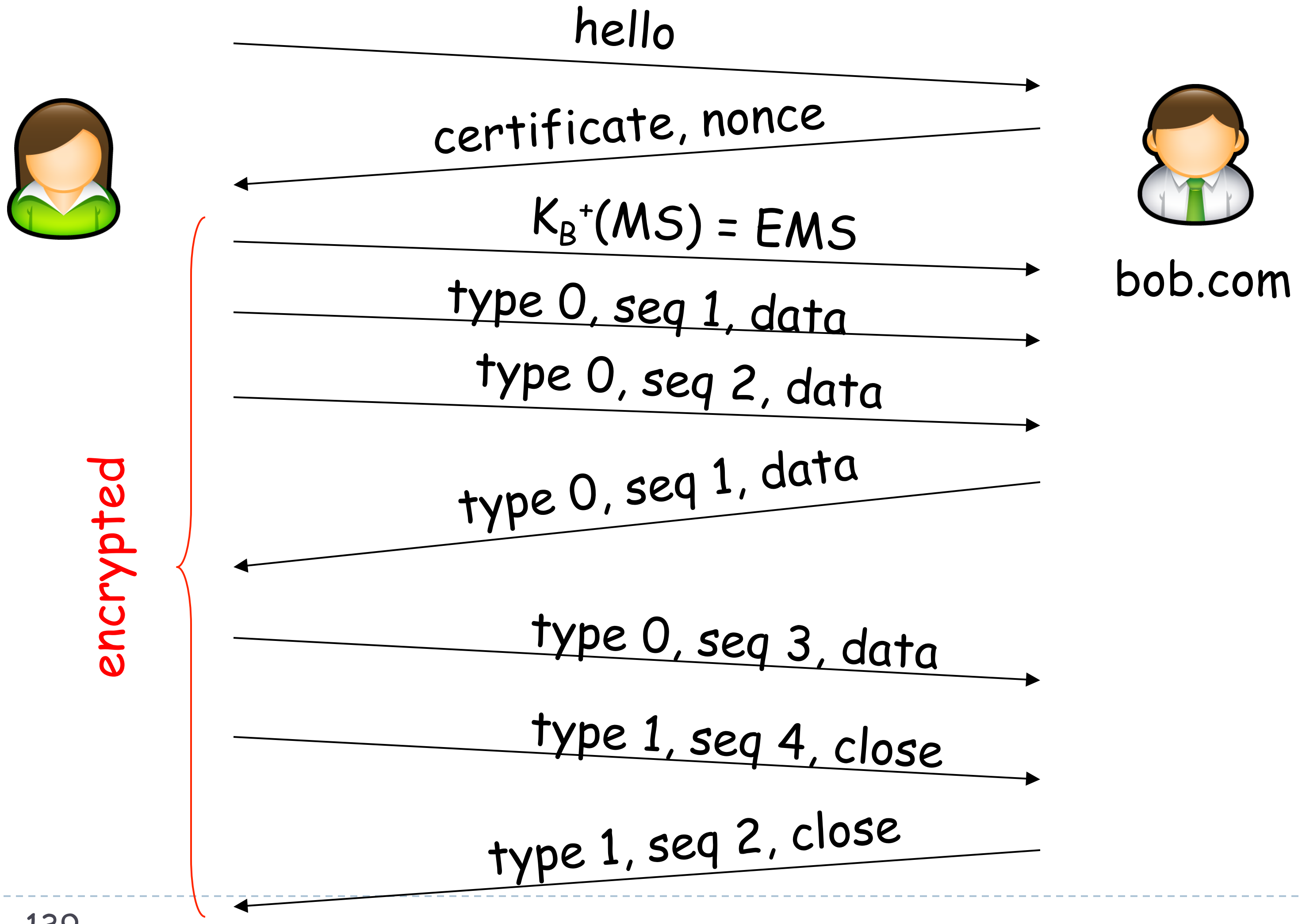
- Truncation attack:
  - attacker forges TCP connection close segment
  - One or both sides thinks there is less data than there actually is.
- Solution: record types, with one type for closure
  - type 0 for data; type 1 for closure
- MAC = MAC(M_x, sequence||type||data)



Encrypted

length | type | data | MAC

SSL: summary ***


hello
certificate, nonce
$K_B^+(MS)$ = EMS
type 0, seq 1, data
type 0, seq 2, data
type 0, seq 1, data
type 0, seq 3, data
type 1, seq 4, close
type 1, seq 2, close
bob.com
encrypted

This version of SSL isn’t complete

- How long are the fields?
- What encryption protocols?
- No negotiation
  - Allow client and server to support different encryption algorithms
  - Allow client and server to choose together specific algorithm before data transfer

Most common symmetric ciphers in SSL

- DES – Data Encryption Standard: block
- 3DES – Triple strength: block
- RC2 – Rivest Cipher 2: block
- RC4 – Rivest Cipher 4: stream

Public key encryption

- RSA

SSL Cipher Suite

- Cipher Suite
  - Public-key algorithm
  - Symmetric encryption algorithm
  - MAC algorithm
- SSL supports a variety of cipher suites
- Negotiation: client and server must agree on cipher suite
- Client offers choices; server picks one

Real SSL: Handshake (1)

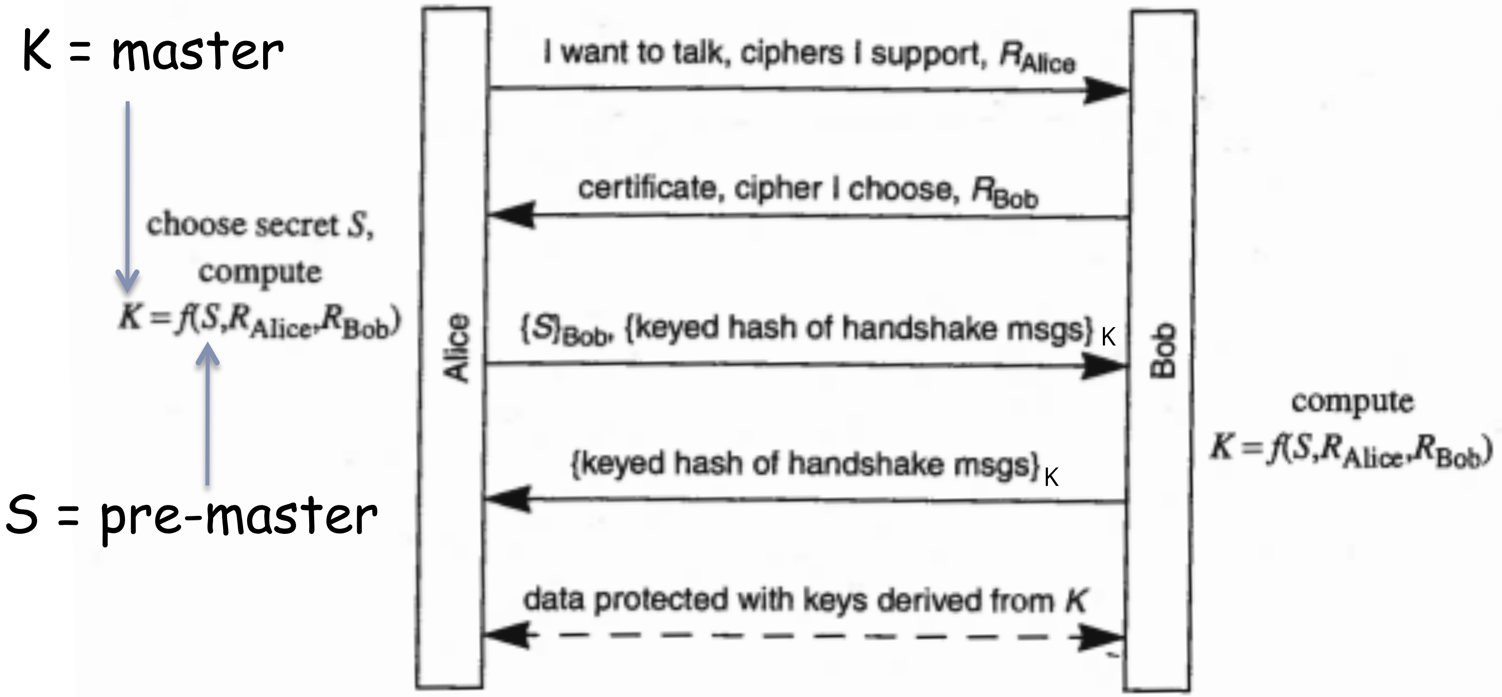
Purpose

1. Server authentication
2. Negotiation: agree on crypto algorithms
3. Establish keys
4. Client authentication (optional)

Real SSL: Handshake (2)

1. Client sends list of algorithms it supports, along with client nonce
2. Server chooses algorithms from list; sends back: choice + certificate + server nonce
3. Client verifies certificate, extracts server's public key, generates pre_master_secret, encrypts with server's public key, sends to server
4. Client and server independently compute encryption and MAC keys from pre_master_secret and nonces
5. Client sends a MAC of all the handshake messages
6. Server sends a MAC of all the handshake messages

Real SSL: Handshake (2)



K = master

S = pre-master

choose secret S, compute $K = f(S, R_{Alice}, R_{Bob})$

Alice

Bob

I want to talk, ciphers I support, R_{Alice}

certificate, cipher I choose, R_{Bob}

$\{S\}_{Bob}$, {keyed hash of handshake msgs}$_K$

compute $K = f(S, R_{Alice}, R_{Bob})$

{keyed hash of handshake msgs}$_K$

data protected with keys derived from K

Protocol 19-1. (simplified) SSLv3/TLS

Real SSL: Handshaking (3)

Last 2 steps protect handshake from tampering

- Client typically offers range of algorithms, some strong, some weak
- Man-in-the middle could delete the stronger algorithms from list
- Last 2 steps prevent this
  - Last two messages are encrypted

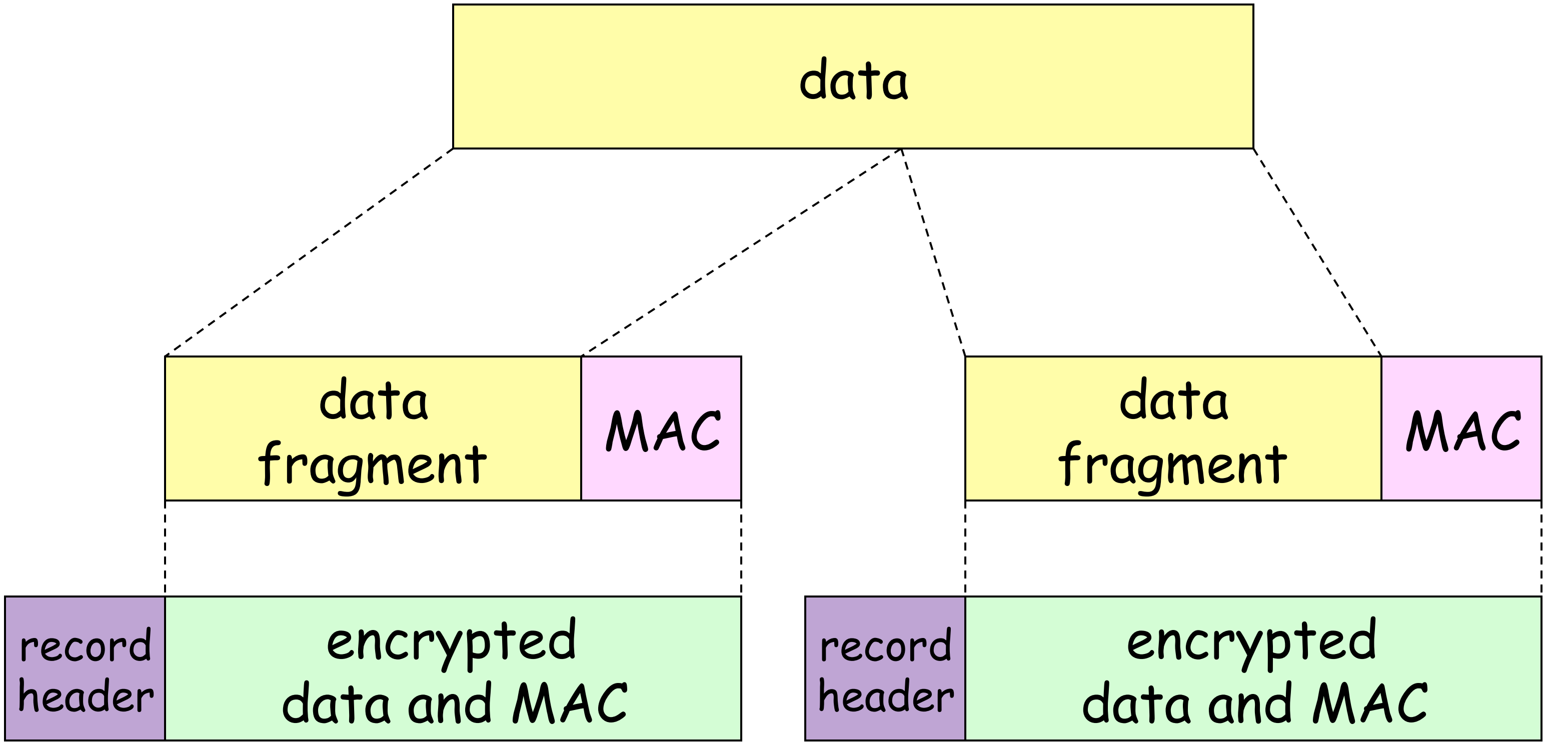
Real SSL: Handshaking (4)

- Why the two random nonces?
- Suppose Trudy sniffs all messages between Alice & Bob.
- Next day, Trudy sets up TCP connection with Bob, sends the exact same sequence of records,.
  - Bob (Amazon) thinks Alice made two separate orders for the same thing.
  - Solution: Bob sends different random nonce for each connection. This causes encryption keys to be different on the two days.
  - Trudy's messages will fail Bob's integrity check.

Real SSL: Key derivation

- Client nonce, server nonce, and pre-master secret input into pseudo random-number generator.
  - Produces master secret
- Master secret and nonces used to generate session keys
  - client MAC key
  - server MAC key
  - client encryption key
  - server encryption key
  - client initialization vector (IV)
  - server initialization vector (IV)

SSL Record Protocol

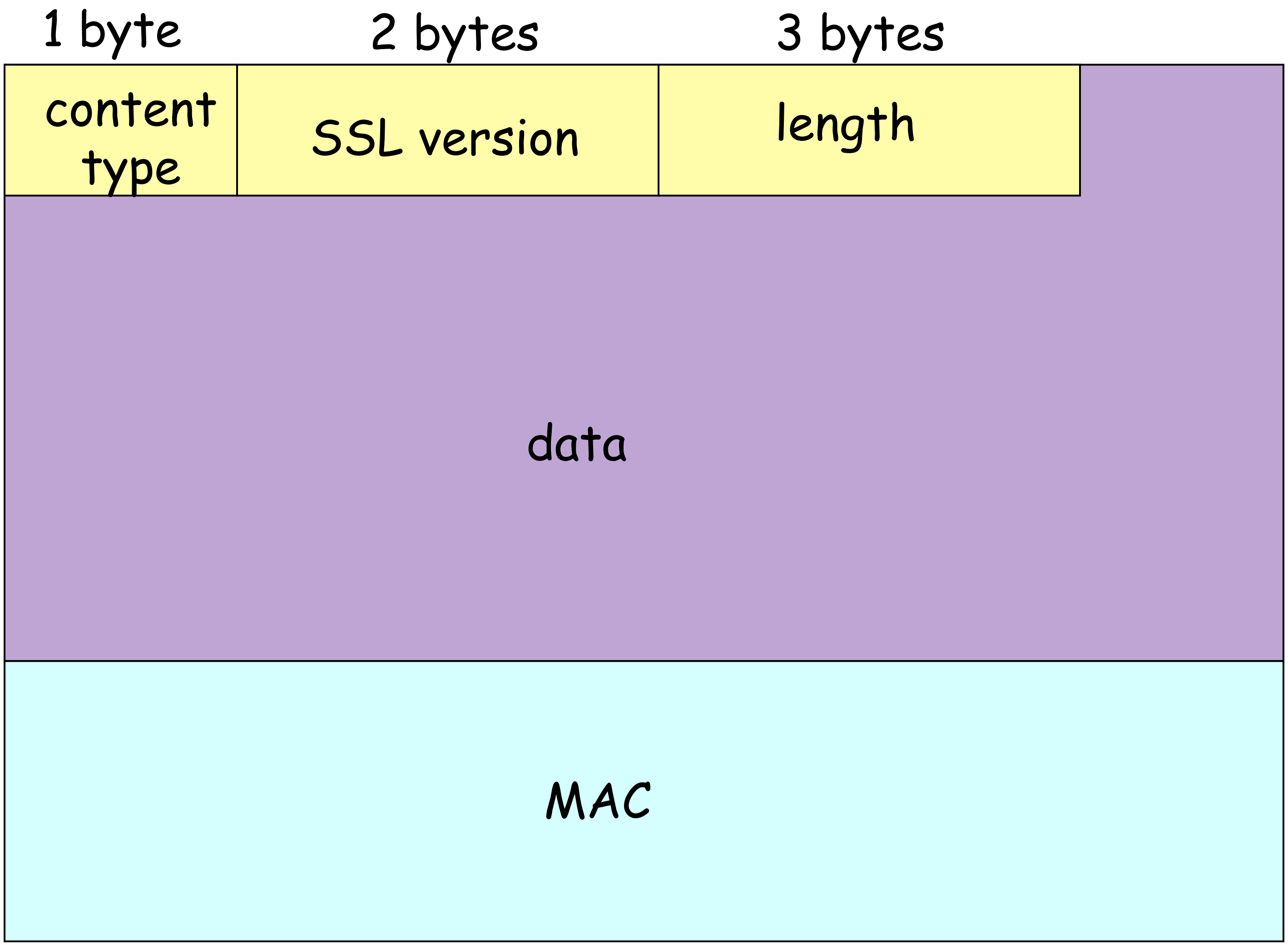


record header: content type; version; length

MAC: includes sequence number, MAC key M_x

Fragment: each SSL fragment max 2^{14} bytes (~16 Kbytes)

SSL Record Format



1 byte

2 bytes

3 bytes

content type

SSL version

length

data

MAC

Data and MAC encrypted (symmetric algo)

Cryptographically protected records

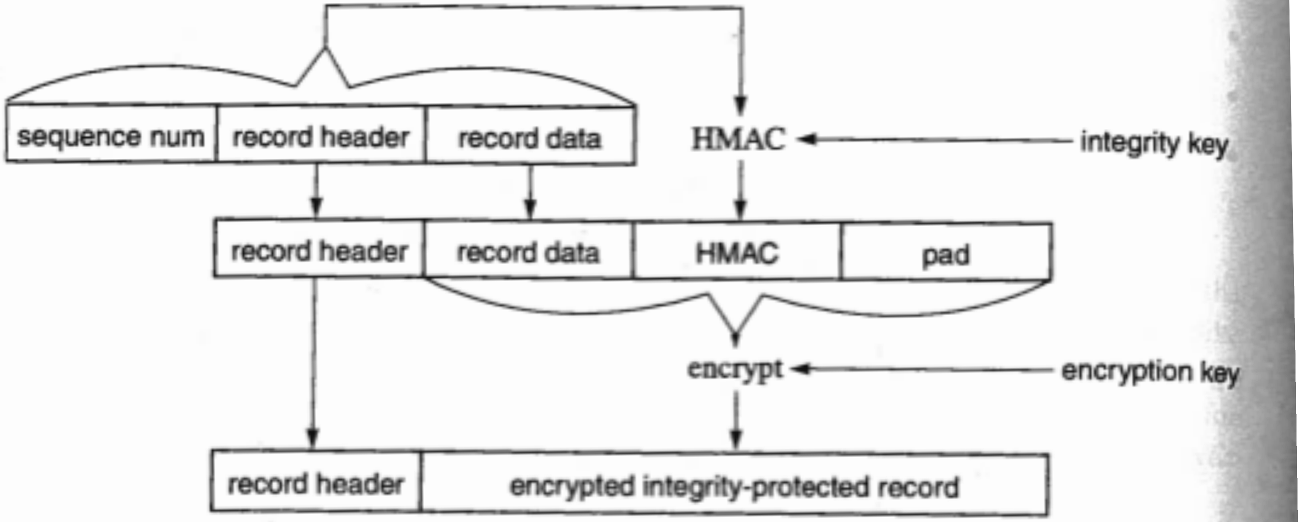
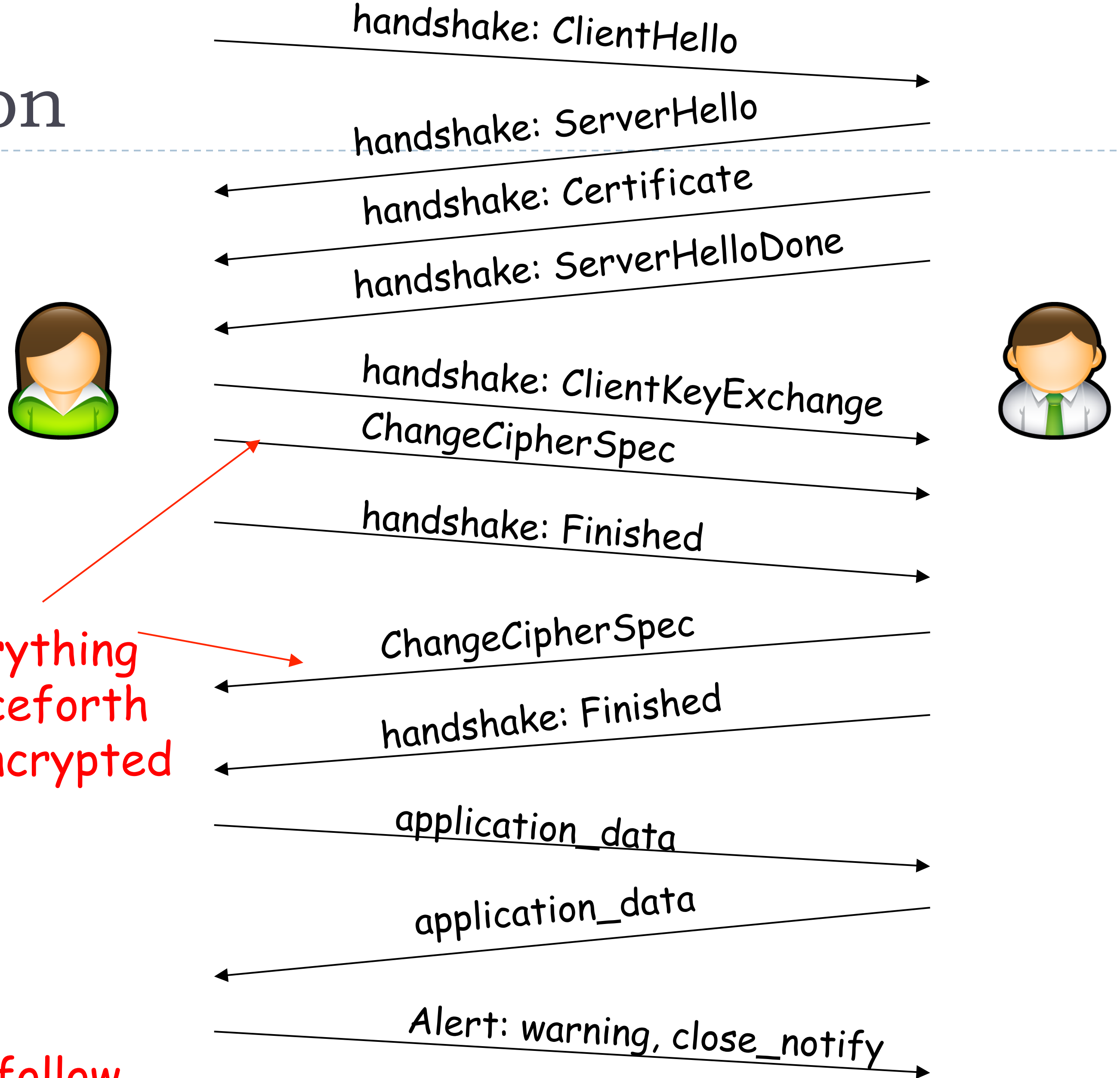


Figure 19-4. Cryptographically protected record format

Sequence number is not explicitly sent, but is part of MAC

Real Connection



handshake: ClientHello

handshake: ServerHello

handshake: Certificate

handshake: ServerHelloDone

handshake: ClientKeyExchange

ChangeCipherSpec

handshake: Finished

ChangeCipherSpec

handshake: Finished

application_data

application_data

Alert: warning, close_notify

Everything henceforth is encrypted

TCP Fin follow

SSL/TLS handshake

- RFC: https://tools.ietf.org/html/rfc2246
- ChangeCipherSpec
  - The change cipher spec message is sent by both the client and server to notify the receiving party that subsequent records will be protected under the newly negotiated CipherSpec and keys
  - TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA
    - DHE = Ephemeral Diffie-Hellman signed with RSA
    - EDE = Encrypt-Decrypt-Encrypt

What is confidentiality at the network-layer?

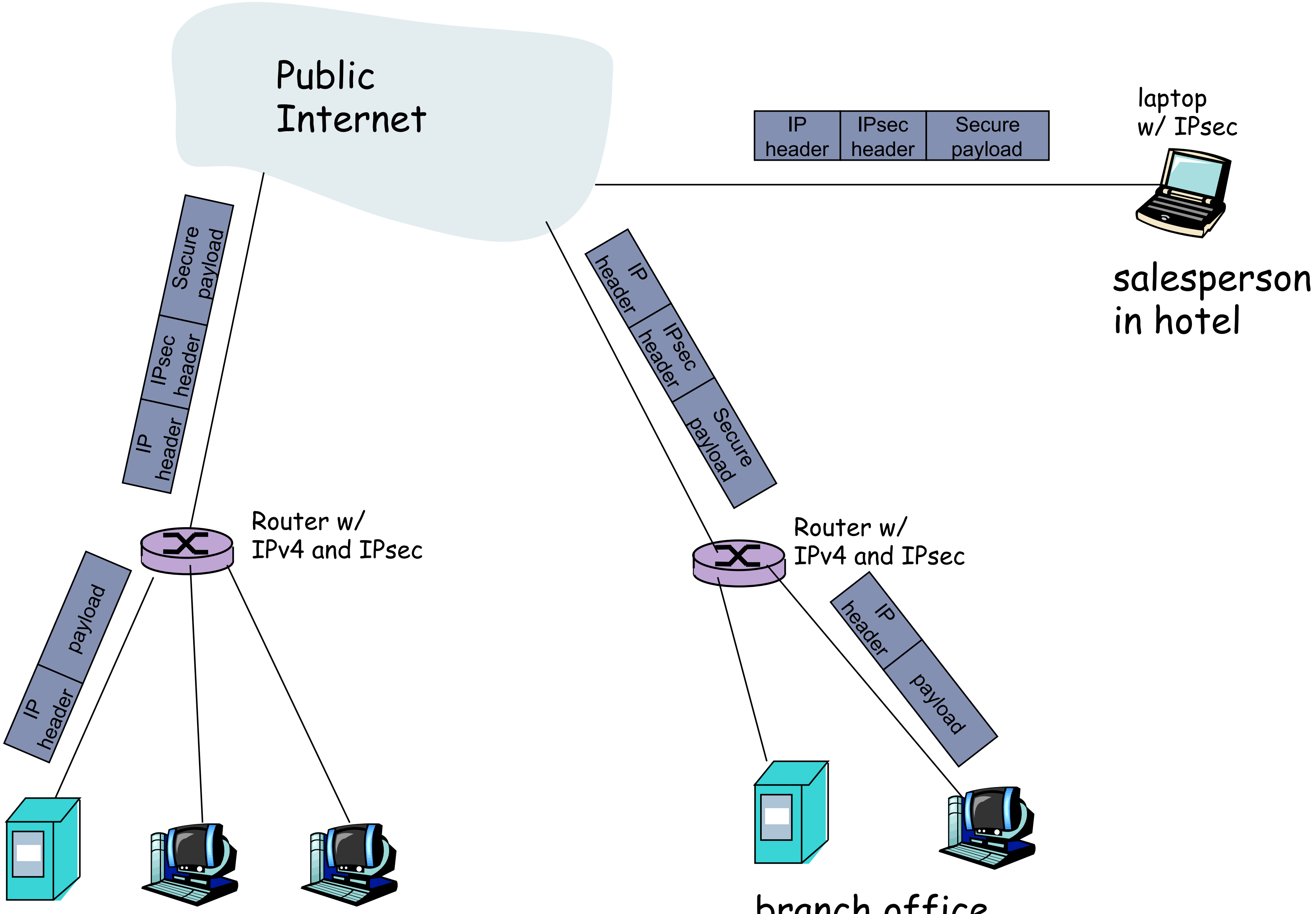
Between two network entities:

- Sending entity encrypts the payloads of datagrams. Payload could be:
  - TCP segment, UDP segment, ICMP message, OSPF message, and so on.
- All data sent from one entity to the other would be hidden:
  - Web pages, e-mail, P2P file transfers, TCP SYN packets, and so on.
- That is, "blanket coverage".

Virtual Private Networks (VPNs)

- Institutions often want private networks for security.
  - Costly! Separate routers, links, DNS infrastructure.
- With a VPN, institution's inter-office traffic is sent over public Internet instead.
  - But inter-office traffic is encrypted before entering public Internet

Virtual Private Network (VPN)



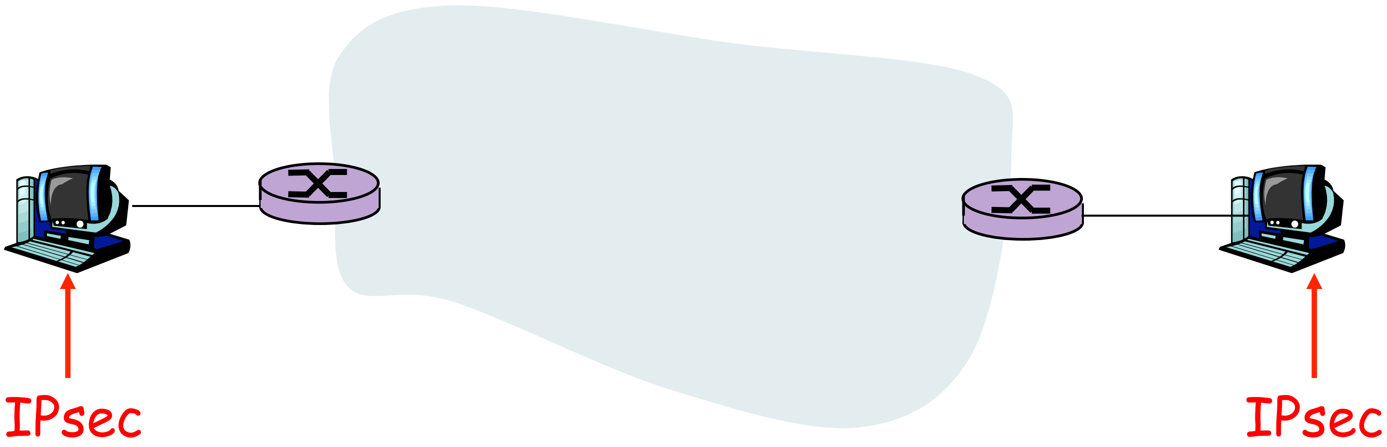
headquarters

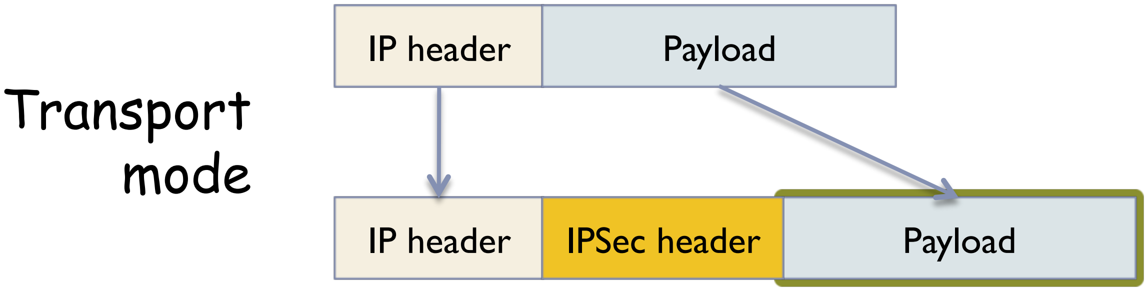
IPsec services

- Confidentiality
- Data integrity
- Origin authentication
- Replay attack prevention

- Two protocols providing different service models:
  - AH = Authentication Header
  - ESP = Encapsulated Security Payload

IPsec Transport Mode





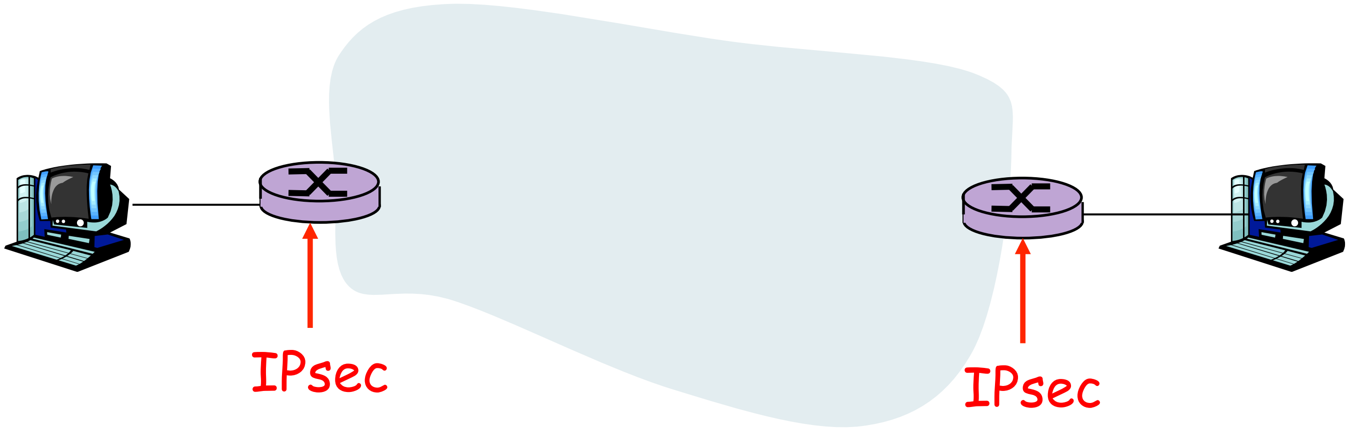
Transport mode

IP header | Payload

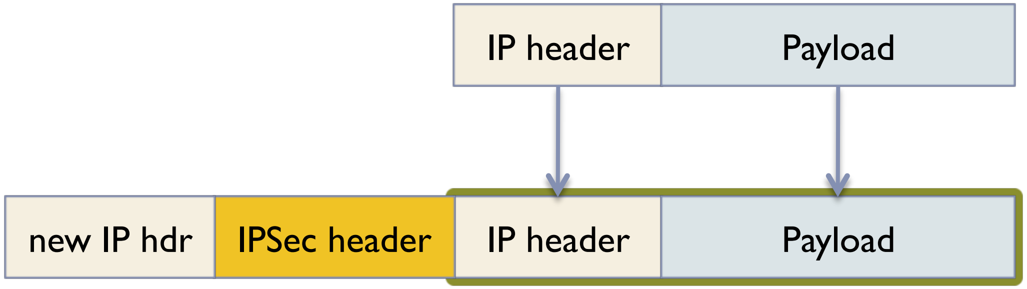
IP header | IPSec header | Payload

- Useful when IPsec is applied end-to-end

IPsec – tunneling mode (1)

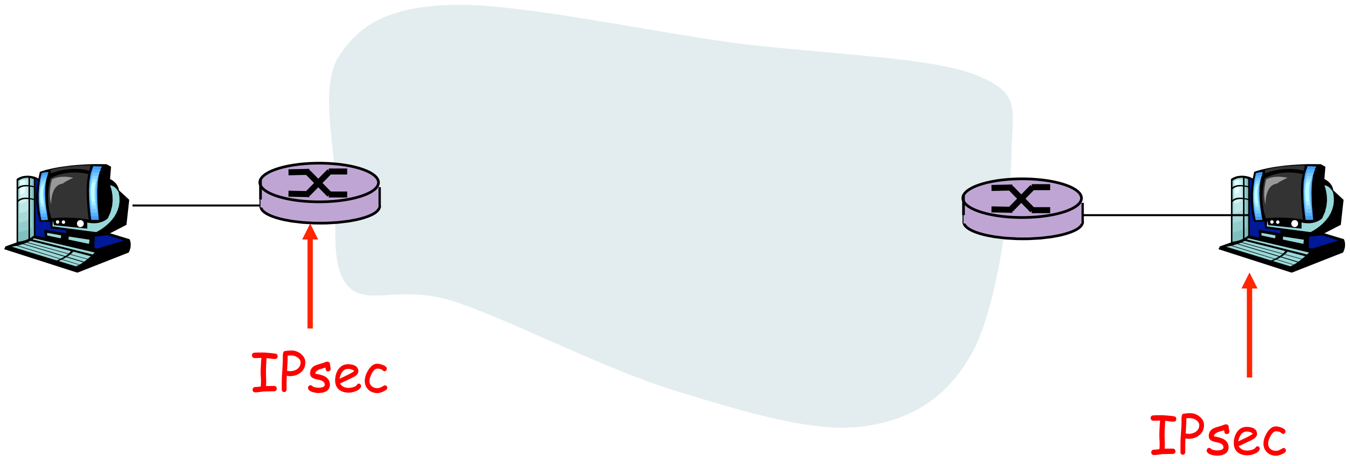


Tunnel mode



- End routers are IPsec aware. Hosts need not be.

IPsec – tunneling mode (2)



- Also tunneling mode.

Two protocols

- Authentication Header (AH) protocol
  - provides source authentication & data integrity but *not* confidentiality
- Encapsulation Security Protocol (ESP)
  - provides source authentication, data integrity, *and confidentiality*
  - more widely used than AH

- Why doe we need AH at all, then?
  - AH does not encrypt the payload
    - Offers integrity protection on payload + part of IP header (excluding TTL, fragment info, etc…), while ESP offers integrity only on payload
    - TCP/UDP header are accessible
  - This works well with firewalls and NAT, which often look at transport layer to decide if/how packets should go through

Four combinations are possible!

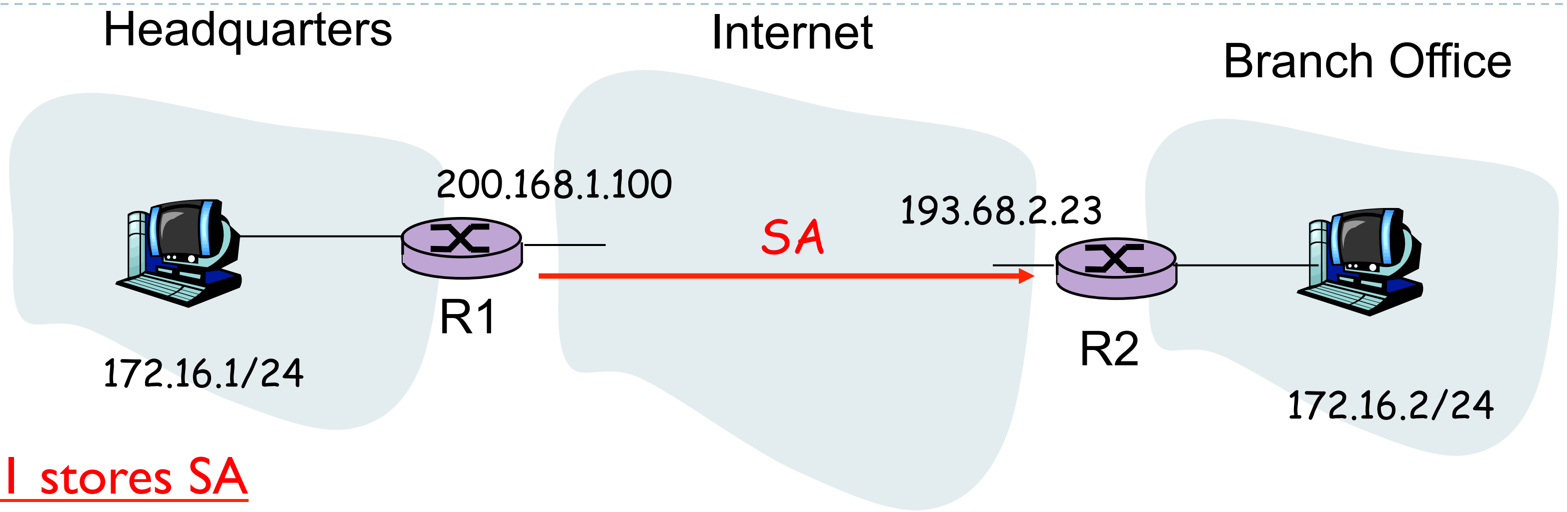
Host mode with AH	Host mode with ESP
Tunnel mode with AH	Tunnel mode with ESP

Most common and most important

Security associations (SAs)

- Before sending data, a virtual connection is established from sending entity to receiving entity.
- Called "security association (SA)"
  - SAs are simplex: for **only one direction**
- Both sending and receiving entities maintain *state information* about the SA
  - Recall that TCP endpoints also maintain state information.
  - IP is connectionless; IPsec is connection-oriented!
- How many SAs in VPN w/ headquarters, branch office, and n traveling salesperson?

Example SA from R1 to R2



R1 stores SA

- 32-bit identifier for SA: *Security Parameter Index (SPI)*
  - *SPI* is included in IPSec header, allows for fast lookups
- the origin interface of the SA (200.168.1.100)
- destination interface of the SA (193.68.2.23)
- type of encryption to be used (for example, 3DES with CBC)
- encryption key
- type of integrity check (for example, HMAC with MD5)
- authentication key

Example SA

Example SA

SPI: 12345
Source IP: 200.168.1.100
Dest IP: 193.68.2.23
Protocol: ESP
Encryption algorithm: 3DES-cbc
HMAC algorithm: MD5
Encryption key: 0x7aeaca…
HMAC key:0xc0291f…

Security Association Database (SAD)

- Endpoint holds state of its SAs in a SAD, where it can locate them during processing.

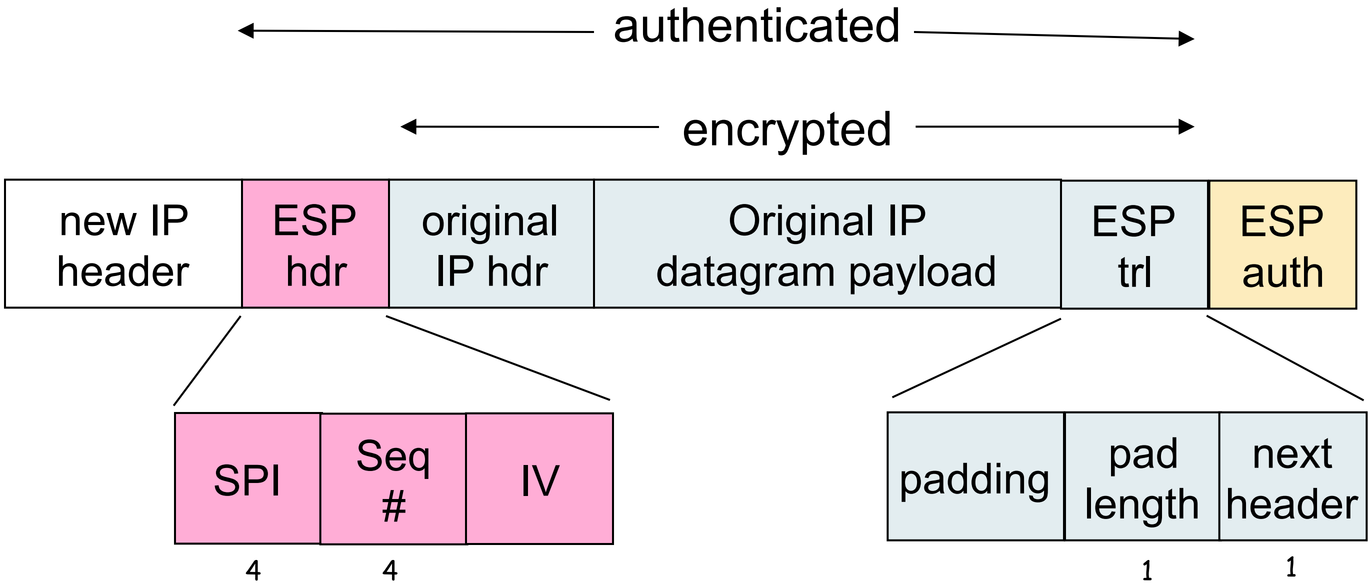
- With branch office and n salespersons
  - Headquarter router stores 2 + 2n SAs in R1's SAD

- When sending IPsec datagram, R1 accesses SAD to determine how to process datagram.

- When IPsec datagram arrives to R2, R2 examines SPI in IPsec datagram, indexes SAD with SPI, and processes datagram accordingly.

IPsec datagram

Focus for now on tunnel mode with ESP



authenticated

encrypted

new IP header | ESP hdr | original IP hdr | Original IP datagram payload | ESP trl | ESP auth

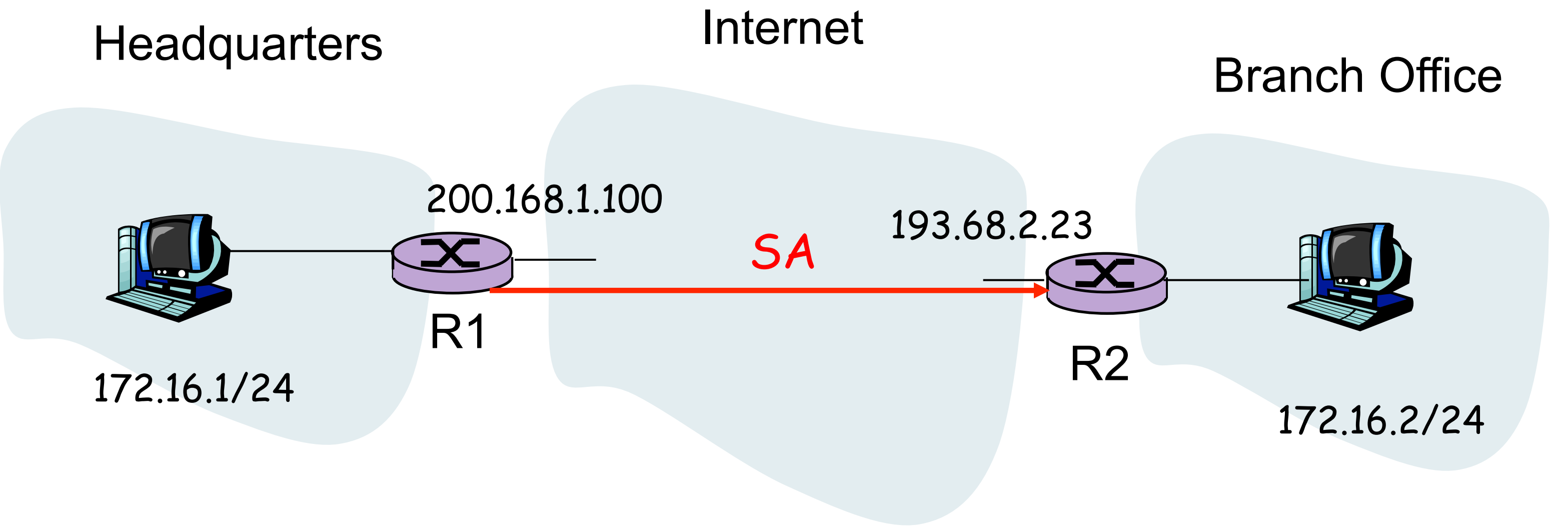
SPI | Seq # | IV

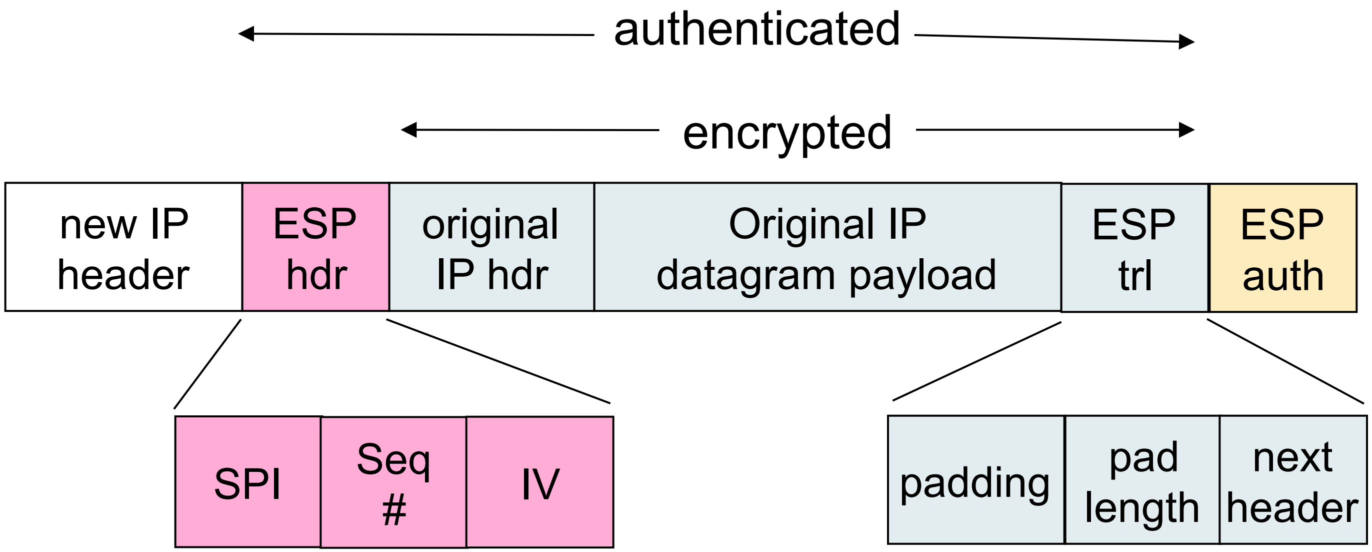
4 4

padding | pad length | next header

1 1

What happens?





new IP header	ESP hdr	original IP hdr	Original IP datagram payload	ESP trl	ESP auth

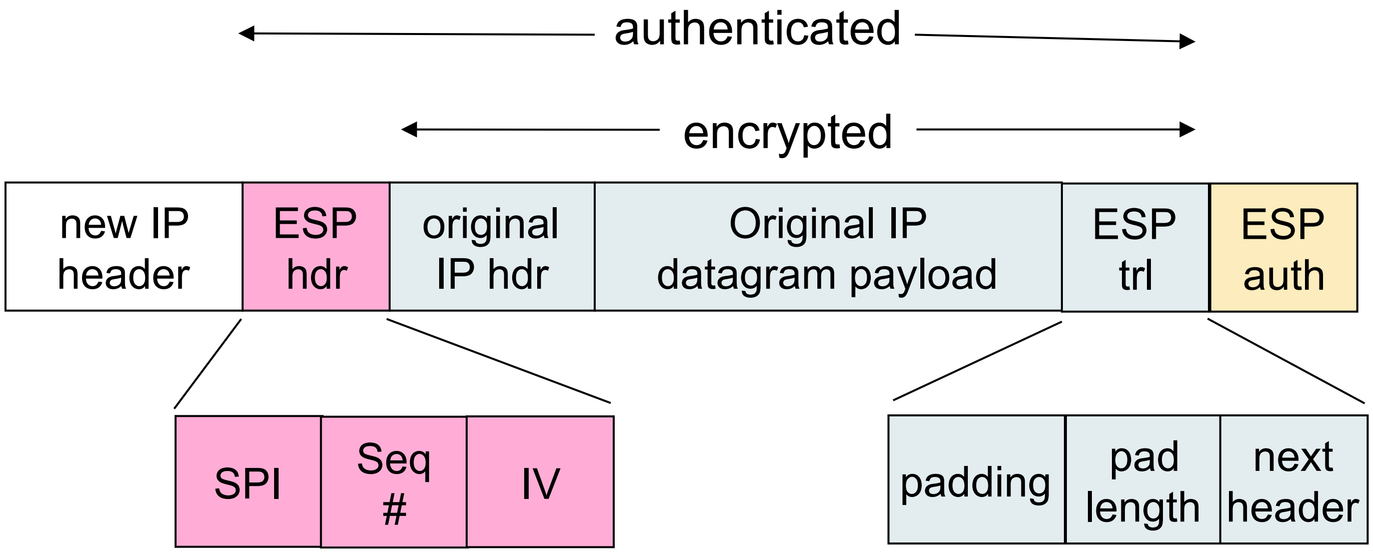
SPI	Seq #	IV

padding	pad length	next header

R1 converts original datagram into IPsec datagram

- Appends an "ESP trailer" field to back of original datagram (which includes original header fields!)
- Encrypts result using algorithm & key specified by SA.
- Appends the "ESP header" to front of this encrypted quantity
- Creates authentication MAC over the obtained datagram, using algorithm and key specified in SA;
- Appends MAC to back, forming *payload*;
- Creates brand new IP header, with all the classic IPv4 header fields, which it appends before payload.

Inside the enchilada:



- ESP trailer: Padding for block ciphers
- ESP header:
  - SPI, so receiving entity knows what to do
  - Sequence number, to thwart replay attacks
- MAC in ESP auth field is created with shared secret key

IPsec sequence numbers

- For new SA, sender initializes seq. # to 0
- Each time datagram is sent on SA:
  - Sender increments seq # counter
  - Places value in seq # field
- Goal:
  - Prevent attacker from sniffing and replaying a packet
    - Receipt of duplicate, authenticated IP packets may disrupt service
- Method:
  - Destination checks for duplicates
  - But doesn't keep track of ALL received packets; instead uses a window

Security Policy Database (SPD)

- Policy: For a given datagram, sending entity needs to know if it should use IPsec.
- Needs also to know which SA to use
  - May use: source and destination IP address; protocol number.
- Info in SPD indicates "what" to do with arriving datagram;
- Info in the SAD indicates "how" to do it.

Summary: IPsec services

- Suppose Trudy sits somewhere between R1 and R2. She doesn't know the keys.
  - Will Trudy be able to see contents of original datagram? How about source, dest IP address, transport protocol, application port?
  - Flip bits without detection?
  - Masquerade as R1 using R1's IP address?
  - Replay a datagram?

Internet Key Exchange

- In previous examples, we assumed the IPsec SAs was manually established (configured) at the endpoints:

Example SA

SPI: 12345

Source IP: 200.168.1.100

Dest IP: 193.68.2.23

Protocol: ESP

Encryption algorithm: 3DES-cbc

HMAC algorithm: MD5

Encryption key: 0x7aeaca…

HMAC key:0xc0291f…

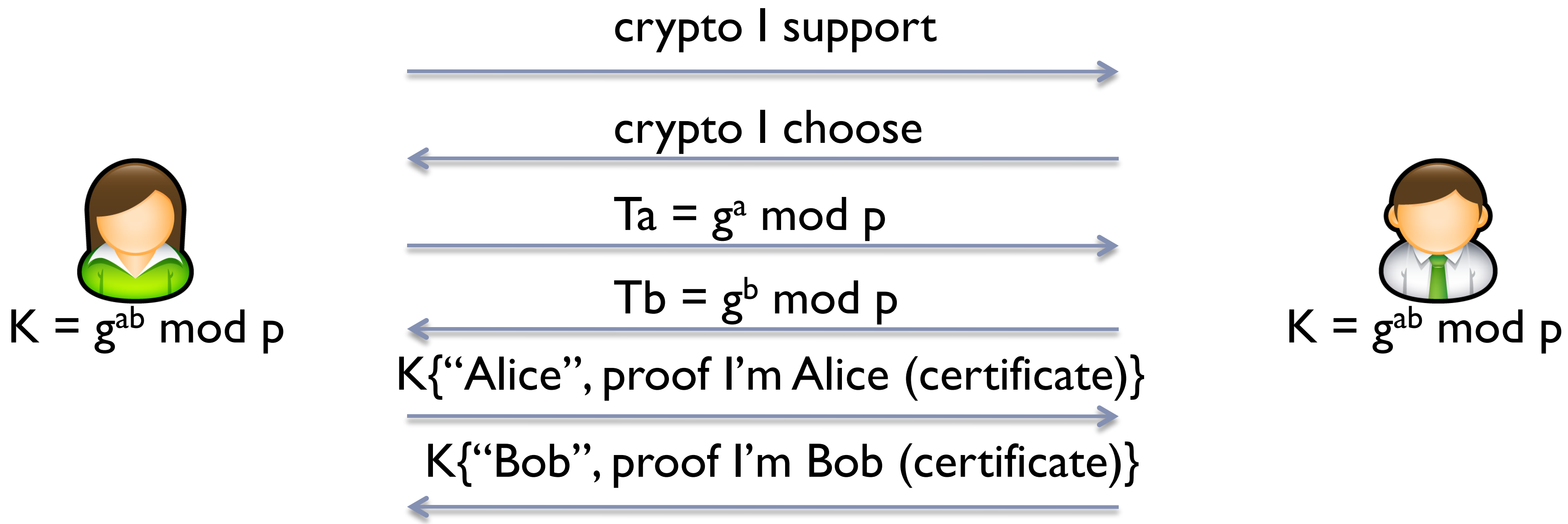
- Such manual keying is impractical for large VPN with, say, hundreds of sales people.
- Instead use *IPsec IKE (Internet Key Exchange)*

IKE Phases

- IKE has two phases
  - Phase 1:
    - Performs mutual authentication and establishment of session keys
    - Also called ISAKMP security association
  - Phase 2:
    - used to securely negotiate the IPsec pair of SAs
    - Sends info used to derive the actual session keys used for ESP/AH

- Phase 1 has two modes: aggressive mode and main mode
  - Aggressive mode uses fewer messages
  - Main mode provides identity protection and is more flexible
    - No party needs to reveal their actual identity in plaintext

IKE Phase-1: Main Mode (simplified)



K = g^{ab} mod p

crypto I support

crypto I choose

Ta = g^a mod p

Tb = g^b mod p

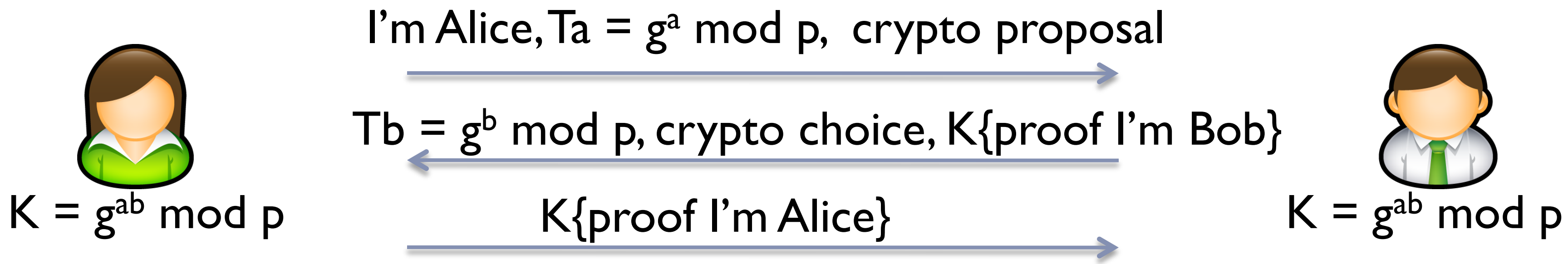
K{“Alice”, proof I’m Alice (certificate)}

K{“Bob”, proof I’m Bob (certificate)}

K = g^{ab} mod p

http://tools.ietf.org/html/rfc2409

IKE Phase-1: Aggressive Mode (simplified)



I'm Alice, Ta = g^a mod p, crypto proposal

Tb = g^b mod p, crypto choice, K{proof I'm Bob}

K{proof I'm Alice}

K = g^{ab} mod p

K = g^{ab} mod p

http://tools.ietf.org/html/rfc2409

IKE: PSK and PKI

- Authentication (proof of who you are) with either
  - pre-shared secret (PSK) or
  - with PKI (pubic/private keys and certificates).
- With PSK, both sides start with secret:
  - then run IKE to authenticate each other and to generate IPsec SAs (one in each direction), including encryption and integrity keys
- With PKI, both sides start with public/private key pair and certificate.
  - run IKE to authenticate each other and obtain IPsec SAs (one in each direction).
  - Similar to handshake in SSL.

IKE Phase-1
Signature vs. Public Key Encryption

- Signature
  - Does not require Alice to know Bob's pub key in advance
  - She will receive Bob's certificate in the last message
  - Identity may be revealed to an attacker who is trying to impersonate one of the parties

- Pub key encryption
  - Alice must know Bob's pub key
  - Both sides reveal their identity only to whom they intend to authenticate themselves

Summary of IPsec

- IKE message exchange for algorithms, secret keys, SPI numbers
- Either the AH or the ESP protocol (or both)
- The AH protocol provides integrity and source authentication
- The ESP protocol (with AH) additionally provides encryption
- IPsec peers can be two end systems, two routers/firewalls, or a router/firewall and an end system

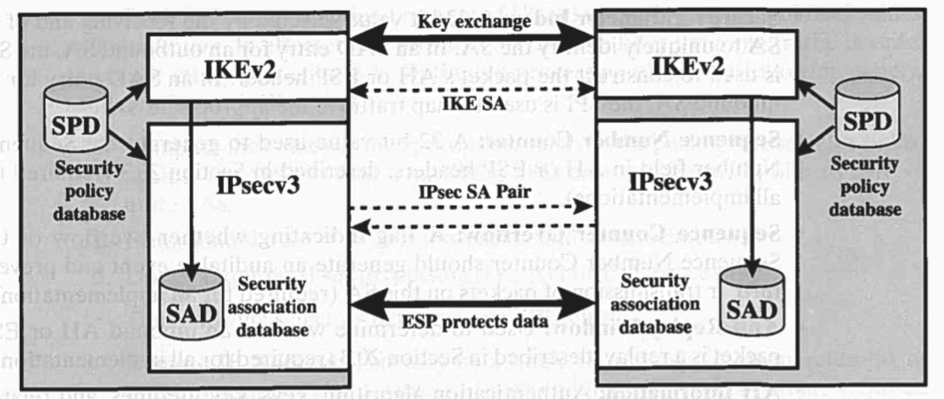


Figure 20.2 IPsec Architecture

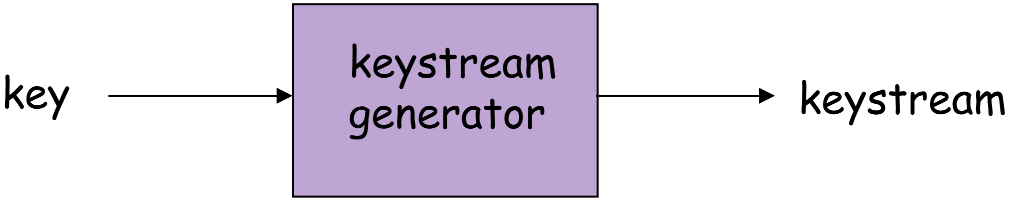
Source: Stallings – "Cryptography and Network Security, Principles and Practice"

Wired Equivalent Privacy

WEP Design Goals

- Symmetric key crypto
  - Confidentiality
  - Station authorization
  - Data integrity
- Self synchronizing: each packet separately encrypted
  - Given encrypted packet and key, can decrypt; can continue to decrypt packets when preceding packet was lost
  - Unlike Cipher Block Chaining (CBC) in block ciphers
- Efficient
  - Can be implemented in hardware or software

Review: Symmetric Stream Ciphers



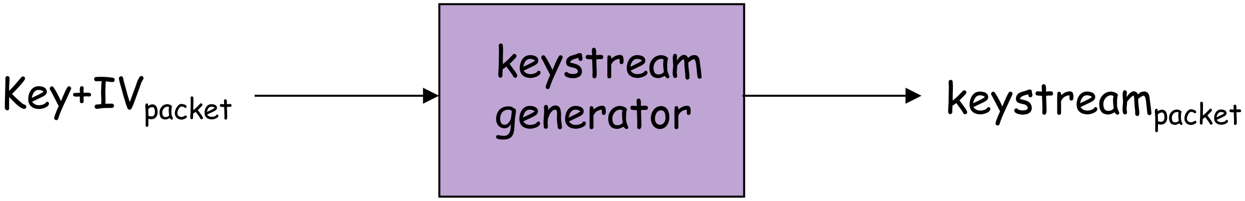
- Combine each byte of keystream with byte of plaintext to get ciphertext
- m(i) = ith unit of message
- ks(i) = ith unit of keystream
- c(i) = ith unit of ciphertext
- $c(i) = ks(i) \oplus m(i)$ ($\oplus$ = exclusive or)
- $m(i) = ks(i) \oplus c(i)$
- WEP uses RC4

Attacks on Stream Ciphers

- Repetition attack
  - if key stream reused, attacker obtains XOR of two plaintexts (P1 xor P2)
  - If P1 is known, P2 is also known
  - Even if no plaintext is known, there are known attacks based on PI xor P2 (e.g., frequency attacks)

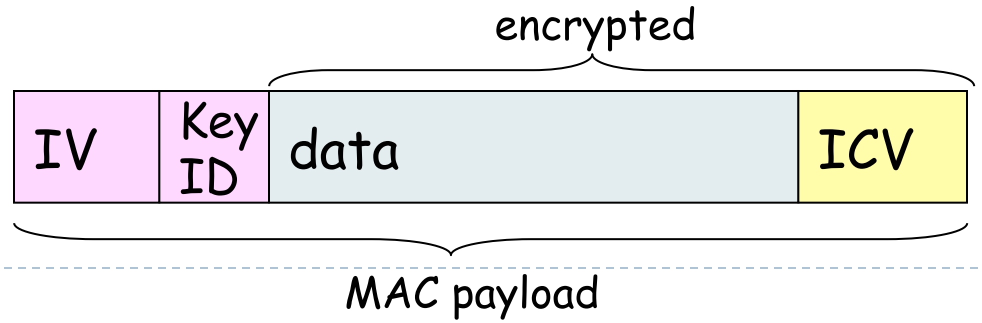
Stream cipher and packet independence

- Recall design goal: each packet separately encrypted
- If for frame n+1, use keystream from where we left off for frame n, then each frame is not separately encrypted
  - Need to know where we left off for packet n
- WEP approach: initialize keystream with key + new IV for each packet:

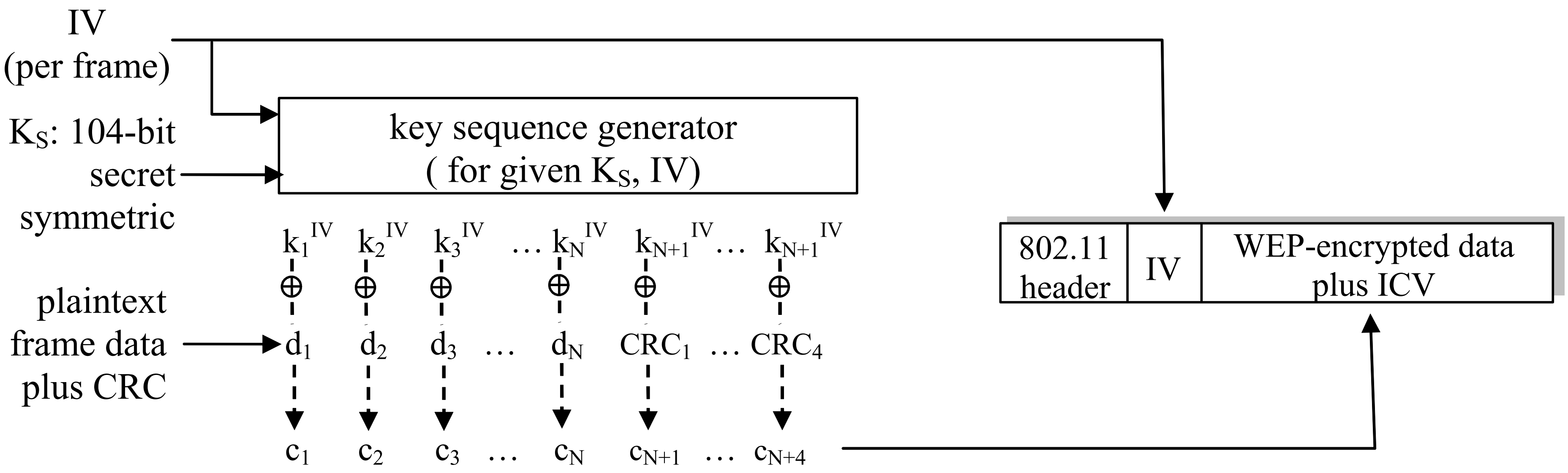

$\text{Key+IV}_{\text{packet}}$
keystream generator
$\text{keystream}_{\text{packet}}$

WEP encryption (1)

- Sender calculates Integrity Check Value (ICV) over data
  - four-byte hash/CRC for data integrity
- Each side has 104-bit shared key
- Sender creates 24-bit initialization vector (IV), appends to key: gives 128-bit key
- Sender also appends keyID (in 8-bit field)
- 128-bit key inputted into pseudo random number generator to get keystream
- data in frame + ICV is encrypted with RC4:
  - Bytes of keystream are XORed with bytes of data & ICV
  - IV & keyID are appended to encrypted data to create payload
  - Payload inserted into 802.11 frame

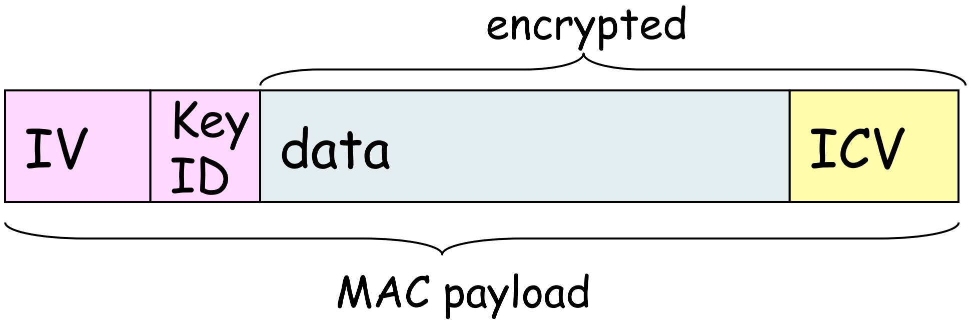


WEP encryption (2)


IV
(per frame)
K_S: 104-bit secret symmetric
key sequence generator
(for given K_S, IV)
plaintext frame data plus CRC
k_1^{IV} k_2^{IV} k_3^{IV} … k_N^{IV} k_{N+1}^{IV} … k_{N+1}^{IV}
⊕ ⊕ ⊕ ⊕ ⊕ ⊕
d_1 d_2 d_3 … d_N CRC_1 … CRC_4
c_1 c_2 c_3 … c_N c_{N+1} … c_{N+4}
802.11 header
IV
WEP-encrypted data plus ICV

New IV for each frame

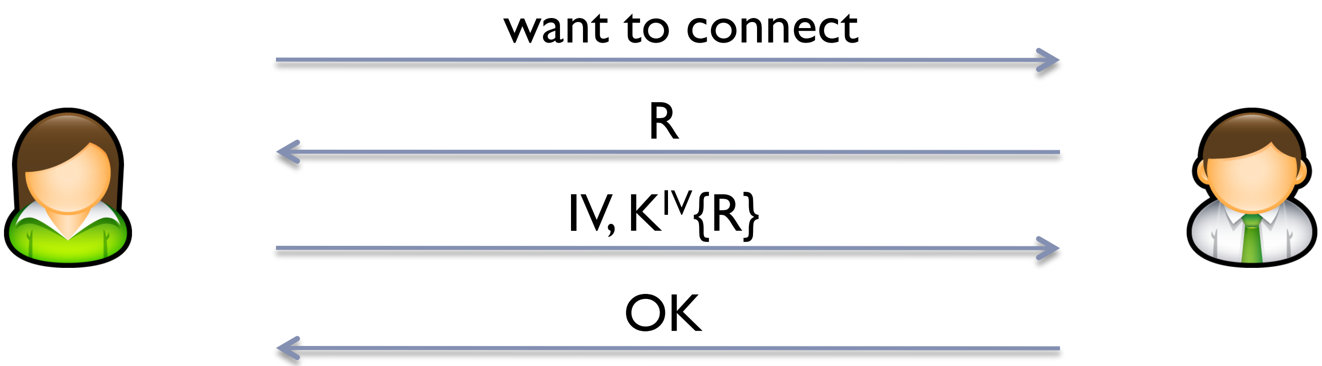
WEP decryption overview



- Receiver extracts IV
- Inputs IV and shared secret key into pseudo random generator, gets keystream
- XORs keystream with encrypted data to decrypt data + ICV
- Verifies integrity of data with ICV
  - Note that message integrity approach used here is different from the MAC (message authentication code) and signatures (using PKI).

WEP Authentication

- Two different auth modes
  - Open System and Shared Secret
- Open System
  - No real authentication, anybody can associate with AP
  - After AP association, device needs to have the correct key, otherwise packets will be rejected (will fail integrity check)
- Shared Secret
  - Device needs to provide credentials before AP association



want to connect

R

IV, $K^{IV}\{R\}$

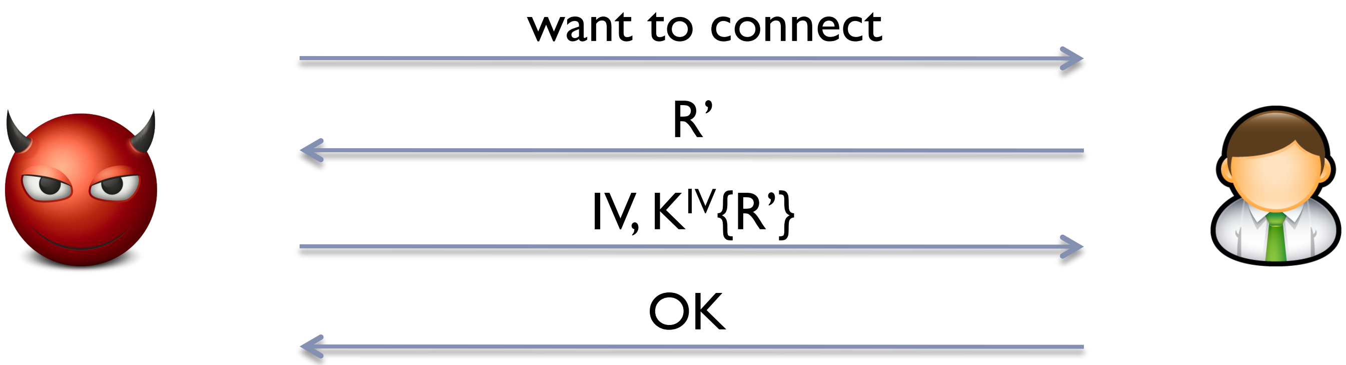
OK

Is there any problem here? (assume attacker records conversation)

see Borisov et al. "Intercepting Mobile Communications: The Insecurity of 802.11"

WEP Authentication

- Problems with Shared Secret authentication
  - Eve eavesdropped R and IV, $K^{IV}\{R\}$
  - Thus, Eve knows the key-stream related to IV
    - Reuse known IV, K^{IV} to authenticate and associate with AP



want to connect

R’

IV, $K^{IV}\{R’\}$

OK

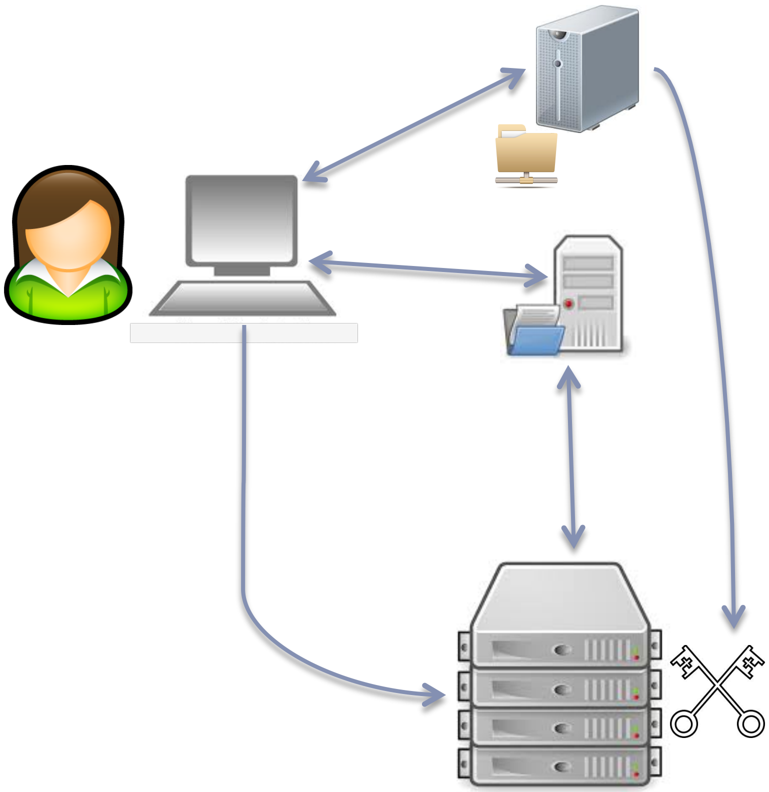
see Borisov et al. “Intercepting Mobile Communications: The Insecurity of 802.11”

Breaking 802.11 WEP encryption

security hole:

- 24-bit IV, one IV per frame, -> IV's eventually reused
- IV transmitted in plaintext -> IV reuse detected
- attack:
  - Trudy causes Alice to encrypt known plaintext $d_1\ d_2\ d_3\ d_4$ …
  - Trudy sees: $c_i = d_i$ XOR k_i^{IV}
  - Trudy knows $c_i\ d_i$, so can compute k_i^{IV}
  - Trudy knows encrypting key sequence $k_1^{IV}\ k_2^{IV}\ k_3^{IV}$ …
  - Next time IV is used, Trudy can decrypt!
- RC4 does not work well with "weak" IVs
  - A. Bittau, M. Handley and J. Lackey. *The Final Nail in WEP's Coffin.* Proceedings of the IEEE Symposium on Security and Privacy, 2006

Kerberos



Kerberos

- **Kerberos** is an authentication protocol and a software suite implementing this protocol.
- Kerberos uses symmetric cryptography to authenticate clients to services and vice versa.
- For example, Windows servers use Kerberos as the primary authentication mechanism, working in conjunction with Active Directory to maintain centralized user information.
- Other possible uses of Kerberos include allowing users to log into other machines in a local-area network, authentication for web services, authenticating email client and servers, and authenticating the use of devices such as printers.
- Services using Kerberos authentication are commonly referred to as “Kerberized”.

Kerberos Components

- Key Distribution Center (KDC)
  - Runs on a *physically secure* node on the network
  - Shares a **master key** with each **principal** (i.e., each user and each resource/service that will be using Kerberos)

- KDC has two components
  - An **authentication server (AS),** which performs user authentication
  - A **ticket-granting server (TGS),** which grants tickets to users

Kerberos Tickets

- Kerberos uses the concept of a **ticket** as a token that proves the identity of a user.
- Tickets are digital documents that store session keys. They are typically issued during a login session and then can be used instead of passwords for any Kerberized services. During the course of authentication, a client receives two tickets:
  - A **ticket-granting ticket (TGT),** which acts as a global identifier for a user and a session key
  - A **service ticket,** which authenticates a user to a particular service
- These tickets include time stamps that indicate an expiration time after which they become invalid. This expiration time can be set by Kerberos administrators depending on the service.

Kerberos Features

- The authentication server keeps a database storing the master keys of the users and services.
- The master key of a user is typically generated by performing a one-way hash of the user-provided password.
- Kerberos is designed to be modular, so that it can be used with a number of encryption protocols, with AES being the default cryptosystem.
- Kerberos aims to centralize authentication for an entire network—rather than storing sensitive authentication information at each user's machine, this data is only maintained in one presumably secure location.

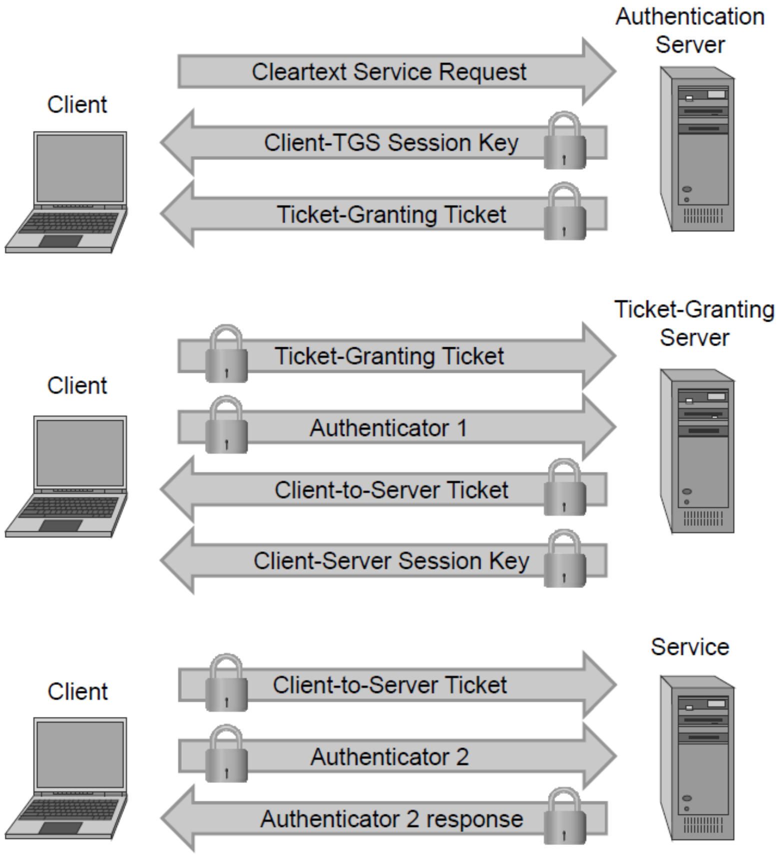
Kerberos v4 at a glance

- Alice logs into her workstation
  - Enters user name and password
  - A master key is derived from the password
  - The workstations asks KDC for a session key S_A for Alice, and then forgets the password she entered
  - S_A will be used to ask KDC for ***tickets*** to access services
  - S_A will expire after a given time (e.g., a few hours)
  - KDC generates SA and sends $K_A\{S_A\}$ and $K_{KDC}\{$“Alice”, S_A, timeout$\}$ to Alice
  - $K_{KDC}\{$“Alice”, S_A, timeout$\}$ is called Ticket Granting Ticket (TGT)

- Alice (a user) wants to talk to (or use) Bob (a service)
  - Alice informs the KDC that she needs Bob, and sends her TGT
  - KDC decrypts TGT to get S_A
  - KDC generates a session key K_{AB}, encrypts K_{AB} with Alice’s session key S_A, encrypts K_{AB} with Bob’s key K_B, and sends them to Alice
  - $K_B\{K_{AB}\}$ is called a ***ticket to Bob***
  - K_{AB} is known only to Alice and Bob (and the KDC), and can be used by Alice and Bob to authenticate each other, encrypt and integrity-protect their communication

Kerberos Authentication

- The client and authentication server authenticate themselves to each other.
- The client and ticket-granting server authenticate themselves to each other.
- The client and requested service authenticate themselves to each other, at which point the service will be provided to the client.



Obtaining a TGT

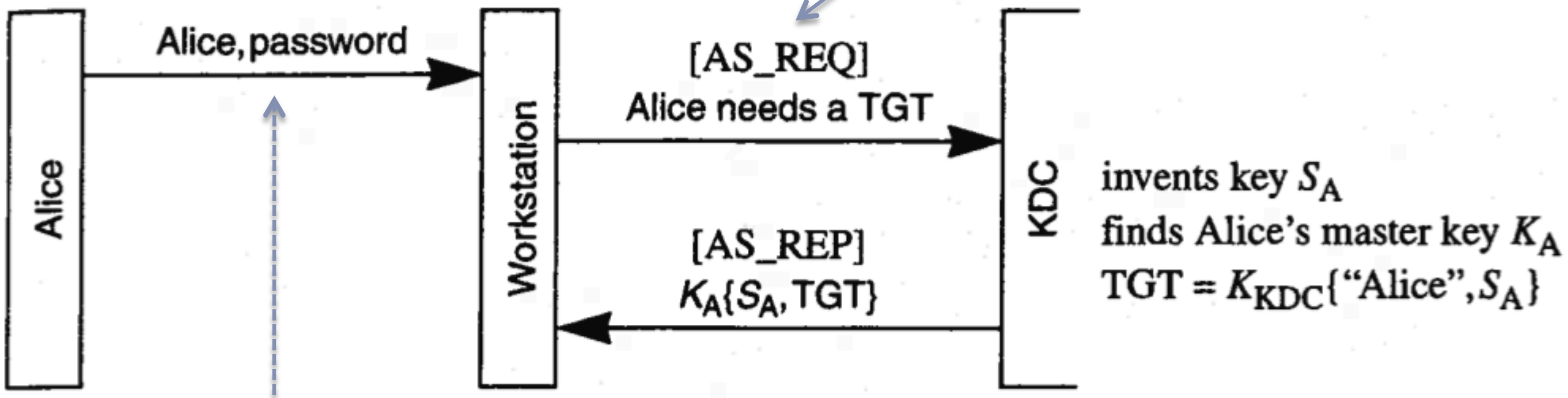


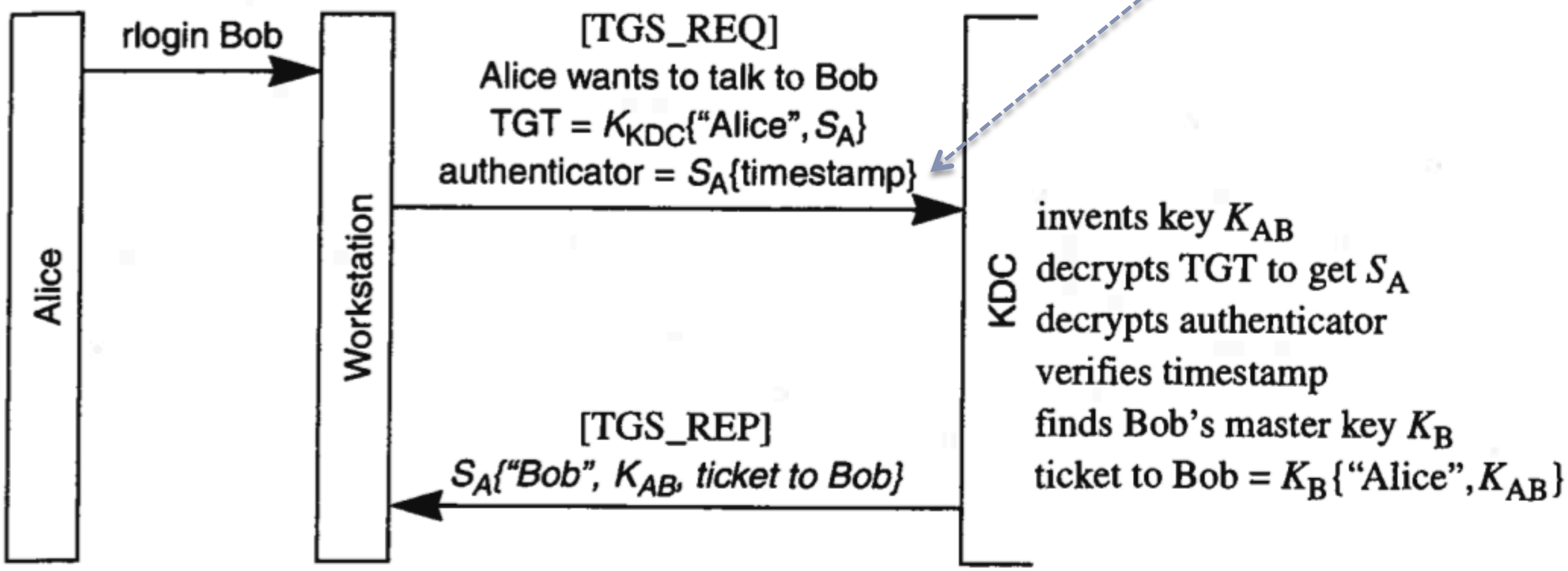
Figure 13-1. Obtaining a TGT

Plaintext message

Alice's master secret derived from entered pswd

Ticket to Bob

authenticator proves Alice knows S_A



Alice

Workstation

KDC

rlogin Bob

[TGS_REQ]
Alice wants to talk to Bob
TGT = K_{KDC}\{"Alice", S_A\}
authenticator = S_A\{timestamp\}

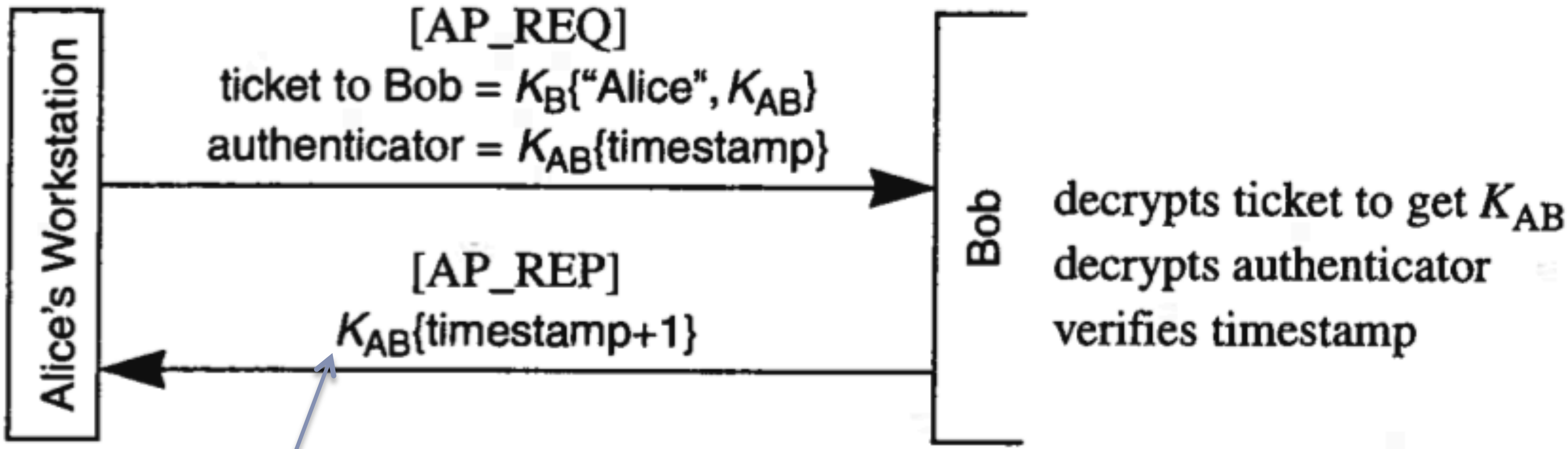
invents key K_{AB}
decrypts TGT to get S_A
decrypts authenticator
verifies timestamp
finds Bob's master key K_B
ticket to Bob = K_B\{"Alice",K_{AB}\}

[TGS_REP]
S_A\{"Bob", K_{AB}, ticket to Bob\}

Figure 13-2. Getting a ticket to Bob for Alice

Need reasonably synchronized clocks!

Logging into Bob



Alice's Workstation

[AP_REQ]

ticket to Bob = K_B{"Alice", K_{AB}}

authenticator = K_{AB}{timestamp}

Bob

decrypts ticket to get K_{AB}

decrypts authenticator

verifies timestamp

[AP_REP]

K_{AB}{timestamp+1}

Figure 13-3. Logging into Bob from Alice's workstation

Provides for mutual auth

Kerberos Advantages

- The Kerberos protocol is designed to be secure even when performed over an insecure network.
- Since each transmission is encrypted using an appropriate secret key, an attacker cannot forge a valid ticket to gain unauthorized access to a service without compromising an encryption key or breaking the underlying encryption algorithm, which is assumed to be secure.
- Kerberos is also designed to protect against replay attacks, where an attacker eavesdrops legitimate Kerberos communications and retransmits messages from an authenticated party to perform unauthorized actions.
  - The inclusion of time stamps in Kerberos messages restricts the window in which an attacker can retransmit messages.
  - Tickets may contain the IP addresses associated with the authenticated party to prevent replaying messages from a different IP address.
  - Kerberized services make use of a "replay cache," which stores previous authentication tokens and detects their reuse.
- Kerberos makes use of symmetric encryption instead of public-key encryption, which makes Kerberos computationally efficient
- The availability of an open-source implementation has facilitated the adoption of Kerberos.

Kerberos Disadvantages

- Kerberos has a single point of failure: if the Key Distribution Center becomes unavailable, the authentication scheme for an entire network may cease to function.
  - Larger networks sometimes prevent such a scenario by having multiple KDCs, or having backup KDCs available in case of emergency.
- If an attacker compromises the KDC, the authentication information of every client and server on the network would be revealed.
- Kerberos requires that all participating parties have synchronized clocks, since time stamps are used.