Implementing Cryptography #1

Ed Zaluska

COMP6230:
Implementing Cybersecurity

Acknowledgements

- partly based on Chapter 2, Pfleeger and Pfleeger, ‘Security in Computing’, pages 37 to 97
- some slides adapted from a slideset created by V. Tan

Key issues

- Cryptography
  - 'the art and science of encryption'
  - (literally 'secret writing')
- Context and terminology
- Symmetric and Asymmetric encryption
- Cryptographic examples
- Typical cryptographic applications

What can we expect from Cryptography?

- **"cryptography is not the solution"**
- **"cryptography is very difficult"**
- **"cryptography is the easy part"**

(all above quotes from Ferguson, N., Schneier, B. and Kohno, T. (2010)

Cryptography Engineering: Design Principles and Practical Applications. Wiley)

“A security system is only as strong as its weakest link"

- often find the digital equivalent of a bank vault fitted to a tent! *(see images)*
- designers need professional paranoia - need to design against the bad guys

  (compared with civil engineer designing a bridge – only has to design against nature)
- “security is a *process*, not a product”

  (Bruce Schneier)

Primary security issues

- **Confidentiality** – messages remain private
- **Integrity** – messages are not modified
- **Authentication** – confidence in the identity of parties involved
- **Non-repudiation** – parties cannot deny having generated data/message

  (often at a particular time)

  (evidence acceptable to a court of law)

Security Threats

- Eavesdropping
- Wiretapping
- Key interception
- Impersonation
- Data duplication
- Cryptanalysis
- Physical security
- Social engineering

(non-exhaustive list!)

Terminology

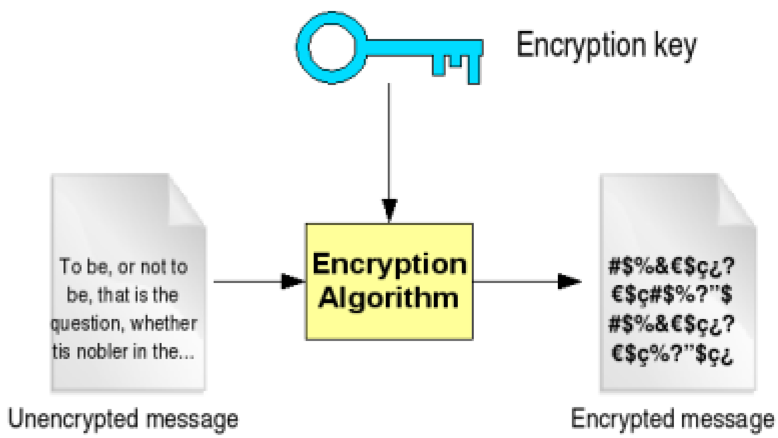
(Although cryptography is **not** the complete solution to security issues, it is nevertheless an essential component)

- plaintext - P or P/T
- ciphertext - C or C/T
- key – K
- encryption (or *enciphering*) P $\longrightarrow$ C

  C = E (K, P)
- decryption (or *deciphering*) C $\longrightarrow$ P
- P = D (K, C) hence P* = D(K, E(K, P))

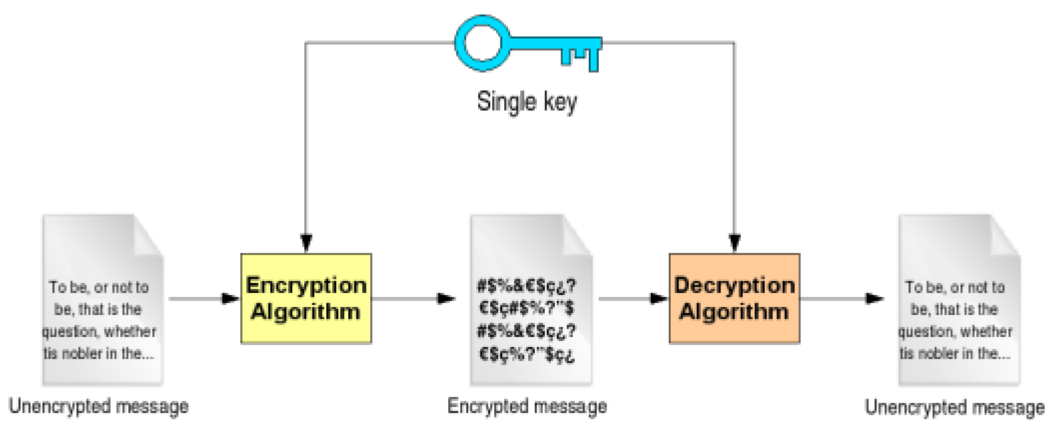
Encryption

- addresses confidentiality/integrity
- uses an encryption algorithm and an encryption key
- for a given plain text message, encrypted version will differ for different key sequences



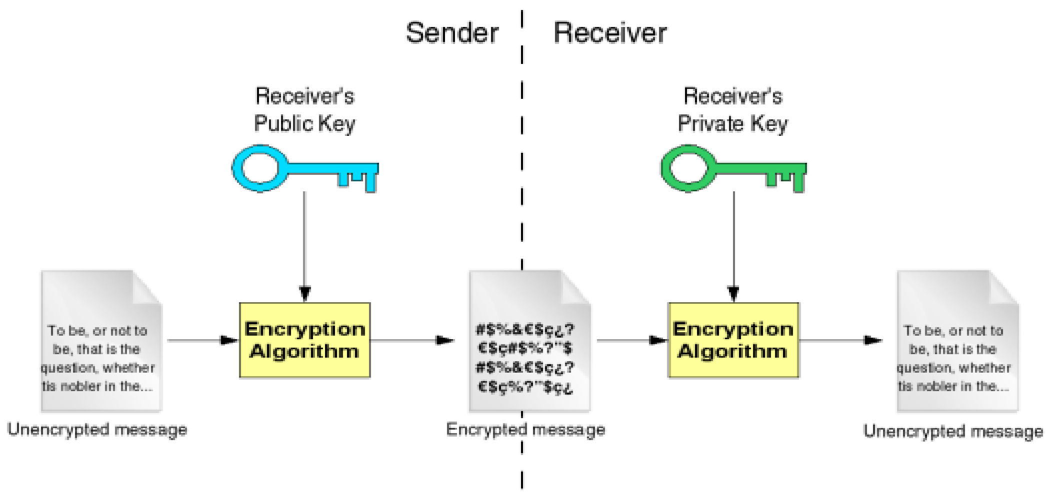
Symmetric encryption

- (also known as 'secret key')
- same key used for encrypting and decrypting
- straightforward implementation, fast operation
- one key, known to sender and recipient
- examples: DES, 3DES, AES



Asymmetric encryption

- (also known as ‘Public Key’ (hence ‘PKI’))
- two types of keys: **private** and **public**
- private key kept secret by owner, public key is *distributed* to whoever needs it (i.e. not secret)
- encryption by public key, decryption by private key (or *vice versa* … explained later)
- slower algorithm compared with symmetric encryption: example – RSA (Rivest/Shamir/Adleman)



Key Distribution Problem

- clearly a difficulty with secret key, but even with public key how do you know that the public key distributed is authentic?
- how do you know who you can trust?
- ‘out of band’ transmission (e.g. courier, postal services)
- often a symmetric *session key* is used –

  - randomly generated (not pseudo-random!)

  - used only for a ‘short’ time

  - typically sent using a public key system

  (relatively slow speed unimportant as small amount of data)

Historical ciphers

- earliest cipher attributed to Julius Caesar (hence ‘Caesar cipher’)
- example of a *monoalphabetic substitution cipher*

  P/T: A B C D etc.

  C/T: D E F G etc.
- example: P/T “BAD” $\longrightarrow$ C/T “EDG”
- highly susceptible to *cryptanalysis*

Simple ciphers

- Caesar cipher – classical Latin alphabet had 23 letters - hence just 22 different alphabets available to use (same letter ordering) (e.g. Caesar used a letter shift of 3)
- Slightly more complex is the *affine cipher*

  (also a *monoalphabetic substitution cipher*)
- $C_X = (aP_X + b)$ modulo m [n.b. key = (a,b)]
- “P_X” is a character

  e.g. (A..Z) – mapped to (0..25) where m = 26
- a, m must be co-prime: a=1 gives the Caesar cipher. Just 286 non-trivial affine ciphers…

Monoalphabetic substitution ciphers

- more generally, if the letters can be in any position, then there are 26! possible alphabets available for use (4 * 10**26)
- surely such a large number would make cryptanalysis very difficult?
- **not true**: many techniques are available to the cryptanalyst:

  e.g. frequency analysis of ciphertext symbols

  in English, E,T, O, A are more frequent than J, Q, X, Z

Codes and ciphers

- for a cipher, length (C) = length (P)
- codes require a *dictionary* or *look-up table* at both ends (hence “code book”)
- C = lookup_table (P)
- P* = lookup_table (C)
- (codebooks must be identical either end)
- potential for data compression if codes shorter than plaintext
- *Huffman coding* good example where codes used for compression, not security

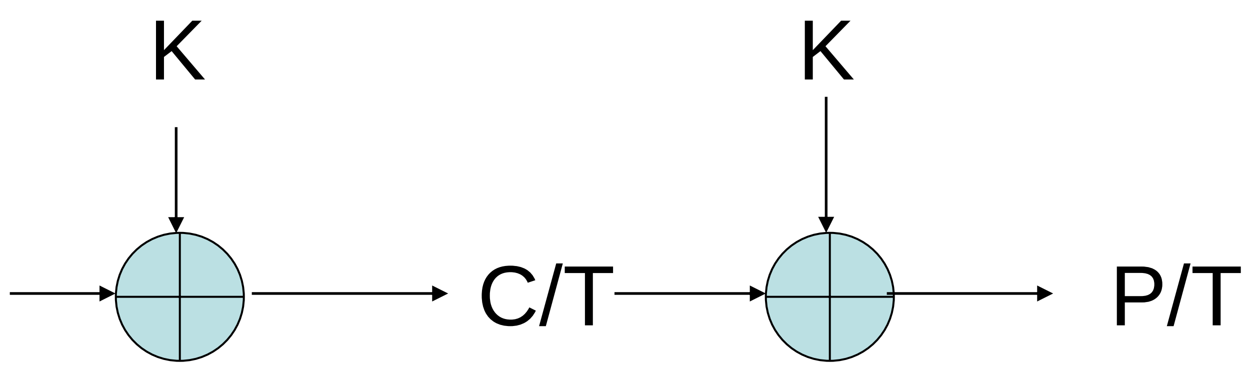
Codebreaking

- long history of codes being used for secret messages (*codebooks* being used either end)

  (hundreds of years)
- long history of third parties intercepting the messages and re-constructing the codebooks (“code breaking”) (or *stealing* the codebooks)

The only unbreakable cipher

- the ‘one-time pad’
- (first proposed in 1882, re-discovered by Vernam in 1917) (hence ‘Vernam cipher’)
- P/T



One-Time Pad

- ‘exclusive-OR’ function (XOR)

input1	input2	output
0	0	0
0	1	1
1	0	1
1	1	0

output = if (input1==0) input2 else not(input2)

  - hence A XOR B XOR B = A

One-Time Pad continued…

Requirements:

- (1) length (K) = length (P/T)
- (2) K **must** be random
  (**not** pseudo-random)
- (3) K must **never** be re-used

a message encrypted with a one-time pad could be *anything*

Random numbers

- all computers (programming languages) have a “random number generator” (RNG), or “pseudorandom number generator” (PRNG)
- also straightforward in hardware – easiest is a linear-feedback shift register (LFSR)
- example in python (need “from random import *”):
  - seed (1)
  - for i in range (0,6): print random()

Random numbers (2)

- a PRNG cannot be used for any cryptographic application, because it is *deterministic*
- always repeats, sequence always the same (can reverse-engineer to discover algorithm being used)
- require a truly-random process – e.g. thermal noise in electronic device, radioactive decay, roulette wheel etc.

Random numbers (3)

- first UK “lottery” in 1957 used thermal noise (“ERNIE”)
- more recently, machines for the UK national lottery select random numbered balls

  (examples of fraud elsewhere…)
- note possibility of a “cryptographically secure pseudo-random number generator” (CSPRNG) – starts with random input from a high-quality source and essentially expands it (can’t increase the original entropy i.e. disorder or uncertainty)
- must not be able to reverse-engineer…

One-Time Pad continued again…

- “length (K) = length (P/T)”

  this is a **major disadvantage** because of the **key distribution problem**
- well-known example from early 1940’s – the cryptanalysis of Soviet one-time pad encoded traffic, the “Venona” project
- ?? How come? German advance on Moscow resulted in pads being re-used
- (a serious breach of cryptographic procedure)

Stream and Block Ciphers

- stream cipher – converts one symbol of P/T at a time
- block cipher – converts a group of P/T symbols together (typically 64 bits, 128 bits or longer)

Confusion and Diffusion

(background to underlying principles)

- *confusion*: complex relationship between P/T and C/T
- *diffusion*: spread changes in P/T over C/T
- (typically stream cipher has low diffusion, block cipher high diffusion)