Authentication

Slides by Hussain Almohri

Modeling Authentication

Goal: A way to allow users(s) access a system

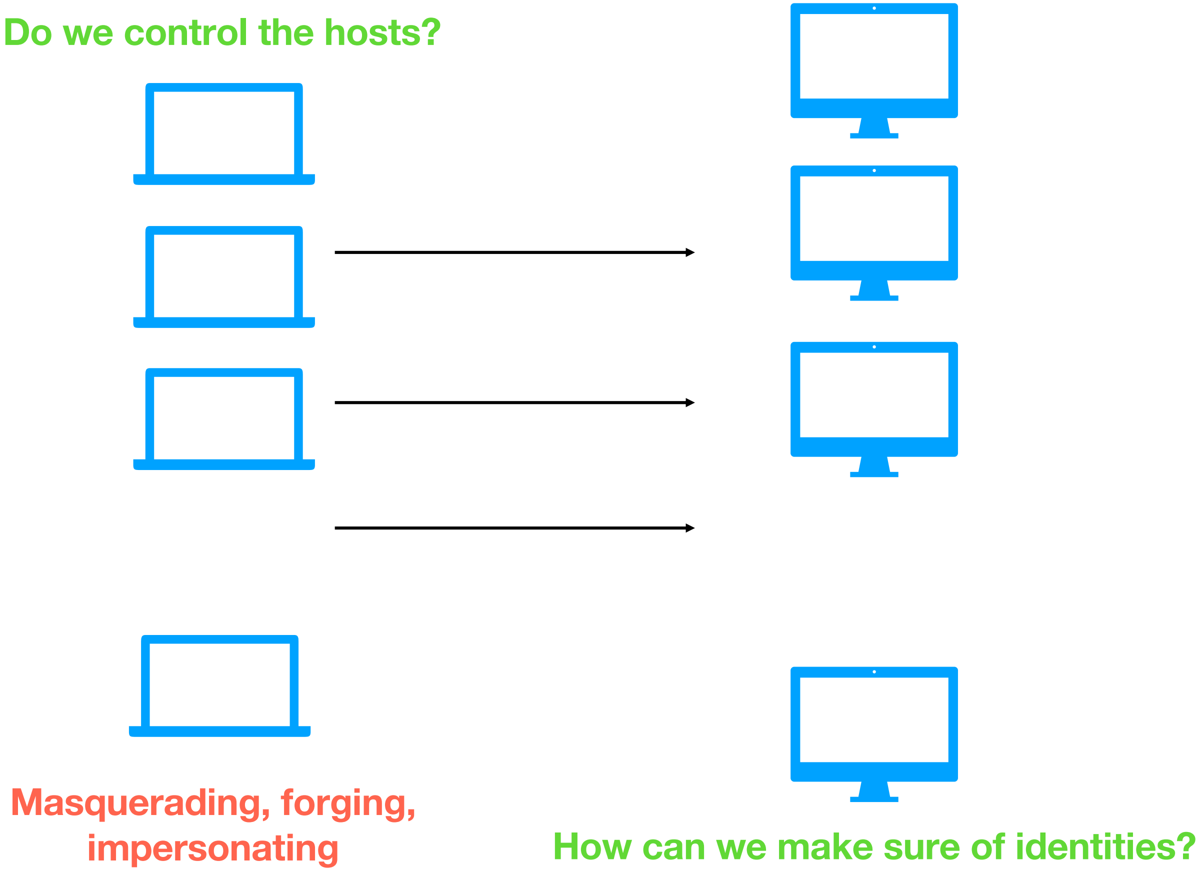
Modeling Authentication

- Need for undeniable, available, and unforgeable method for ensuring that a user is allowed access if and only if that user has the right to access.
- Authentication comprises a singular identity, something that **cannot be two**!
- Form of authentication is determined by the application.

Basic Authentication

- Secrets, why?
- Usernames, why?
- Multiple parties
- Convenience
- Multi-factor authentication, why?
- Multi-level authentication, why?

The problem in open networks



Authentication in Open Networks

In the context of secure computer communications, authentication means verifying the identity of the communicating principals to one another.

(Needham and Shroeder, 1978)

Needham and Shroeder Protocol

- Authentication for multiple interactive parties, single direction communication (e.g., mail), and third party authentication.
- Assumptions:
  - Participating parties can execute encryption/decryption functions using keys that are cryptographically secure.
  - Intruders can alter or copy parts of a communication.
  - Each party has a secure environment, such as a workstation.
  - Parties “choose” to communicate securely (not forced to).
- Goals:
  - Ensure detection of tampering attacks
  - Be secure against traffic analysis
  - Maximize network efficiency

Basic Crypto Tools

- Symmetric encryption: a single key encrypts and decrypts traffic, shared between two or three parties.
- Asymmetric (public-key) encryption:
    - Originated by Diffie and Hellman in 1976.
    - Two keys: PK, SK. Anyone can know PK and can encrypt messages using PK. But only one knows SK and can decrypt messages using SK.
    - Knowledge of one key provides no clue about the other.

Connecting A to B

- The system has an authentication server and a name server.
- Core Idea: produce a message that only B can understand. B must be sure that A originated the message.

Connecting A to B

Nonce: used only once 👇

$$\mathbf{A} \xrightarrow{A,\ B,\ I_{A1}} \mathbf{AS}$$

AS looks up KA and KB and generates a new key CK

Encrypted with A's key, KA 👇

$$\mathbf{A} \xleftarrow{\{I_{A1},\ B,\ CK,\ \{CK,\ A\}^{KB}\}^{KA}} \mathbf{AS}$$

What happens if B is not specified here?

Encrypted with B's key, KB 👇

$$\mathbf{A} \xrightarrow{\{CK,\ A\}^{KB}} \mathbf{B}$$

Connecting A to B

B must make sure, CK is for A.

$$A \xleftarrow{\quad \{I_B\}^{CK} \quad} B$$

A assures B that, yes, CK is owned by me.

$$A \xrightarrow{\quad \{I_B - 1\}^{CK} \quad} B$$

Connecting A to B (PK)

$$\mathbf{A} \xrightarrow{\text{A, B}} \mathbf{AS}$$

Encrypted with AS's secret key, SKAS 👇🏻

$$\mathbf{A} \xleftarrow{\{PKB, B\}^{SKAS}} \mathbf{AS}$$

A knows AS's PKAS, used to decrypt the message.

Encrypted with B's key, KB 👇🏻

$$\mathbf{A} \xrightarrow{\{I_A, A\}^{PKB}} \mathbf{B}$$

$$\mathbf{B} \xrightarrow{\text{B, A}} \mathbf{AS}$$

$$\mathbf{B} \xleftarrow{\{PKA, A\}^{SKAS}} \mathbf{AS}$$

Connecting A to B (PK)

$$\mathbf{A} \xleftarrow{\{\{I_A, I_B\}^{SKB}\}^{PKA}} \mathbf{B}$$

$$\mathbf{A} \xrightarrow{\{\{I_B\}^{SKA}\}^{PKB}} \mathbf{B}$$

Kerberos

- An authentication system designed by Miller and Neuman for open network computing environments.
- Part of MIT's Project Athena
- Report by Steiner, Neuman, and Schiller (1988)

Requirements

- Security: circumventing kerberos should not be trivial
- Reliability: access to services will depend on it
- Transparent: users should not notice the authentication taking place
- Scalable: should be able to work with other protocols
- Kerberos should not rely on the security of the authenticating parties
- No authenticated user should be left behind

Kerberos System

- User 👨🏻: a human being using the system
- Client 👨🏻 💻 : an entity that uses the system (not necessarily a person)
- Server 🖥️: Responds to client requests, together with client form a network application
- Principals: Kerberos clients
- Service: An abstraction of actions to be performed; servers are processes that provide services

Kerberos System

- Private key (key) 🔑: A large number assigned to a principal
- Private keys applied to user passwords 🔑 👨🏻
- Master machine: hosts the definitive copy of the kerberos database.
- Slave machine: hosts a replica of the kerberos database.

What is kerberos?

- A third party authentication *service*, trusted by two other parties to be accurate and secure.
- It relies on Needham and Schroeder key exchange protocols.
- Stores a database of keys. Each key is only shared between kerberos and a single client.
- Clients and users register with kerberos prior to using it.
- Also uses session keys, which are temporary private keys

Levels of protection

- Authentication only when initiating a request
  - Subsequent messages trusted according to network address
- Authentication of each messages, but could be transparent
- Authentication of each messages, with encryption: private messages

Kerberos database

- A record of private key, user ID, and expiry time
- Decoupled from personal user information, handled by a name server, increasing security of sensitive data
- Administration server (KDBM): provides read/write access to database, only runs on the machine hosting the DB
- Authentication server: needs read-only access to DB and can run on any replica
- Database propagation software: provides replicas with a recent view of the master DB

Naming

- Both clients and servers are named.
- A name consists of a primary name, an instance, and a realm, expressed as *name.instance@realm*.
- Each realm has its own kerberos setup.

Primary name (user) -> treese.root <- Instance (privilege)

rlogin.priam@ATHENA.MIT.EDU <- Realm

Primary name (server) ->

Instance ->

Proving Identity

1. The user obtains credentials to be used to request access to other services.

2. The user requests authentication for a specific service.

3. The user presents those credentials to the end server.

Credentials

- Tickets: A ticket is used to pass the identity of the person (possessing the ticket) between the authentication server and the end server.
- Ticket has information to proving the person who presents it is the one who was issued for.
- A ticket is good for a single server and a single client, but could be used multiple times (before expiration).

A shared key between client and target server 👇

$$\{s, c, addr, timestamp, life, K_{s,c}\}K_s$$

Encrypted using target server's key 👆

Credentials

- Authenticator contains the information for proving identity of client possessing the ticket.
- Can only be used once, each time the client wants to use the service.
- Client generates the authenticator.

$$\{c, addr, timestamp\}K_{s,c}$$

Encrypted using shared key in ticket 👆

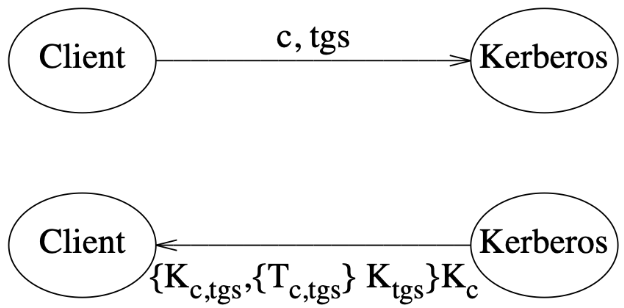
Initial Authentication

1. The user enters username, which is sent with a TGS request to auth server.

2. If kerberos knows client, a random session key is generated.

3. Generates a ticket (T_c, tgs) for TGS: client's name, name of the ticket-granting server, current time, ticket lifetime, client IP, session key, encrypted using key (K_{tgs}) btw TGS and auth server.

4. Response (ticket and random key) sent back encrypted with client's key (K_c).

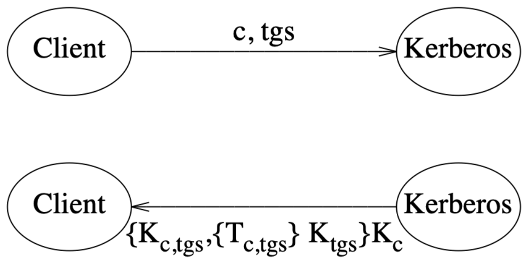


TGS: Ticket Granting Server

Initial Authentication

5. Client is asked to enter password, which is used to generate the client key, in turn used to decrypt the password.

6. Ticket and session key are stored for future use.



Requesting Service

- Assume, client now has a ticket from a TGS for a desired server.

Session key 👇

- Application generates an authenticator: $\{A_c\}K_{c,s}$

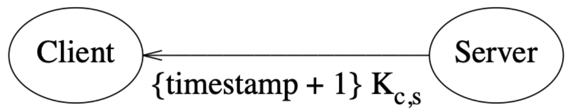
Ticket from TGS 👇

- Client sends $\{A_c\}K_{c,s}$ and $\{T_{c,s}\}K_s$ to the server.

- Server decrypts ticket, uses key in ticket to decrypt authenticator, compares authenticator and ticket, if matched, server will allow the request to proceed.

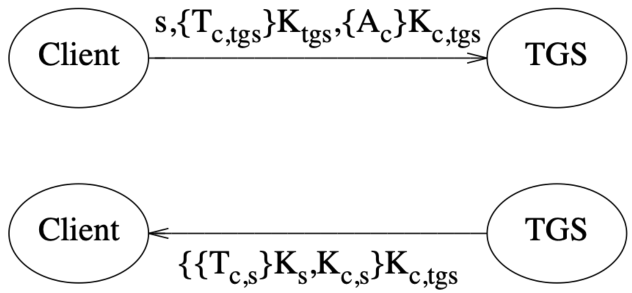
Proving the server's ID

- Server adds one to the client's timestamp, encrypts it with the shared session key and sends it back to the client.

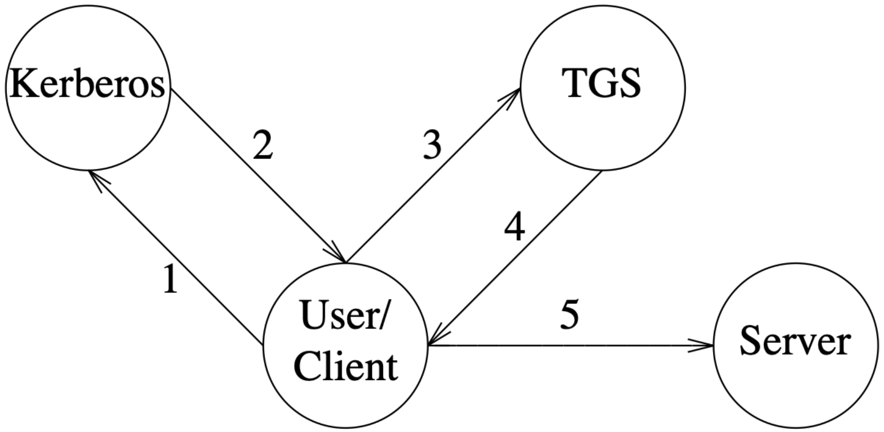


Getting the tickets

- A ticket granting server (TGS) is a server that demands authentication just like any other server.
- Request to TGS contains the server name, a ticket from kerberos encrypted with the TGS key, and an authenticator.
- TGS builds a new ticket with client's name, the server name, the current time, the client's IP address and a new session key.
- TGS replies with new ticket, encrypted with session key between client and TGS in the original ticket from auth service.

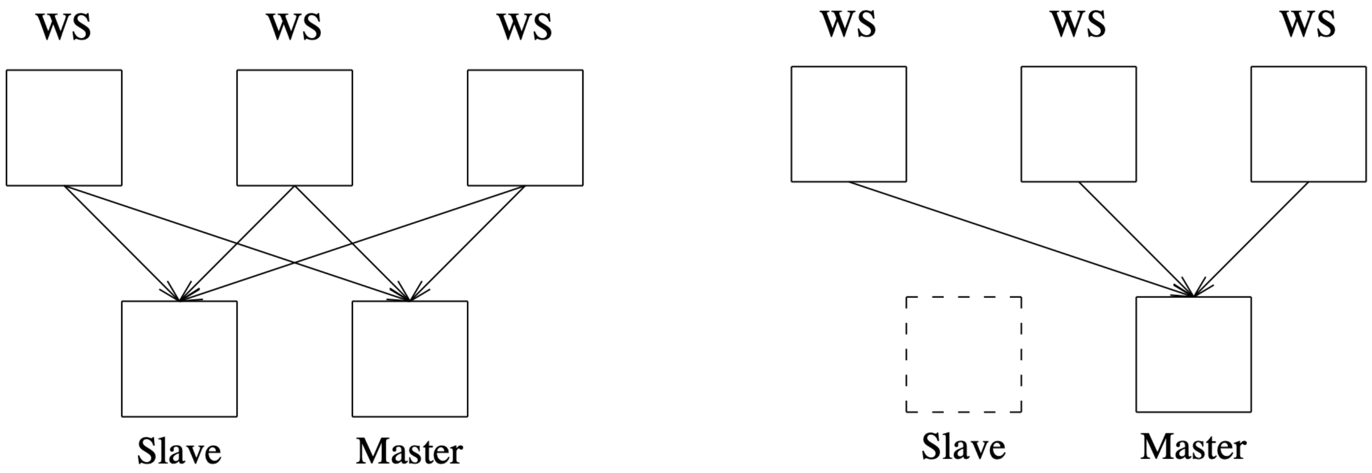


Protocol Summary



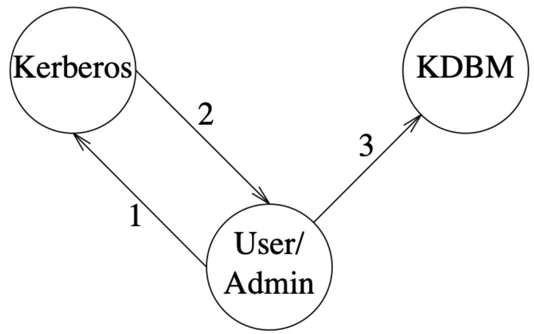
1. Request for TGS ticket
2. Ticket for TGS
3. Request for Server ticket
4. Ticket for Server
5. Request for service

Auth. vs. Admin.



KDBM

- KDBM allows for changing passwords or adding users.
- It must receive a ticket from kerberos itself not a TGS.
- Only allows access if principal name with request match or the principal has admin access.



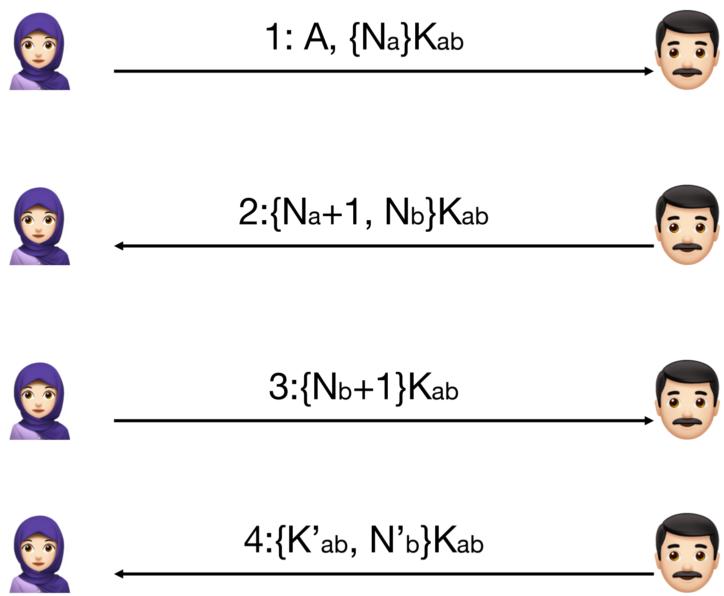
1. Request for KDBM ticket
2. Ticket for KDBM
3. *kadmin* or *kpasswd* request

The Andrew RPC handshake

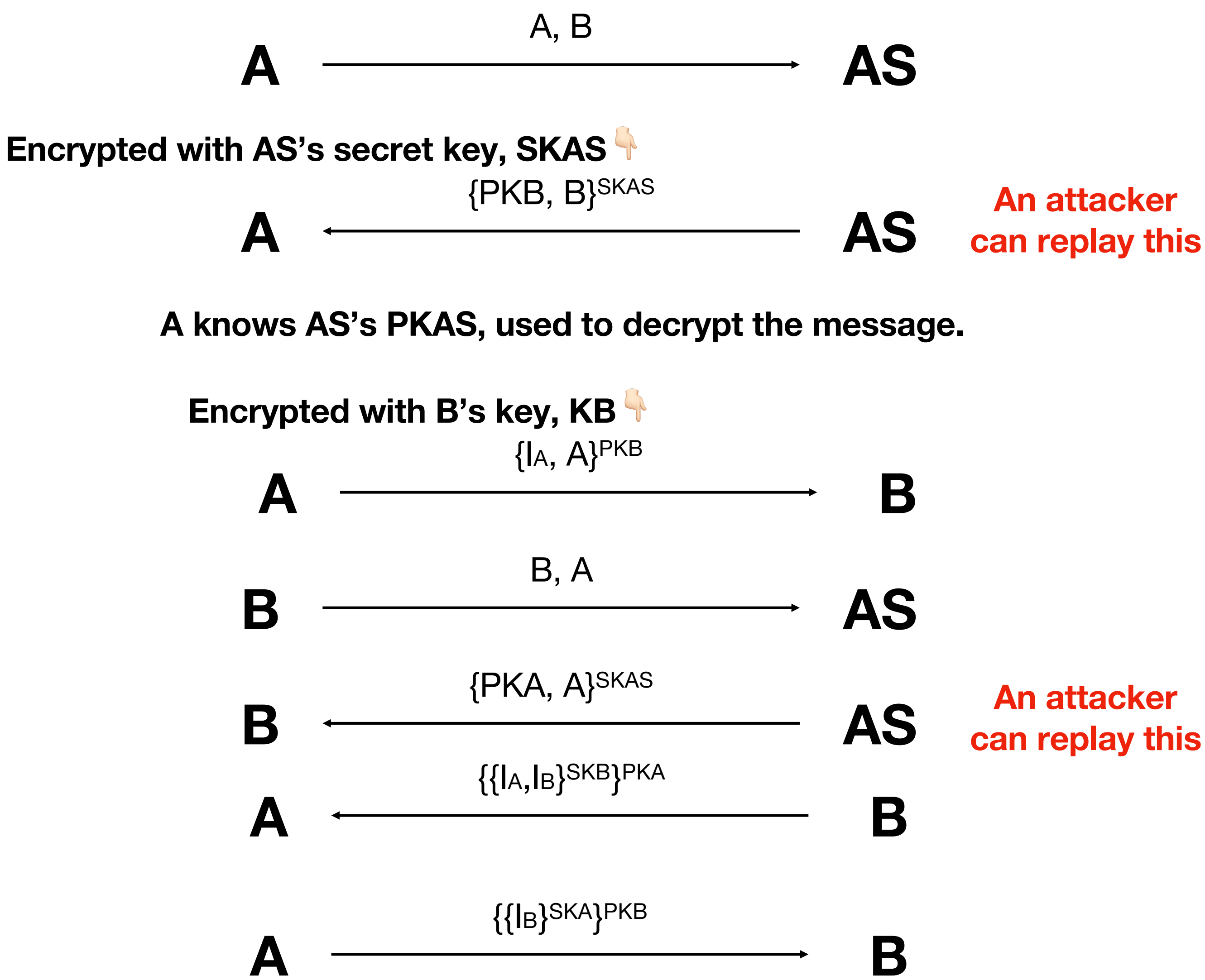
Goal: using K_{ab} derive K'_{ab}

Guarantees:

1. **B knows K'_{ab} is fresh and is only shared with A**
2. **A believes B mentioned K'_{ab} along with a nonce N'_b**
3. **But, is K'_{ab} fresh?!**



Fix: Add N_a to last message.



Needham and Shroeder Protocol

Authentication of heterogeneous systems

- A single authentication authority provides good security but cannot be practical for very large networks covering vast areas.
- There is a need for global trust in an environment that principals do not trust one another.
- Two desired properties:
  - knowing who created a message (authentication and integrity), and
  - knowing who can read a message you create (confidentiality).
- Therefore, we need a “secure channel” to “communicate securely”.

[Gasser et al. 1989]

Message Auth Code

- Hash functions
  - A checksum of a message m, created using a one-way encryption function, H(m).
  - It is computationally infeasible to have $H(m_1) = c$ and $H(m_2) = c$, where $m_1 \neq m_2$.
- Providing authentication using hash:
  - MAC: Message Authentication Code
  - Sender: SK(H(m)), m -> Receiver.
  - Receiver: PK(H(m)) = t, if t = m, then the sender is authenticated.

Message Auth Code

Assume Alice and Bob share a 🔑

🧕 ——📜, 📒——→ 👨

Bob can use the key to authenticate the message m.

Alice: (🔑 || 📜) => 📒

Bob: (🔑 || 📜) => 📔; if 📒 == 📔, then accept 📜

Authentication with passwords

- Main method of authentication
- Prone to phishing and dictionary attacks
- Are password managers secure?
- Usability + Security -> SSO

What is Single Sign-On?

- Instead of a username/password combination for each website, use "one" account with an identity provider to login into a service provider.

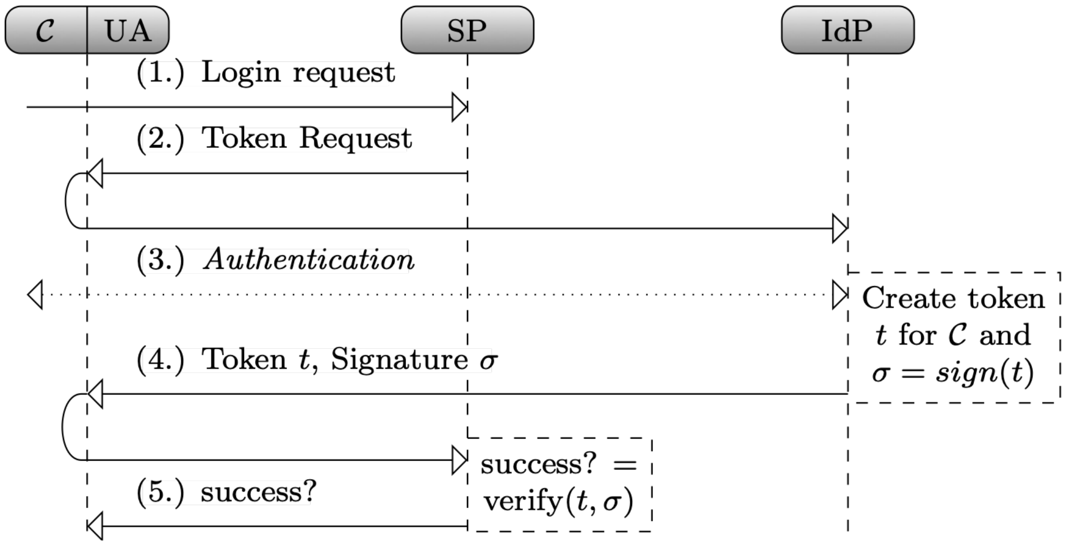


Figure 1: Single Sign-On (SSO) overview.

Source: Mainka et al. (2016)

OpenID

Adds IdP discovery and association (establishing a shared key on the fly)

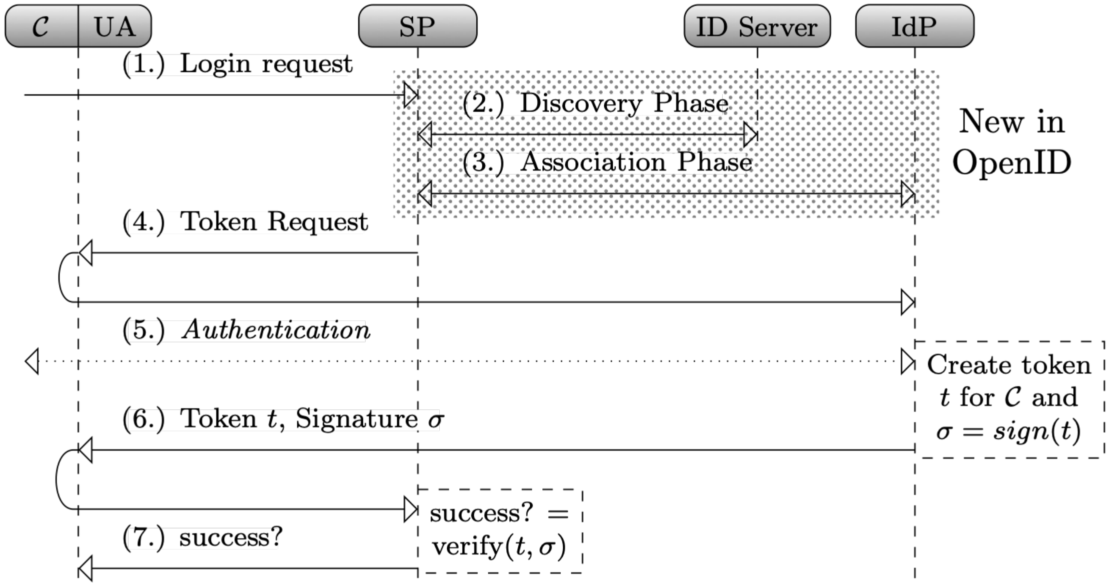


Figure 2: OpenID overview.

Source: Mainka et al. (2016)

OpenID

Source: Mainka et al. (2016)

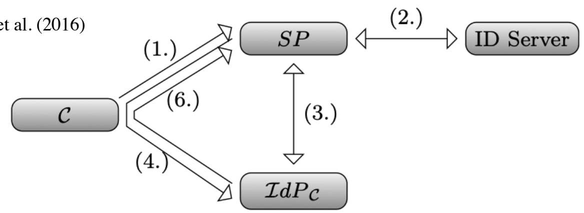


Figure 5: The OpenID protocol simplified in 5 steps. Steps are numbered according to Figure 2.

1. Discovery: C starts by sending URL.ID$_c$, e.g. http://idp1.com/alice to SP. SP fetches the URL.IDc on the IdP website.

2. Association: SP uses discovered IdP and performs a key exchange with it. The shared key is saved using a random value chosen by the IdP.

3. Token Processing: SP responds to C, redirecting it to IdP. IdP gives a token to C. Then, IdP redirects C to SP with the token information.

Malicious IdPs can harm others

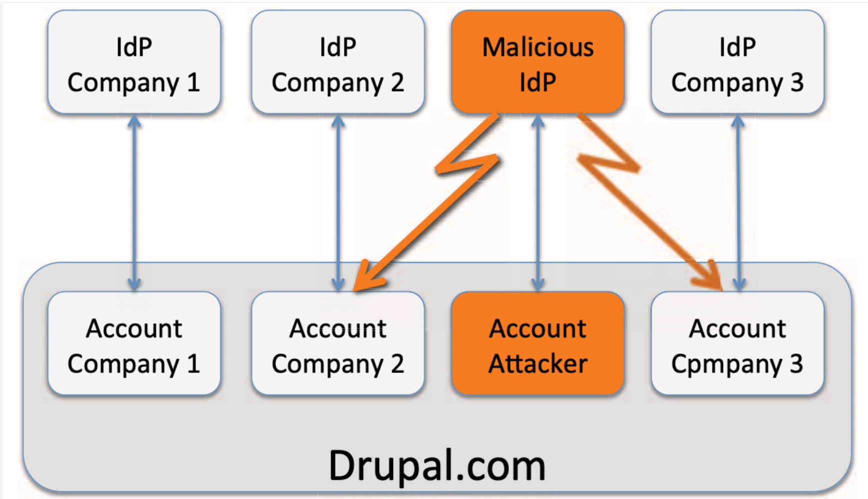


Figure 4: OpenID based SPs like Drupal allow a malicious IdP to compromise all other compartments.

Source: Mainka et al. (2016)

Trust in OpenID

- A certificate authority's trust is one for all.
- An untrustworthy IdP should only affect authentication of accounts associated with it.

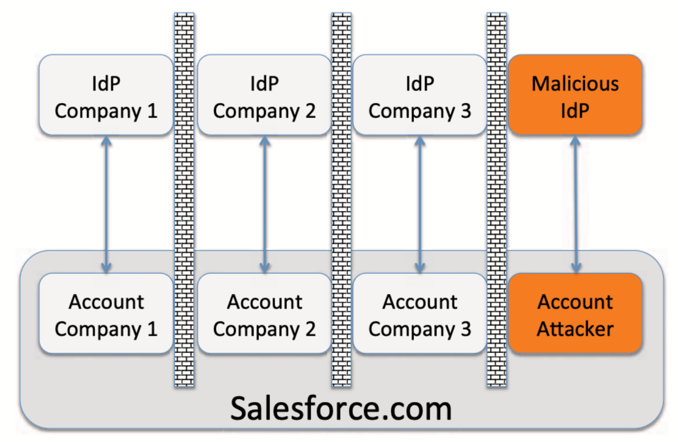


Figure 3: SPs like Salesforce enforce a strict separation between compartments. A malicious IdP can only attack its own compartment.

Source: Mainka et al. (2016)

Token Processing

- The most critical part of the protocol
- Parameters follow a structure; a parsing error is a disaster!
- Freshness of token is necessary (nonces and timestamps)
- IdP should be verified: using unique identifier, key, and signature of IdP messages

Attacks

- ID Spoofing:
  - The attacker only sets up a malicious IdP to impersonate a victim.
- Strategies

1. The malicious IdP generates a token with the user's identity and sends it to the SP. (A second discovery can fix this)

2. Discovery returns an IdP URL and a 2nd URL with victim's ID. Attacker generates a token with his ID. SP allows the attacker to log in but with the victim's ID.

3. SP uses additional user info to authenticate, such as email. Attacker issues a token with own's ID and victim's email.

Attacks

- Key confusion
  - Attacker forces a key of his choice when verifying a token at the SP's side.
  - All keys between IdP and SP are trusted. The core vulnerability is that SP uses a handle to load a key.
- Strategies

  1. Key handle is set by IdP. A malicious IdP can overwrite a legitimate key handle with its own, and confuse the SP.

  2. Key handle can be part of a token (that can be signed by a malicious IdP)
     - t_* = (URL.ID_V , URL.IdP_V , URL.SP, β), where β is a handle by the malicious IdP_V.

Attacks

- Token recipient confusion

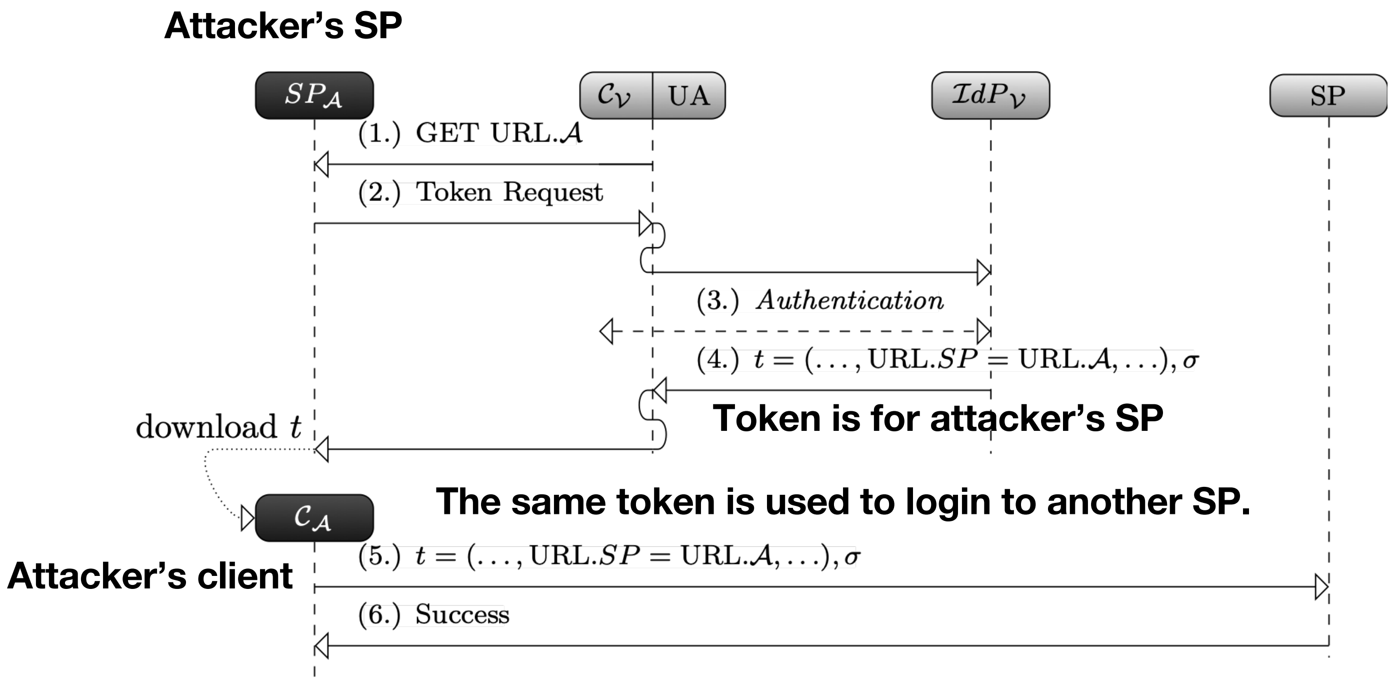


Figure 6: Token Recipient Confusion Attack.