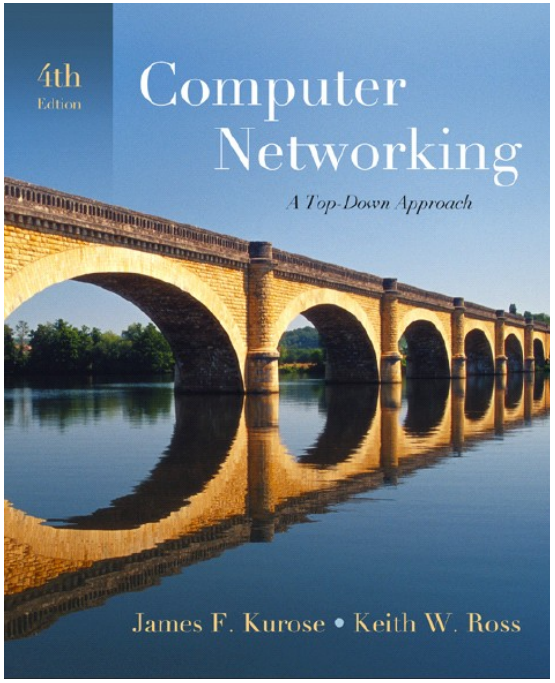
Chapter 8
Network Security



Adapted from
Computer Networking: A Top Down Approach ,
4th edition.
Jim Kurose, Keith Ross
Addison-Wesley, July 2007.

Network security

Foundations:

- what is security?
- cryptography
- authentication
- message integrity
- key distribution and certification

Security in practice:

- application layer: secure e-mail
- transport layer: Internet commerce, SSL, SET
- network layer: IP security

What is network security?

Secrecy (confidentiality): only sender, intended receiver should understand message contents

- sender encrypts messages
- receiver decrypts messages

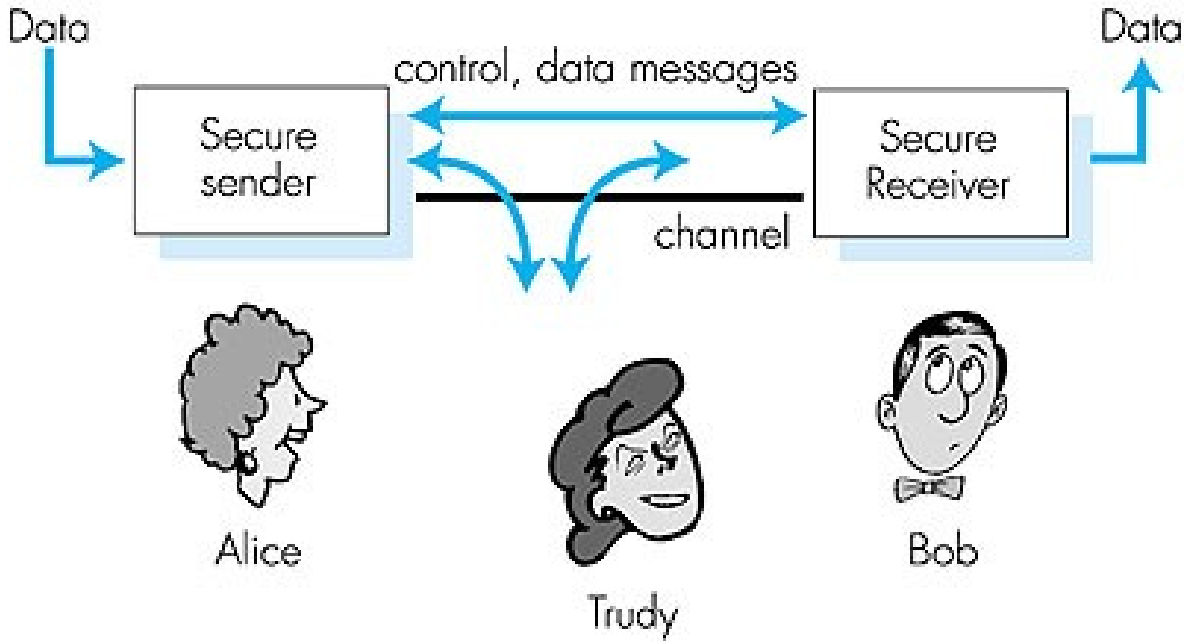
Authentication: sender, receiver want to confirm identity of each other

Message Integrity: sender, receiver want to ensure message not altered (in transit) without detection

Confidentiality:

Or “The art of encryption”

Friends and enemies: Alice, Bob, Trudy



- well-known in network security world
- Bob, Alice want to communicate 'securely'
- Trudy, the intruder may intercept, delete, add messages

There are bad guys (and girls) out there!

Q: What can a “bad guy” do?

A: a lot!

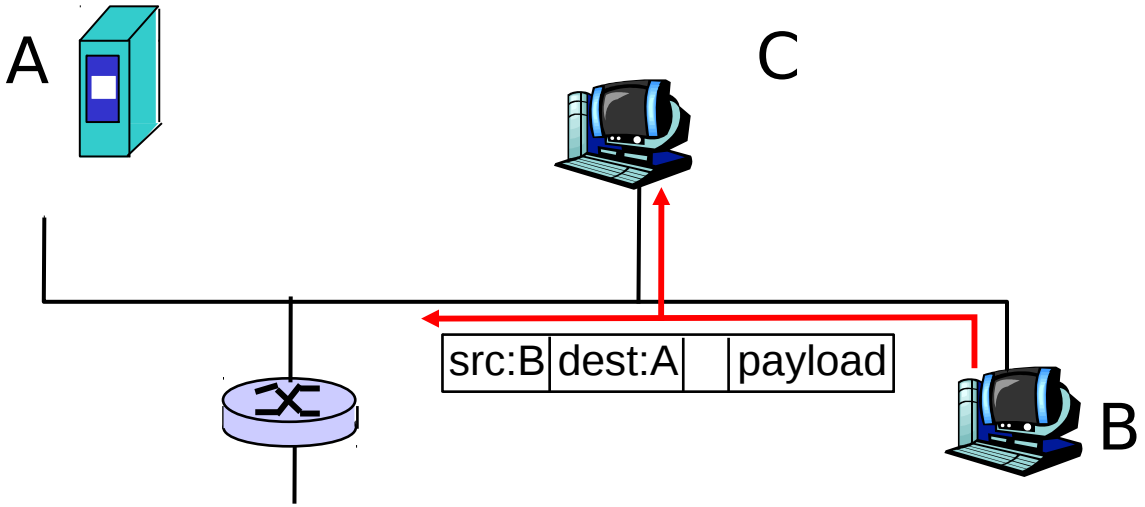
- *eavesdrop:* intercept messages
- actively *insert* messages into connection
- *impersonation:* can fake (spoof) source address in packet (or any field in packet)
- *hijacking:* “take over” ongoing connection by removing sender or receiver, inserting himself in place
- *denial of service*: prevent service from being used by others (e.g., by overloading resources)

more on this later ……

Internet security threats

Packet sniffing:

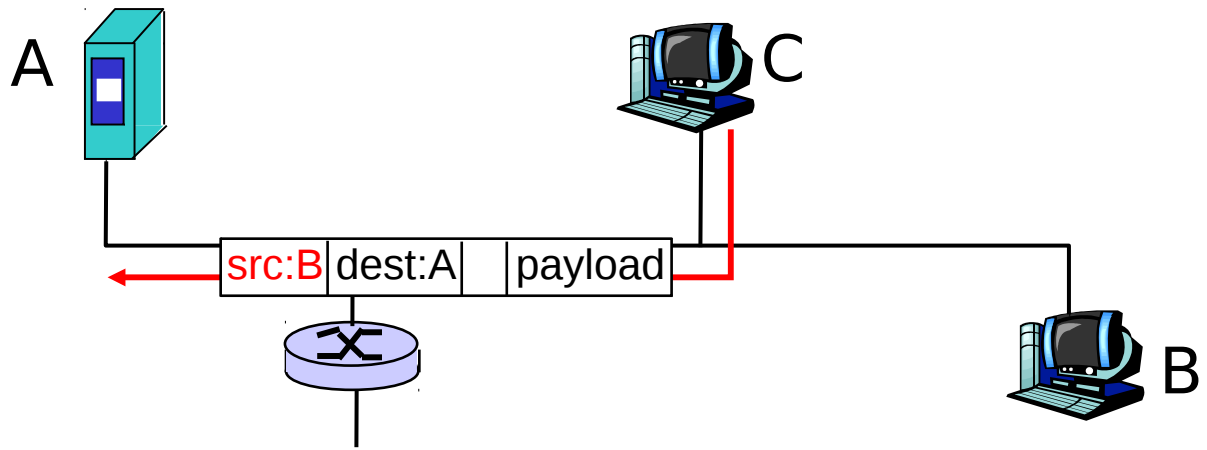
- broadcast media (remember CSMA/CD protocol)
- promiscuous NIC reads all packets passing by
- can read all unencrypted data (e.g. passwords)
- e.g.: C sniffs B's packets



Internet security threats

IP Spoofing:

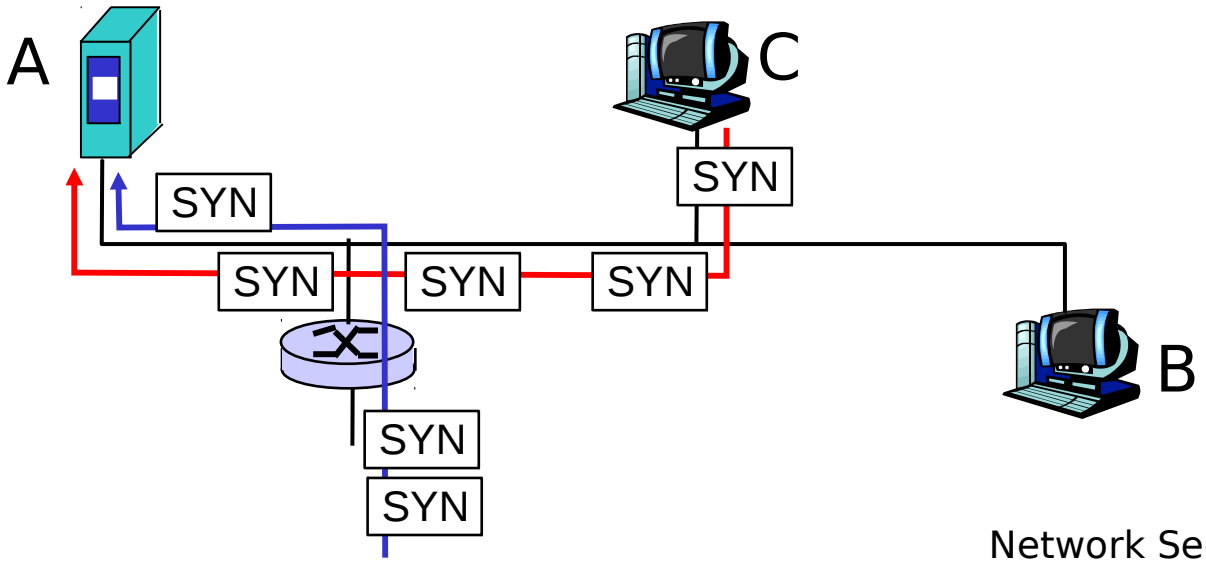
- can generate raw IP packets directly from application, putting any value into IP source address field
- receiver can't tell if source is spoofed
- e.g.: C pretends to be B



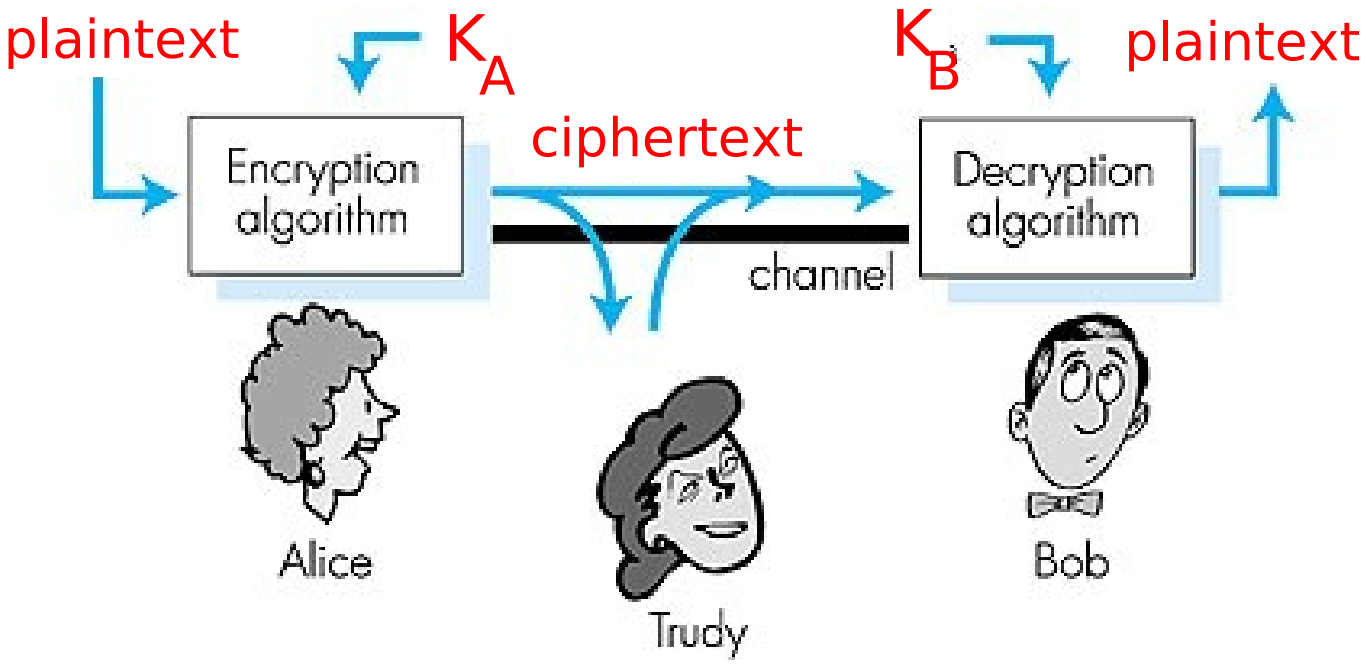
Internet security threats

Denial of service (DOS):

- flood of maliciously generated packets swamp receiver
- Distributed DOS (DDOS): multiple coordinated sources swamp receiver
- e.g., C and remote host SYN-attack A



The language of cryptography



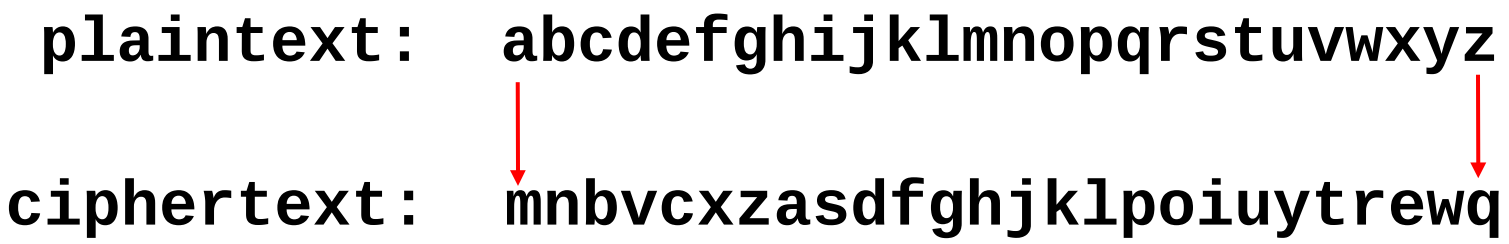
symmetric key crypto: sender, receiver keys identical

public-key crypto: encrypt key *public*, decrypt key *secret*

Symmetric key cryptography

substitution cipher: substituting one thing for another

- monoalphabetic cipher: substitute one letter for another



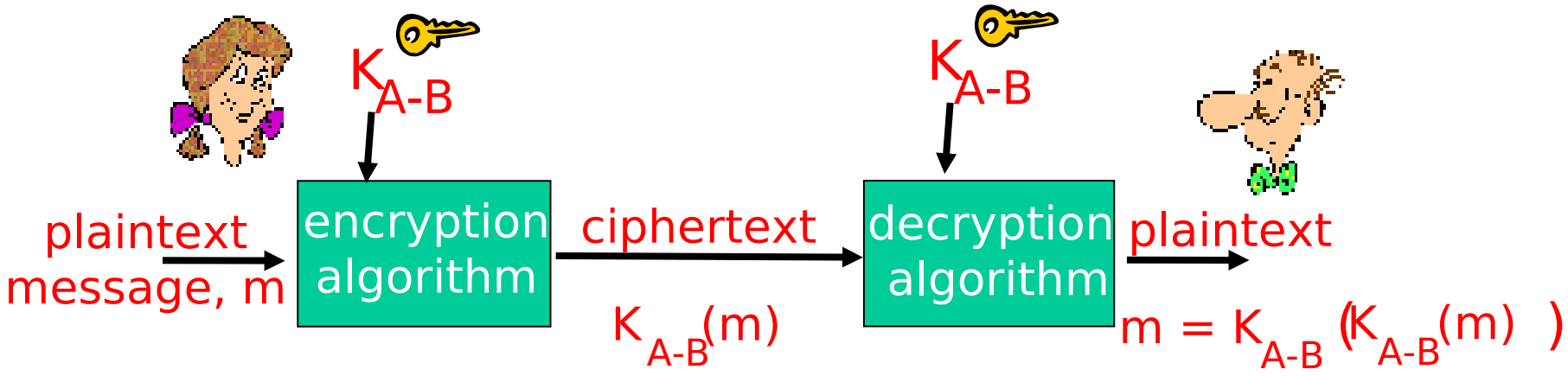
E.g.: **Plaintext: bob. i love you. alice**

ciphertext: nkn. s gktc wky. mgsbc

Q: How hard to break this simple cipher?:

- brute force (how hard?)
- Other methods?

Symmetric key cryptography



symmetric key crypto: Bob and Alice share know same (symmetric) key: K_{A-B}

- e.g., key is knowing substitution pattern in mono alphabetic substitution cipher
- Q: how do Bob and Alice agree on key value?

Symmetric key crypto: DES

DES: Data Encryption Standard

- US encryption standard [NIST 1993]
- 56-bit symmetric key, 64 bit plain text input
- How secure is DES?
  - DES Challenge: 56-bit-key-encrypted phrase ('Strong cryptography makes the world a safer place') decrypted (brute force) in 4 months
  - no known backdoor decryption approach
- making DES more secure
  - use three keys sequentially (3-DES) on each datum
  - use cipher-block chaining

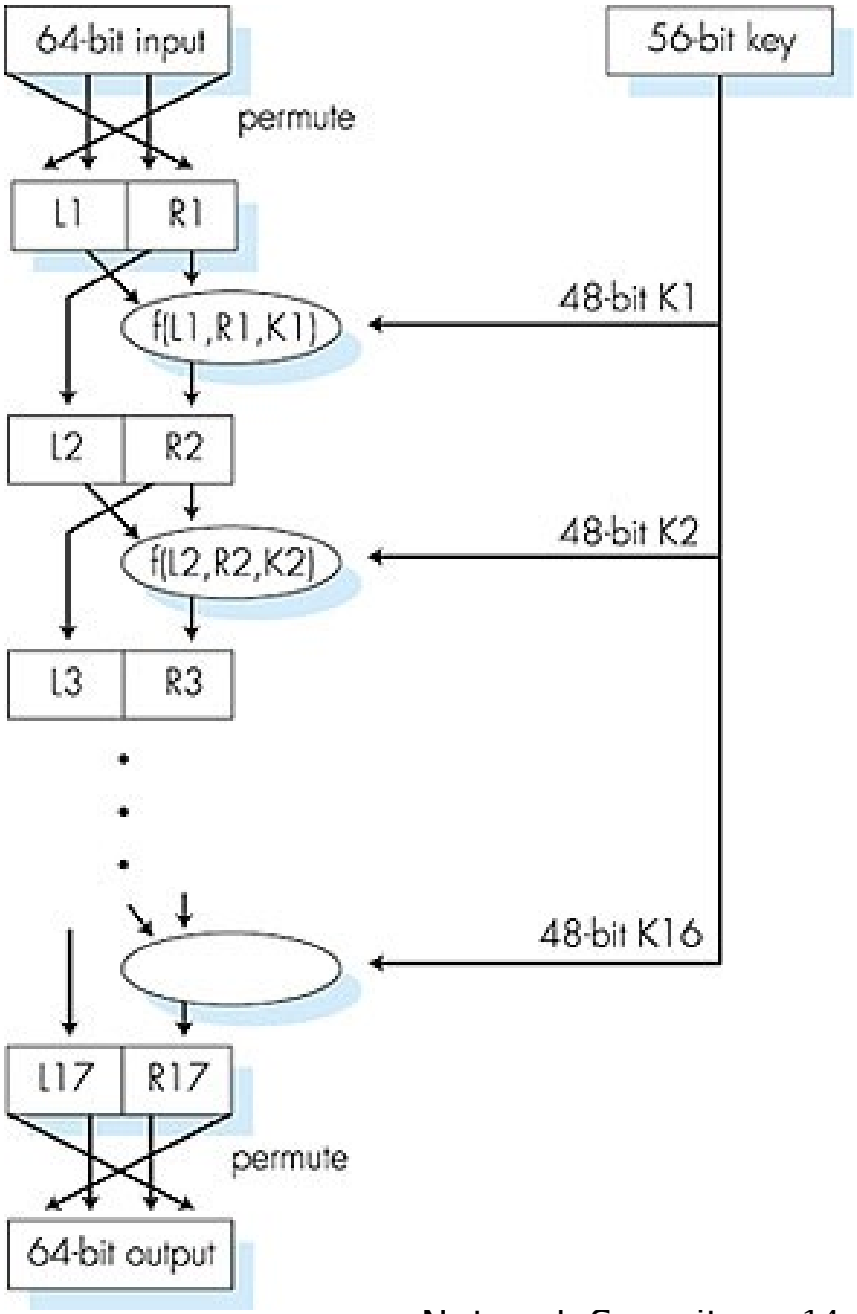
Symmetric key crypto: DES

DES operation

initial permutation

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

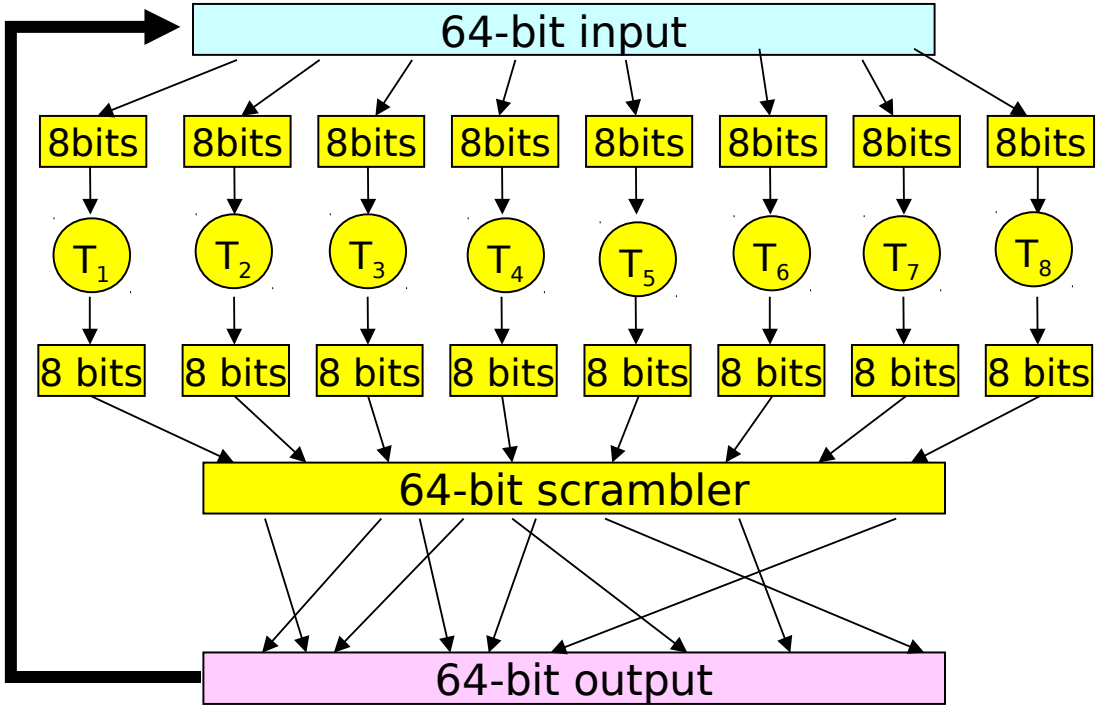
final permutation



AES: Advanced Encryption Standard

- new (Nov. 2001) symmetric-key NIST standard, replacing DES
- processes data in 128 bit blocks
- 128, 192, or 256 bit keys
- brute force decryption (try each key) taking 1 sec on DES, takes 149 trillion years for AES

Block Cipher



- one pass through: one input bit affects eight output bits
- multiple passes: each input bit afects all output bits
- block ciphers: DES, 3DES, AES

Public Key Cryptography

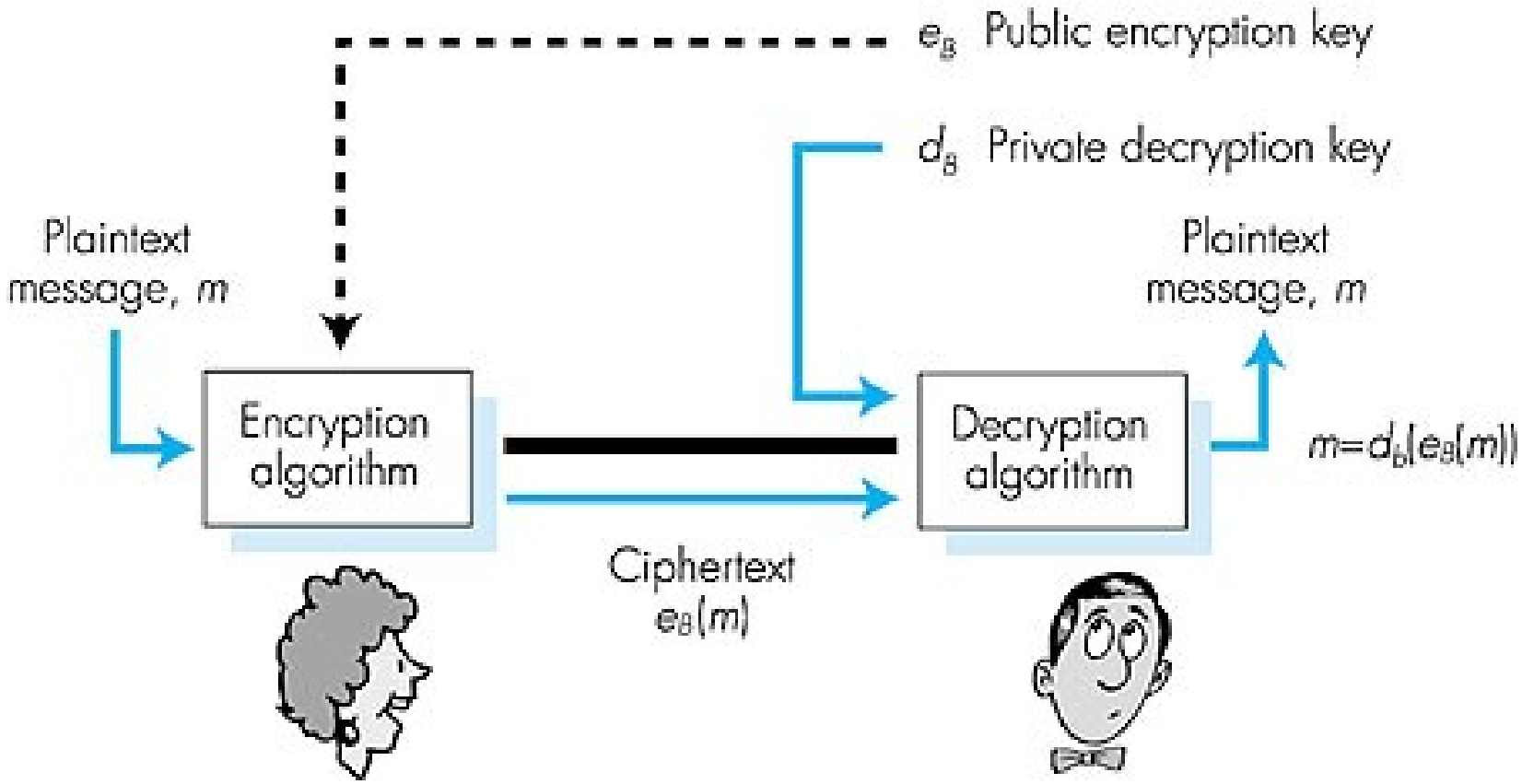
symmetric key crypto

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

public key cryptography

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

Public key cryptography



Public key encryption algorithms

Two inter-related requirements:

1. need $d_B(\ )$ and $e_B(\ )$ such that

$$d_B(e_B(m)) = m$$

2. need public and private keys for $d_B(\bullet)$ and $e_B(\bullet)$

RSA: Rivest, Shamir, Adelson algorithm

RSA: Choosing keys

1. Choose two large prime numbers *p*, *q*. (e.g., 1024 bits each)

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

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

4. Choose *d* such that *ed-1* is exactly divisible by *z*. (in other words: *ed* mod *z* = *1*). and z,d have no common factors as well

5. *Public* key is *(n,e)*. *Private* key is *(n,d)*.

RSA: keys in practice

1. Choose p and q with similar order
   - same bitlenght

2. p-q should not be too small,
   - otherwise p (or q) is ~ sqrt(n)

3. p and q are *pseudo-primes*
   - not necessarily *true* primes
   - Strong primes is a restriction (large factors)

4. e is commonly:
   - 3 (weak)
   - 65537 (this is 2^16+1)

RSA: Encryption, decryption

0. Given (n,e) and (n,d) as computed above

1. To encrypt bit pattern, m, compute

$c = m^e \bmod n$ (i.e., remainder when m^e is divided by n)

2. To decrypt received bit pattern, c, compute

$m = c^d \bmod n$ (i.e., remainder when c^d is divided by n)

Magic happens!

$$m = (m^e \bmod n)^d \bmod n$$

RSA: Why:

$$m = (m^e \bmod n)^d \bmod n$$

Number theory result: If p,q prime, $n = pq$, then

$$x^y \bmod n = x^{y \bmod (p-1)(q-1)} \bmod n$$

$$(m^e)^d \bmod n = m^{ed} \bmod n$$

$$= m^{ed \bmod (p-1)(q-1)} \bmod n$$

(using number theory result above)

$$= m^1 \bmod n$$

(since we chose ed to be divisible by $(p-1)(q-1)$ with remainder 1)

$$= m$$

RSA: prime numbers to play with

A list of primes up to 1000:

2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59,61,67,71,
73,79,83,89,97,101,103,107,109,113,127,131,137,139,
149,151,157,163,167,173,179,181,191,193,197,199,211,
223,227,229,233,239,241,251,257,263,269,271,277,281,
283,293,307,311,337,347,349,353,359,367,373,379,383,
389,397,401,409,419,421,431,433,439,443,449,457,461,
463,467,479,487,491,499,503,509,521,523,541,547,557,
563,569,571,577,587,593,599,601,607,613,617,619,631,
641,643,647,653,659,661,673,677,683,691,701,709,719,
727,733,739,743,751,757,761,769,773,787,797,809,811,
821,823,827,829,839,853,857,859,863,877,881,883,887,
907,911,919,929,937,941,947,953,967,971,977,983,991,
997

RSA example:

Choose *p=5, q=7*.
Then *n=35, z=(5-1)(7-1)=24*.

e=? (so *e, z* relatively prime).
Could not be 3, but could be 5,7,11,13 etc...
choose e=5 (a better choice would be e≠p...)

d=? For really small numbers, by trial and error:
try d=11,13,17...29 until

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

RSA simplistic example:

For $p=5$, $q=7$. Then $n=35$, $z=24$.

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

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

Suppose message m='l'=12

	letter	m	$c = m^e \bmod n$
encrypt:	l	12	17

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

Issues with this simplistic example:

e should be different than p,q

Once we choose e, how to find d?

Trial and error will not solve it for large keys

In practice, e=65537 (e=3 is weak)

Use Extended Euclidean Algorithm
to find a suitable d (more info soon...)

Bézout's identity:

There exist two integers x,y that satisfy:

$$ax + by = gcd(a,b)$$

where a,b are also integers, and
gcd(a,b) is their greatest common divisor

This is called the ***Bézout's identity.***

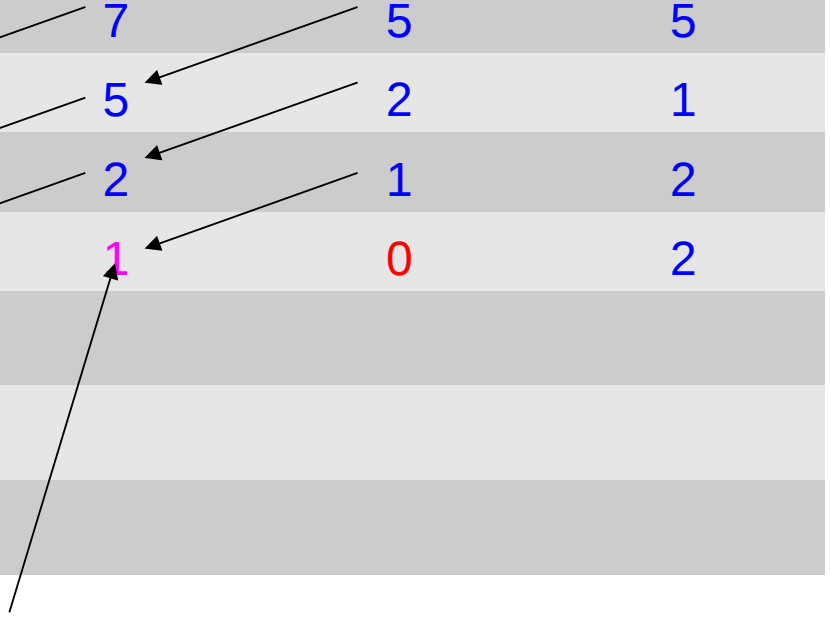
If we think a,b are z,e then we know that:

- gcd(z,e) = 1 (they are relative primes)
- zx + ed = gcd(z,e) = 1

Euclidean Algorithm gcd(a,b):

An example of the Euclid's algorithm:
say, p=11, q=5, and e=7. So, n=55 and z=40
Making sure that e and z are relative primes:

Dividend	Divisor	Remainder	Quotient
40	7	5	5
7	5	2	1
5	2	1	2
2	1	0	2



gdc(z,e) = 1

Extended Euclidean Algorithm:

The Extended Euclidean Algorithm can compute x,y given a,b...

For each step, compute the expression:

$r_i = r_{i-2} - q_i r_{i-1}$

Starting with $r_1 = a$ and $r_2 = b$

The last row gives us x and y

Extended Euclidean Algorithm:

An example with z=40 and e=7 again:

step	quotient	remainder		
1		40		40=40x1+7x0
2		7		7=40x0+7x1
3	5	5	=(40x1+7x0)-(40x0+7x1)x5	5=40x1+7x(-5)
4	1	2	=(40x0+7x1)-(40x1+7x(-5))x1	2=40x(-1)+7x6
5	2	1	=(40x1+7x(-5))-(40x(-1)+7x6)x2	1=40x3+7x(**-17**)
6	1	0		

Problem: we found that d=-17

But: 40x(3-7)+7x(-17+40)=40x(-4)+7x23 = 1

So d=23 is also suitable

Extended Euclidean Algorithm:

Another way using a table with z=40 and e=7:

step	x	y	w	k
1	1	0	40	
2	0	1	7	
3				
4				
5				
6				

$k_i=(int)w_{i-1}/w_i$

$x_i = x_{i-2} - k_{i-1} x_{i-1}$, $y_i = y_{i-2} - k_{i-1} y_{i-1}$

and $w_i = w_{i-2} - k_{i-1} w_{i-1}$

Extended Euclidean Algorithm:

Another way using a table with z=40 and e=7:

step	x	y	w	k
1	1	0	40	
2	0	1	7	5
3	1	-5	5	1
4	-1	6	2	2
5	3	**-17**	1	
6				

If d is negative, d=d+z will be suitable

d=-17+40=23

Extended Euclidean Algorithm:

Another way with p=7, q=11, z=60 and e=13:

step	x	y	w	k
1	1	0	60	
2	0	1	13	
3				
4				
5				
6				

d=

Back to the previous RSA example:

For $p=5$, $q=7$. Then $n=35$, $z=24$.

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

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

	letter	m	m^e	$c = m^e \bmod n$
encrypt:	l	12	248832	17

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

c^d = 48196857210675091509141182522307169 7 - ***too big !!*** (int type)

How to solve overflow problems:

$C^d = 17^{29}$: *break it into powers that are multiple of 2.*

$$17^{29} = 17^{16} \cdot 17^8 \cdot 17^4 \cdot 17$$

$17^4 \bmod 35 = 11$ *hence* $17^4 \bmod 35 = 11 \bmod 35$,

So 11 *can substitute* 17^4, 17^8, and 17^{16} *in the expression*:

$$17^{29} \bmod 35 = 11^4 \cdot 11^2 \cdot 11 \cdot 17 \bmod 35$$

Also $11^2 \bmod 35 = 16$

$$17^{29} \bmod 35 = 16^2 \cdot 16 \cdot 11 \cdot 17 \bmod 35$$

This is equivalent to do calculate x, y and z as follows:

$$x = n^4 \bmod 35$$
$$y = x^2 \bmod 35$$
$$z = y^2 \bmod 35$$

$$m = z \cdot y \cdot x \cdot c \bmod 35 \text{ (smaller !!)}$$

Alternatively: use a similar method to the one used in CRC.

Remember that (c.c.c) mod n = (c.c) (c mod n) ….

Even better solution:

Repeated Squaring: calculate $y = x^e \bmod n$

```
int repeatsquare( int x, int e, int n) {

   y=1;//initialize y to 1, very important
   while (e >  0) {
      if (( e % 2 ) == 0) {
         x = (x*x) % n;
         e = e/2;
      }
      else {
         y = (x*y) % n;
         e = e-1;
      }
   }
   return y; //the result is stored in y
}
```

RSA:how strong is it??

RSA Challenges:

- Historically: a prize was offered to anyone who could break an RSA key of a certain size (See www.rsa.com/rsalabs/), **challenge no longer active.**
- US$200,000.00 for a 2048 bits factorization problem
- Last challenge solved (no money for it...):
  - RSA-768 Factored in 2009 by T. Kleinjung et.al.
  - Using a powerful parallel machine and very clever algorithms (4400 1GHZ CPU years, 3 calendar years)
- Currently RSA-2048 is commonly used in practice
- RSA key's size matters, see next...

RSA:how strong is it??

Common sense calculation:

- Brute force factorization
  - Try all the prime number P that are smaller than the Key
  - When Key mod P = 0 then found the factors
- How long can it take depends on the RSA key size
- Suppose we have a key of 200 bits and the factors are approximately of the same order of digits
- Each key will have ~10^{100} trial divisions to do
- A 1Gflops machine could do 10^9 trials per second
- Say we have 10^9 machines in a cluster (massively parallel mach.)
- As we have only ~10^8 seconds/year, it would take
  - 10^{74} years !!!
  - Remember that the Earth is ~10^9 years old

RSA:how strong is it??

However:

- In the past few months new algorithms are threatening RSA
- Researcher think that within 5 years RSA may be obsolete
- In the mean time, TLS may have to use 16384 bits instead of the current standard of 2048 bits
- What then? One answer is ECC (Elliptical Curve Cryptography)
- Unfortunately, ECC is plagued by patent issues, making it difficult to implement in open source style
- BlackBerry X Sony
- Recent claims that 2 major mathematical discoveries increased the threats to RSA are sketchy, difficult to find detailed information on the web (for obvious reasons)
- ***Note: we still rely on increasing the key size for security***

RSA:how to implement it

A simplistic approach:

- *Translate each byte to a decimal number*
- *Use the RSA algorithm for each character*
  - ***Problem 1:*** only a few possible numbers will be used
    - e.g., 256 possible characters may be mapped into non-existing character if the key is large

  suppose that p=89 q=131
  - n = 11659 z = 11440
  - e = 3 and ed mod z = 1:
  - d = ? (does not matter for now)
  - 256 chars can now be encrypted to the interval:
    - 0 < (m^e)%n < 11658

RSA:how to implement it

Problem 1: char set goes beyond current representation

For example,

- **A** is char 65 (decimal)
- (65^e) % n = 6468
- But ASCII goes to 255 only, so cannot encode **6468**

RSA:how to implement it

Suppose that problem 1 is solved by a clever encoding.

Problem 2: very easy to break encryption due to language characteristics

- e.g., English uses double character such as 'll', 'rr' etc.
- A 'dictionary' attack could decrypt the message without ever finding the factorization

- Example: suppose encode 'l' (ascii 108 in decimal)
- In a longer message, find several 540 540 (doubles)
- By trial and error, 'r' (162), 'l' (108), 's' (115) etc...
- 162^e%n = 7652...
- 108^e%n = 540 found a big chunk of the message
- Trial and error with English words containing 'll'

RSA:how to implement it

To overcome problems 1 and 2 described before:

- Use *multiple chars* encoding
  - i.e., use a string as a “number”
  - Every ***char*** is encrypted differently depending on their combination with others
- When encoding messages, use padding:
  - Use fixed string length (enc_length)
  - m % enc_length is different than zero
  - add random chars to m to make
    - length(m+padding) % enc_length = 0

RSA: another important property

The following property will be *very* useful later:

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

use public key first, followed by private key

use private key first, followed by public key

Result is the same!

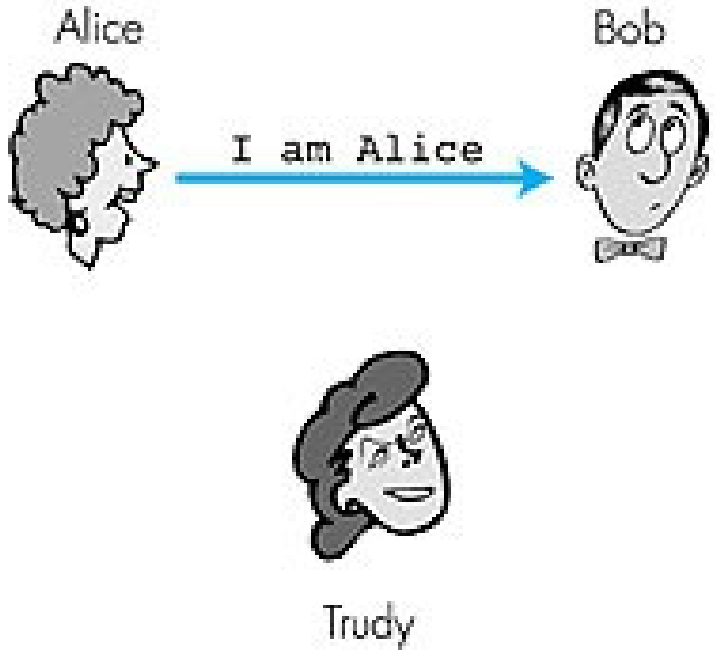
Authentication:

Ensure that the communication is with the "correct" machine

Authentication

Goal: Bob wants Alice to prove her identity to him

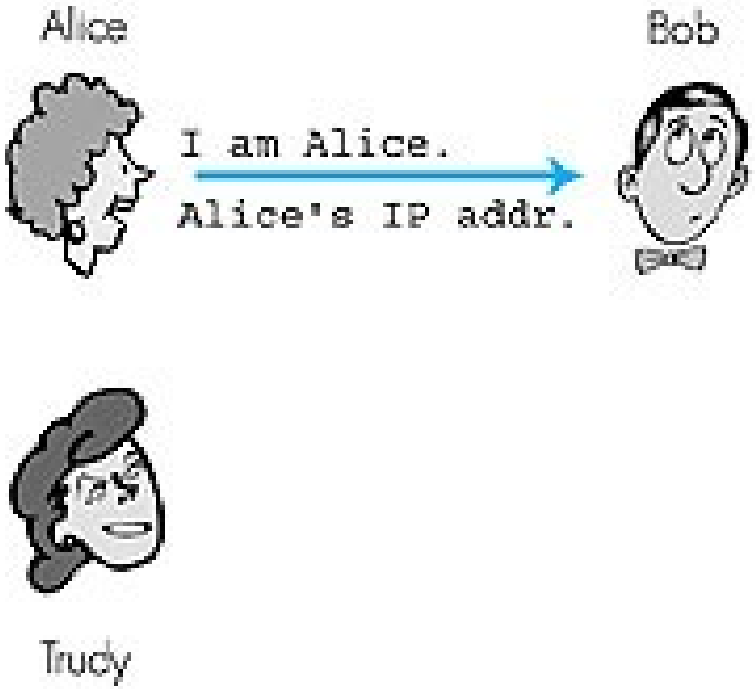
Protocol ap1.0: Alice says "I am Alice"



Failure scenario??

Authentication: another try

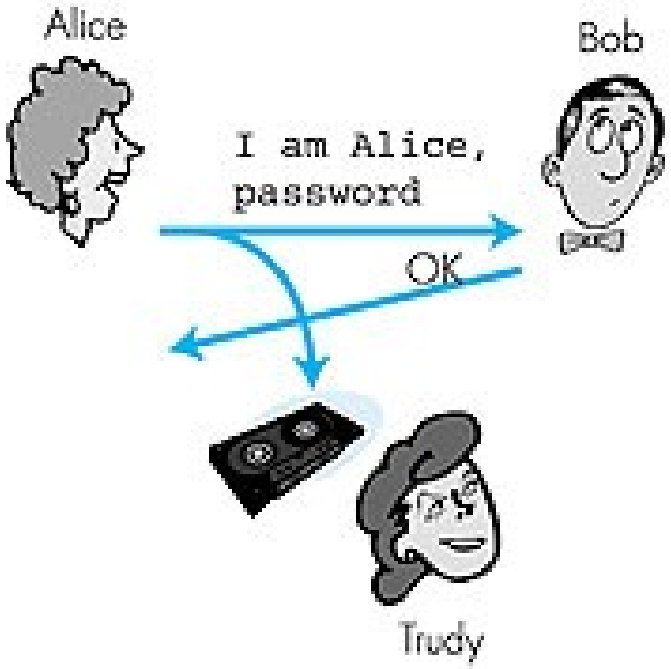
Protocol ap2.0: Alice says "I am Alice" and sends her IP address along to prove it.



Failure scenario??

Authentication: another try

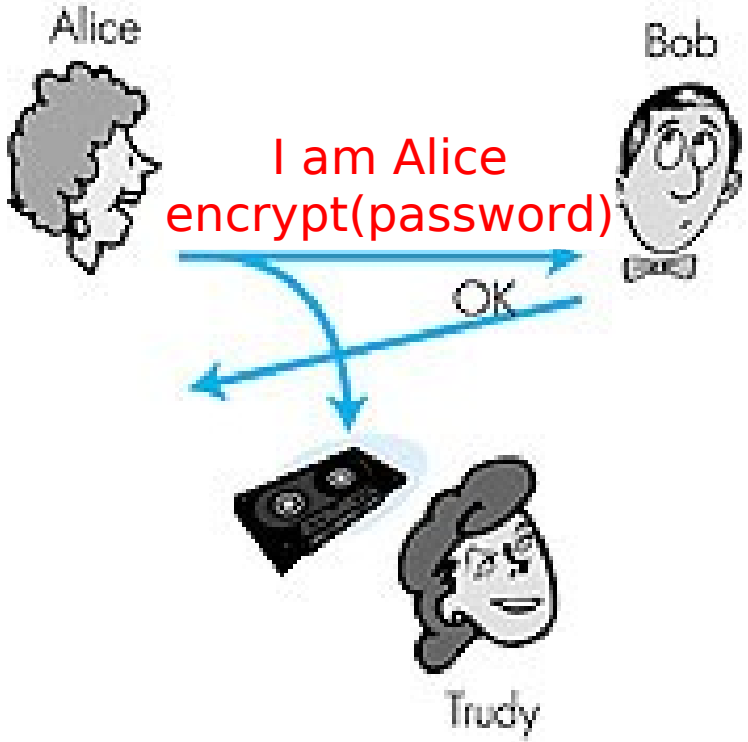
Protocol ap3.0: Alice says "I am Alice" and sends her secret password to prove it.



Failure scenario?

Authentication: yet another try

Protocol ap3.1: Alice says "I am Alice" and sends her *encrypted* secret password to prove it.



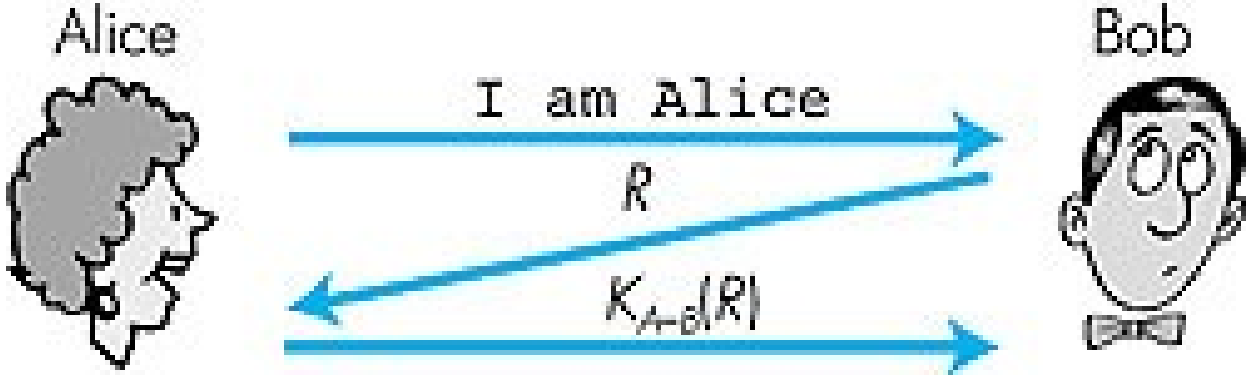
Failure scenario?

Authentication: yet another try

Goal: avoid playback attack

Nonce: number (R) used only once in a lifetime

ap4.0: to prove Alice live, Bob sends Alice nonce, R. Alice must return R, encrypted with shared secret key



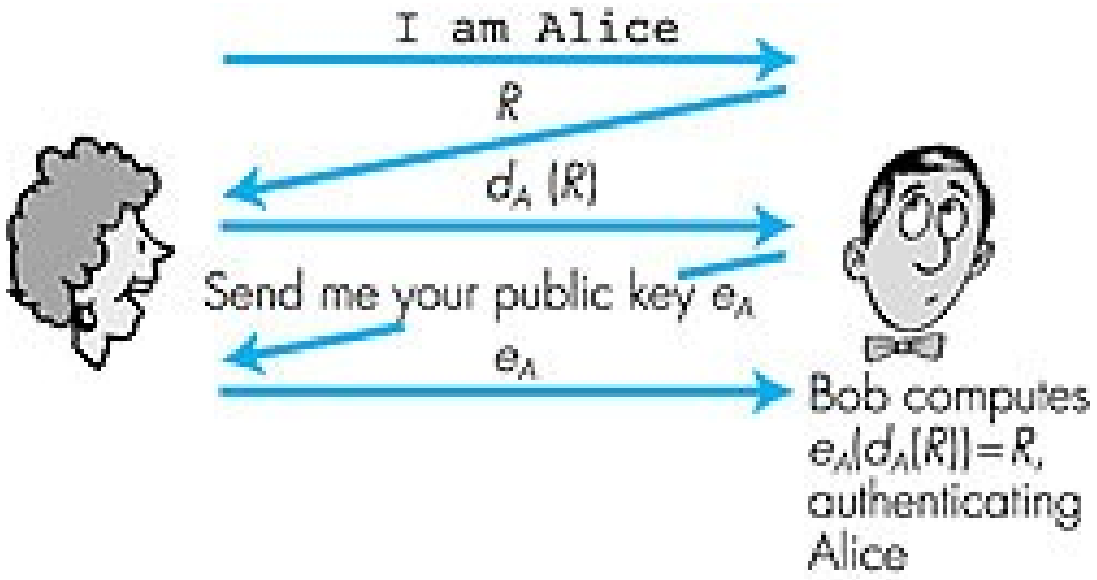
Failures, drawbacks?

Authentication: ap5.0

ap4.0 requires shared symmetric key

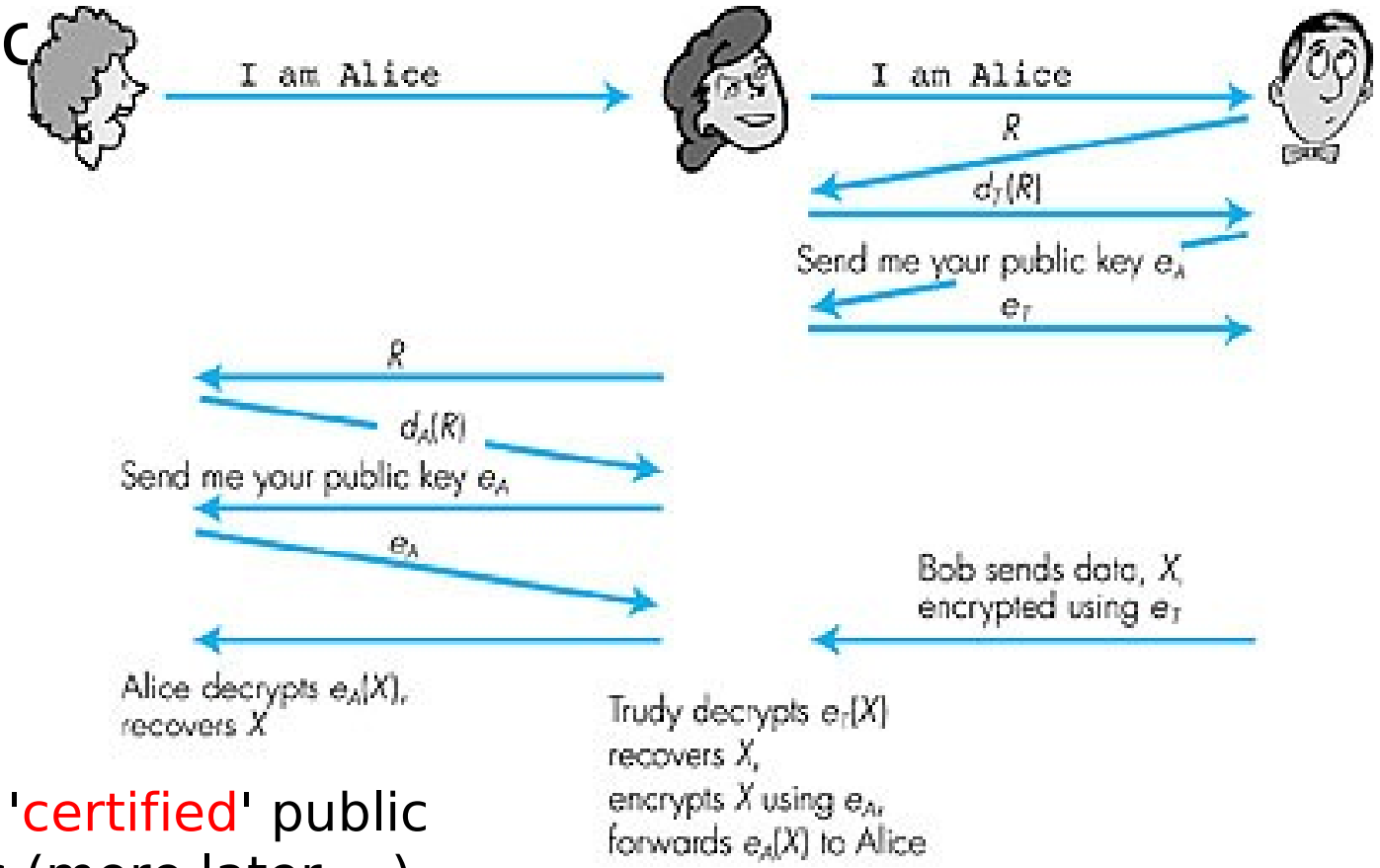
- problem: how do Bob, Alice agree on key
- can we authenticate using public key techniques?

ap5.0: use nonce, public key cryptography



ap5.0: security hole

Man (woman) in the middle attack: Trudy poses as Alice (to Bob) and as Bob (to Alic



Need 'certified' public keys (more later …)

Message Integrity:

Ensure that the message is not tampered with.

Message Integrity

Bob receives msg from Alice, wants to ensure:

- message originally came from Alice
- message not changed since sent by Alice

Cryptographic Hash:

- takes input m, produces fixed length value, H(m)
  - e.g., as in Internet checksum
- computationally infeasible to find two different messages, x, y such that H(x) = H(y)
  - equivalently: given m = H(x), (x unknown), can not determine x.
  - note: Internet checksum *fails* this requirement!

Internet checksum: poor crypto hash function

Internet checksum has some properties of hash function:

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

But given message with given hash value, it is easy to find another message with same hash value:

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

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

different messages but identical checksums!

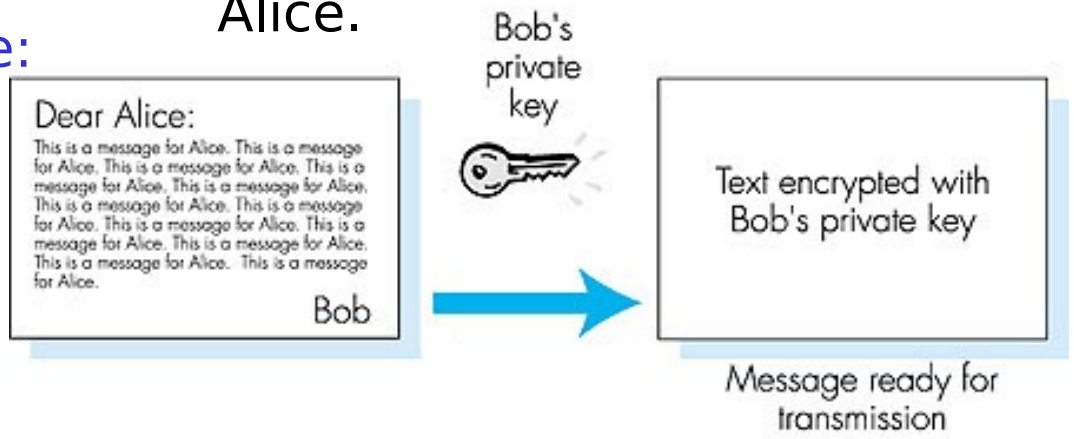
Digital Signatures

Cryptographic technique analogous to hand-written signatures.

- Sender (Bob) digitally signs document, establishing he is document owner/creator.
- Verifiable, non-forgeable: recipient (Alice) can verify that Bob, and no one else, signed document.

Simple digital signature for message m:

- Bob encrypts m with his public key d_B, creating signed message, $d_B(m)$.
- Bob sends m and $d_B(m)$ to Alice.



Digital Signatures (more)

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

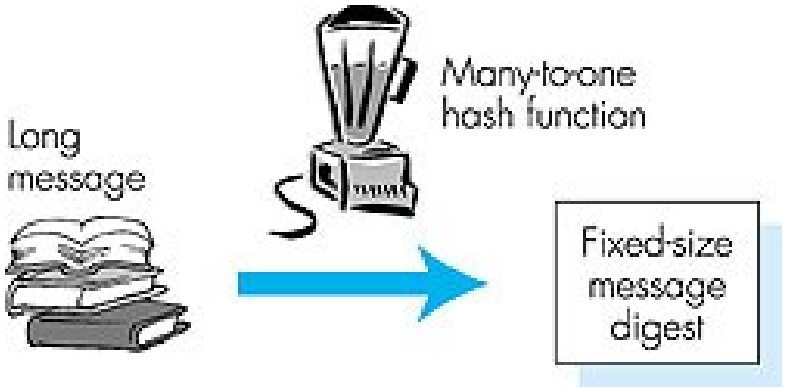
Alice thus verifies that:

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

Non-repudiation:

- Alice can take *m*, and signature $d_B(m)$ to court and prove that Bob signed *m*.

Message Digests



Computationally expensive to public-key-encrypt long messages

Goal: fixed-length,easy to compute digital signature, 'fingerprint'

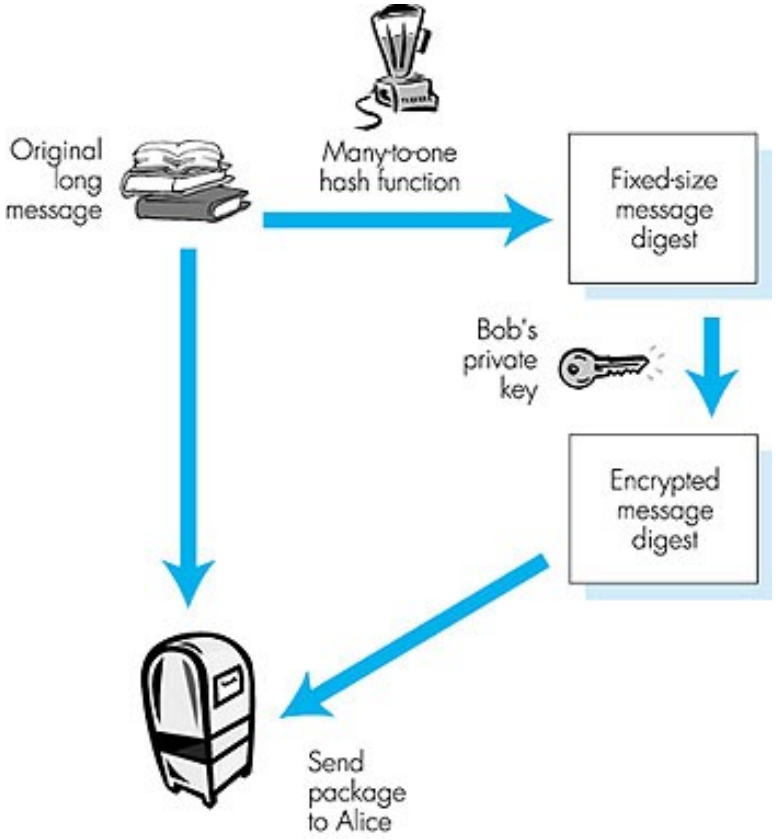
- apply hash function H to *m*, get fixed size message digest, *H(m).*

Hash function properties:

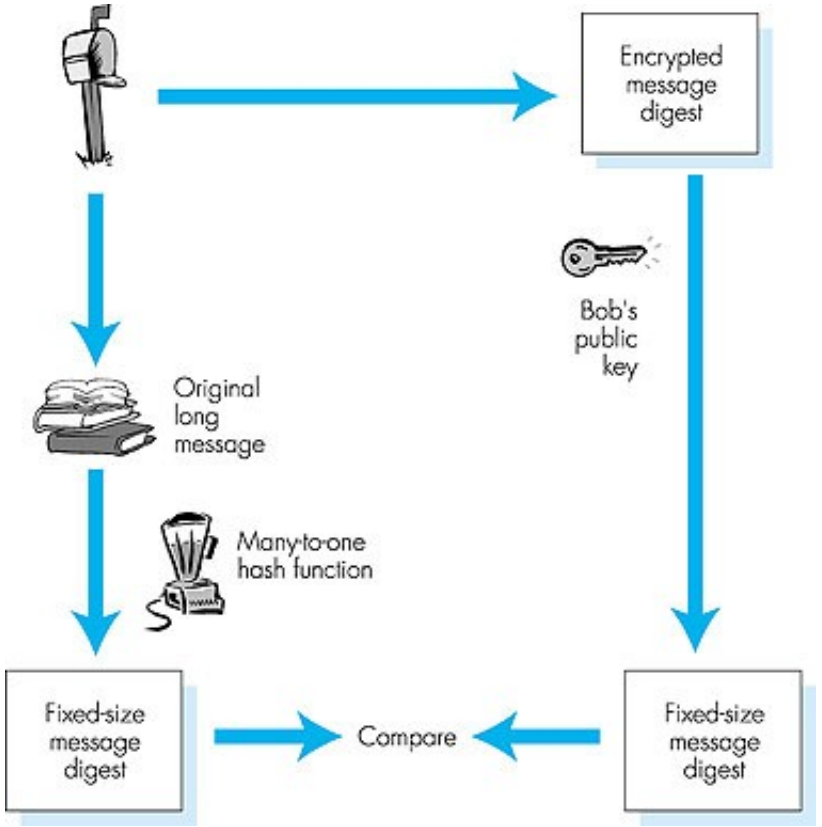
- Many-to-1
- Produces fixed-size msg digest (fingerprint)
- Given message digest x, computationally infeasible to find m such that x = H(m)
- computationally infeasible to find any two messages m and m' such that H(m) = H(m').

Digital signature = Signed message digest

Bob sends digitally signed message:



Alice verifies signature and integrity of digitally signed message:



Hash Function Algorithms

- Internet checksum would make a poor message digest.
  - Too easy to find two messages with same checksum.
  - Even using a 128-bit CRC it would be easy to find a second message to fit to the CRC
- MD5 hash function widely used.
  - Computes 128-bit message digest in 4-step process.
  - arbitrary 128-bit string x, appears difficult to construct msg m whose MD5 hash is equal to x.
- SHA-1 is also used.
  - US standard
  - 160-bit message digest

Hash Function Algorithms

- MD5
  - Try that using md5sum command in Linux ...
  - MD5 is a very reliable way to fingerprint a file
  - From rfc1321: ...*"The MD5 algorithm] takes as input a message of arbitrary length and produces as output a 128-bit "fingerprint" or "message digest" of the input. It is conjectured that it is computationally infeasible to produce two messages having the same message digest, or to produce any message having a given pre-specified target message digest"...*

Trusted Intermediaries

Problem:

- How do two entities establish shared secret key over network?

Solution:

- trusted key distribution centre (KDC) acting as intermediary between entities

Problem:

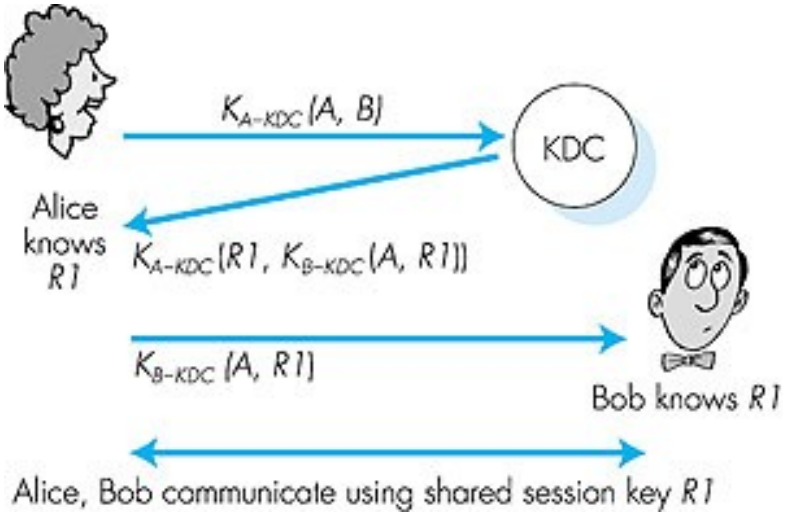
- When Alice obtains Bob's public key (from web site, e-mail, diskette), how does she know it is Bob's public key, not Trudy's?

Solution:

- trusted certification authority (CA)

Key Distribution Center (KDC)

- Alice,Bob need shared symmetric key.
- KDC: server shares different secret key with each registered user.
- Alice, Bob know own symmetric keys, $K_{A\text{-}KDC}$ $K_{B\text{-}KDC}$, for communicating with KDC.



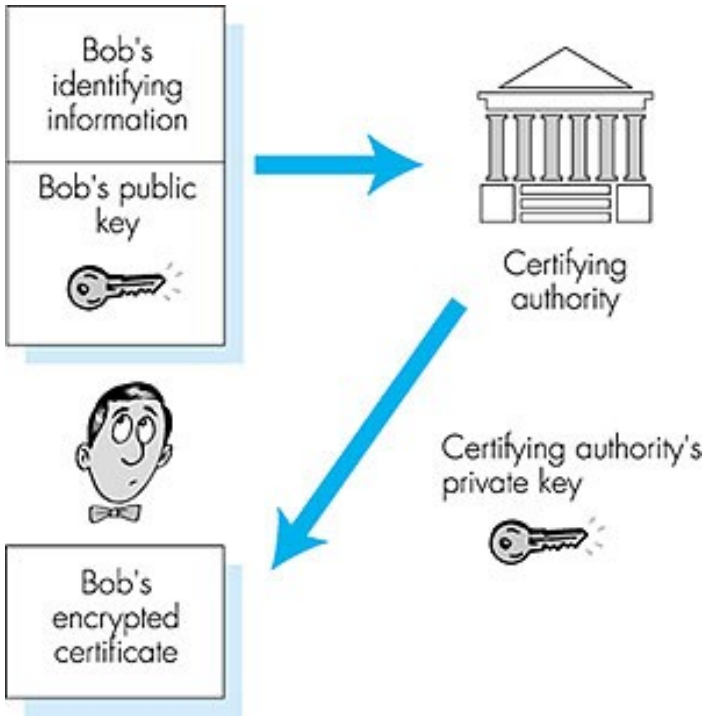
- Alice communicates with KDC, gets session key R1, and $K_{B\text{-}KDC}(A,R1)$
- Alice sends Bob $K_{B\text{-}KDC}(A,R1)$, Bob extracts R1
- Alice, Bob now share the symmetric key R1.

Key Distribution Center (KDC)

- Example: Kerberos
- See http://web.mit.edu/kerberos/
- ''Kerberos is a network authentication protocol''
- ''...firewalls assume that the bad guys are on the outside, which is often a very bad assumption...''
- Kerberos is freely available from MIT!
- Limitations of such approach:
  - Assumes users uses 'good' passwords
  - Assumes that only the network connections are vulnerable to attacks
- ==> importance of OS security cannot be underestimated

Certification Authorities

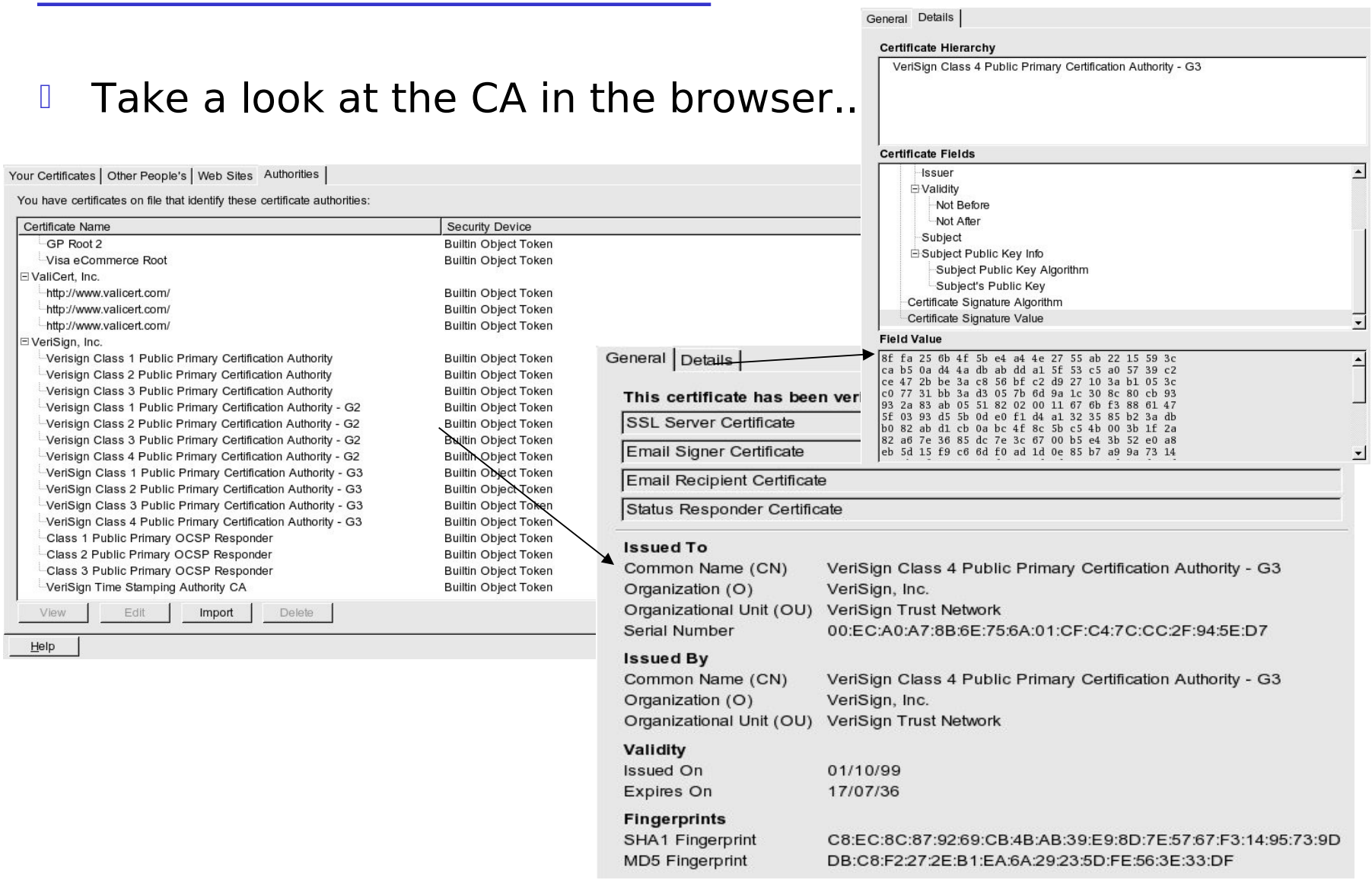
- Certification authority (CA) binds public key to particular entity.
- Entity (person, router, etc.) can register its public key with CA.
  - Entity provides proof of identity to CA.
  - CA creates certificate binding entity to public key.
  - Certificate digitally signed by CA.



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

Certification Authorities

- Take a look at the CA in the browser..

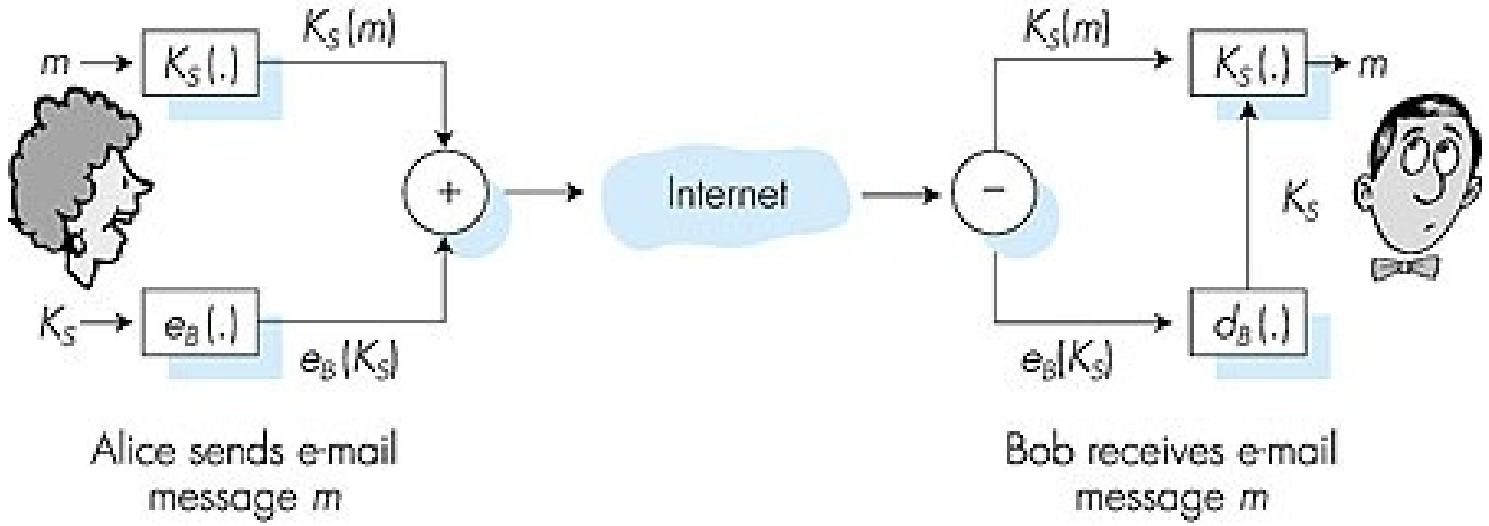


Secure email:

Putting it all together

Secure e-mail

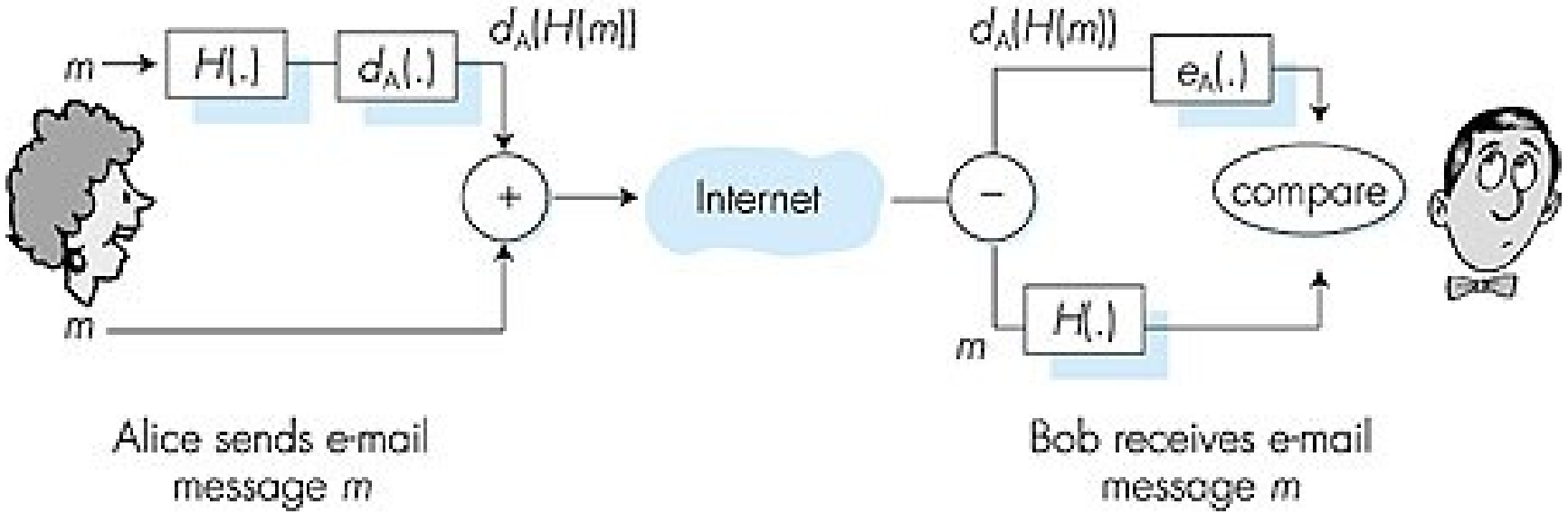
- Alice wants to send secret e-mail message, m, to Bob.



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

Secure e-mail (continued)

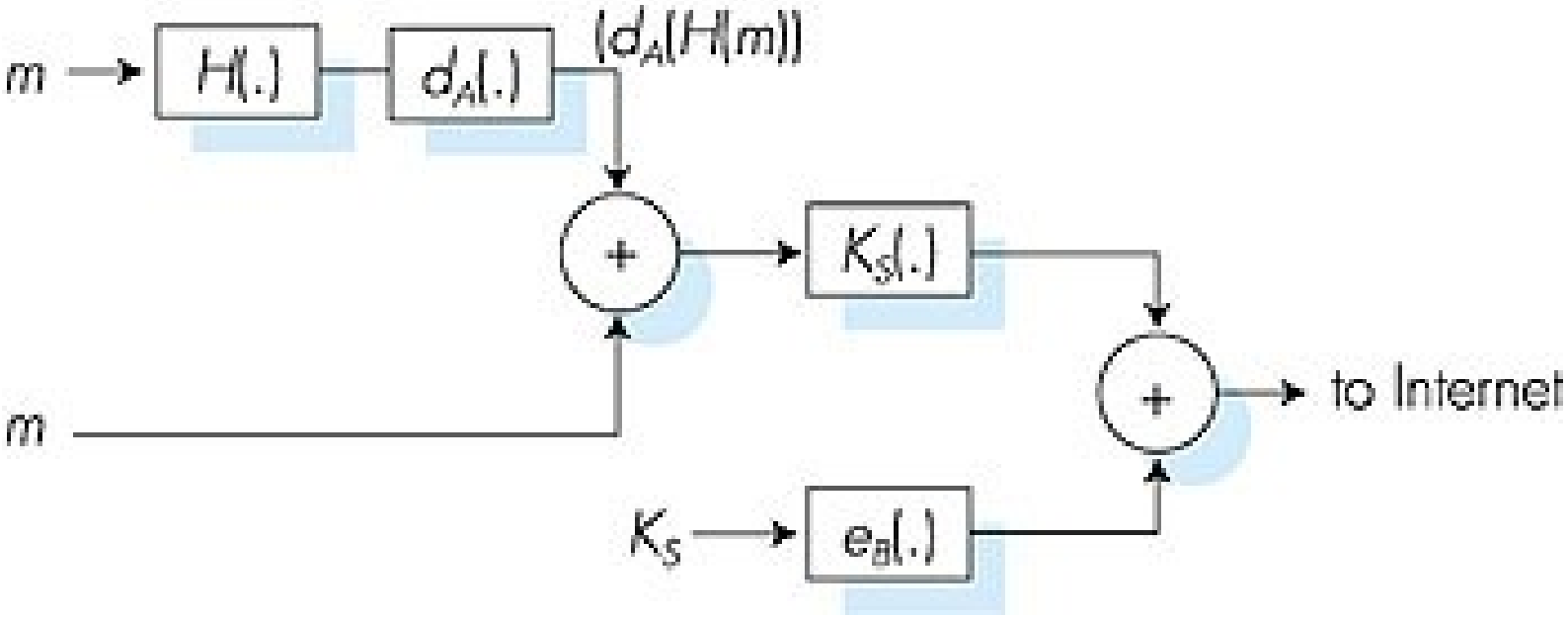
- Alice wants to provide sender authentication message integrity.



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

Secure e-mail (continued)

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



Note: Alice uses both her private key, Bob's public key.

Pretty good privacy (PGP)

- Internet e-mail encryption scheme, a de-facto standard.
- Uses symmetric key cryptography, public key cryptography, hash function, and digital signature as described.
- Provides secrecy, sender authentication, integrity.
- Inventor, Phil Zimmerman, was target of 3-year federal investigation.

A PGP signed message:

```
---BEGIN PGP SIGNED MESSAGE---
Hash: SHA1

Bob:(secret message)


---BEGIN PGP SIGNATURE---
Version: PGP 5.0
Charset: noconv
yhHJRHhGJGhgg/12EpJ+lo8gE4vB3mqJ
   hFEvZP9t6n7G6m5Gw2
---END PGP SIGNATURE---
```

Pretty good privacy (PGP)

- Freely available on http://web.mit.edu/network/pgp.html
- Look also www.pgp.com
- Zimmermann has received technical awards
  - 2001: he was inducted into the CRN Industry Hall of Fame
  - 2000: InfoWorld named him one of the Top 10 Innovators in E-Business
  - 1999: Louis Brandeis Award from Privacy International
  - 1998: Lifetime Achievement Award from Secure Computing Magazine
  - 1996: the Norbert Wiener Award from Computer Professionals for promoting the responsible use of technology.

Secure Protocols

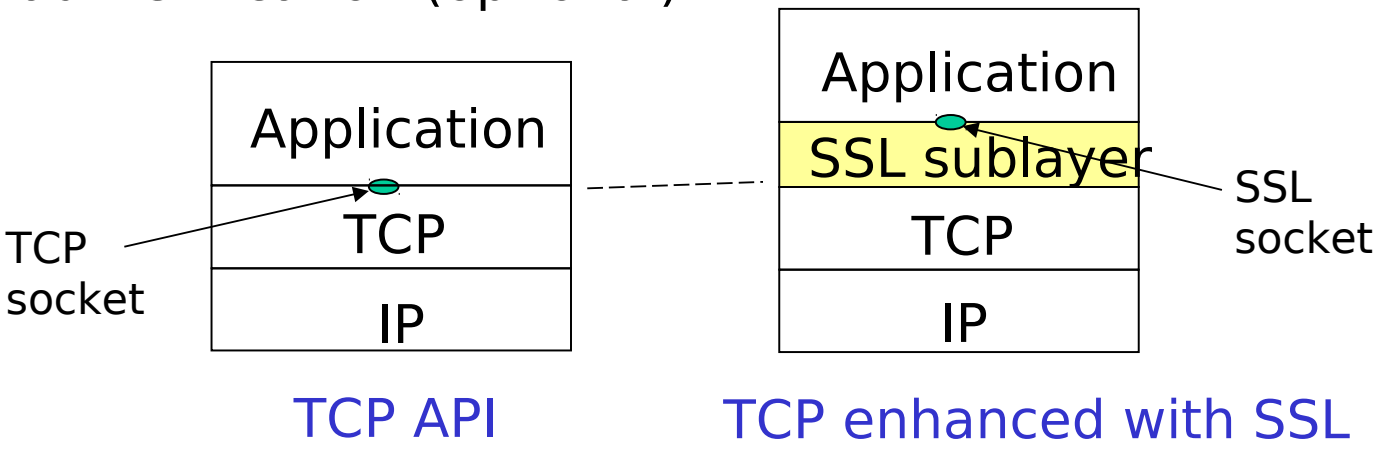
Protocols that do not use encryption are inherently insecure!

Secure sockets layer (SSL)

- PGP provides security for a specific network app.
- SSL works at transport layer. Provides security to any TCP-based app using SSL services.
- SSL: used between WWW browsers, servers for I-commerce (shttp).
- SSL security services:
  - server authentication
  - data encryption
  - client authentication (optional)
- Server authentication:
  - SSL-enabled browser includes public keys for trusted CAs.
  - Browser requests server certificate, issued by trusted CA.
  - Browser uses CA's public key to extract server's public key from certificate.
- Visit your browser's security menu to see its trusted CAs.

Secure sockets layer (SSL)

- provides transport layer security to any TCP-based application using SSL services.
  - e.g., between Web browsers, servers for e-commerce (shttp)
- security services:
  - server authentication, data encryption, client authentication (optional)



SSL (continued)

Encrypted SSL session:

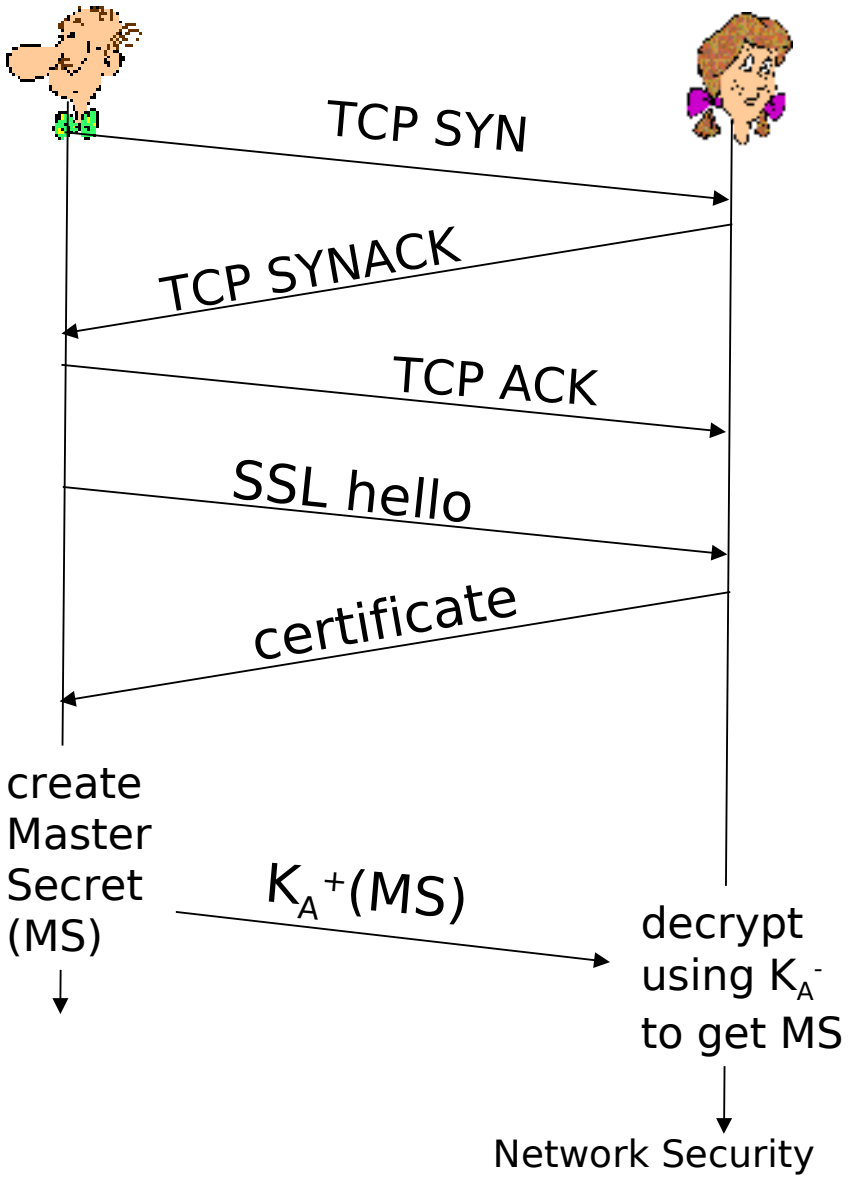
- Browser generates symmetric session key, encrypts it with server's public key, sends encrypted key to server.
- Using its private key, server decrypts session key.
- Browser, server agree that future msgs will be encrypted.
- All data sent into TCP socket (by client or server) encrypted with session key.

- SSL: basis of IETF Transport Layer Security (TLS).
- SSL can be used for non-Web applications, e.g., IMAP.
- Client authentication can be done with client certificates.

SSL: three phases

1. Handshake:

- Bob establishes TCP connection to Alice
- authenticates Alice via CA signed certificate
- creates, encrypts (using Alice's public key), sends master secret key to Alice
  - nonce exchange not shown



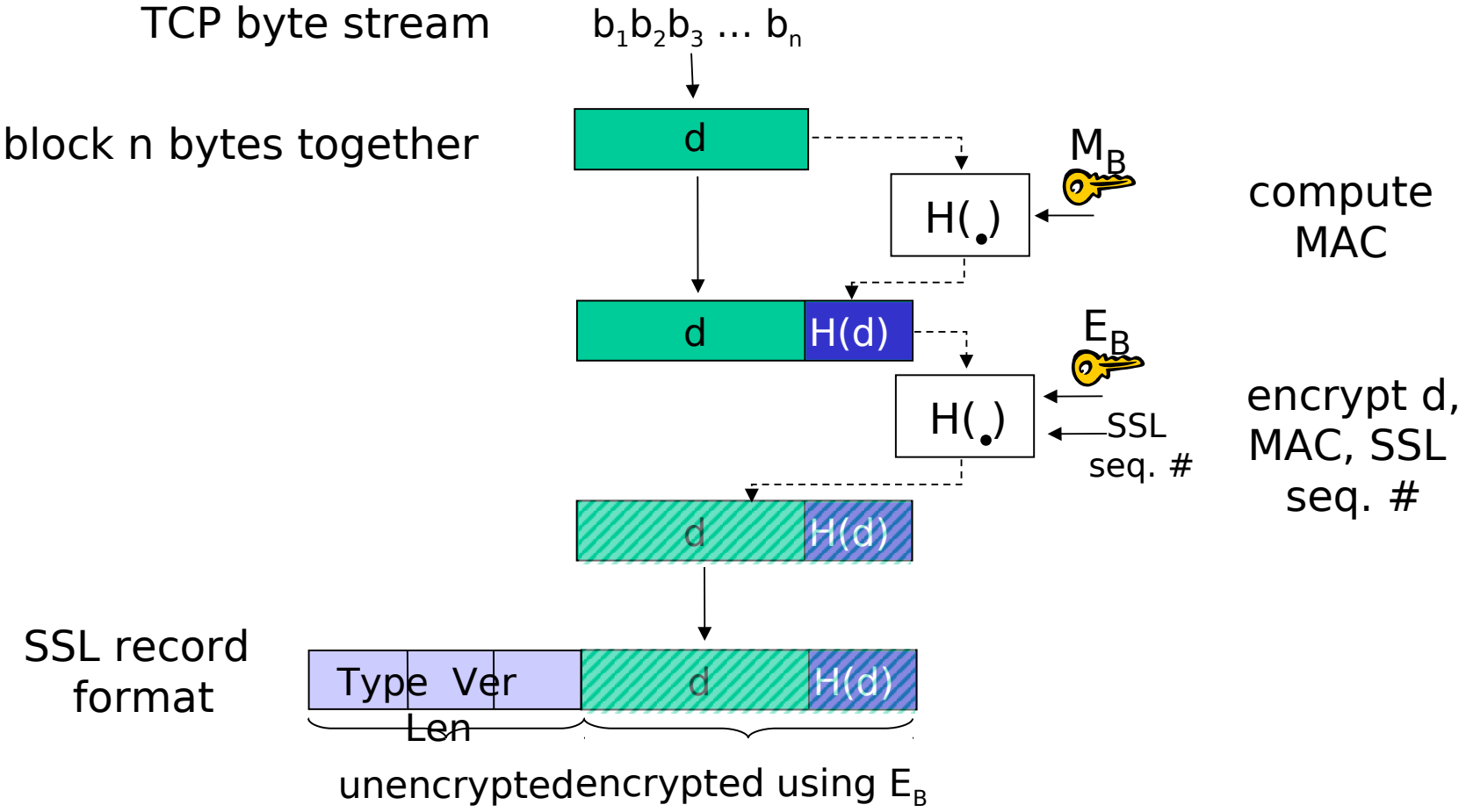
SSL: three phases

2. Key Derivation:

- Alice, Bob use shared secret (MS) to generate 4 keys:
  - E_B: Bob->Alice data encryption key
  - E_A: Alice->Bob data encryption key
  - M_B: Bob->Alice MAC (Message Authentication Code) key
  - M_A: Alice->Bob MAC key
- encryption and MAC algorithms negotiable between Bob, Alice
- why 4 keys?

SSL: three phases

3. Data transfer



Secure electronic transactions (SET)

- designed for payment-card transactions over Internet.
- provides security services among 3 players:
  - customer
  - merchant
  - Merchant's bank

  All must have certificates.
- SET specifies legal meanings of certificates.
  - apportionment of liabilities for transactions
- Customer's card number passed to merchant's bank without merchant ever seeing number in plain text.
  - Prevents merchants from stealing, leaking payment card numbers.
- Three software components:
  - Browser wallet
  - Merchant server
  - Acquisition gateway
- See text for description of SET transaction.

SSH: an example of secure connection

- Telnet or rsh are not secure
- They transmit login/passwords over the network
- SSH is safer because it encrypts the login/password
- Authenticates the hosts
- Keeps keys on the local user's directory
- Example of know_hosts file:
  - `hostname1,130.113.118.147 ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAIEAsmnfyxDMN7o1UrXuvjchDDFGRVdwRLVC+/pVoXvrVl5Byxp/GQSdWJeYzMyEyKaNQ+IgFpiBGqnsgfk8uQJCzyJnB3nkYSAhVlz2emjuC6kuJ8yFgoIxON4v9NVEeSgSEIua6aVBi4a4tfy2sSj15aYzWPSOmJoG+hnt6lEaDY0`

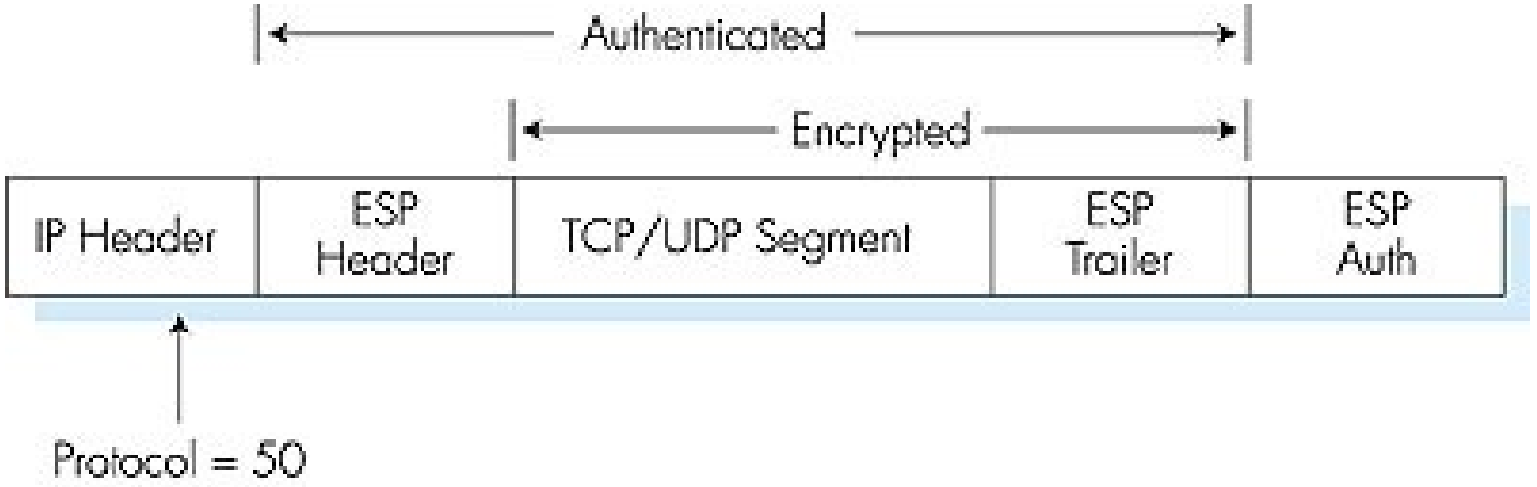
Ipsec: Network Layer Security

- Network-layer secrecy:
  - sending host encrypts the data in IP datagram
  - TCP and UDP segments; ICMP and SNMP messages.
- Network-layer authentication
  - destination host can authenticate source IP address
- Two principle protocols:
  - authentication header (AH) protocol
  - encapsulation security payload (ESP) protocol

- For both AH and ESP, source, destination handshake:
  - create network-layer logical channel called a service agreement (SA)
- Each SA unidirectional.
- Uniquely determined by:
  - security protocol (AH or ESP)
  - source IP address
  - 32-bit connection ID

ESP Protocol

- Provides secrecy, host authentication, data integrity.
- Data, ESP trailer encrypted.
- Next header field is in ESP trailer.
- ESP authentication field is similar to AH authentication field.
- Protocol = 50.



Authentication Header (AH) Protocol

- Provides source host authentication, data integrity, but not secrecy.
- AH header inserted between IP header and IP data field.
- Protocol field = 51.
- Intermediate routers process datagrams as usual.

AH header includes:

- connection identifier
- authentication data: signed message digest, calculated over original IP datagram, providing source authentication, data integrity.
- Next header field: specifies type of data (TCP, UDP, ICMP, etc.)

Wireless Security

Wireless networks without encryption??

IEEE 802.11 security

- *war-driving:* drive around Bay area, see what 802.11 networks available?
  - More than 9000 accessible from public roadways
  - 85% use no encryption/authentication
  - packet-sniffing and various attacks easy!
- securing 802.11
  - encryption, authentication
  - first attempt at 802.11 security: Wired Equivalent Privacy (WEP): a failure
  - current attempt: 802.11i

Wired Equivalent Privacy (WEP):

- authentication as in protocol *ap4.0*
  - host requests authentication from access point
  - access point sends 128 bit nonce
  - host encrypts nonce using shared symmetric key
  - access point decrypts nonce, authenticates host
- no key distribution mechanism
- authentication: knowing the shared key is enough

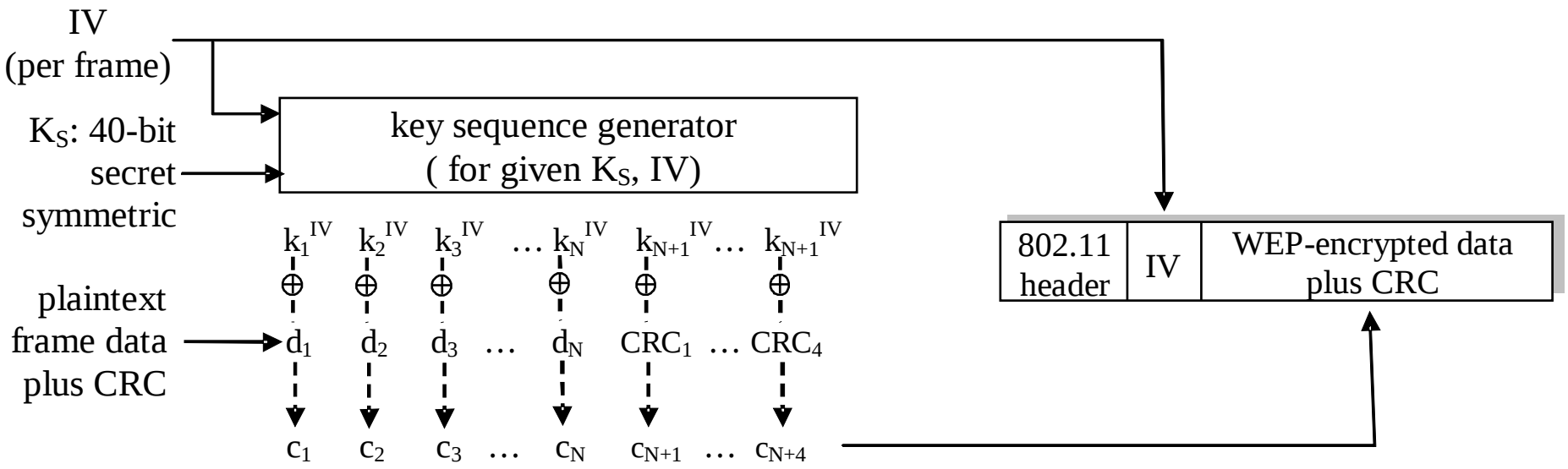
WEP data encryption

- host/AP share 40 bit symmetric key (semi-permanent)
- host appends 24-bit initialization vector (IV) to create 64-bit key
- 64 bit key used to generate stream of keys, k_i^{IV}
- k_i^{IV} used to encrypt ith byte, d_i, in frame:

$$c_i = d_i \text{ XOR } k_i^{IV}$$

- IV and encrypted bytes, c_i sent in frame

802.11 WEP encryption



Sender-side WEP encryption

Breaking 802.11 WEP encryption

security hole:

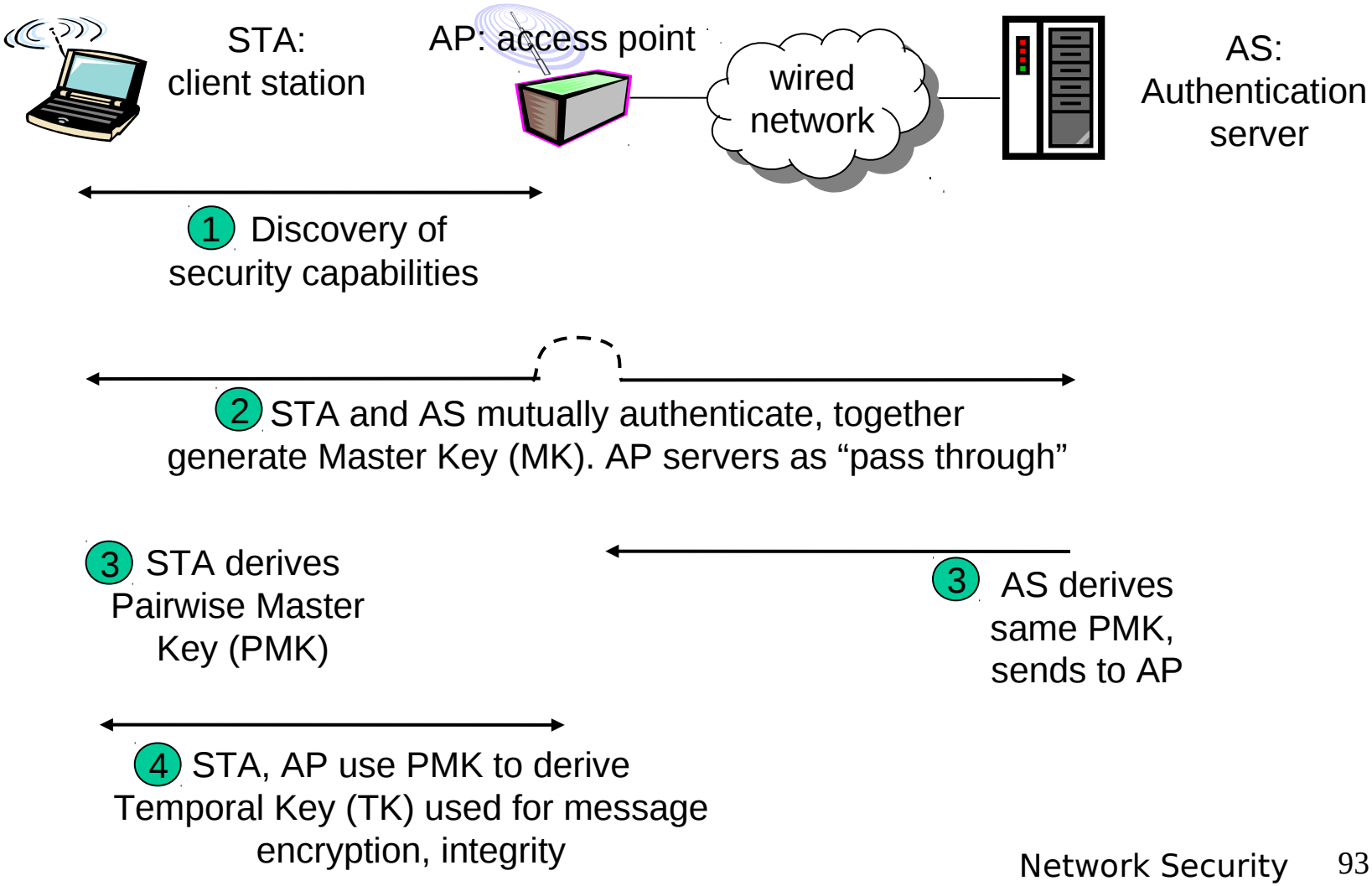
- 24-bit IV, one IV per frame, -> IV's eventually reused
- IV transmitted in plaintext -> IV reuse detected

- attack:
  - Trudy causes Alice to encrypt known plaintext d_1 d_2 d_3 d_4 …
  - Trudy sees: $c_i = d_i$ XOR k_i^{IV}
  - Trudy knows c_i d_i, so can compute k_i^{IV}
  - Trudy knows encrypting key sequence $k_1^{IV} k_2^{IV} k_3^{IV}$…
  - Next time IV is used, Trudy can decrypt!

802.11i: improved security

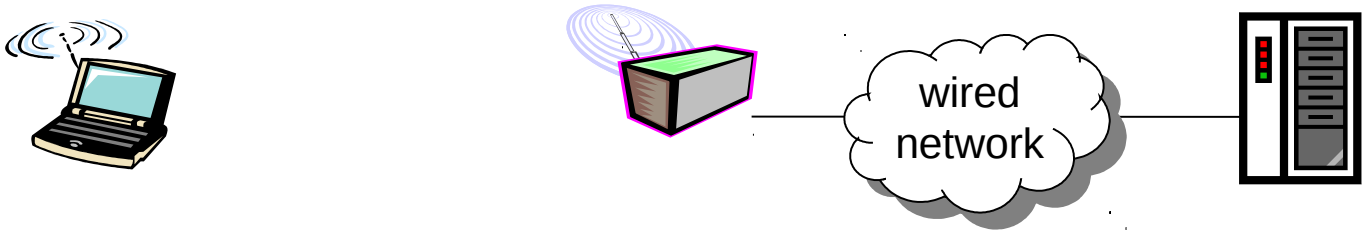
- numerous (stronger) forms of encryption possible
- provides key distribution
- uses authentication server separate from access point

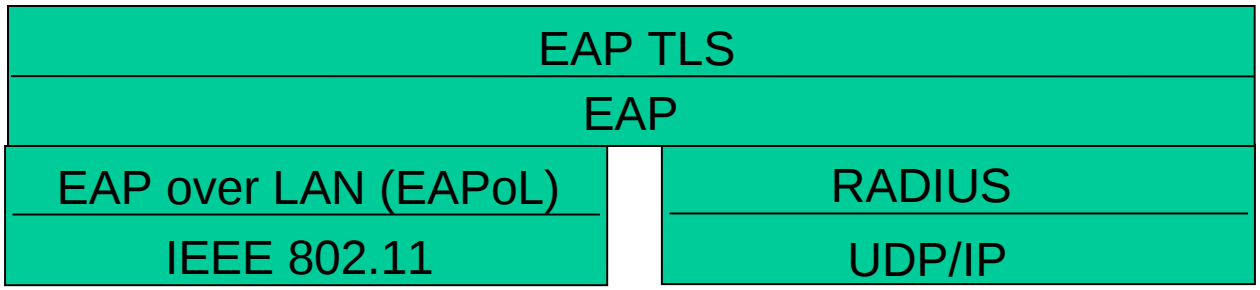
802.11i: four phases of operation



EAP: extensible authentication protocol

- EAP: end-end client (mobile) to authentication server protocol
- EAP sent over separate “links”
  - mobile-to-AP (EAP over LAN)
  - AP to authentication server (RADIUS over UDP)





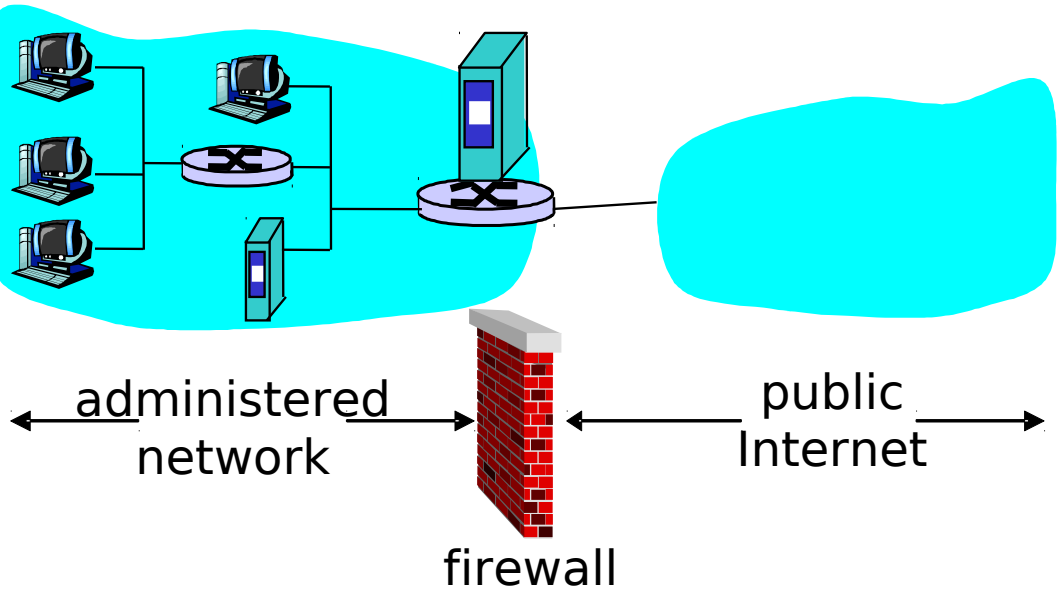
Firewalls

... and the doors in the back of the networks...

Firewalls

firewall

isolates organization's internal net from larger Internet, allowing some packets to pass, blocking others.



Firewalls: Why

prevent denial of service attacks:

- SYN flooding: attacker establishes many bogus TCP connections, no resources left for “real” connections

prevent illegal modification/access of internal data.

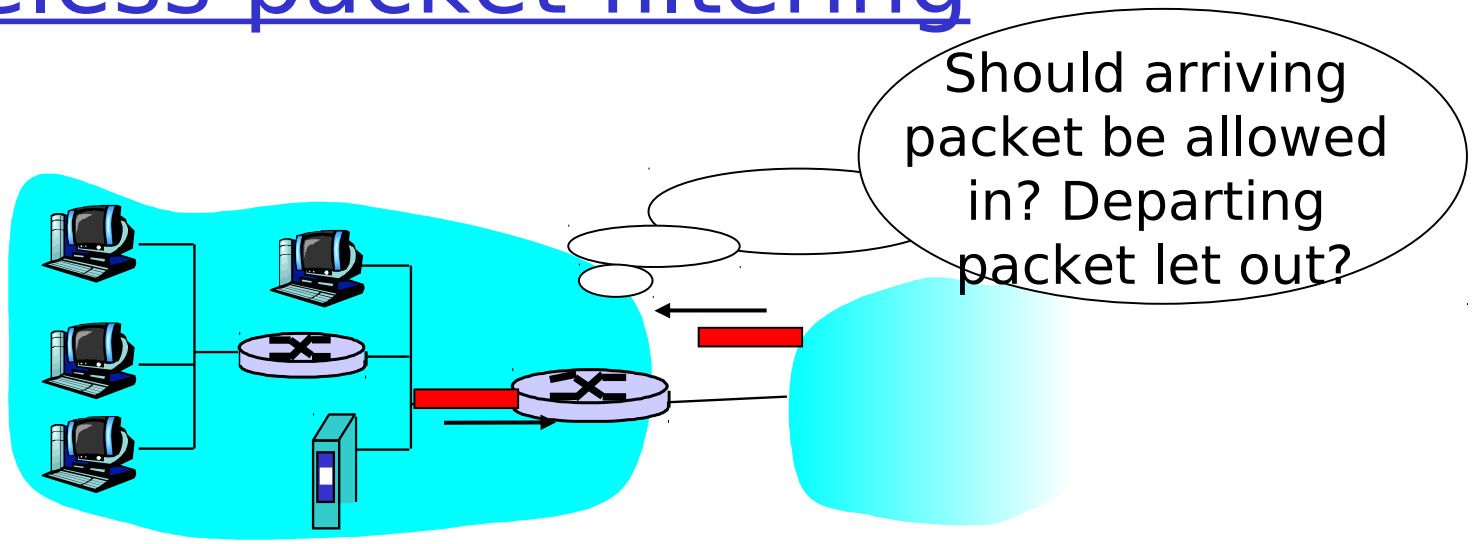
- e.g., attacker replaces CIA’s homepage with something else

allow only authorized access to inside network (set of authenticated users/hosts)

three types of firewalls:

- stateless packet filters
- stateful packet filters
- application gateways

Stateless packet filtering



- internal network connected to Internet via router firewall
- router filters packet-by-packet, decision to forward/drop packet based on:
  - source IP address, destination IP address
  - TCP/UDP source and destination port numbers
  - ICMP message type
  - TCP SYN and ACK bits

Stateless packet filtering: example

- example 1: block incoming and outgoing datagrams with IP protocol field = 17 and with either source or dest port = 23.
  - all incoming, outgoing UDP flows and telnet connections are blocked.
- example 2: Block inbound TCP segments with ACK=0.
  - prevents external clients from making TCP connections with internal clients, but allows internal clients to connect to outside.

Stateless packet filtering: more examples

Policy	Firewall Setting
No outside Web access.	Drop all outgoing packets to any IP address, port 80
No incoming TCP connections, except those for institution's public Web server only.	Drop all incoming TCP SYN packets to any IP except 130.207.244.203, port 80
Prevent Web-radios from eating up the available bandwidth.	Drop all incoming UDP packets - except DNS and router broadcasts.
Prevent your network from being used for a smurf DoS attack.	Drop all ICMP packets going to a "broadcast" address (eg 130.207.255.255).
Prevent your network from being tracerouted	Drop all outgoing ICMP TTL expired traffic

Access Control Lists

- *ACL:* table of rules, applied top to bottom to incoming packets: (action, condition) pairs

action	source address	dest address	protocol	source port	dest port	flag bit
allow	222.22/16	outside of 222.22/16	TCP	> 1023	80	any
allow	outside of 222.22/16	222.22/16	TCP	80	> 1023	ACK
allow	222.22/16	outside of 222.22/16	UDP	> 1023	53	---
allow	outside of 222.22/16	222.22/16	UDP	53	> 1023	----
deny	all	all	all	all	all	all

Stateful packet filtering

- stateless packet filter: heavy handed tool
  - admits packets that “make no sense,” e.g., dest port = 80, ACK bit set, even though no TCP connection established:

action	source address	dest address	protocol	source port	dest port	flag bit
allow	outside of 222.22/16	222.22/16	TCP	80	> 1023	ACK

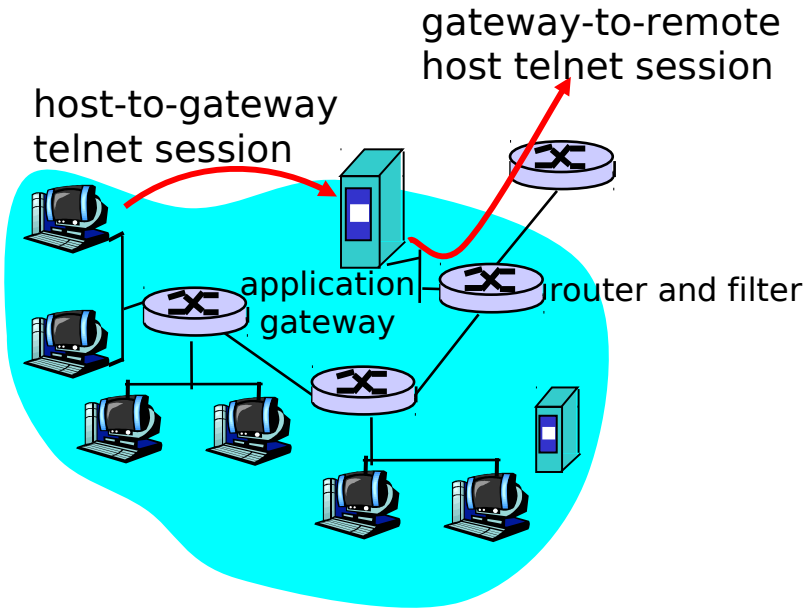
- *stateful packet filter:* track status of every TCP connection
  - track connection setup (SYN), teardown (FIN): can determine whether incoming, outgoing packets “makes sense”
  - timeout inactive connections at firewall: no longer admit packets

Stateful packet filtering

- ACL augmented to indicate need to check connection state table before admitting packet

action	source address	dest address	proto	source port	dest port	flag bit	check conxion
allow	222.22/16	outside of 222.22/16	TCP	> 1023	80	any	
allow	outside of 222.22/16	222.22/16	TCP	80	> 1023	ACK	X
allow	222.22/16	outside of 222.22/16	UDP	> 1023	53	---	
allow	outside of 222.22/16	222.22/16	UDP	53	> 1023	----	X
deny	all	all	all	all	all	all	

Application gateways



- filters packets on application data as well as on IP/TCP/UDP fields.
- example: allow select internal users to telnet outside.
  1. require all telnet users to telnet through gateway.
  2. for authorized users, gateway sets up telnet connection to dest host. Gateway relays data between 2 connections
  3. router filter blocks all telnet connections not originating from gateway.

Limitations of firewalls and gateways

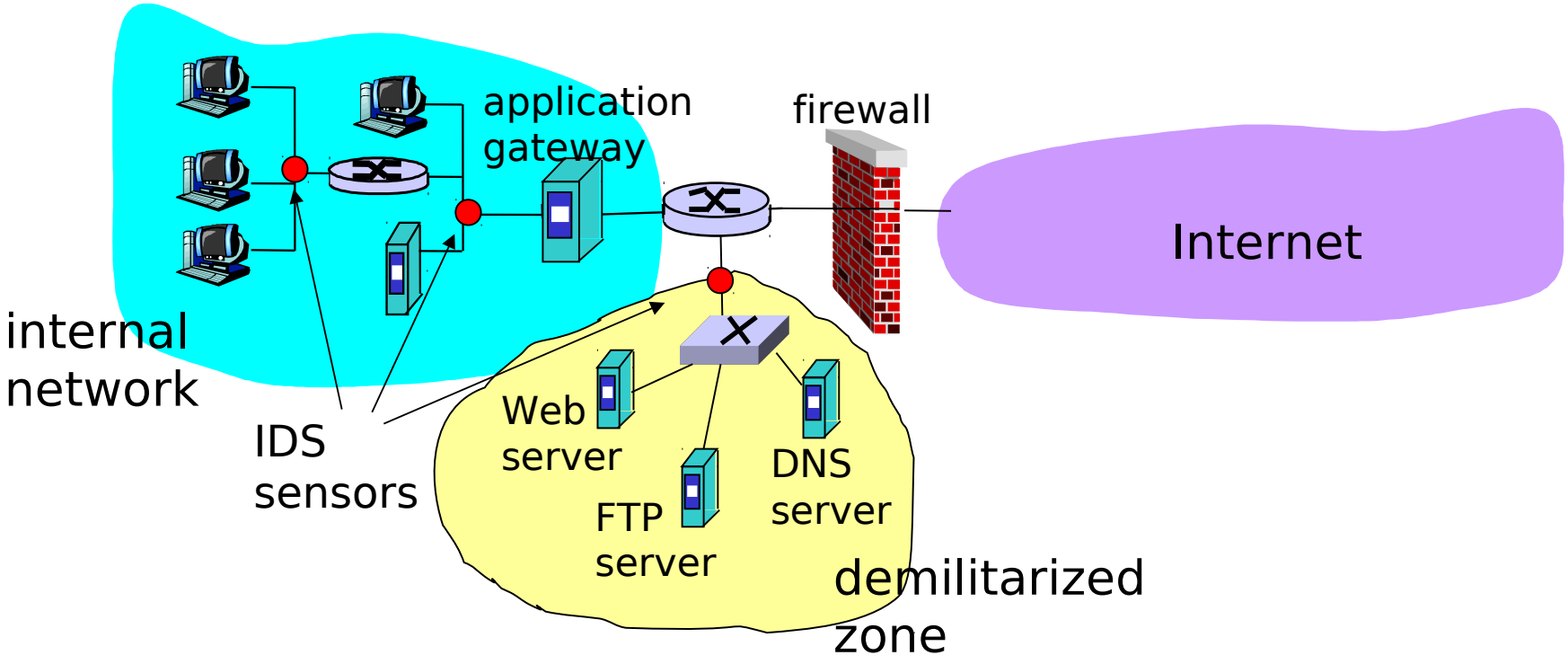
- IP spoofing: router can't know if data "really" comes from claimed source
- if multiple app's. need special treatment, each has own app. gateway.
- client software must know how to contact gateway.
  - e.g., must set IP address of proxy in Web browser
- filters often use all or nothing policy for UDP.
- tradeoff: degree of communication with outside world, level of security
- many highly protected sites still suffer from attacks.

Intrusion detection systems

- packet filtering:
  - operates on TCP/IP headers only
  - no correlation check among sessions
- *IDS: intrusion detection system*
  - *deep packet inspection:* look at packet contents (e.g., check character strings in packet against database of known virus, attack strings)
  - examine correlation among multiple packets
    - port scanning
    - network mapping
    - DoS attack

Intrusion detection systems

- multiple IDSs: different types of checking at different locations



Network Security (summary)

Basic techniques……

- cryptography (symmetric and public)
- authentication
- message integrity

…. used in many different security scenarios

- secure email
- secure transport (SSL)
- IP sec