Computer Security

INFR10067

Fall 2025

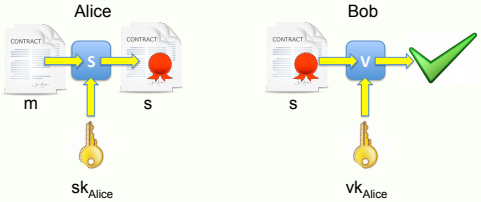
Cryptography

Digital Signatures

Myrto Arapinis and **Markulf Kohlweiss**
School of Informatics
University of Edinburgh

Goal

Data integrity and origin authenticity in the public-key setting



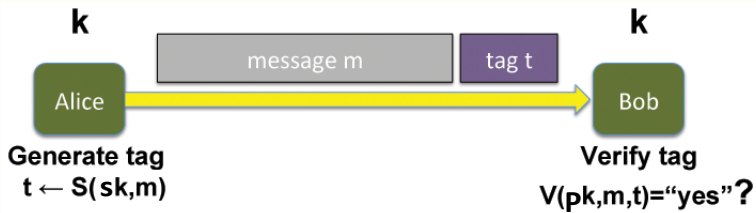
- key generation algorithm: $G : \rightarrow \mathcal{K}_{pk} \times \mathcal{K}_{sk}$
- signing algorithm $S : \mathcal{K}_{sk} \times \mathcal{M} \rightarrow \mathcal{S}$
- verification algorithm $V : \mathcal{K}_{pk} \times \mathcal{M} \times \mathcal{S} \rightarrow \{1, 0\}$
- s.t. $\forall (sk, vk) \in G$, and $\forall m \in \mathcal{M}$, $V(vk, m, S(sk, m)) = 1$

Advantages of digital signatures over MACs

MACs

- are not publicly verifiable (and so not transferable)
  No one else, except Bob, can verify t.
- do not provide non-repudiation
  t is not bound to Alice's identity only. Alice could later claim she didn't compute t herself. It could very well have been Bob since he also knows the key k.

Advantages of digital signatures over MACs



Digital signatures

- are publicly verifiable - anyone can verify a signature
- are tansferable - due to public verifiability
- provide non-repudiation - if Alice signs a document with sk, she cannot deny it later
  Alice is responsible for keeping her secret key sk secret.

Security

A good digital signature schemes should satisfy existential unforgeability.

Existential unforgeability

- Given $(m_1, S(sk, m_1))$, ..., $(m_n, S(sk, m_n))$ (where m_1, ..., m_n chosen by the adversary)
- It should be hard to compute a valid pair $(m, S(sk, m))$ without knowing sk for any $m \notin \{m_1, \dots, m_n\}$

Textbook RSA signatures

- $G_{RSA}() = (pk, sk)$ where $pk = (N, e)$ and $sk = (N, d)$ and $N = p \cdot q$ with p, q random primes and $e, d \in \mathbb{Z}$ st. $e \cdot d \equiv 1 \pmod{\phi(N)}$
- $\mathcal{M} = \mathcal{C} = \mathbb{Z}_N$
- Signing: $S_{RSA}(sk, x) = (x, x^d \pmod N)$ where $pk = (N, e)$
- Verifying: $V_{RSA}(pk, m, x) = \begin{cases} 1 & \text{if } m = x^e \pmod N \\ 0 & \text{otherwise} \end{cases}$
  where $sk = (N, d)$
- st $\forall (pk, sk) = G_{RSA}(), \forall x, V_{RSA}(pk, x, S_{RSA}(sk, x)) = 1$
  Proof: exactly as proof of consistency of RSA encryption/decryption

Problems with “textbook RSA signatures”

Textbook RSA signatures are not secure

The “textbook RSA signature” scheme does not provide existential unforgeability

- Suppose Eve has two valid signatures $\sigma_1 = M_1^d \mod N$ and $\sigma_2 = M_2^d \mod N$ from Bob, on messages M_1 and M_2.
- Then Eve can exploit the homomorphic properties of RSA and produce a new signature

$$\sigma = \sigma_1 \cdot \sigma_2 \mod N = M_1^d \cdot M_2^d \mod N = (M_1 \cdot M_2)^d \mod N$$

which is a valid signature from Bob on message $M_1 \cdot M_2$.

How to use RSA for signatures

Solution

Before computing the RSA function, apply a hash function H.

- Signing: $S_{RSA}(sk, x) = (x, H(x)^d \pmod N)$
- Verifying: $V_{RSA}(pk, m, x) = \begin{cases} 1 & \text{if } H(m) = x^e \pmod N \\ 0 & \text{otherwise} \end{cases}$

Computer Security

INFR10067

Fall 2025

Secure communications

Public Key Infrastructure

Myrto Arapinis and **Markulf Kohlweiss**
School of Informatics
University of Edinburgh

Public keys

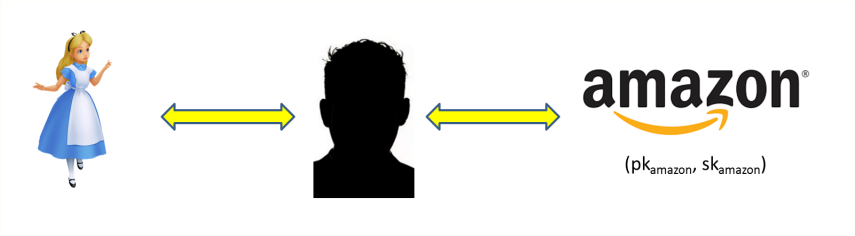


Figure: How does Alice trust that pk_{Amazon} is Amazon's public key?

Public-key encryption schemes are secure only if the authenticity of the public key is assured

Distribution of public keys

1. Public announcements - participants broadcast their public key
   ☹ does not defend against fake certificates
2. Publicly available directories - participants publish their public key on public directories
   ☹ does not defend against fake certificates
3. Public-key authority - participants contact the authority for each public key it needs
   ☹ bottleneck in the system
4. public-key certificates - CAs issue certificates to participants on their public key
   ☺ as reliable as public-key authority but avoiding the bottleneck

Critical step: Certificate validation

- **The crux of TLS security**: Server authenticates itself to client
- **What is a certificate?**
  - A public key + signature of that key
    - Encoded in an insanely asinine byte format
  - Associated information (domain name, organization, etc.)
  - Essentially says: “I am google.com, and my public key is [key]”
- **Certificate validation process**:
  1. Browser receives certificate from server
  2. Verifies the certificate’s signature
  3. If signature is valid →certificate and public key are trusted
  4. Browser proceeds with connection

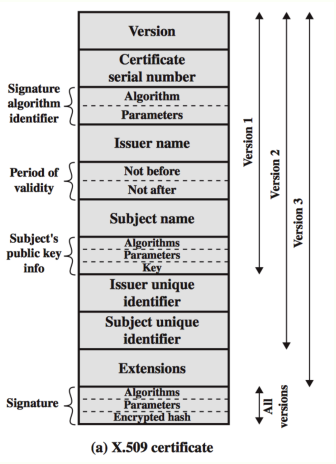


Figure: Image from Cryptography and Network Security - Principles and Practice - William Stallings

Certificate Authorities: The trust foundation

- **Problem**: How does the browser know which public key to use for verification?
- **Solution**: Certificate Authorities (CAs)
  - Public keys hardcoded in your browser/OS.
  - Trusted organizations that issue certificates.
  - Act as trusted third parties.
- **CA's role**:
  - Verify that public keys belong to claimed entities.
  - Sign certificates with their private keys.
  - Protect their private keys from compromise.
- **Without CAs**: No way to distinguish legitimate servers from MITM attackers!

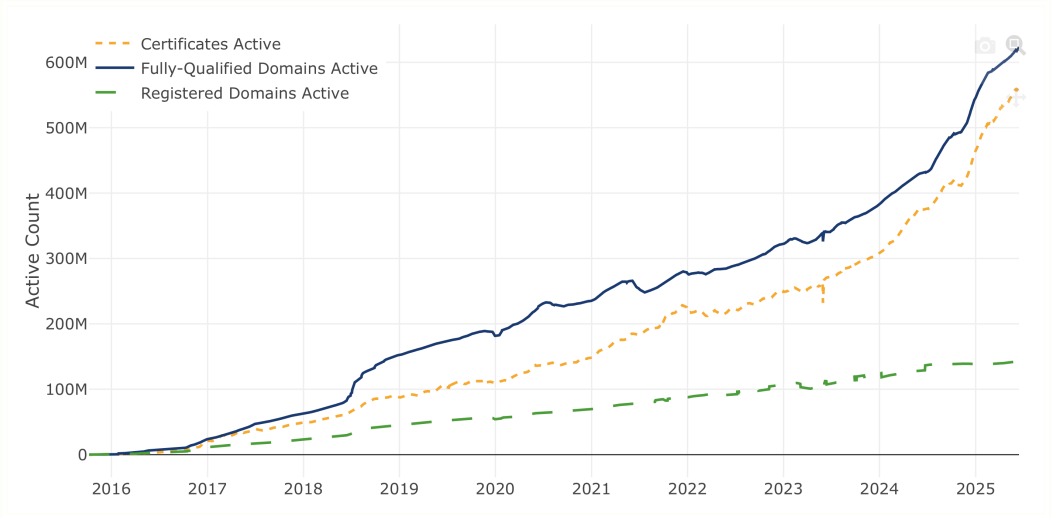
Traditional CA validation nightmare

- **Getting a certificate before 2015**:
  - Contact a Certificate Authority (Verisign, Thawte, etc.).
  - Pay $50-300+ per certificate per year.
    - Completely arbitrary extortionate amounts.
  - Manual validation process takes days/weeks.
- **Domain Validation (DV) process**:
  1. Submit CSR (Certificate Signing Request).
  2. CA sends email to `admin@domain.com`.
  3. Click verification link in email.
  4. Wait for manual review and approval.
  5. Download and install certificate.
- **Extended Validation (EV) certificates**:
  - Even more expensive ($150-1000+/year).
  - Weeks of back-and-forth communication.

Let’s Encrypt: revolution in TLS certificates (2015)

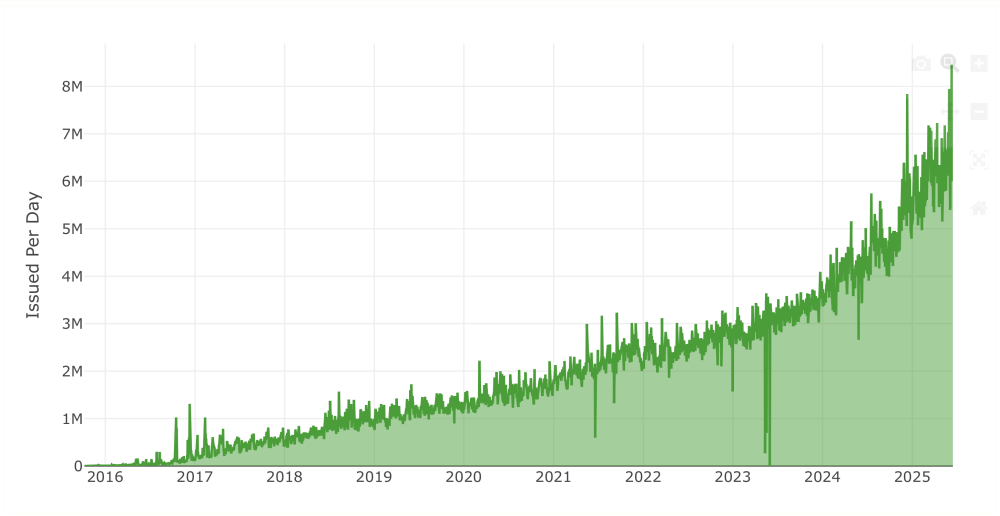
- **Game-changing**:
  - **Free** certificates for everyone.
  - **Automated** issuance and renewal.
  - **90-day** certificate lifetime (encourages automation).
  - Open source, non-profit initiative.
- **ACME protocol** (Automated Certificate Management Environment):
  - Domain validation via HTTP or DNS challenges.
  - Prove control of domain programmatically.
  - Certificate issued in seconds, not days.
  - Automatic renewal before expiration.
- **Impact on the web**:
  - HTTPS adoption jumped from 40% to 90%+ of web traffic.
  - Eliminated cost barrier for small websites.
  - Made “HTTPS everywhere” a reality.
- **Philosophy**: Encryption should be the default, not a luxury.

Let’s Encrypt: active certificates



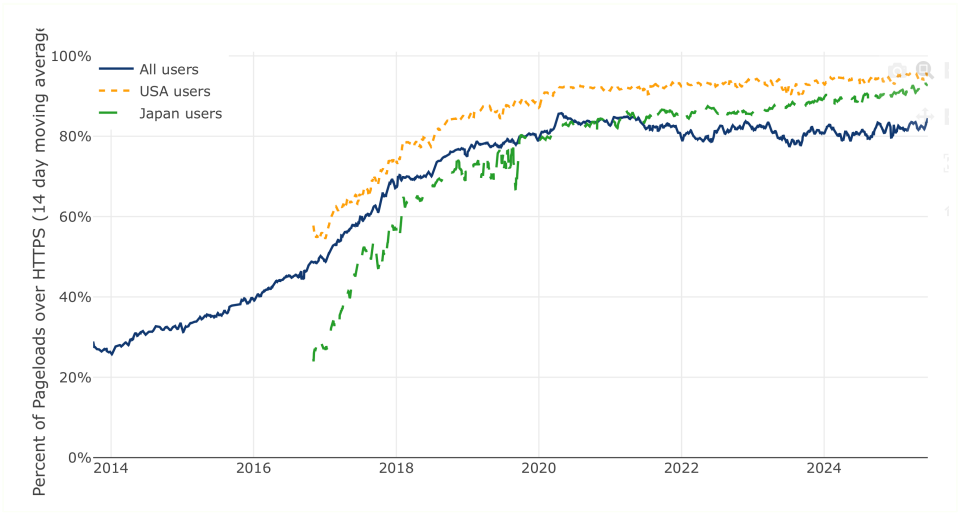
Source: Let’s Encrypt

Let’s Encrypt: certificates issued



Source: Let’s Encrypt

Percentage of HTTPS traffic in Firefox



Source: Let's Encrypt

Certificate chains in practice

- **Real-world example**: Connecting to `www.google.com`
- **Certificate chain structure**:
  - **Certificate 0**: `www.google.com`'s certificate.
  - **Certificate 1**: Intermediate CA (Google Trust Services).
  - **Certificate 2**: Root CA (GlobalSign).
- **Verification process**:
  1. Check Google's signature on Certificate 0.
  2. Check GlobalSign's signature on Certificate 1.
  3. Trust GlobalSign's root certificate (pre-installed).
- **Chain of trust**: Each certificate vouches for the next.

Exploring certificates with OpenSSL

- **Connect and view certificate**:
  - `openssl s_client -connect www.google.com:443`
  - Shows certificate chain and raw certificate data
- **Parse certificate details**:
  - `openssl x509 -text -noout`
  - Reveals subject, issuer, validity dates, algorithms
- **Certificate markers**:
  - `s:` = subject (who the certificate is for)
  - `i:` = issuer (who signed the certificate)
- Try this at home! Great way to understand the certificate ecosystem.

```
openssl s_client -connect www.google.com:443 -servername
www.google.com | openssl x509 -text -noout
```

Public key certificates

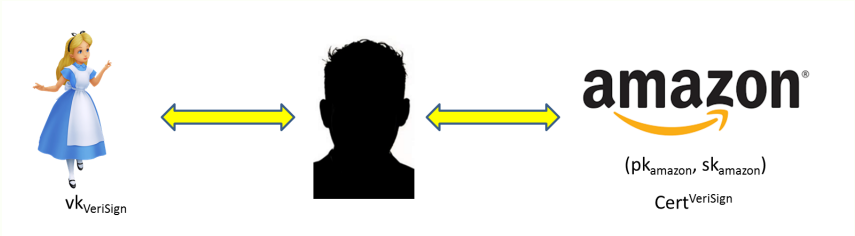
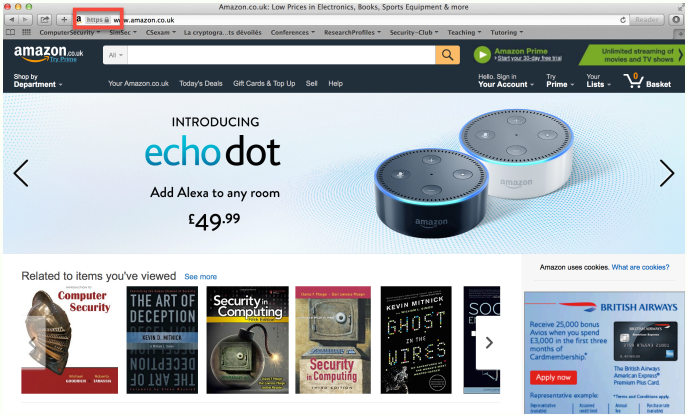


Figure: Alice can now verify Amazon's certificate

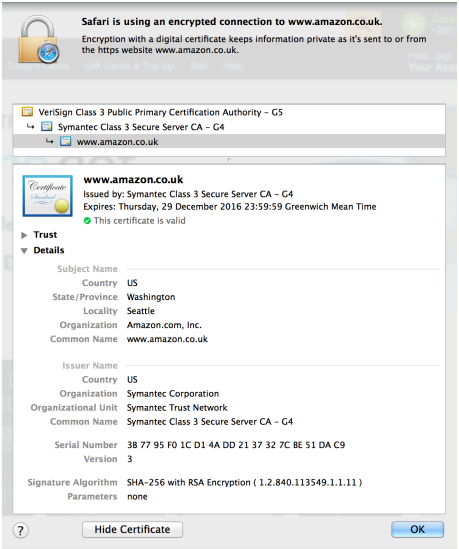
Using public key certificates to secure the Internet



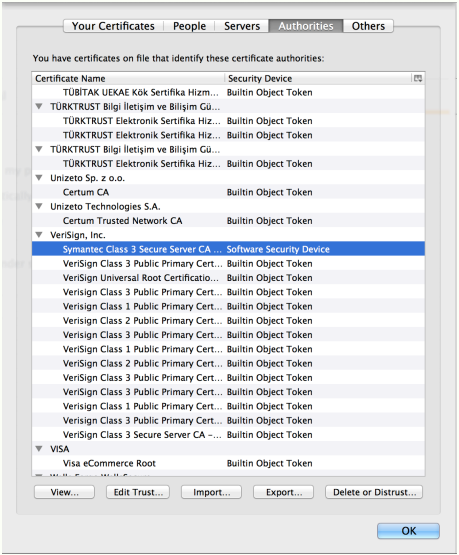
A very important implicit assumption

The browser is trusted to be “secure”

Amazon's certificate



Browser root certificates



Revocation

- A certificate needs to be revoked if the corresponding private key has been compromised.
- Certificate Revocation Lists (CRLs) are the solution adopted in X.509.
- Online Certificate Status Protocol (OCSP) stapling is the modern solution to this problem.