



Presents

Security Concepts

Some Security Concepts

- ▶ Robust software has the ability
 - ✓ “to cope with errors during execution and to handle erroneous input”
- ▶ Three types of robustness
 - ✓ Safe: when the system can detect, respond to or prevent accidental harm
 - ✓ Secure: when the system can detect, respond to or prevent intentional harm
 - ✓ Survivable: when the system is both safe and secure

Some Security Concepts

- ▶ Software Engineering focuses on eliminating defects
 - ✓ Removing any faults that prevent the software from working as specified
 - ✓ Ensuring the software handles the normal and reasonable situations and inputs correctly, including invalid inputs
- ▶ Software Engineering does not focus on intentional attacks
 - ✓ Attacks often involve attempting to put the system into an abnormal situation or unusual state
 - ✓ Attacks often use bizarre, unreasonable and highly unusual inputs

Some Security Concepts

▶ Security flaw

- ✓ A defect in or a feature of the software that can be exploited by an attacker
- ✓ Defect fixed for normal operations may still be a security flaw
- ✓ Not all defects are security flaws
- ✓ Only defects that can be exploited are security flaws

▶ A vulnerability is a set of circumstances that allow an attacker to exploit a security flaw

▶ A mitigation is the removal of a vulnerability either

- ✓ By fixing the underlying security flaw; or
- ✓ Applying a workaround to prevent attackers from accessing the security flaw

Some Security Concepts

- ▶ Not all security flaws can be fixed
 - ✓ The cost of fixing the flaw may be prohibitive
 - ✓ The flaw may be complex or involve multiple components which means it may be a systemic problem and not a single defect

STRIDE Attack Definitions

- ▶ STRIDE is an acronym for categorizing attacks
 - ✓ *Spoofing*: Pretending to be something or someone else
 - ✓ *Tampering*: Unauthorized modification of anything in a system or application
 - ✓ *Repudiation*: Denying responsibility for something
 - ✓ *Information Disclosure*: Providing information to unauthorized parties
 - ✓ *Denial of Service*: Making system resources unavailable for use
 - ✓ *Elevation of Privilege*: Performing actions that are not authorized

Security: Basic Principles

- ▶ Design with the objective that the API will eventually be accessible from the public internet
 - ✓ Even if there are no immediate plans to do so
- ▶ Use a common authentication and authorization pattern, preferably based on existing security components
 - ✓ Avoid creating a unique solution for each API
- ▶ Least Privilege
 - ✓ Access and authorization should be assigned to API consumers based on the minimal amount of access they need to carry out the functions required

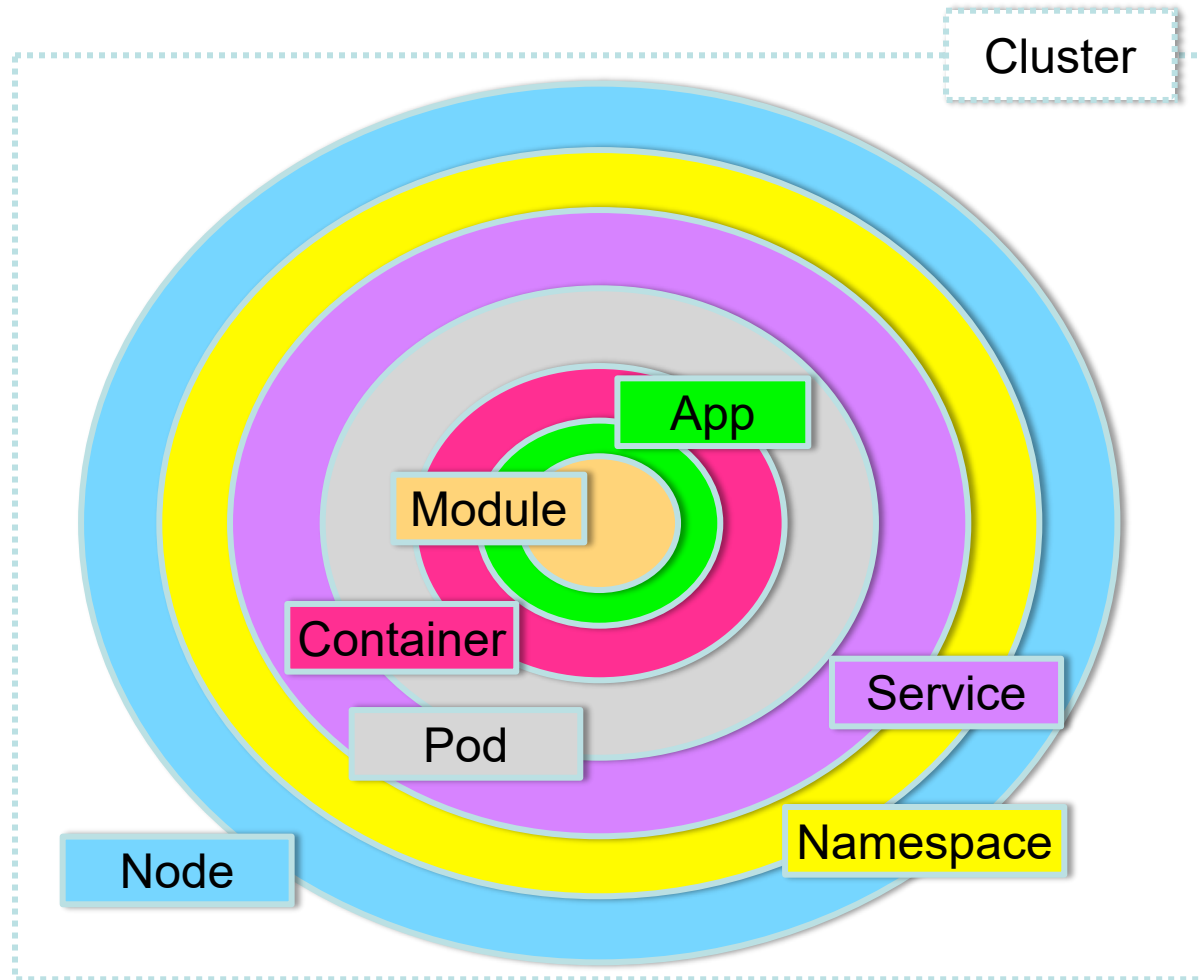
Security: Basic Principles

- ▶ Maximize entropy (randomness) of security credentials
 - ✓ Use API Keys rather than username and passwords for API
- ▶ Balance performance with security with reference to key lifetimes and encryption/decryption overheads
- ▶ Standard secure coding practices should be integrated
- ▶ Security testing capability is incorporated into the development cycle
 - ✓ Continuous, repeatable and automated tests to find security vulnerabilities in APIs and web applications during development and testing

OWASP Secure Coding Principles

Principle	Example
Minimize attack surface area	Use a “security” gateway
Establish secure defaults	Password aging and complexity should be enabled.
Principle of Least privilege	A middleware server only requires access to the network, read access to a database table, and the ability to write to a log.
Principle of Defense in depth	In Kubernetes assign TLS certificates to a namespace and user group. (The more the merrier.)
Fail securely	Treat security checks as an error event
Don't trust services	Make sure a delegate service's security policies are in sync with YOURS.
Separation of duties	Admins do admin work, users do user work, admin does not do user work
Avoid security by obscurity	Hoping the bad actors won't find password files stored on a machine is a bad idea
Keep security simple	Using standard salting methods is a lot easier to maintain than creating a big authentication algorithm that is proprietary to your service
Fix security issues correctly	Treat the cause not the symptom

Defense in Depth



Security: Tactics

Item	Comments
Code	Treat everything as hostile until safety is determined. Consider migrating to programming paradigm that puts security at the forefront, such as functional programming
Container	<ul style="list-style-type: none">• If possible, build container images from scratch and store in private repository• No root users• Avoid accessing host
Kubernetes	<ul style="list-style-type: none">• Use namespaces• Use Roles Based Access Control (RBAC)• Using network policies
Node/VM	Node/VM: Use service mesh

Security: Some Best Practices

- ▶ Use OAuth for user identity and access control
- ▶ Use 'defense in depth' to prioritize key services
- ▶ Avoid writing your own crypto code
 - ✓ libsodium <https://github.com/jedisct1/libsodium>
 - ✓ Bouncy Castle <https://www.bouncycastle.org/>
- ▶ Use automatic security updates (in a controlled manner)
- ▶ Use a distributed firewall with centralized control
 - ✓ Calico <https://www.projectcalico.org/>

Security: Some Best Practices

- ▶ Get your containers out of the public network, use an API Gateway
- ▶ Use security scanners for your containers
 - ✓ Twistlock
 - ✓ Clair <https://coreos.com/clair/docs/latest/>
 - ✓ rat scans by default
 - ✓ Docker does it too
- ▶ Monitor everything with a tool
 - ✓ Prometheus
 - ✓ InfluxDB
 - ✓ Stated
 - ✓ Cockpit

Authentication and Authorization

► Authentication

- ✓ Uses agent's information to identify them
- ✓ Verifies the agent's credentials
- ✓ Must occur before any authorization happens
- ✓ Confirming the truth of some piece of data used by agent to identify themselves

“How can you prove who you are?”

Authentication and Authorization

► Authorization

- ✓ Checks an agent's right to access a resource
- ✓ Validates the agent's permissions
- ✓ Occurs after the identity of the agent is confirmed
- ✓ Specifies the rights, permissions and privileges of an authenticated agent

*“How do we know what you
are allowed to do?”*


Password Fatigue


- ▶ Feeling experienced by managing too many user ids and passwords
- ▶ Creates a social engineering security risk
 - ✓ Users use the same password everywhere – a security vulnerability
 - ✓ Users do not change their passwords regularly
 - ✓ Users tend to use easily remembered (easily cracked) passwords
 - ✓ Users tend to record passwords and account information insecurely
- ▶ The various authentication credentials used are called “secrets”
 - ✓ A main security vulnerabilities is poor secrets management


Single Sign-On


- ▶ Single Sign-On (SSO)
- ▶ User can log in with a single ID and password to multiple systems
- ▶ Authentication is shared between the systems
- ▶ The systems are independent but are related in some way
- ▶ Also referred to as a federated login across networks


Welcome back.

 Sign in with Google

 Sign in with Facebook

 Sign in with Apple

 Sign in with Twitter

 Sign in with email

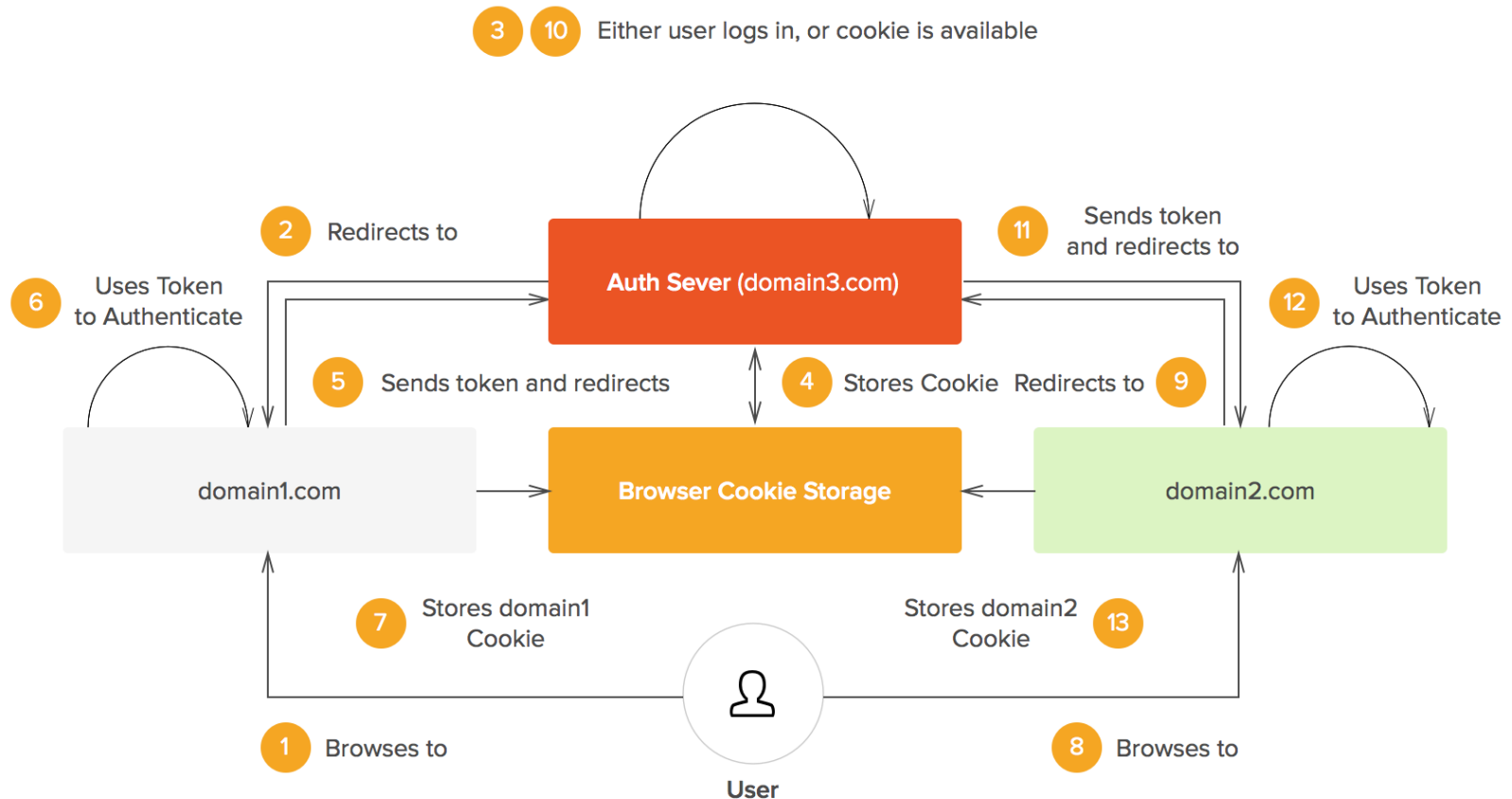
No account? [Create one](#)

Click "Sign In" to agree to Medium's [Terms of Service](#) and acknowledge that Medium's [Privacy Policy](#) applies to you.

Password Fatigue

- ▶ Feeling experienced by managing too many user ids and passwords
- ▶ Creates a social engineering security risk
 - ✓ Users use the same password everywhere – a security vulnerability
 - ✓ Users do not change their passwords regularly
 - ✓ Users tend to use easily remembered (easily cracked) passwords
 - ✓ Users tend to record passwords and account information insecurely
- ▶ The various authentication credentials used are called “secrets”
 - ✓ A main security vulnerabilities is poor secrets management

Identity Broker and SSO



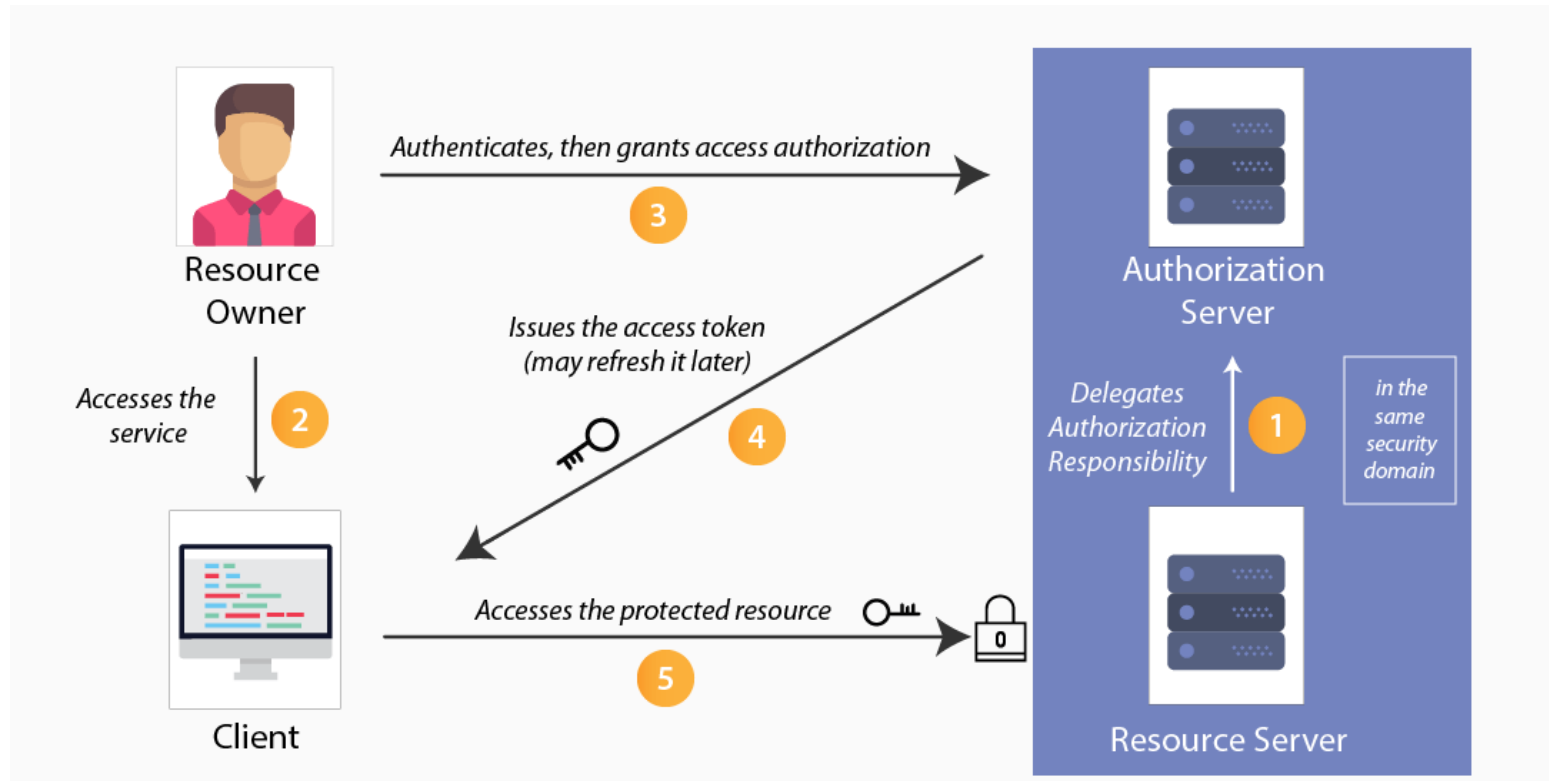
SSO Protocols

- ▶ There exist a variety of implementations of SSO
 - ✓ Extensive list at:
 - https://en.wikipedia.org/wiki/List_of_single_sign-on_implementations
 - ✓ Some are open standards like FreeIPA from Redhat
 - ✓ Others are proprietary like Facebook Connect
- ▶ Significant challenges are:
 - ✓ How to authenticate the authenticators
 - ✓ How to communicate credentials securely
 - ✓ How to manage secrets e.g., should credentials expire?

OAuth Open Authorization

- ▶ Mechanism for providing access to a server by a client by
 - ✓ Delegating authorization to a broker who authenticates client
 - ✓ Returns OAuth token used by the client to access the server
- ▶ OAuth 2 is a complete rewrite of OAuth 1
 - ✓ Not backward compatible with OAuth 1
 - ✓ The two versions are essentially separate protocols
 - ✓ OAuth 2 added support for web applications, desktop applications, mobile phones, and smart devices
- ▶ Major advantage: devices and apps don't store credentials
 - ✓ They only need store tokens that expire

OAuth2 In Operation



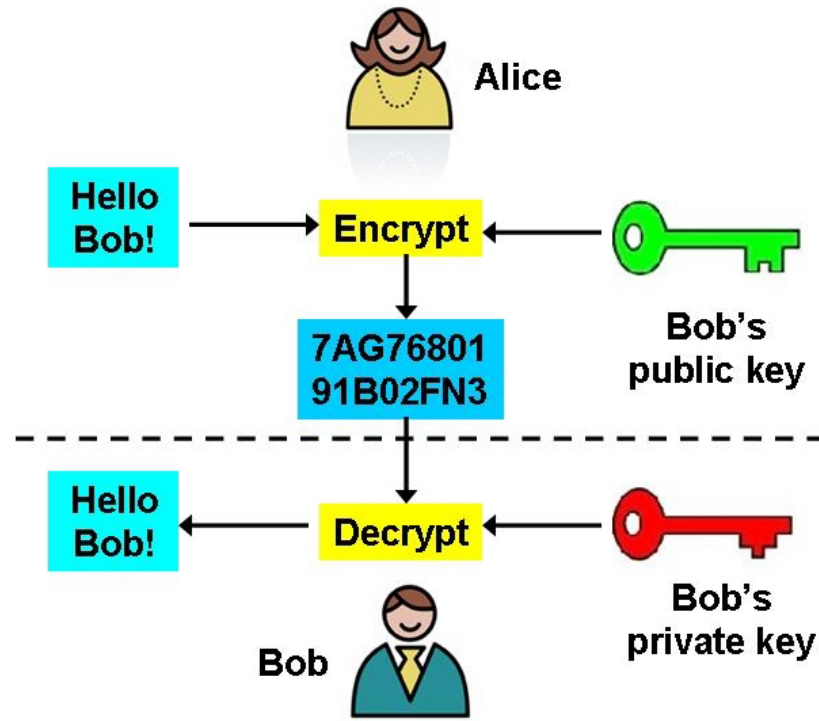
Asymmetric Encryption

► Uses a public/private key pair

- ✓ The public key can encrypt text sent to the key owner
- ✓ Only the key owner's private key can decrypt the cipher text
- ✓ The public key cannot decrypt

► In practice

- ✓ Stronger symmetric encryption is used with a random key
- ✓ The symmetric key is then encrypted with a public key



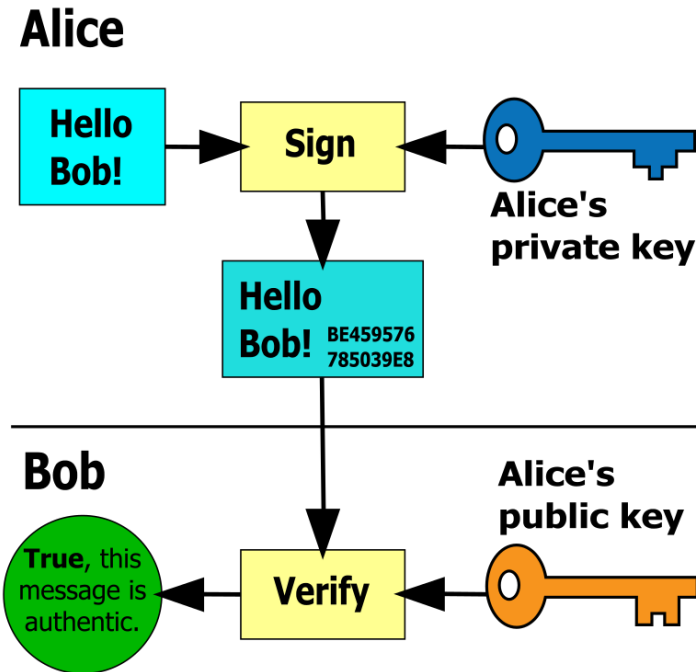
Digital Signatures

► To sign a message

- ✓ A hash of the message is made then encrypted with a private key
- ✓ This is the digital signature
- ✓ Only the owner of the private key can create a signature

► Verification

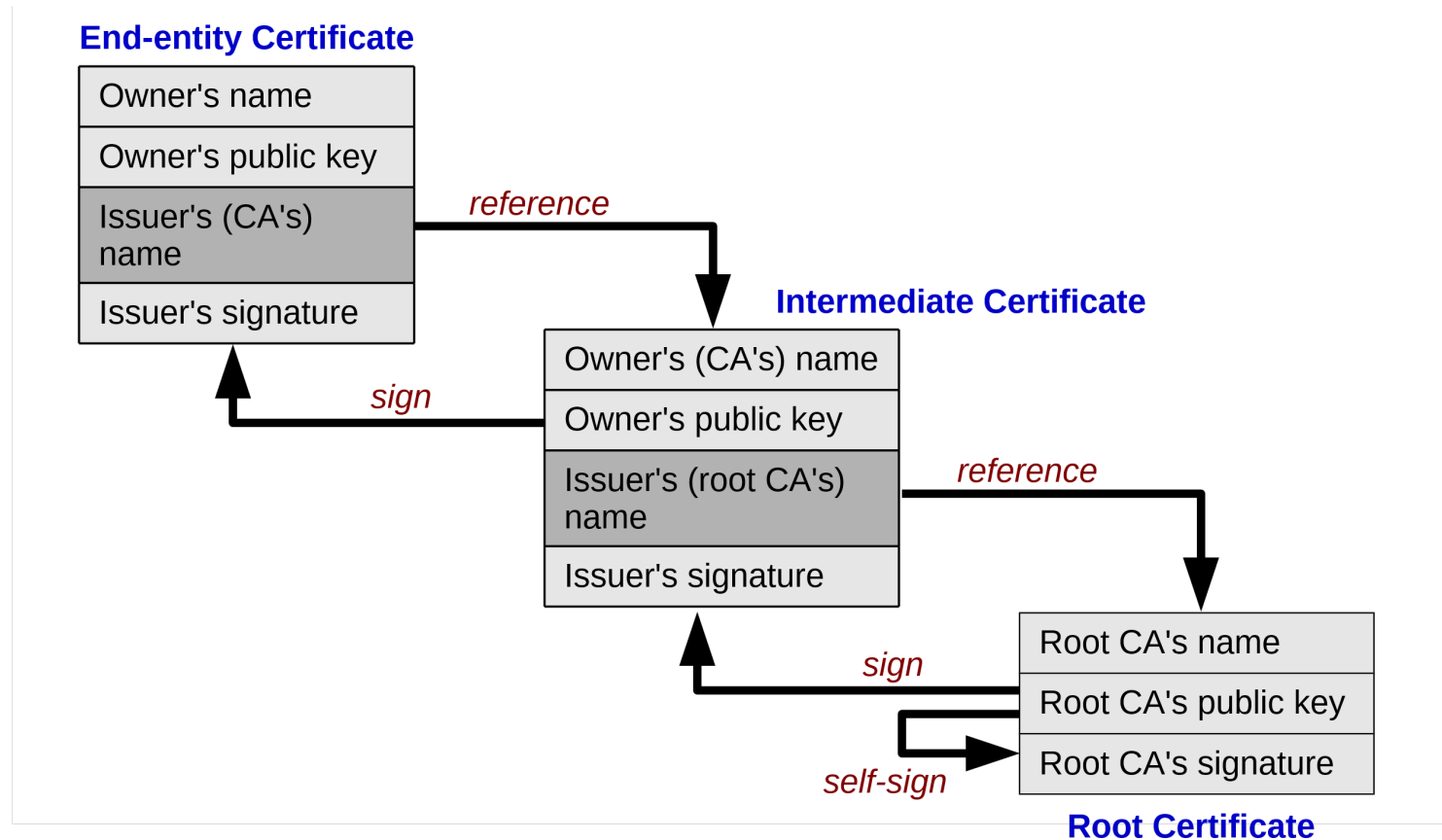
- ✓ The signature is decrypted with the sender's public key
- ✓ The decrypted hash is compared to a new hash of the message
- ✓ A match = verified authentic



Certificates and Trust

- ▶ An X509 digital certificate is a cryptographic ID document
 - ✓ My certificate is used to verify my identity
 - ✓ Issued by a CA or certificate authority who signs my certificate with their private key to verify it is really mine
 - ✓ The CA signed certificate acts a trusted third party that has vouched for me
- ▶ The CA's certificate is signed by another CA
 - ✓ The chain of CA signatures starts with a root certificate or trust anchor for a "chain of trust" - signatures can be verified
- ▶ Every CA must meet strict requirements and undergo a compliance audit
 - ✓ There are about 50 trusted root CAs

Chain of Trust

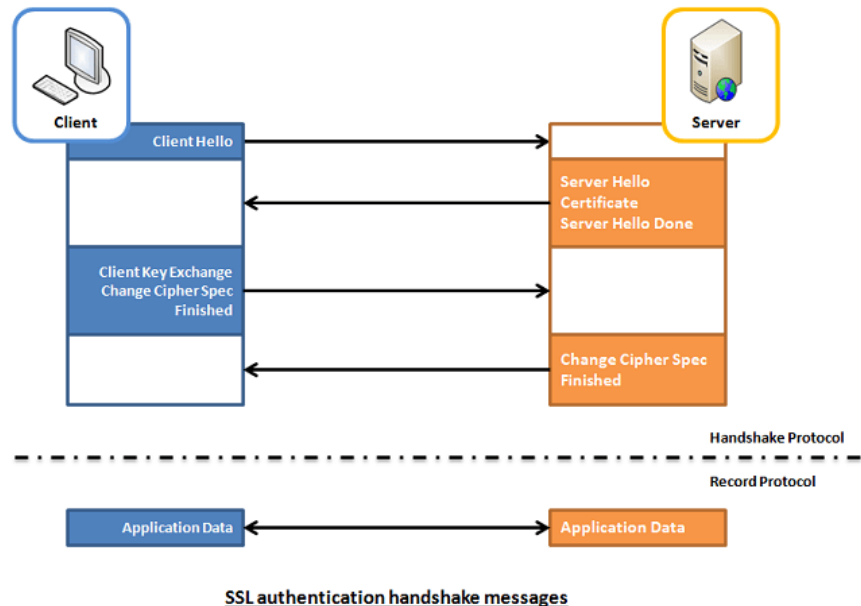


TSL – Transport Security Layer

- ▶ Cryptographic protocol for End-to-end security of data sent between applications over the Internet
 - ✓ Derived from Secure Sockets Layers (SSL)
 - ✓ Used to establish secure browser sessions with HTTPS
 - ✓ Also used for email, video/audio conferencing, IM, VOIP, and other services
- ▶ Implementation of security in transit
 - ✓ Information in transit is secure from tampering
 - ✓ Does not ensure security at rest - Information may be compromised at either before or after transmission
 - ✓ In cases where the identity of the server is not in question self signed certificates may be used (most browsers will warn about this)

TSL – Transport Security Layer

- ▶ Starts with a “handshake”
 - ✓ Certificate is given to the client to verify the server ID during the session
 - ✓ Asymmetric keys are created for the session
- ▶ Session keys are used to encrypt the data in transit



mTLS - Mutual TLS

- ▶ Client connects to server
- ▶ Server presents its TLS certificate
- ▶ Client verifies the server's certificate
- ▶ Client presents its TLS certificate (unlike TLS, the client must also have a certificate)
- ▶ Server verifies the client's certificate
- ▶ Server grants access
- ▶ Client and server exchange information over encrypted TLS connection

Questions

