Distributed Systems (COMP9243)

Lecture 7: Security

① Introduction

② Cryptography

③ Secure protocols and communication

④ Authentication

⑤ Authorisation

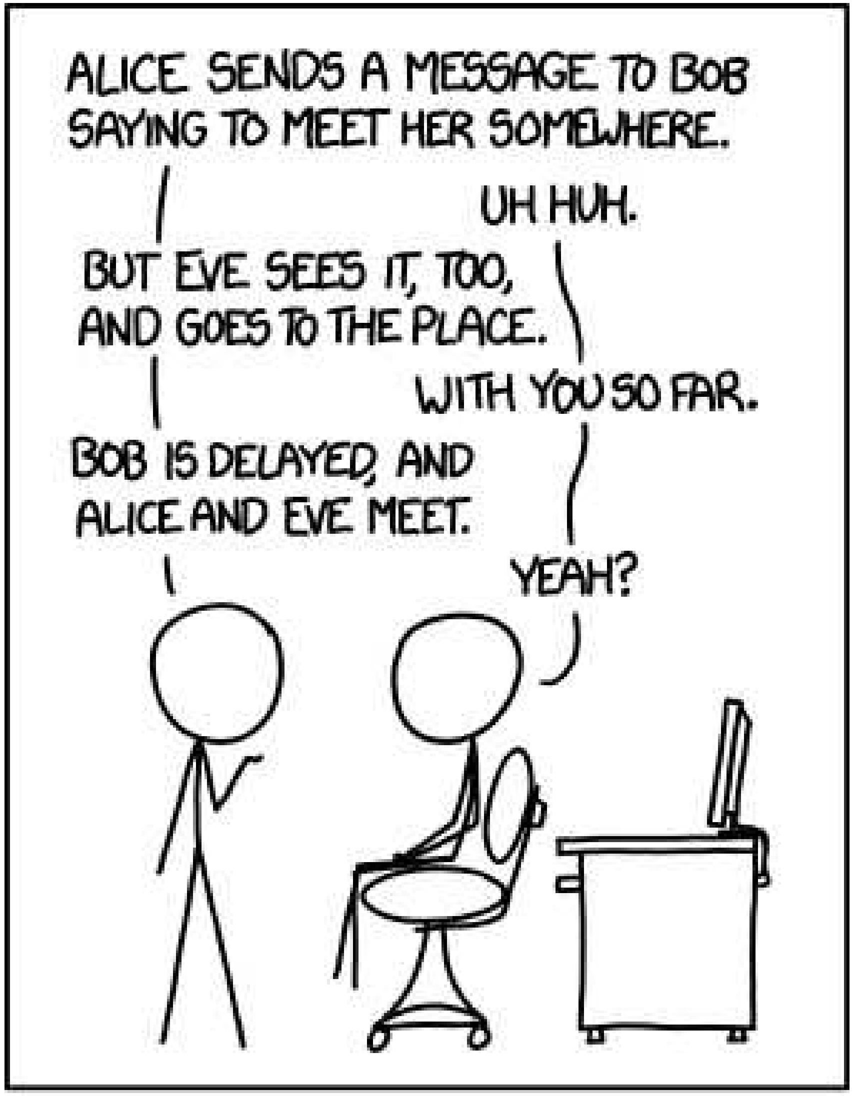
Security in Distributed Systems

Confidentiality: information disclosed/services provided only to authorised parties

Integrity: alterations can only be made in an authorised way

Availability: system is ready to be used by authorised parties

The Cast



I'VE DISCOVERED A WAY TO GET COMPUTER SCIENTISTS TO LISTEN TO ANY BORING STORY.

The Good Guys:

- ➜ Alice, Bob
- ➜ Want to communicate securely

The Bad Guys:

- ➜ Eve
- ➜ The eavesdropper — tries to thwart Alice and Bob's plans

The Alice and Bob After Dinner Speech:

- ➜ google it for more about Alice and Bob

Authorised Actions

Security is about making sure that only authorised actions are performed in the system.

Example Actions:

- ➜ Reading data
- ➜ Modifying data (writing, creating, deleting)
- ➜ Using a service
- ➜ Managing a service

All of these could be abused if performed in unauthorised ways.

Examples?

Security Policy

Security is a question of tradeoffs

Security Policy:

- A statement of security requirements
- Describes which actions entities in a system are allowed to take and which ones are prohibited
  - Entities: users, services, data, machines, etc.
  - Operations: read, write, send, start, stop, etc.

Example:

- Everyone (staff and students) has an account
- Access to course accounts must be approved
- Only course accounts can modify grades

Anything missing?

Breaking Security

Vulnerability:

A *vulnerability* is a weakness in the system that could potentially be exercised (accidentally triggered or intentionally exploited) to cause a breach or violation of the system's security policy.

Threat:

A *threat* is a possible breach of security policy (the potential for an attack). A concrete threat consists of a *threat-source* and an exercisable vulnerability.

Attack:

When a vulnerability is exercised we call this an *attack*.

Classes of Security Threats

Interception: unauthorised party has gained access to a service or data

Interruption: service or data become unavailable, unusable, destroyed, etc.

Modification: unauthorised changing of data or tampering with a service (so that it no longer adheres to its specifications)

Fabrication: additional data or activity are generated that would normally not exist

Attacking a Distributed System

Attacking the Communication Channel:

- ➜ Eavesdropping
- ➜ Masquerading
- ➜ Message tampering
- ➜ Denial of service

Attacking the Interfaces:

- ➜ Unauthorised access
- ➜ Denial of Service

Attacking the Systems:

- ➜ Applications
- ➜ OS
- ➜ Hardware

Protecting a Distributed System

Controls:

Authentication: verify the claimed identity of an entity

Authorisation: determine what actions an authenticated entity is authorised to perform

Auditing: trace which entities access what

Message Confidentiality: secret communication

Message Integrity: tamperproof messages

Security Mechanisms

Good Mechanisms:

Encryption: transform data into something an attacker cannot understand

- A means to implement confidentiality
- Support for integrity checks (check if data has been modified)

Signatures and Digests support for integrity, authentication

Secure Protocols support for authentication, authorisation

Secure Communication support confidentiality and integrity

Security Architecture based on sound principles such as: small TCB, Principle of Least Privilege, support for authorisation

Less Good Mechanisms:

Obscurity: count on system details being unknown

Intimidation: count on fear to keep you safe

Why Security is Hard

Weakest Link:

- ➜ Security of a system is only as strong as its weakest link
- ➜ Need to make sure all weak links are removed
- ➜ One bug is enough
- ➜ People are often the weakest link

Complexity:

- ➜ Security involves many separate subsystems
- ➜ Complex to set up and use
- ➜ People won't use complex systems

Pervasiveness:

- ➜ Application level
- ➜ Middleware level
- ➜ Network level
- ➜ OS level, Hardware Level

How To Make It Easier

Distribution of Mechanisms:

- ➜ Trusted Computing Base (TCB): *those parts of the system that are able to compromise security*
- ➜ The smaller the TCB the better.
- ➜ May have to implement key services yourself
- ☑ Physically separate security services from other services

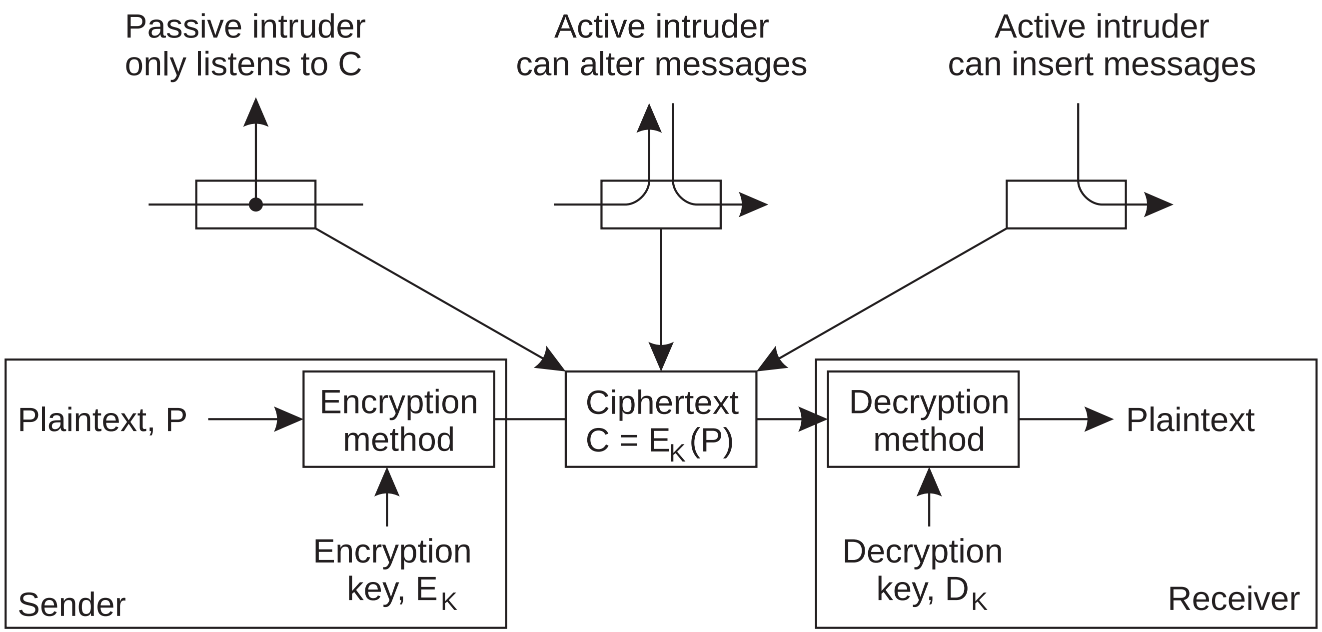
Simplicity:

- ➜ Simplicity contributes to trust
- ➜ Very difficult to make a simple secure system

FOUNDATIONS

- ➜ Cryptography
  - Ciphers
  - Signatures and Digests
  - Secure Communication
  - Security Protocols
- ➜ Authentication
- ➜ Authorisation

The Basic Idea:



- ➜ Map cleartext (or plaintext) T to ciphertext (or *cryptogram*) C
- ➜ Mapping is by a well-known function parameterised by a key K
- ➜ T infeasible to reconstruct from C without knowledge of key
- ➜ $E(K_E, T) = \{T\}_{K_E}$; $D(K_D, C) = \{C\}_{K_D}$; $\{\{T\}_{K_E}\}_{K_D} = T$

Cryptographer:

➜ Uses cryptography to convert plaintext into ciphertext

Cryptanalyst:

➜ Uses cryptanalysis to attempt to turn ciphertext back into plaintext

➜ Cryptanalysis: the science of making encrypted data unencrypted

Encryption

The essence of encryption functions:

Find a function E that is easy to compute, but for which it is hard to compute T from $\{T\}_{K_E}$ without a matching decryption key K_D for K_E.

- ➜ "Hard to compute" means that it must take at least hundreds of years to reverse E without knowledge of K_D or to compute K_D
- ➜ Such functions are known as one-way functions.

Cipher must be resilient to:

- ➜ Ciphertext only attacks
- ➜ Known plaintext attacks
- ➜ Chosen plaintext attacks
- ➜ Brute-force attacks

What properties should a good cipher possess?

- Confusion and Diffusion
  - *Confusion*: every bit of key influences large number of ciphertext bits
  - *Diffusion*: every bit of plaintext influences large number of ciphertext bits
- Fast to compute, ideally in hardware. Is this always good?
- Not critically depend on users selecting "good" keys
- Have been heavily scrutinised by experts
- Based on operations which are provably "hard" to invert
- Easy to use

In practice, keys are of finite length. Consequences?

- ➜ Finite key space ⇒ susceptible to exhaustive search
- ➜ Longer keys ⇒ more time needed for brute-force attack
  - Time to guess a key is exponential in the number of bits of the key
- ✗ Longer keys also make E and D more expensive
- ➜ Cipher must be secure against any systematic attack significantly faster than exhaustive search of key space

Basic Ciphers

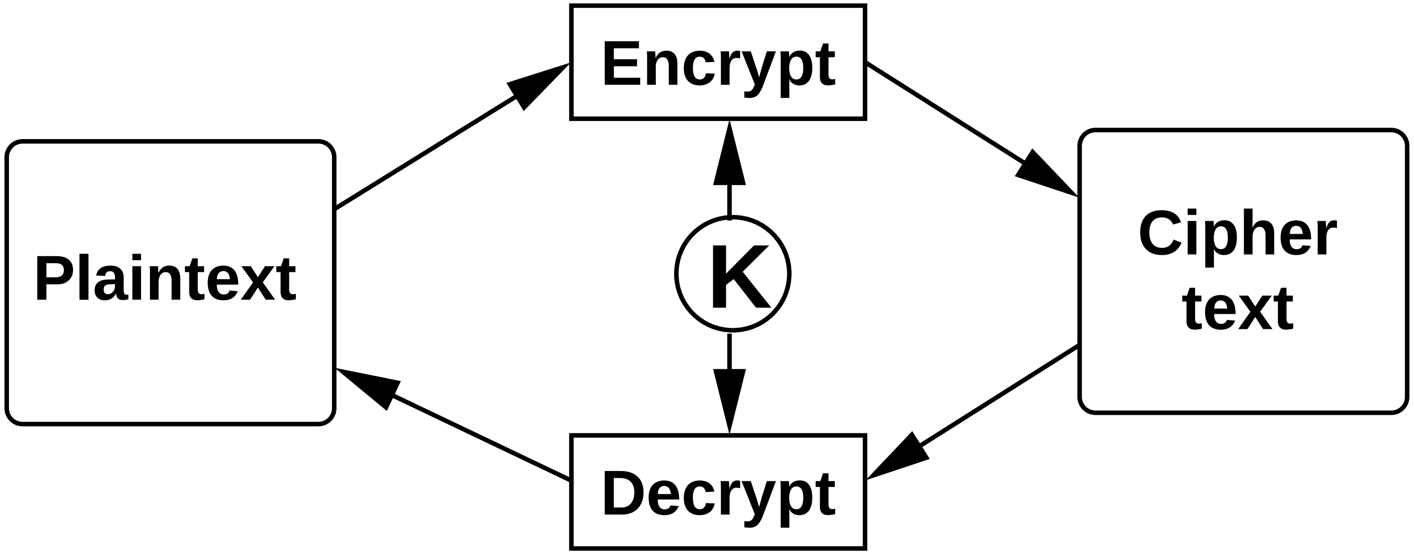
Substitution Ciphers:

- ➜ Each plaintext character replaced by a ciphertext character
- ➜ Caesar cipher: shift alphabet x positions
  - Easy to break using statistical properties of language
- ➜ Book cipher: replace words by location of word in book
  - Knowledge of book is the key

One Time Pads:

- ➜ Random string XORed with plaintext
- ➜ Information theoretically secure
- ➜ Random string must:
  - Have no pattern or be predictable
  - Not be reused
  - Not be known by cryptanalyst
- ➜ Key distribution problem

SYMMETRIC CIPHERS



- ➜ Secret key: $K_E = K_D$
- ☑ fast $\Rightarrow$ suited for large data volumes
- ☒ Secure channel is needed to establish the shared, secret key
- ➜ How many keys needed for N agents?
  - ➻ For any two agents, one key is needed

Tiny Encryption Algorithm (TEA)

Symmetric encryption algorithm by Wheeler & Needham:

- ➜ Encode a 64-bit block (`text`) consisting of two 32-bit integers
- ➜ Using a 128-bit key (`k`) represented by four 32-bit integers
- ➜ Despite its simplicity, TEA is a secure and reasonably fast encryption algorithm
- ➜ Can easily be implemented in hardware
- ➜ Approximately three times as fast as DES
- ➜ Achieves complete diffusion

```c
void encrypt (unsigned long k[], unsigned long text[])
{
 unsigned long y = text[0], z = text[1];
 unsigned long delta = 0x9e3779b9, sum = 0; int n;
 for (n = 0; n < 32; n++) {
   sum += delta;
   y += ((z << 4) + k[0]) ^ (z+sum) ^ ((z >> 5) + k[1]);
   z += ((y << 4) + k[2]) ^ (y+sum) ^ ((y >> 5) + k[3]);
 }
 text[0] = y; text[1] = z;}
```

- 32 rounds: shift and combine the halves of `text` with the four parts of the key
- Constant `delta` is used to obscure the key in portions of the plaintext that do not vary
- Confusion (xor operations and shifting of the text) and diffusion (shifting and swapping of the two halves of the text)

```
void decrypt (unsigned long k[], unsigned long text[])
{
 unsigned long y = text[0], z = text[1];
 unsigned long delta = 0x9e3779b9, sum = delta << 5;  int n;
 for (n = 0; n < 32; n++) {
   z -= ((y << 4) + k[2]) ^ (y + sum) ^ ((y >> 5) + k[3]);
   y -= ((z << 4) + k[0]) ^ (z + sum) ^ ((z >> 5) + k[1]);
   sum -= delta;
 }
 text[0] = y; text[1] = z;
}
```

Other Symmetric Ciphers

Data Encryption Standard (DES):

- Developed by IBM for US government
- 56 bit key. No longer considered safe.
- Triple DES: $2x56$ bit key. encrypt-decrypt-encrypt

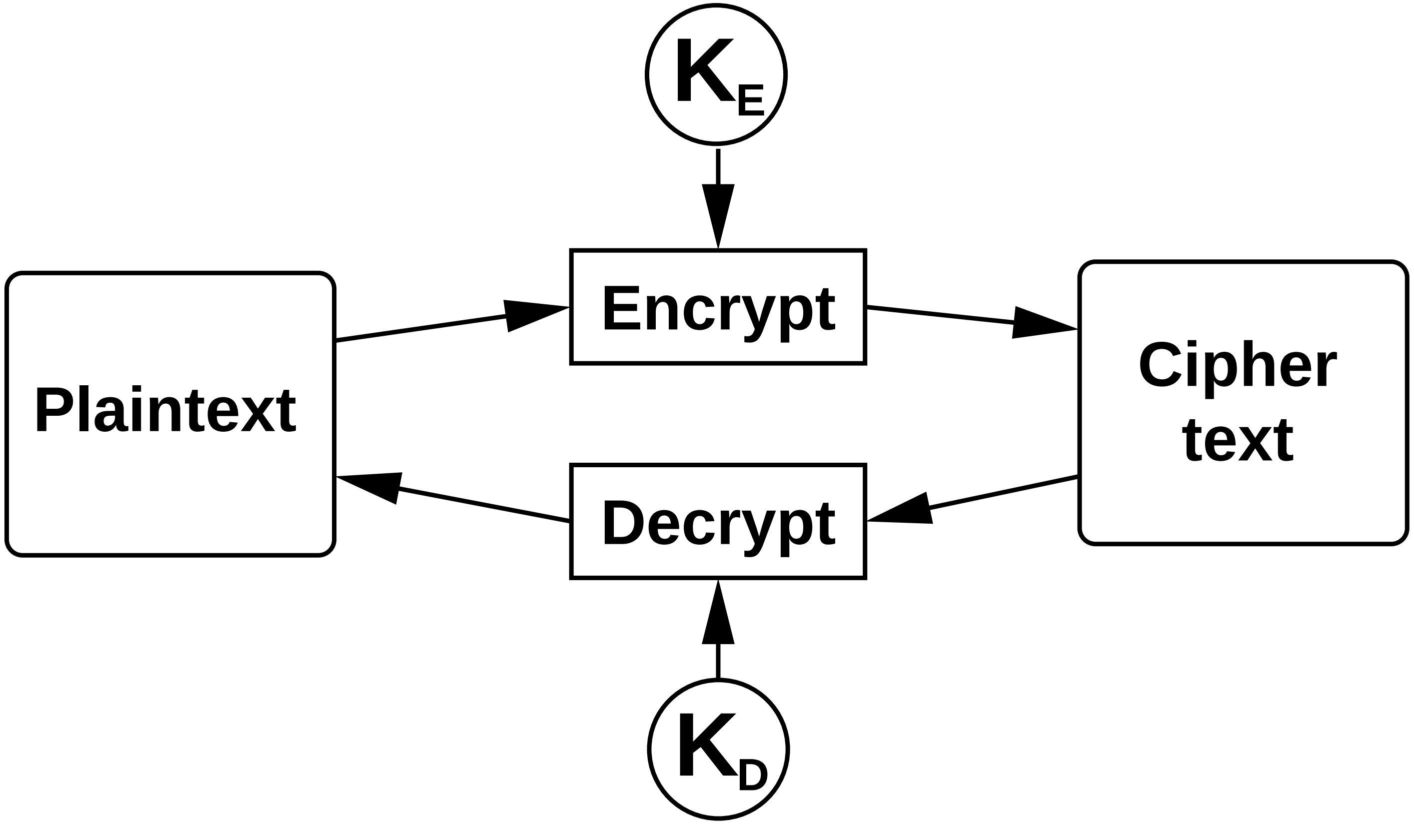
International Data Encryption Algorithm (IDEA):

- Uses 128-bit key to encrypt 64-bit blocks
- Approximately three times as fast as DES
- Same function for encryption and decryption (like DES)

Advanced Encryption Standard (AES):

- Defined in 2001, to replace DES
- Variable block and key length; specification 128, 192, or 256 bit keys and 128, 192 or 256 bit blocks

Asymmetric ciphers



- Due to Diffie & Hellman & Merkle (1976)
- Instead of one secret key per pair of agents, one public/private key pair per agent
- $K_E \neq K_D$, K_D infeasible to compute from K_E

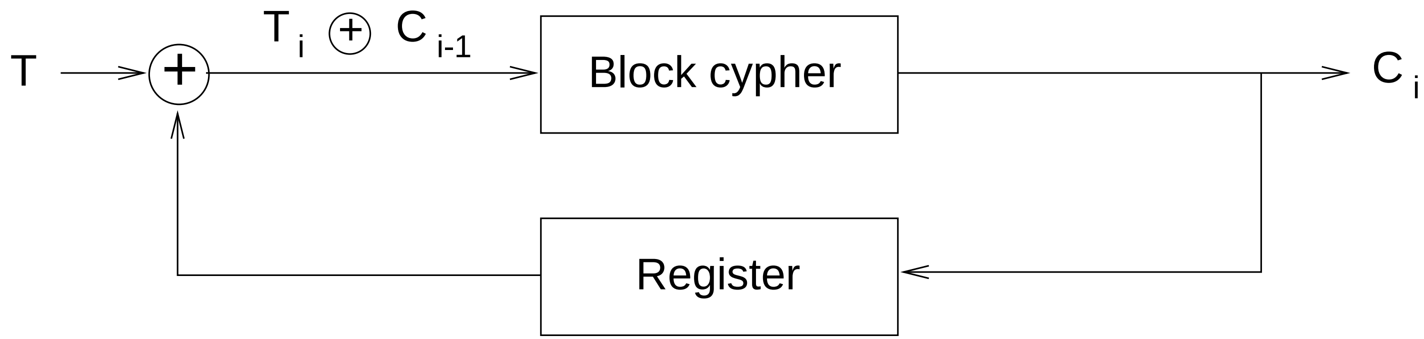
- each agent can publish public key $K_E =: K_P$, keep private key $K_D =: K_p$ secret
- Too slow to encrypt large volumes of data
- Examples: RSA and variants of Diffie & Hellman's original algorithm, such as ElGamal

How they work:

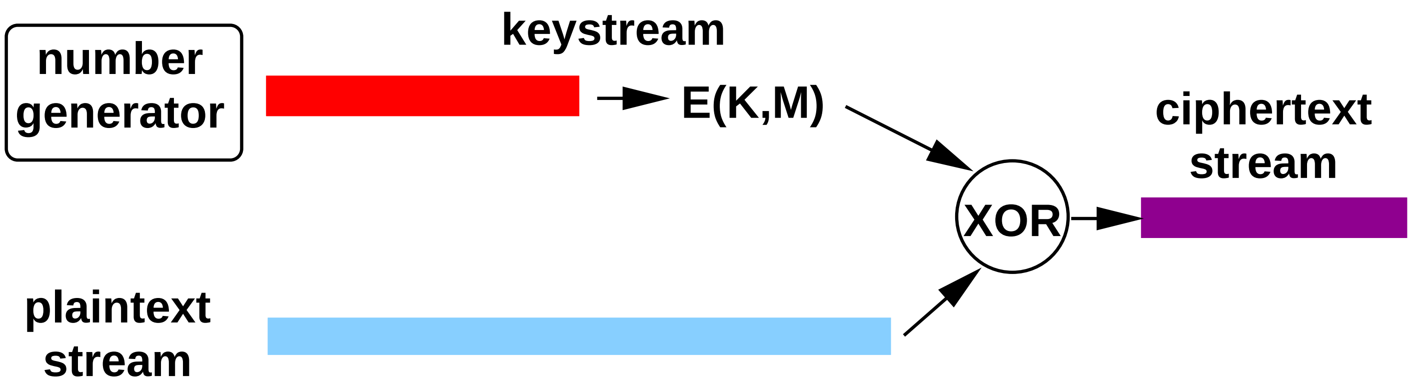
- **Trap-door functions:** one-way functions with a secret exit
- Easy to compute in one direction, but infeasible to invert unless a secret (secret key) is known
- Key pair is usually derived from a common root (such as large prime numbers) such that it is infeasible to reconstruct the root from the public key

Block ciphers

- ➜ Encrypt fixed-size blocks of data (e.g., 64 bits), one at a time
- ➜ Requires some padding in the last block why is this a weakness?
- ➜ Blocks of ciphertext are independent
  - Attacker may spot repeating patterns and infer relationship to plaintext how?
- ➜ Cipher block chaining



STREAM CIPHER



- ➜ Encode a given plaintext bit by bit (e.g., voice)
- ➜ Xor a keystream (sequence of 'random' bits) with the plaintext
- ➜ Keystream: Output of a random number generator encoded with a block cipher algorithm
- ➜ How does the receiver reconstruct the plaintext?
  - Generate the same keystream and xor it with the ciphertext
  - requires starting value of RNG and the secret key
- ➜ Under which conditions can partial message loss be tolerated?

Note: This is not the same as a One Time Pad

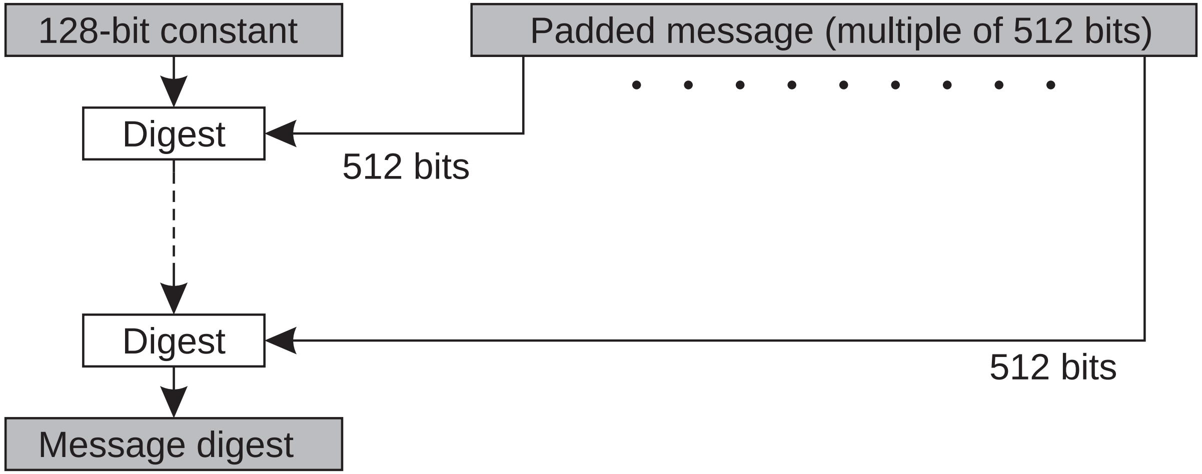
Secure Hash (Digest)

Cryptographically ensure message integrity and authenticate originator.

How can we check whether a message has been altered?

- ➜ Secure digest or hash
- ➜ Fixed-length value condensing information in the message
- ➜ Given message M and hash $H(M)$, it must be very hard to find M' with $H(M) = H(M')$
- ➜ If hash $H(M)$ is the same after transmission, message is unaltered with very high likelihood

Hash functions:



➜ Not unlike encryption functions, but not information preserving

➜ Most widely used algorithms: MD5 and SHA

➜ Rivest's MD5 algorithm: 128-bit digest; more efficient than SHA. But considered broken.

➜ SHA is standardised, more secure. Current SHA-2, SHA-3.

➜ Any symmetric encryption algorithm could be used as hashing function with cipher block chaining, but

- less efficient and
- requires use of a key

Must be resilient to:

- ➜ Collision:
  - find m_1 and m_2 such that $H(m_1) = H(m_2)$
  - related to birthday attack
- ➜ Pre-image:
  - given h, find m such that $H(m) = h$
- ➜ Second pre-image:
  - given m_1 find m_2 such that $H(m_1) = H(m_2)$

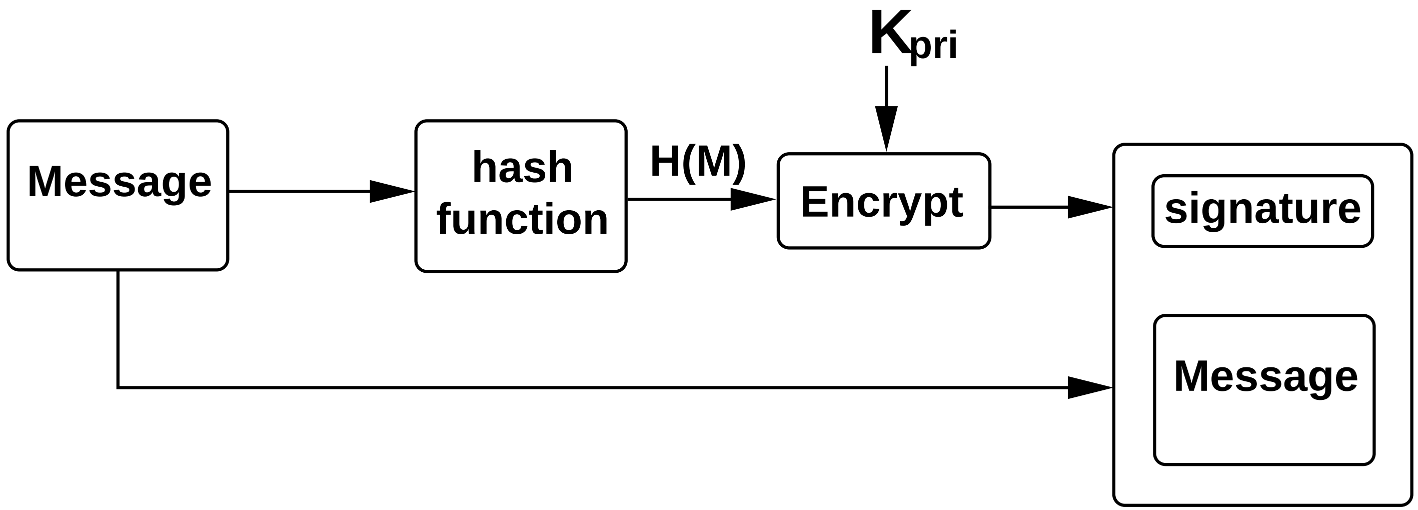
Does a hash provide:

- ➜ confidentiality?
- ➜ integrity?
- ➜ authenticity?
- ➜ non-repudiation?

Digital Signature

➜ How to verify who sent the message

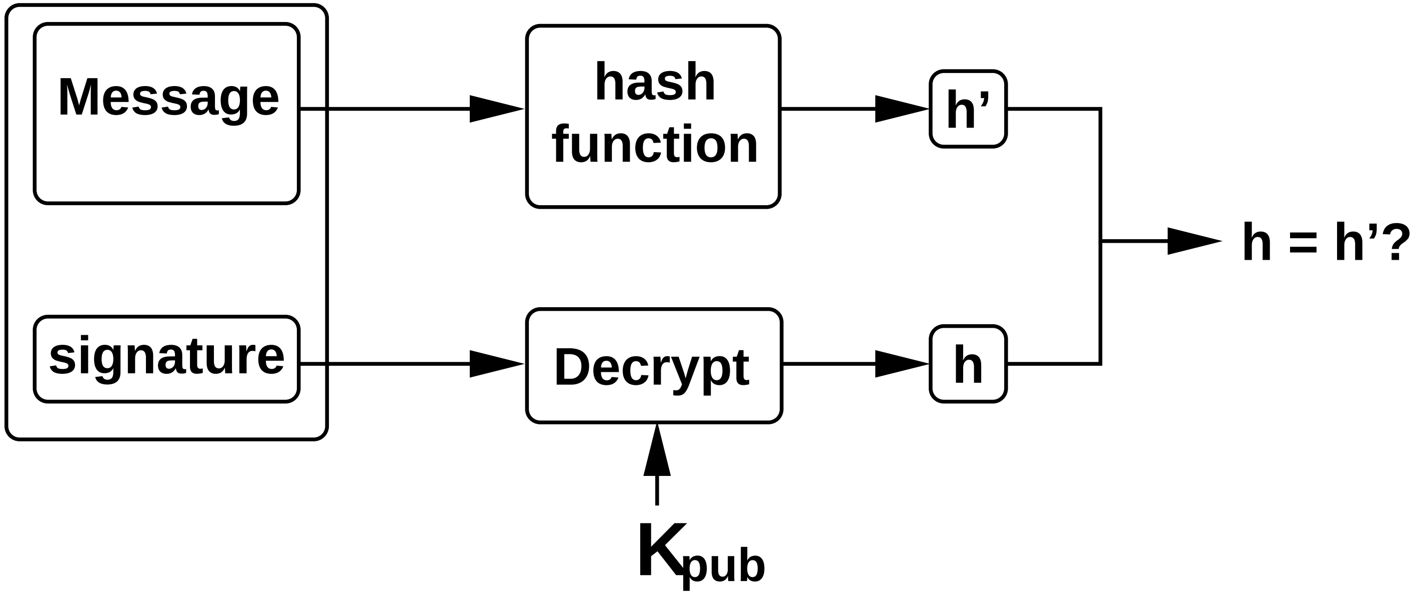
Sender:



➜ Given a message M and sender private key K_{pri}, signed message:

$$(M, \{H(M)\}_{K_{\text{pri}}})$$

Receiver:



- ➜ Recipient uses matching public key K_{pub} to recover digest
- ➜ Compare recovered digest to result of computing $H(M)$
- ➜ If same, sent message must be unaltered and sender the owner of K_{pri}

Secure Protocols

Protocol: rules governing communication

Security protocol: protocol that performs a security-related function (usually authentication)

Goal: Survive malicious attacks:

- ➜ Lies
- ➜ Modifying data
- ➜ Injecting data
- ➜ Malicious behaviour

Threat Assumptions:

- ➜ Can communication channel be intercepted?
- ➜ Can data stream be modified?
- ➜ Are participants malicious?

How to Build a Cryptographic Protocol

Use:

- ➜ encryption
- ➜ secure digest
- ➜ signatures
- ➜ random number generators

Protocol mechanisms:

- ➜ Challenge-Response
  - nonce – used to uniquely relate two messages together
    What properties should a nonce have?
- ➜ Ticket – secured information to be passed to another party Why is this useful?
- ➜ Session keys – for secure communication Why is this useful?

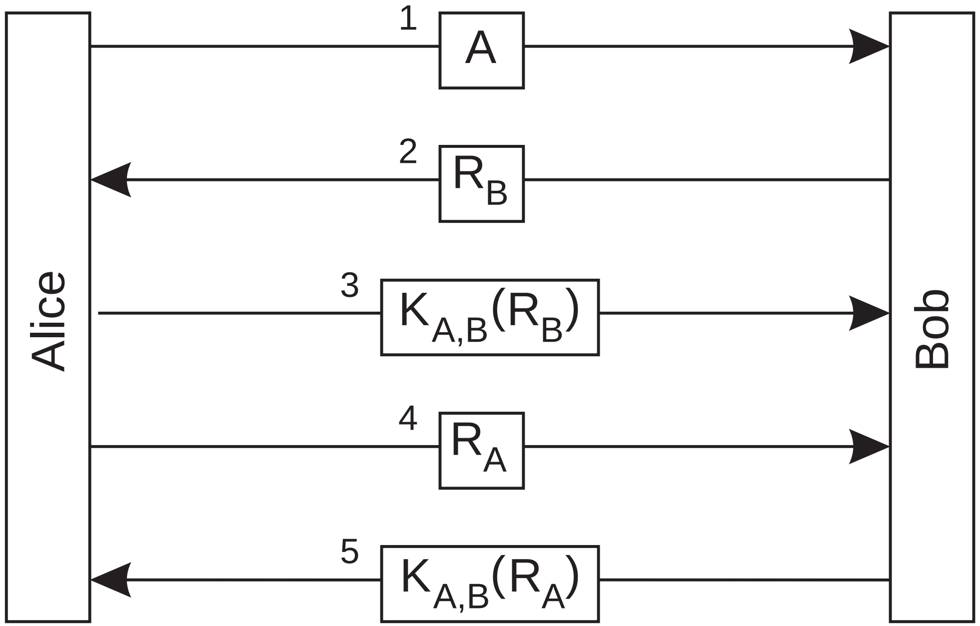
Principles:

- ➜ A message must contain all relevant information
- ➜ Don't allow parties to do things identically
- ➜ Don't give away valuable information to strangers

A SIMPLE PROTOCOL

Authentication

➜ Alice knows it's Bob, Bob knows it's Alice



HOW TO BREAK A PROTOCOL

Man-in-the-Middle:

- ➜ Take on the role of Alice to Bob and Bob to Alice
- ➜ Alice $\rightarrow$ Eve: challenge
- ➜ Eve $\rightarrow$ Bob: challenge
- ➜ Eve $\leftarrow$ Bob: response
- ➜ Alice $\leftarrow$ Eve: response

Reflection:

- Use Alice to respond to Alice's challenge
- Alice $\rightarrow$ Eve: challenge
- Alice $\leftarrow$ Eve: challenge
- Alice $\rightarrow$ Eve: response
- Alice $\leftarrow$ Eve: response

Replay:

- Re-use Bob's old message to respond to Alice's challenge
- Alice $\rightarrow$ Bob: challenge
- Alice $\leftarrow$ Eve $\leftarrow$ Bob: response
- Alice $\rightarrow$ Eve: challenge
- Alice $\leftarrow$ Eve: response

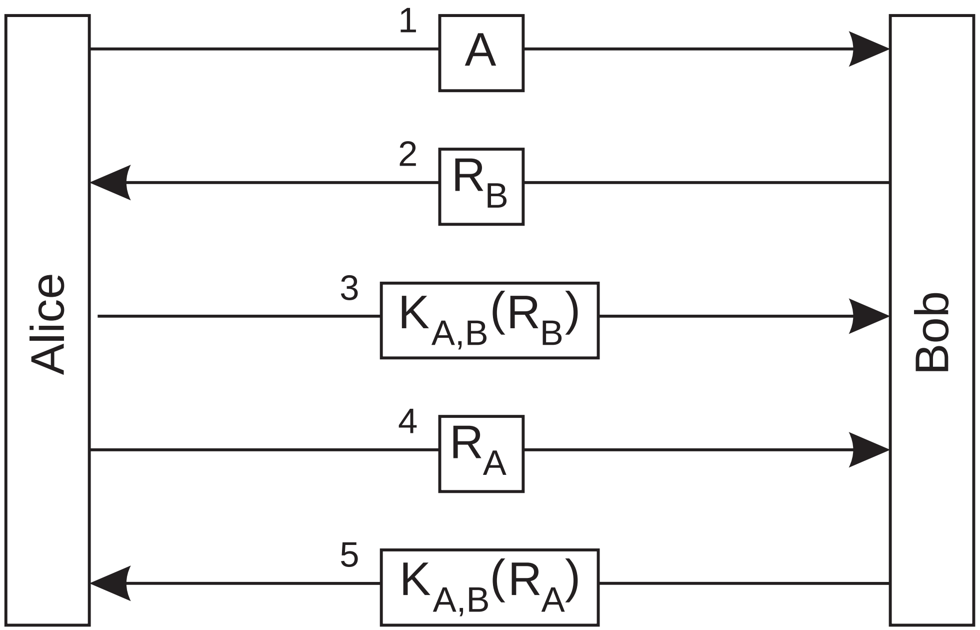
Message Manipulation:

- ➜ Change the message from Alice to Bob
- ➜ Alice sends: let's meet at 3pm by the bridge
- ➜ Eve intercepts and changes
- ➜ Bob receives: let's meet at 2pm by the oak

Changed Environment/Assumptions:

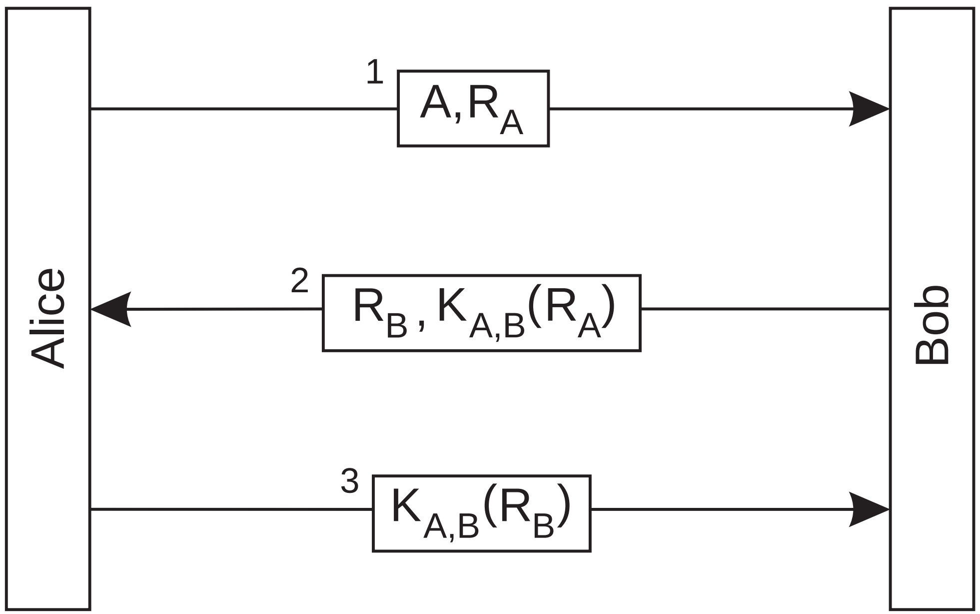
- ➜ Bob is no longer trustworthy
- ➜ Bob sells Alice's secrets to the tabloid press!

Authentication



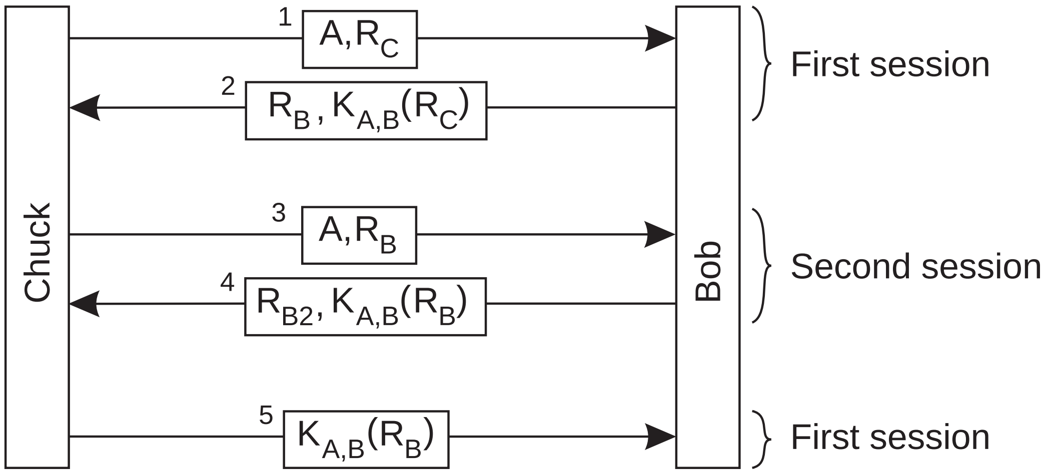
Vulnerable?

Optimising the Protocol



Oops!

➜ Vulnerable to reflection attack



Is this different from Man-in-the-middle?

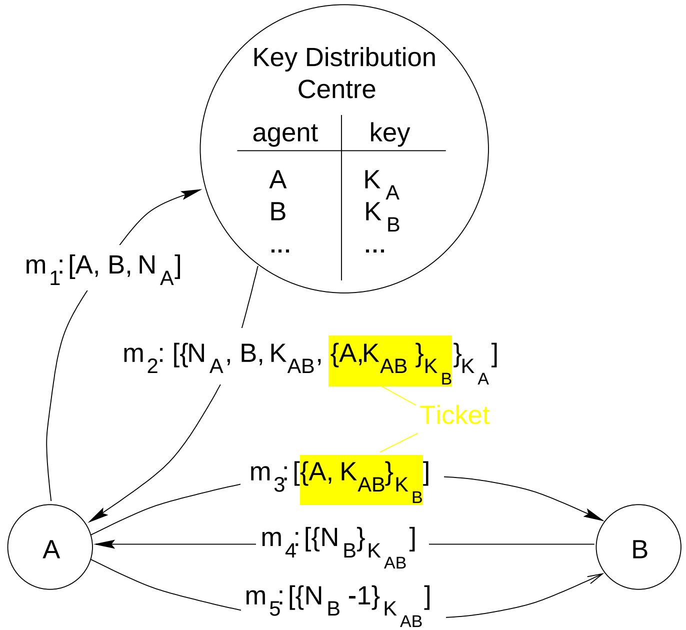
KEY DISTRIBUTION

A set of keys provides a secure channel for communication.

How does the secure channel get established in the first place?

- ➜ Use separate channel to establish keys
- ➜ Use key distribution protocols
- ➜ Protocols vary depending on whether symmetric or asymmetric encryption is used
- ➜ Often symmetric keys are communicated over a channel using an asymmetric cipher

Distribution of Symmetric Keys (Needham-Schroeder)



- ➜ Central key distribution centre D
- ➜ Each agent A shares a (symmetric) key K_A with D
- ➜ A wants to communicate with B, asks D for session key K_{AB}
- ➜ After key distribution protocol, both A and B know that they share a key provided by D.

Properties of the symmetric key distribution protocol:

- ➜ Ticket and challenge implicitly authenticate A and B.
- ➜ Nonce and challenge protect against replay attacks.
- ➜ D is centralised resource (hierarchical scheme possible).
- ➜ Every agent must trust D.
- ➜ D maintains highly sensitive information (secret keys), compromising D compromises all communication.
- ➜ Large number of keys required (one per pair of agents), manufactured by D on-the-fly.
- ➜ D must take care to make key sequence non-predictable.

Any vulnerabilities?

SECURE COMMUNICATION

Properties of a Secure Channel:

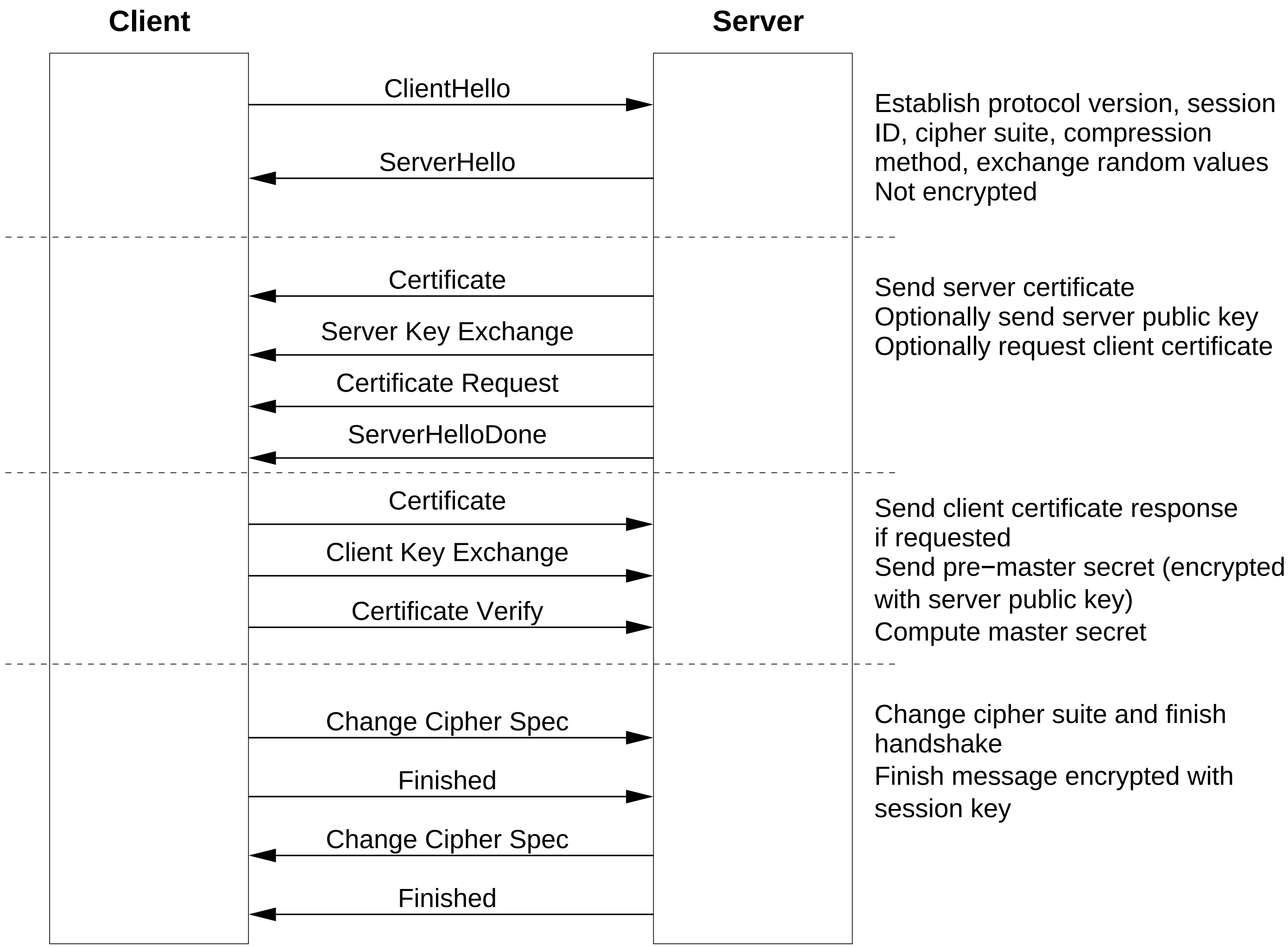
- ➜ Authentication
- ➜ Message confidentiality
- ➜ Message integrity

EXAMPLE: SSL (AND TLS)

Secure Socket Layer:

- ➜ Application level protocol for secure channel
- ➜ Handshake protocol: establish and maintain session
  - ➜ Authentication
- ➜ Record protocol: secure channel
  - ➜ Confidentiality, Integrity
- ➜ Flexible: can choose ciphers to use
- ➜ Most widely used to secure HTTP (https: URLs)
- ➜ TLS (Transport Layer security): IETF standard based on SSL 3.0
- ➜ TLS 1.0: RFC 2246, TLS 1.2: RFC RFC 5246, TLS 1.3 proposed standard

SSL Handshake Protocol:



SSL Record Protocol:

Application Data

Fragment/combine

Record protocol units

Compress

Compressed units

Hash

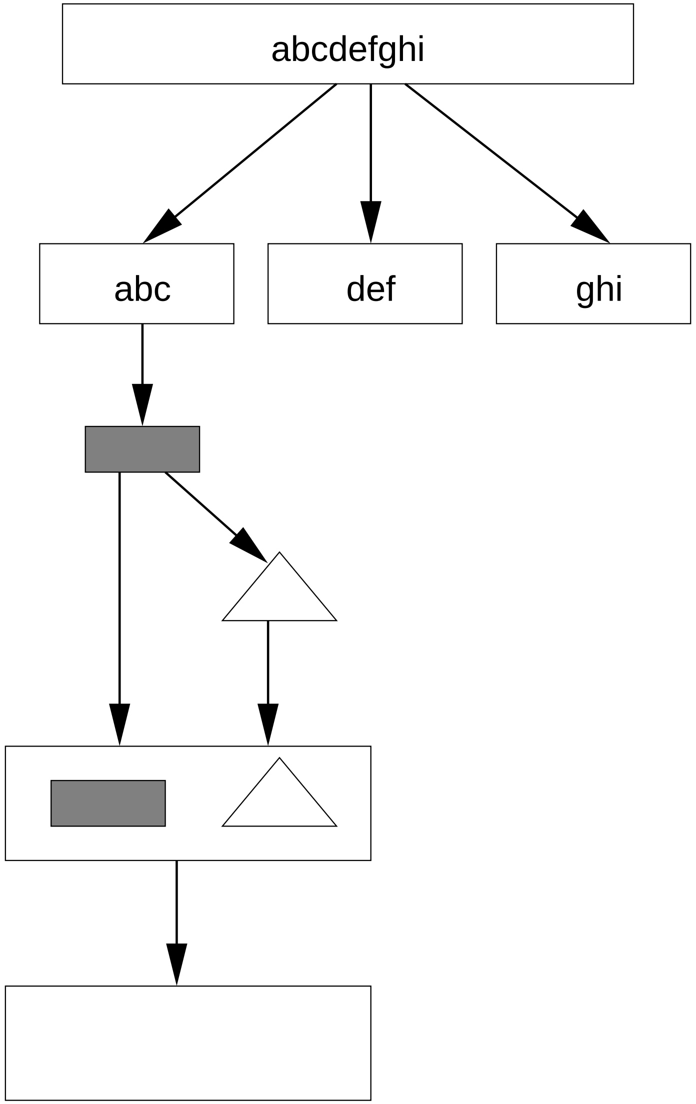
Message Authentication Code

Encrypt

Encrypted

Transmit

TCP packet



SECURE GROUP COMMUNICATION

Two types:

Confidential group communication:

- ➜ All group members share the same secret key
- ✗ Need to trust all members
- ➜ Separate keys for each pair
- ✗ Scalability problem
- ➜ Public key cryptography
- ✗ Everyone knows each others keys

Secure replicated servers:

- ➜ Secure Replicated Servers: protecting from malicious group members
- ➜ Collect responses from all servers and authenticate each
- ✗ Not transparent
- ➜ Secret sharing:
  - ➜ All group members know part of a secret.
  - ➜ Recipient combines answers from k members, decrypts with special decryption function D.
  - ➜ If successful: these k members are honest.
  - ➜ If not: try other combination of answers.

AUTHENTICATION

Verify the claimed identity of an entity (principal)

Authentication Requires:

➜ Representation of identity
  - Unix user id, email address, student number, bank account

➜ Some way to verify the identity
  - Password, reply to email, student card, PIN

➜ Different levels of authentication

Credentials:

➜ *Speaks for* a principal

➜ Example: certificate stating identity of a principal

➜ Combine credentials

➜ Role-based credentials

Approaches to Authentication:

Password: provide some secret information

Shared secret key: challenge and response encoded with shared secret key

Key distribution centre: keys stored at KDC, never sent over network

Public key: exchange session key encoded with public keys

Hybrid: use public keys to set up a secure channel and then authenticate

KERBEROS

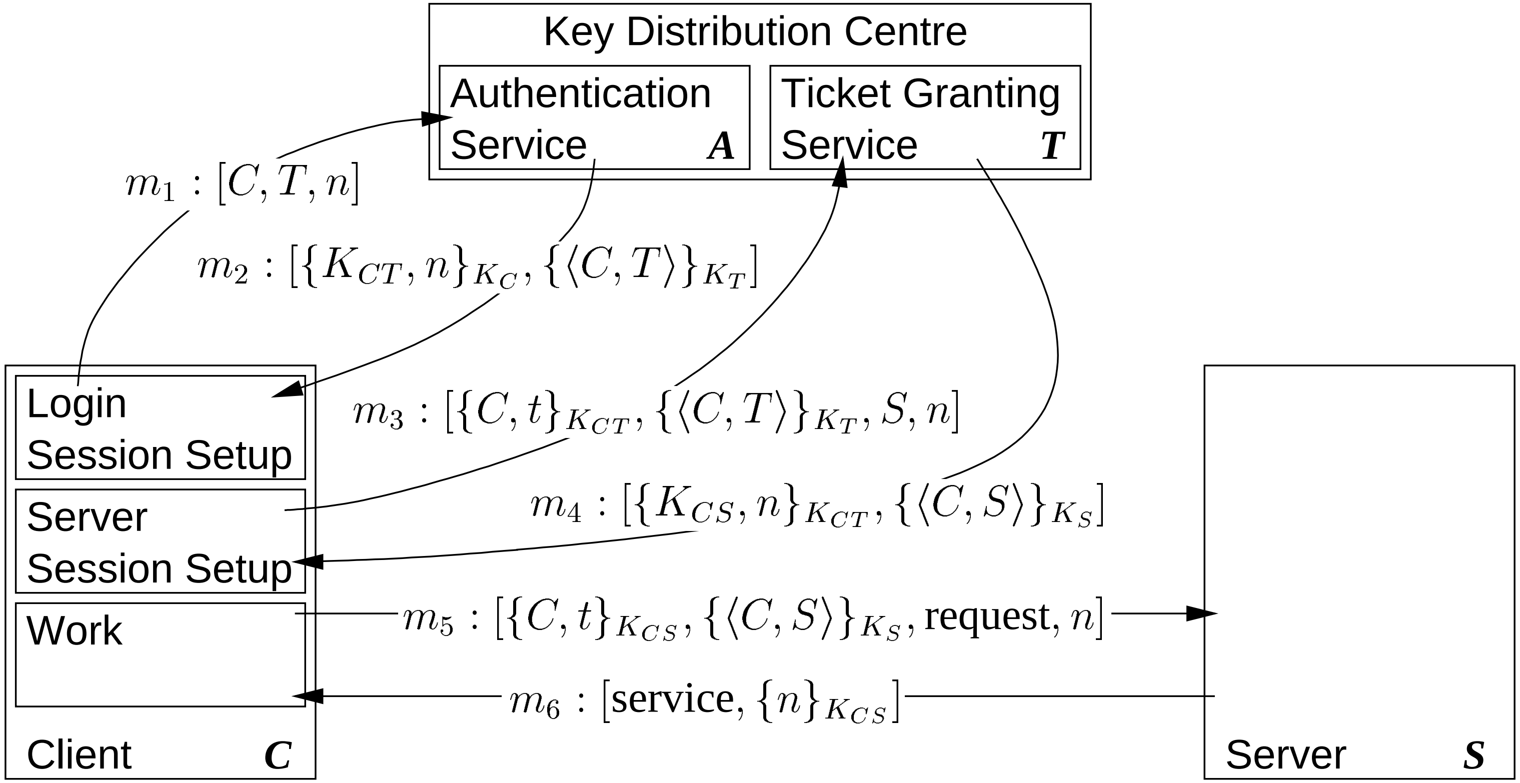
- ➜ Commercial authentication system developed at MIT
- ➜ Based on Needham and Schroeder protocol
- ➜ Integrates symmetric key encryption, distribution and authentication into commercial computer systems.
- ➜ Assumptions:
  - secure central server
  - insecure network

    → never transmit cleartext passwords
  - insecure workstations (shared between users)

    → hold user passwords on workstations for very short periods only

    → hold no system keys on workstations

Kerberos Authentication:



- ➜ Central KDC contains
  - Authentication service A,
    knows all user logins and their passwords (secret keys)
    as well as identity and key of T;
  - Ticket granting service T,
    knows all servers and their secret keys
- ➜ Kerberos protocol has three phases:
  ① login session setup (user authentication)
  ② server session setup (establishing secure channel to server)
  ③ client-server RPC
- ➜ Uses time-limited tickets

Distribution of Public Keys

Major weakness of Needham-Schroeder and Kerberos:

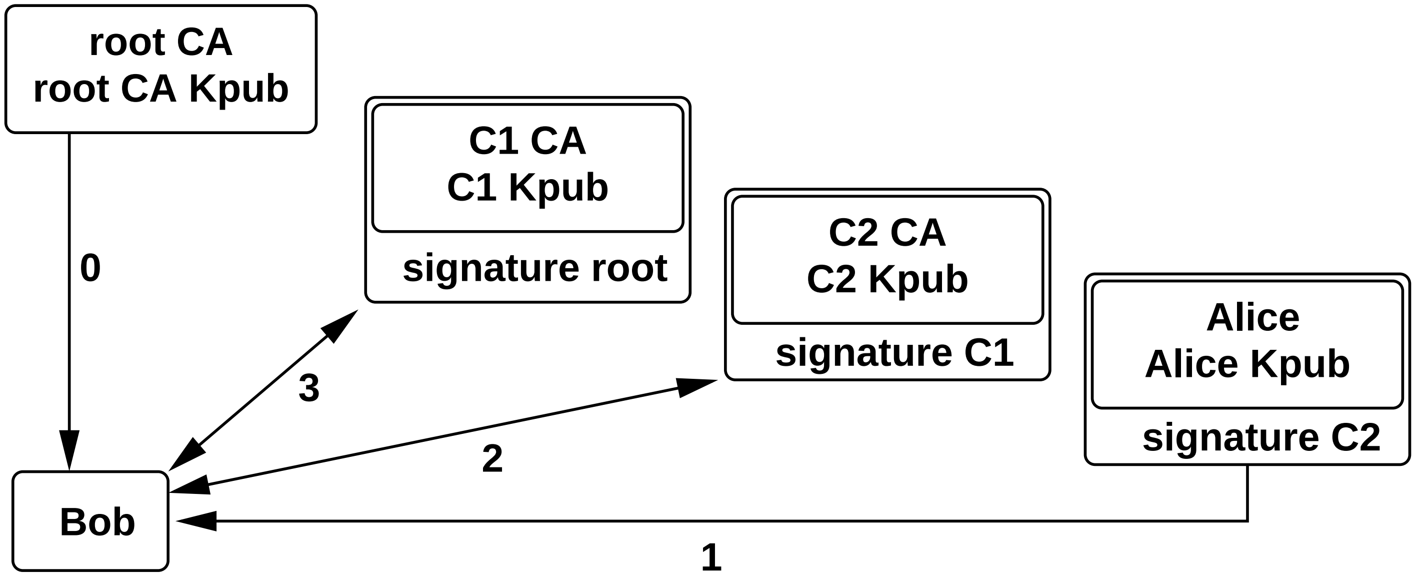
- ➜ Key distribution centre as a central authority
- ➜ Compromised keys can be used to decrypt past communication

Public Key Infrastructure (PKI):

- ➜ Public keys can be exposed without risk
- ➜ Distribution centre only establishes link between identities and public keys

Certificates and certification authorities:

- ➜ A certificate links an identify with a public key
- ➜ Distribution centres are called certificate servers or certificate directories



Checking of certificates is recursive:

- ➜ To establish trust in Alice's certificate signed by C_2, Bob may need to obtain C_2's certificate
- ➜ Bob uses the public key of C_2 to validate Alice's certificate
- ➜ C_2 is signed by C_1
- ➜ This may lead to a chain of certificates
- ➜ Terminated by self-signed certificate of a root certification authority (who Bob trusts)

How to communicate certificates to clients?

- ➜ Secure channel between certificates server and client?
- ➜ Digital signatures establish the validity of certificates
- ➜ Formatted according to X509.1 standard or PGP format

Whose signature?

- ➜ Certification authorities sell certification as a service
- ➜ Alternatively, web of trust avoids any central authority

Are certificates valid forever?

- ➜ Certificates may have an expiry date to reduce risk of security breach
- ➜ After a certificate expires, a new one must be generated and signed
- ➜ Alternatively, certificates may be revoked
- ➜ Revocation is only effective if receiver regularly checks the certificate server

AUTHORISATION AND ACCESS CONTROL

Determine what actions an authenticated entity is authorised to perform

Access Rights:

➜ The rights required to access (perform an operation on) a given resource

Two aspects:

Access Control: verify access rights

Authorisation: grant access rights

Ensuring that authorisation and access control are respected

Non-distributed Protection:

- ➜ Global mechanisms
- ➜ Global policies
- ➜ Examples:
  - Users
  - File permissions
  - Separate address spaces

Distributed Protection:

- ➜ Service specific
  - Web servers and .htaccess: authentication, access control
- ➜ Application specific

Access Control Matrix

	Objects			
Subjects	O_1	O_2	O_3	O_4
S_1	terminate	wait, signal, send	read	
S_2	wait, signal, terminate			read, execute write, control
S_3		wait, signal, receive		
S_4	control		execute	write

- ➜ Access permissions of a given subject to a given object
- ➜ Specifies allowed operations

Properties of the access matrix:

- ➜ Rows define subjects' protection domains
- ➜ Columns define objects' accessibility
- ➜ Dynamic data structure: frequently changes
  - permanent changes (e.g. `chmod`)
  - temporary changes (e.g. setuid flag)
- ➜ Matrix is very sparse with many repeated entries
  - ➽ usually not stored explicitly

Design considerations in a protection system:

- Propagation of rights:
  - Can someone act as an agent's proxy?
- Restriction of rights:
  - Can an agent propagate a subset of their rights?
- Amplification of rights:
  - Can an unprivileged agent perform some privileged operations?
- Revocation of rights:
  - Can a right, once granted, be remove from an agent?
- Determination of object accessibility
  - Who has which rights on an object?
- Determination of agent's protection domain
  - What is the set of objects an agent can access?

Access control lists (ACLs):

Object	*Subjects*			
	S_1	S_2	S_3	S_4
`/etc/passwd`	read	read, write	–	read

- Column-wise representation of the access matrix
- Each object associated with a list of (subject, rights) pairs
  - requires explicit authentication
- Usually supports concept of group rights (domain classes) (granted to each agent belonging to the group)
- Often simplified to a simple fixed-size list (e.g., UNIX *user-group-others* or VMS *system-owner-group-world*)
- Can have negative rights as well (e.g., to simplify exclusion from groups)

Properties of ACLs:

- ➜ Propagation: meta-right to change ACL (e.g., owner can `chmod`)
- ➜ Restriction: meta-right to change ACL
- ➜ Amplification: (e.g., setuid)
- ➜ Revocation: remove from ACL
- ➜ Object accessibility: explicit in ACL
- ➜ Protection domain: hard (if not impossible)

Capabilities:

- ➜ An element of access matrix
- ➜ Capabilities list (C-list) associated with each subject, which defines a protection domain
- ➜ Each capability can confer a single or a set of rights
- ➜ Capabilities can confer negative rights
- ➜ Capabilities must be protected against forgery and theft
- ➜ Capability used as an object name:
  - evidence of access permission
  - independent of authentication
  - don't need to trust intermediary

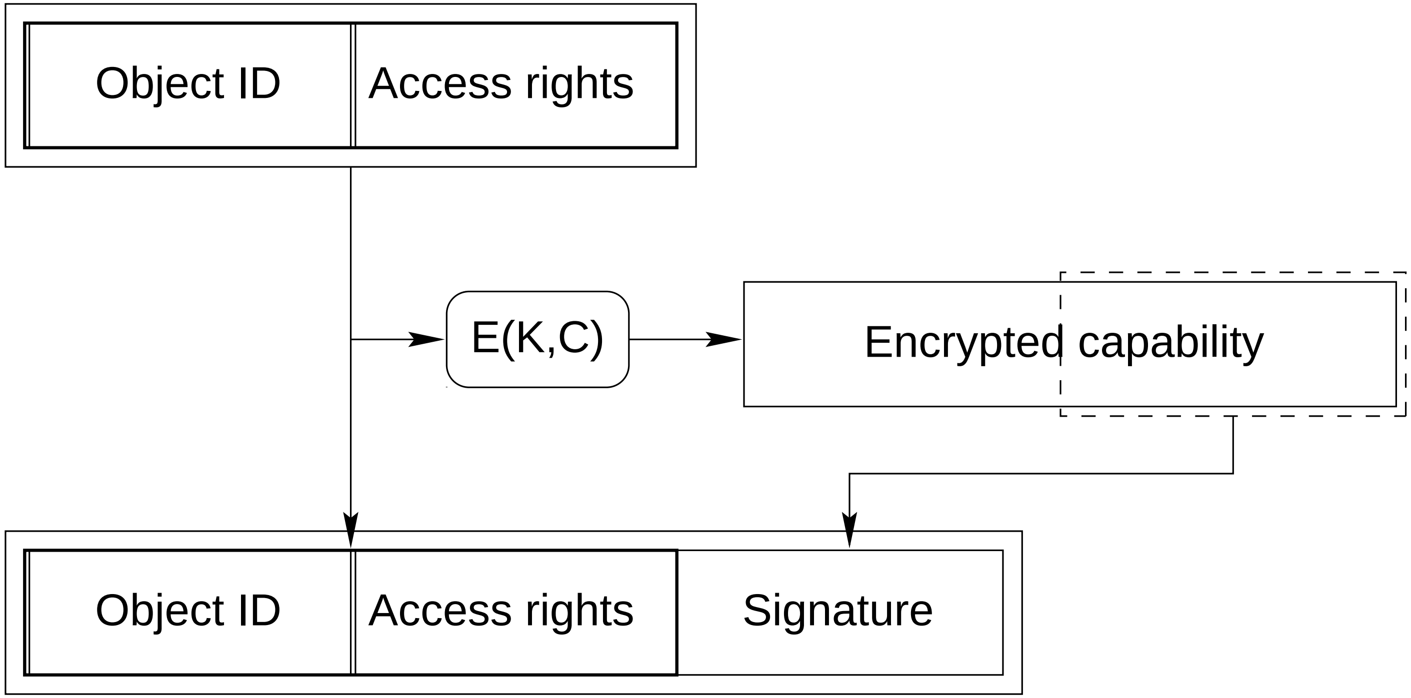
Properties of capabilities:

- Propagation: copy capability (but need to be careful about confinement)
- Restriction: may be supported by derived capabilities
- Amplification: may have amplification capabilities
- Revocation: difficult, requires invalidation
- Object accessibility: hard (if not impossible)
- Protection domain: explicit in C-list

Three basic approaches to making caps tamper-proof:

- ➜ Tagged capabilities:
  - protected by hardware (tag bit)
  - controlled by OS (only kernel can turn on tag bit)
  - used in most historical capability systems (Plessey 250, CAP, Hydra, System/38)
- ➜ Partitioned (segregated) capabilities:
  - protected by OS: Capabilities kept in kernel space
  - used in Mach, Grasshopper, EROS, seL4
- ➜ Sparse capabilities:
  - protected by sparseness (obscurity)
  - used in Monash Password Capability System, Amoeba, Mungi

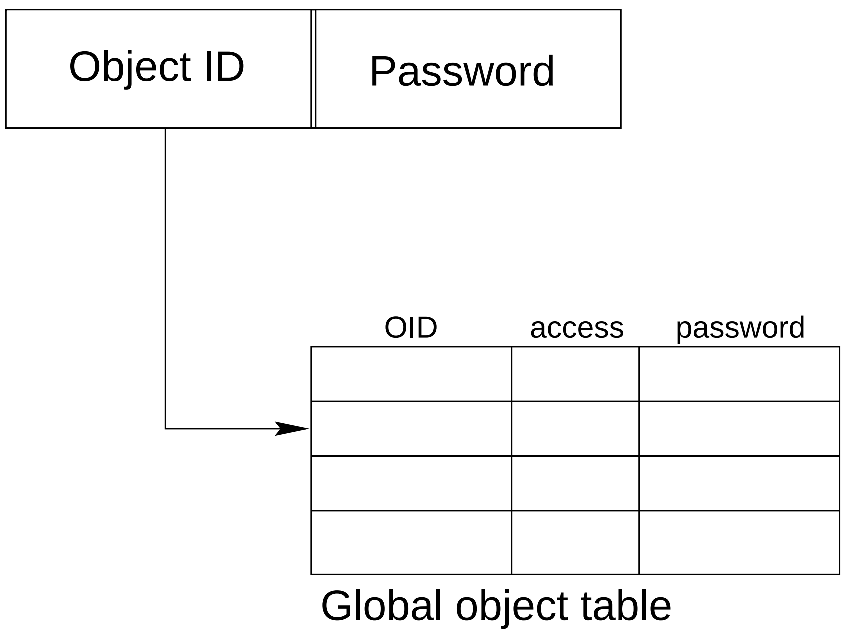
Signature capabilities:



- ☑ tamper proof via encryption with secret kernel key
- ☑ can be freely passed around
- ☒ need to encrypt on each validation

Password capabilities:

- ➜ Invented for Monash U's Password Capability System
- ➜ "Random" bitstring is password, not derived from other parts of capability.
- ➜ Validation requires checking against global object table.



FIREWALLS

Properties:

- ➜ When communicating with untrusted clients/servers `
- ➜ Disconnects part of system from outside world
- ➜ Incoming communication inspected and filtered

Two types:

- ➜ Packet-filtering gateway
- ➜ Application-level gateway

Three Myths of Firewalls:

① We've got the place surrounded

② Nobody here but us chickens

③ Sticks and Stones may break my bones, but words will never hurt me

How to Break Security?

Encryption:

- ➜ find weaknesses in algorithms
- ➜ find weaknesses in implementations
- ➜ attack underlying intractable problem
- ➜ brute force

Protocols:

- ➜ find weakness in protocol design (try MitM, reflection attacks)
- ➜ find vulnerability in implementation

Authentication:

- ➜ find keys or passwords
- ➜ social engineering

Authorisation and Access Control:

- ➜ find problems with Access Control Matrix
- ➜ find and exploit bugs to escalate privileges

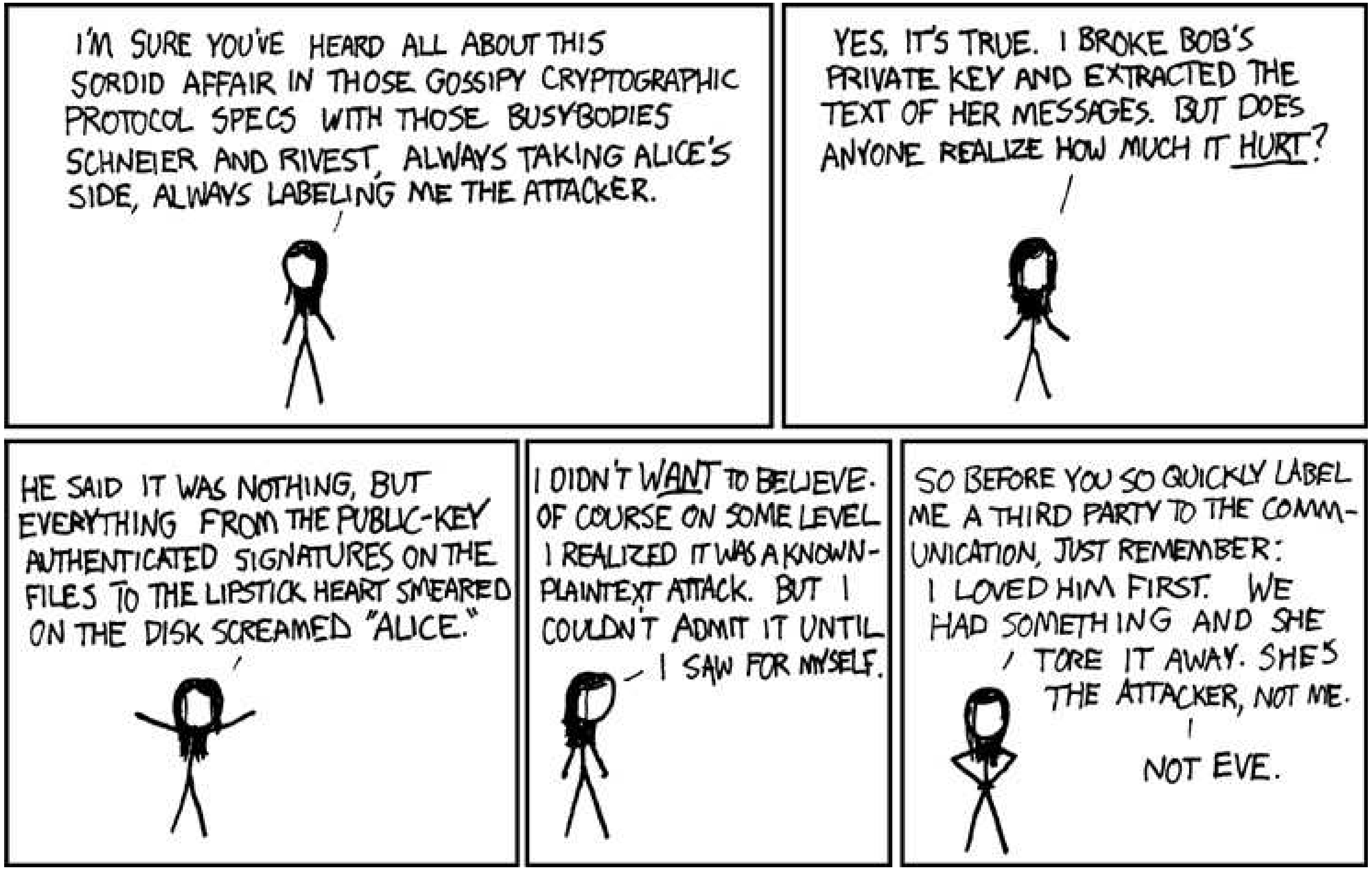
Reading list

Ross J. Anderson Security Engineering: A Guide to Building Dependable Distributed Systems. Covers many pitfalls of building secure systems, with many real-world examples.

HOMEWORK

Look up how protocols have been broken in the past. Find examples where:

- ➜ the protocol was broken
- ➜ the cryptography was broken
- ➜ the implementation was broken



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.