# Principles of Distributed Systems

inft-3507

Dr. J.Burns

**ADA University**

Lectures are based on: https://www.distributed-systems.net/index.php/books/ds4/

## Chapter 09: Security

## Dependability

### Basics

A dependable system provides availability, reliability, safety, maintainability, confidentiality, and integrity.

- **Confidentiality**: refers to the property that information is disclosed only to authorized parties.

- **Integrity**: alterations to a system's assets can be made only in an authorized way, ensuring accuracy and completeness.

# Dependability

### Basics

A dependable system provides availability, reliability, safety, maintainability, confidentiality, and integrity.

- Confidentiality: refers to the property that information is disclosed only to authorized parties.

- Integrity: alterations to a system's assets can be made only in an authorized way, ensuring accuracy and completeness.

### Alternative

We attempt to protect against security threats:

1. Unauthorized information disclosure (confidentiality)

2. Unauthorized information modification (integrity)
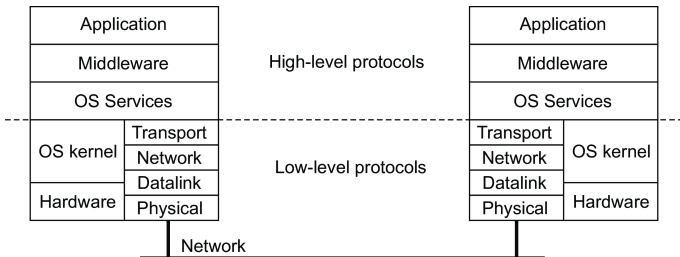
3. Unauthorized denial of use (availability)

## Security mechanisms

- Encryption: transform data to something an attacker cannot understand, or that can be checked for modificatons.

- Authentication: verify a claimed identity.

- Authorization: check an authenticated entity whether it has the proper rights to access resources.

- Monitoring and auditing: (continuously) trace access to resources

## Security principles

- Fail-safe defaults: defaults should already provide good protection.
  Infamous example: the default "*admin,admin*" for edge devices.

- Open design: do not apply security by obscurity: every aspect of a
  distributed system is open for review.

- Separation of privilege: ensure that critical aspects of a system can never
  be fully controlled by just a single entity.

- Least privilege: a process should operate with the fewest possible privileges.

- Least common mechanism: if multiple components require the same
  mechanism, then they should all be offered the same implementation of
  that mechanism.

# Where to implement security mechanisms?



| Application | | | High-level protocols | Application | | |
| Middleware | | | | Middleware | | |
| OS Services | | | | OS Services | | |
| OS kernel | Transport | | Low-level protocols | Transport | OS kernel |
| | Network | | | Network | |
| | Datalink | | | Datalink | |
| Hardware | Physical | | | Physical | Hardware |
| | Network | | | | |

### Observation

We are increasingly seeing end-to-end security, meaning that mechanisms are implemented at the level of applications.

# Where to implement security mechanisms?
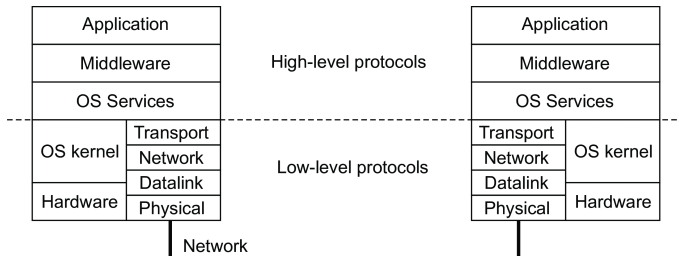


## Observation

We are increasingly seeing end-to-end security, meaning that mechanisms are implemented at the level of applications.

## Issue: which layer do we trust?

Trusted Computing Base: The set of all security mechanisms in a (distributed) computer system that are necessary and sufficient to enforce a security policy.

# On privacy

### Observation

Privacy and confidentiality are closely related, yet are different. Privacy can be invaded, whereas confidentiality can be breached $\Rightarrow$ ensuring confidentiality is not enough to guarantee privacy.

# On privacy

### Observation

Privacy and confidentiality are closely related, yet are different. Privacy can be invaded, whereas confidentiality can be breached $\Rightarrow$ ensuring confidentiality is not enough to guarantee privacy.

### Right to privacy

The right to privacy is about "a right to appropriate flow of personal information." Control who gets to see what, when, and how $\Rightarrow$ a person should be able to stop and revoke a flow of personal information.

# On privacy

### Observation

Privacy and confidentiality are closely related, yet are different. Privacy can be invaded, whereas confidentiality can be breached $\Rightarrow$ ensuring confidentiality is not enough to guarantee privacy.

### Right to privacy

The right to privacy is about "a right to appropriate flow of personal information." Control who gets to see what, when, and how $\Rightarrow$ a person should be able to stop and revoke a flow of personal information.

### General Data Protection Regulation (GDPR)

The GDPR is a comprehensive set of regulations aiming to protect personal data.

## GDPR: Database perspective

| GDPR regulation | Impact on database systems | |
|---|---|---|
| | **Attributes** | **Actions** |
| Collect data for explicit purposes | Purpose | Metadata indexing |
| Do not store data indefinitely | TTL | Timely deletion |
| Inform customers about GDPR metadata associated with their data | Purpose, TTL, Origin, Sharing | Metadata indexing |
| Allow customers to access their data | Person id | Metadata indexing |
| Allow customers to erase their data | TTL | Timely deletion |
| Do not use data for objected reasons | Objections | Metadata indexing |
| Allow customers to withdraw from algorithmic decision-making | Automated decisions | Metadata indexing |
| Safeguard and restrict access to data | | Access control |
| Do not grant unlimited access to data | | Access control |
| Audit operations on personal data | Audit trail | Monitor and log |
| Implement appropriate data security | | Encryption |
| Share audit trails from affected systems | Audit trail | Monitor and log |

# Cryptography
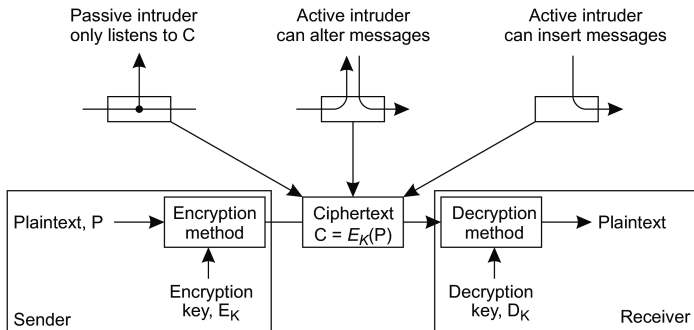


## Basic concepts

- Plaintext: the original message or data $(P)$

- Ciphertext: the encrypted version of the the plaintext $(C)$

- Encryption key: input $E_K$ to a function for encryption: $C = E_K(P)$

- Decryption key: input $D_K$ to a function for decryption: $P = D_K(C)$

## Cryptosystems

Symmetric : *if $P = D_K(E_K(P))$ then $D_K = E_K$.*

Asymmetric : *if $P = D_K(E_K(P))$ then $D_K \neq E_K$.*
Also called public-key systems with a publicly known key $PK$ and secret key $SK$

### Examples

Let $PK_X$ denote public key of $X$ and $SK_X$ the associated secret key.

Confidential message : *if m is to be kept private: $C = PK_{receiver}(m)$.*

Authenticated message : *if m is to be authenticated: $C = SK_{sender}(m)$.*

### Homomorphic encryption

Mathematical operations on plaintext can be performed on the corresponding ciphertext: if $x$ and $y$ are two numbers, then

$$E_K(x) \star E_K(y) = E_K(x \star y)$$

## Hash functions

### Description

A hash function $H$ takes a message $m$ of arbitrary length as input and produces a bit string $h$ having a fixed length as output:

$$h = H(m) \text{ with length of } h \text{ fixed}.$$

### Example: digital signature

Alice computes a digest from $m$; encrypts the digest with her private key; encrypted digest is sent along with $m$ to Bob:

$$\text{Alice: } send\ [m, sig]\ with\ sig = SK_A(H(m)).$$

Bob decrypts digest with Alice's public key; separately calculates the message digest. If both match, Bob knows the message has been signed by Alice:

$$\text{Bob: } receive\ [m, sig],\ compute\ h' = H(m)\ and\ verify\ h' = PK_A(sig).$$

# Key management

## Essence

How do Alice and Bob get the correct (often shared) keys so that they can set up secure channels?

## Diffie-Hellman key exchange

Assume two large, nonsecret numbers $p$ and $g$ (with specific mathematical properties):
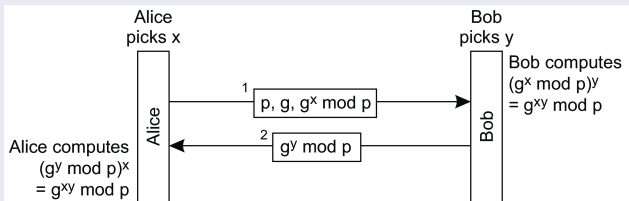
# DH key exchange: example

### Multiparty computation

Can we protect private data while computing statistics? Who has the highest salary without revealing salaries? Can we compute the number of votes cast for a specific candidate without revealing who voted for whom?

## DH key exchange: example

### Multiparty computation

Can we protect private data while computing statistics? Who has the highest salary without revealing salaries? Can we compute the number of votes cast for a specific candidate without revealing who voted for whom?

### Oblivious transfer

Alice has $n$ secret messages $m_1, \ldots, m_n$. Bob is interested (and allowed) to know only message $m_i$. Which message he wants to know should be kept secret to Alice; all messages $m_j \neq m_i$ should be kept secret to Bob.

# DH key exchange: example

### Multiparty computation

Can we protect private data while computing statistics? Who has the highest salary without revealing salaries? Can we compute the number of votes cast for a specific candidate without revealing who voted for whom?

### Oblivious transfer

Alice has $n$ secret messages $m_1, \ldots, m_n$. Bob is interested (and allowed) to know only message $m_i$. Which message he wants to know should be kept secret to Alice; all messages $m_j \neq m_i$ should be kept secret to Bob.

### Solution

Bob generates a number $Q$ that Alice, in turn, uses to generate $n$ different encryption keys $PK_1, \ldots, PK_n$: $m_i^* = PK_i(m_i)$
Bob uses $Q$ to generate a decryption key $SK_i$ that matches only $PK_i$. When Bob receives $m_1^*, \ldots, m_n^*$ he can decrypt only $m_i^*$. $SK_i(m_j^*)$ (with $i \neq j$) will fail.

# 1-out-of-2 oblivious transfer
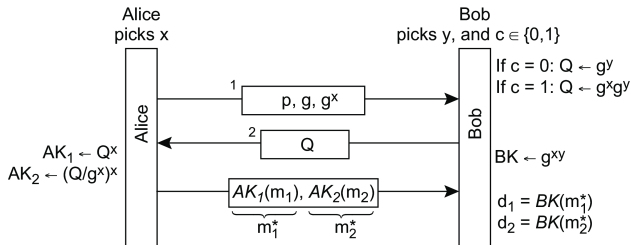


### Analysis

- $c = 0 \Rightarrow Q = g^y, AK_1 = BK = g^{xy}, AK_2 = g^{xy-x^2}$.
- $c = 1 \Rightarrow Q = g^{x+y}, AK_1 = g^{x^2+xy}, AK_2 = BK = g^{xy}$.
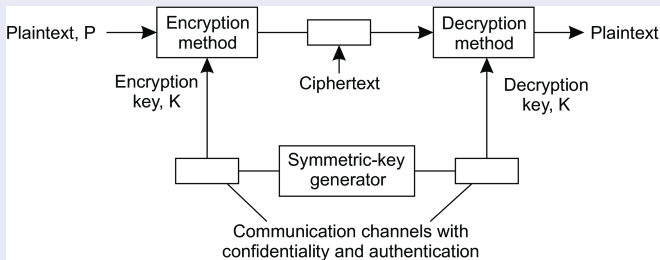
## Example, continued

### Preliminaries

- $P_1$ and $P_2$ need to compute $F(a, b)$.

- Parameter $a$ is secret and known only to $P_1$; secret $b$ known only to $P_2$.

- $a \in \mathbf{X}$ and $b \in \mathbf{Y}$; $\mathbf{X}$ and $\mathbf{Y}$ are finite.

- Construct a $|\mathbf{X}| \times |\mathbf{Y}|$ matrix $\mathbf{F}$.

- $\mathbf{F}[i, j] = F(x_i, y_j)$ for each pair $(x_i, y_j) \in \mathbf{X} \times \mathbf{Y}$.

### Solution

- $P_1$ generates $|\mathbf{X}| \cdot |\mathbf{Y}|$ unique key pairs $(K_i, K_j)$

- Construct $\mathbf{F}^*[i, j] = K_i(K_j(F(x_i, x_j)))$. Assume $a = x_i$.

- $P_1$ permutes $\mathbf{F}^*$ and sends it along with $K_i$ to $P_2$

- $P_1$ sends $Q$ using a 1-out-of-$|\mathbf{Y}|$ oblivious transfer.

- Assume $b = y_j$. Using $Q$, $P_2$ can construct $K_j$, and only $K_j$

- $P_2$ decrypts $\mathbf{F}^*[i, j]$, corresponding to $F(a, b)$.

# What is needed to distribute keys

## Symmetric-key distribution



Plaintext, P → Encryption method → Ciphertext → Decryption method → Plaintext

Encryption key, K

Decryption key, K

Symmetric-key generator

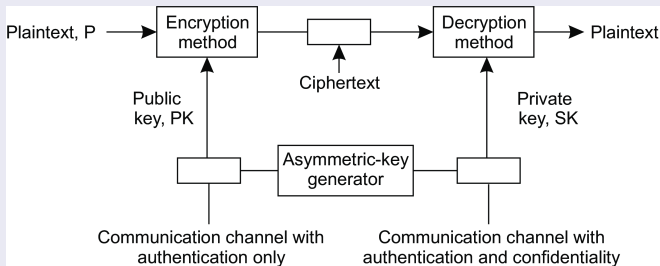Communication channels with confidentiality and authentication

## Observation

In general, we will need a secure channel to distribute the secret key to the communicating parties.

# What is needed to distribute keys

## Public-key distribution



## Observation

No need for a scure channel in the case of the public key, but you do need to know that the key is authentic $\Rightarrow$ have the public key be signed by a certification authority. Note, we do need to trust that authority, or otherwise make sure that its signature can be verified as well.

# Authentication

### Essence

Verifying the claimed identity of a person, a software component, a device, and so on.

### Means of authentication

1. Based on what a client knows, such as a password or a personal identification number.

2. Based on what a client has, such as an ID card, cell phone, or software token.

3. Based on what a client is, i.e., static biometrics such as a fingerprint or facial characteristics.

4. Based on what a client does, i.e., dynamic biometrics such as voice patterns or typing patterns.

## Authentication versus message integrity

### Observation

Authentication without integrity (and *vice versa*) is meaningles:

- Consider a system that supports authentication but no mechanisms to ensure message integrity. Bob may know for sure that Alice sent *m*, but how useful is that if he doesn't know that *m* may have been modified?

- Consider a system that guarantees message integrity, but does not provide authentication. Can Bob be happy with a guaranteed unmodified message that states he just won $1,000,000?

# Using a shared secret key
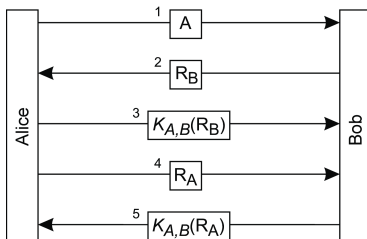


### Steps

1. Alice announces she wants to talk to Bob.

2. Bob returns a nonce.

3. Alice encrypts the nonce with the shared key $K_{A,B}$, thus proving that she owns $K_{A,B}$ $\Rightarrow$ Bob knows he's talking to Alice.

4. Alice sends a nonce to Bob.

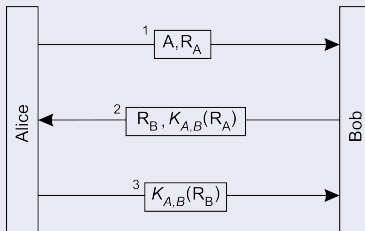5. Bob returns proof that he owns the shared secret key as well $\Rightarrow$ Alice knows she's talking to Bob.

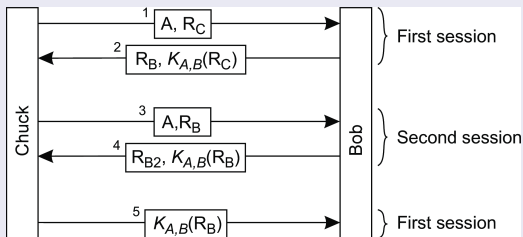## About optimizations



Let's reduce the number of messages

Alice → Bob: 1 | A,R_A |

Bob → Alice: 2 | R_B, K_{A,B}(R_A) |

Alice → Bob: 3 | K_{A,B}(R_B) |
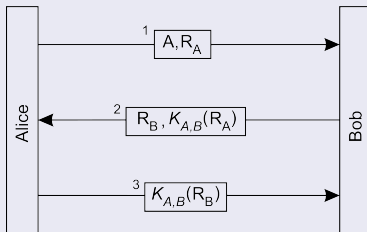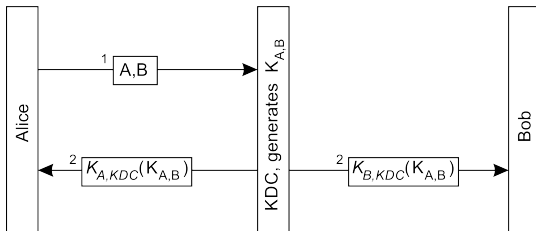
# About optimizations

Let's reduce the number of messages



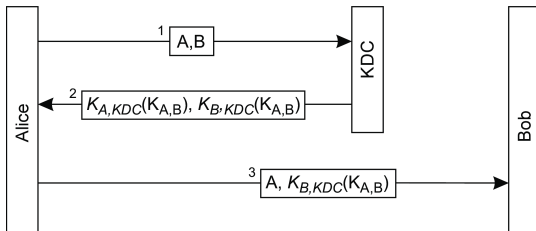We just broke the protocol

## Using a Key Distribution Center



### Basics

Every client has a secret key shared with the KDC.

1. Alice tells the KDC that she wants to talk to Bob
2. The KDC sends a fresh secret key, shared by Alice and Bob
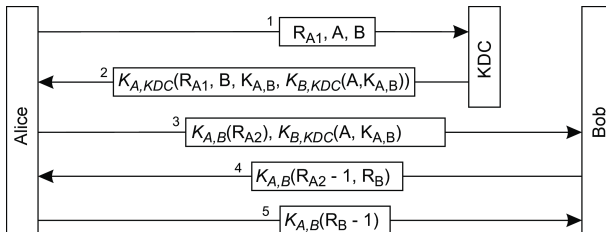
# Using a Key Distribution Center



### Basics

Using a ticket is practically better:

1. Alice tells the KDC that she wants to talk to Bob

2. The KDC sends a fresh secret key, shared by Alice and Bob

3. Alice tells Bob that she wants to talk, along with the key to be used.
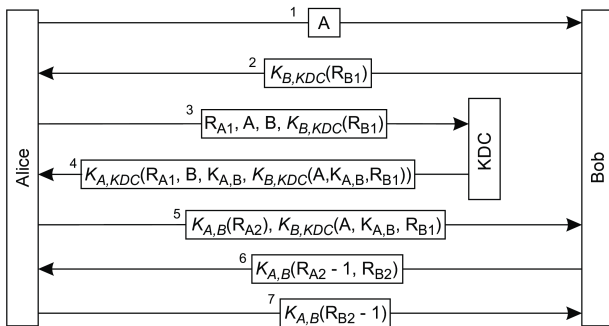
# The Needham-Schroeder protocol



### Important observation

In the case of request-response messages, you want to make sure that the received response, is associated with the sent request. Mitigates replay attacks.

### General principle

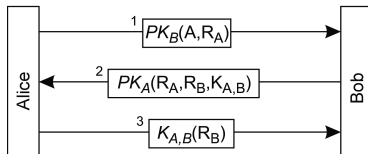Use nonces to relate any combination of request-response messages.

## Mitigate against reuse of keys



### Some observations

- Note how *B1* ties message #2 to #5

- Note that by returning $R_{A2} - 1$ in #6, Bob proves he knows $K_{A,B}$

- And, likewise, in the case of Alice in #6 (by modifying $R_{B2}$).

# Using public keys



### Steps

1. Alice tells Bob she wants to talk, sending a nonce $R_A$, and encrypting the message with Bob's public key.

2. Bob generates a shared secret session key $K_{A,B}$, proves he is the owner of $PK_B$ by decrypting $R_A$, and challenges Alice to prove she owns $PK_A$.

3. Alice decrypts the response, and proves to Bob that she is Alice by then sending Bob's nonce back encrypted with the generated session key $K_{A,B}$.

## Practical example: Kerberos
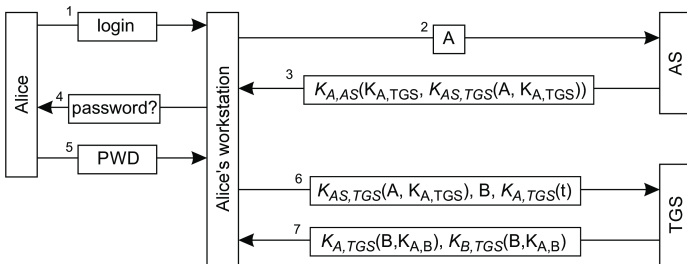


### Essence

1,2 Alice types in her login name.

  3 The Authentication Service returns a ticket $K_{AS,TGS}(A, K_{A,TGS})$ that she can use with the Ticket Granting Service.

4,5 To be able to decrypt the message, Alice must type in her password. She is then logged in. Using the AS in this way, we have a single sign-on system.

6,7 Alice wants to talk to Bob, and requests the TGS for a session key.

## Transport Layer Security



$$SK^*_{C,S} = f(H(m_1|m_2), SK^{DH}_{C,S})$$
$$SK^{**}_{C,S} = f(H(m_1|m_2|m_3), SK^{DH}_{C,S})$$
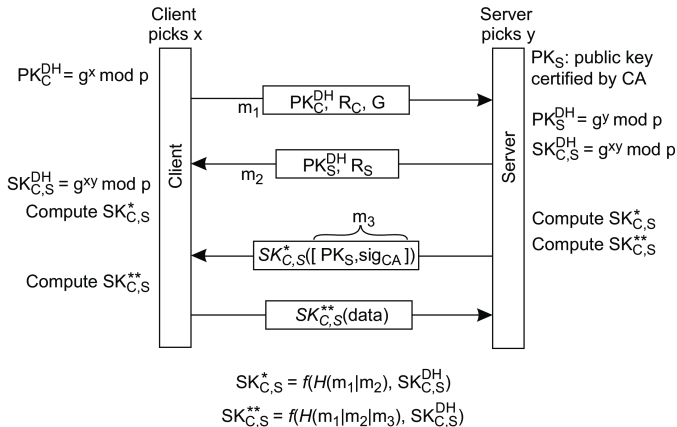
- $G$ denotes a specific set of parameter settings, called a group (e.g., values for $p$ and $g$).

## Transport Layer Security
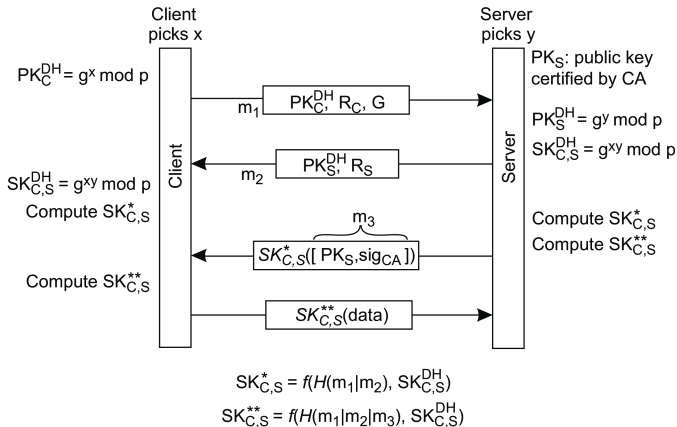


- The client uses a nonce $R_C$; the server uses $R_S$
- $H(m_1|m_2)$ denotes the hash over the concatenation of $m_1$ and $m_2$

# On trust

### Definition

*Trust is the assurance that one entity holds that another will perform particular actions according to a specific expectation.*

### Important observation

- Expectations have been made explicit ⇒ no need to talk about trust?

- Example: Consider a Byzantine fault-tolerant process group of size *n*

  - Specificiation: the group can tolerate that at most $k \leq (n-1)/3$ processes go rogue.
  - Realisation: for example PBFT.
  - Consequence: if more than *k* processes fail, all bets are simply off.
  - Consequence: it's not about trust, it's all about meeting specifications.

- Observation: if a process group often does not meet its specifications, one may start to doubt its reliability, but this is something else than (dis)trusting the system.

# Sybil attack

## Essence: Just create multiple identities, but owned by one entity

- In the case of a peer-to-peer network:

```
1  H = set of honest nodes
2  S = set of Sybil nodes
3  A = Attacker node
4  d = minimal fraction of Sybil nodes needed for an attack
5
6  while True:
7      s = A.createNode()        # create a Sybil node
8      S.add(s)                  # add it to the set S
9
10     h = random.choice(H)      # pick an arbitrary honets node
11     s.connectTo(h)            # connect the new sybil node to h
12
13     if len(S) / len(H) > d:   # enough sybil nodes for...
14         A.attack()            # ...an attack
```

# Sybil attack

**Essence: Just create multiple identities, but owned by one entity**

- In the case of a peer-to-peer network:

```
1  H = set of honest nodes
2  S = set of Sybil nodes
3  A = Attacker node
4  d = minimal fraction of Sybil nodes needed for an attack
5
6  while True:
7      s = A.createNode()        # create a Sybil node
8      S.add(s)                  # add it to the set S
9
10     h = random.choice(H)      # pick an arbitrary honets node
11     s.connectTo(h)            # connect the new sybil node to h
12
13     if len(S) / len(H) > d:   # enough sybil nodes for...
14         A.attack()            # ...an attack
```

- In the case of a Web-of-trust:

  - Endorse a public key without an out-of-band check.
  - *Bob* checks with $k > 1$ others that they have endorsed *Alice*'s key.
  - *Alice* creates $k > 1$ identities each stating her key is valid.

## Eclipse attack

### Essence: Try to isolate a node from the network

Example: a hub attack in the case of a gossip-based service. In this case, when exchanging links to other peers, a colluding node returns links only to other colluders.
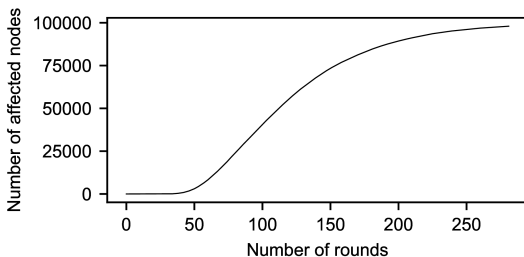


Affected node: has links only to colluders.

### General solution

Use a centralized certification authority.

# Preventing Sybil attacks: Blockchain solutions

### Essence: creating an identity comes at a cost

In the case of permissionless blockchains:

- Proof-of-Work: Let validators run a computational race. This approach requires considerable computational resources

- Proof-of-Stake: Pick a validator as a function of the number of tokens it owns. This approach requires risking loss of tokens.

## Preventing Sybil attacks: Decentralized accounting

### A simple example

- Each node $P$ maintains a list of nodes interested in doing work for $P$: the choice set of $P$ ($choice(P)$).

- Selecting $Q \in choice(P)$ depends on $Q$'s work for others (i.e., its reputation).

- $P$ maintains a (subjective) view on reputations. Of course, $P$ knows precisely what it has done for others, and what others have done for $P$.

- $P$ can compute a capacity ($cap(Q)$:

$$cap(Q) = \max\{MF(Q,P) - MF(P,Q), 0\}$$

with $MF(P,Q)$ the amount of work that $P$ has, or could have contributed to work done for $Q$, including the work done by others.

# Preventing Sybil attacks: Decentralized accounting

### Essence: Keep track of work that nodes do for each other

- Assume $R$ directly contributed 3 units of work for $Q$, and $R$ had processed 7 units for $P$ $\Rightarrow$ $P$ may have contributed 3 units of work for $Q$, through $R$.

- Reasoning: $R$ may never have been able to work for $Q$, if it had not worked for $P$.

## Preventing Sybil attacks: Decentralized accounting

### How Sybil attacks are prevented

- Let $Q \in choice(P)$ create $n$ Sybil nodes $Q_1^*, \ldots, Q_n^*$; $Q = Q_0^*$

- For work by $Q_i^*$ for $Q_j^*$ to increase $cap(Q_i^*)$:

  1. $Q_j^*$ needs to have worked for some node $R$
  2. $R$ needs to have worked for $P$

  In other words: $Q$ can successfully attack only if it had worked for honest nodes. Also, honest nodes have to work for $Q$: the total capacity $Tcap(Q)$ of the Sybils must grow, with
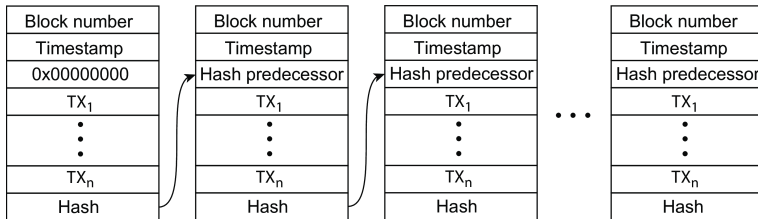
  $$Tcap(Q) = \sum_{k=0}^{n} cap(Q_k^*)$$

- Assume that $P$ works 1 unit for $Q_i^* \Rightarrow MF(P, Q_i^*)$ increases by 1 unit $\Rightarrow$ $cap(Q_i^*)$ drops by 1 unit, and so does $Tcap(Q)$.

- As soon as $Tcap(Q)$ drops to 0, $P$ will look at other nodes.

## Trusting a system: Blockchains

### Essence

One needs to know for sure that the information in a blockchain has not been tampered with: data integrity assurance. Solution: make sure that no change can go unnoticed (recall: a blockchain is an append-only data structure).

| Block number | | Block number | | Block number | | Block number |
|---|---|---|---|---|---|---|
| Timestamp | | Timestamp | | Timestamp | | Timestamp |
| 0x00000000 | | Hash predecessor | | Hash predecessor | | Hash predecessor |
| $TX_1$ | | $TX_1$ | | $TX_1$ | | $TX_1$ |
| $\vdots$ | | $\vdots$ | | $\vdots$ | $\cdots$ | $\vdots$ |
| $TX_n$ | | $TX_n$ | | $TX_n$ | | $TX_n$ |
| Hash | | Hash | | Hash | | Hash |

### Observation

Any change of block $B_k$, will affect its hash value, and thus that of $B_{k+1}$, which would then also need to be changed, in turn affecting the hash value of $B_{k+2}$, and so on.
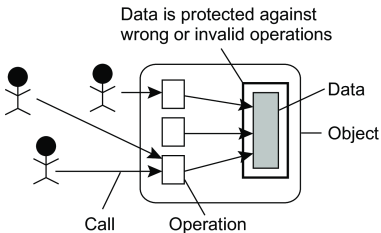
# Access control: General model

## Authorization

Making sure that authenticated entities have only access to specific resources.
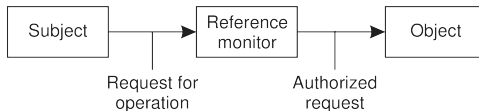


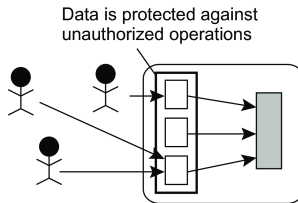## Observation

The reference monitor needs to be tamperproof: it is generally implemented under full control of the operating system, or a secure server.
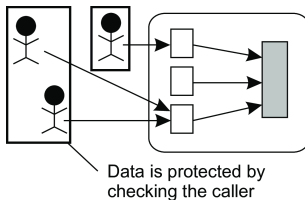
## Protection



...against invalid operations    ...against unauthorized access

...against unauthorized invokers

## Access control policies

1. **Mandatory access control**: A central administration defines who gets access to what.

2. **Discretionary access control**: The owner of an object can change access rights, but also who may have access to that object.

3. **Role-based access control**: Users are not authorized based on their identity, but based on the role they have within an organization.

4. **Attribute-based access control**: Attributes of users and of objects they want to access are considered for deciding on a specific access rule.

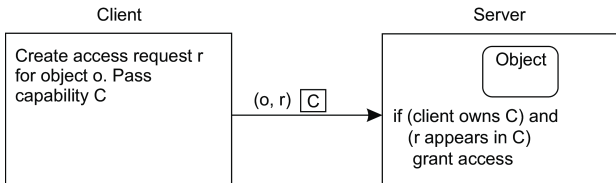# Access control matrix

### Theory

Construct a matrix in which $M[s, o]$ describes the access rights subject $s$ has with respect to object $o$. Impractical, so use access control lists or capabilities.
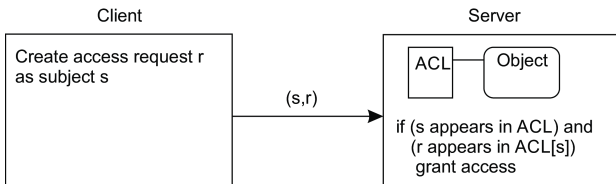


Access control list



Capabilities

## Special case: Attribute-based Access Control

Distinguish different classes of attributes:

- User attributes: name, data of birth, current roles, home address, department, qualifiers obtained, contract status, etc. May also depend on role (e.g., teacher or student).

- Object attributes: anything – creator, last-modified time, version number, file type, file size, but also information related to its content.

- Environmental attributes: describe the current state of the system, e.g., date and time, current workload, maintenance status, storage properties, available services, etc.

- Connection attributes provide information on the current session, e.g., IP address, session duration, available bandwidth and latency estimates, type and strength of security used.

- Administrative attributes: reflect global policies, e.g., minimal security settings, general access regulations, and maximum session durations.

# Example: the Policy Machine

### Essence

A server maintains sets of (*atrribute*,*value*) pairs, distinguishing users, applications, operations, and objects. At the core, we formulate access control rules.

### Access control rules

- Assignment: A user $u$ can be assigned to an attribute $ua$: $u \rightarrow ua$. An object to an attribute: $o \rightarrow oa$; an attribute to an attribute: $ua_1 \rightarrow ua_2$ (meaning that if $u \rightarrow ua_1$, then $u \rightarrow ua_2$. Leads to rules like *allowed*($ua, ops, oa$): users assigned to $ua$ are allowed to execute operations in $ops$ on objects assigned to $oa$.

- Prohibition: explicitly state what is not allowed, such as *denied*($u, ops, os$). Also: *denied*($u, ops, \neg os$), meaning denial when $u$ wants to perform $o$ assigned to $ops$ on an object not in $os$.

- Obligation: automated action upon an event, such as denying copying of information:

$$\text{when } u \text{ reads } f \in fs \text{ then } denied(u, \{write\}, \neg fs).$$

# Delegation

### What's the issue?

Alice makes use of an e-mail service provider who stores her mailbox. She is
required to log in to the provider to access her mail. Alice wants to use her own
local mail client. How to allow that mail client to act on behalf of Alice? How to
delegate Alice's access rights to her mail client?

### Observation

It is not a good idea to hand over all user credentials to an application: why
would the application or the machine be trusted? $\Rightarrow$ use a security proxy.

# Security proxy
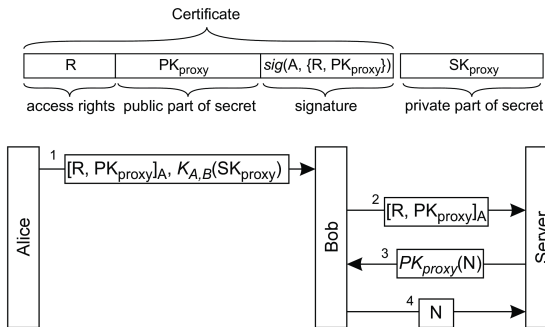


### How it works

1. Alice passes some rights $R$ to Bob, together with a secret key $SK_{proxy}$
2. When Bob wants to exercise his rights, he passes the certificate
3. The server wants Bob to prove he knows the secret key
4. Bob proves he does, and thus that Alice had delegated $R$.

## Example: Open Authorization (OAuth)

### Four different roles

- **Resource owner**: typically an end user.
- **Client**: an application that one would like to act on behalf of the resource owner,
- **Resource server**: An interface through which a person would normally access the resource.
- **Authorization server**: an entity handing out certificates to a client on behalf of a resource owner.

### Initial steps

1. The client application registers itself at the authorization server and receives its own identifier, *cid*.

2. Alice wants to delegate a list $R$ of rights $\Rightarrow$

   $$\text{Client: } send \ [cid, R, H(S)]$$

   with a hash of a temporary secret $S$

## Completing the process

### Final steps

③ Alice is required to log in and confirm delegation $R$ to the client.

④ Server sends a temporary authorization code $AC$ to client.

⑤ Client requests a final access token:

$$\text{Client: } \textit{sends } [\textit{cid}, AC, S].$$

Sending $S$ to the authorization server allows the latter to verify the identity of the client (by computing $H(S)$).

The authorization server has now (1) verified that Alice wants to delegate access rights to the client, and (2) has verified the identity of the client $\Rightarrow$ it returns an access token to the client.

## Example: decentralized authorization

### WAVE (and keeping it very simple)

Essence: Alice delegates rights to Bob, Bob delegates some of those rights to Chuck.

- When Check wants to exercise his rights, there should be no need for Alice or Bob to be online.
- No one but Alice, Bob, and Chuck need to be aware of the delegation.

### Essentials

Alice delegates rights $R$ to Bob, for which he creates a keypair $(PK_B^R, SK_B^R)$:

$$A \text{ sends: } PK_B^R(\underbrace{[R|SK_A^R]}_{m_1}))$$
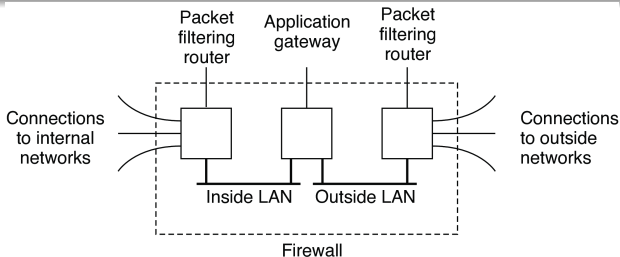
Bob delegates parts of those rights $R'$ to Chuck, assuming he is allowed to do so:

$$B \text{ sends: } PK_C^{R'}(\underbrace{[R'|m_1|SK_B^R]}_{m_2})$$

# Firewalls

## Essence

Simply prevent anything nasty coming in, but also preventing unwanted outbound traffic.



## Different types of firewalls

- Packet-filtering gateway: operates as a router and makes filters packets based on source and destination address.
- Application-level gateway: inspects the content of an incoming or outgoing message (e.g., gateways filtering spam e-mail).
- Proxy gateway: works as a front end to an application, filtering like an application-level gateway (e.g., Web proxies).

## Intrusion detection systems

### Two flavors

- **Signature-based**: matches against patterns of known network-level intrusions. Problematic when series of packets need to be matched, or when new attacks take place.

- **Anomaly-based**: assumes that we can model or extract typical behavior to subsequently detect nontypical, or anomalous behavior. Relies heavily on modern artificial-intelligence technologies.

## Intrusion detection systems

### Two flavors

- **Signature-based**: matches against patterns of known network-level intrusions. Problematic when series of packets need to be matched, or when new attacks take place.

- **Anomaly-based**: assumes that we can model or extract typical behavior to subsequently detect nontypical, or anomalous behavior. Relies heavily on modern artificial-intelligence technologies.

### Using sensors

Key idea is to manage false and true positives (FP/TP) as well as false and true negatives (FN/TN). Maximize accuracy and precision:

$$\text{Accuracy:} \quad \frac{TP + TN}{TP + TN + FP + FN}$$

$$\text{Precision:} \quad \frac{TP}{TP + FP}$$