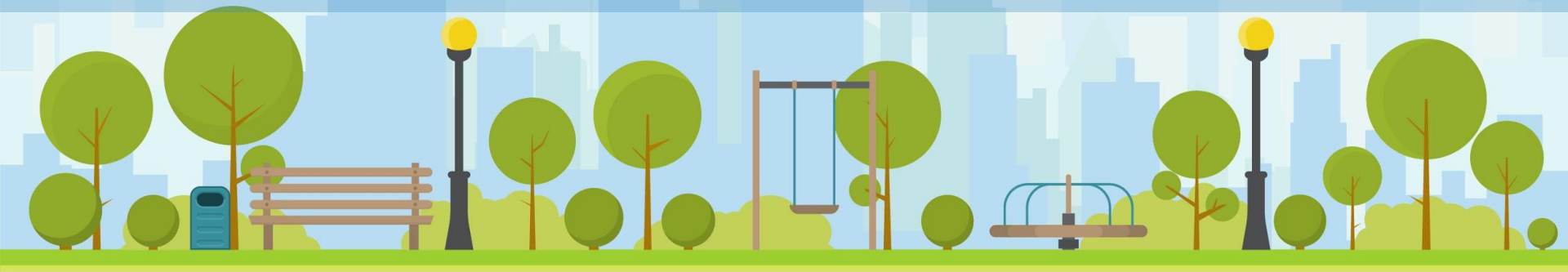

DEVOPS
PLAYGROUND
LONDON

DevOps Playground

Inspiring tech-enthusiasts to explore
new technology during hands-on monthly events.

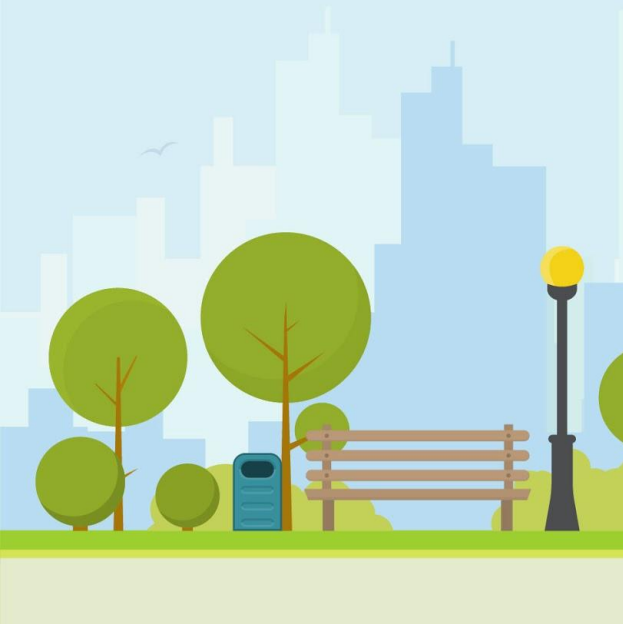


Hands on with Mutual TLS Authentication for Microservices

Roger Carhuatocto

@Chilcano
https://linkedin.com/in/chilcano
https://holisticsecurity.io

Synopsis



While Transport Layer Security (TLS) protects communications between two parties for confidentiality and integrity, the Public Key Certificates (x.509) and the Certificate Authority (CA) will allow users to set a cryptographic identity and build a trust relationship between these parties.

With all these concepts, we will be able to secure (encrypt) data in transit and authenticate the parties in scenarios such as Client-Server or Service-Service.

During this Playground you will learn:

- What One-Way, Two-Way or Mutual TLS are.
- How to use Public Key Certificates.
- How to use Mutual TLS together for securing communications among Microservices.
- How to automate everything.

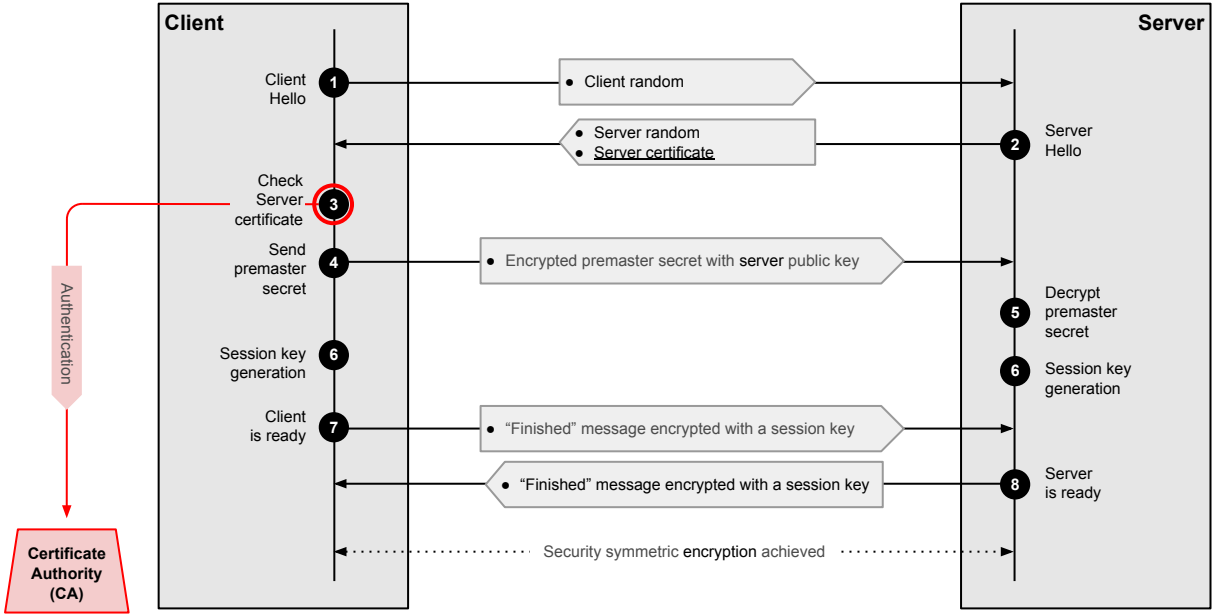
What is TLS?

"Transport Layer Security (TLS), and its now-deprecated predecessor, Secure Sockets Layer (SSL), are cryptographic protocols designed to provide communications security over a computer network. Several versions of the protocols are widely used in applications such as email, instant messaging, and voice over IP, but its use as the Security layer in HTTPS remains the most publicly visible". (https://en.wikipedia.org/wiki/Transport_Layer_Security)

TLS is able to work over these layers

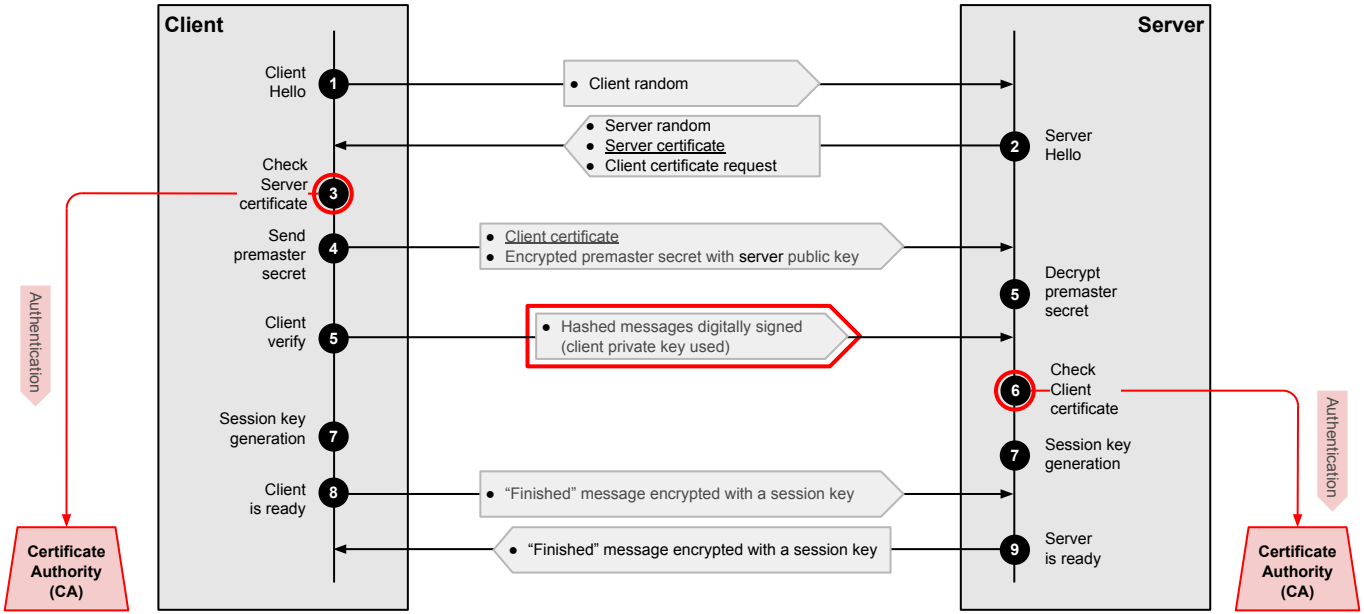
Layer	Function	Example of protocols and/or equipment
Application - 7	Services affecting end user applications	SMTP
Presentation - 6	Presentation Layer	JPEG - MIDI - MPEG - PICT - TIFF - GIF - HTTPS - SSL - TLS
Session - 5	Session Layer	NetBIOS - NFS - PAP - SCP - SQL - ZIP
Transport - 4	Transport Layer	TCP - UDP
Network - 3	Network Layer	Routers - Layer 3 Switches - IPsec - IPv4 - IPv6 - IPX - RIP
Data Link - 2	Data Link Layer	Switches - ARP - ATM - CDP - FDDI - Frame Relay - HDLC - MPLS - PPP - STP - Token Ring
Physical - 1	Physical Layer	Hubs - Bluetooth - Ethernet - DSL - ISDN - 802.11 - WiFi

One-way TLS (Server Authn)



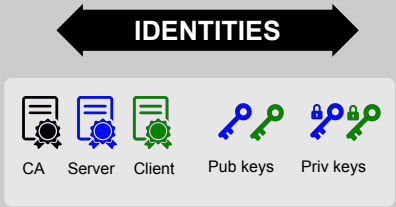
1. The client initiates the handshake by sending a "hello" message to the server. The message will include which TLS version the client supports, the cipher suites supported, and a string of random bytes known as the "client random."
2. In reply to the client hello message, the server sends a message containing the server's TLS certificate, the server's chosen cipher suite, and the "server random," another random string of bytes that's generated by the server.
3. The client verifies the server's TLS certificate with the certificate authority that issued it. This confirms that the server is who it says it is, and that the client is interacting with the actual owner of the domain.
4. The client sends one more random string of bytes, the "premaster secret." The premaster secret is encrypted with the public key and can only be decrypted with the private key by the server. (The client gets the public key from the server's TLS certificate.)
5. The server decrypts the premaster secret using its asymmetric private key.
6. Both client and server generate session keys from the client random, the server random, and the premaster secret. They should arrive at the same results.
7. The client sends a "finished" message that is encrypted with a session key.
8. The server sends a "finished" message encrypted with a session key.

Two-way TLS (Client + Server Authn)



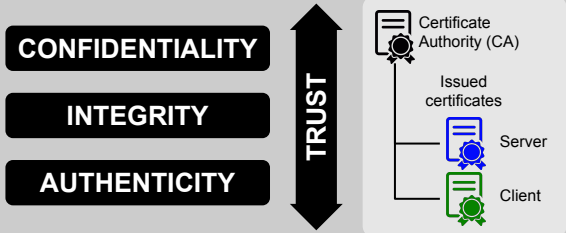
What is the Security approach?

1 Identity-based Security



- Based on asymmetric-key cryptography (key pair)
- Public Key Certificate (X.509):
  - Public Key
  - Identity (hostname, organization or individual)
  - Signature
- Private Key

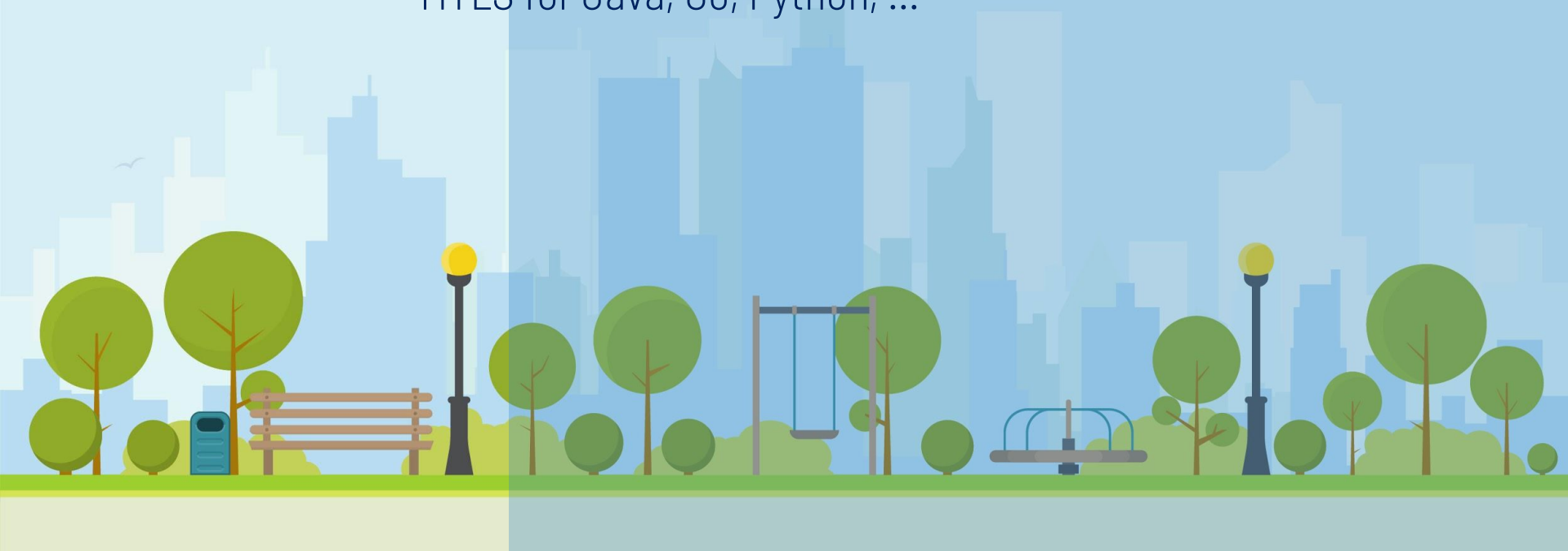
2 Trust-based Services



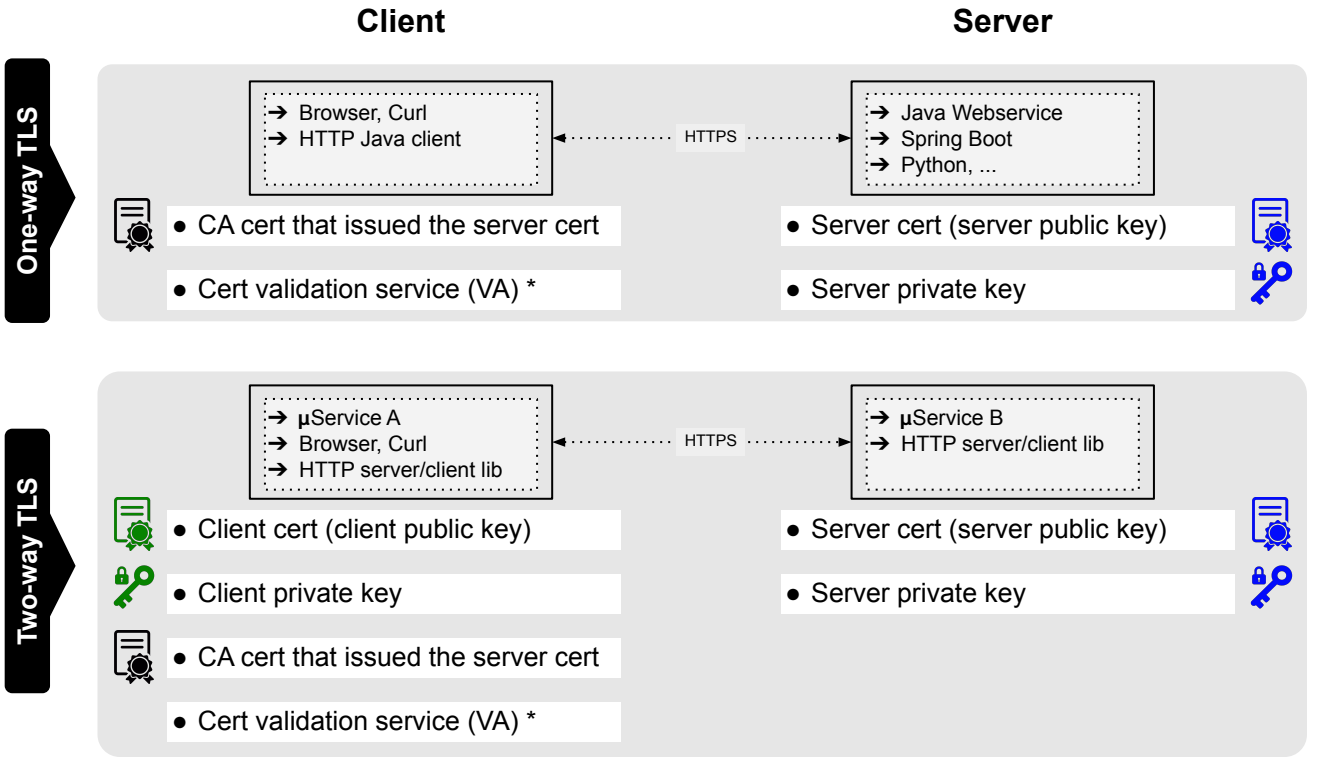
- Public Key Infrastructure (PKI) to manage the X.509 Certificates lifecycle:
  - Certificate Authority (CA)
  - Registration Authority (RA)
  - Validation Authority (VA)
  - Directory
  - Policy Certification (contract)

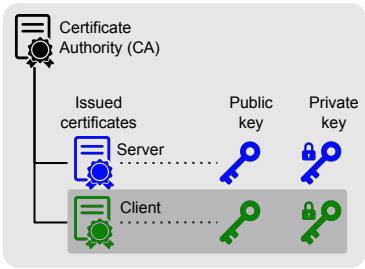
Use case 1:

MTLS for Java, Go, Python, ...

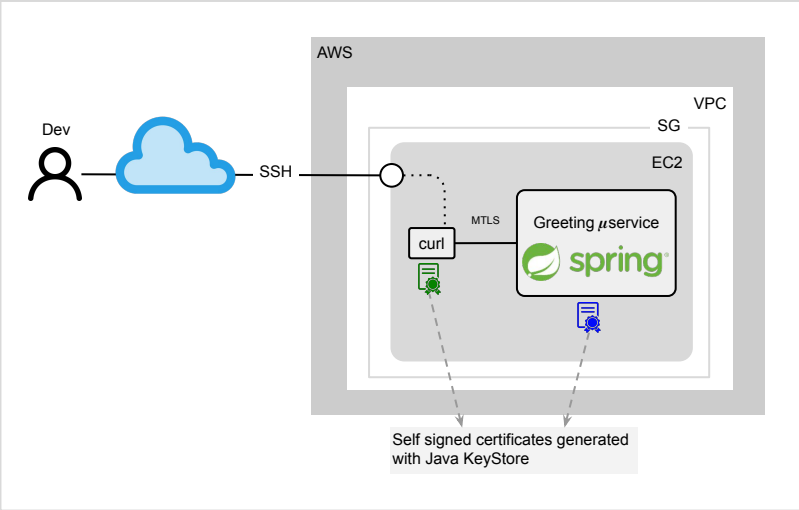


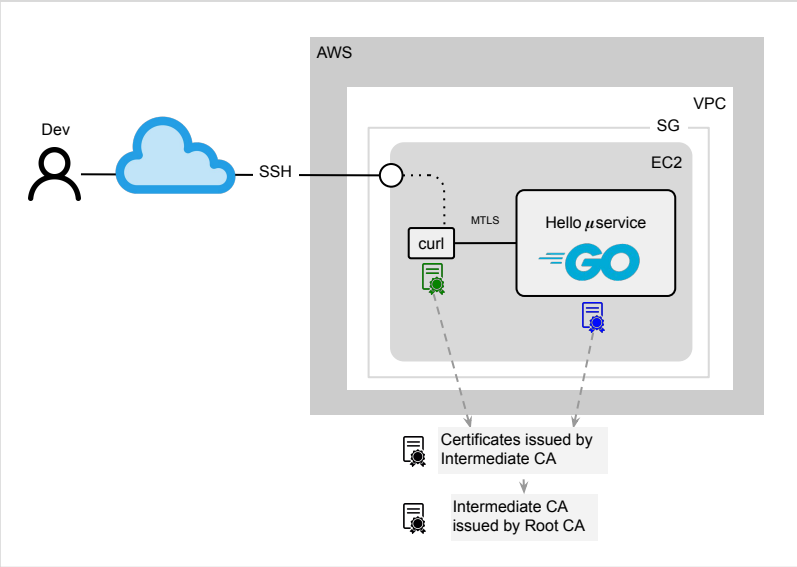
Use case 1: Java, Go, Python, ...





Use case 1: Hands-on time!



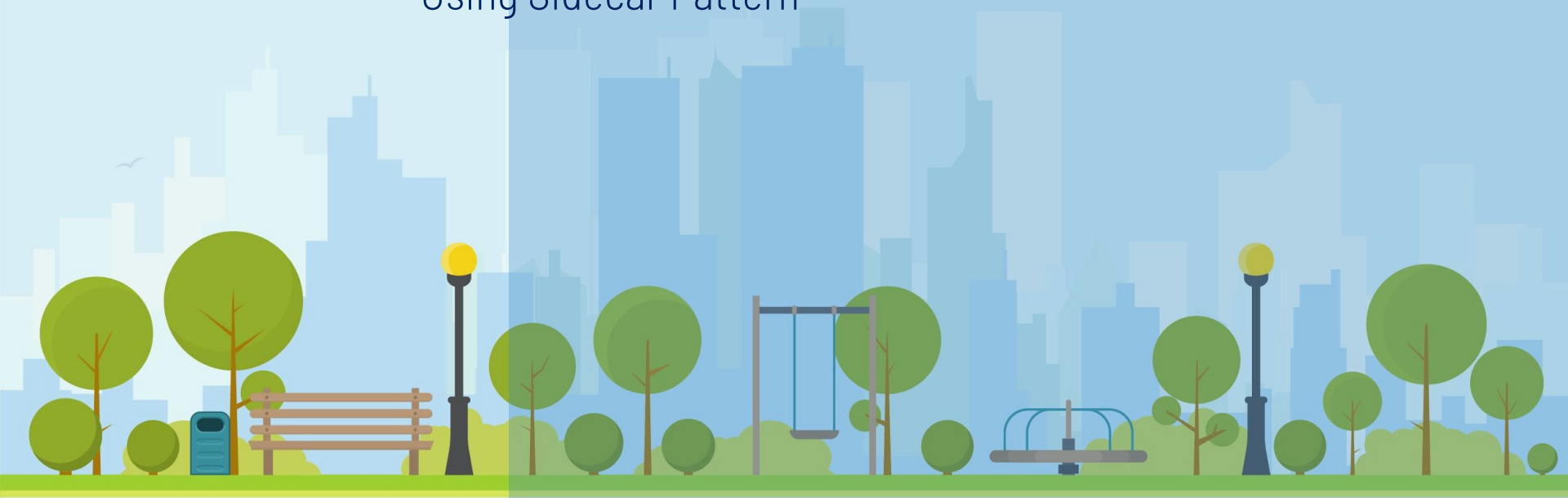


Use case 1: Conclusions

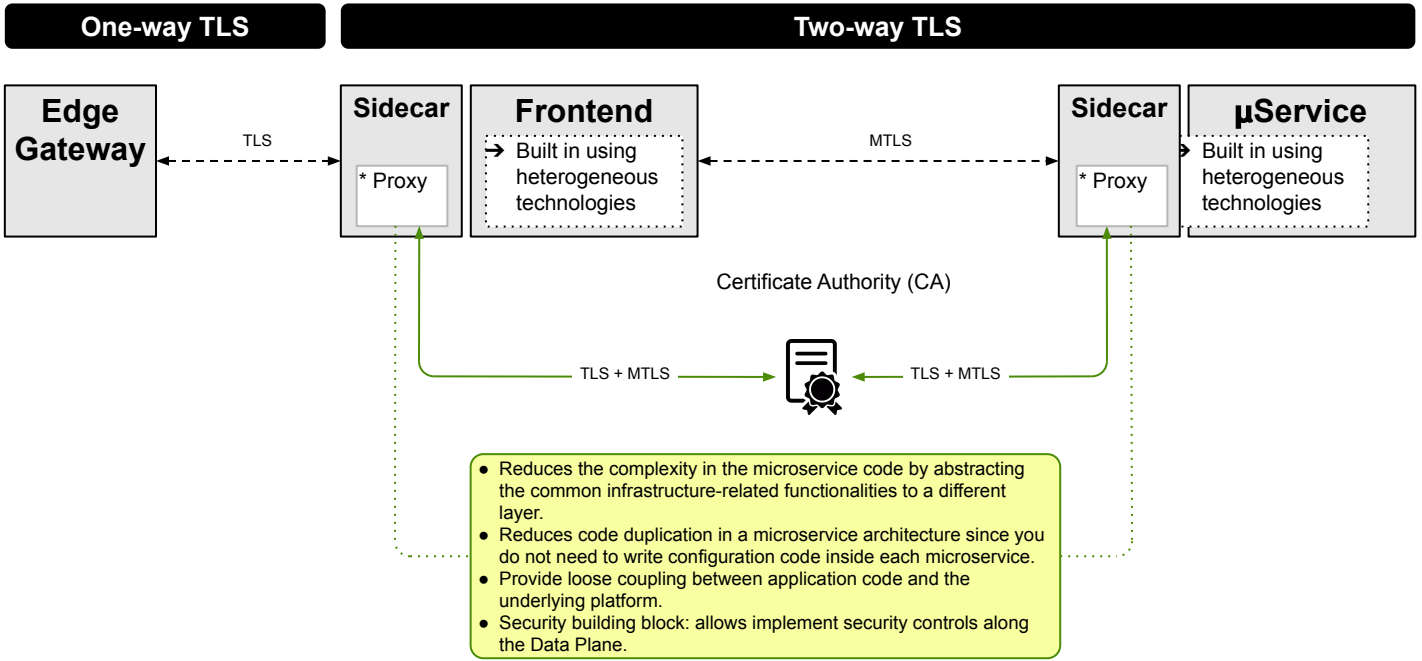
- How to manage the certs, public and private keys lifecycle?
  - Key rotation, keys revocation, secure key storage, validation, keys delivery, …
- How to configure TLS and distribute keys dynamically in all workloads?
  - How to assign corresponding cert and keys to specific workload?
    - Assign key-pair A to μservice A, not μservice B.
  - Will new TLS config (crypto suite, TLS termination, TLS policy validation, …) and keys need to evict current μservice and deploy it again with fresh config?
  - When using different technologies, Will μservices need to be modified to work with TLS?
  - How do I enable TLS and key in μservices from time zero?
    - "Chicken-Egg" Problem: bootstrap "secret" in order to authorize and initialize the TLS communication and distribution of keys.

Use case 2:

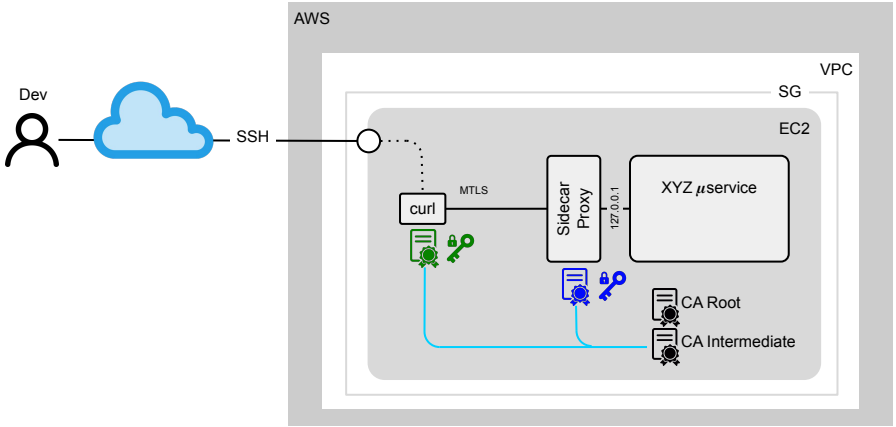
Using Sidecar Pattern



Use case 2: Sidecar Pattern



Use case 2: Hands-on time!



Sidecar Pattern:
- Proxy, Ingress/Egress
- Acme protocol
- Secret Discovery Service (SDS)

Caddy
https://caddyserver.com/docs/quick-starts/reverse-proxy

CA & PKI:
- Step Certificates
- Acme protocol
- Step SDS

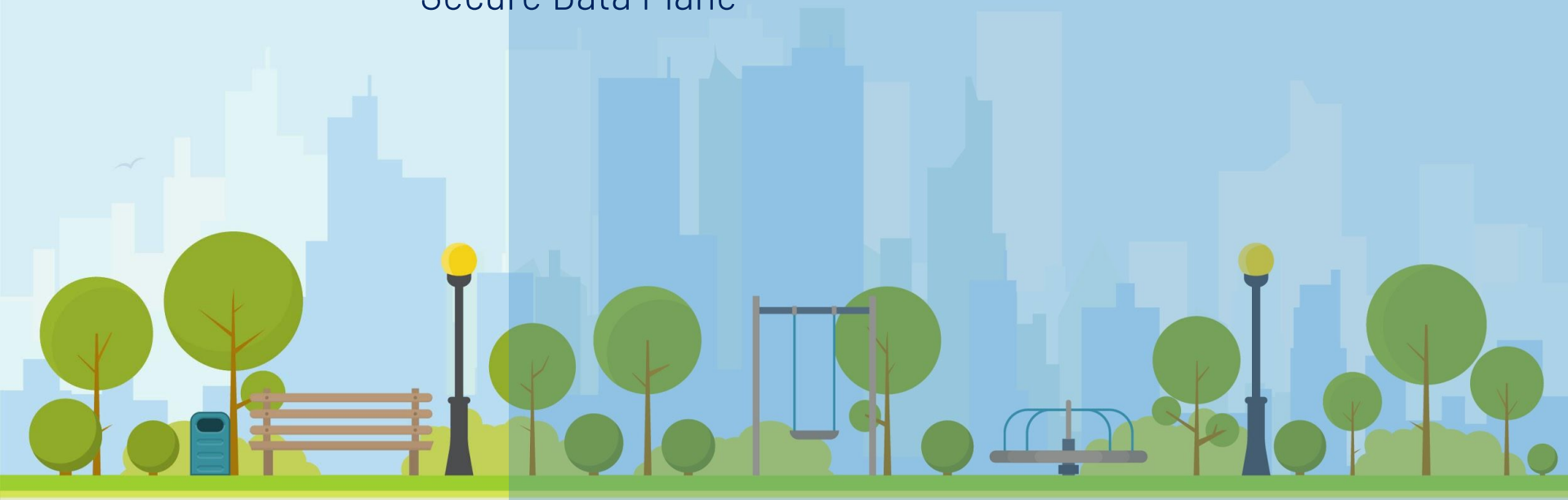
smallstep™
https://github.com/smallstep/step-sds
https://github.com/smallstep/certificates

Use case 2: Conclusions

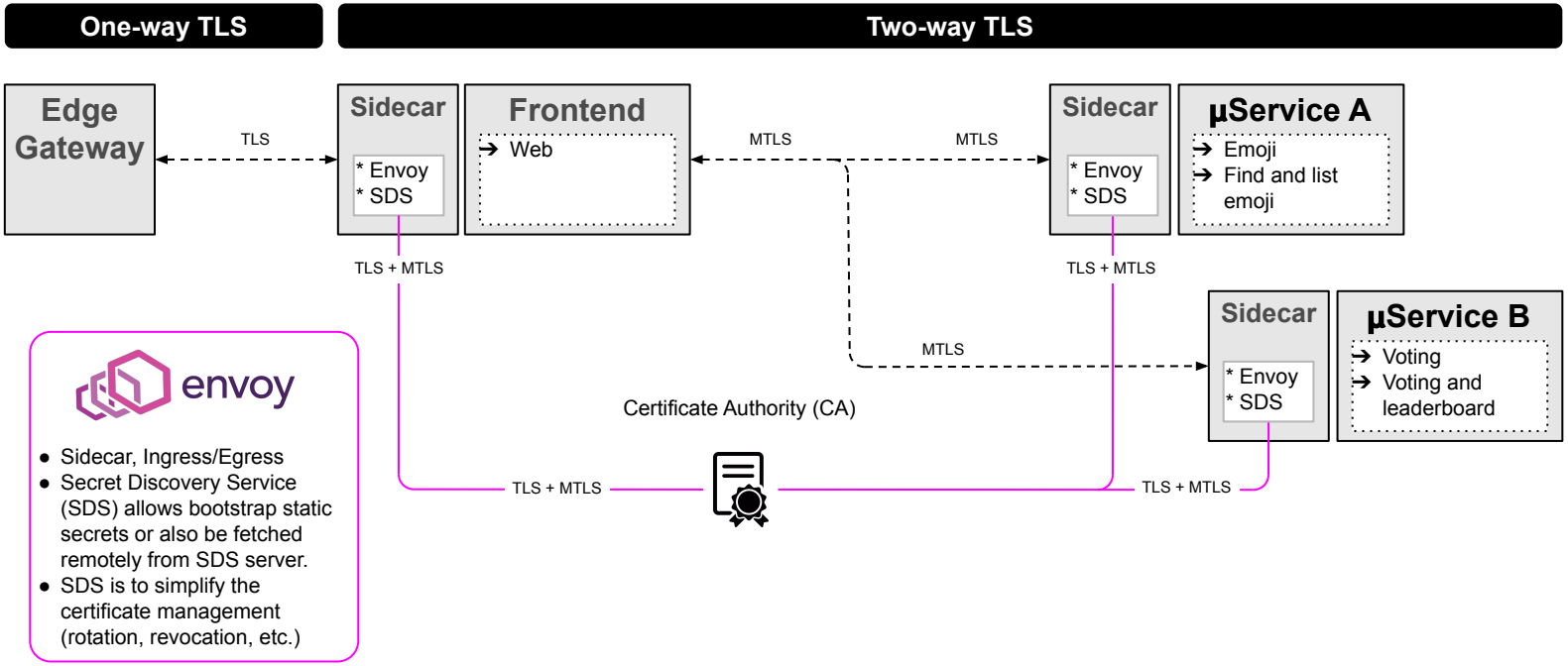
- CA/PKI: SmallStep Certificate
  - Private CA (internal without human interaction) with Acme (RFC 8555) protocol support.
- Sidecar Proxy: Caddy Server
  - HTTP Lightweight Proxy, written in Go without dependencies, powerful API, support HTTPS by default and obtain automatically a Let's Encrypt TLS Certificate.
  - Support Acme protocol and ZeroSSL and can be extended through modules.
  - Lack (?) of monitoring and telemetry features.
  - Chicken-Egg problem: No clear Secret Bootstrapping Process.
- Service Mesh MVP.
  - It does not have all the support of the marketing machinery that Istio, and therefore Envoy Proxy, does, but really you can build a secure Data Plane for your microservices from minute zero.

Use case 3:

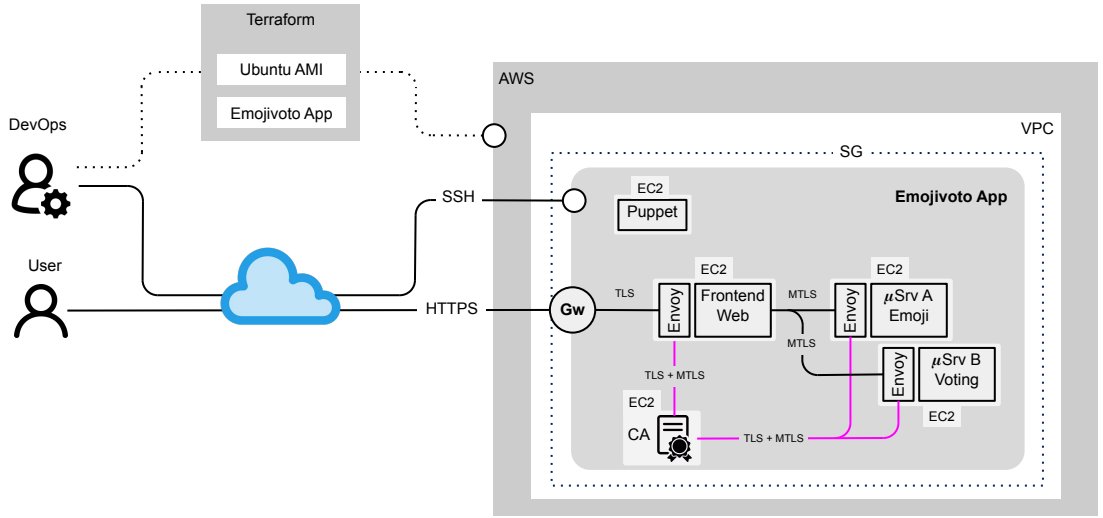
Secure Data Plane



Use case 3: Secure Data Plane



Use case 3: Hands-on time!



App and µservices:
- Emojivoto

https://github.com/buoyantio/emojivoto

Proxy Pattern:
- Sidecar
- Ingress/Egress
- Secret Discovery Service (SDS) API

envoy
https://www.envoyproxy.io/docs/envoy/latest/configuration/security/secret#config-secret-discovery-service

CA & PKI:
- Step Certificates
- Step SDS

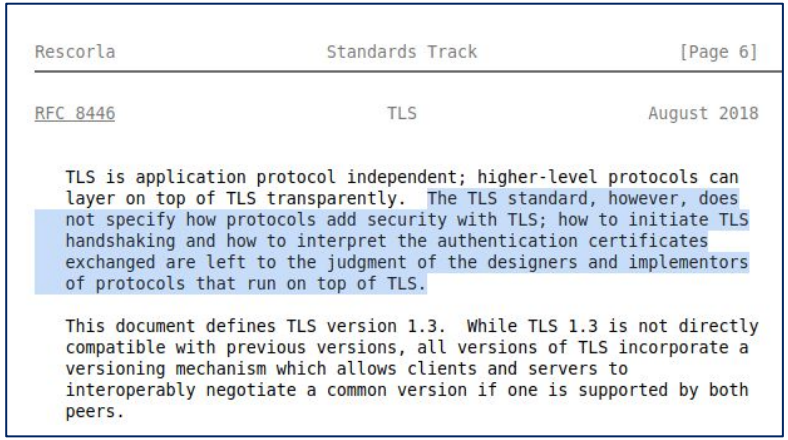
smallstep™
https://github.com/smallstep/step-sds
https://github.com/smallstep/certificates

Conclusions

- Higher latency compared to that of other secure encryption protocols.
  - TSL have a 5ms extra latency
- TLS 1.0 to 1.2 versions still are vulnerable to attacks.
  - Susceptible to MitM attacks, as well as POODLE, DROWN, and SLOTH.
- Few platforms support TLS 1.3.
- Certificates and crypto keys.
  - Too difficult manage the certificate life cycle.

Recommendations

- TLS anywhere and everywhere.
  - Supporting TLS 1.3 you will be able to improve the latency.
  - TLS by default in your Applications.
- Container Patterns.
  - Sidecar, Ambassador, Adapter, Initializer, etc.
- PKI/CA.
  - Automate the certificates lifecycle.
  - Use short-lived certificates.



Rescorla Standards Track [Page 6]

RFC 8446 TLS August 2018

TLS is application protocol independent; higher-level protocols can layer on top of TLS transparently. The TLS standard, however, does not specify how protocols add security with TLS; how to initiate TLS handshaking and how to interpret the authentication certificates exchanged are left to the judgment of the designers and implementors of protocols that run on top of TLS.

This document defines TLS version 1.3. While TLS 1.3 is not directly compatible with previous versions, all versions of TLS incorporate a versioning mechanism which allows clients and servers to interoperably negotiate a common version if one is supported by both peers.

https://tools.ietf.org/html/rfc8446

Thank you

Any questions? Stay for a chat!