

Lecture 8

Network security

In this last chapter, we look at something that is needed (although not usually implemented) in multiple network layers: security.

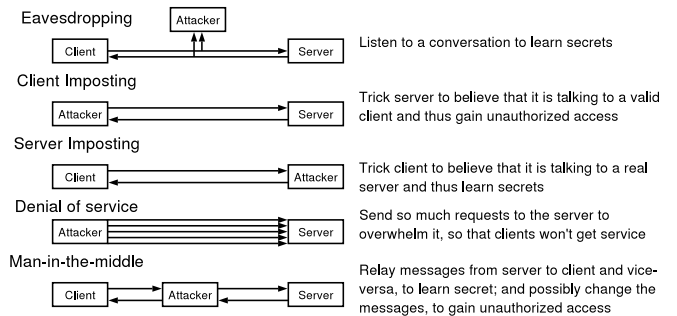
References:

- Textbook: Chapter 11.
- RFCs: 1321 (MD5), 2104 (HMAC).

Network(0234B)

Primary security threats scenarios

Normally, clients and servers somehow believe that they are talking to each other. An attacker can play many different roles...



Network(0234B)-8.1

What we might want

We want our communication to possess some or all these properties:

- **Confidentiality** (privacy): only the intended recipient will understand the message.
- **Integrity:** the intended recipient should be able to confirm the message has not been altered in transit.
- **Authentication:** For the sender, receiver or both to confirm that they are talking to the intended peer rather than an attacker.
- **Non-repudiation:** For the receiver to be able convince others that it is really the sender who made the message when the need comes.

Sometimes we want the reverse: **deniability**. It makes sure the receiver cannot convince others in this way. E.g., avoid law enforcing. In other words, to be able to say "he makes it all up" in front of the court—even though he **did** send the message. This is atypical in network communications.

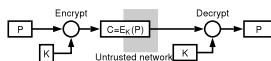
Network(0234B)-8.2

Role played by Firewalls

- A firewall **limits where the attacker can start their attack**.
- E.g., if you drop all packets heading port 23 to your organization in your router, an attacker cannot attack your `telnet` service from outside.
- But this is usually only part of the story...
 - Most organization needs to provide some means for **its own staff** to access its network from outside, so one cannot block everything.
 - At times we need protection against staff of the same organization.
 - **Malware** (like viruses and worms) running inside the company allows attackers to run programs **from within** the organizations.
 - If all computers in your organization are running malware, nothing can stop your information to get lost. But if just a few computers have the problem you don't want the problem to spread to every computer in the network.

Network(0234B)-8.3

Confidentiality with Symmetric Key encryption



P =plaintext, K =shared key, C =ciphertext, E_K =encryption function.

- E_K should have the property that given C but without K , it is extremely difficult to recover P .
- We also want that given **many** different C 's and their corresponding P 's, it is still extremely difficult to recover K .
- Common ones: DES ("Data Encryption Standard"), triple-DES, IDEA (Intern'l Data Encryption Alg) and AES ("Advanced Encryption Standard").
- They all perform **substitutions** and **permutations** hard enough to make it difficult to break, but not so hard that it becomes time consuming to encrypt and decrypt.
 - Substitutions replace a bit group with another, permutations shuffle bits around.

Network(0234B)-8.4

Why it is secure?

Why we believe that it is difficult to attack, say, DES? E.g., why one cannot recover a message without the key?

- Let's say we use a simple **substitution cipher** to encrypt an English sentence: the key is a **permutation** of the 256 bytes of ASCII.

A message is encrypted by substituting all bytes x in the message with the x -th entry in the permutation. To decrypt, we substitute with the inverse permutation.

- But this can be **broken easily**: build the distribution of each byte (and each adjacent pair of bytes) in the ciphertext, and match it against that of standard English. The message is most likely recovered.

Why we think that a more sophisticated scheme cannot be broken by a more sophisticated attack?

Network(0234B)-8.5

Kerckhoff's principle

There is nearly no **provably** unbreakable scheme. Instead, somebody design a scheme and ask all really bright mathematicians to break it.

- "Basic principle of computer security": the scheme is "unbreakable" because bright people tried and can't break it.

Don't trust a scheme just because the ciphertext looks random!

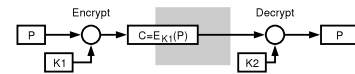
Once a scheme is trusted, one can make new schemes by proving that it is "at least as good", i.e., if you can break a new scheme, another old scheme is also broken. Then anyone who trust the old scheme should also trust the new one.

- Consequence: **Kerckhoff's principle**—schemes must work even when **encryption and decryption algorithm are not kept secret**.
- So the algorithms use **keys**, which are secret. The algorithm must be secure using any key unknown to others.

Network(0234B)-8.6

Confidentiality with Asymmetric Key encryption

Symmetric key schemes have a difficulty: **how to exchange the secret** without exposing them? This is avoided by asymmetric key encryption.



- Guarantee of asymmetric key: given one of $K1$ and $K2$, it is extremely difficult to find the other.
- So it is okay to **let everybody know $K1$ (the public key)**. We don't need a confidential channel to exchange the key. But we still need to ensure authentication. More about it later.
- So **everyone can encrypt**, but only the one with $K2$ (the **secret key**) can decrypt the resulting message.
- The most well known asymmetric key encryption algorithms is RSA, which is based on difficulty of factoring large numbers.

Network(0234B)-8.7

Public key example: RSA

- Let's start from the well known **Fermat Theorem**: Let p be 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 $a \pmod{pq}$** .
 a is plaintext, raising to e -th (d -th) power \pmod{pq} is encryption (decryption).

Network(0234B)-8.8

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.

Network(0234B)-8.9

Example

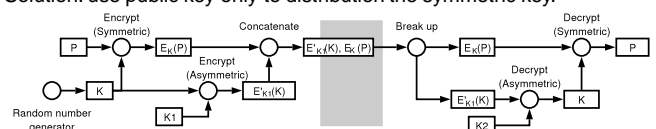
- We use very small p, q and e . In practice huge numbers are used.
- If $p = 5, q = 11$, then $pq = 55, (p - 1)(q - 1) = 40$.
- So under modulo 55, $a^1, a^4, a^{16}, a^{64}, \dots$, are all a ($0 \leq a < 55$).
Let's try a small one: $9^{41} = 9(9^2)^{20} = 9(26^{20}) = 9(26^2)^{10} = 9(16^{10}) = 9(16^2)^5 = 9(36^5) = 9 \times 36(36^4) = 49(36^3)^2 = 49(31)^2 = 49 \times 26 = 9$. 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 $a^{ed} \equiv a \pmod{55}$.

Network(0234B)-8.10

Combining the two schemes

Asymmetric key algorithms have two deficiencies: it's **slow** (compared to symmetric algorithms), and it requires **longer keys** (make it even slower). RSA needs at least 1024 bits, while triple-DES do with just 112.

Solution: use public key only to distribution the symmetric key.

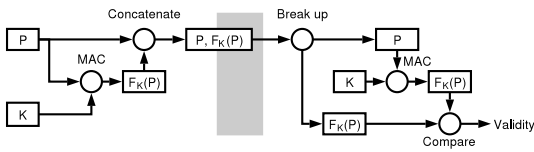


- A random **session key** is used for symmetric encryption. The session key itself is encrypted using the public key of the receiver.
- The receiver recover the session key using its private key, and then the plaintext with the recovered session key.

Network(0234B)-8.11

Authentication with symmetric keys

Sometimes we only need authentication, not encryption. In this case a **message authentication code (MAC)** may be used.



MAC does not need reversibility, so it's easier to make stronger schemes. An example is "Data Authentication Algorithm", based on DES.

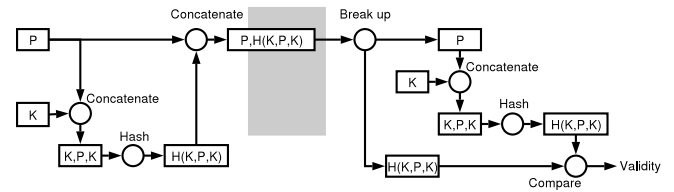
MAC (and cryptographic hashes) makes **fixed-length** hashes from variable size messages. They require (or "assume") that it is hard to make up another message with the same hash.

So they must not be too short, otherwise brute-force search breaks them.

Network(0234B)-8.12

HMAC

We can simplify the algorithm by removing the dependency of the key. So we use a cryptographic hash algorithm instead, resulting in HMAC (Hashed MAC).



Typical cryptographic hashes like **MD5** (Message Digest 5) and **SHA-1** (Secure Hash Algorithm) are more efficient to compute than MAC, so it improves performance.

The actual algorithm defined in RFC 2104 is a bit more complicated to prevent various kinds of attacks.

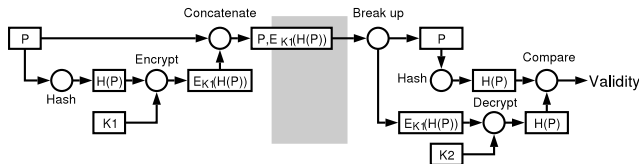
Network(0234B)-8.13

Non-repudiation with asymmetric keys

Unlike symmetric key algorithms, asymmetric key algorithms provide **non-repudiation** very easily: the sender just makes a secret encryption key K_1 and publishes the decryption key K_2 .

Nobody can make up the message as they don't know K_1 .

We usually call this **digital signature**. When actually used, in general a hash is signed instead of the whole message—to improve efficiency.



Network(0234B)-8.14

Key distribution for public keys

Public keys **do not need confidentiality**, so easier to handle. E.g., it can be put into web pages or as DNS resources.

How we know a public key is "real" (authenticated)? Doing it **manually**, e.g., with a phone call, is too tedious.

- We can trust somebody else (called a Certificate Authority, CA) to help us verify the identities. E.g., a government can act as a CA.
- Then the CA can **sign** a document that says something like "Isaac To, with HKID ZZZ, is checked to own the key pair with public key XXX". Such signed documents are called **certificates**.
- But for the government to be able to sign the document and for the users to be able to check the document, it needs a key—and we again have to ensure that its key is not faked!
- We can hard-code them, but CAs and their keys are changing.

Network(0234B)-8.15

Public Key Infrastructure (PKI)

The answer: a **tree** of agencies to make certificates.

- There are a few agencies called the **root certificate authorities (root CAs)**, which everybody knows about. Their public keys are hard-coded into programs.
- It will sign certificates to **delegate** its power of making certificates, i.e., making certificates saying that "The XYZ authority is certified as a CA". The CAs can then delegate its power further.
- Each CA can certify the keys of a user, giving her a certificate **and all certificates** from root CA to the certifying CA.
- When the user wants to give her key to somebody else, she would also give all the certificates from the root CA as well.
- Certificates are put into a format (e.g., X.509), to enable automatic processing.

Network(0234B)-8.16

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.

Network(0234B)-8.17

Cryptography in data link

- When a link is vulnerable to attacks (like Wireless), the standards may allow frames to be encrypted **between endpoints** of a **link**.
- 802.11b defines "**WEP**", Wired Equivalent Privacy, for encrypting frames. The encryption: XOR the message with a pseudo-random sequence generated from a key shared by the access point (AP) and the user.
- Unluckily, the pseudo-random sequence used (RC4) is **quite weak** (i.e., can be partially guessed) and used **repeatedly**.
Somebody listening to a busy link can recover a 128-bit key within a day. It doesn't follow the advice "never use your own scheme", causing much trouble.
- Even worse, most APs use **one key for all users**. I.e., each user can decrypt data of all other users.
- WEP is now an obfuscation scheme rather than a security scheme. A future revision of WEP promises to make things better.
Before that happen, use IPSec.

Network(0234B)-8.18

Cryptography on top of Network layer

- IPSec defines **Encapsulated security payload** (ESP) in the IP layer for confidentiality and authentication.
It also define "Authentication header" (AH) for authentication, but it is duplicated functionality and is replaced by ESP.
- ESP is designed as an **IPv6 header and trailer**: previous header says "next header" is ESP (50); and ESP header contains a "next header" field to continue the chain. The ESP itself contains some parameters for the encryption. The trailer contains a HMAC for authentication check.
- IPSec is mandated in IPv6. It is back-ported to IPv4; if used, its layout is similar to ESP in IPv6.
- The whole IP packet can be used as a ESP payload (using IP tunneling, aka IPIP). This is often 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.

Network(0234B)-8.19

Cryptography above Transport layer

TCP does not have provision for encryption, but the user can add encryption on top of it. This can be done in two ways:

- With a **library** on top of the socket calls. Whenever you send messages into the socket, the library encrypts it. Whenever you read message out of the socket, the library decrypts it.

The de-facto standard is SSL: **Secure Socket Layer**. The benefit is that the program nearly does not need to be changed: it just has to be compiled against the library.

Quite some programs are modified to support this, e.g., servers and clients of telnet, ftp, web, icq, mail, database, etc.

- Within the **application code**. This is needed whenever the end users do not have a direct connection (e.g., TCP) between each other, E.g., E-mail applications. Various schemes are added to E-mail to add security to it, like PGP (Pretty Good Privacy) and S/MIME (Secure MIME).

Network(0234B)-8.20