

Network security

Lecture 9 Network security

References:

-

- Internet is **designed** at a time when security is **not the main issue**.
When it is shared only by trusted universities to share information, there is really not very important to make sure things won't go to the wrong hands.
 - When companies and the public rush into it, many questions arise...
 - Can **other** companies connects to and **misuse** my internal servers?
 - Can the **wrong staff** of my own company does the same?
 - If I connect to a sub-branch through the Internet, will people in the middle be able to **eavedrop** to it?
 - Can they **modify** the words I send to the other side?
 - Can somebody **pretend** to be the one I talk to, so I'm talking to the wrong person from the beginning?
- Unluckily, all answers are "yes" unless something is done to prevent them.

POS(0234B)

POS(0234B)-9.1

Host based authentication

- Let's start with something **simple** (but achieves little).
- A C library can provide functions to **checks the IP address and DNS names** of the other party before actual communication.
E.g., `tcpd`, the TCP wrapper library. See `hosts_access(3)`.
- The system administrator can then **configure** who can connect to which computer by writing some files.
In case of `tcpd`, `/etc/hosts.allow` and `/etc/hosts.deny`.
- This solves can first problem about **where** can a connection starts... with considerable administration overheads.
Every program in every computer in an organization must be configured correctly. Modern OS will deny most access by default to make it more manageable.
- The **who** problem, and the other problems when communicating with the outside, must be solved by other means.

POS(0234B)-9.2

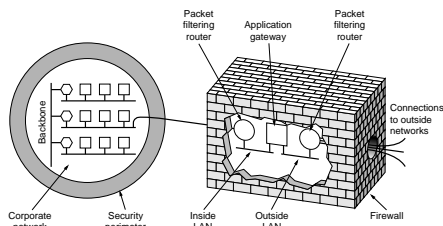
Packet filter and firewall

- A more drastic solution: do it at **network** or **transport** layer: **routers**.
- Using this solution, packets from or to the wrong place are simply **dropped** at the middle of the connection.
- Usually, an organization has a **single router** connecting to the external world. That router, called a **firewall**, can be configured to drop packets.
- E.g., if a company use the TCP/IP protocol only internally, it can **deny every packet** with source or destination outside the company.
Or cutting the outgoing connection. Security is easy if nobody need the service.
- Most companies will want some service from outside (e.g., web, ftp, mail). If this is the case, the company can keep packets **from internal non-privileged ports** to selected external ports (e.g., http, ftp, smtp).
Why we must limit to non-privileged ports?
- What about companies that **also provide network service**?

POS(0234B)-9.3

Security perimeter

The idea: Make sure connections from outside are only made to an **application gateway**, which may **redirect** it to one computer in the internal network.



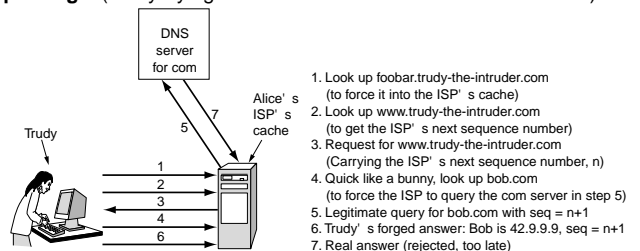
This way we establish a **security perimeter**, where **incoming connections** are only **redirected** from the application gateway.

Appl gateway inspects the appl level headers to see whether to allow or discard it.

POS(0234B)-9.4

DNS spoofing

Host-based authentication and fire wall schemes trust the identity of the host, i.e., IP address is "real". This can be a source of attack: **DNS spoofing**... (Trudy trying to make Alice believe bob.com is 42.9.9.9)



DNS with wrong information in cache is called poisoned DNS cache. At times it is better to hard-code IP addresses rather than relying on DNS. Correspondingly, hardware address of the same network can be spoofed by poisoning ARP cache.

POS(0234B)-9.5

Other attacks: DoS, DDoS

A type of attack that is **very** difficult to protect against: Denial of Service.

- The idea: the attacker simply **send a huge number of requests** to overwhelm your server.
But this usually leave connection which can be traced back to the intruder.
- One form of DoS: send a SYN TCP connection request, ignoring the SYN+ACK reply. The server keeps a table to remember them; once this overflows, further connections are refused. (**SYN attack**.)
Leaves no trace in the network, and indeed the original SYN can be from a non-existent IP address! This can be dealt with in various ways, e.g., dropping random SYN identifier from the table when it overflows.
- A even more difficult one: **Distributed DoS (DDoS)**. The intruder **cracks many computers** (e.g., by worms) and ask them to send a lot of request to the **same server**.
No real effective way to deal with it except to shut it down and call police.

POS(0234B)-9.6

What problem a fire wall really solves

It surprising how **little** a fire wall achieves.

- Firewalls can't stop **eavesdropping** and **alteration** of packets.
- Firewall might help a little in dealing with **DoS** and **DDoS**, but when enough packets comes **the fire wall or router** is saturated.
But at least the users within the security perimeter is unaffected.
- A fire wall can't stop **virus** and **worms** to get into the security perimeter through E-mails and web servers.
- Once malicious code gets in, it can build a **tunnel** through a **permitted channel** (e.g., the HTTP port) for intruders to control the computer.
Thus attack the other computers in the security perimeter easily.

So security cannot be done by fire wall **alone**. But fire wall does help **delaying** the exploit of mistakes of administrators.
E.g., an intruder may have to wait until the next IIS bug is discovered.

POS(0234B)-9.7

Basic cryptography

- Now let's start thinking about **how to prevent eavesdropping and detect modification** of our message.
- One can deal with this problem by **building a private link** to everybody it want to talk securely with, but it is very **expensive**.
- Another way: use **cryptography**.
- Idea: Exchange messages in a way that **only the recipient understands**. For anyone in the middle, it looks like **random noise**.
- Since the intruder don't understand the random noise, he **cannot get much information** even if he can listen.
- Since the intruder don't know how to **create the correct random noise**, he cannot **change** the message without getting detected.

POS(0234B)-9.8

Kerckhoff's principle

An algorithm which turns data to pseudo-noise and back is called **encryption and decryption** algorithm.

- Kerckhoff's principle**: any reasonable scheme must work even when **encryption and decryption algorithm** are **not kept secret**. Instead, algorithms should make use of **keys** which are secret.
- Why?** It is **very difficult** to construct **good** encryption algorithms, i.e., which produce output which really looks like random noise.
- If **every** user has to design such a scheme, it is likely that most people are using **incorrect** algorithms that are easily broken.
I.e., with a little bit of information about the data, it is possible to recover the complete data by looking at the encrypted data.
- Kerckhoff's principle allows a **few** algorithms to be designed, tested by all experts for weaknesses, and then used by everybody.

POS(0234B)-9.9

Notations

- Alice (A), Bob (B), Trudy (T)**: **initiator** and **recipient** of communication, and an **intruder**.
- Big Brother (BB)**: a person that **everybody trusts**.
Depending on assumptions, Alice might or might not trust Bob.
- Plaintext (P)** and **ciphertext (C)**: **unencrypted** and **encrypted** data to be communicated.
- Keys (K, K_A, K_B)**: **keys** used for encrypting or decrypting data. Subscripts are owner of the secrets.
We will also use them like functions. So $K(P) = C$.
- Encryption K_A^e and decryption E_A^d keys**: Alice's keys for encryption and decryption, if **they differ**.
We drop the superscripts in contexts where it is obvious. Again, we use them like functions, in such a way that $E_A^d(E_A^e(P)) = P$.

POS(0234B)-9.10

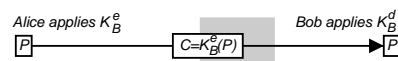
Secrecy with symmetric key and public key algorithms

For symmetric key cryptographic algorithms, **both Alice and Bob knows K**, which must be kept secret.



Given K, both Alice and Bob can do E_K and D_K .

Public key algorithms uses different keys for encryption and decryption.



- Everybody** know K_B^e , not just Alice (or Bob). On the other hand, **nobody** other than Bob knows K_B^d .
- Given K_B^e , it's **computationally infeasible** (takes centuries) to find K_B^d .
Even B, who does the encryption, cannot recover the P if he somehow loss it.

POS(0234B)-9.11

Why 2 schemes

For symmetric key systems, every **pair** of users shares a key. So in total there are n^2 keys for 2 people for n people communicating.

- Alice must make sure the secret shared with Bob is **sent only to Bob** and nobody else.
- And... **how to send?** We need another secure medium. Now everybody should know what is chicken-and-egg problem.

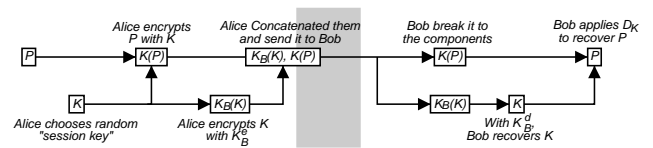
For public key systems, every person has **exactly two** keys.

- Alice just try to protect her **own** decryption key. Everything else she know **need no protection**.
- She can announce her encryption key **in public** for others to talk to her securely. We call the key **public key**.
- There is only 1 problem: public key algorithms are typically **slow**.

POS(0234B)-9.12

Combining the two schemes

The symmetric key system has the merit of being fast, while the public key system has the merit of being easy. Is it possible to combine the merits?



- The slow algorithm E_B and D_B are only used to encrypt and decrypt the session key, which is **short**.
- The fast algorithm E_K is used for encrypting the long data P .
- On **interactive** use, K is often communicated at the **beginning** of a connection using public keys. After that a symmetric key algorithm is used with K as key.

POS(0234B)-9.13

Digital signature

Apart from secrecy, encryption can be used for **signing**.

- Alice creates a document P , and want to send it Bob. But Bob want to know it is **really something written by Alice**.
- So Alice creates a **signature** $S_A(P)$ from P , and sends $(P, S_A(P))$ to Bob. Bob then separates the message and **checks** that P does indeed corresponds to $S_A(P)$.
- A scheme should ensure that
 1. Truly cannot create $S_A(P)$.
 2. Alice shouldn't be able to **deny** that she sent the message (**nonrepudiation**). This requires that Bob cannot create $S_A(P)$.

Alternatively, one can have a scheme with **deniability**, i.e., only Alice and Bob can create the message, and only Alice and Bob know who created it. In network security we seldom have such scenario.

POS(0234B)-9.14

Symmetric key signature

Symmetric schemes are good at the 1st requirement, but bad at the 2nd.

To have nonrepudiation, there must be some secret known by Alice, not Bob. But then **how Bob can check it?** Tough.

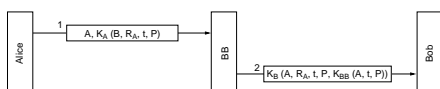
But if there is somebody (Big Brother, or BB) which **everybody trust**:

- With a key K_A **shared with BB**, Alice encrypts the identity of Bob and the message P to be signed. Alice sends $A, E_{K_A}(B, P)$ to BB. BB knows A is sending: no one else can encrypt such message. (Except...)
- BB then decrypt it to recover P and B . BB encrypts (A, P) with a secret key K_{BB} that it shares with nobody, to be used as **signature**. So the signing is actually done by BB.
- BB now encrypt the (A, P) and its signature with a key K_B shared with Bob, and send him the result $E_{K_B}(A, P, E_{K_{BB}}(A, P))$. Bob knows that BB is sending, and trust BB has checked Alice was sending.

POS(0234B)-9.15

Symmetric key signature (cont'd)

- If Bob trust BB, BB knows that Alice is sending and tells Bob, then **Bob knows as well** that Alice is sending.
- Why Alice can't deny? Because **only BB can create** $K_{BB}(A, P)$, not Bob. If Alice deny, BB can decrypt it and get (A, P) . Thus showing sometimes in the past, BB trusts that Alice sends P .
- What about **replays**? Just **add a timestamp** t to P so that Bob can **reject very old messages**, and add a random number R_A to the dialog with A and B to distinguish between **recent** messages:



- Bob **keeps a list of recent** R_A 's. If it is seen again, it is simply discarded as duplicate.

POS(0234B)-9.16

Public key signature

- Public key signature **significantly simplify** the problem.
- The idea: Alice just have **another pair of key** for signing. In this case, the **decryption** key is **public**, while the **encryption** key is **private**. Some public key algorithms has the property that $E_A(D_A(P)) = P$. In this case the same public key for secrecy can be re-used (reversing the role of D and E).
- To send P signed, she can send $E_A(P)$ to Bob.
- Bob **decrypts** the message using D_A , get P , and conclude that it must come from Alice (no one else can encrypt such a message). Everybody can do what Bob does, so there is really no secrecy although the message is encrypted. If secrecy is desired, it can be encrypted with **Bob's** public encryption key.
- Again, there is problem of efficiency: this requires the whole P to be encrypted with the **slow** public-key algorithm. The world will be much simpler if public-key algorithms are fast.

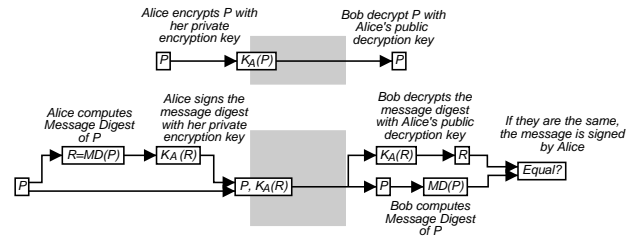
POS(0234B)-9.17

Message digest

- This time **symmetric key doesn't come as rescue**, as signature using symmetric key needs a BB which we don't want to require.
- What we want: **use public key to sign only small data**, which derives from every bit of a large message.
- What about just **signing the checksum** of the large message? Alice creates P , computes the checksum R , and sends $(P, E_A(R))$. Then Bob will believe that Alice sent a message with such a checksum.
- But there is a problem: even if Bob believes that, **Bob needs not believe the message is P !**
Bob's reasoning: Trudy could have made a different message P' with the same checksum, and sent Bob $(P', E_A(R))$, after seeing Alice sends $(P, E_A(R))$.
- For Trudy to attack, she must find $P' \neq P$ with a fixed R . A "checksum" algorithm which this takes forever to find is said to be a **one-way (or trap-door) function**. The result is called a **message digest**.

POS(0234B)9.18

Public key signing with and without message digest



How long is $MD(P)$? Currently, usually 128, 160 or 256 bits.

Trudy can create arbitrary message P' and find $MD(P')$. Her chance of ending up with R should be 2^{-128} to 2^{-256} , depending on key length.

I.e., she has to try at least 2^{128} times to break it.

But it is much weaker if Alice is just **approving** that message...

POS(0234B)9.19

Birthday attack

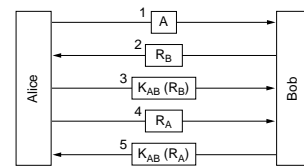
- If Alice creates a message P and Trudy needs to find another message P' such that $MD(P) = MD(P')$, it is really that difficult.
- But if Trudy is to find **two** messages P and P' of **opposite** meaning but with **matching message digest**, it would be much easier.
- Trudy would create 2^{64} messages of the same meaning, and 2^{64} ones of the same opposite meaning. Most likely **a pair have the same MD**.
Intuitively, because there are then $(2^{64})^2$ pairs that might match. It is called birthday attack due to the phenomenon used in teaching probability: if there are 23 people, it is likely that some will have matching birthday.
- But Trudy needs 2^{128} time for matching all the pairs, right? Wrong. She just **keep a hash table** of 2^{64} entries, so each match is $O(1)$ time.
So $O(2^{64})$ time to create the hashes and the same order of time to check them.
- If Trudy sends Alice P for **approval** and send $(P, E_A(MD(P)))$, Trudy can modify it to $(P', E_A(MD(P)))$ without getting caught.

POS(0234B)9.20

Authentication with symmetric keys

Authentication is similar to **signing**, except that we **don't need unreputation**, but we need to be able to handle **replay** in a way that doesn't rely on a clock. This usually means **challenge-response**.

Will a simple symmetric key protocol works without BB? E.g.,



Each tries to **challenge** the other side by sending a random number to encrypt, and is trusted if the other side correctly encrypt (**respond to**) it.

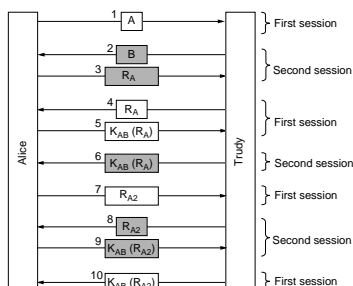
Trudy can't do the encryption, since she doesn't know the secret. Right?

POS(0234B)9.21

Reflection attack

Wrong if Bob can **initiate** a connection to Alice using the **same protocol**.

Trudy wait until Alice initiate a connection, and then initiate a connections with Alice. All Alice's answers are **reflected** back to Alice.



POS(0234B)9.22

How to prevent that?

Problem: answering the response can be treat as **providing a service** called *encrypting with the secret key*. Trudy can thus use this service.

How to prevent that? Make the "service" useless for Trudy.

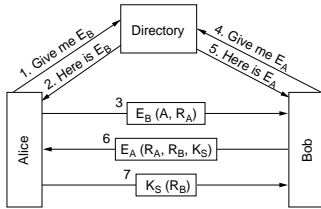
- Require **Alice** to prove who she was **before** Bob give out valuable info. Our scheme has that. So the other rules are still essential.
- Alice and Bob should use **different** shared keys. So they should share 2 different keys, not only 1.
- Alice and Bob should select challenges from **different** set.
E.g., Challenge of Alice is always even, and Bob's challenge is always odd. Or append the ID of the one making the challenge.

POS(0234B)9.23

Authentication with public keys

Again, public keys **simplify** provably correct implementation. Bob is trusted if he can decrypt Alice's challenge and send it back, encrypted.

Most implementation exchange a **session key** at the end, which will be used for further communication in the same connection. One example:



A directory is used, so Alice need not know Bob at the beginning. The directory needs to be authenticated, but since there is only 1 directory this is easy.

POS(0234B)-9.24

The idea of Certificates

- There is a way to **avoid contacting the directory**: Alice can send Bob E_A , and Bob can send Alice E_B .
- But... then Trudy can just send her public key but say she's Alice, and Bob will encrypt anything using Trudy's public key!
- How to prevent this? If a **trusted person signs** a message saying E_A is Alice's public key, then Alice can send the signed message to Bob.
- So the "trusted person" is a **central point to check the identity** of each user.
- The trusted person is known as a **Certification Authority (CA)**, and the message " E_A is Alice's public key" is called Alice's **certificate**.
- The protocol: Alice sends her certificate to Bob, Bob sends his certificate to Alice, and then step 3, 6 and 7 continues.
There is nothing secret about certificates .

POS(0234B)-9.25

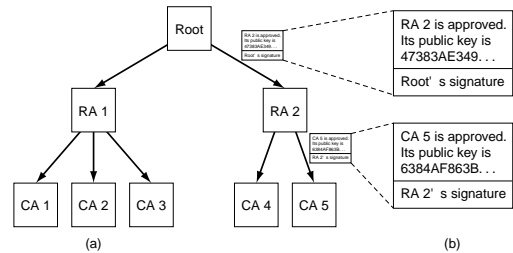
Certification of certification

- There is one remaining problem: **nobody is so trustworthy** that everybody trusts!
And nobody can sustain the workload to check the identity of everybody else.
- The answer: a few **root CAs** issuing certification to **regional Authorities (RAs)**, in a signed message like " E_{RA1} is the public key of RA1, who is approved to sign document in region 1".
- Then RA1 can sign public keys for Alice and Bob. The certification of RA1 is also given to them.
- When Alice and Bob send their certificates, they **also send the certifications of their RAs**.
- All the certifications are in the **same format** (X.509, described in RFC3280), so programs can perform the checks automatically.

POS(0234B)-9.26

Public Key Infrastructure (PKI)

This scheme can be further extended to form a tree (actually forest, since there are multiple roots) of CAs. We call such a collection of authorities a **Public Key Infrastructure (PKI)**.



When checking signatures, whole chain of authentication must be checked, from a root CA that one trusts.

POS(0234B)-9.27

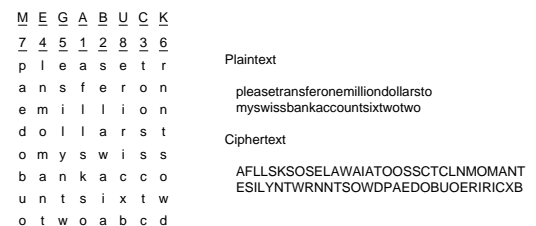
Substitution ciphers

- Now let's see some old and new approaches to symmetric ciphers.
- **Caesar cipher**: key K is a number between 1 to 25, the cipher change every letter to the K -th letter after it in the alphabet.
So if $K = 3$, K ("attack") = "dwwdfn". Very weak: can be trivially broken by trying all 25 possibilities.
- **Monoalphabetic cipher**: key K is a permutation of the 26 characters, the cipher changes the i -th letter in the alphabet by the i -th letter in K .
So if $K = "pyfgcrlaoeuidhtnsqjkbxmvvz"$, K ("attack") = "pkpflu".
- Even monoalphabetic cipher (with a key space of $26!$) is **weak**: it doesn't change the distribution. E.g., after encrypting text, the most frequent characters are likely to correspond to 'e', 't', 'o', etc.
- Problem: 8-bit alphabet is too small. A **larger** (say 128-bit) **alphabet** will make it unbreakable in practice, but will be **very difficult to use**.
It will have huge key space, i.e., huge keys.

POS(0234B)-9.28

Transposition ciphers

- Another approach: shuffle the characters so that no one except the right person knows the ordering to read the message.



- The key can be viewed as a permutation of the 1–64, specifying the position **where each letter goes**.
- Since this doesn't change the probability distribution of letters, it is again quite **easy to break**.

POS(0234B)-9.29

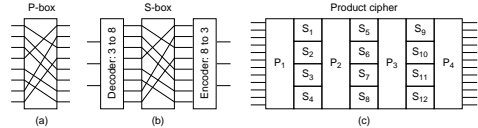
One Time Pad (OTP)

- Let's see a **provably unbreakable** cipher.
- Key: a **long sequence of random bits** shared by Alice and Bob.
- To encrypt a n-bit message, Alice extract the first n bits of her random bit sequence, **XOR** it with the message, and discard the used random bits.
- To decrypt a n-bit ciphertext, Bob **XOR** it with the first n bits of the his random bit sequence, and discard the used random bits.
- Each random bit is **used once**, so it's called One Time Pad (OTP).
- If the sequence is really random, then the ciphertext is just as random (has equal chance to be 0 or 1, independent on any other bit). So the code is unbreakable.
- But this requires Alice and Bob to **share a long sequence of random bits**, which is basically **impossible**.

POS(0234B)-9.30

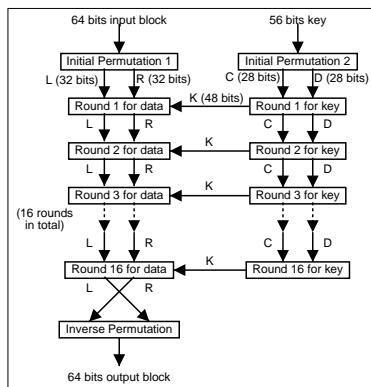
Modern approach

- Modern ciphers: **heavily** strengthen the weak ciphers, and use hardware to encrypt and decrypt.
- Substitution is usually done by **xor** the bits in plaintext with bits in key, although other methods are also used. Standard hardware does that.
- Transposition is usually done to **bits**, using a **fixed** ordering. This can be realized by hardware called a P-box.
- It is strengthened significantly by **repeating the process** many times. The idea: each bit in the plaintext should influence all bits of ciphertext. Important in message digest as well. Their design is very similar to ciphers.



POS(0234B)-9.31

Example: DES



DES works in 16 rounds, best illustrated as a circuit.

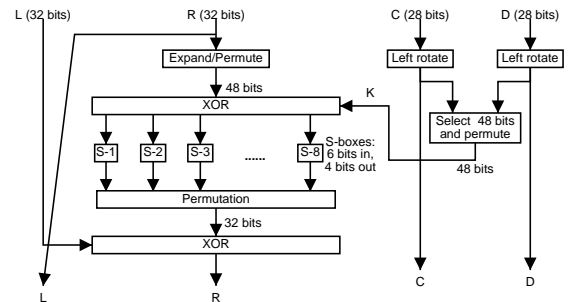
Each round involves shuffling data and substituting them.

Each step is reversible, so we can decrypt given the key.

All operations are simple, to ease hardware implementations.

POS(0234B)-9.32

Each round...



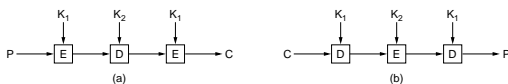
L is xor'ed with a number that depends on (R, C, D) in a complicated way. R is passed unchanged, so decryption can redo the "complicated" computation.

DES has no major known weakness, except the **short key length**.

POS(0234B)-9.33

Triple DES

- In 1979, IBM finds that the 56-bit key length of DES is **too small**. With a large cluster, a brute-force attack trying all keys finish in weeks or days.
- Logical solution is to double its key length. But on the other hand, IBM wants to make sure old hardware is still good.
- Answer: **3-DES**—just apply the DES 3 times, using 2 different keys. Encrypting and decrypting works like this:



- Why not **2-DES**? Given both plaintext and ciphertext, one (with enough resources) can perform a **"meet-at-the-middle"** attack. Perform (plain) DES encryption on the plaintext with all possible keys, and store all results in a hash table. Now DES-decrypt the ciphertext with all possible keys trying to find a match in the hash table.

POS(0234B)-9.34

Public key example: RSA

- Let's start from the well known **Fermat Theorem**: Denote p as a prime, and let $a < p$ be another positive integer. Then $a^{p-1} \equiv 1 \pmod{p}$. Why? If you multiply a numbers t between 1 to $p - 1$ by a and take $\text{mod}(p)$, you get distinct numbers t' . Multiply all such $at \equiv t' \pmod{p}$ formulae you get $a^{p-1}(p-1)! \equiv (p-1)! \pmod{p}$, which implies the needed equivalence.
- If we have a different prime q , we also have $a^{q-1} \equiv 1 \pmod{q}$. This implies $a^{(p-1)(q-1)} \equiv 1 \pmod{pq}$, and thus $a^{k(p-1)(q-1)+1} \equiv a \pmod{pq}$. Since $a^{(p-1)(q-1)}$ is 1 mod p and 1 mod q ; and the only such number $< pq$ is 1.
- Interestingly, the latter formula $a^{k(p-1)(q-1)+1} \equiv a \pmod{pq}$ can be **extended** to cases when a is arbitrary integer, not only for cases when it is not multiple of p or q .
- So if 2 nos e and d multiplies to $k(p-1)(q-1) + 1$ for some k , **raising a to the e -th power and then d -th power get back $a \pmod{pq}$** . It sounds like encryption and decryption, right?

POS(0234B)-9.35

How to find e and d?

- Any number **relatively prime** to $(p-1)(q-1)$ can be used as e . To make it easier, usually one would just use a prime of moderate size.
Then unless you are unlucky enough that e divides $p-1$ or $q-1$, you are fine. Usually a fixed prime (e.g., 65537) is used as e .
- Using **Euclidean algorithm** to find gcd of e and $(p-1)(q-1)$, you get 1 at the end. From it you can derive two integers r and s for which $re + s(p-1)(q-1) = 1$. Then r can be used as d .
- The private key is then (pq, d) , the public key is (pq, e) . Any message to be encrypted should be less than pq .
- The only known efficient way to recover d from (pq, e) is to **factorize** pq . With pq large enough (say 600 bits) this **takes a long time**.
- How to **find large primes**? Luckily there is fast algorithm to **test** whether a number is prime, and primes are **reasonably dense**.
Start from a random huge odd no. and add 2 repeatedly until we get a prime.

POS(0234B)9.36

Example

- We use very small p, q and e . In practice huge numbers are used.
- If $p = 5, q = 11$, then $pq = 55, (p-1)(q-1) = 40$.
- So under modulo 55, $a^{41}, a^{81}, a^{121}, a^{161}$, etc., are all a ($0 \leq a < 55$).
Let's try a small one: $9^{41} = 9(9^2)^{20} = 9(26^{20}) = 9(26^2)^{10} = 9(16^{10}) = 9(16^2)^5 = 9(36^5) = 9 \times 36(36^4) = 49(36^2)^2 = 49(31)^2 = 49 \times 26 = 9$. Note that by repeated squaring, computing exponents of huge numbers are not prohibitively slow.
- Suppose we choose $e = 7$ (relatively prime to 40).
- When we compute $\text{gcd}(7, 40)$, we find $40 \div 7$, get (5,5) as (divisor, remainder). Then $7 \div 5 = (1, 2), 5 \div 2 = (2, 1)$. $\text{GCD}=1$.
- Now $1 = 5 - 2 \times 2$. Since $2 = 7 \times 1 - 5, 1 = 5 - (7 \times 1 - 5) \times 2 = 5 \times 3 - 7 \times 2$. Since $5 = 40 - 7 \times 5, 1 = (40 - 7 \times 5) \times 3 - 7 \times 2 = 40 \times 3 - 7 \times 17$. So $d = -17 = 23 \pmod{40}$.
Note that $ed = 7 \times 23 = 161$, so $a^{ed} \equiv a \pmod{55}$.

POS(0234B)9.37

Where is Cryptography: the concerns

- Encryption and decryption can happen at **nearly all layer**.
- Why all layers? Why not choose one and do it once and for all?
 - Security of **lower layer cannot protect the upper layer**. E.g., if encryption is performed on network layer, then packets will never end up in the hand of a wrong computer. But it can end up in the wrong hand of a user in the same computer.
 - Security of **upper layer cannot protect the control packets of the lower layer**. E.g., even if you encrypt all your mails, an intruder can still send control segment in the (unprotected) transport-level saying "please terminate this connection", causing trouble on the end user.
- So security needs to be implemented **in all layers** that are considered to be vulnerable to attacks.

POS(0234B)9.38

Cryptography on top of data link

- Frames are encrypted **between 2 endpoints** of a link.
- Each link layer needs a different implementation, so it's **not generic**. Used only when physical media is **easily eavesdropped**, e.g., wireless.
- 802.11b allows the use of "**WEP**", Wired Equivalent Privacy, to encrypt all frames. Unluckily the encryption is **very weak**.
So weak that an arbitrary person listening to network traffic between two ends of a busy link can recover a 128-bit key within 1 day.
- The problem of WEP: employ a **weak RC4 algorithm and misuse** it.
RC4: given a key and a chosen initial value (IV) for the packet, an infinite sequence of pseudo random number is created to implement OTP. Misuse: same key used for all users, IV reused too frequently, no strong check for modified message, etc. But RC4 itself is weak and leak bits, so one can't expect too much anyway.
- A future revision of WEP promise to make it better.

POS(0234B)9.39

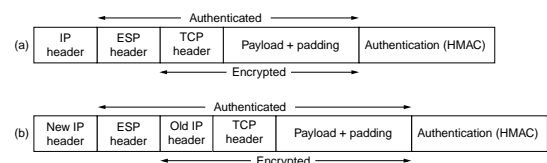
Cryptography on top of Network layer

- A more generic solution is to **encrypt IP packet**.
- IPv6 mandates the allowance of encapsulated security payload (ESP). IPv4 doesn't have provision for it, but there is a network-level protocol (**IPSec**) which does the same, the question is how many people use it.
- IPSec on IPv4 works in 2 modes:
 - In **transport mode**, an IP packet have its **payload** encrypted, and the protocol number is changed to ESP (50) to notify that it is encrypted.
Original protocol number is stored in the ESP header.
 - In **tunnel mode**, an IP packets between the end-points (including the header) is **entirely encrypted**.
- This is usually used to build Virtual Private Network (**VPN**), i.e., connect two private, trusted networks with untrusted Internet while keeping message unintelligible by any eavesdropper outside.

POS(0234B)9.40

IPSec header

The following should clarify the difference between (a) transport mode and (b) tunnel mode.



Here you see a special term **HMAC** (Hashed Message Authentication Code). It is a cryptographic hash of the payload, like Message Digest.

The secret session key is **appended** to the message before the hash is computed, which make it impossible for intruders to forge a message.

POS(0234B)9.41

Cryptography on top of Transport layer

To actually know who actually sends and can read a message (rather than which host sends and can read it), one has to do encryption on the **connection**, encrypting **segments**.

- Some (user-mode) software **library routines** performs encryption before putting segments into TCP connections.
- The de-facto standard is SSL: **Secure Socket Layer**.
- **Each** program must be written and compiled against it.
Quite some programs are modified to support this, e.g., servers and clients of telnet, ftp, web, icq, mail, database, etc.
- Support authentication and encryption, using algorithms chosen from a configurable set.

POS(0234B)-9.42

Cryptography on top of Application Layer

- Implementing encryption in transport layer requires both the server and the client to be rewritten. What if that is not possible?
- A classic example is **E-mail**: the E-mail system is long established when very few people care about security.
- It is impractical to change the E-mail backbone. But there is still a place to implement security: **on top application level protocol**.
- The idea: send mails using the usual unencrypted channel, but the **E-mail contents transferred are encrypted**.
- The most popular choice is to use a **MIME type** to indicate that the message is encrypted or signed. Then the message is ASCII-armored and sent just like any attachment.

POS(0234B)-9.43