Authentication in Distributed Systems

Austen Barker
Access Control

principal → Operation → Reference monitor → Object
Challenges of a distributed system

**Autonomy**: Path from principal to an object may be long and convoluted.

**Size**: Usually much bigger, multiple authority sources.

**Heterogeneity**: Different methods of securing channels.

**Fault tolerance**: One part can fail but the rest must remain operational.
Terms

Principals: People, machines, roles, groups

Channels: Ports, Wires, Encrypted channels, Network addresses

Compound Principals: Bob as admin, Alice for Bob, Alice and Bob
Trusted Computing Base

The hardware/software that security depends on.

Ideally for access control only the encryption method and keys are in a TCB.

Fail Secure: In the case of Untrusted component failure

Deny access it should have granted

Caching: need some method of invalidating entries
Statements

A | B, A quotes B

A => B, A speaks for B or A says B

S ≡ S’, S equals S’

S ⊃ S’, S implies S’

S ^ S’, S and S’

“ s means that s is an an axiom or provable by axioms
Principals

Two basic operators ^ (and) and | (quoting).

\[(A \rightarrow B) \equiv (A = A \land B)\]

A speaks for B implies that if A says S then it implies that B says S.

Additional operators in the form of as, for, says.
Handoff and Credentials

“(A says ( B =>A)) ⊃ (B => A) Defines a handoff, essentially A giving permission for B to speak for A.

A` says (B` => A`) together with premises is B’s credentials for A. Each premise has a lifetime, the lifetime of the credentials is the lifetime of the shortest lived premise.
Channels and Encryption

It is essential to be able to know the possible senders on a channel.

If we know this then a channel has integrity. Finding an A such that $C \Rightarrow A$

Authenticating a channel, only allowed if $A$ does.

If we know the receivers as well the channel has secrecy.

With subchannels we say, $C \Rightarrow X \Rightarrow S$, or $C|X \Rightarrow S$. 
Credentials and Encryption

Ideally the encryption algorithm used to encrypt a message $x$ where the algorithm is of the form

$$\text{Decrypt}(K, \text{Encrypt}(K\', x)) = x$$

and has the following

Secrecy: If you know $\text{Encrypt}(K\', x)$ but not $K$, you can’t compute $x$

Integrity: If you choose $x$ but don’t know $K\'$ you can’t compute a $y$ such that $\text{Decrypt}(K, y) = x$

If you encrypt $S$ with $K\'$ any entity that possesses $K$ can infer that $K \Rightarrow s$. This is a certificate.
Credentials and Encryption

Two kinds of Encryption

Shared Key: $K=K'$, have to worry about transmitting key securely.

- AES, DES, RC4, etc

Public/Private key: $K \neq K'$ and can’t compute one from the other. $K$ is public and $K'$ is private,

- RSA, slower, can be used to set up a symmetric solution
Key identifiers

Can be any one of the following for a identifier $K'$ for receiver $R$

An index into a table of keys that $R$ has.

Encrypt($K_m, K$) where $K_m$ is a master key that only $R$ knows

A pair ($K'_r, \text{Encrypt} (K', K)$) where $K'_r$ is a key identifier for $K'$
Broadcast encryption

Sending a message without knowing who will receive it.

Shared key can also simulate public key with a relay using key identifiers.
Node to Node Secure channels

Uses shared key to secure channels between nodes. Then multiplexes to obtain more secure channels.

Relies on a network interface that can dissect a packet, then encrypt/decrypt on the fly.

The key should be easily changed without incurring a high cost.

Each node sends a public key to the other,

The use this to encrypt symmetric keys and send them to each other.

Each node will compute key identifiers.
Principals with Names

People can’t usually rely on unique identifiers or keys as names.

Must verify if C => A

Two methods Push and Pull

Push: sender collects A’s credentials and presents them.

Pull: receiver looks up A in some database to get credentials, name lookup.
Single Certificate Authorities

In a simple system: There is only one CA, everyone trusts CA, everyone knows its public key Kca such that Kca => CA

It issues a certificate so Kca says Ka => A.

With public key, don’t have to speak to CA, one can just look at the certificates.

Can use an intermediate database O with short lifetimes Authorized by A

Kca says (O| Ka ^ Ka ) => A
Kerberos

A gets a certificate CA (authentication server) that says $\text{Kca} \Rightarrow \text{A}$

$\text{Kca}$ (ticket granting ticket) is then used to get tickets from relay $\text{R}$ (ticket granting server)

This relay can then authenticate a channel from $\text{A}$ to some resource $\text{B}$. It splices the ticket granting ticket and a known ticket for $\text{B}$ into one that says $\text{Kb} \text{ says } \text{Kab} \Rightarrow \text{A}$.

$\text{R}$ sends a certificate that can be used for $\text{B}$ to access $\text{A}$. 
Multiple Authorities

More practical to have a decentralized system.

When you want to authenticate a channel you start from an authority that you believe has a certain name and then traverse the shortest path.

Requires the destination to trust each authority on the path.
Groups

A group has no public key or channel, other principals speak for the group as members.

A group name yields one or more group certificates. Kca says P1 => G, Kca says P2 => G, ...

Certificates are clumsy to use as authentication for a group.

One way to do this is to have a group key that is encrypted by each members private key. So each member says Kg says S. To verify one must assert Kg => G.
Roles and Programs

A role is a mechanism used by a principal to provide access to select programs. The paper writes this as “A as R” for A assuming role R.

A says R says S. Treating R as a weaker principal. Therefore defining it as A|R so A as R says S.

A program/role can be represented as a digest in order to distinguish it from others.
Loading Programs and Booting

There is no problem if a node A trusts a file system to speak for a program.

If there isn’t that trust then it will compute the digest of the program and looks up credentials for D→P.

One can also issue certificates to trusted programs so computing the digest isn’t necessary.

Bringing a machine up is similar although,

Upon boot it creates a key pair and someone (usually an admin) issues a certificate for it Kca says Km → M

When M gives control to a program booted it hands over all hardware resources.

M invents a key pair for a new node, M must know a keys or digests for programs run to trust P with resources, M must protect its key from any programs and only readable on boot.
Delegation

A principal should be able to delegate its authority to another.

“ \( A^B | A \Rightarrow B \) for A

\((A \land B | A) \) says \( B | A \Rightarrow B \) for A

\( B | A \Rightarrow B \) for A.

B can speak for B for A by quoting A.

Timeouts are then used to revoke delegations. Then B must ask A to refresh it.
Login

Delegation works in the case of logging into a system with some differences.
   It assumes a user's key Ku is available only while logging in.
   A rather long lifetime is generally needed.

A temporary key pair is made at login time and is known as a session key. This is used to refresh login info.
   It is forgotten when a user logs out.

An specific highly available agent can be used to refresh credentials for long-running jobs.
**Authenticating IPC**

Any distributed system requires authenticating messages from one process to another.

It is necessary to have a process speak for multiple principals.

For instance we have Kws => WS as Taos as accounting for bwl. Which then runs as process pr which then speaks on C | pr

So reading file foo has the form C|pr says “read foo”
IPC channels

In most cases we have a sending process, a receiving process, and a secure channel. This secure channel is obtained by multiplexing another channel between two nodes. 
Cs| prs-prr.

To ensure integrity we care that the message came from the sender.
We encode principal A as an authentication identifier.
Multiplex the process with the id, the id is not reused.
Authentication Agent

Sending and receiving agents to verify messages.

Credentials are a collection of certificates from the sender used to prove that $Caid => A$.

- Gets message with an aid that is interpreted as $aid$ says $s$.
- Checks the channel it came in on and concludes $Cs|prs|aid$ says $s$.
- Calls on the local agent to learn the principal $A$ to verify $A$ says $s$ through $Caid = Cs|prs|aid$
- Caches that $Caid => A$.

For the sender it responds to cache requests and keeps track of,

- The meaning $A$ of each $aid$.
- Certificates to prove that $Cs|aid => A$
- The processes are allowed to speak for $aid$. 
Agents also hand off authorities from one process to another. An authority is an *aid* that a process speaks for.

Each process has an authority given by user login or a program running in the process.

To handle revocation of a certificate all certificates are checked whenever one expires and refresh those about to expire to keep costs down.

To grant access a call like `Read(file, authority a)` will be seen as Ca says “read file f”

An ACL is obtained for f and it will check if Ca speaks for a principal that has read access.
Access Control

ACL will grant a request if $A \rightarrow B$ and $B$ is a principal that has the requisite rights.

Due to the difficulty of generating a proof of this we must structure the form of an ACL to ensure reasonable speed and completeness.

```
principal ::= forList | principal ∧ forList
forList ::= asList | forList for asList
asList ::= properPrincipal | asList as role
role ::= pathName
properPrincipal ::= pathName | channel

*Roles ∩ properPrincipals = ∅
Input is a set of premises $P => Q$
```
Access Control

The inputs to the checking algorithm being the ACL, requesting principal, and the premises.

To deal with long lookup times one can tag each principal in an ACL or group cert or send with the credentials a tag and only looking it up if a tag appears or if it is marked for unconditional lookup.

Denying access to a specific principal is tricky, they can have more than one name, certificates are not securely stored.

In case of doubt deny access anyway.