Automate your Go TLS certificate distribution

1st Nov 2019

Johan Brandhorst
Utility Warehouse

Today we will

- Learn about TLS and PKI
- How to do this today
- Problems with current approaches
- Introduce Certify
- Show how Certify simplifies TLS and PKI management
- Achieve the fuzzy feeling

Why?

- We write applications
- Applications handle data
- How do we secure our and our customers data?

Knight gopher by Egon Elbre (https://twitter.com/egonelbre)

Transport Layer Security (TLS)

- Formerly named SSL
- TLS 1.3 latest version (shipped with Go 1.12)
- Record and Handshake protocols
- Uses PKI for handshakes

Source: *Netcon Consulting* (https://www.netcon-consulting.com/en/expertise/email.html)

Public Key Infrastructure (PKI)

- Public / Private key pairs <-> users and machines
- Certificates have keys
- Certificate authorities (CA) sign certificates

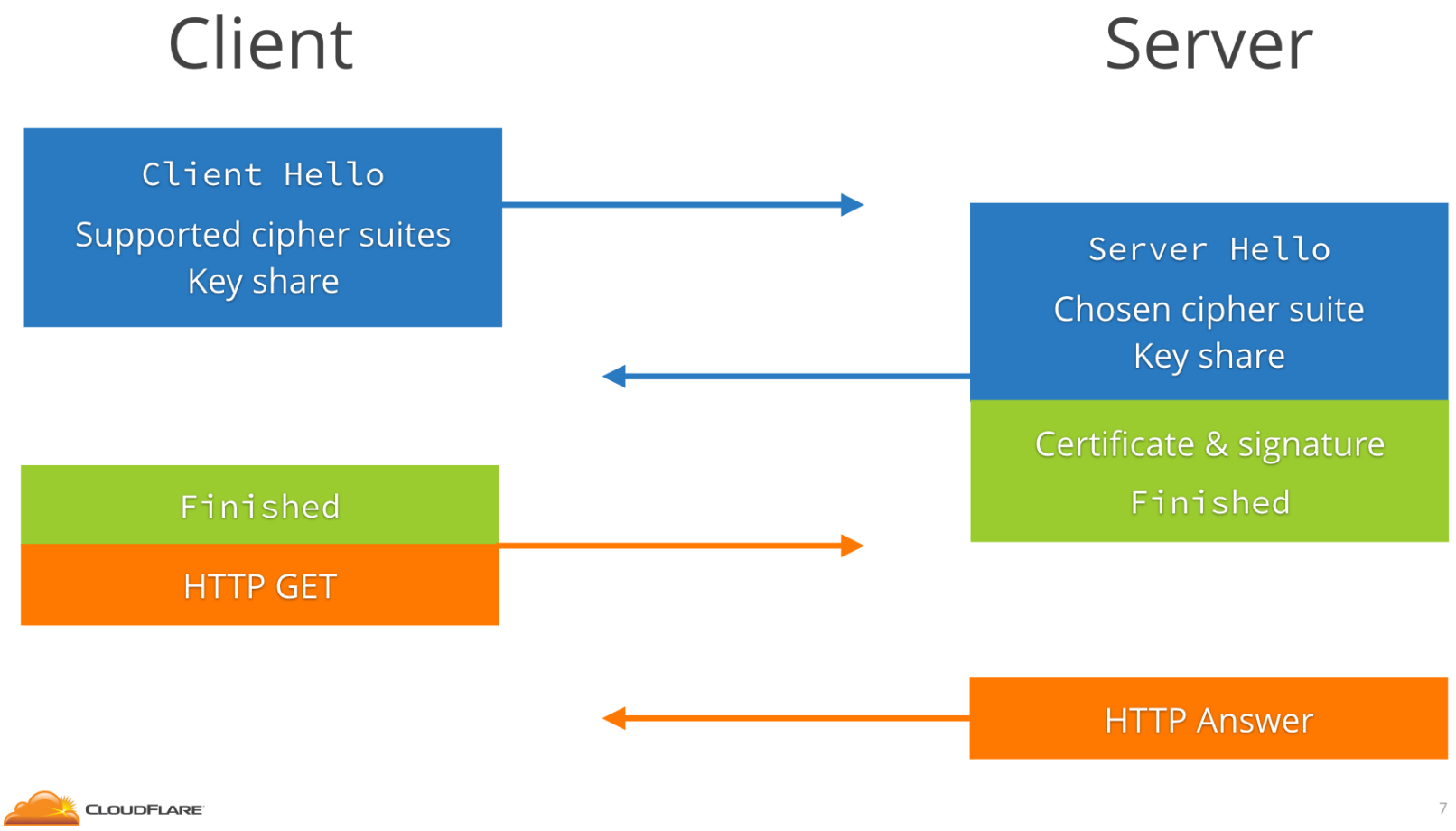
Source: svgrepo.com (https://www.svgrepo.com/svg/69779/certificate)

Certificate signing

- Certificate Signing Request (CSR)
- Contains Subject Alternative Names (SANs), IPSANs
- CA signs CSR, creating a certificate.

Source: svgrepo.com (https://www.svgrepo.com/svg/49687/circular-label-with-certified-stamp)

TLS 1.3 Handshake



Source: Cloudflare (https://blog.cloudflare.com/tls-1-3-overview-and-q-and-a/)

Certificate issuing and renewal

- Self sign / Certificate Authority (💰💰💰)
- Manually mount files
- Can we automate this?

Let's Encrypt

- Free
- Automated (certbot)
- golang.org/x/crypto/acme/autocert (golang.org/x/crypto/acme/autocert)

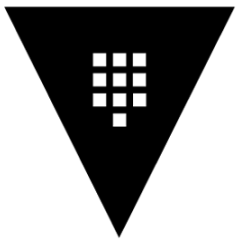
```
func main() {
    m := &autocert.Manager{
        Cache:      autocert.DirCache("secret-dir"),
        Prompt:     autocert.AcceptTOS,
        HostPolicy: autocert.HostWhitelist("example.org", "www.example.org"),
    }
    s := &http.Server{
        Addr:      ":https",
        TLSConfig: m.TLSConfig(),
    }
    s.ListenAndServeTLS("", "")
}
```

Not quite

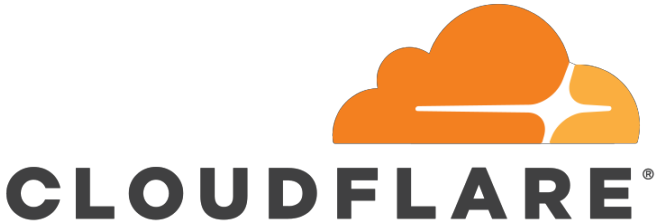
- What about internal taffic?
- What about client side certificates?
- What about rate limits?

CA issuers with APIs

- Hashicorp Vault
- Cloudflare CFSSL
- AWS Certificate Manager Private Certificate Authority (ACMPCA)
- Others

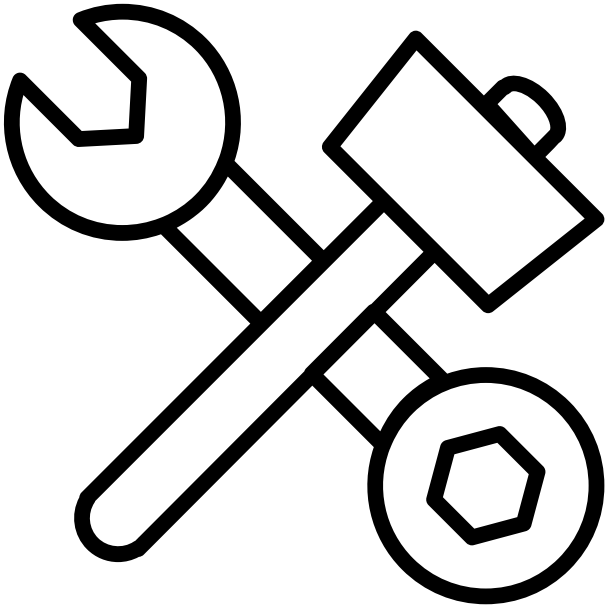


HashiCorp Vault



Running your own CA

- Generate a root certificate
- Generate and sign intermediate certificate
- Distribute root certificate



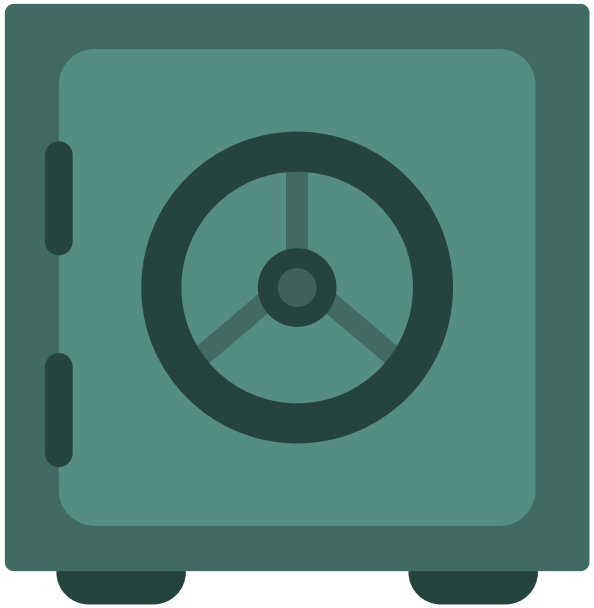
Source: svgrepo.com (https://www.svgrepo.com/svg/35877/hammer-and-double-side-wrench-in-cross)

Generate the root certificate

- Generate the root certificate

```
$ go run $(go env GOROOT)/src/crypto/tls/generate_cert.go -ca -duration 87600h -host "Root"
```

- Store the private key safely!
- Don't be like NordVPN



Source: svgrepo.com (https://www.svgrepo.com/svg/35110/safe-box)

Generate and sign the intermediate certificate

- Generate the intermediate certificate

```
$ vault write -format=json pki/intermediate/generate/internal | jq -r '.data.csr' > int.csr
```

- Sign using root cert and key

```
$ cat > openssl.cnf <<EOF
[ intermediate_ca ]
basicConstraints = critical,CA:true
keyUsage = cRLSign, keyCertSign
EOF
$ openssl x509 -req -extfile openssl.cnf -extensions intermediate_ca \
   -days 365 -in int.csr -CA cert.pem -CAkey key.pem \
   -CAcreateserial -out int.pem -sha256
```

- Store back in vault

```
$ vault write pki/intermediate/set-signed certificate=@int.pem
```

Distribute the root certificate

- Ensure all clients and servers roots are updated

```
// Possible certificate files; stop after finding one.
var certFiles = []string{
    "/etc/ssl/certs/ca-certificates.crt",                // Debian/Ubuntu/Gentoo etc.
    "/etc/pki/tls/certs/ca-bundle.crt",                  // Fedora/RHEL 6
    "/etc/ssl/ca-bundle.pem",                            // OpenSUSE
    "/etc/pki/tls/cacert.pem",                           // OpenELEC
    "/etc/pki/ca-trust/extracted/pem/tls-ca-bundle.pem", // CentOS/RHEL 7
    "/etc/ssl/cert.pem",                                 // Alpine Linux
}
```

Source: crypto/x509 (https://golang.org/src/crypto/x509/root_linux.go)

```
cat cert.pem >> /etc/ssl/cert.pem
```

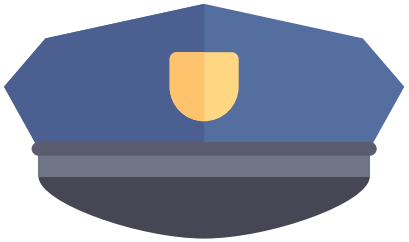
- Ensure applied after base OS updates

Create certificates on demand

- Generate local CSR
- Using Vaults API

```
$ vault write pki/roles/myrole allow_any_name=true
$ vault write pki/sign/myrole csr=@local.csr
```

- How do we ensure authenticity?
- How do we limit who can create what certificate?



Source: svgrepo.com (https://www.svgrepo.com/svg/10013/police-cap)

Vault PKI roles

- Configure roles to use specific domains

```
$ vault write pki/roles/uw allowed_domains=["uw.com", "uw.co.uk"]
$ vault write pki/roles/myapp allowed_domains=["myapp.com", "app.myapp.com"]
```

- Tie tokens to specific roles with policy
- Create users for each role

```
$ vault write auth/userpass/users/uw
$ vault write auth/userpass/users/myapp
```

Authenticated certificates

- Opens the door to mTLS authentication
- Removes the need for traditional machine auth (JWT/Sessions)
- No JWT? No extra roundtrip to create/blacklist tokens

Source: svgrepo.com (https://www.svgrepo.com/svg/10106/fingerprint)

Server side mTLS authentication

```
var tlsConfig = &tls.Config{
    ClientAuth: tls.RequireAndVerifyClientCert,
}
```

```
func VerifyClient(next http.Handler, allowedCNs ...string) http.Handler {
    m := map[string]struct{}{}
    for _, cn := range allowedCNs {
        m[cn] = struct{}{}
    }
    return http.HandlerFunc(func(w http.ResponseWriter, r *http.Request) {
        for _, c := range r.TLS.VerifiedChains {
            if _, ok := m[c[0].Subject.CommonName]; ok {
                next.ServeHTTP(w, r)
                return
            }
        }
        http.Error(w, "client is unauthenticated", http.StatusUnauthorized)
    })
}
```

```
var h = VerifyClient(mux, "uw.com", "uw.co.uk")
```

Putting it all together

- We have an authenticted API for issuing
- We can issue specific certs to specific clients
- Autocert automated this for Let's Encrypt
- Can we automate this in the same way?

Source: wikidata.org (https://www.wikidata.org/wiki/Q42400102)

Introducing Certify

- github.com/johanbrandhorst/certify (github.com/johanbrandhorst/certify)
- As easy as autocert but for your internal services

CERTIFY

Features

- Exposes a simple interface for issuers

```
type Issuer interface {
    Issue(ctx context.Context, cn string, conf *certify.CertConfig) (*tls.Certificate, error)
}
```

- Both client and server side certificates
- Lazy requests certificates as necessary
- Requests are deduplicated
- Sidecar deployment for non-Go apps
- Supports Vault, CFSSL and AWS CM PCA

HashiCorp Vault

Using the Vault issuer

```
issuer := &vault.Issuer{
    // URL, Token and Role are required
    URL: &url.URL{
        Scheme: "https",
        Host:   "my-local-vault-instance.com",
    },
    Token: "myVaultToken",
    Role:  "myVaultRole",
    // TimeToLive configures how long new certificates should
    // be valid for. Defaults to the max of the role.
    TimeToLive: time.Hour * 24 * 30,
    // OtherSubjectAlternativeNames defines custom OID/UTF8-string SANs.
    OtherSubjectAlternativeNames: nil,
}
```

Configure the server

```
c := &certify.Certify{
    CommonName: "uw.co.uk",
    Issuer:     issuer,
    // Cache certificates in memory.
    Cache: certify.NewMemCache(),
    // Refresh cached certificates when < 24H left before expiry.
    RenewBefore: 24 * time.Hour,
    // Optionally configure properties of the certificates.
    CertConfig: &certify.CertConfig{
        SubjectAlternativeNames:   []string{"uw.com"},
        IPSubjectAlternativeNames: []net.IP{net.IPv6loopback},
        // Can also specify a custom private key generator, defaults to ECDSA p256.
    },
}
```

```
s := &http.Server{
    Addr: ":https",
    TLSConfig: &tls.Config{
        GetCertificate:       c.GetCertificate,
        GetClientCertificate: c.GetClientCertificate,
    },
}
```

Certify sidecar

- jfbrandhorst/certify (https://hub.docker.com/r/jfbrandhorst/certify)
- Proxies incoming traffic
- Configured via the environment
- Supports Vault/CFSSL/AWS issuers out of the box

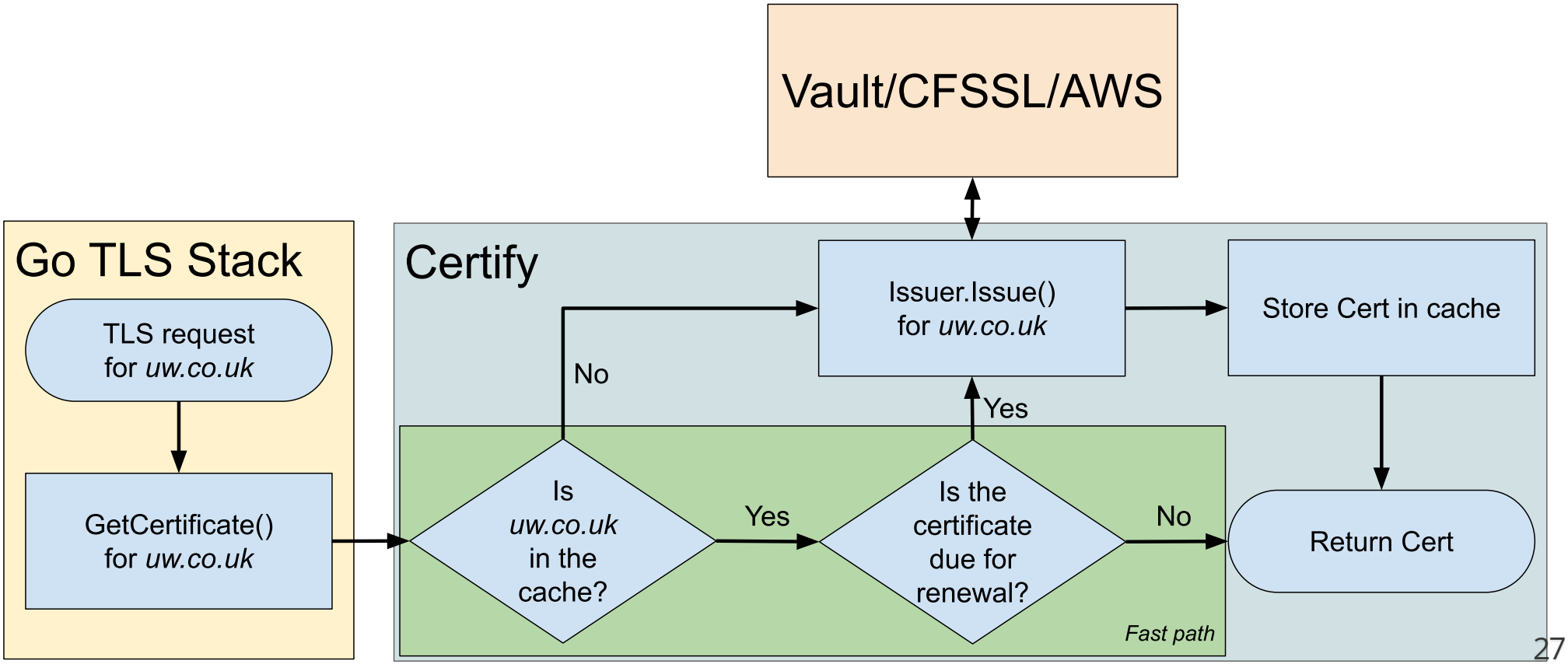
```
$ docker run --rm -it \
   -e ISSUER=vault \
   -e COMMON_NAME=uw.co.uk \
   -e VAULT_URL=https://my-vault-instance.internal.network \
   -e VAULT_TOKEN=TOKENSOMETHINGSOMETHING \
   -e VAULT_ROLE=uw \
   -e LOG_FORMAT=text \
   jfbrandhorst/certify
INFO[0000] Proxying traffic from ":443" to "localhost:80"
```

How does it work?

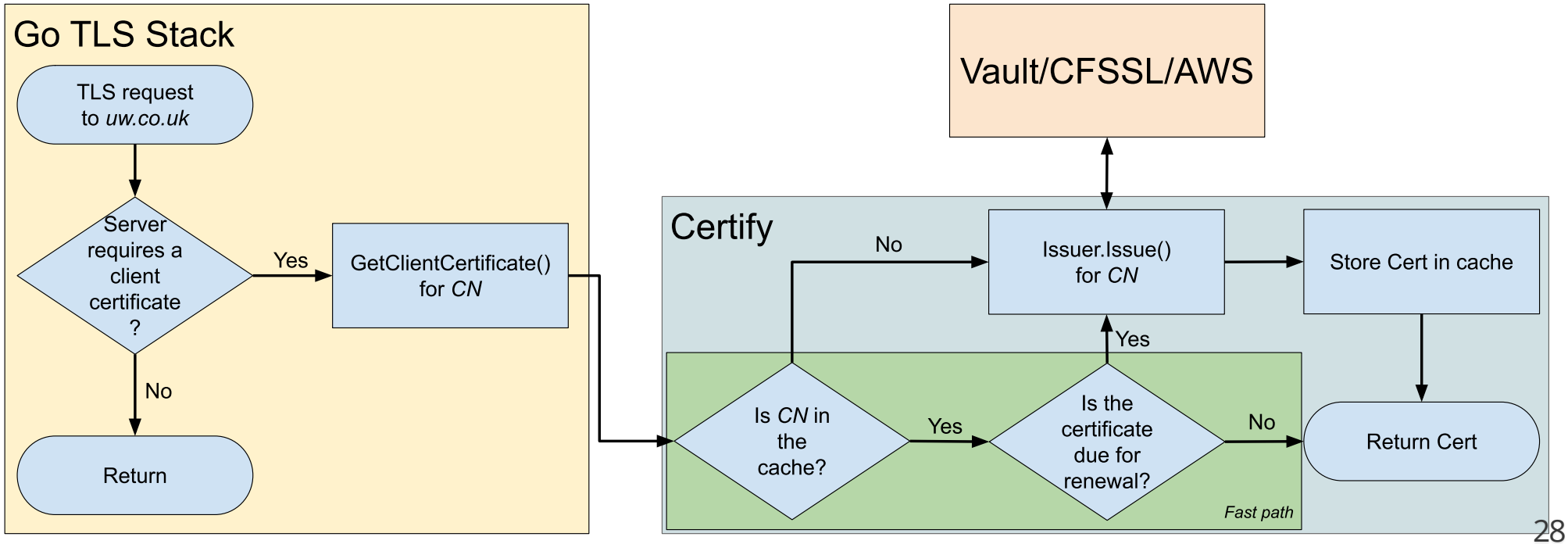
```
type Config struct {
    ...
    // GetCertificate returns a Certificate based on the given
    // ClientHelloInfo. It will only be called if the client supplies SNI
    // information or if Certificates is empty.
    GetCertificate func(*ClientHelloInfo) (*Certificate, error) // Go 1.4

    // GetClientCertificate, if not nil, is called when a server requests a
    // certificate from a client. If set, the contents of Certificates will
    // be ignored.
    GetClientCertificate func(*CertificateRequestInfo) (*Certificate, error) // Go 1.8
    ...
}
```

Source: crypto/tls (https://golang.org/pkg/crypto/tls/#Config)

Server side


Vault/CFSSL/AWS
Go TLS Stack
TLS request for uw.co.uk
GetCertificate() for uw.co.uk
Certify
Issuer.Issue() for uw.co.uk
Store Cert in cache
No
Yes
Is uw.co.uk in the cache?
Yes
Is the certificate due for renewal?
No
Return Cert
Fast path

Client side


Go TLS Stack
TLS request to *uw.co.uk*
Server requires a client certificate ?
Yes
No
GetClientCertificate() for *CN*
Return
Vault/CFSSL/AWS
Certify
No
Issuer.Issue() for *CN*
Store Cert in cache
Yes
Is *CN* in the cache?
Yes
Is the certificate due for renewal?
No
Return Cert
Fast path

The Vault issuer

- github.com/johanbrandhorst/certify/issuers/vault (https://github.com/johanbrandhorst/certify/tree/master/issuers/vault)
- Generate a CSR
- Sign CSR with Vault API client

Sign Certificate

This endpoint signs a new certificate based upon the provided CSR and the supplied parameters, subject to the restrictions contained in the role named in the endpoint. The issuing CA certificate is returned as well, so that only the root CA need be in a client's trust store.

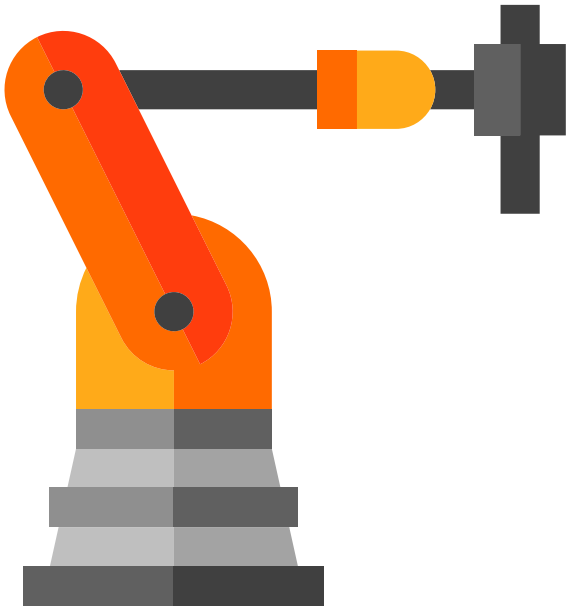
Method	Path
`POST`	`/pki/sign/:name`

Source: Vault PKI API (https://www.vaultproject.io/api/secret/pki/index.html#sign-certificate)

- Parse certificate from response

Taking this one step further

- We have automated the whole workflow
- We're using certificates for authentication
- Certificates are now as easy to issue as tokens
- Can we use token-scale lifetimes for certificates?



Source: flaticon.com (https://www.flaticon.com/free-icon/machinery_145584)

Conclusion

- We learned about TLS and PKI
- We deployed Vault with PKI
- We configured Vault to discriminate users
- We automated our internal TLS certificate deployment with Certify
- We retired tokens for internal service authentication
- One step closer to that fuzzy feeling

Thank you

Johan Brandhorst
Utility Warehouse
@johanbrandhorst (http://twitter.com/johanbrandhorst)
https://jbrandhorst.com (https://jbrandhorst.com)