Lectures 11 -12

Security mechanisms

Brian Carpenter

Some slides originally from

Clark Thomborson

COMPSCI 314 S2C 2011

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.

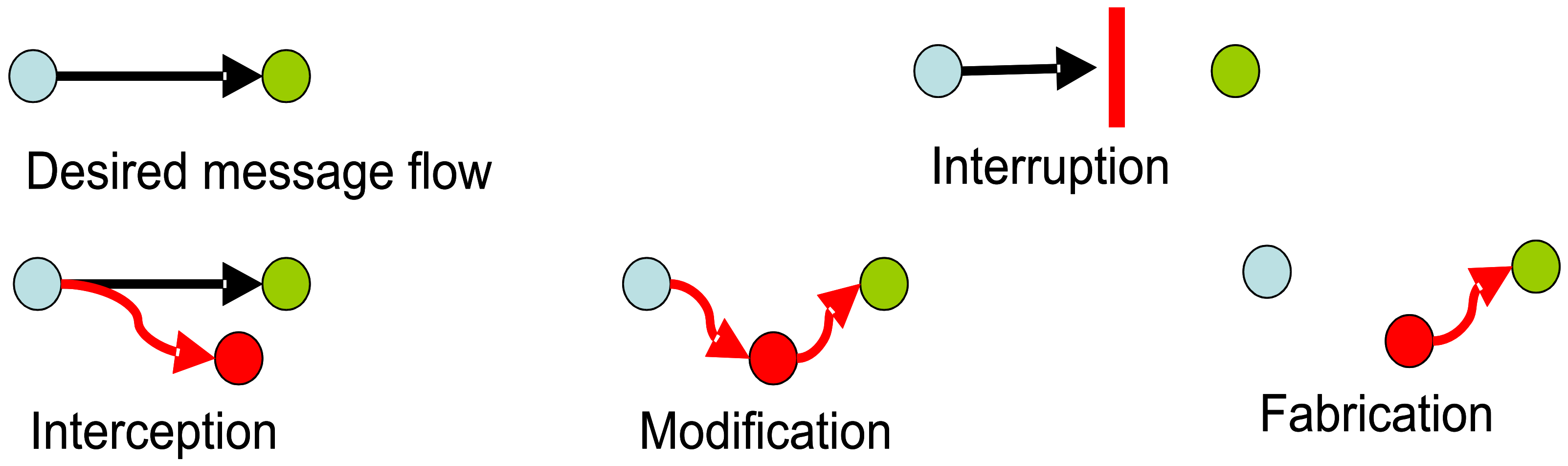
Network attacks (Stallman)

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.

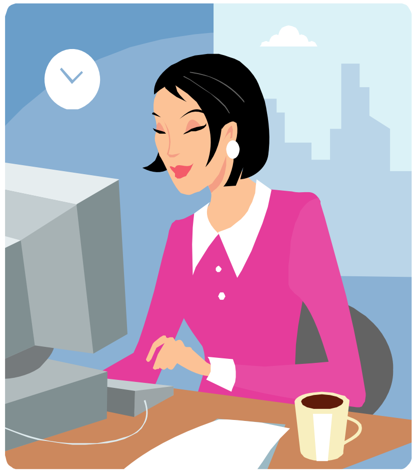


Who are these people anyway?

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.

See http://en.wikipedia.org/wiki/Alice_and_Bob for some other standard characters.

Apologies to anyone called Alice, Bob, Eve...


Alice

Eve

Bob

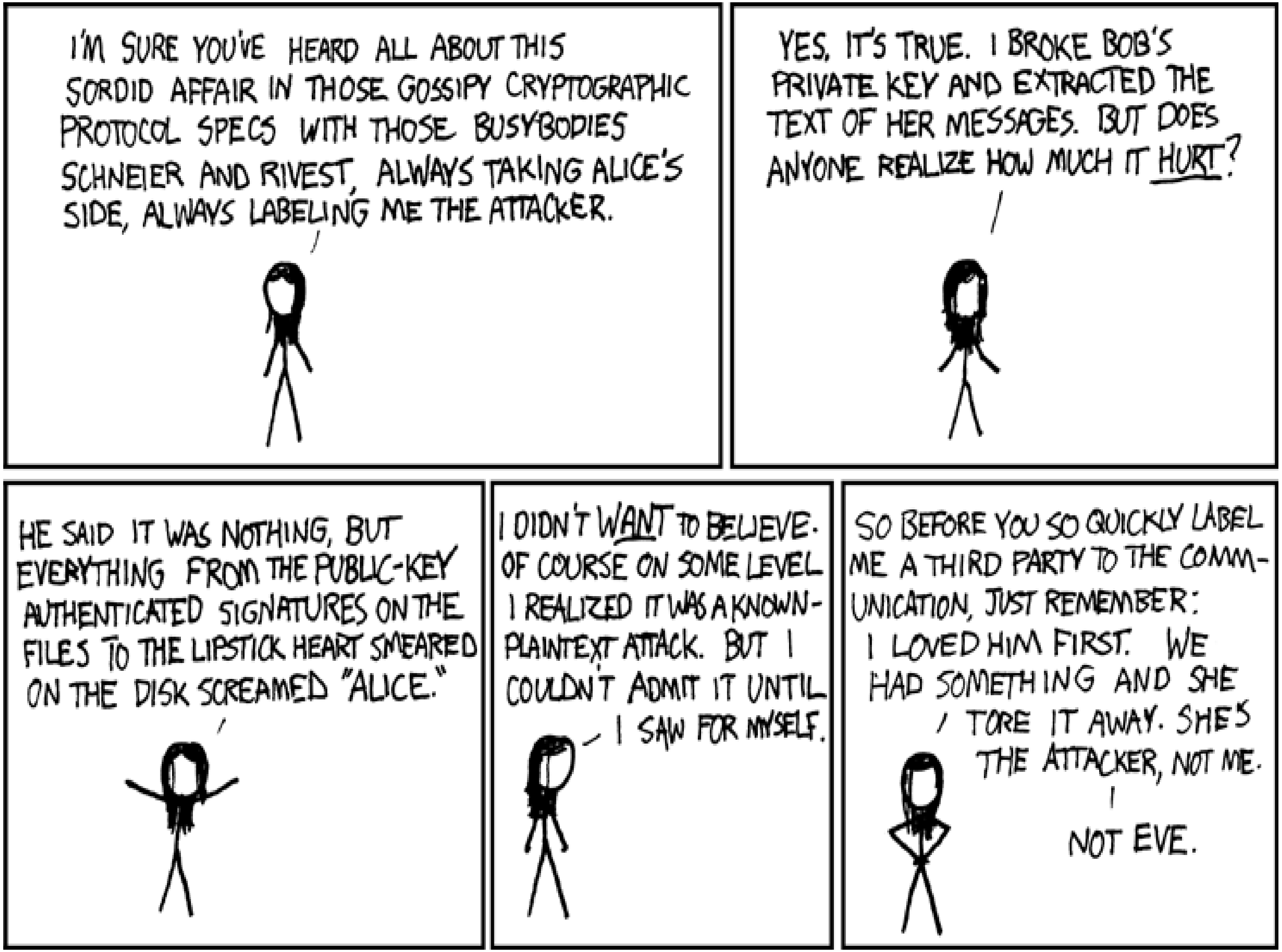
Darkside security (Thomborson)

One person's functional goal is another person's security threat – and vice versa.

Stallman's attack model is appropriate for Alice, in her traditional role in an analysis of communication security.

- Alice is talking to Bob.
- Eve is trying to eavesdrop: she poses a threat to the confidentiality of Alice's conversation.
- Mallory is trying to modify messages, posing a threat to the integrity of Alice's conversation.

Let's consider Eve's point of view...


I'M SURE YOU'VE HEARD ALL ABOUT THIS SORDID AFFAIR IN THOSE GOSSIPY CRYPTOGRAPHIC PROTOCOL SPECS WITH THOSE BUSYBODIES SCHNEIER AND RIVEST, ALWAYS TAKING ALICE'S SIDE, ALWAYS LABELING ME THE ATTACKER.
YES, IT'S TRUE. I BROKE BOB'S PRIVATE KEY AND EXTRACTED THE TEXT OF HER MESSAGES. BUT DOES ANYONE REALIZE HOW MUCH IT HURT?
HE SAID IT WAS NOTHING, BUT EVERYTHING FROM THE PUBLIC-KEY AUTHENTICATED SIGNATURES ON THE FILES TO THE LIPSTICK HEART SMEARED ON THE DISK SCREAMED "ALICE."
I DIDN'T WANT TO BELIEVE. OF COURSE ON SOME LEVEL I REALIZED IT WAS A KNOWN-PLAINTEXT ATTACK. BUT I COULDN'T ADMIT IT UNTIL I SAW FOR MYSELF.
SO BEFORE YOU SO QUICKLY LABEL ME A THIRD PARTY TO THE COMM-UNICATION, JUST REMEMBER: I LOVED HIM FIRST. WE HAD SOMETHING AND SHE TORE IT AWAY. SHE'S THE ATTACKER, NOT ME.
NOT EVE.

Source: http://xkcd.com/177/, reproduced with permission (http://xkcd.com/license.html).

Evading Walter

If Alice is a prisoner, she cannot communicate with Bob unless she has permission from Walter (her warden).

- Stegocommunication threat: Alice might find a surreptitious way to communicate with Bob.

Have you ever wondered about the stegomessages which might be sent, without your knowledge, by your computer?

- Using open-source code can mitigate this threat...
- Do you trust the person who compiled the code you are using?
- Do you trust the people who wrote the compiler? http://doi.acm.org/10.1145/358198.358210

Evading Mallory

If Alice doesn't have a right to integrity in her messaging...

- Threat: Alice might add error-correcting codes to her messages, which would allow Bob to “undo” Mallory's modifications.

Many of your documents have metadata which could reveal information you would not want to reveal.

- Many lawyers have learned, the hard way, never to send contractual offers in MS Word format.
- How can you be confident that Alice (your word-processing software, or your OS) won't actually preserve some information you “delete” from a document or a filesystem?

Evading Daniel

What if Alice doesn't have a right to availability in her messaging?

- Threat: Alice might find a way to evade Daniel, whose goal is to deny service.

If your computer or browser (Alice) is taken over by malware, will you be able to shut off its communications?

- You can easily unplug a wired-Ether connection... but can you shut down all of the comm channels on your cellphone or PDA?
- What can you do if your computer or cellphone doesn't respond to its “off” switch?

Evading Fabian

What if Bob doesn't have the right to know that it was actually Alice who sent the message signed “Alice”?

- Threat: Alice might find a way to sign her messages so that Fabian (a fabricator) can not fool Bob.

I don't have a white-hat scenario for this threat... can you think of one?

The point I'm trying to make in these last few slides is that “Security” is not a well-defined property, until you assign roles to participants and decide who is wearing a “white-hat”!

Back to CIA...

Most networks are designed for the “CIA” goals:

Security goal #1: confidentiality for Alice & Bob

Security goal #2: integrity for Alice & Bob

Security goal #3: availability for Alice & Bob

It seems to be infeasible to achieve all three goals.

In the usual design for a “secure communication system”,

availability is compromised, and

Alice and Bob gain (partial) confidentiality with encryption. Eve can tell that Alice & Bob are sending messages to each other, but she can't understand what the messages mean.

Encryption = coding with a secret

Coding schemes are designed to be decoded by an algorithm that is widely known.

Encryption schemes are codes which need secret knowledge to decode them.

`Xibu jt uijt tjnqmf fodszqujpo?`

cyphertext

Types of Secrets...

The decoding algorithm might be a secret.

- “Security by obscurity”: a bad idea (unless it's your only option)

A generally-known algorithm might require a secret input: the decoding “key”.

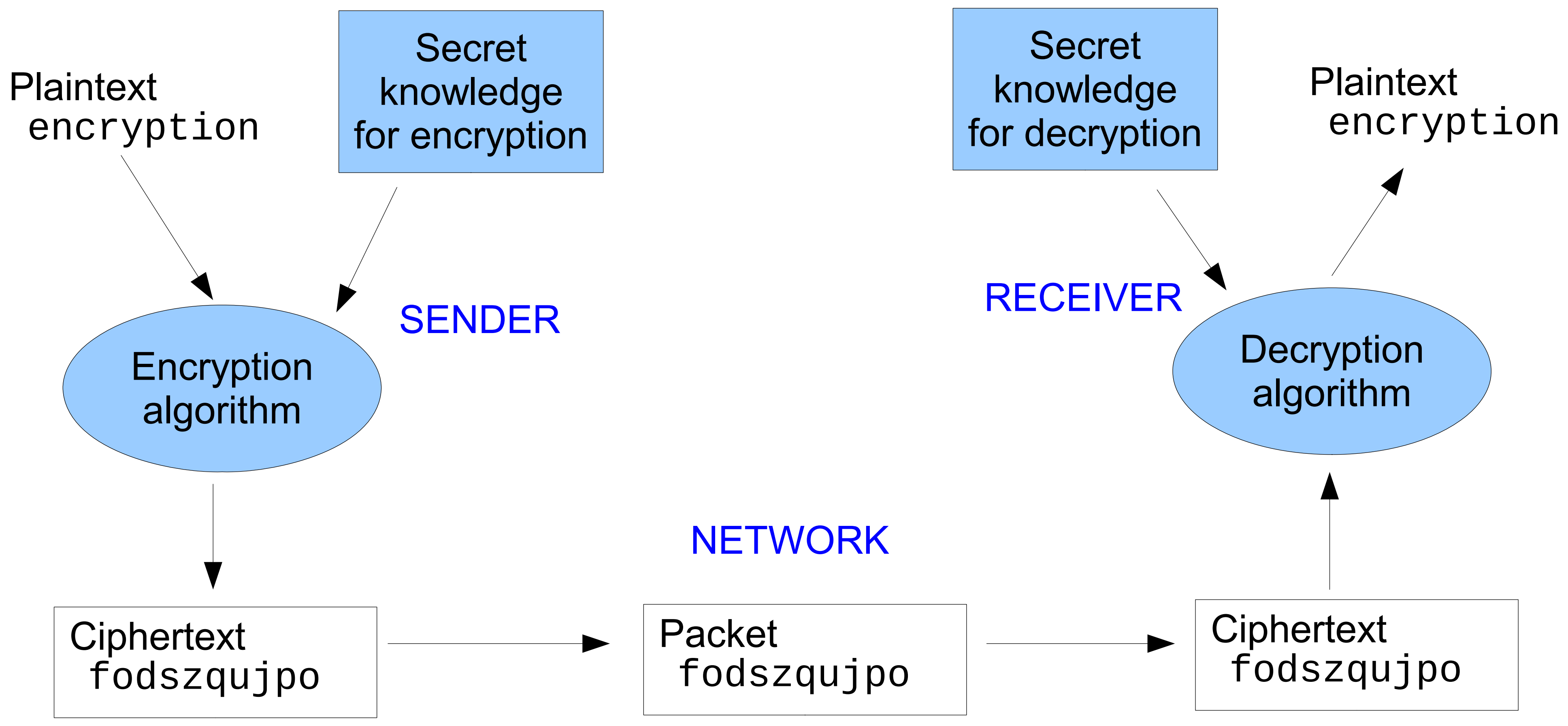
```
Xibu jt uijt tjnqmf fodszqujpo?
What is this simple encryption?
```

plaintext
or cleartext

The algorithm is "go back N letters"

The secret knowledge (the key) is "N=1"

Encryption and decryption



Without the secret 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 secret knowledge is a <u>key</u> k

The ciphertext $C = E_k(P)$

The decryption algorithm is D

Its secret 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.

- Disclosing E and D can be beneficial. See Tomlinson (1853) and Kerckhoffs (1887).
- But you <u>must</u> keep your key secret!

A cryptographic system is called “strong” if experts believe no one will crack an encoded message within the next 20 years – not even using millions of computers trying all possible keys.

- Is the Caesar code strong?

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.

Another simple algorithm: Exclusive Or (XOR)

a	***b***	***XOR(a,b)***
0	**0**	**0**
0	**1**	**1**
1	**0**	**1**
1	**1**	**0**

The truth table for XOR

- XOR is a symmetric cryptographic algorithm:

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

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

- In this example there are $2^6 = 64$ possible keys, so it’s easy to find the key by trial and error.

Example: original DES* (1977)

Divide message into 64 bit blocks of plaintext

Encrypt each block with a 56 bit key

DES builds on simple XOR encryption using multiple cycles and transpositions to make the result more pseudo-random.

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

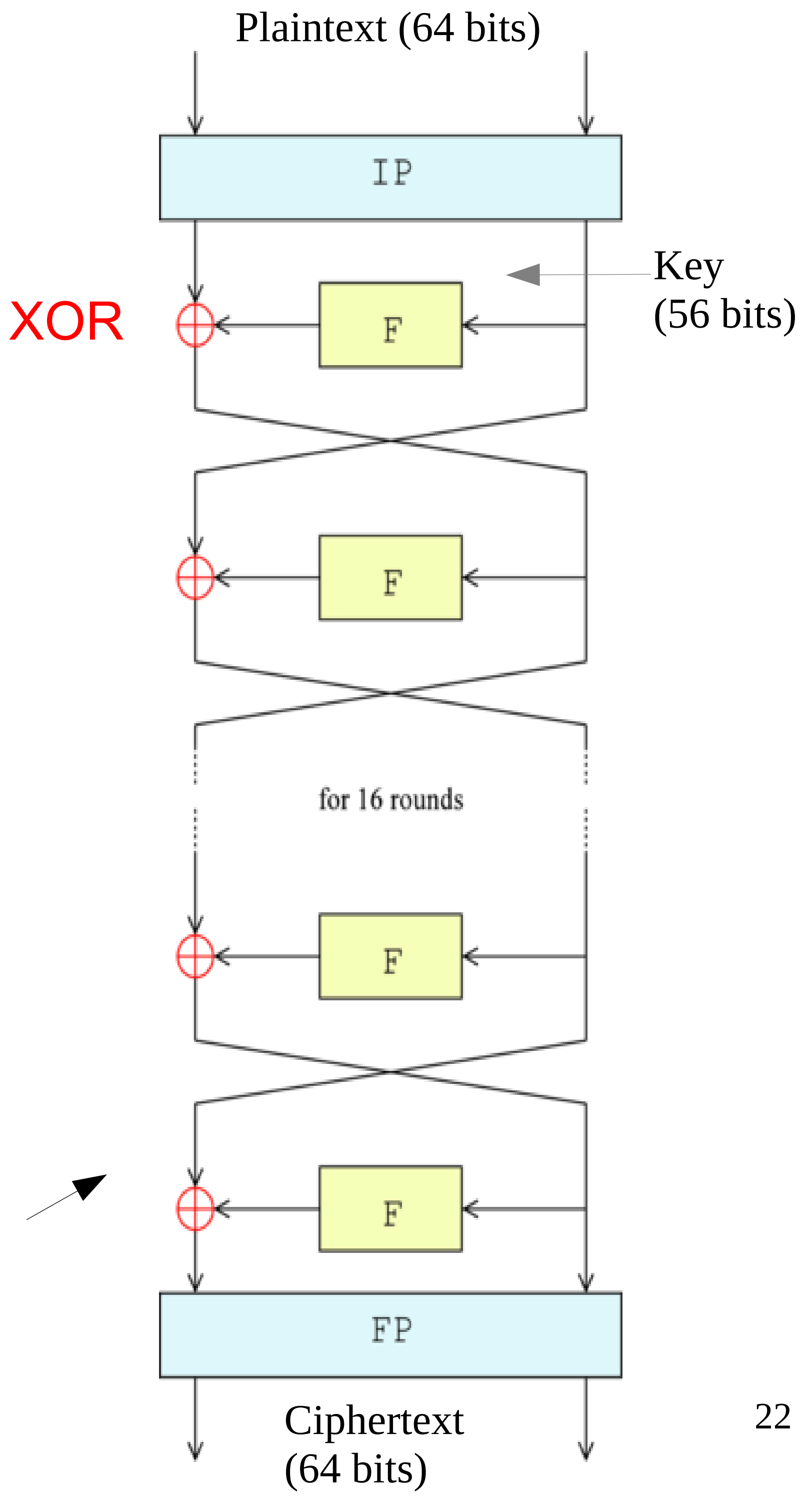
* DES = Data Encryption Standard

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)



Thanks Wikipedia!

Cracking DES

DES should be very hard to crack without knowing the 56 bit key. So let’s consider trial and error...

2^{56} seems like a big number of trials! That’s 72,057,594,037,927,936. Is it enough?

Apparently not! In July 1998, the EFF*'s DES cracker (Deep Crack) broke a DES key in 56 hours. Cost: $250,000US.

* EFF = Electronic Frontier Foundation

Block vs. Stream Ciphers

DES is a block cipher – it produces 64-bit blocks of cipher text from 64-bit blocks of cleartext.

- If Alice encrypts the same cleartext *M* under the same key *k*, she'll get the same ciphertext $E_k(M)$.

Alice must change keys more often than she sends the same cleartext block.

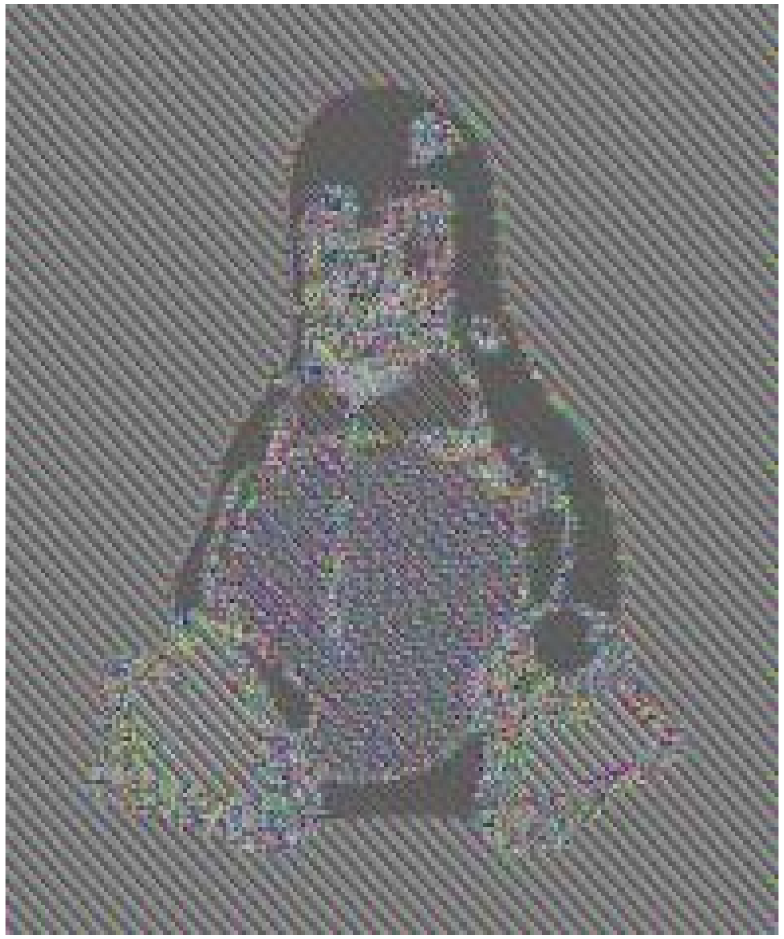
- Otherwise she'll reveal information about her “repeated blocks” to Eve. (See the next slide.)

Alice wants to send long messages without worrying about repeats and key-changes.

- She really wants a stream cipher, not a block cipher.

Cryptanalysis: Repeated Blocks

If we DES-encode all 64-bit blocks of a picture (in bitmap format) under the same key...



http://en.wikipedia.org/wiki/Block_cipher_modes_of_operation,

http://www.isc.tamu.edu/~lewing/linux/

A Quick-and-Dirty Stream Cipher

CBC = Cipher Block Chaining

- Before each 64-bit plaintext block P_n is encrypted, XOR it with the previous cyphertext block C_{n-1}
- Repeated blocks are now very rare, so the trivial pattern-matching attack of the previous slide fails.
- A clever attacker can still guess a pattern of repeats...

Alice should not repeat herself, if she's using a CBC cipherstream and is worried about a clever Eve.

- Alice should compress her cleartext before encrypting it.

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

Problems with symmetric keys

Both ends must know the same key

- Doubles the risk of leaks
- Can't determine who leaked the key

Initialisation problem: How can Alice send a key safely to Bob without encrypting it?

- In practice: use an existing secure channel (post?, telephone?), monitor the first few uses of a new key, use Diffie-Hellman key-exchange, ...

If I want secure links to 100,000 customers, then 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 $n' = (p-1) \times (q-1)$

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

Alice uses Bob's public key to encrypt a message to Bob; only Bob can decrypt it.

But anybody could pretend to be Alice!

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.

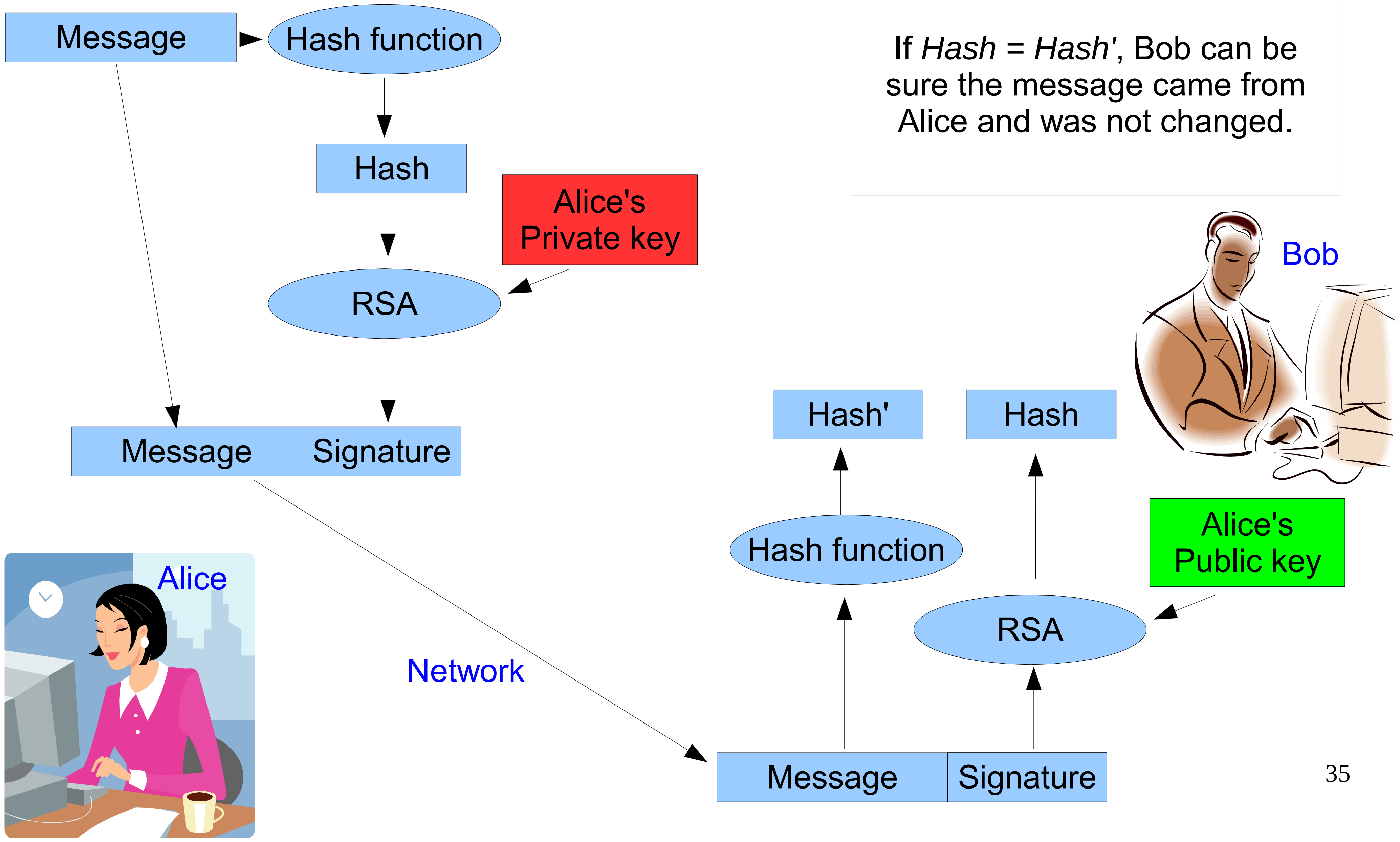
Input is any length of message, and output is a fixed length hash value (at least 128 bits).

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 create a fraudulent message that has the same hash as the genuine message

SHA-256 and SHA-512 are commonly used, and still seem secure, but are nearing the end of their useful life. http://csrc.nist.gov/groups/ST/hash/sha-3/Round2

Signing a message: overview



If *Hash* = *Hash'*, Bob can be sure the message came from Alice and was not changed.

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.

“Source 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 assume that Eve is trying to pretend to be Bob.

A message that merely says "I'm Bob" proves nothing... and might be suspicious! (Would Bob really send such a message?)

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. (But... how do you identify a reputable source??!)

Trust and Trustworthiness

Security analysts distinguish “trust” from “trustworthiness”.

If Alice believes that she has a valid copy of Bob's public key, then she “trusts” this key whenever she relies on it (i.e. to verify a message from Bob).

If Alice actually has Bob's public key, then her reliance on the validity of this key is appropriate – we say this key is “trustworthy”.

What can Alice do with a trusted 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, to herself, and to anyone else who trusts this key, that Bob really did send a message (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 Daniel (since Daniel 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

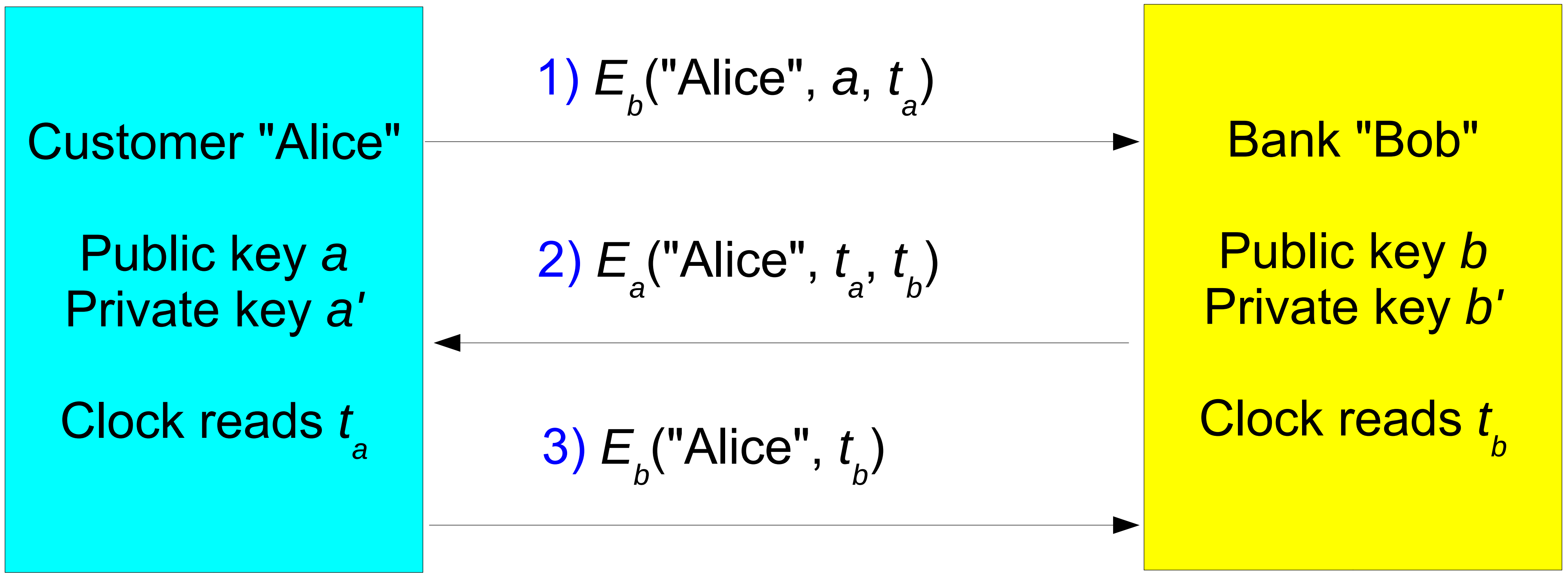
a, a' are Alice's public and private keys

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?



Alice provides her key and timestamp

Bob confirms timestamp and adds his own

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?

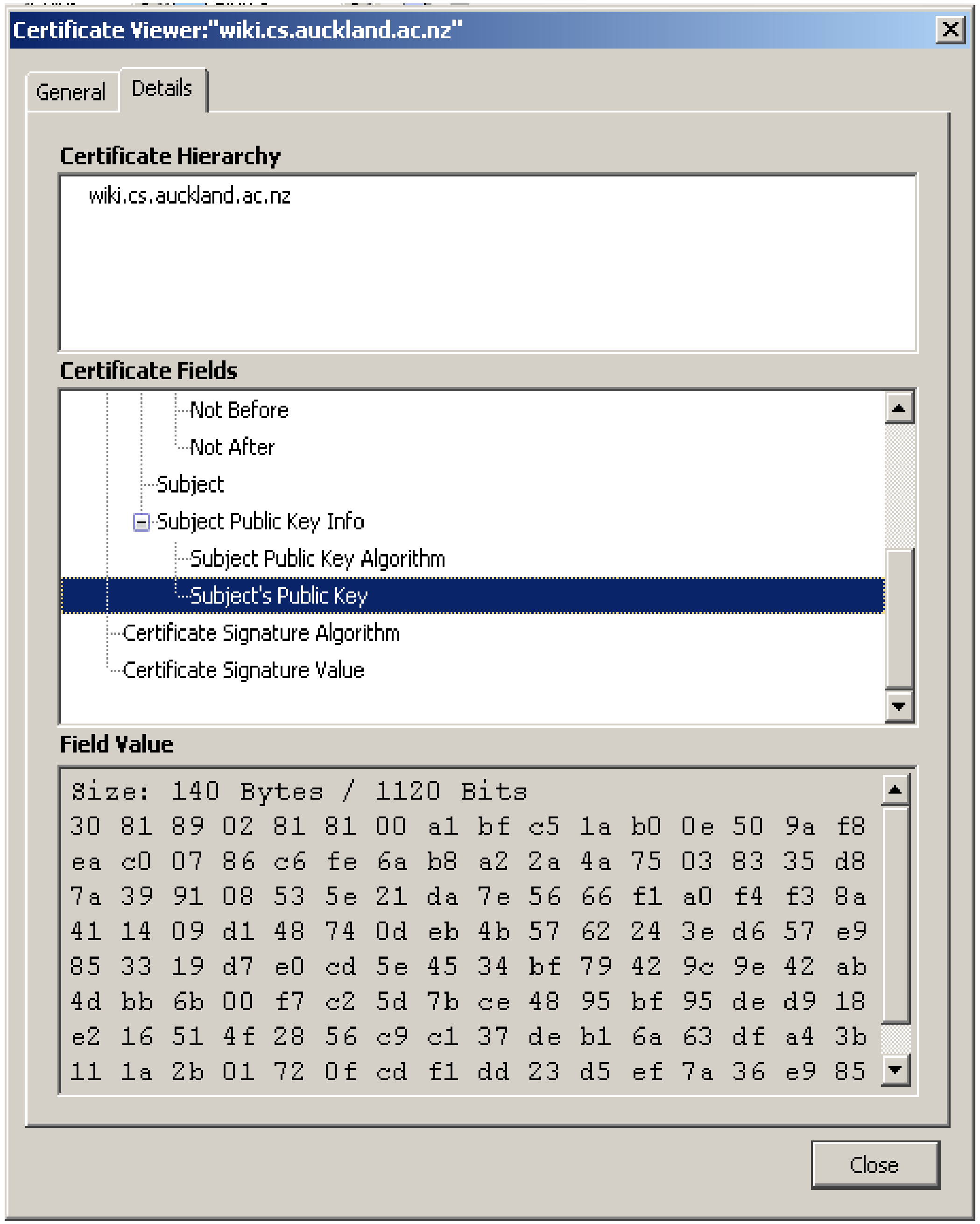
Typically this needs a reliable shared secret. The simplest kind is a pre-arranged password sent over an encrypted channel (e.g. encrypted with Bob's public key).

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.)



Certificate Hierarchy

wiki.cs.auckland.ac.nz

Certificate Fields

- Not Before
- Not After
- Subject
- Subject Public Key Info
  - Subject Public Key Algorithm
  - Subject's Public Key
- Certificate Signature Algorithm
- Certificate Signature Value

Field Value

```
Size: 140 Bytes / 1120 Bits
30 81 89 02 81 81 00 a1 bf c5 1a b0 0e 50 9a f8
ea c0 07 86 c6 fe 6a b8 a2 2a 4a 75 03 83 35 d8
7a 39 91 08 53 5e 21 da 7e 56 66 f1 a0 f4 f3 8a
41 14 09 d1 48 74 0d eb 4b 57 62 24 3e d6 57 e9
85 33 19 d7 e0 cd 5e 45 34 bf 79 42 9c 9e 42 ab
4d bb 6b 00 f7 c2 5d 7b ce 48 95 bf 95 de d9 18
e2 16 51 4f 28 56 c9 c1 37 de b1 6a 63 df a4 3b
11 1a 2b 01 72 0f cd f1 dd 23 d5 ef 7a 36 e9 85
```

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.