ELEC5471M

Data Communications and Network Security

Lectures on Network Security
Handout 1: Encryption Techniques

Mohsen Razavi

Institute of Integrated Information Systems
School of Electronic and Electrical Engineering
University of Leeds

Network Security

In 4 lectures and several lab sessions we will cover

- The basics of cryptography
- Symmetric and public encryption/decryption techniques
- Authentication problem
- Email security
- Security at the transport layer (SSL)

The assessment will be based on two projects to be run in weeks 8-11

- A MATLAB project on public key cryptography
- A Wireshark project on SSL
- Altogether, they constitute 30% of your total mark.

What would network security entail?

Consider the email example. As a user, what would you expect from this service?

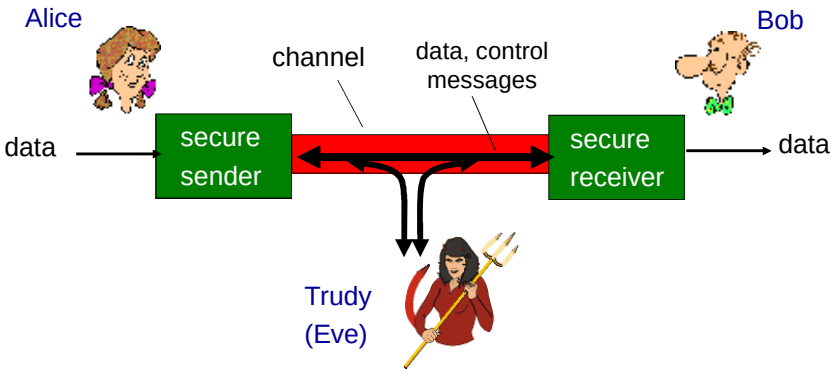
- We expect the message arrives safely with no changes in the content → message integrity
- We may like that nobody, but the recipient, learns about the content → confidentiality
- The sender needs to make sure that the intended recipient picks up the email; similarly, the recipient needs to make sure that the message is genuinely sent by the declared sender → authentication
- We also need protection from
  - unwanted spam emails
  - Phishing and/or harmful emails
  - Attacks that may make you unable to use your email service

All these, and more, would lie under the general subject of network security

- Here, we only get to address some of these issues.

Friends and enemies: Alice, Bob & Trudy (Eve)

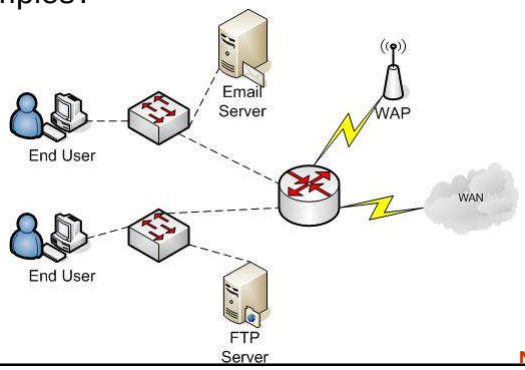
- Well-known in network security world
- Bob, Alice (lovers!) want to communicate "securely"
- Trudy (intruder) may intercept, delete or add messages; impersonate; hijack a session; cause denial of service



* Some pictures/slides are taken from Kurose & Ross, Computer Networking: A Top Down Approach, 6th edition

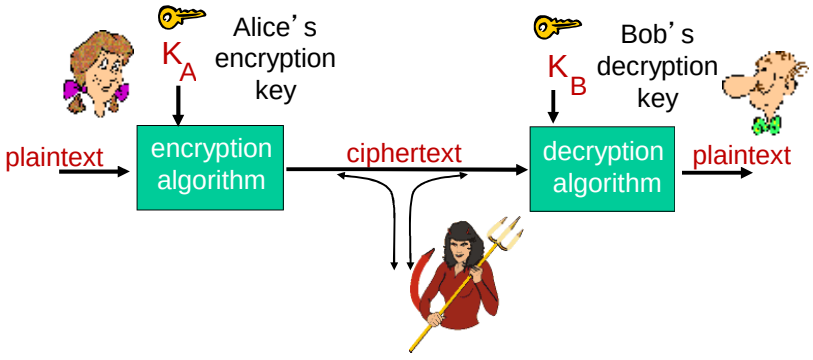
Who might Alice and Bob be?

- … well, *real-life* Bobs and Alices!
- Web browser/server for electronic transactions (e.g., on-line purchases)
- on-line banking client/server
- DNS servers
- routers exchanging routing table updates
- other examples?



Language of Cryptography

- The common nomenclature:



m plaintext message

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

$m = K_B(K_A(m))$

Encryption

- What is the best way of encrypting a message?
- Let's first see a simple, but imperfect, method:

substitution cipher: substituting one thing for another

- monoalphabetic cipher: substitute one letter for another

```
plaintext:  abcdefghijklmnopqrstuvwxyz
            |                       |
ciphertext: mnbvcxzasdfghjklpoiuytrewq
```

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

Encryption key: mapping from set of 26 letters to set of 26 letters

But, how strong is this scheme?

Breaking an encryption scheme

- cipher-text only attack: Trudy has ciphertext she can analyze

  two approaches:
  - brute force: search through all keys
  - statistical analysis

- known-plaintext attack: Trudy has plaintext corresponding to ciphertext
  - e.g., in monoalphabetic cipher, Trudy determines pairings for a,l,i,c,e,b,o,

- chosen-plaintext attack: Trudy can get ciphertext for chosen plaintext

Encryption

- So, now, what is the best way of encrypting a message?
- Suppose your message m is a sequence of bits to be securely sent
- Your objective is to map m to the ciphertext $K_A(m)$ such that the chance of Eve's correctly guessing each bit in the original message is the worst possible.
- Knowing $K_A(m)$, what would be the optimum probability (from Alice-Bob viewpoint) for Eve's correctly guessing each bit in m?

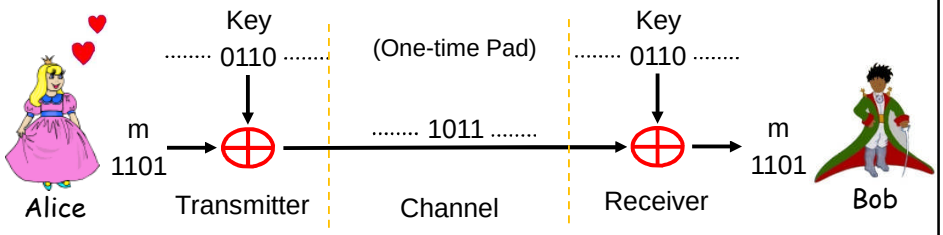
One-time Pad

- Your objective is to map m to the ciphertext $K_A(m)$ such that the chance of Eve's correctly guessing each bit in m is 1/2.
- One-time pad: Suppose Alice and Bob share a truly random sequence of bits, K_S, that nobody else knows about. Then, they can follow the following encryption/decryption scheme:

Encryption
$K_A(m) = m \oplus K_S$

$\oplus$: bit-wise XOR

Decryption
$K_B(K_A(m)) = K_A(m) \oplus K_S$
$= m$

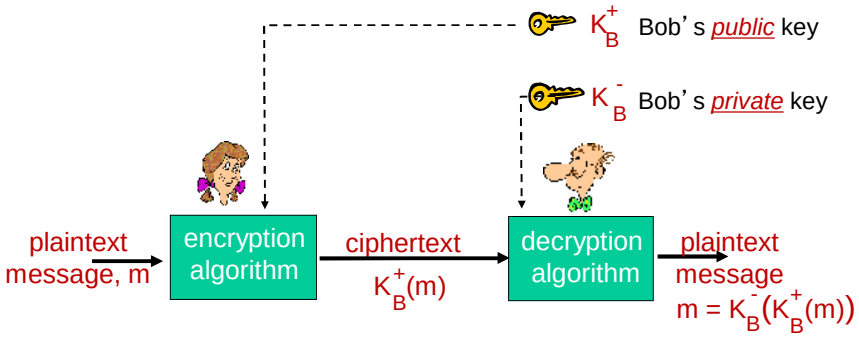


Symmetric Key Encryption

- One-time pad is an example of *symmetric* key encryption: Alice and Bob use the same key for encryption and decryption
- One-time pad is extremely secure (information theoretically provable, check it against different types of attacks), but has two implementation challenges:
  - 1- How Alice and Bob share a secret key to begin with? Do they have to come together every time? Remember that some of Alices and Bobs are simply machines, routers, …
  - 2- One-time pad needs a key as long as the message. At the price of losing some of the security, can we devise other protocols that are practically secure and more efficient in the usage of the precious key?
  - We will see next how we can address these two issues

Public Key Cryptography

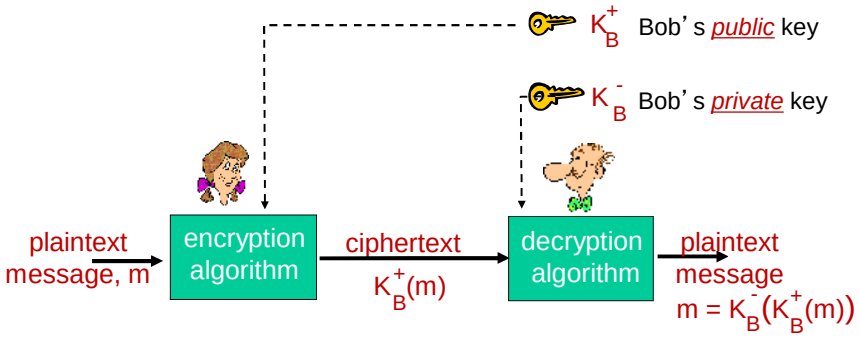
- Key idea: Encryption key is *different* from the decryption key. It is like a safe that can be locked with a simple key, but cannot be opened with the same key.



- Bob's encryption key, K_B^+, can then be made *public*, so that everyone can send secure messages to Bob.
- Bob is the only one with a private key, K_B^-, that can decrypt the message.
- So long as K_B^- cannot be derived from K_B^+, this scheme is secure.

Public Key Cryptography

- Key idea: Encryption key is different from the decryption key. It is like a safe that can be locked with a simple key, but cannot be opened with the same key.

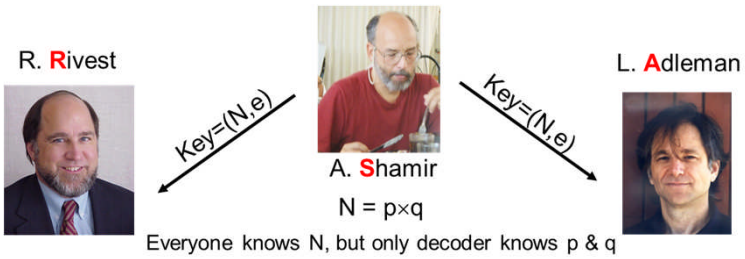


- Key advantage: Alice and Bob need not share a secret key like in symmetric crypto; In fact, they can use this scheme to share a secret key!

But, how can we devise such a scheme?

RSA Algorithm

- The first public-key scheme was proposed by Diffie & Hellman in 1976.
- The one, which got widespread use, however, was proposed two years later by Rivest, Shamir, and Adleman, and is known as RSA.



- The security of RSA relies on the computational complexity of the factoring problem. If N is a large number, it takes a very long time to find prime numbers p & q:

$$O((\log N)^{(\log N)^{1/3}})$$

Number of operations for best known factoring algorithm on a classical computer

Some maths revision: Modular Arithmetic

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

  $(a + b) \bmod n = [(a \bmod n) + (b \bmod n)] \bmod n$

  $(a - b) \bmod n = [(a \bmod n) - (b \bmod n)] \bmod n$

  $(a \times b) \bmod n = [(a \bmod n) \times (b \bmod n)] \bmod n$

- Thus

  $a^d \bmod n = (a \bmod n)^d \bmod n$

- Example: x=14, n=10, d=2:

  $(x \bmod n)^d \bmod n = 4^2 \bmod 10 = 6$

  $x^d = 14^2 = 196 \quad x^d \bmod 10 = 6$

RSA in more detail

- Message: just a bit pattern
- Bit pattern can be uniquely represented by an integer number
- Thus, encrypting a message is equivalent to encrypting a number.

example:

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

RSA: Public-Private key creation

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 called "relatively prime").
4. Choose d such that $ed-1$ is exactly divisible by z. (in other words: $ed \bmod z = 1$).
5. *Public* key is $\underbrace{(N,e)}_{K_B^+}$. *Private* key is $\underbrace{(N,d)}_{K_B^-}$.

RSA: Encryption/Decryption

0. given (N,e) and (N,d) as computed above
1. to encrypt message m ($<N$), compute
   $c = m^e \bmod N$
2. to decrypt received bit pattern, c, compute
   $m = c^d \bmod N$

magic happens! $m = (\underbrace{m^e \bmod N}_{c})^d \bmod N$

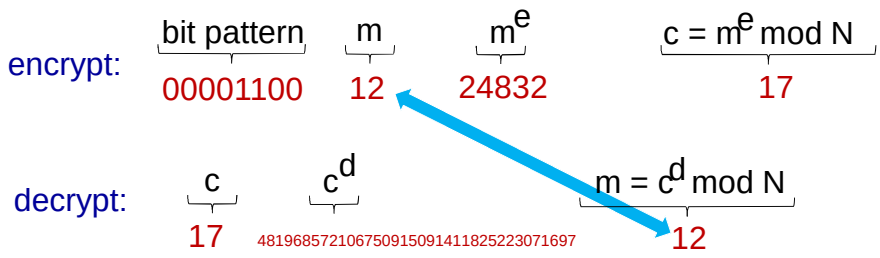
RSA Example

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

e=5 (so *e & z* are relatively prime).

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

encrypting 8-bit messages.



Why does RSA work?

- Must show that $c^d \bmod N = m$ where $c = m^e \bmod N$
- Fact: for any x and y: $x^y \bmod N = x^{(y \bmod z)} \bmod N$
  - where N= pq and z = (p-1)(q-1)
- Thus,

$$c^d \bmod N = (m^e \bmod N)^d \bmod N$$
$$= m^{ed} \bmod N$$
$$= m^{(ed \bmod z)} \bmod N$$
$$= m^1 \bmod N$$
$$= m$$

RSA: Another important property

The following property will be *very* useful later:

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

result is the same!

Follows directly from modular arithmetic:

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

Encryption in practice

- Flashback: Challenges of one-time pad:
  - 1- How Alice and Bob share a secret key to begin with? Do they have to come together every time? Remember that some of Alices and Bobs are simply machines, routers, …
  - 2- One-time pad needs a key as long as the message. At the price of losing some of the security, can we devise other protocols that are practically secure and more efficient in the usage of the precious key?
- Public-key can solve the first problem, but still the exponentiation needed in RSA is computationally intensive.
- It would be more practical if we find efficient ways for encrypting data even if they need a symmetric key.
- Idea: We can generate the required session keys using RSA, and then use relevant symmetric schemes for encryption/decryption.

Block Ciphers

- Block ciphers
  - Break plaintext message in equal-size blocks
  - Encrypt each block as a unit
- Example with 3-bit-long blocks:

input	output	input	output	input	output	input	output
000	110	010	101	100	011	110	000
001	111	011	100	101	010	111	001

- Imagine for a long block how many mappings we have!
- Problem: The same plaintext is mapped to the same ciphertext
- Solution: Cipher-Block Chaining (CBC)
  - Add a random block r(i) to the message block m(i); send $K_A(m(i) \oplus r(i))$ and r(i) to Bob
  - Smarter soln: Initialization Vector (IV). Alice sends r(0) to Bob. The first block of data is added to r(0). For all other blocks, r(i) is the same as the encrypted message in the previous block.

Data/Advanced Encryption Standard (DES/AES)

DES operation

Initial permutation

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

Followed by a final permutation

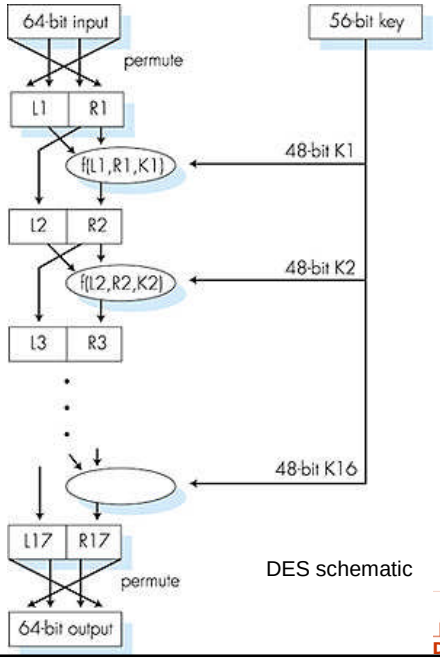
AES operation

Replaced DES in 2001.

Processes data in block of128 bits.

128, 192, or 256 bit keys

Brute force decryption: if taking 1 sec on DES, takes 149 trillion years for AES



DES schematic