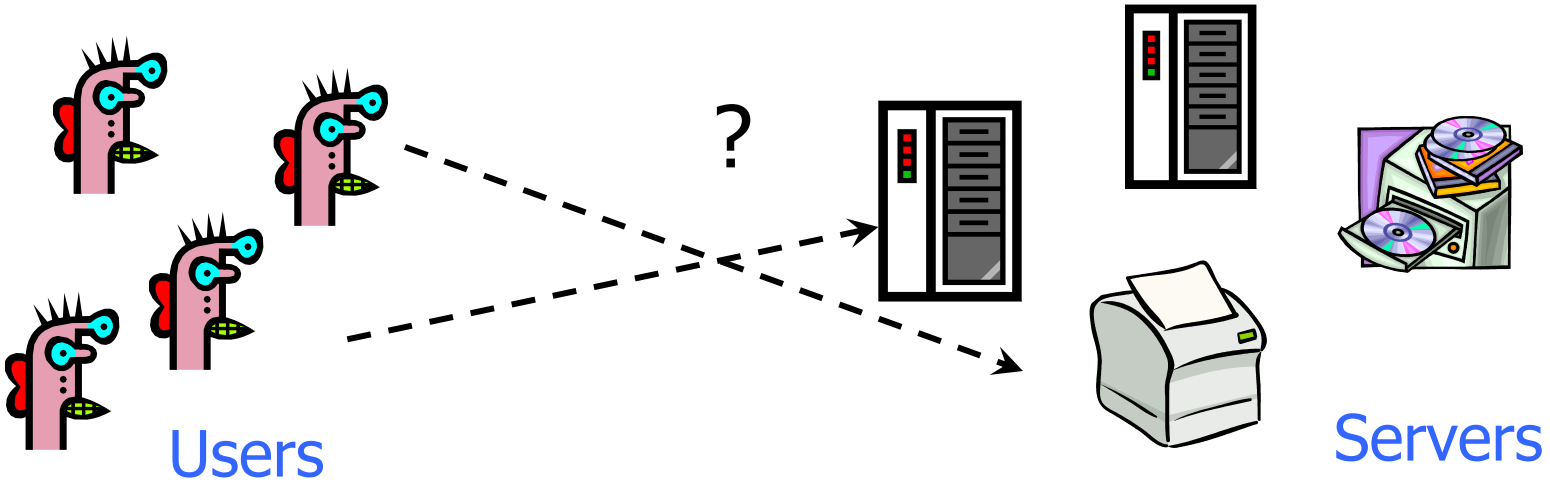
CS 5450

Authentication Protocols

Vitaly Shmatikov

Many-to-Many Authentication



How do users prove their identities when requesting services from machines on the network?

Naïve solution: every server knows every user's password

- Insecure: break into one server ⇒ compromise all users
- Inefficient: to change password, user must contact every server

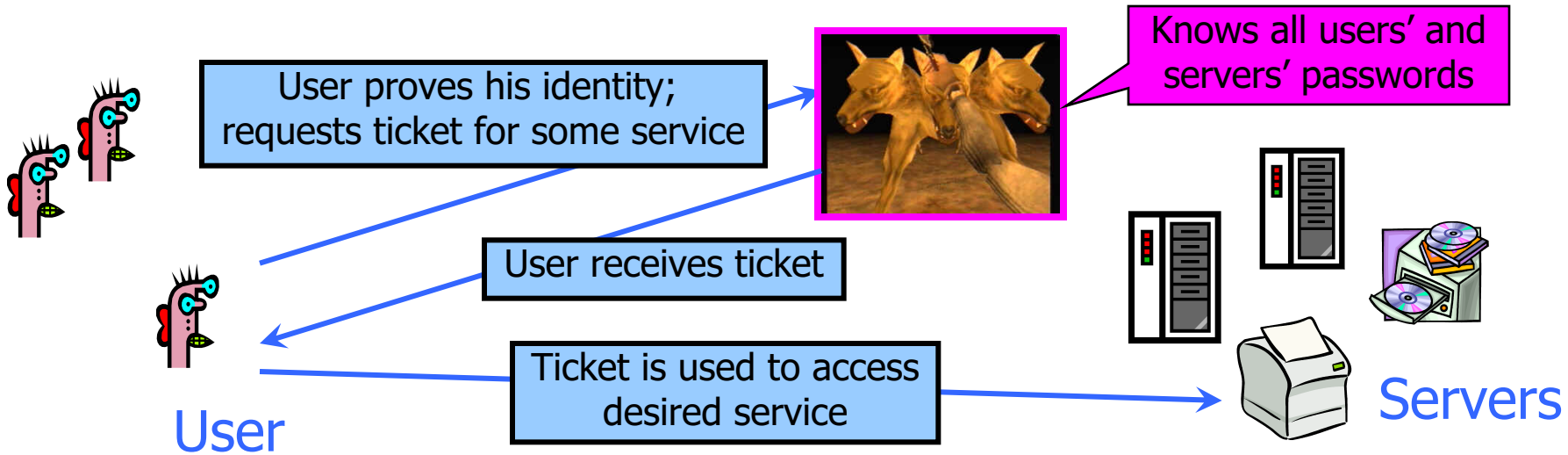
Requirements

- Security
  - … against attacks by passive eavesdroppers and actively malicious users
- Transparency
  - Users shouldn’t notice authentication taking place
  - Entering password is Ok, if done rarely
- Scalability
  - Large number of users and servers

Threats

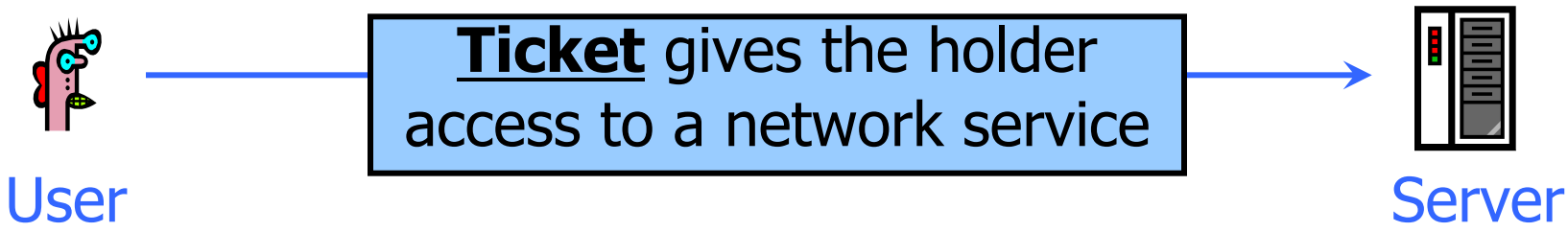
- User impersonation
  - Malicious user with access to a workstation pretends to be another user from the same workstation
- Network address impersonation
  - Malicious user changes network address of his workstation to impersonate another workstation
- Eavesdropping, tampering, replay
  - Malicious user eavesdrops, tampers, or replays other users' conversations to gain unauthorized access

Solution: Trusted Third Party



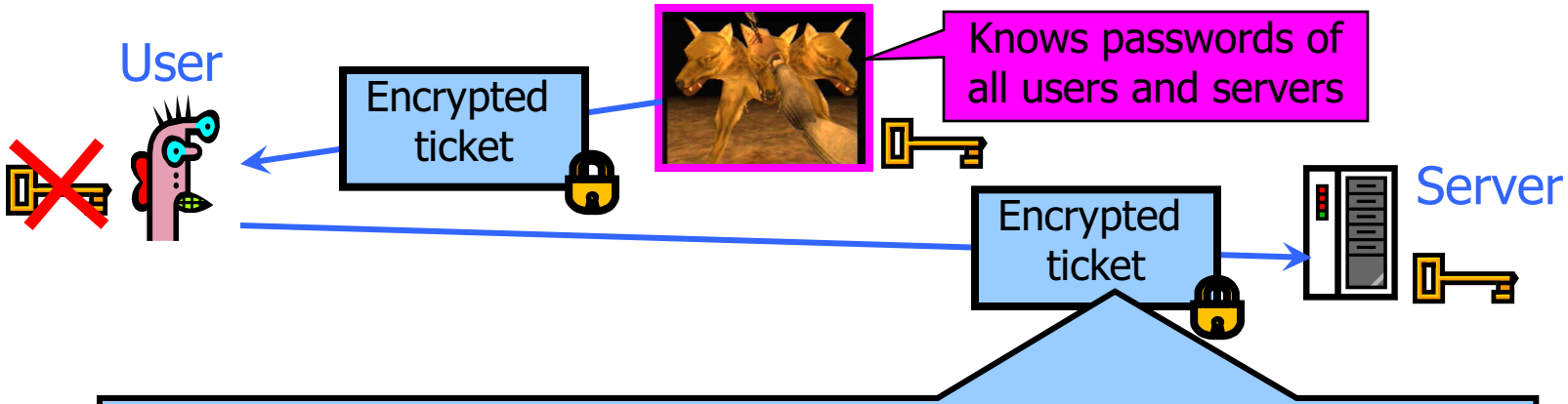
- Trusted authentication service on the network
  - Knows all passwords, can grant access to any server
  - Convenient (but also the single point of failure!)
  - Requires high level of physical security

What Should a Ticket Look Like?



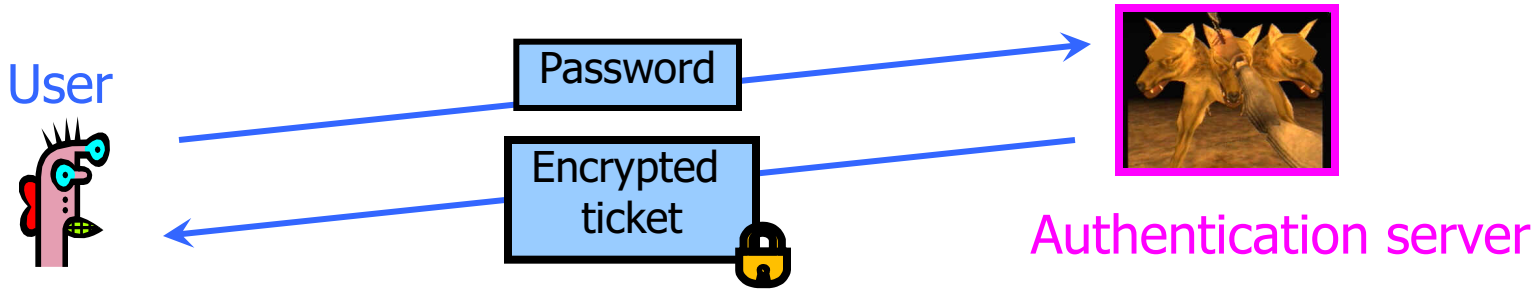
- User should not be able to access server without first proving his identity to authentication service
- Ticket proves that user has authenticated
  - Authentication service encrypts some information with a key known to the server (but not the user!)
    - The only thing the user can do is pass the ticket to the server
    - Hash functions would've worked well, but this is 1980s design
  - Server decrypts the ticket and verifies information

What Should a Ticket Include?



- User name
- Server name
- Address of user's workstation
  - Otherwise, a user on another workstation can steal the ticket and use it to gain access to the server
- Ticket lifetime
- A few other things (session key, etc.)

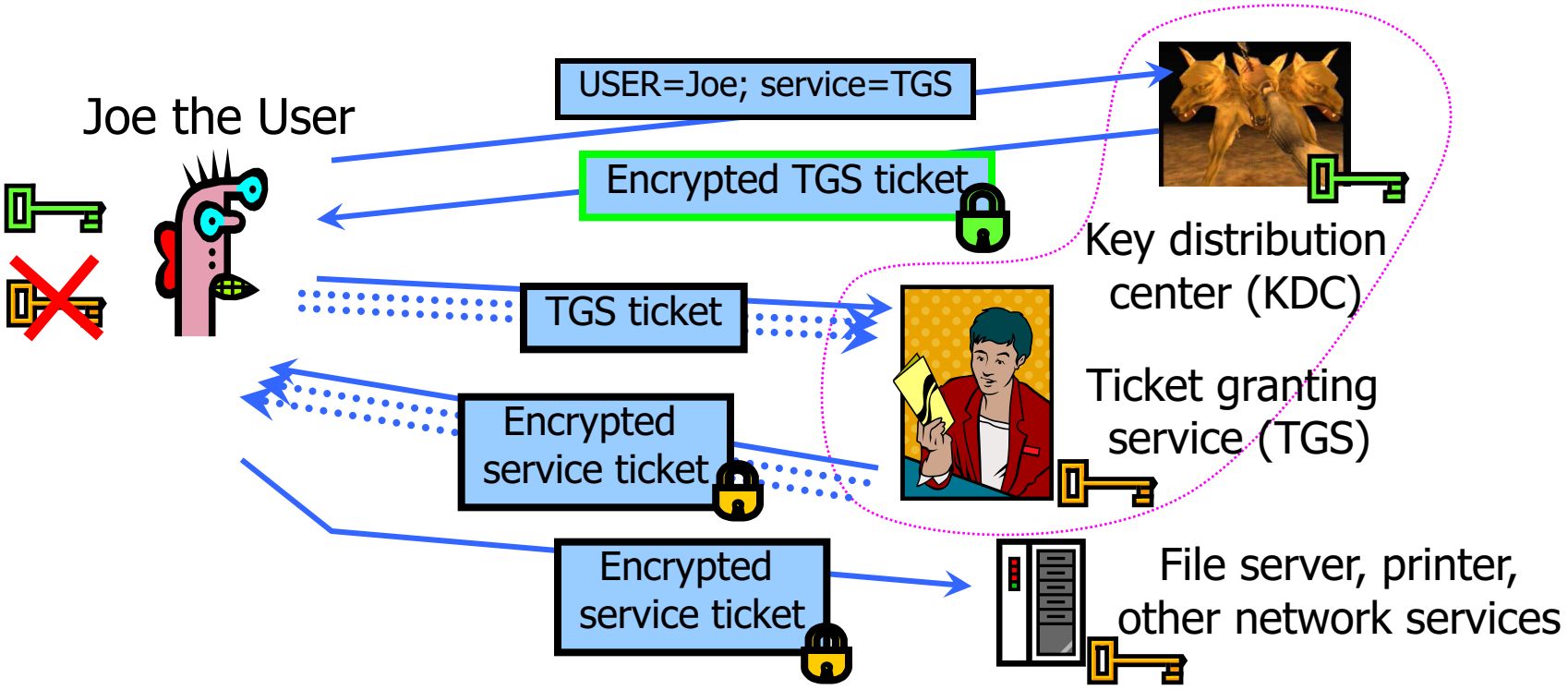
Naïve Authentication



- Insecure: passwords are sent in plaintext
  - Eavesdropper can steal the password and later impersonate the user to the authentication server
- Inconvenient: need to send the password each time to obtain the ticket for any network service
  - Separate authentication for email, printing, etc.

Two-Step Authentication

- Prove identity once to obtain a special TGS ticket
- Use TGS to get tickets for any network service



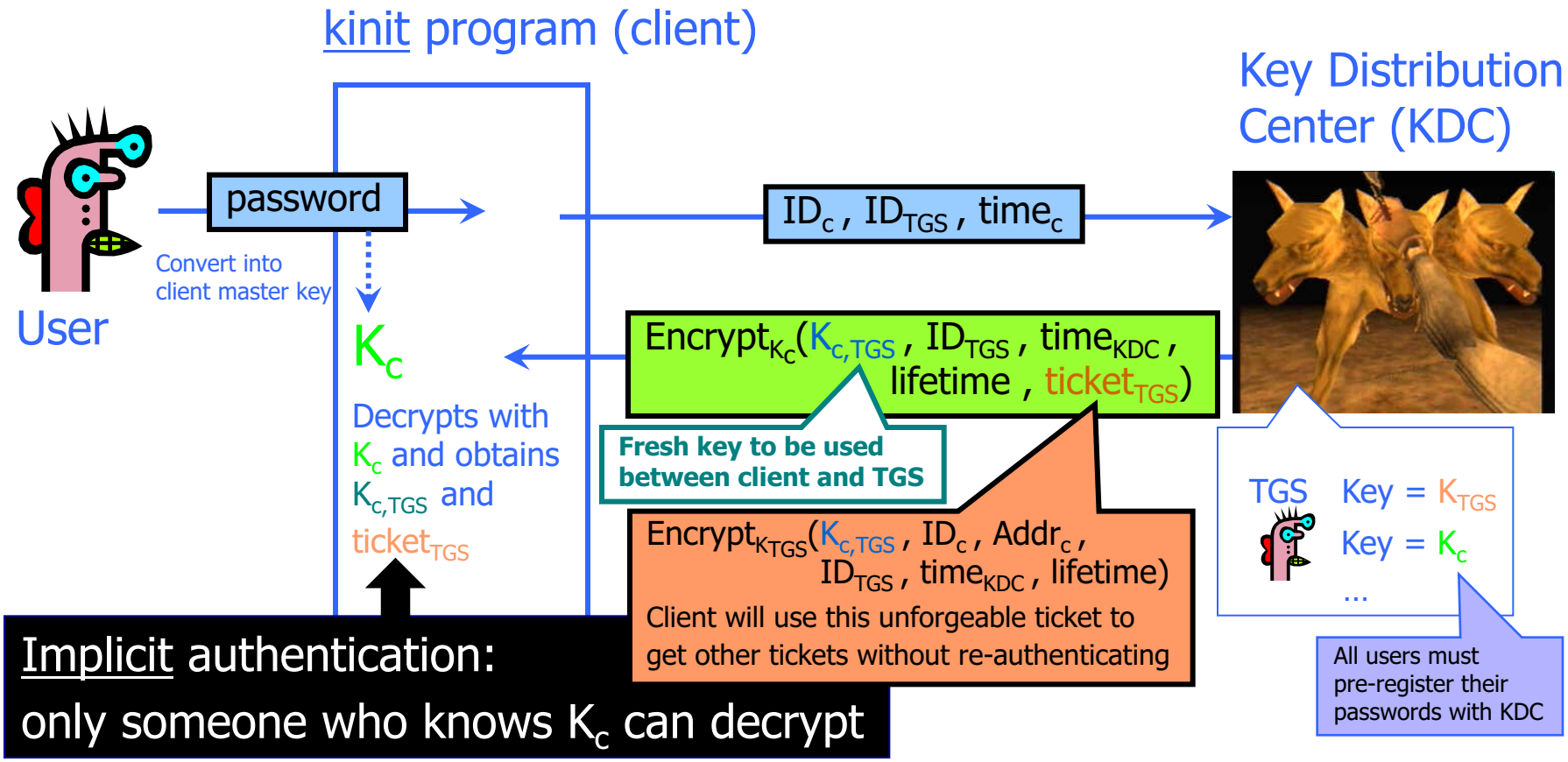
Threats

- Ticket hijacking
  - Malicious user may steal the service ticket of another user on the same workstation and try to use it
    - Network address verification does not help
  - Servers must verify that the user who is presenting the ticket is the same user to whom the ticket was issued
- No server authentication
  - Attacker may misconfigure the network so that he receives messages addressed to a legitimate server
    - Capture private information from users and/or deny service
  - Servers must prove their identity to users

Symmetric Keys in Kerberos

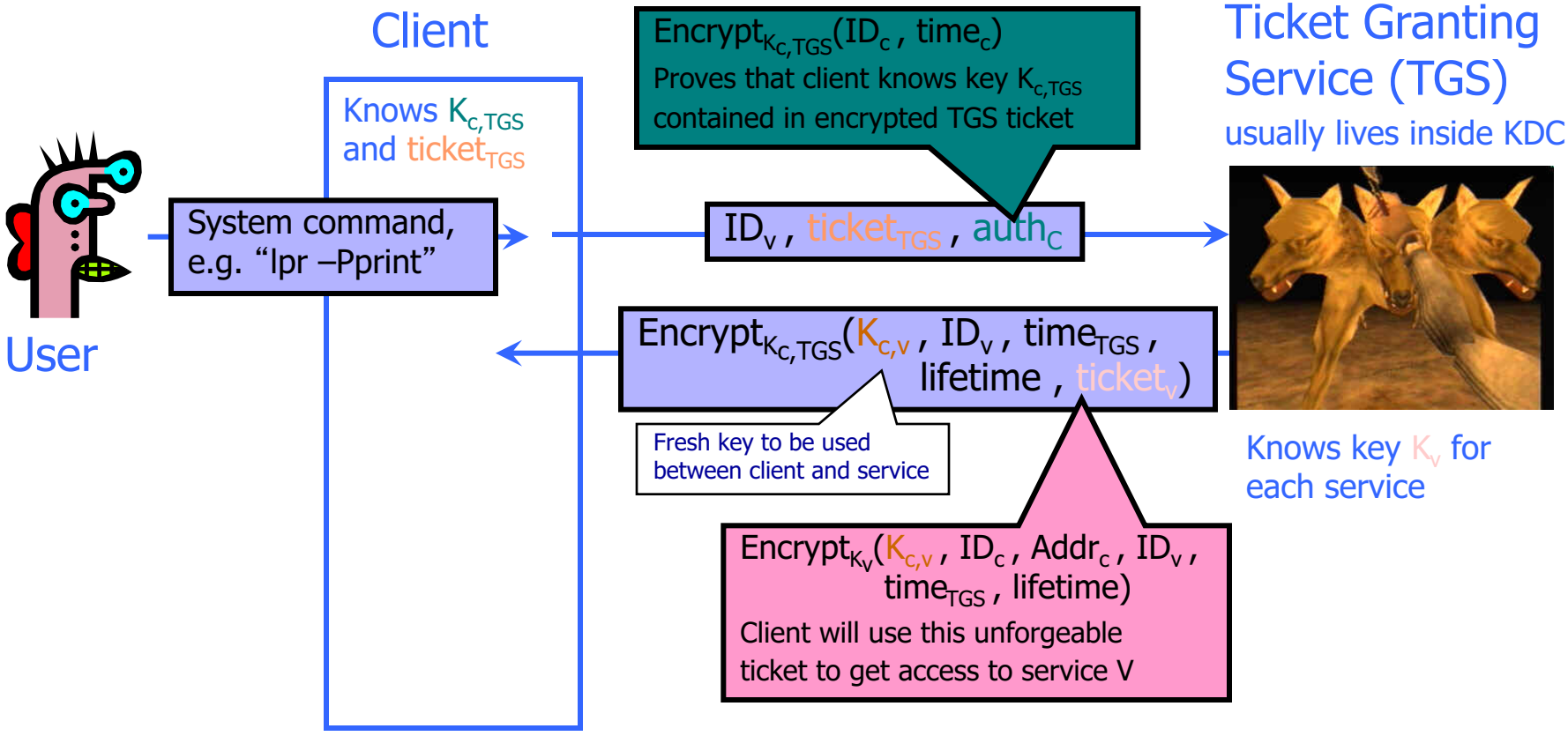
- K_c is long-term key of client C
  - Derived from the user's password
  - Known to the client and the key distribution center (KDC)
- K_{TGS} is long-term key of TGS
  - Known to KDC and the ticket granting service (TGS)
- K_v is long-term key of network service V
  - Known to V and TGS; each service V has its own long-term key
- $K_{c,TGS}$ is short-term session key betw. C and TGS
  - Created by KDC, known to C and TGS
- $K_{c,v}$ is short-term session key between C and V
  - Created by TGS, known to C and V

"Single Logon" Authentication



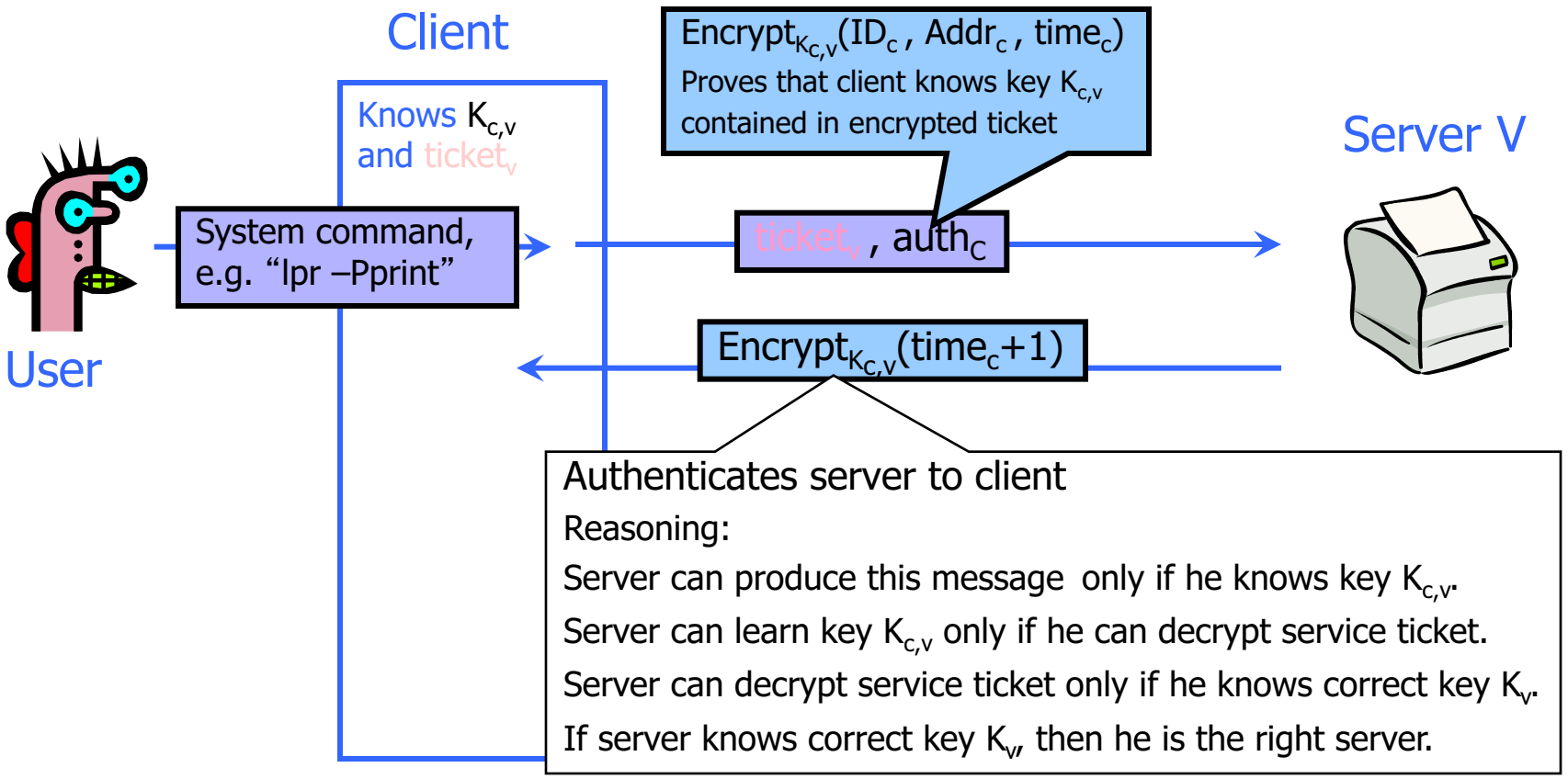
- Client only needs to obtain TGS ticket **once** (say, every morning)
- Ticket is encrypted; client cannot forge it or tamper with it

Obtaining a Service Ticket



- Client uses TGS ticket to obtain a service ticket and a short-term session key for each network service (printer, email, etc.)

Obtaining Service

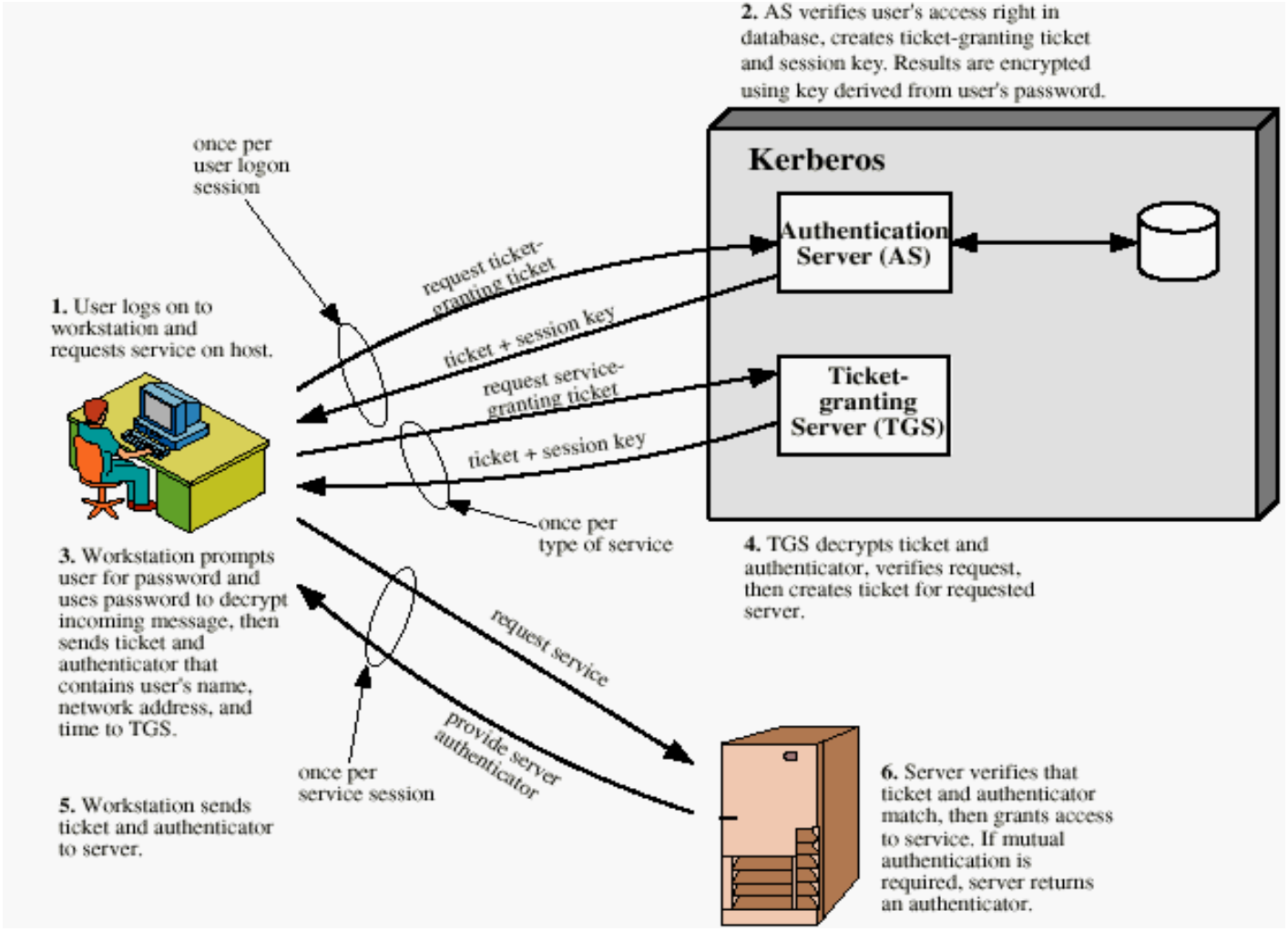


- For each service request, client uses the short-term key for that service and the ticket he received from TGS

Kerberos in Large Networks

- One KDC isn't enough for large networks (why?)
- Network is divided into realms
  - KDCs in different realms have different key databases
- To access a service in another realm, users must…
  - Get ticket for home-realm TGS from home-realm KDC
  - Get ticket for remote-realm TGS from home-realm TGS
    - As if remote-realm TGS were just another network service
  - Get ticket for remote service from that realm's TGS
  - Use remote-realm ticket to access service
  - N(N-1)/2 key exchanges for full N-realm interoperation

Summary of Kerberos



Important Ideas in Kerberos

- Short-term session keys
  - Long-term secrets used only to derive short-term keys
  - Separate session key for each user-server pair
    - Re-used by multiple sessions between same user and server
- Proofs of identity based on authenticators
  - Client encrypts his identity, addr, time with session key; knowledge of key proves client has authenticated to KDC
    - Also prevents replays (if clocks are globally synchronized)
  - Server learns this key separately (via encrypted ticket that client can't decrypt), verifies client's authenticator
- Symmetric cryptography only

Kerberos Version 5

- Better user-server authentication
  - Separate subkey for each user-server session instead of re-using the session key contained in the ticket
  - Authentication via subkeys, not timestamp increments
- Authentication forwarding (delegation)
  - Servers can access other servers on user's behalf, eg, can tell printer to fetch email
- Realm hierarchies for inter-realm authentication
- Explicit integrity checking + standard CBC mode
- Multiple encryption schemes, not just DES

Practical Uses of Kerberos

- Microsoft Windows
- Email, FTP, network file systems, many other applications have been kerberized
  - Use of Kerberos is transparent for the end user
  - Transparency is important for usability!
- Local authentication
  - login and su in OpenBSD
- Authentication for network protocols
  - rlogin, rsh
- Secure windowing systems

What Is SSL / TLS?

- Secure Sockets Layer and Transport Layer Security protocols
  - Same protocol design, different crypto algorithms
- De facto standard for Internet security
  - "The primary goal of the TLS protocol is to provide privacy and data integrity between two communicating applications"
- Deployed in every Web browser; also VoIP, payment systems, distributed systems, etc.

SSL / TLS Guarantees

- End-to-end secure communications in the presence of a network attacker
  - Attacker completely 0wns the network: controls Wi-Fi, DNS, routers, his own websites, can listen to any packet, modify packets in transit, inject his own packets into the network
- Scenario: you are reading your email from an Internet café connected via a r00ted Wi-Fi access point to a dodgy ISP in a hostile authoritarian country

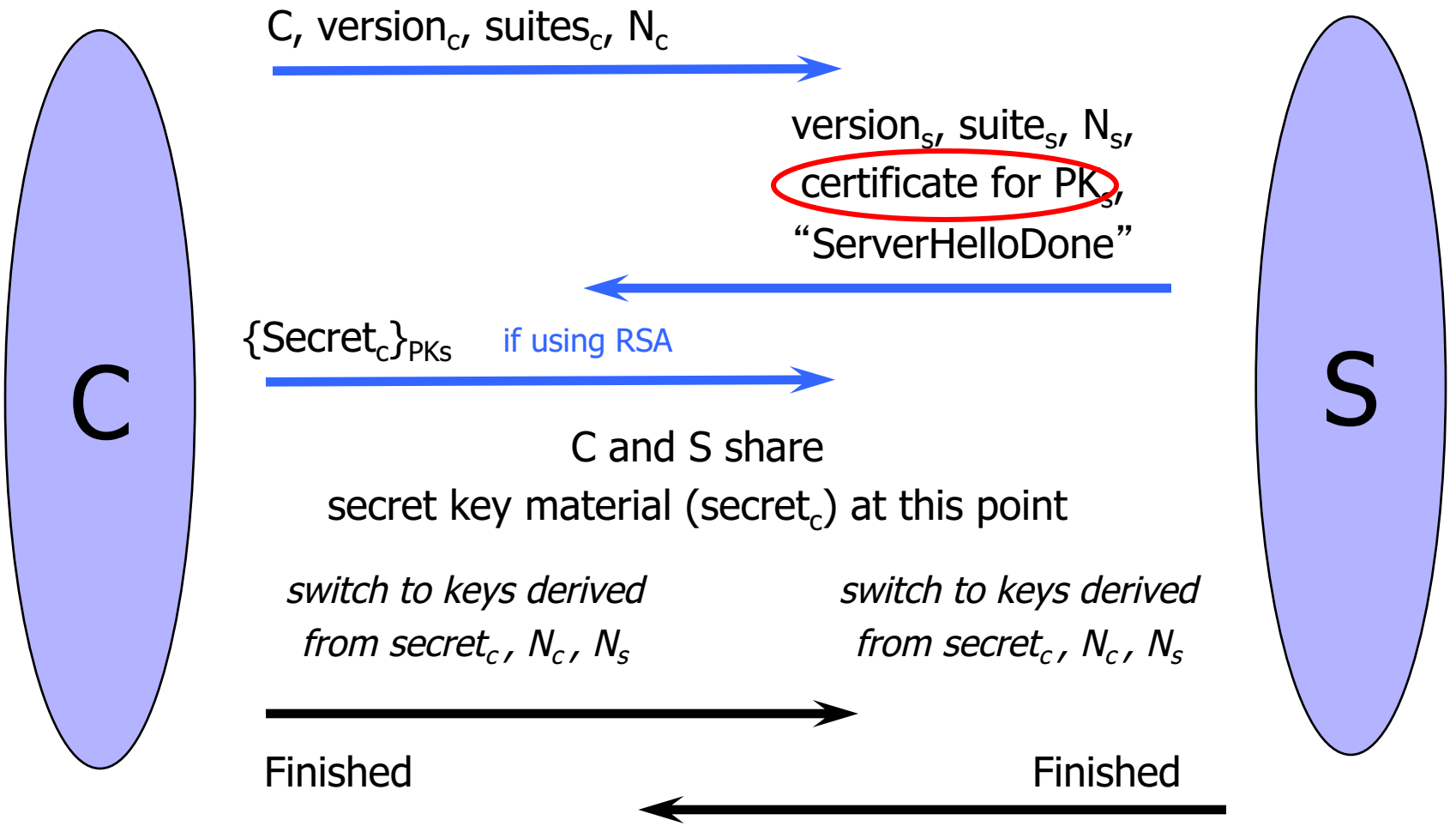
SSL Basics: Two Protocols

- Handshake protocol
  - Uses public-key cryptography to establish several shared secret keys between the client and the server
- Record protocol
  - Uses the secret keys established in the handshake protocol to protect confidentiality, integrity, and authenticity of data exchange between the client and the server

SSL Handshake Protocol

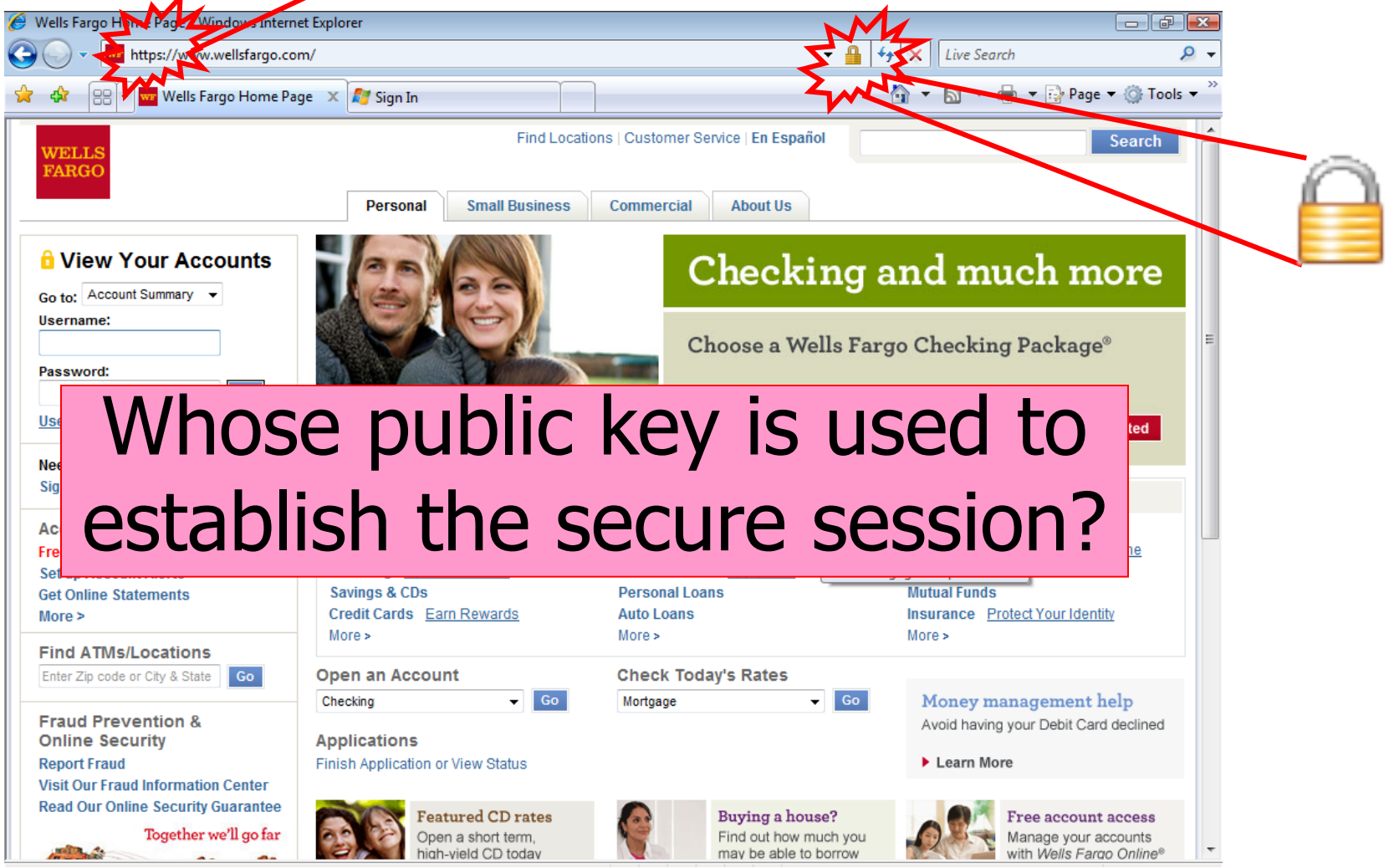
- Runs between a client and a server
  - For example, client = Web browser, server = website
- Negotiate version of the protocol and the set of cryptographic algorithms to be used
  - Interoperability between different implementations
- Authenticate server and client (optional)
  - Use digital certificates to learn each other's public keys and verify each other's identity
  - Often only the server is authenticated
- Use public keys to establish a shared secret

“Core” SSL Handshake



Motivation

https://

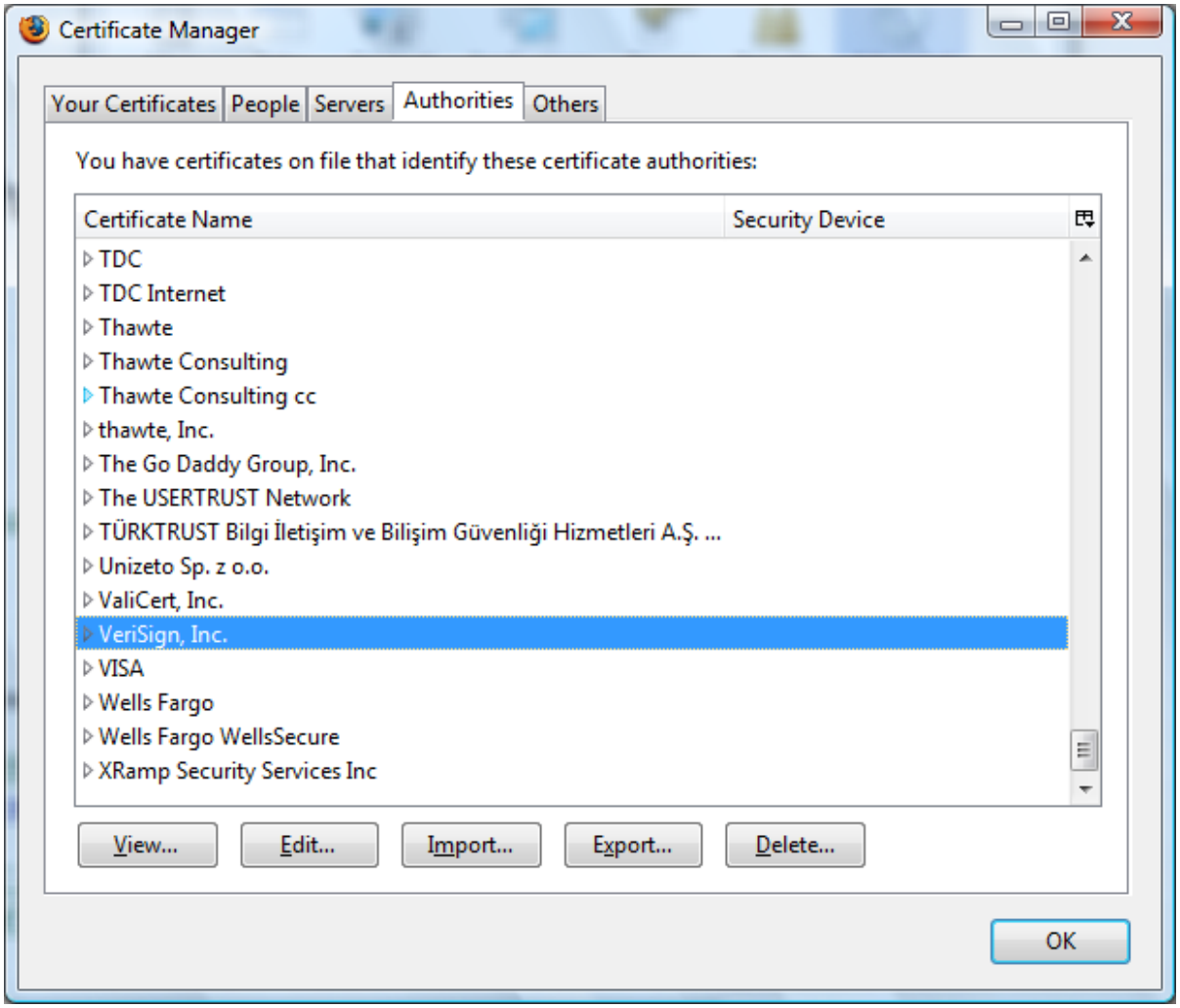


Whose public key is used to establish the secure session?

Distribution of Public Keys

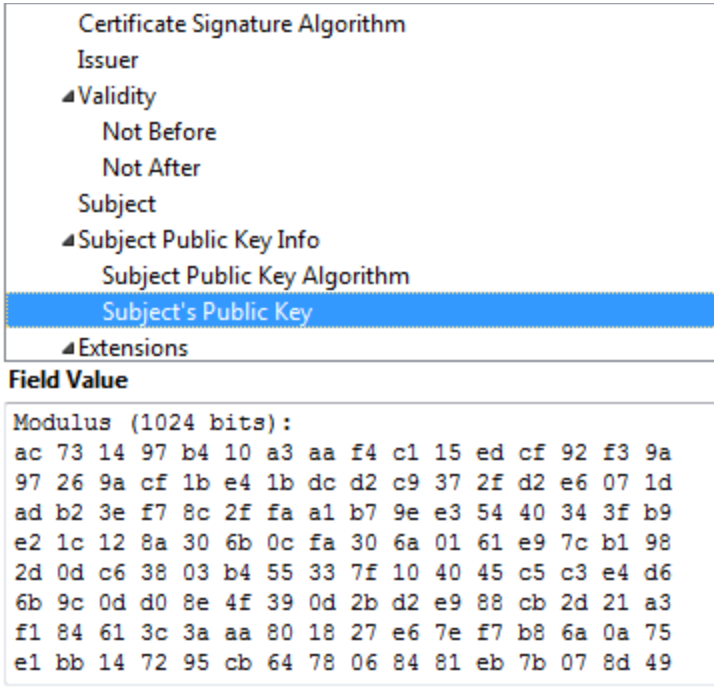
- Public-key certificate
  - Signed statement specifying the key and identity
    - sig_{Alice}(“Bob”, PK_B)
- Common approach: certificate authority (CA)
  - An agency responsible for certifying public keys
  - Browsers are pre-configured with 100+ of trusted CAs
  - A public key for any website in the world will be accepted by the browser if certified by one of these CAs

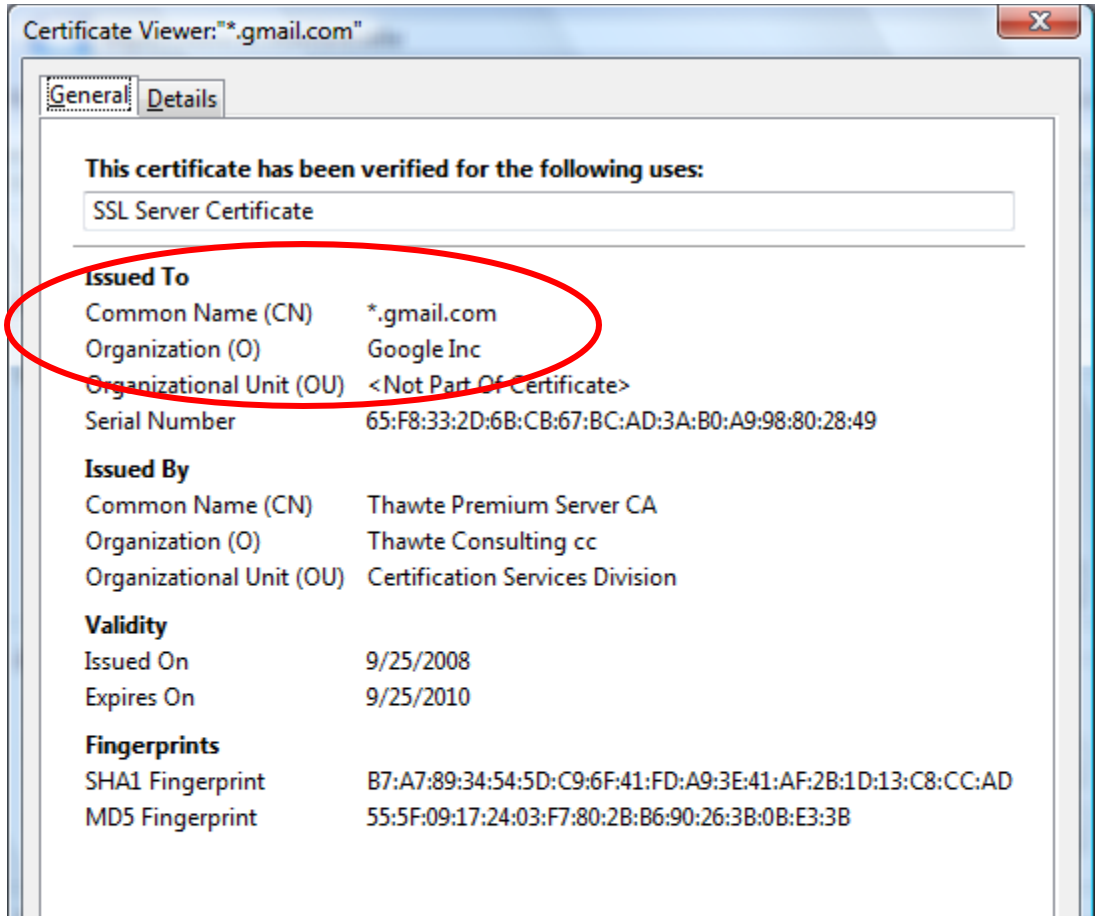
Trusted Certificate Authorities



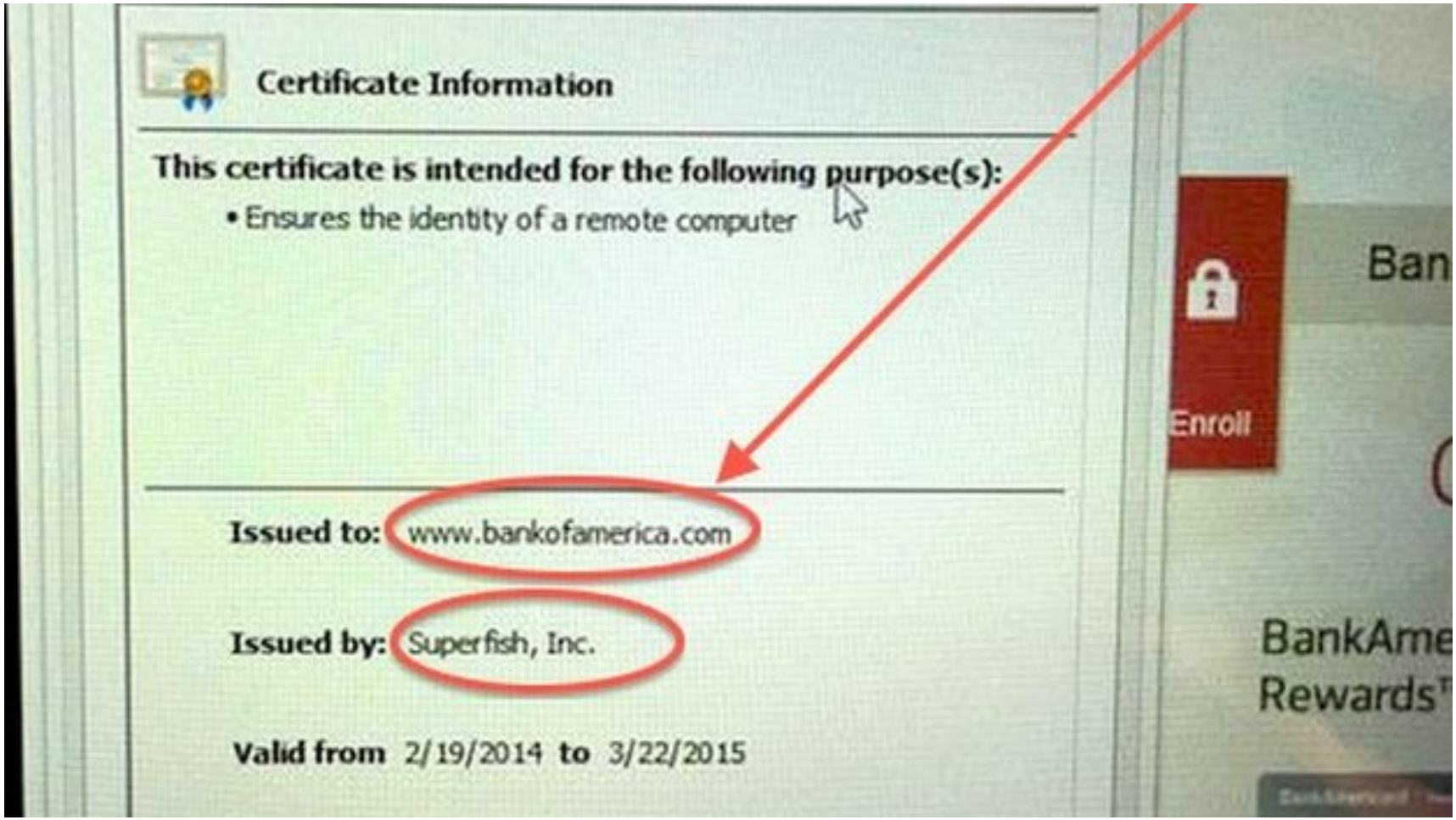
Example of a Certificate

Important fields





Another Example of a Certificate



Root Certificates in Lenovo

In the news

Lenovo hit by lawsuit over Superfish adware
CNET - 3 days ago
Sarah Tew/CNET. **Lenovo** may find itself in a courtroom over its **Superfish adware** fiasco.

Interview with Lenovo's CTO will scare anyone still thinking of buying a Lenovo product
BGR - 2 days ago

Lenovo's Chief Technology Officer Discusses the Superfish Adware Fiasco - NYTimes.com
Bits - The New York Times - 3 days ago

More news for lenovo superfish adware

Lenovo Sued Over Superfish Adware : NPR
www.npr.org › News › Business NPR
2 days ago - Renee Montagne talks to Jordan Robertson of Bloomberg News about computer maker **Lenovo**, which allowed controversial spyware to be ...

Lenovo users lawyer up over hole-filled, HTTPS-breaking ...
arstechnica.com/.../**lenovo**-users-lawyer-up-over-hole-filled... Ars Technica
4 days ago - In the wake of last week's **Lenovo's Superfish** debacle, at least one ... and that **Superfish adware** "does not present a security risk," despite ...

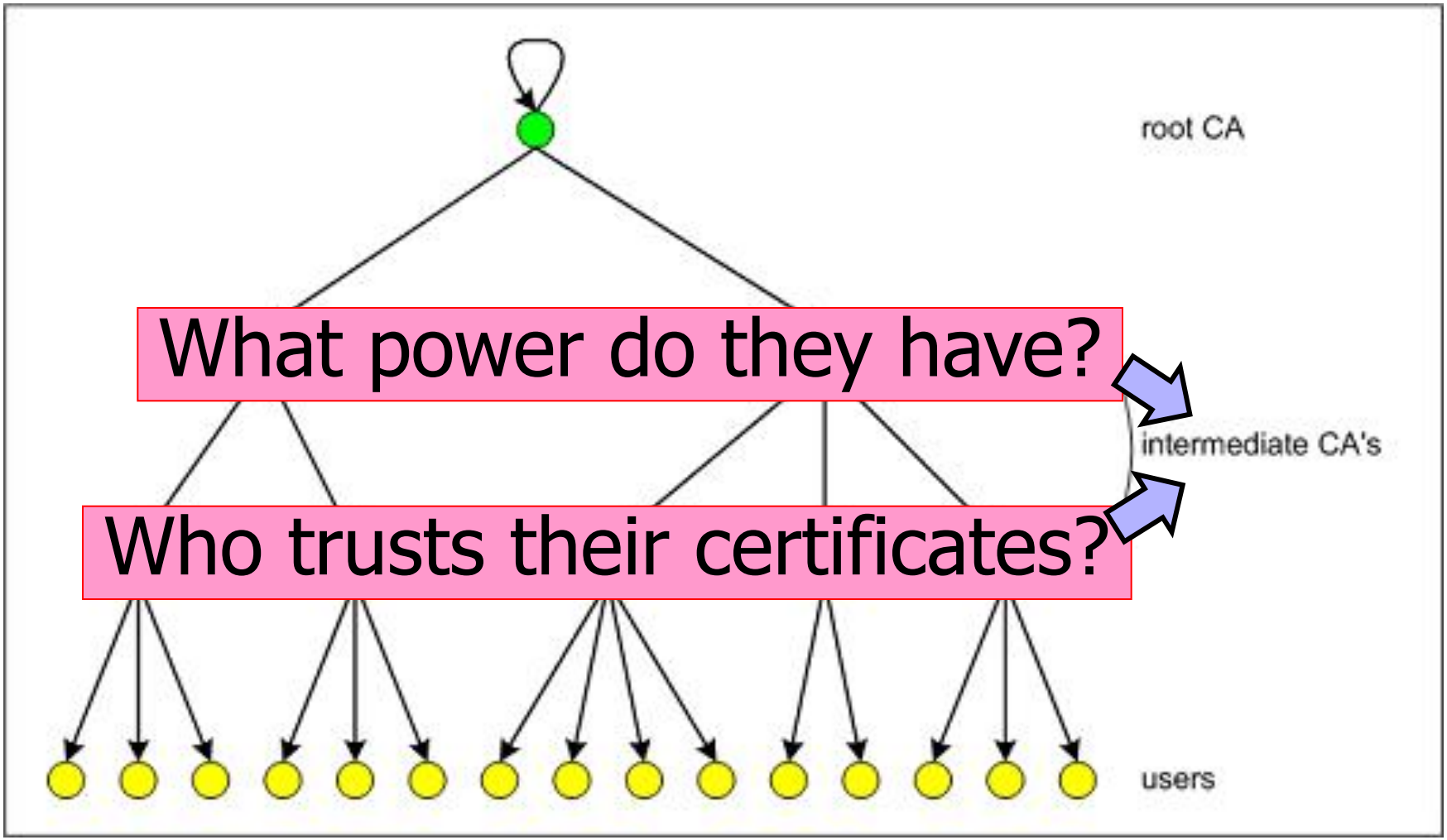
Lenovo's Chief Technology Officer Discusses the Superfish ...
bits.blogs.nytimes.com/.../**lenovos**-chief-technology-officer-discusses-the...
3 days ago - The **adware** was intended to serve **Lenovo** users targeted ads, but the company **Lenovo** partnered with to do this, **Superfish**, did so by hijacking ...

Lenovo Sued Over Superfish Adware | News & Opinion ...
www.pcmag.com › Reviews › Software › Security PC Magazine
4 days ago - Not surprisingly, the controversy over **Lenovo** installing **Superfish adware** into its consumer PCs has resulted in a lawsuit. According to the suit, ...

CA Hierarchy

- Browsers, operating systems, etc. have trusted root certificate authorities
  - My Chrome includes certificates of 195 trusted root CAs
- A Root CA signs certificates for intermediate CAs, they sign certificates for lower-level CAs, etc.
  - Certificate "chain of trust"
    - $sig_{Verisign}$("Cornell", $PK_{Cornell}$), $sig_{Cornell}$("Vitaly S.", PK_{Vitaly})
- CA is responsible for verifying the identities of certificate requestors, domain ownership

Certificate Hierarchy



Flame

- Cyber-espionage virus (2010-2012)
- Signed with a fake intermediate CA certificate accepted by any Windows Update service
  - … created using an MD5 chosen-prefix collision against an obscure Microsoft Terminal Server Licensing Service certificate that was enabled for code signing and still used MD5 hash function
  - Pre-dates public knowledge of hash collisions in MD5
    - Evidence of state-level cryptanalysis?

TurkTrust

- In Jan 2013, a rogue *.google.com certificate was issued by an intermediate CA that gained its authority from the Turkish root CA TurkTrust
  - TurkTrust accidentally issued intermediate CA certs to customers who requested regular certificates
  - Ankara transit authority used its certificate to issue a fake *.google.com certificate in order to intercept and filter SSL traffic from its network
- This rogue *.google.com certificate was trusted by every browser in the world

TrustWave

- In Feb 2012, admitted issuing an intermediate CA certificate to a corporate customer
  - Purpose: “re-sign” certificates for “data loss prevention”
  - Translation: forge certificates of third-party sites in order to spy on employees’ encrypted communications with the outside world
- Customer can now forge certificates for any site in world… and they will be accepted by any browser!
  - What if a “re-signed” certificate leaks out?
- Do other CAs do this?

Komodia

- Israeli startup
- From their website: “Our advanced SSL hijacker SDK is a brand new technology that allows you to access data that was encrypted using SSL and perform on the fly SSL decryption”
  - Installs its own root certificate
  - Goal: re-sign SSL certificates, proxy/MITM connections
- Same private key on all machines, easily extracted
  - Anyone can issue fake Komodia certificates, do man-in-the-middle attacks on any machine with Komodia

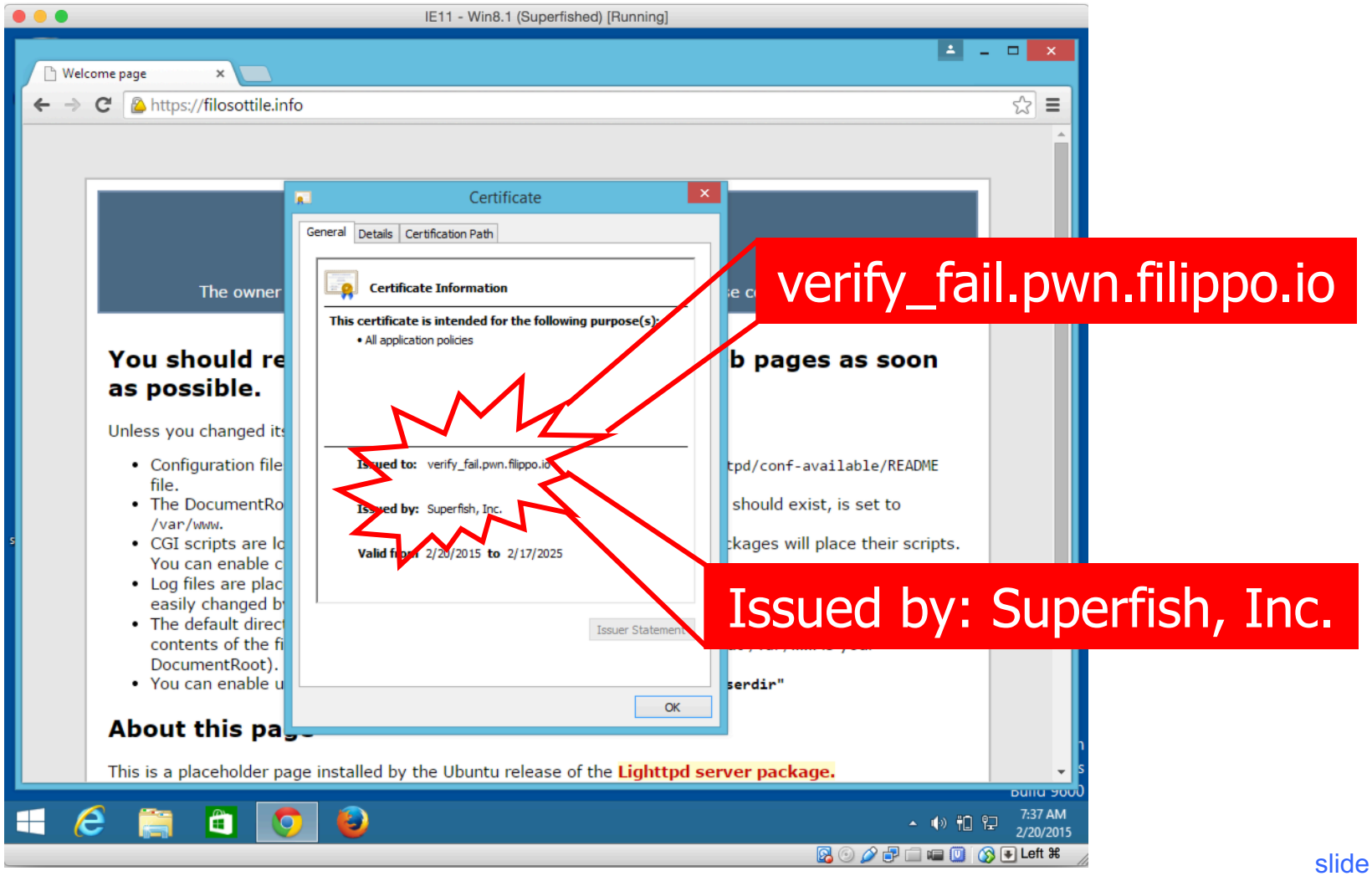
It Gets Worse

https://blog.filippo.io/komodia-superfish-ssl-validation-is-broken/

- What happens if a MITM attacker serves a self-signed certificate to a Komodia client?
- Komodia re-signs and turns it into a trusted certificate
  - But it will also change the name in the certificate, which won't match what the browser is expecting and user will see a warning - maybe not so bad
- But if attacker puts target domain into "alternate name" field, Komodia won't touch it and browser will think the certificate is completely valid

Complete SSL Fail

https://blog.filippo.io/komodia-superfish-ssl-validation-is-broken/



Statement from Superfish CEO

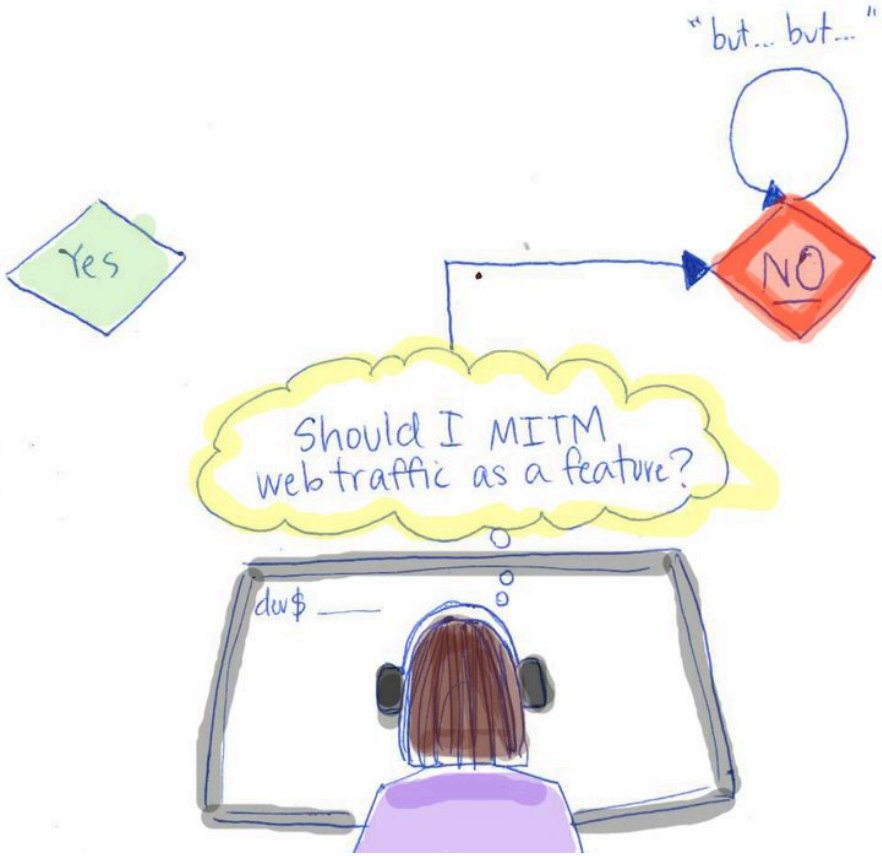
There has been significant misinformation circulating about Superfish software that was pre-installed on certain Lenovo laptops. The software shipped on a limited number of computers in 2014 in an effort to enhance the online shopping experience for Lenovo customers. Superfish's software utilizes visual search technology to help users achieve more relevant search results based on images of products they have browsed.

Despite the false and misleading statements made by some media commentators and bloggers, the Superfish software does not present a security risk. In no way does Superfish store personal data or share such data with anyone. Unfortunately, in this situation a vulnerability was introduced unintentionally by a 3rd party. Both Lenovo and Superfish did extensive testing of the solution but this issue wasn't identified before some laptops shipped. Fortunately, our partnership with Lenovo was limited in scale. We were able to address the issue quickly. The software was disabled on the server side (i.e., Superfish's search engine) in January 2015.

Not Just Komodia

- PrivDog
  - "Your privacy is under attack!"
- Provides "private Web browsing"
  - Translation: replaces ads on webpages with other ads from "trusted sources"
- Re-signs certificates to MITM SSL connections
- Accepts self-signed certificates and turns them into trusted certificates
- Founded by the CEO of Comodo CA

Just Say No

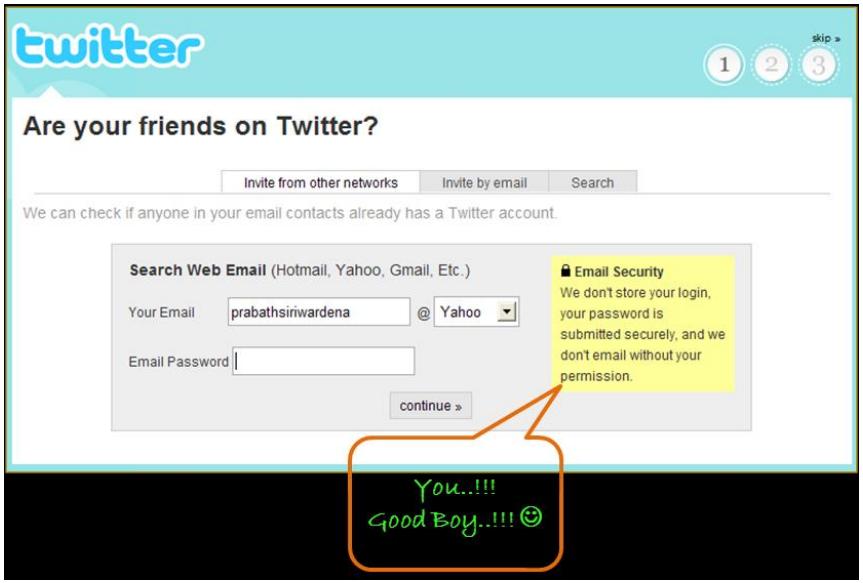


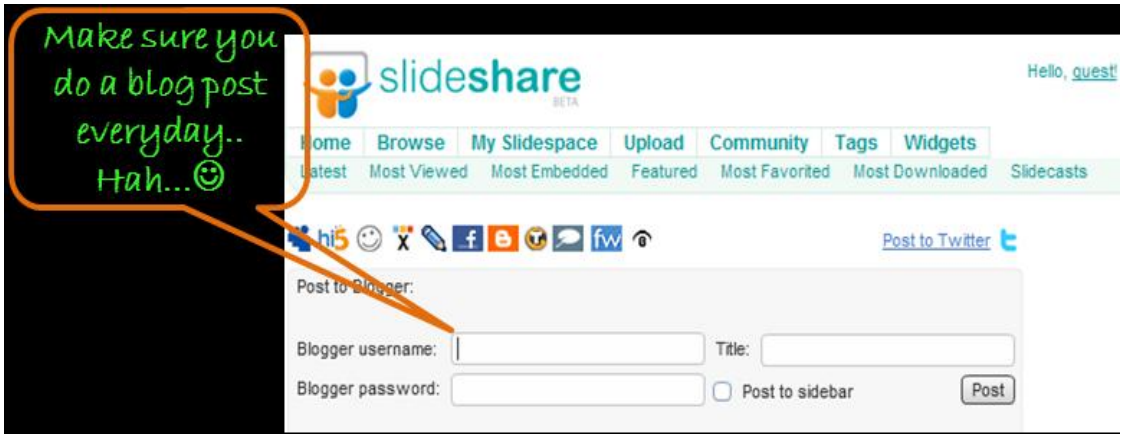
Credit: Adrienne Porter Felt (Google)

OAuth

The following slides shamelessly jacked from WS02

Before OAuth





Issues (1)

- Third-party applications are required to store the resource owner's credentials for future use, typically a password in cleartext
- Servers are required to support password authentication, despite the security weaknesses created by passwords

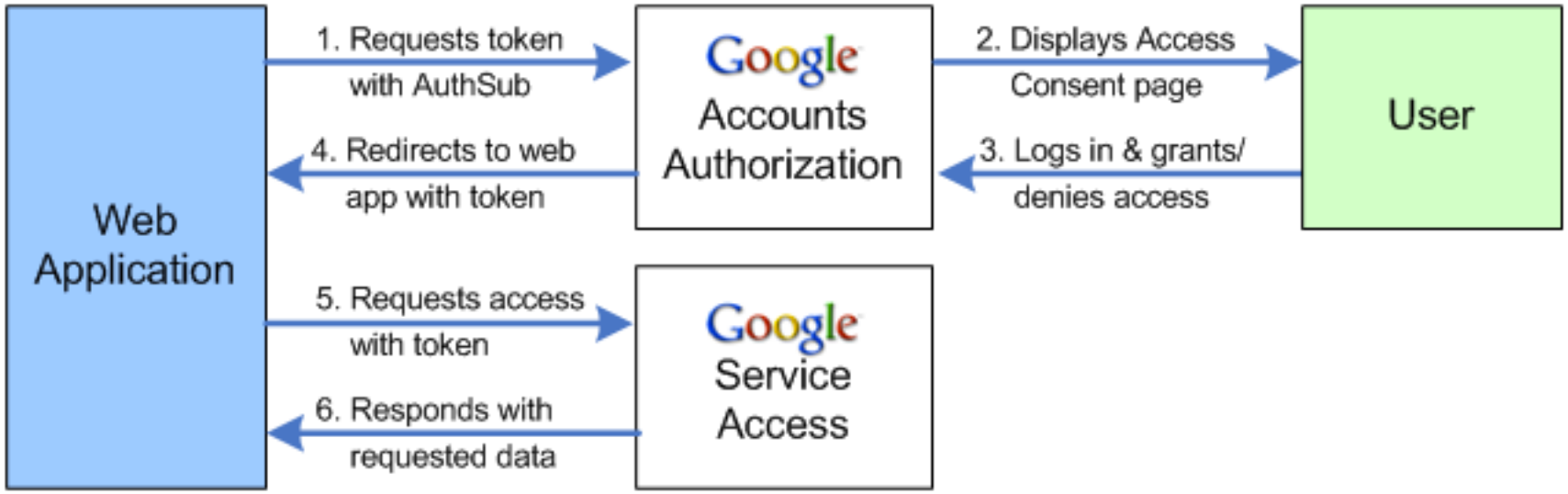
Issues (2)

- Third-party applications gain overly broad access to the resource owner's protected resources, resource owners cannot restrict duration or access to a limited subset of resources
- Resource owners cannot revoke access to an individual third party without revoking access to all third parties, and must do so by changing their password
- Compromise of any third-party application results in compromise of the end-user's password and all of the data protected by that password

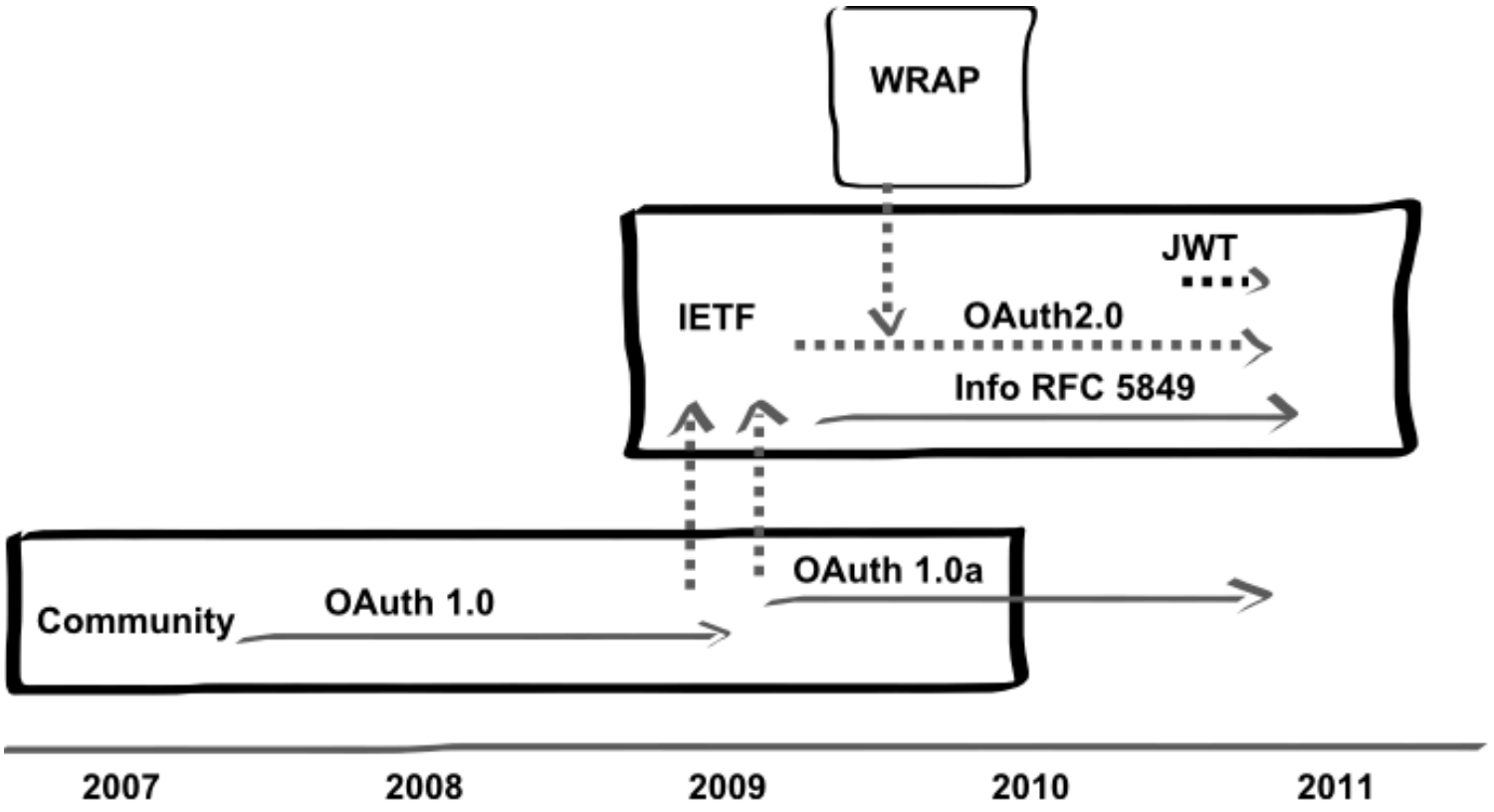
Delegation

Before OAuth

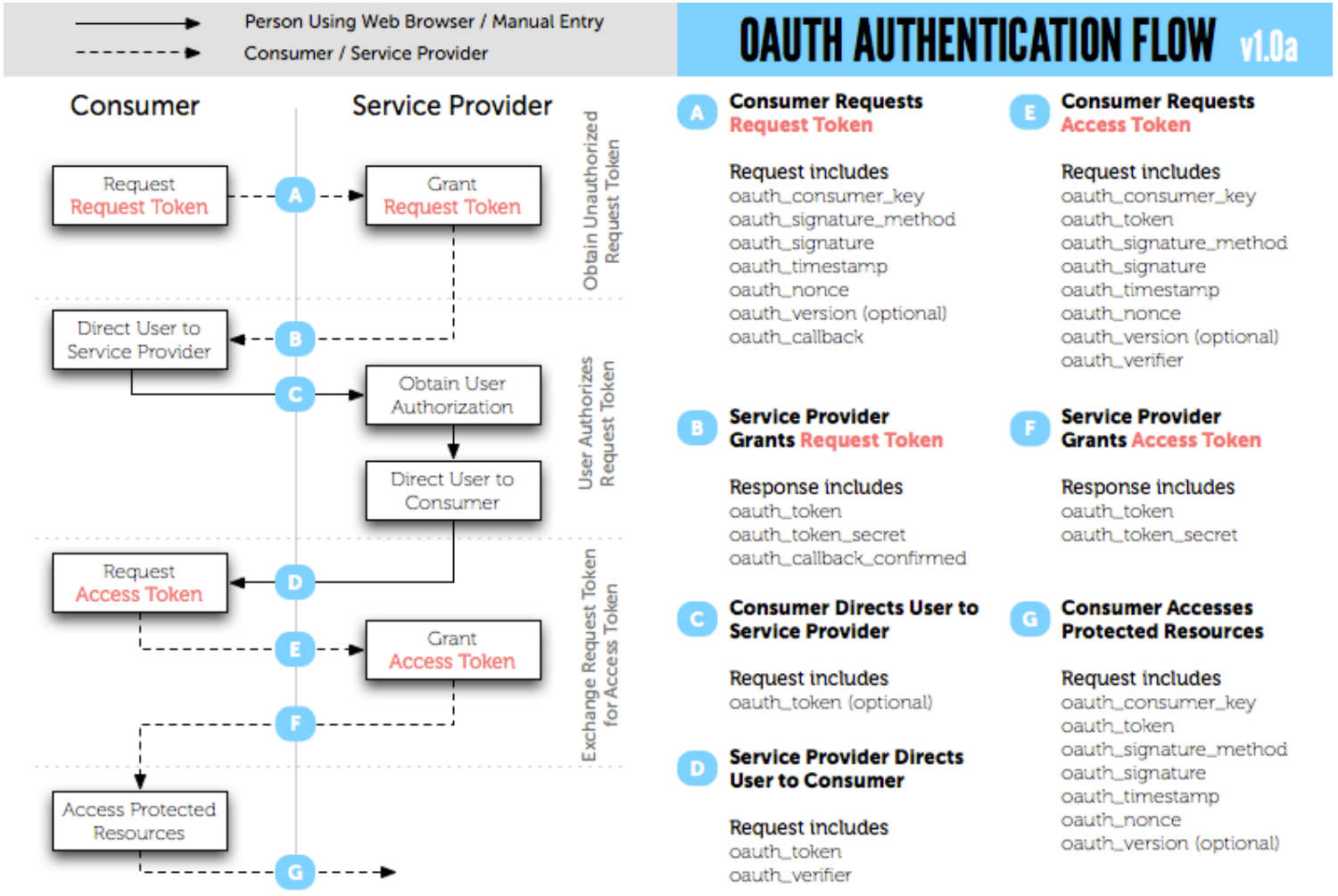
Google calendar



OAuth Evolution



OAuth 1.0a

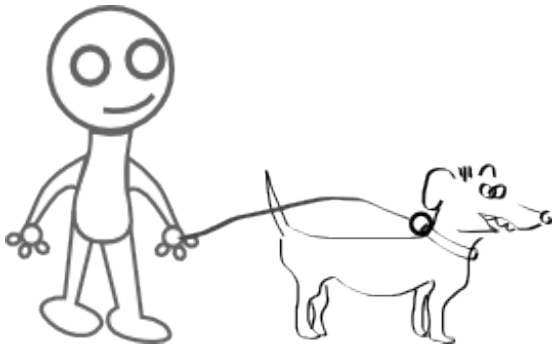

Person Using Web Browser / Manual Entry
Consumer / Service Provider
OAUTH AUTHENTICATION FLOW v1.0a
Consumer
Service Provider
Request Request Token
A
Grant Request Token
Obtain Unauthorized Request Token
Direct User to Service Provider
B
C
Obtain User Authorization
Direct User to Consumer
User Authorizes Request Token
Request Access Token
D
E
Grant Access Token
F
Exchange Request Token for Access Token
Access Protected Resources
G
A Consumer Requests Request Token
Request includes
oauth_consumer_key
oauth_signature_method
oauth_signature
oauth_timestamp
oauth_nonce
oauth_version (optional)
oauth_callback
B Service Provider Grants Request Token
Response includes
oauth_token
oauth_token_secret
oauth_callback_confirmed
C Consumer Directs User to Service Provider
Request includes
oauth_token (optional)
D Service Provider Directs User to Consumer
Request includes
oauth_token
oauth_verifier
E Consumer Requests Access Token
Request includes
oauth_consumer_key
oauth_token
oauth_signature_method
oauth_signature
oauth_timestamp
oauth_nonce
oauth_version (optional)
oauth_verifier
F Service Provider Grants Access Token
Response includes
oauth_token
oauth_token_secret
G Consumer Accesses Protected Resources
Request includes
oauth_consumer_key
oauth_token
oauth_signature_method
oauth_signature
oauth_timestamp
oauth_nonce
oauth_version (optional)

OAuth 1.0 Issues

- Complexity in validating and generating signatures
- No clear separation between Resource Server and Authorization Server
- Browser-based re-redirections

OAuth 2.0: Resource Owner

- An entity capable of granting access to a protected resource
- When the resource owner is a person, it is referred to as an end-user



OAuth 2.0: Resource Server

- The server hosting the protected resources, capable of accepting and responding to protected resource requests using access tokens

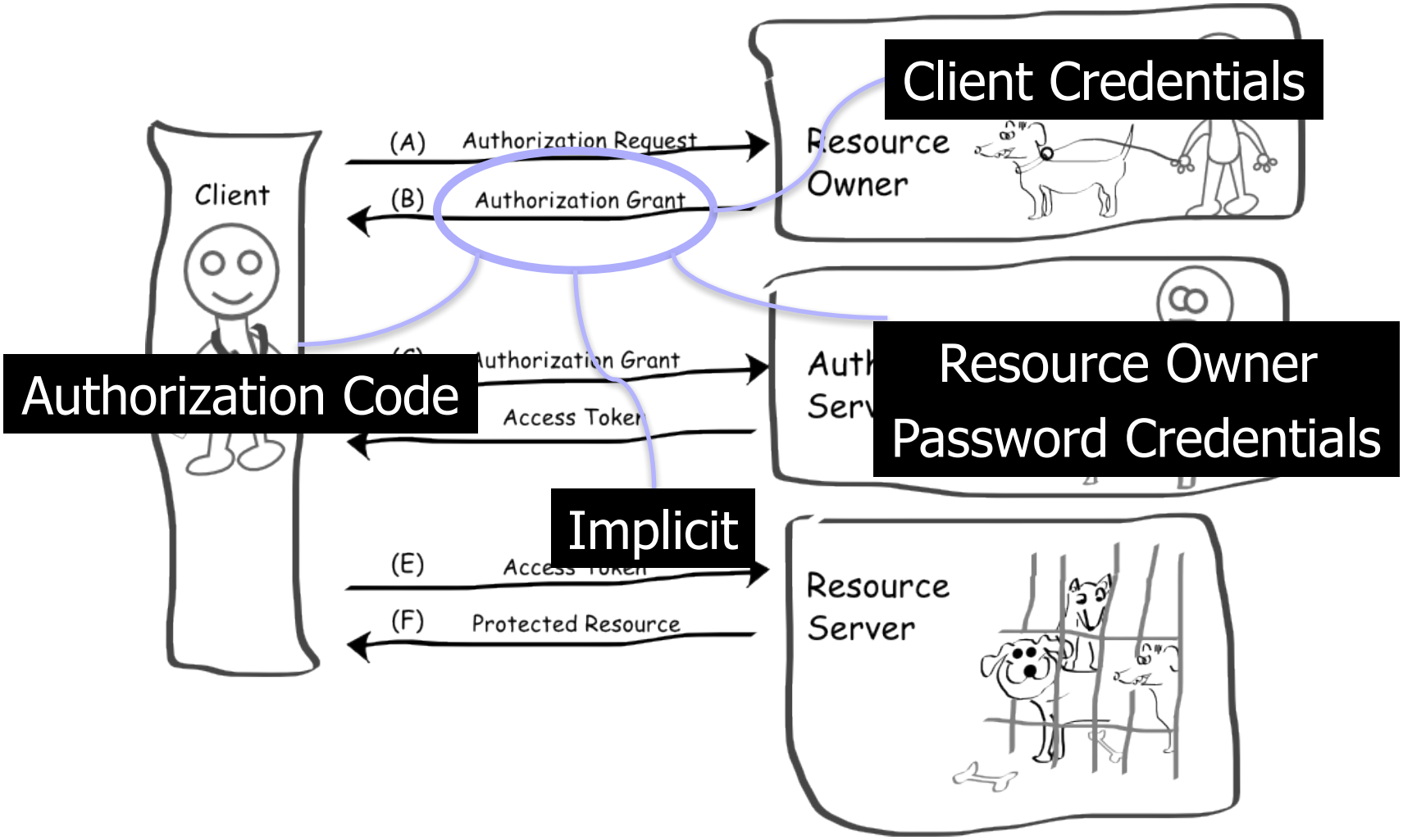
OAuth 2.0: Client

- An application making protected resource requests on behalf of the resource owner and with its authorization

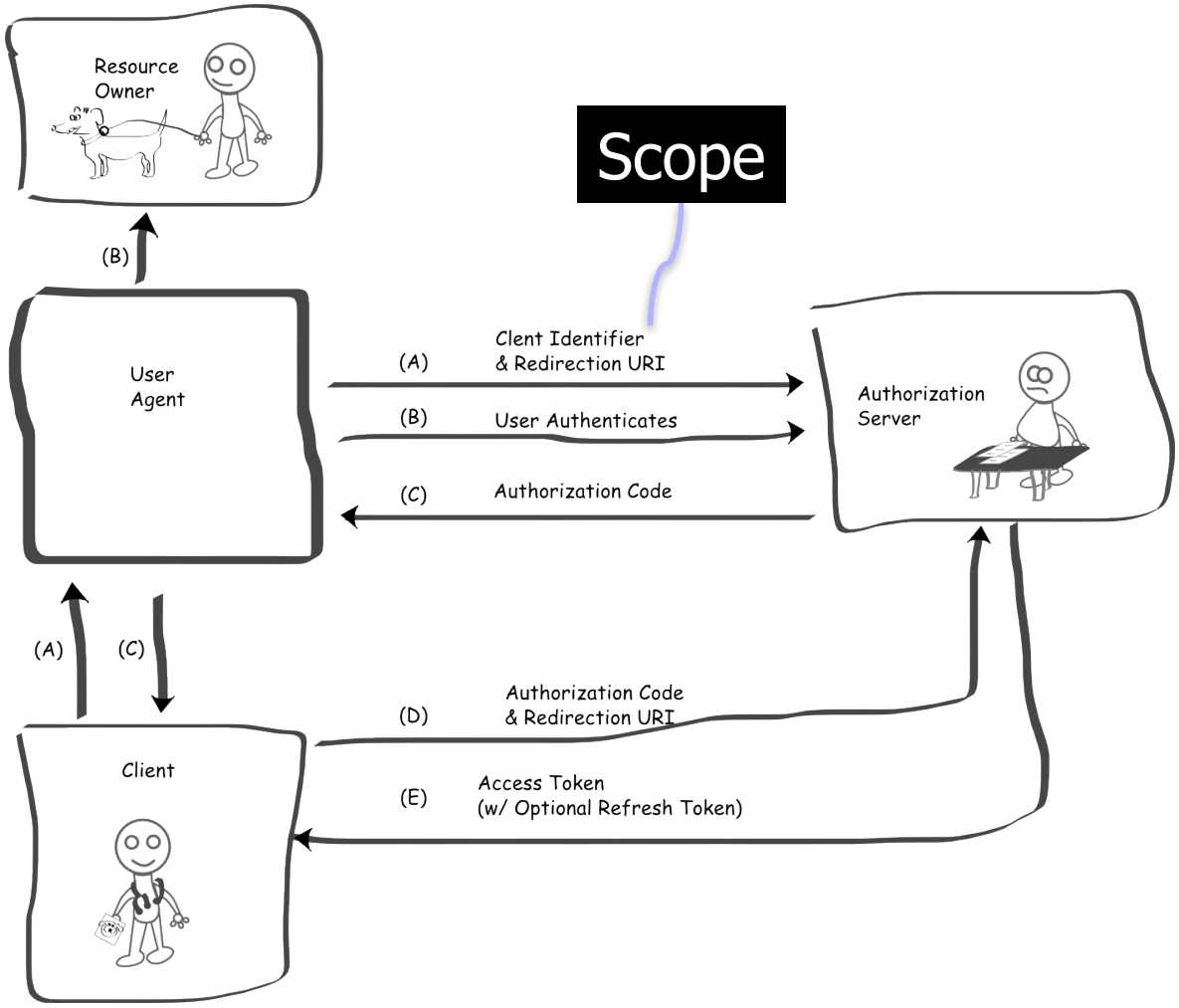
OAuth 2.0: Authorization Server

- The server issuing access tokens to the client after successfully authenticating the resource owner and obtaining authorization

Authorization Grants in OAuth 2.0



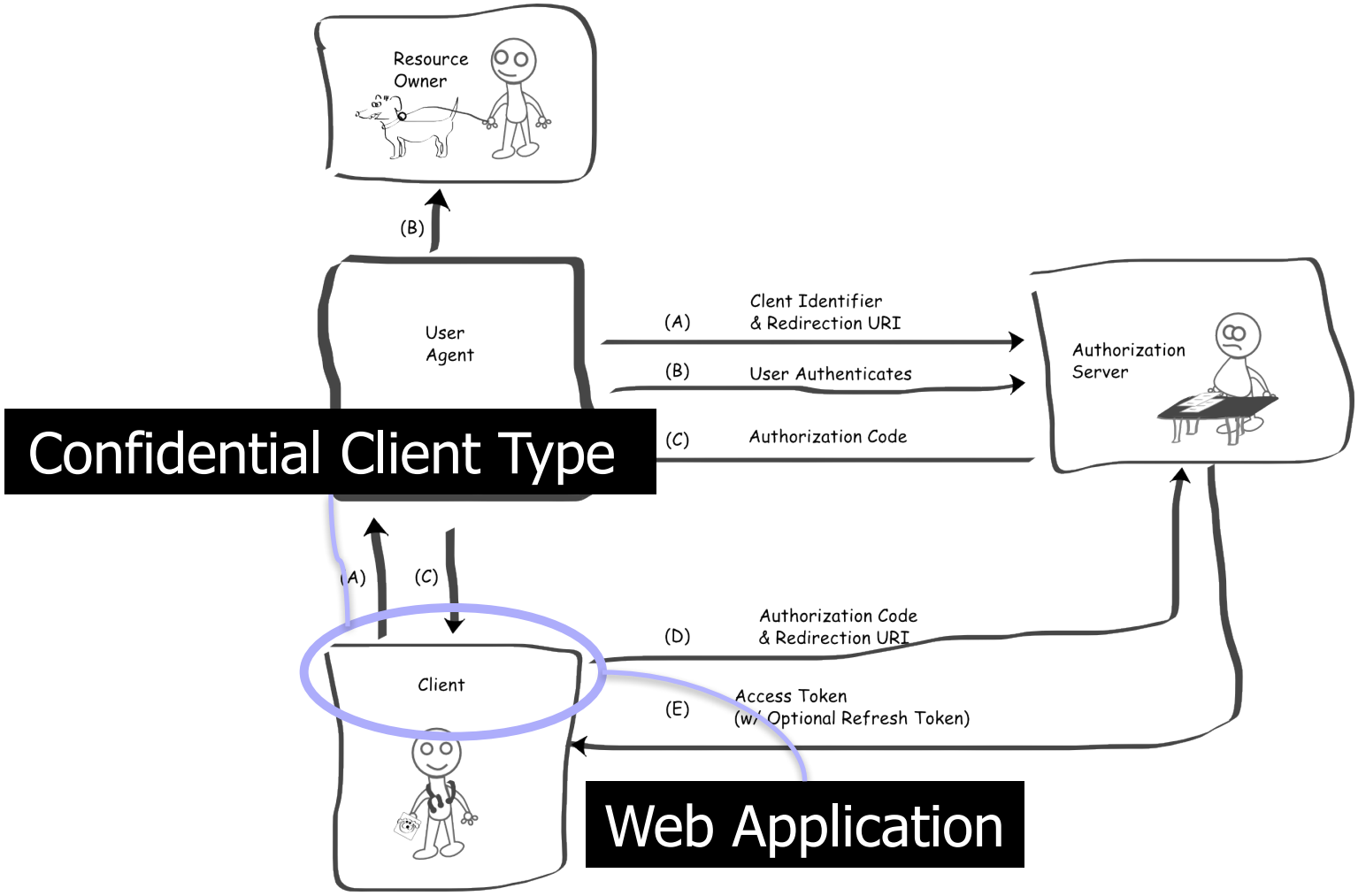
Authorization Code



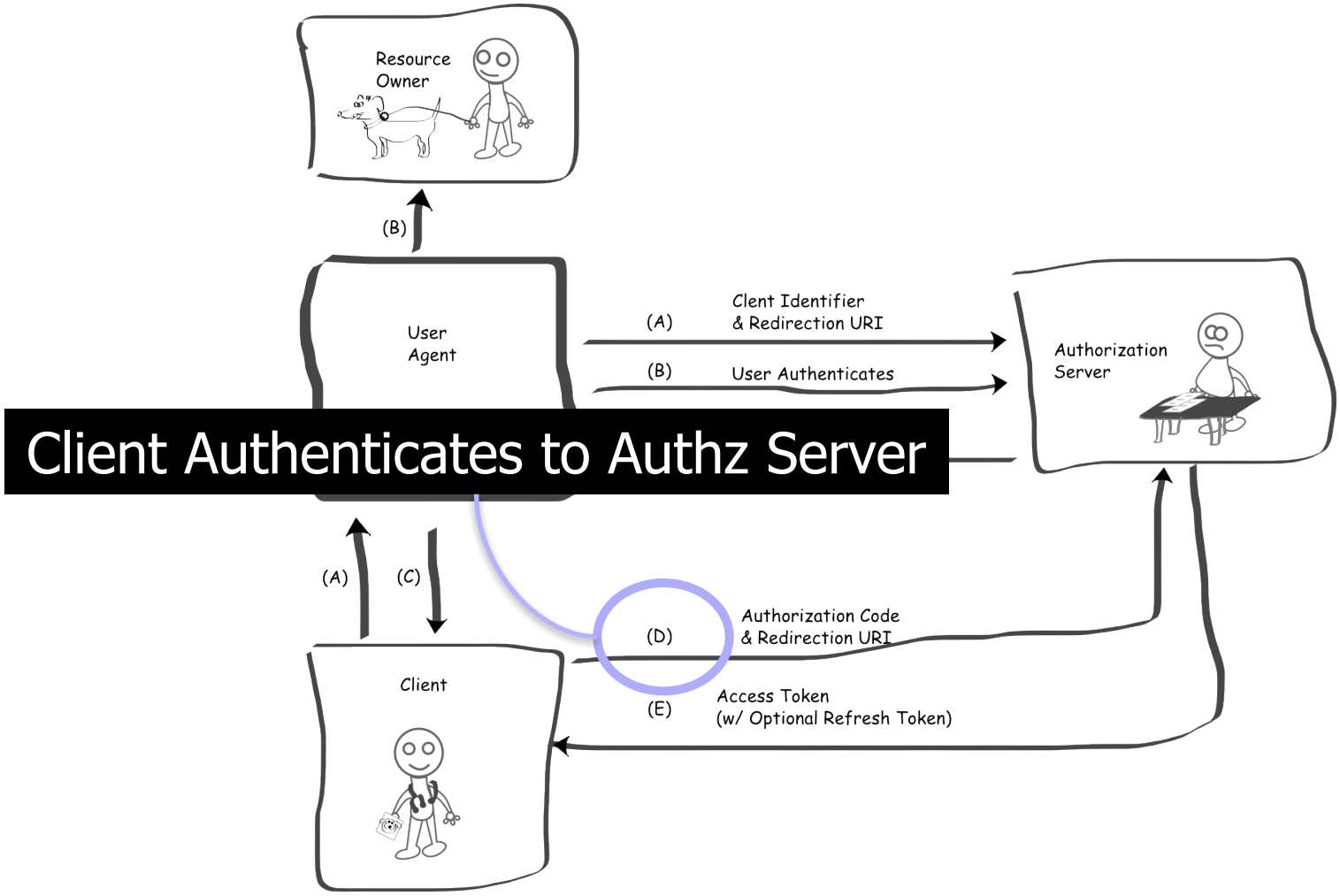
Scope

- Defined by the Authorization Server
- Indicates what resource client wants access and which actions he wants to perform on that
- The value of the scope parameter is expressed as a list of space-delimited, case-sensitive strings
  - The strings are defined by the authorization server

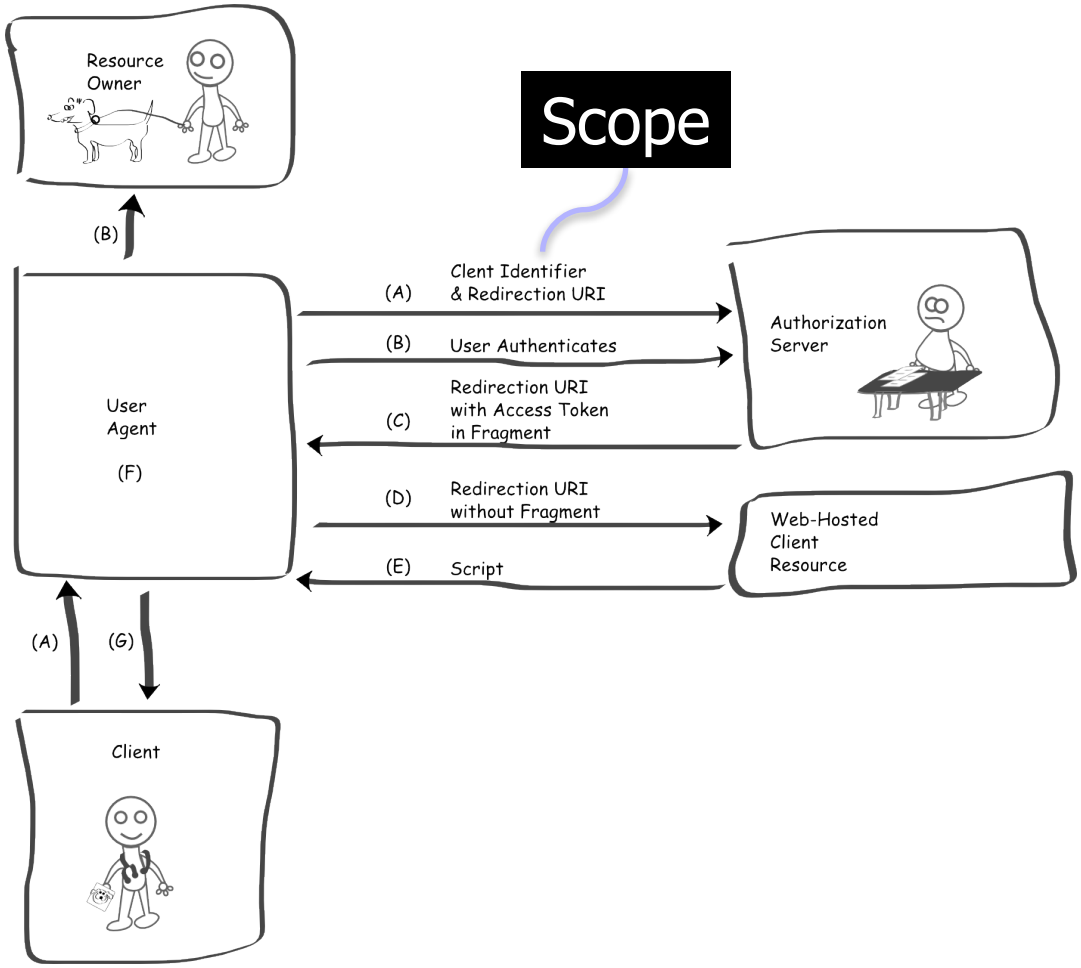
Authorization Code



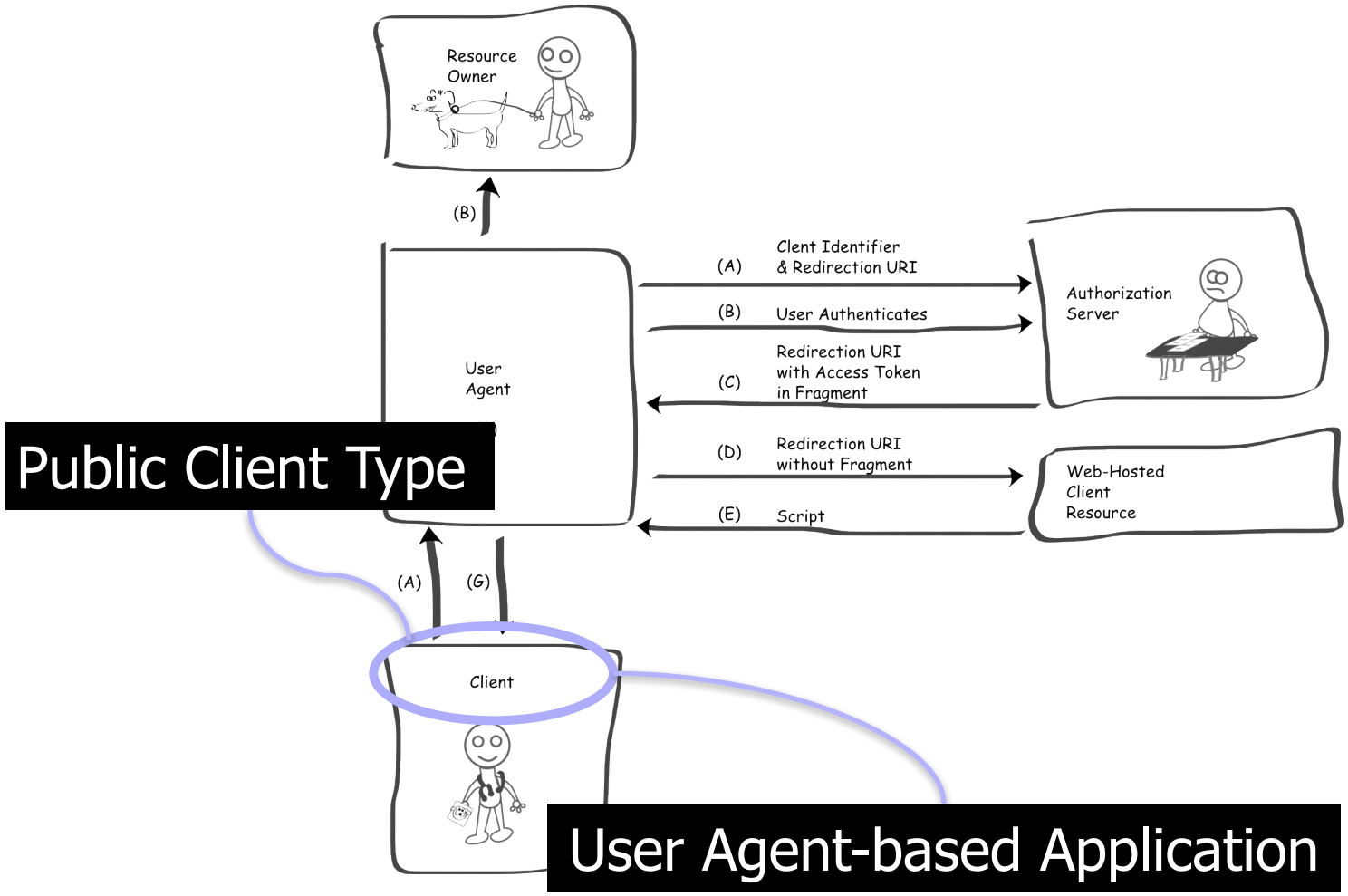
Authorization Code



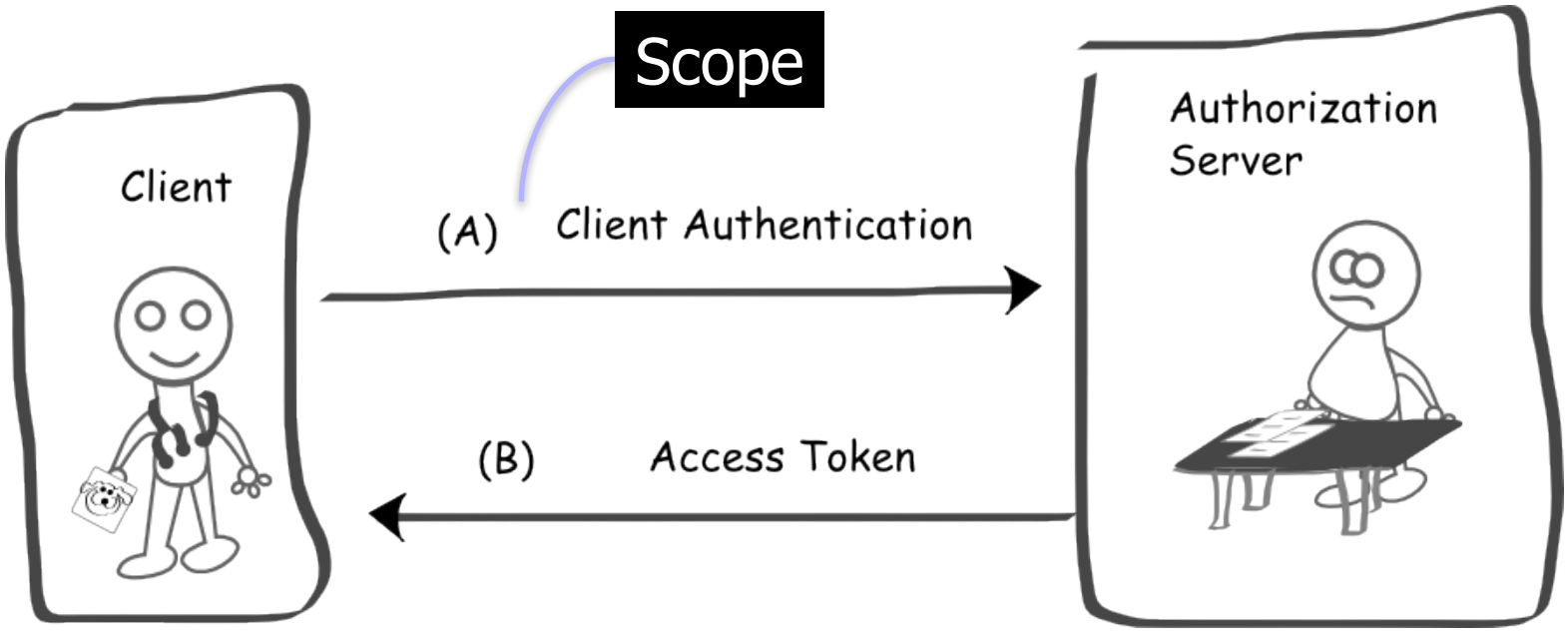
Implicit


Resource Owner
Scope
(B)
Clent Identifier & Redirection URI
(A)
(B)
User Authenticates
Redirection URI with Access Token in Fragment
(C)
Authorization Server
User Agent
(F)
Redirection URI without Fragment
(D)
Web-Hosted Client Resource
(E)
Script
(A)
(G)
Client

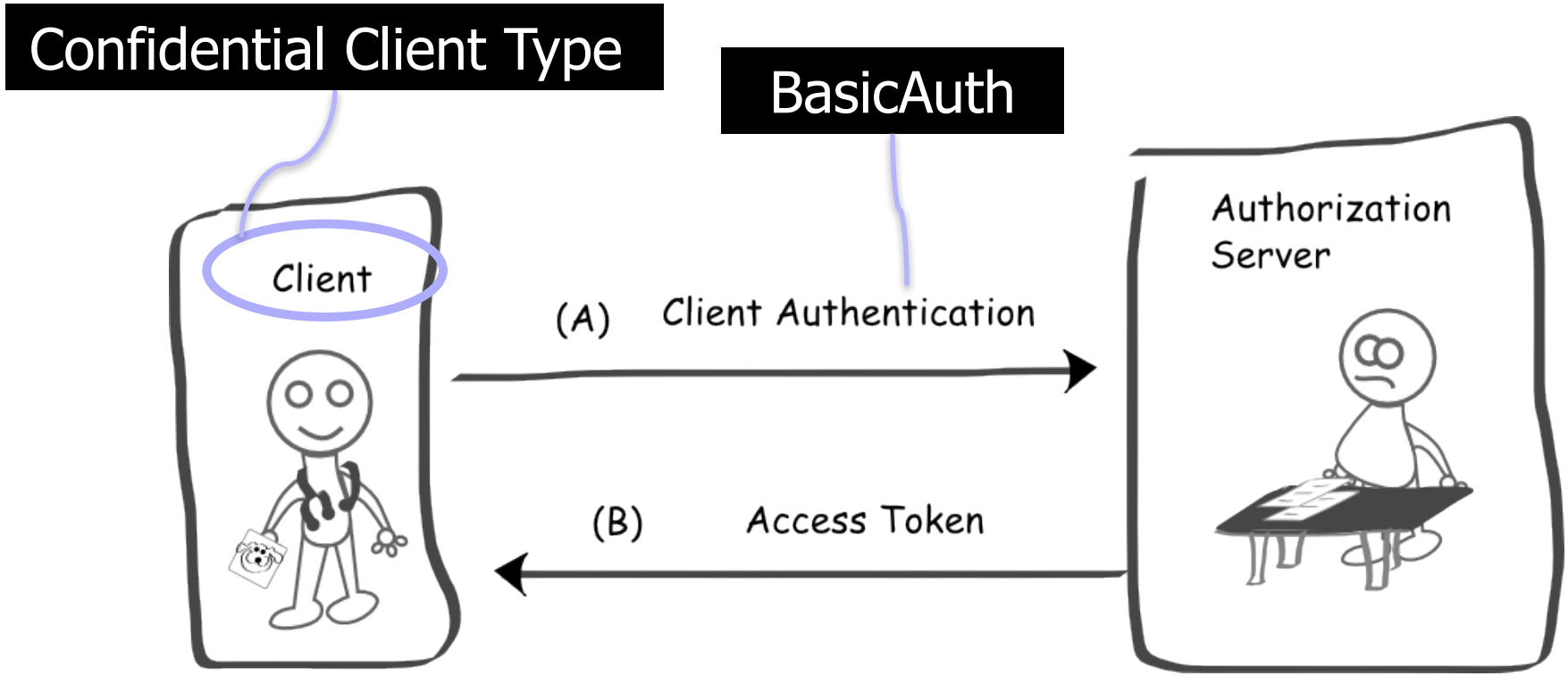
Implicit


Resource Owner
(B)
User Agent
(A) Clent Identifier & Redirection URI
(B) User Authenticates
(C) Redirection URI with Access Token in Fragment
(D) Redirection URI without Fragment
(E) Script
Authorization Server
Web-Hosted Client Resource
(A)
(G)
Client
Public Client Type
User Agent-based Application

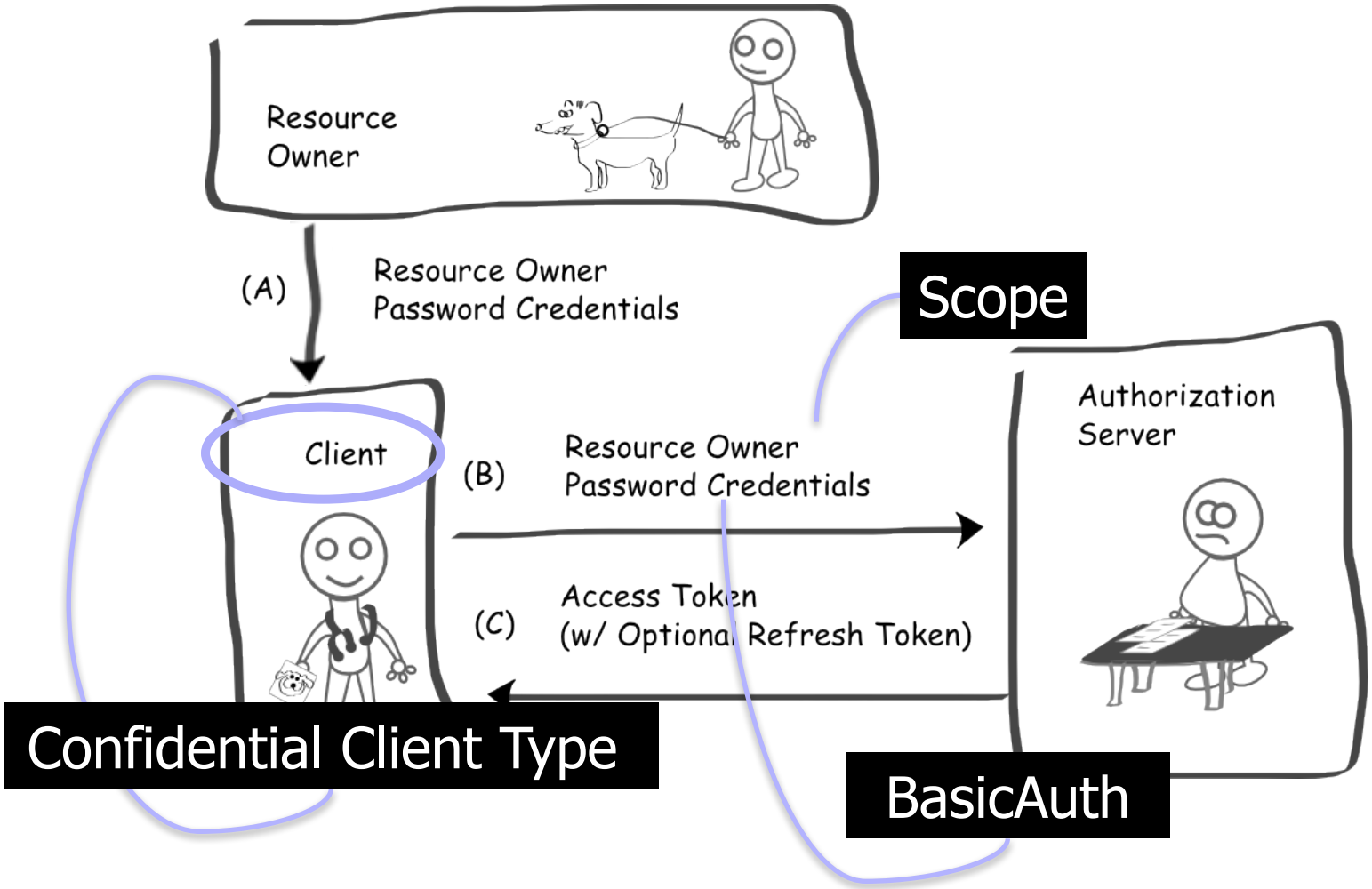
Client Credential



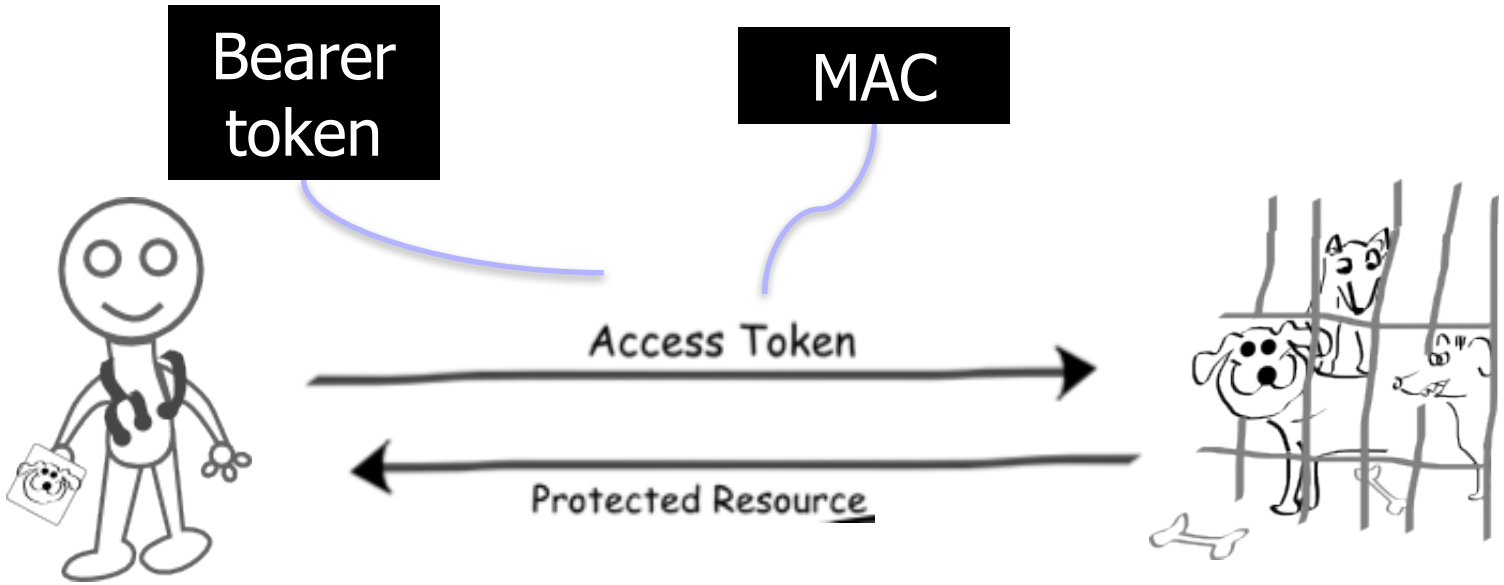
Client Credential



Resource Owner Pwd Credential



Runtime



Bearer Token and MAC

- Any party in possession of a bearer token (a "bearer") can use it to get access to the associated resources (without demonstrating possession of a cryptographic key)
- HTTP MAC access authentication scheme