信任链管理

Austin Zhai
2023.02.05

1. TL;DR

CA:

```
openssl genrsa -out ca.key 2048
openssl req -new -x509 -days 365 -key ca.key -out ca.crt
```

Server:

```
openssl genrsa -out server.key 2048
openssl req -new -key server.key -out server.csr
faketime '2022-12-01 12:00:00' openssl x509 -req -days 10 -in server.csr \
    -CA ../ca/ca.crt -CAkey ../ca/ca.key -CAcreateserial -out server.crt
```

1. TL;DR

```go
package main

import (
    "log"
    "net/http"
)

func handler(w http.ResponseWriter, req *http.Request) {
    w.Header().Set("Content-Type", "text/plain")
    w.Write([]byte("This is an example server.\n"))
}

func main() {
    http.HandleFunc("/", handler)
    err := http.ListenAndServeTLS("0.0.0.0:443",
        "server.crt", "server.key", nil)
    log.Fatal(err)
}
```

```go
package main

import (
    "crypto/tls"
    "crypto/x509"
    "io/ioutil"
    "log"
    "net/http"
)

func main() {
    ca, err := ioutil.ReadFile("ca.crt")
    if err != nil {
        log.Fatal(err)
    }
    pool := x509.NewCertPool()
    pool.AppendCertsFromPEM(ca)

    client := &http.Client{
        Transport: &http.Transport{
            TLSClientConfig: &tls.Config{
                RootCAs: pool,
                //InsecureSkipVerify: true,
            },
        },
    }
    _, err = client.Get("https://www.austin.com")
    if err != nil {
        log.Fatal(err)
    }
}
```

Get "https://www.austin.com": x509: certificate has expired or is not yet valid: current time 2023-02-08T21:19:15+08:00 is after 2022-12-11T04:00:00Z

1. TL;DR

```
go install gitlab.moresec.cn/moresec/cert
```

Mac:

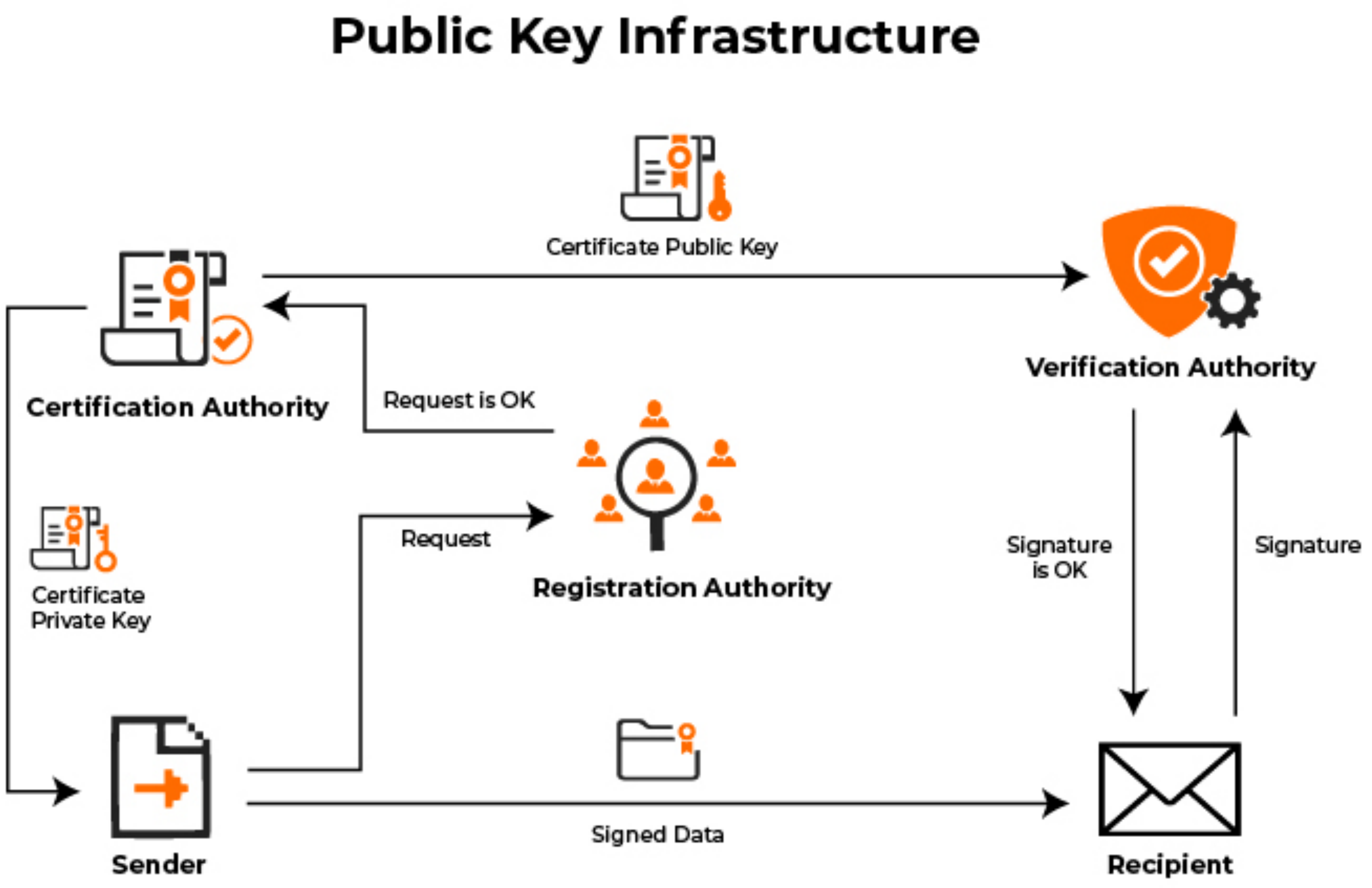
```
cert -p ~/ -lt $(date -j -f "%Y:%m:%d %H:%M:%S" '2024:02:01 22:00:05' +%s)
```

Linux:

```
cert -p ~/ -lt $(date -d '2024-02-01 22:00:05' +%s)
```

2. 信任体系

PKI



- CA
- RA
- VA
- PKCS

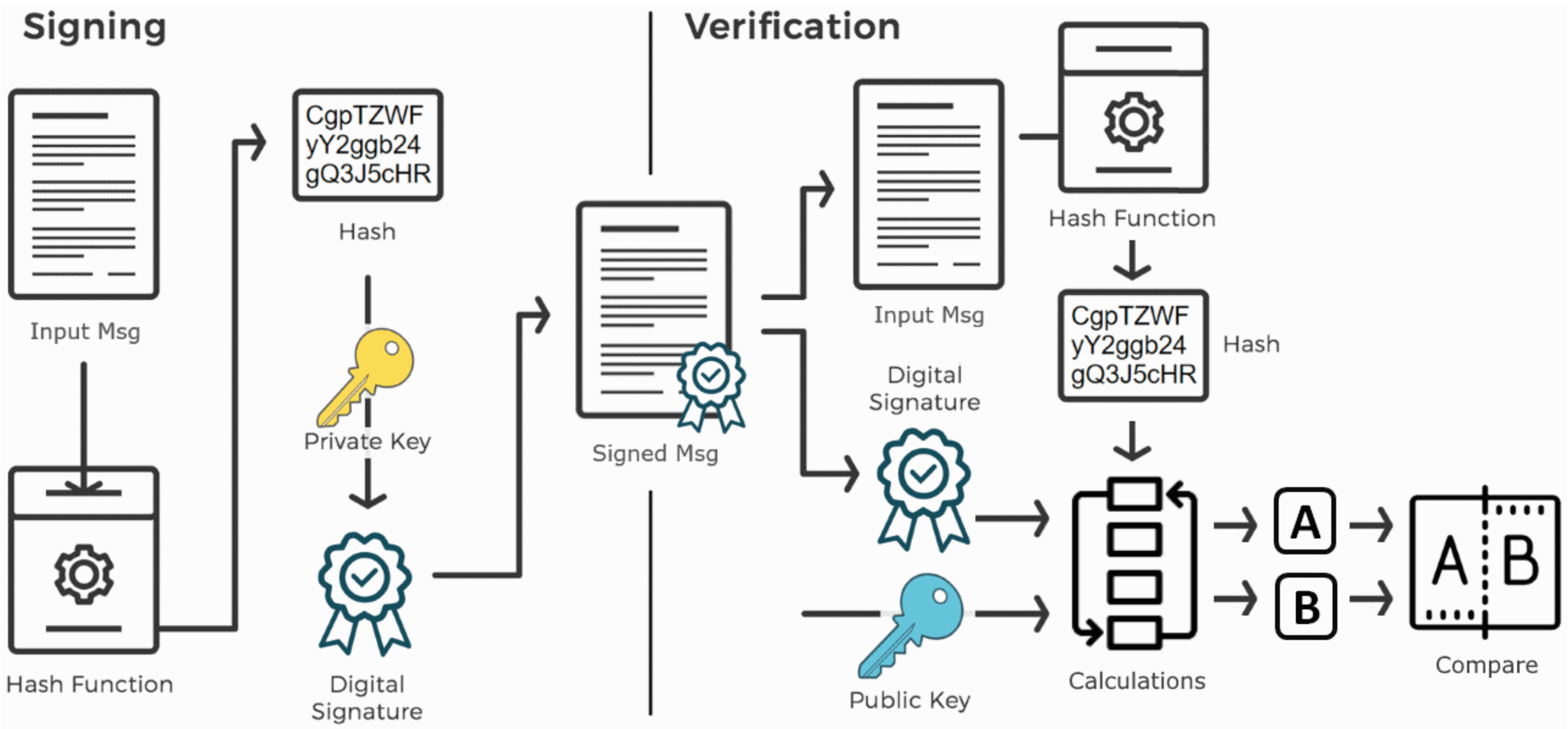
2. 信任体系

其他

- PGP (Pretty Good Privacy)
- SPKI (Simple Public Key Infrastructure)
- 区块链
- …

3. 信任模型

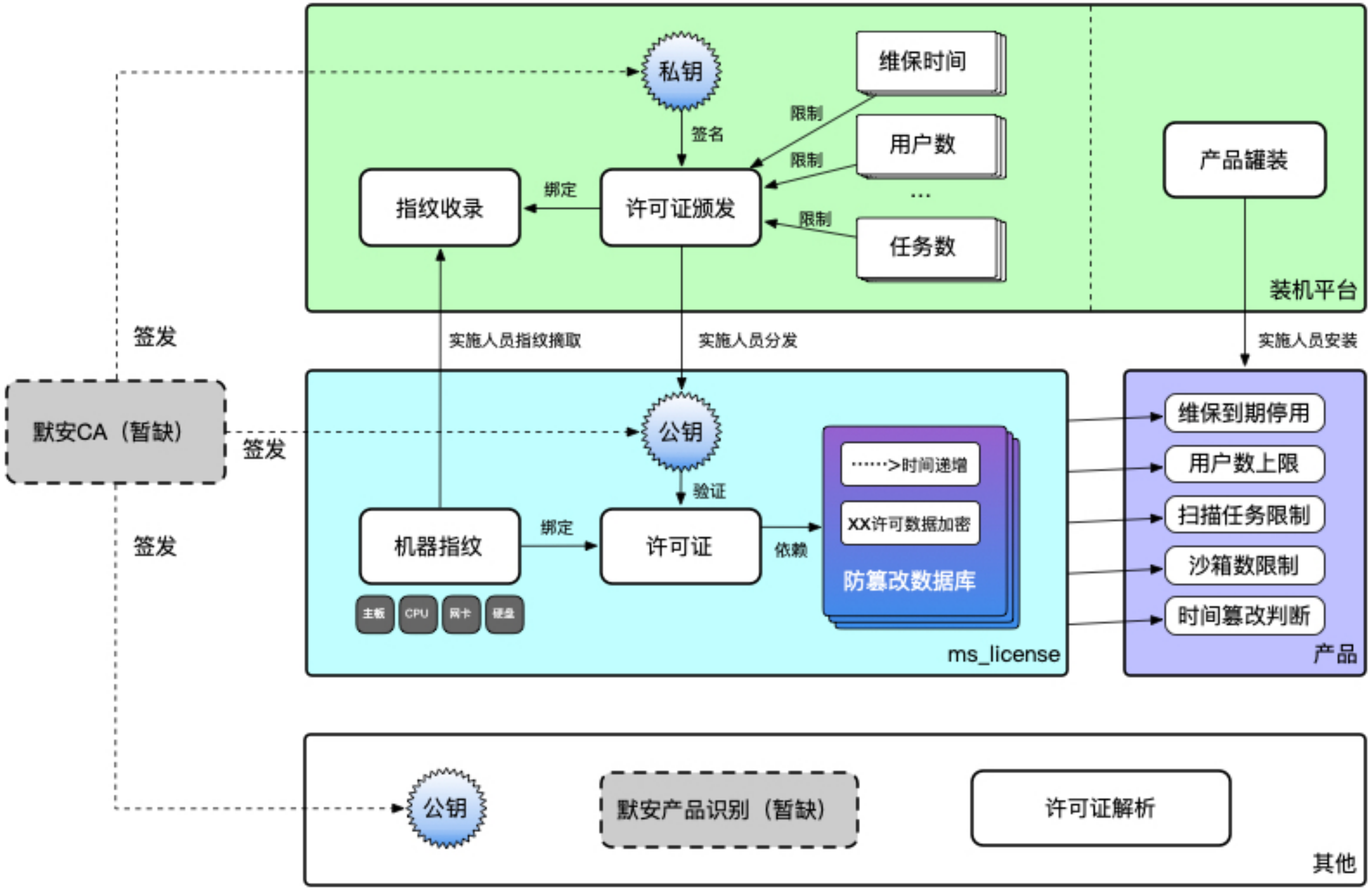
签名



Signing
Input Msg
Hash Function
CgpTZWF
yY2ggb24
gQ3J5cHR
Hash
Private Key
Digital
Signature
Signed Msg
Verification
Input Msg
Hash Function
CgpTZWF
yY2ggb24
gQ3J5cHR
Hash
Digital
Signature
Public Key
Calculations
A
B
A
B
Compare

3. 信任模型

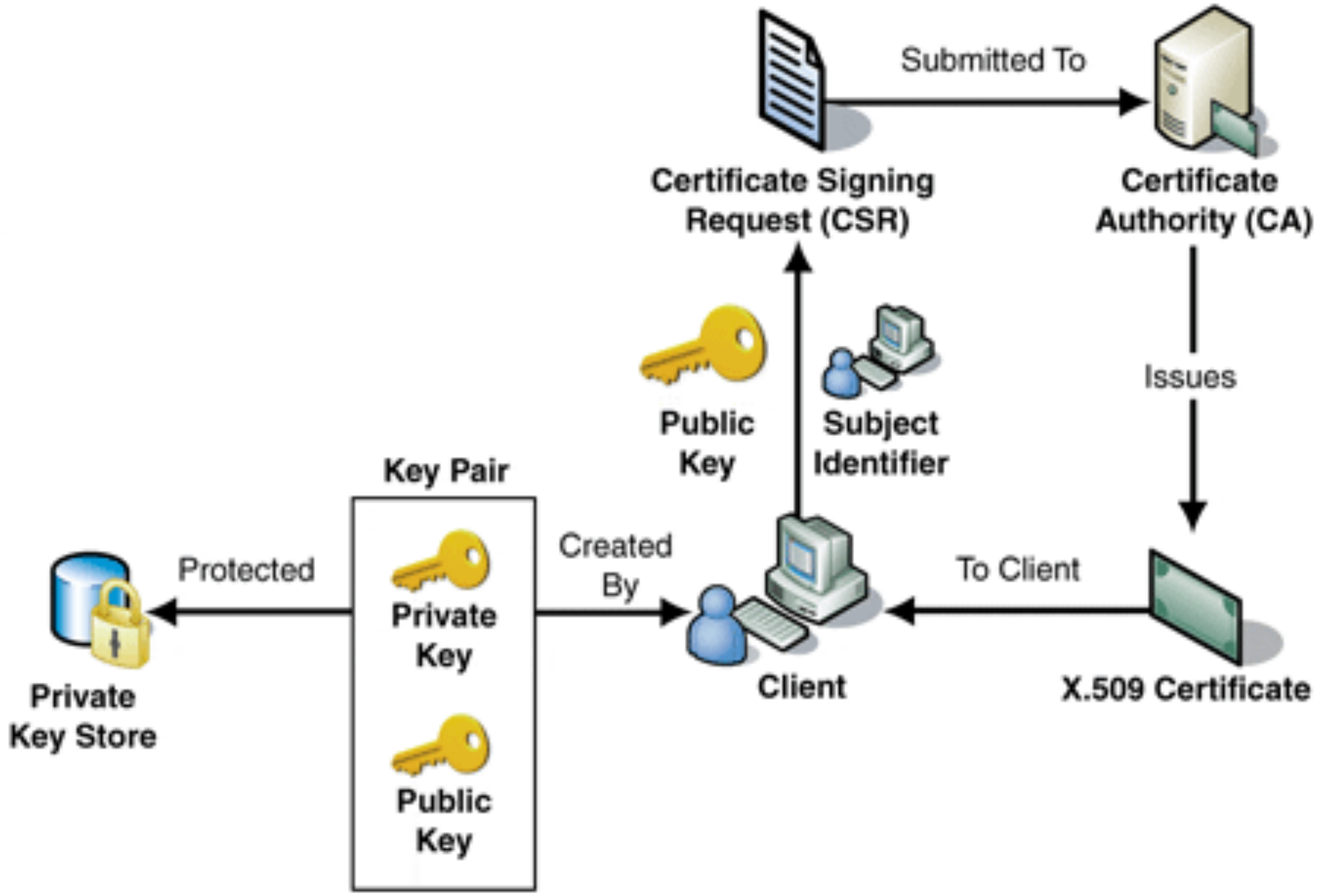
签名应用



私钥
维保时间
限制
用户数
限制
签名
绑定
指纹收录
许可证颁发
...
限制
任务数
产品罐装
装机平台
签发
实施人员指纹摘取
实施人员分发
实施人员安装
默安CA（暂缺）
签发
公钥
……>时间递增
XX许可数据加密
验证
绑定
依赖
机器指纹
许可证
防篡改数据库
主板
CPU
网卡
硬盘
ms_license
维保到期停用
用户数上限
扫描任务限制
沙箱数限制
时间篡改判断
产品
签发
公钥
默安产品识别（暂缺）
许可证解析
其他

3. 信任模型

CA签名



注意：CA是不会拿到私钥的

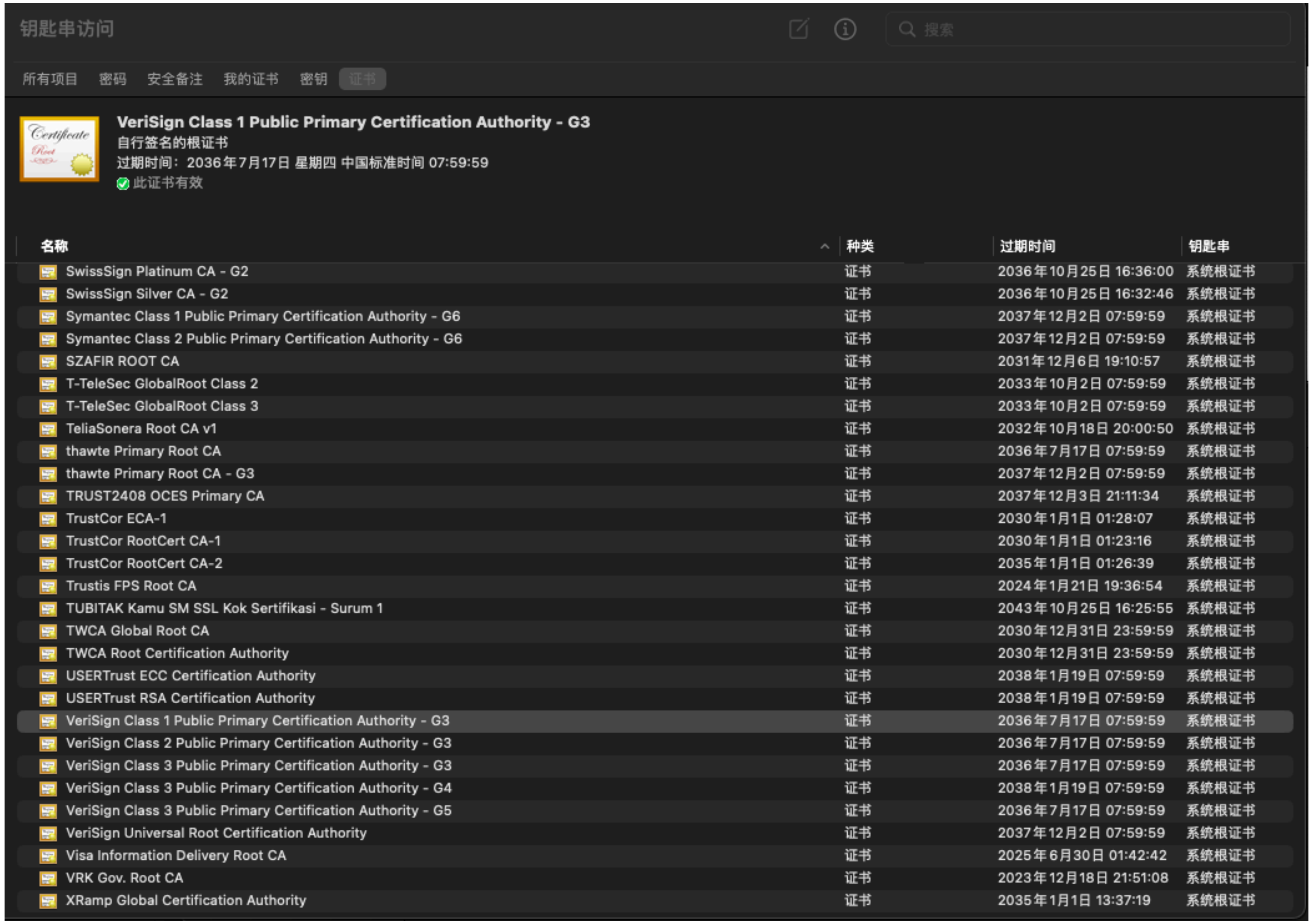
openssl genrsa -out server.key 2048

openssl req -new -key server.key -out server.csr

openssl req -text -in server.csr -noout

3. 信任模型

谁信任CA



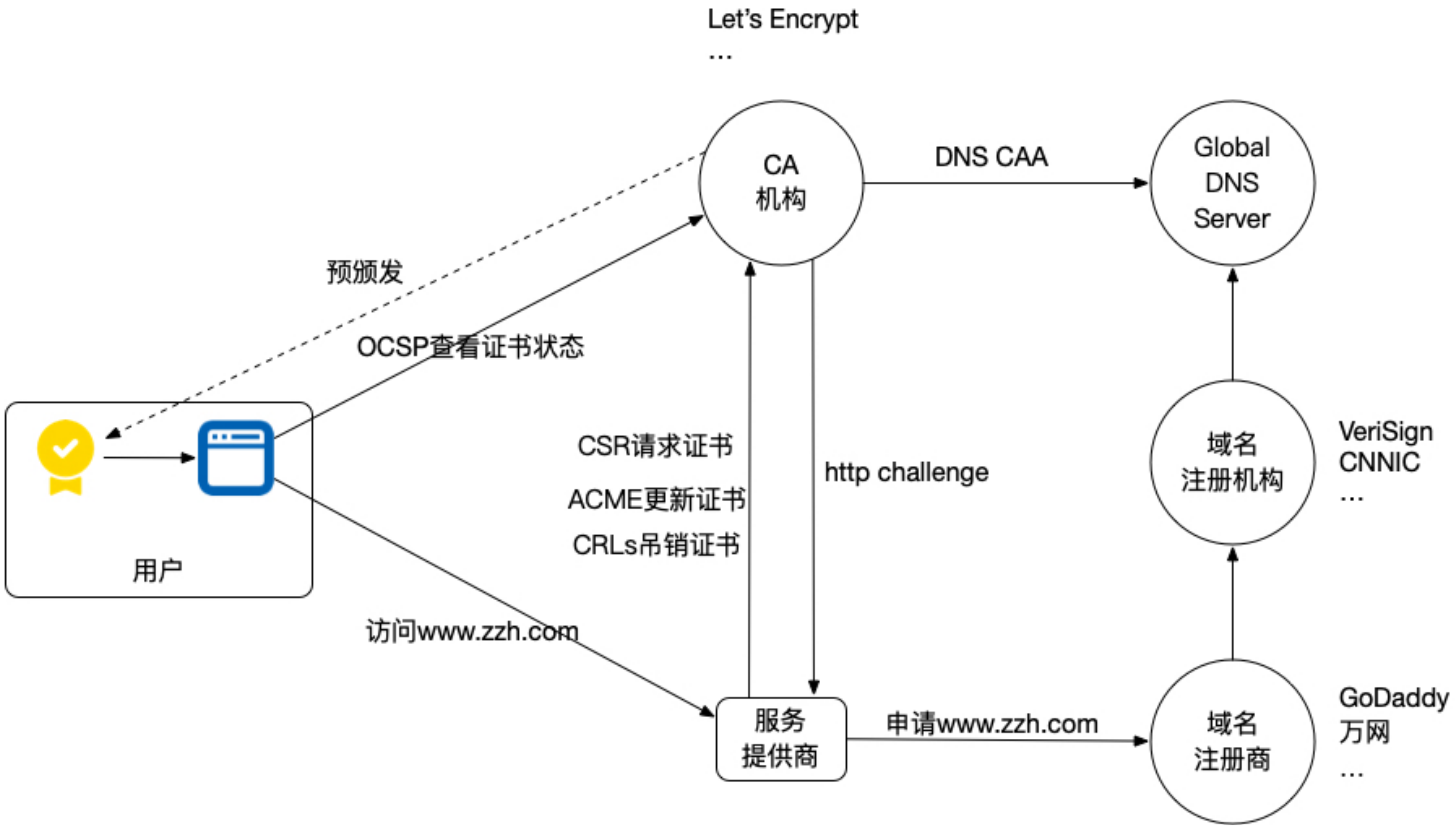
名称	种类	过期时间	钥匙串
SwissSign Platinum CA - G2	证书	2036年10月25日 16:36:00	系统根证书
SwissSign Silver CA - G2	证书	2036年10月25日 16:32:46	系统根证书
Symantec Class 1 Public Primary Certification Authority - G6	证书	2037年12月2日 07:59:59	系统根证书
Symantec Class 2 Public Primary Certification Authority - G6	证书	2037年12月2日 07:59:59	系统根证书
SZAFIR ROOT CA	证书	2031年12月6日 19:10:57	系统根证书
T-TeleSec GlobalRoot Class 2	证书	2033年10月2日 07:59:59	系统根证书
T-TeleSec GlobalRoot Class 3	证书	2033年10月2日 07:59:59	系统根证书
TeliaSonera Root CA v1	证书	2032年10月18日 20:00:50	系统根证书
thawte Primary Root CA	证书	2036年7月17日 07:59:59	系统根证书
thawte Primary Root CA - G3	证书	2037年12月2日 07:59:59	系统根证书
TRUST2408 OCES Primary CA	证书	2037年12月3日 21:11:34	系统根证书
TrustCor ECA-1	证书	2030年1月1日 01:28:07	系统根证书
TrustCor RootCert CA-1	证书	2030年1月1日 01:23:16	系统根证书
TrustCor RootCert CA-2	证书	2035年1月1日 01:26:39	系统根证书
Trustis FPS Root CA	证书	2024年1月21日 19:36:54	系统根证书
TUBITAK Kamu SM SSL Kok Sertifikasi - Surum 1	证书	2043年10月25日 16:25:55	系统根证书
TWCA Global Root CA	证书	2030年12月31日 23:59:59	系统根证书
TWCA Root Certification Authority	证书	2030年12月31日 23:59:59	系统根证书
USERTrust ECC Certification Authority	证书	2038年1月19日 07:59:59	系统根证书
USERTrust RSA Certification Authority	证书	2038年1月19日 07:59:59	系统根证书
VeriSign Class 1 Public Primary Certification Authority - G3	证书	2036年7月17日 07:59:59	系统根证书
VeriSign Class 2 Public Primary Certification Authority - G3	证书	2036年7月17日 07:59:59	系统根证书
VeriSign Class 3 Public Primary Certification Authority - G3	证书	2036年7月17日 07:59:59	系统根证书
VeriSign Class 3 Public Primary Certification Authority - G4	证书	2038年1月19日 07:59:59	系统根证书
VeriSign Class 3 Public Primary Certification Authority - G5	证书	2036年7月17日 07:59:59	系统根证书
VeriSign Universal Root Certification Authority	证书	2037年12月2日 07:59:59	系统根证书
Visa Information Delivery Root CA	证书	2025年6月30日 01:42:42	系统根证书
VRK Gov. Root CA	证书	2023年12月18日 21:51:08	系统根证书
XRamp Global Certification Authority	证书	2035年1月1日 13:37:19	系统根证书

3. 信任模型

信任自建CA

3. 信任模型

互联网



Let's Encrypt
...
CA
机构
DNS CAA
Global
DNS
Server
预颁发
OCSP查看证书状态
CSR请求证书
ACME更新证书
CRLs吊销证书
http challenge
域名
注册机构
VeriSign
CNNIC
...
用户
访问www.zzh.com
服务
提供商
申请www.zzh.com
域名
注册商
GoDaddy
万网
...

3. 信任模型

CA 私钥泄漏?

CA compromise [edit]

See also: Supply chain attack

If the CA can be subverted, then the security of the entire system is lost, potentially subverting all the entities that trust the compromised CA.

For example, suppose an attacker, Eve, manages to get a CA to issue to her a certificate that claims to represent Alice. That is, the certificate would publicly state that it represents Alice, and might include other information about Alice. Some of the information about Alice, such as her employer name, might be true, increasing the certificate's credibility. Eve, however, would have the all-important private key associated with the certificate. Eve could then use the certificate to send digitally signed email to Bob, tricking Bob into believing that the email was from Alice. Bob might even respond with encrypted email, believing that it could only be read by Alice, when Eve is actually able to decrypt it using the private key.

A notable case of CA subversion like this occurred in 2001, when the certificate authority VeriSign issued two certificates to a person claiming to represent Microsoft. The certificates have the name "Microsoft Corporation", so they could be used to spoof someone into believing that updates to Microsoft software came from Microsoft when they actually did not. The fraud was detected in early 2001. Microsoft and VeriSign took steps to limit the impact of the problem.[40][41]

In 2008, Comodo reseller Certstar sold a certificate for mozilla.com to Eddy Nigg, who had no authority to represent Mozilla.[42]

In 2011 fraudulent certificates were obtained from Comodo and DigiNotar,[43][44] allegedly by Iranian hackers. There is evidence that the fraudulent DigiNotar certificates were used in a man-in-the-middle attack in Iran.[45]

In 2012, it became known that Trustwave issued a subordinate root certificate that was used for transparent traffic management (man-in-the-middle) which effectively permitted an enterprise to sniff SSL internal network traffic using the subordinate certificate.[46]

3. 信任模型

数字证书x509

1. PKI + X509 = PKIX
2. ITU-T + RFC5280标准

6. bash

```
$ step certificate inspect https://smallstep.com
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 315176808519640433969378695694933015829744 (0x39e38ad549791a5c79d50e5d040f0ba9cf0)
    Signature Algorithm: SHA256-RSA
        Issuer: C=US,O=Let's Encrypt,CN=Let's Encrypt Authority X3
        Validity
            Not Before: Oct 11 15:40:10 2018 UTC
            Not After : Jan 9 15:40:10 2019 UTC
        Subject: CN=smallstep.com
        Subject Public Key Info:
            Public Key Algorithm: ECDSA
                Public-Key: (256 bit)
                X:
                    ed:68:3b:c4:84:b8:a3:9f:38:29:7f:fb:2f:cb:6b:
                    f4:94:0d:2a:d4:a3:4c:98:a9:9b:f2:47:48:80:5b:
                    22:46
                Y:
                    a9:b3:87:42:0f:95:d8:a9:ad:6e:bf:cc:80:4f:28:
                    2d:d9:58:1c:dd:c3:4e:99:b9:25:de:b8:f4:34:ad:
                    c0:b5
                Curve: P-256
        X509v3 extensions:
            X509v3 Key Usage: critical
                Digital Signature
            X509v3 Extended Key Usage:
                TLS Web Server Authentication, TLS Web Client Authentication
            X509v3 Basic Constraints: critical
                CA:FALSE
            X509v3 Subject Key Identifier:
                39:A7:D6:A9:16:89:64:0C:5B:88:4F:21:67:04:AE:01:C4:4E:BB:74
            X509v3 Authority Key Identifier:
                keyid:A8:4A:6A:63:04:7D:DD:BA:E6:D1:39:B7:A6:45:65:EF:F3:A8:EC:A1
            Authority Information Access:
                OCSP - URI:http://ocsp.int-x3.letsencrypt.org
                CA Issuers - URI:http://cert.int-x3.letsencrypt.org/

            X509v3 Subject Alternative Name:
                DNS:smallstep.com
            X509v3 Certificate Policies:
                Policy: 2.23.140.1.2.1
                Policy: 1.3.6.1.4.1.44947.1.1.1
            Unknown extension 1.3.6.1.4.1.11129.2.4.2

    Signature Algorithm: SHA256-RSA
         2f:08:4b:b1:56:37:12:4f:e4:0f:9e:44:5d:5a:42:31:92:47:
         34:ff:1f:db:06:dc:a8:6f:9c:89:8d:4e:58:0b:e8:a5:f7:73:
         5b:41:8e:65:77:fd:39:cf:98:5c:8f:b3:61:10:f2:42:80:a3:
         77:0a:e7:4b:2c:ca:ed:45:1a:a7:6d:d8:9b:41:06:95:ed:ae:
         d1:90:45:1d:06:32:c6:2a:3e:94:15:a1:86:3e:e0:af:57:d5:
         b4:a0:e9:db:6c:32:88:a2:98:53:bd:72:c9:39:e1:3f:54:aa:
         ce:d2:78:eb:59:ad:e6:29:89:ee:80:27:20:b0:c0:40:01:74:
         9e:8d:4c:a1:40:cf:eb:84:c4:14:55:0b:ed:5d:cc:c4:ef:e3:
         c9:3e:48:30:a6:fe:ad:49:dc:aa:7b:2b:95:04:31:ac:47:3f:
         1b:8e:31:6d:29:4c:d1:a9:c1:68:a2:95:c8:20:1d:0d:43:05:
         e6:f2:6c:fd:90:53:1e:fa:b1:01:8c:21:e1:eb:6f:d1:ee:44:
         af:de:81:5a:b2:97:89:b7:c2:29:91:83:29:42:1d:d4:af:7d:
         96:a6:56:7b:8b:ab:b1:50:ac:67:d0:f9:dc:bd:fe:cd:a2:12:
         ca:16:80:c0:a2:02:49:a5:f7:22:2c:22:77:6c:56:d0:87:a2:
         5f:28:bb:c6
$
```

Public Key

Name

Signature

This is an X.509 v3 cert, with a unique serial number issued by Let's Encrypt valid from Oct 11 - Jan 9.

The key is meant for signing stuff - to authenticate using TLS.

This is not a CA's certificate

Unique identifiers for subject & issuer pub keys (used during cert path validation)

OCSP responder URL and certificate URL

Policies the CA follows (mostly ignored)

An extension to support certificate transparency that's not recognized by step cli.

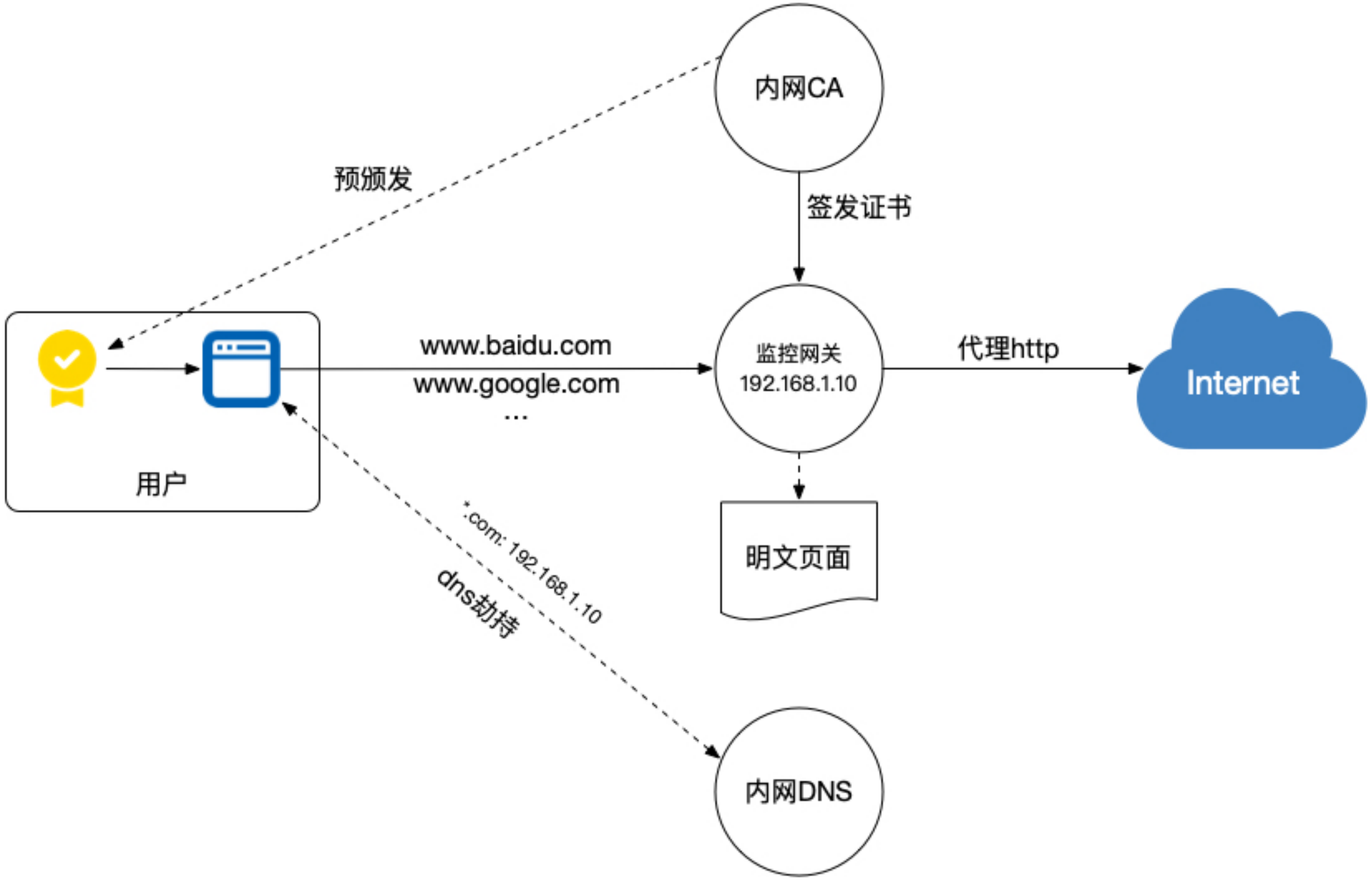
3. 信任模型

证书文件格式

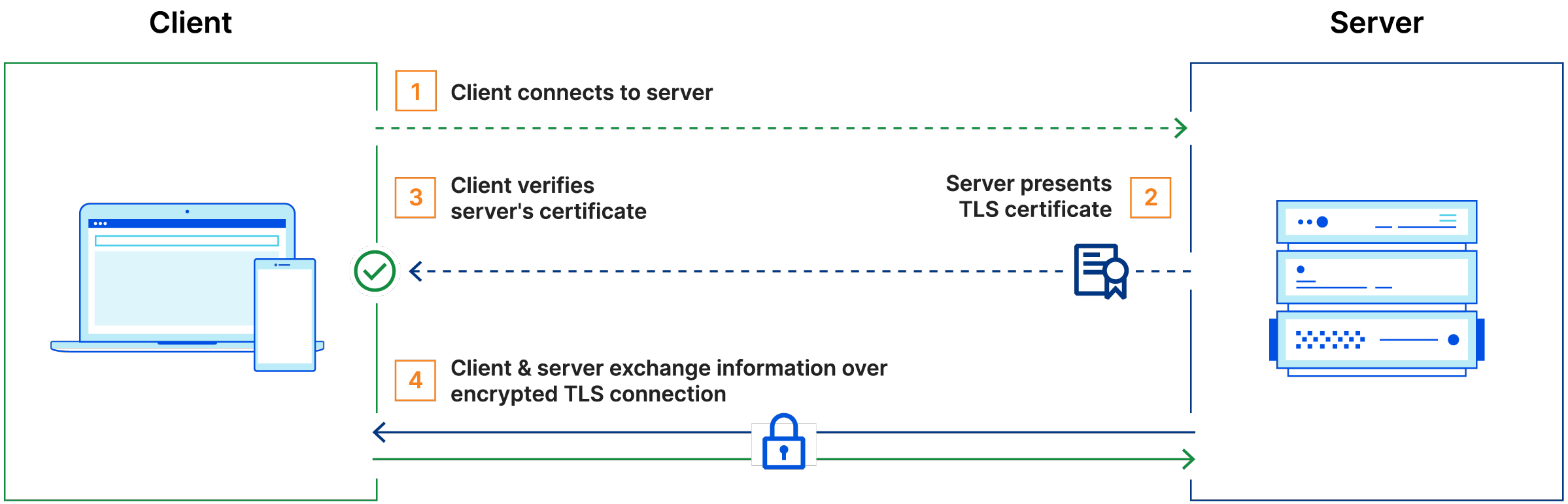
- Pem：rfc1442，存储证书、公私钥，一般base64。后缀.crt .cer .cert .pem .key
- P12：rfc7292 & pkcs12，密码保护的证书文件。后缀.p12 .pfx
- DER：ASN.1格式证书。后缀.der
- P7：pkcs7，保存公钥。后缀.p7b .p7c
- JKS：java key store，方便打包。后缀.jks .keystore

3.信任模型

上网行为监控场景



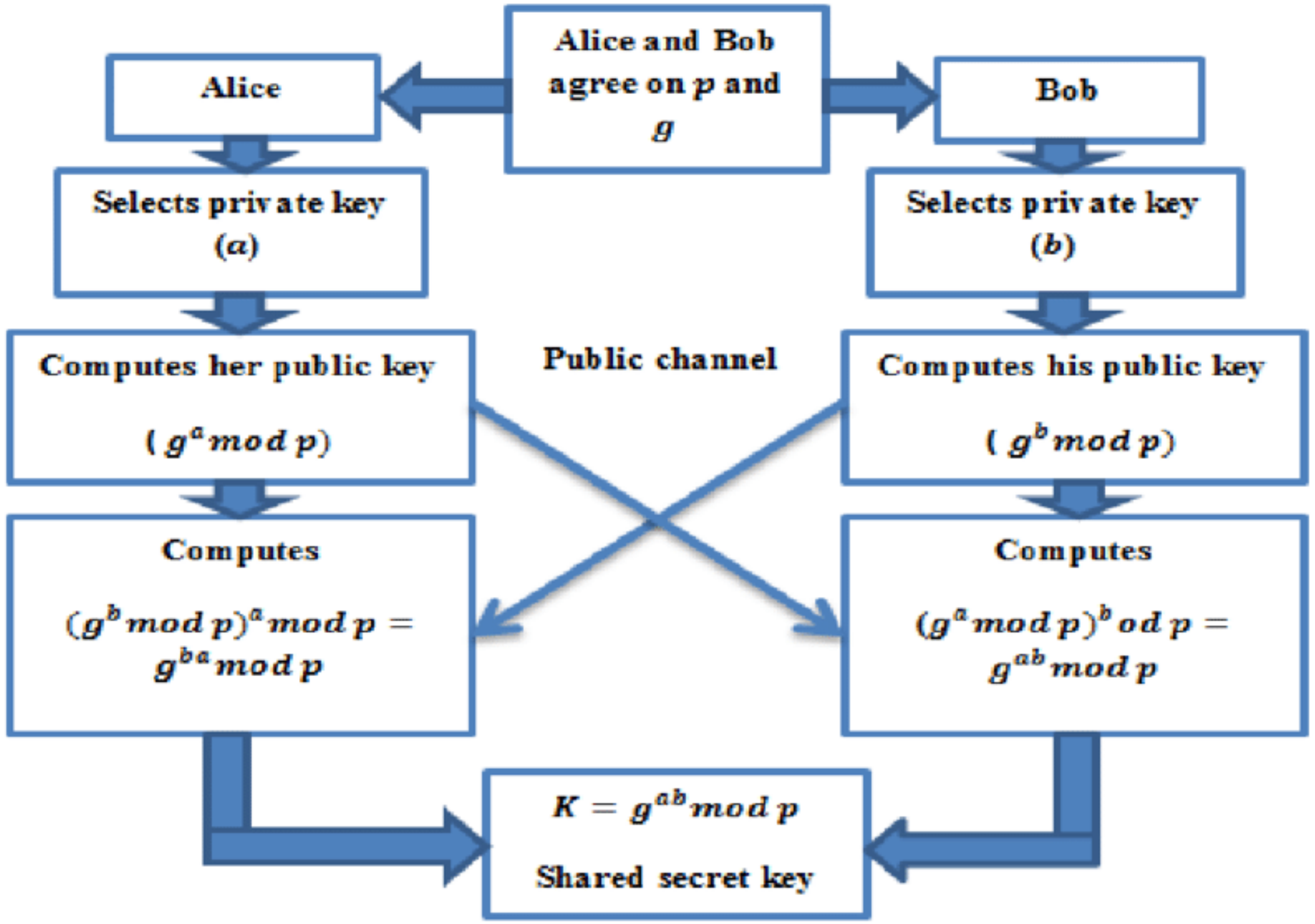
4. TLS



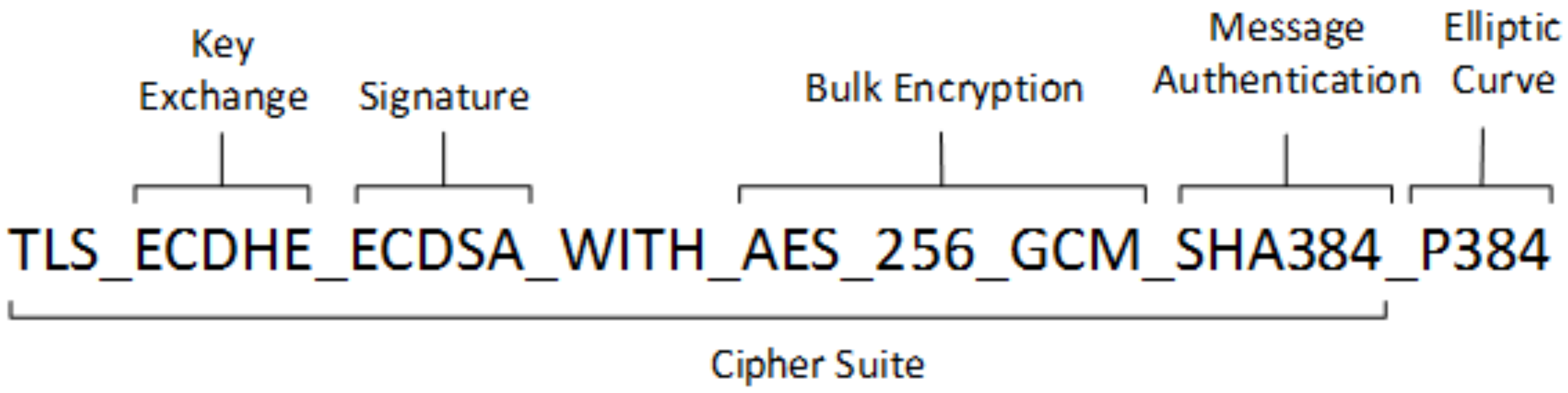
Client
Server
1 Client connects to server
3 Client verifies server's certificate
Server presents TLS certificate 2
4 Client & server exchange information over encrypted TLS connection

4. TLS

DH密钥交换



tls.Config.MinVersion = tls.VersionTLS13
保障前向安全性



4. TLS

InsecureSkipVerify可以吗?

若信任未知服务端，一旦服务端被冒充，则可以被逐步分析出应用层协议

4. TLS

InsecureSkipVerify=false

```go
if !c.config.InsecureSkipVerify {
    opts := x509.VerifyOptions{
        Roots:         c.config.RootCAs,
        CurrentTime:   c.config.time(),
        DNSName:       c.config.ServerName,
        Intermediates: x509.NewCertPool(),
    }

    for _, cert := range certs[1:] {
        opts.Intermediates.AddCert(cert)
    }
    var err error
    c.verifiedChains, err = certs[0].Verify(opts)
    if err != nil {
        c.sendAlert(alertBadCertificate)
        return err
    }
}
```

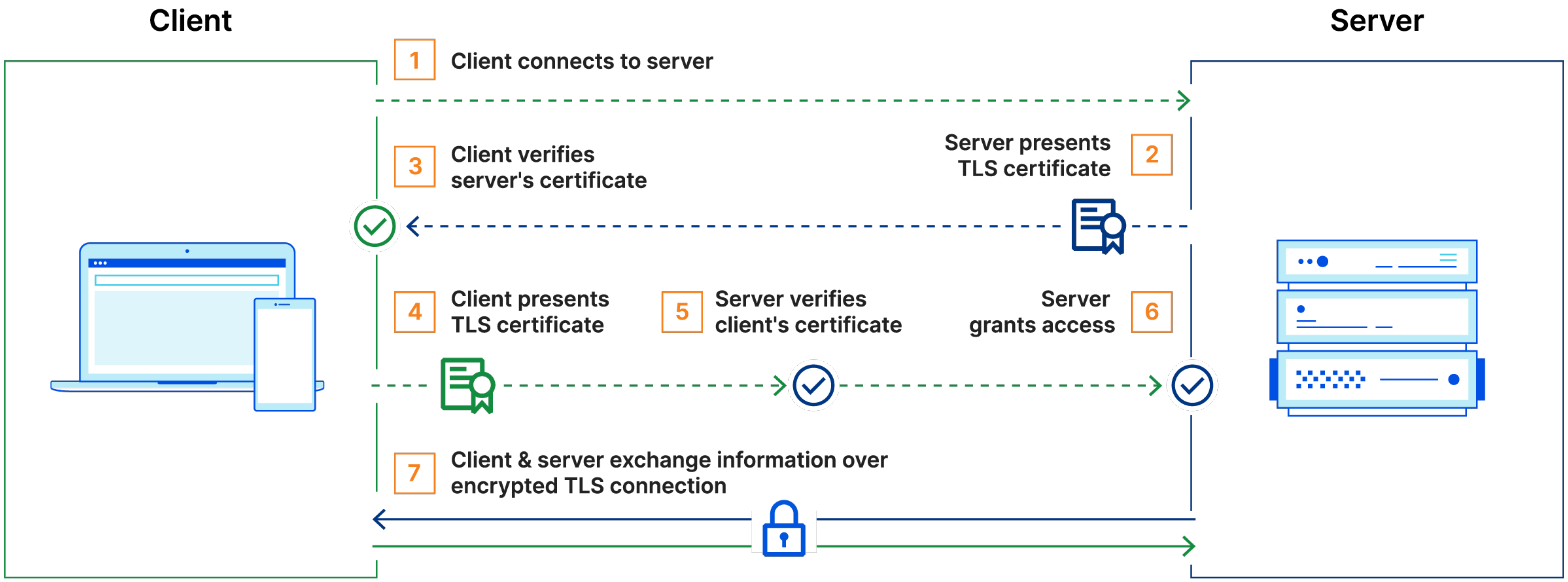
根证书

时间

SNI=hostname

4. TLS

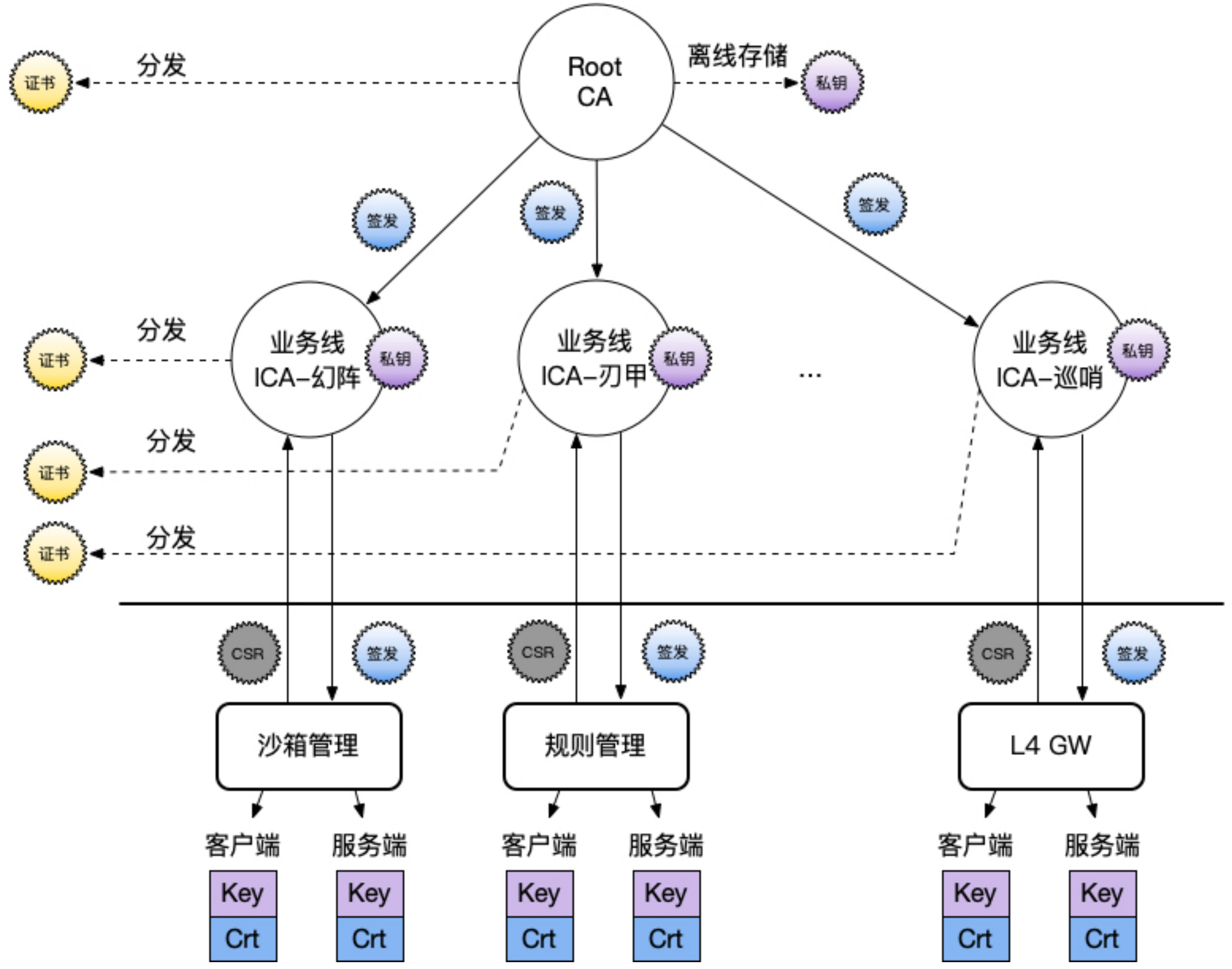
mTLS



Client
Server
1 Client connects to server
2 Server presents TLS certificate
3 Client verifies server's certificate
4 Client presents TLS certificate
5 Server verifies client's certificate
6 Server grants access
7 Client & server exchange information over encrypted TLS connection

5. 构建信任链

在线签发证书



5. 默安信任链

离线签发证书 - 例如客户环境更改ip

方法：

1. ICA私钥随工具携带在产品里
2. 携带CSR在线签发

注意：

1. 私钥在二进制内的安全防护
2. 防止工具泄漏后的任意签发

6. 基石

数论

RSA Algorithm

Key Generation

Select p, q	p and q both prime; $p \neq q$
Calculate $n = p \times q$	
Calculate $\phi(n) = (p-1)(q-1)$	
Select integer e	$\gcd(\phi(n), e) = 1;\ 1 < e < \phi(n)$
Calculate d	$de \bmod \phi(n) = 1$
Public key	$KU = \{e, n\}$
Private key	$KR = \{d, n\}$

Encryption

Plaintext:	$M < n$
Ciphertext:	$C = M^e \pmod n$

Decryption

Plaintext:	C
Ciphertext:	$M = C^d \pmod n$

已知模数n和指数e

求d需要ϕ(n)

求ϕ(n)需要大素数p, q

找p, q需要因式分解

6. 基石

RSA有效性

The number of primes smaller than x is approximately $\frac{x}{\ln x}$. Therefore the number of 512 bit primes (approximately the length you need for 1024 bit modulus) is approximately:

$$\frac{2^{513}}{\ln 2^{513}} - \frac{2^{512}}{\ln 2^{512}} \approx 2.76 \times 10^{151}$$

The number of RSA moduli (i.e. pair of two distinct primes) is therefore:

$$\frac{(2.76 \times 10^{151})^2}{2} - 2.76 \times 10^{151} = 1.88 \times 10^{302}$$

Now consider that the observable universe contains about 10^{80} atoms. Assume that you could use each of those atoms as a CPU, and each of those CPUs could enumerate one modulus per millisecond. To enumerate all 1024 bit RSA moduli you would need:

$$\begin{aligned} 1.88 \times 10^{302} ms/10^{80} &= 1.88 \times 10^{222} ms \\ &= 1.88 \times 10^{219} s \\ &= 5.22 \times 10^{215} h \\ &= 5.95 \times 10^{211} \text{years} \end{aligned}$$

Just as a comparison: the universe is about 13.75×10^9 years old.

7. 参考

https://smallstep.com/blog/everything-pki/

https://www.blackhat.com/presentations/bh-usa-99/EdGerck/certover.pdf

https://www.giac.org/paper/gsec/625/trust-model-pgp-x509-standard-pki/101441

https://en.wikipedia.org/wiki/Automatic_Certificate_Management_Environment

https://www.rfc-editor.org/rfc/rfc5280

https://zh.wikipedia.org/zh-hans/X.509

https://www.quarkay.com/code/467/speed-up-ssl-verification-on-iphone-for-let-s-encrypt-cert-by-OCSP

https://letsencrypt.org/docs/challenge-types/

https://en.wikipedia.org/wiki/DNS_Certification_Authority_Authorization

https://crypto.stackexchange.com/questions/3043/how-much-computing-resource-is-required-to-brute-force-rsa

https://en.wikipedia.org/wiki/Certificate_authority