COMPSCI S314 S2T 2009
lectures 11-21

Low level protocols, routing

Prof. Brian Carpenter

- 10, 11 – Security mechanisms
- 12 -14 – Link control, flow control
- 15 -17 – Ethernet, wireless
- 18, 19 – Bridges, switches, VLANs
- 20, 21 – Routing
  - Dates: 11 August ... 18 September
  - *Term test on 14 August is on Cris Calude's lectures only*
  - Approximately covers Shay 7.1-7.5, 8, 9 (not 9.6), 10
  - Assignments 2 and 3
  - **Questions:** brian@cs.auckland.ac.nz or room 303s.587 (most days between 10 a.m. and 4 p.m.)

About the level of detail

- I sometimes give more technical details than the text books, because I want all students to feel that they can see how the various protocols could be coded in a programming language.
- The level of detail given in Shay should be sufficient for the exam questions.

Lectures 10, 11:

Security mechanisms

- Introduction (Shay 7.1)
- Encryption (Shay 7.2-7.4)
- Authentication (Shay 7.4-7.5)

Security 101

Properties of secure data: **CIA**

- Confidentiality: no unauthorised user can read
- Integrity: no unauthorised user can write
- Availability: all authorised users can read and write

Confidentiality - provided by *encryption*

Integrity - provided by *authentication* and *cryptographic signature*

Availability - means preventing *denial of service attacks*

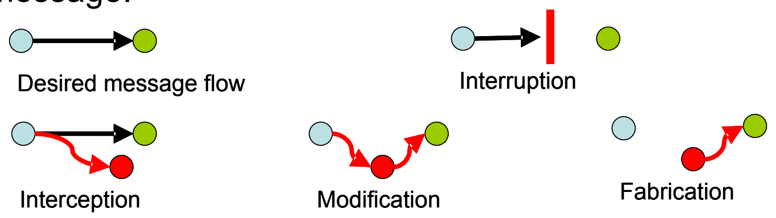
For now we'll consider techniques for encryption and authentication.

Security functions

- The Gold Standard, and some additional functions:
- Authentication: are you who you say you are?
  - All claims to identity can be verified.
- Authorisation: who is permitted to do which operations to what?
  - Users can't increase their own authority.
- Auditing: what has happened on this system?
  - System administrators can investigate problems.
- Identification: what human (or object) is this?
  - Different from authentication (a proof of an identity) or authorisation (a decision to allow an activity).
- Non-repudiation: can you prove this event really did happen?
- ➔ To learn more: Lampson, "Computer Security in the Real World", *IEEE Computer 37:6*, June 2004.

Types of attack

- *Modification* or *man in the middle*: an attacker changes a message;
- *Interruption* or *denial of service*: an attacker prevents delivery, often by floods of rubbish packets;
- *Fabrication* or *spoofing*: an attacker injects a message;
- *Interception* or *eavesdropping*: an attacker reads a message.



Encryption is more than coding

- Coding schemes are designed to be decoded by an algorithm.
- Encryption schemes also need special knowledge to decode them.
- **`Xibu jt uijt tjnqmf fodszqujpo?`**

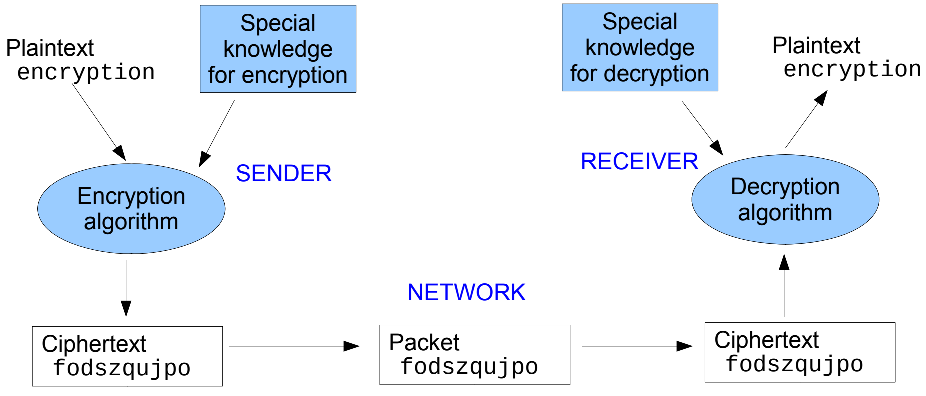
cyphertext

Encryption is more than coding

- Coding schemes are designed to be decoded by an algorithm.
- Encryption schemes also need special knowledge to decode them.
- **`Xibu jt uijt tjnqmf fodszqujpo?`**
- **`What is this simple encryption?`**
  - The algorithm is "go back N letters"
  - The special knowledge is "N=1"

plaintext
or cleartext

Encryption and decryption



Without the special knowledge (key), an intruder on the network cannot understand the packet, and cannot change it or insert a new one without being detected.

Terminology

- Call the plaintext (the message) P
- The encryption algorithm is E
  - Its special knowledge is a <u>key</u> k
  - The ciphertext $C = E_k(P)$
- The decryption algorithm is D
  - Its special knowledge is a <u>key</u> k'
  - The plaintext $P = D_{k'}(C)$
  - By definition, $P = D_{k'}(E_k(P))$

The Caesar code

- Probably the oldest cryptographic algorithm
- *E* is: go forward N letters in the alphabet, rotating from Z to A.
  - *k* is N
- *D* is: go back N letters in the alphabet, rotating from A to Z.
  - *k'* is N
- When $k = k'$ we speak of a *symmetric-key* algorithm or a *shared key*. Both ends must know the same secret key.

What makes a good cryptographic algorithm?

- Assuming it's widely used, there's no point in trying to keep the algorithms *E* and *D* secret.
  - But you <u>must</u> keep your key secret
- It must be very hard (i.e. need thousands or more years of computing), even with complete knowledge of the algorithm, to try out all possible keys.
- Is the Caesar code a good algorithm?

How big should the key be?

- Obviously this depends on the exact *E* and *D* algorithms, but assume that the attacker has a few supercomputers.
- Let's assume (s)he can check one million keys per second.
- That's 31,536,000,000,000 keys per year.
- To be reasonably safe for 1000 years, you certainly need a pool of 31,536,000,000,000,000 keys to choose from.
- That's almost 2^{55} (a 55 bit binary number).
- Modern cryptography goes further than that, as we'll see.

Example: original DES* (1977)

- Divide message into 64 bit blocks of plaintext
- Encrypt each block with a 56 bit key
  - The encryption process includes 18 major steps, including transposition of bit strings and XOR between parts of the message and parts of the key
  - The output is a 64 bit block of ciphertext

0	0	0
0	1	1
1	0	1
1	1	0

The truth table for exclusive-or (XOR)

***** DES = Data Encryption Standard

Why DES uses XOR

- Note that XOR is in itself a simple symmetric cryptographic algorithm

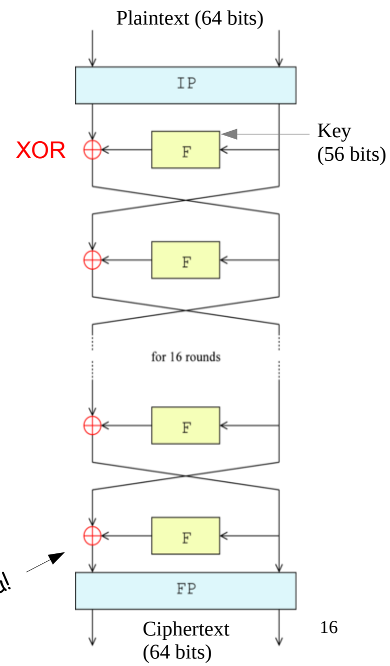
P=110011, k=010101 → C=XOR(110011,010101)=100110

C=100110, k=010101 → P=XOR(100110,010101)=110011

- What DES does is build on this property using multiple cycles and transpositions to make the result more pseudo-random
- Hard to crack without knowing the 56 bit key.
  - Is 56 bits enough?

Overview of DES

- IP = Initial Permutation (transposition)
- F = "Feistel" function (see Shay p. 289 for more details)
- FP = Final Permutation (swap and transposition, reverse of IP)
- In July 1998, the EFF's DES cracker (Deep Crack) broke a DES key in 56 hours. Cost: $250,000.



Thanks Wikipedia!

One step harder - DES-CBC

- CBC = Cipher Block Chaining
- Before each 64 bit plaintext block P_n is encrypted, XOR it with the previous cyphertext block C_{n-1} to add extra variability.
- After DES was broken, the first countermeasure was to use DES-CBC.
- Then...

Triple DES

- Basically, apply DES three times running, so that $C = E_{k3}(D_{k2}(E_{k1}(P)))$
- where *E* is DES encryption and *D* is DES decryption
  - if *k1=k2=k3* this is single DES for backwards compatibility
- Triple DES is still regarded as reasonably safe, but is slow, especially in software-only implementations.

Advanced Encryption Standard (AES)

- Preferred to Triple DES due to longer keys and greater complexity
  - Also has better software performance
- 128 bit block cipher with 128, 192 or 256 bit keys
- Mathematically complex
  - like DES, involves transposition steps and XOR, but also includes substitution tables in each round
  - currently regarded as safe for all practical purposes

The problem with symmetric keys

- Both ends must know the same key
- Doubles the risk of leaks
- Need to send the key initially from A to B, and how do you send the key in complete safety?
- If I need secure links to 100,000 customers I have to manage 100,000 keys

Asymmetric keys

- Suppose I could decrypt using *k'* and tell all my customers to encrypt using *k.*
- If I keep *k'* secret, nobody else can decrypt messages that were encrypted using *k.*
- So if I receive a message encrypted with *k* saying "Today's AES key is 11011....011101" , only I can decrypt it, and the AES key is safe.
- In this case *k* is my public key (everybody knows it) and *k'* is my private key (nobody else knows it).

RSA* algorithm

- Choose two large prime numbers *p* and *q*
  - Let $n = pq$
  - Let $n' = (p-1)x(q-1)$
- Find *k* which has no common factors with *n'.* *k* will be the encryption (public) key.
- Find *k'* such that *(kk'-1)* is an exact multiple of *n'.* *k'* will be the decryption (private) key.
- Encryption consists of raising each block of the plaintext to the power *k*, modulo *n*.
- Decryption consists of raising each block of the cyphertext to the power *k'*, modulo *n*.

* Rivest, Shamir and Adleman

Magic?

- RSA is based on number theory and seems like magic, but it works. Go through the example in Shay, or look at the excellent Wikipedia entry.

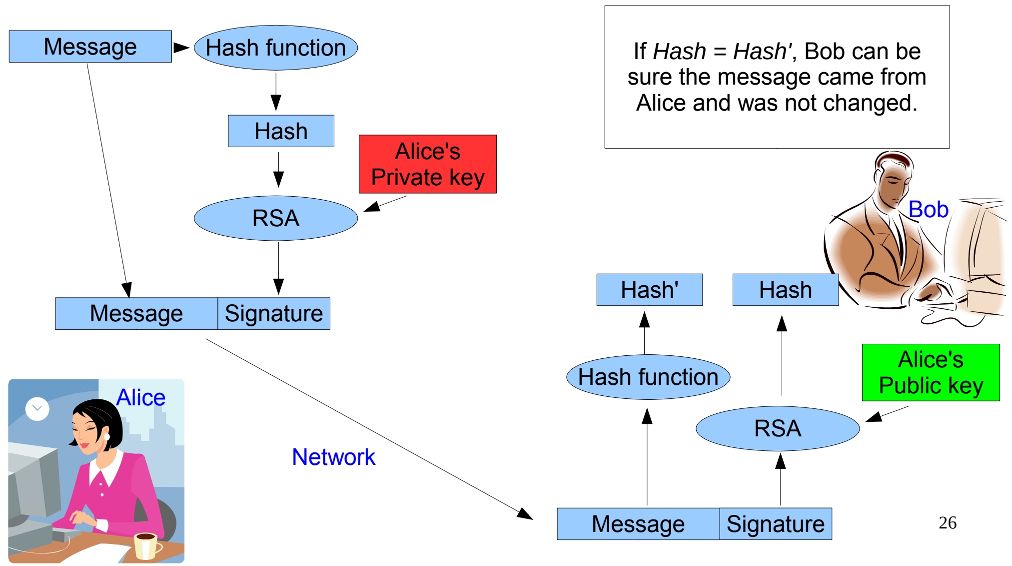
Two ways to use RSA keys

1. Alice uses Bob's public key to encrypt a message to Bob; only Bob can decrypt it.
   - But anybody could pretend to be Alice!
2. Alice uses her private key to encrypt a hash of her message; Bob uses Alice's public key to decrypt and check the hash value.
   - Only Alice can perform this encryption, so the encrypted hash is a digital signature.
   - If the hash matches, Bob knows that Alice sent the message <u>and</u> nobody changed it.
   - More magic: in fact, Alice uses RSA decryption to "encrypt" the hash, and vice versa.

Cryptographic Hash Functions

- These are functions somewhat like a checksum or CRC, but designed for cryptographic use.
  - Input is any length of message, and output is a fixed length hash value (at least 128 bits).
- Its mathematical design is not aimed at bit error detection, like a normal CRC, but at resistance to attack or detection of forgery.
  - In particular it should be very hard to find a fraudulent message that has the same hash as the genuine message
  - Preferred hash functions today are called SHA-256 and SHA-512

Signing a message: overview



Who are Alice and Bob anyway?

- In many analyses of security algorithms, Alice and Bob are the two parties trying to communicate securely, and often Eve is the person trying to listen in or interfere
  - Apologies to anyone called Alice, Bob or Eve...


Alice


Eve


Bob

What problems do Alice and Bob face?

- At the start, they can trust nothing - any message could be forged or read by Eve. They have to assume that:
  - Eve can see all their packets.
  - Eve can store packets and play them back later.
  - Eve can send her own packets with forged IP addresses.
  - Eve has a <u>lot</u> of computing power.

The importance of authentication

- We could spend the whole semester on security, but will focus on authentication.
- Authentication that a message was sent by a given source and not tampered with is the key to preventing most types of attack:
  - detects modification and spoofing of messages
  - prevents repudiation of genuine messages
  - helps detection of floods of invalid messages
  - helps to secure the sending of encryption keys across an initially insecure channel

How to authenticate that Bob is Bob

- We have to assume that Eve is trying to pretend to be Bob.
- So a message saying "I'm Bob" is suspect.
- A message signed with Bob's private key, that Alice can check with Bob's public key, is OK.
- But a message saying "Hi, I'm Bob and here's my public key", signed with the correspnding private key, isn't OK. Why not?

Who do you trust?

- If *www.BobsWebSite.org* lists Bob's public key, are you willing to believe it?
- If yes:
  - How do you know that Eve didn't create that web site?
  - How do you know that Eve didn't hack that web site, even if it's one that Bob created?
  - Are you sure you aren't looking at *www.BobsWebS1te.org*?
- Really, you can only trust a public key from a highly reputable source.

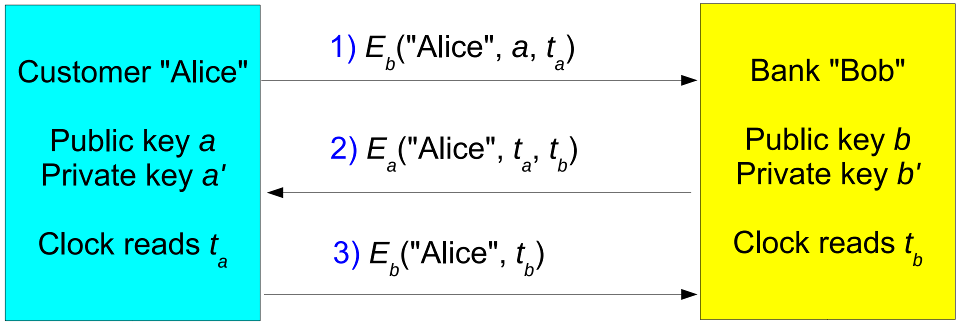
What can Alice do with a reputable public key for Bob?

- Check that it really is Bob who's sending messages to her and that they are unchanged (since Eve cannot forge Bob's RSA signature).
- Prove later that he really did send them (since Alice cannot forge Bob's RSA signature).
- Send a secure message to Bob providing a symmetric key for AES encryption (since Eve cannot read a message encrypted with Bob's public key).
- Efficiently discard any flood of bogus messages from Eve (since she cannot forge anybody else's RSA signature)

A simple authentication protocol

- *Problem:* Convince a bank called Bob that you really are a customer called Alice.
- *Notation:*
  - *E* is RSA encryption
  - *D* is RSA decryption
  - *a, a'* are Alice's public and private keys
  - *b, b'* are Bob's public and private keys
  - thus $E_a(P)$ is plaintext *P* encrypted with Alice's public key, etc.
  - t_a, t_b are clock times on Alice's and Bob's clocks

Does this work?



1) Alice provides her key and timestamp
2) Bob confirms timestamp and adds his own
3) Alice confirms Bob's timestamp

What did Bob and Alice learn?

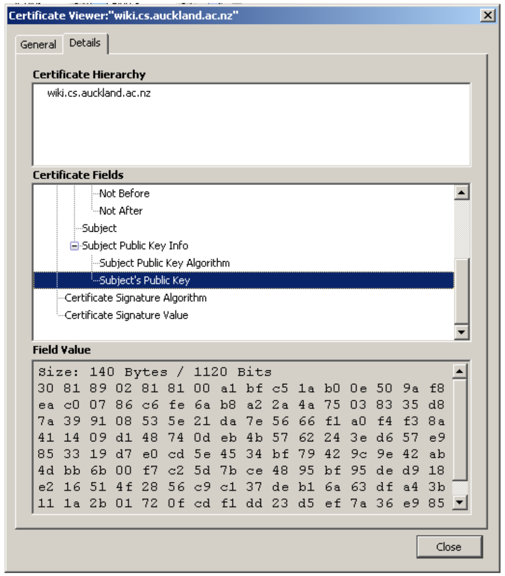
- Bob knows that "Alice" knew his public key
- Bob knows a public key for "Alice"
- Bob knows that "Alice" received his timestamp
- Alice knows that Bob knows her public key
- Alice knows that Bob received her timestamp
- Eve couldn't decipher the messages, but could store them
- *Has Alice proved her identity to Bob's server? (Authentication)*
- *Is Alice allowed to use Bob's service? (Authorization)*
- *Can Eve use a copy of message 3 to gain service? (Eavesdrop, then Replay; or Intercept, then Inject)*
- *What is the value of the timestamps?*

Authentication pitfalls

- How does Alice know she's talking to the genuine Bob?
  - This needs a source of trust for Bob's public key, typically an X.509 certificate
- How does Alice convince Bob she's the genuine Alice?
  - Typically this needs a reliable shared secret. The simplest kind is a pre-arranged password sent over an encrypted channel (e.g. encrypted with Bob's public key).

X.509 certificate

- This is a document that is cryptographically signed by a trusted third party known as a CA (Certification Authority).
- Apart from the signature and administrative material, it contains the public key.
- ("X.509" identifies a particular international standard.)



Trust is recursive

- Instead of trusting Bob's web site, Alice now has to trust Bob's CA.
- Web browsers have the public keys for reputable CAs built into them.
- Now Alice has to trust the web browser.
- So she has to trust the download site where the web browser came from.
- Which means trusting the download site's CA.
- Trust is not easy...

Summary on encryption and authentication

- We've seen how symmetric and asymmetric encryption systems work.
- They can be used to create secure channels and to check message authenticity.
- They can be used to build authentication protocols, but only based on some prior knowledge (a public key) and on some trusted third party.
- We'll see specific examples (TLS and SSH) later.