The road to TLS 1.3

Tom Tervoort

Contents

Introduction to TLS

The initial problem

The initial problem

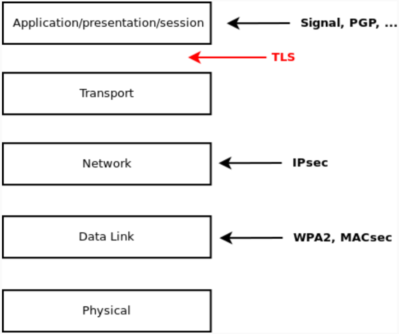
- Early 90's: advent of the web and e-commerce
- Shopping online involves transmitting credit card numbers
- ... but the internet is an untrusted network

The solution: a secure channel

- Cryptographically protect a connection between two parties
- Protect against sniffing, spoofing, tampering, replaying, reordering etc.
- Identification: prove knowledge of a secret key

- Assumption: attacker has full control over the network
  - However, the attacker has no control over the end-points
- How realistic is this?

TLS/SSL: a secure channel on 'layer $4\frac{1}{2}$'



- TLS requires a reliable transport layer protocol; i.e. TCP
- Alternative for (unicast) UDP: DTLS
- Currently in development: TLS for QUIC

TLS vs. SSL

- SSL: Secure Sockets Layer
- TLS: Transport Layer Security
- TLS is SSL's successor
- Six versions: SSL 2.0, SSL 3.0, TLS 1.0, TLS 1.1, TLS 1.2, TLS 1.3

Three sub-protocols

- Handshake protocol
    - Negotiate protocol version and ciphers
    - Establish session keys (asymmetric cryptography)
    - Authenticate server (and optionally client)
- Record protocol
    - Encrypted/authenticated communications
    - Replay/reorder protection
- Alert protocol
    - Notifications and errors
    - Plaintext or encrypted

The TLS 1.2 handshake

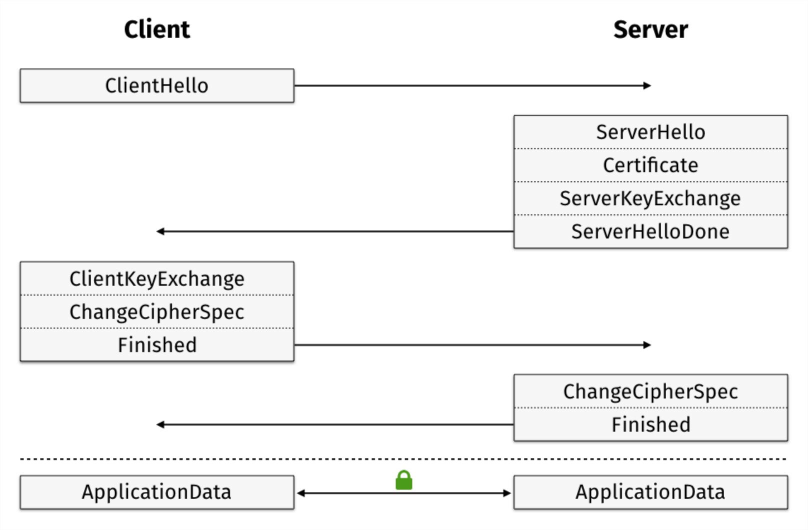


Image source: Tim Taubert

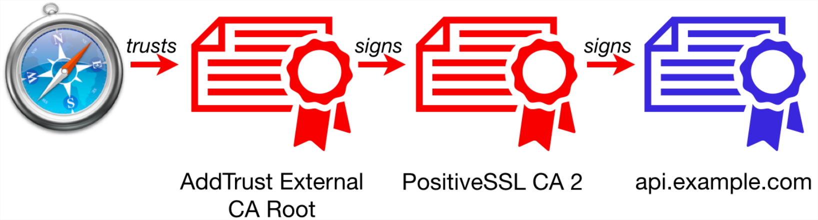
Authentication within TLS

- How does Alice know she's talking to Bob?
- Step 1: verify that Bob owns a private key matching a certain public key
    - Digitally sign part of handshake (including Alice's nonce)
    - Incorporate in key exchange protocol, then verify Alice and Bob share the same keys

Authentication within TLS

- How does Alice know about Bob's public key?
- Step 2: Bob presents a certificate, which contains:
  - His public key
  - Information about his identity (e.g. domain name, e-mail address)
  - Information about certificate validity (e.g. expiration date)
  - A signature by a certificate authority (CA)
- The CA vouches that the certificate information is accurate
  - Bob has proved the key-identity link to the CA

Authentication within TLS



- How does Alice know she can trust the CA?
- Step 3: Bob sends the CA's certificate chain:
  - Intermediate certificate: validate and move up
  - Root certificate: should be in Alice's trust store

Image source: Sasha Romijn

Which root certificates to trust?

- Opportunistic encryption: trust everything
  - No protection against man-in-the-middle attacks
- Simple approach: store single certificate in the application
- Corporate Public Key Infrastructure (PKI)
  - Admin distributes company trust store
- Browsers: worldwide PKI
  - Browsers trust about 100 CA's to each authenticate the entire internet

Problems and vulnerabilities

SSL 1.0

- Never released
- Initially did not authenticate packets, and had no replay protection
- Later: authentication with CRC checksums, easy to break

SSL 2.0

- Released by Netscape in 1995
- Handshake messages not protected
    - Man-in-the-middle can force both parties to use weaker cryptography
- The redesigned SSL 3.0 was released one year later

SSL 3.0: Padding oracle attack

- Open question during SSL design: encrypt-then-authenticate, or the other way around?
- Chose authenticate-then-encrypt
- When decryption fails: `decryption_failed` error
- When decryption succeeds but integrity check fails: `bad_record_mac` error

- Vaudenay in 2002: type of error reveals information about plaintext
- Attacker can exploit this to completely decrypt communications

TLS 1.0: BEAST

- TLS 1.0 did not implement CBC encryption ‘by the book’
- Bard in 2006: published “a challenging but feasible” attack
- Did not seem exploitable in HTTPS

- Duong and Rizzo in 2011: BEAST attack
- Stole cookies with a Java applet
- Lot of publicity
- TLS vulnerabilities became a popular research subject

CRIME

- By design: TLS does not hide message length
- Useful feature: TLS compression

- How well a message compresses reveals information about its content
- Theoretical vulnerability: compression-oracle attack
- BEAST researchers in 2012: not-so-theoretical attack against HTTPS
- Attackers can steal cookies

Artefacts of the Crypto Wars

- Until 1996: U.S. export controls on cryptography
- Maximum key size for exported products: 40 bits
  - Breakable by U.S. government in the 90's
  - And now, by anyone else
- Netscape: "U.S. edition" and "International edition"
  - Only 40-bit ciphers in international edition
  - American servers: can support both strong and export crypto
  - Important reason for cipher negotiation mechanism
- Export cryptography stuck around for compatibility reasons

Downgrade attacks

- Intention: client and server pick strongest mutually supported protocol and cipher
- In practice: attacker could intervene
- POODLE attack: downgrade to SSL 3.0, enabling padding oracle attack
- DROWN, FREAK and LOGJAM: exploit export crypto to achieve man-in-the-middle attack
    - DROWN and LOGJAM work even if the client does not support export ciphers

Weak cryptography: “it’s only theoretical”

- Five original options for symmetric encryption
- DES
  - Diffie and Hellman in 1977: DES cipher is insecure
  - 1997 DESCHALL Project: publicly cracked; complete key recovery
- RC4
  - Roos in 1995: statistical biases in keystream
  - Vanhoef and Piessens in 2015: decrypt an HTTP cookie in 75 hours
- 3DES, IDEA, RC2
  - As used in TLS: not suitable for encrypting large amounts of data
  - Bhargavan and Leurent in 2016: HTTP cookie decrypted in two days (Sweet32)

Weak cryptography: “it’s only theoretical”

- Bleichenbacher in 1998: attack on RSA encryption with PKCS#1 padding
- Still used by TLS 1.2, with special mitigations
- ROBOT attack in 2017: mitigations insufficient

More complexity, more bugs

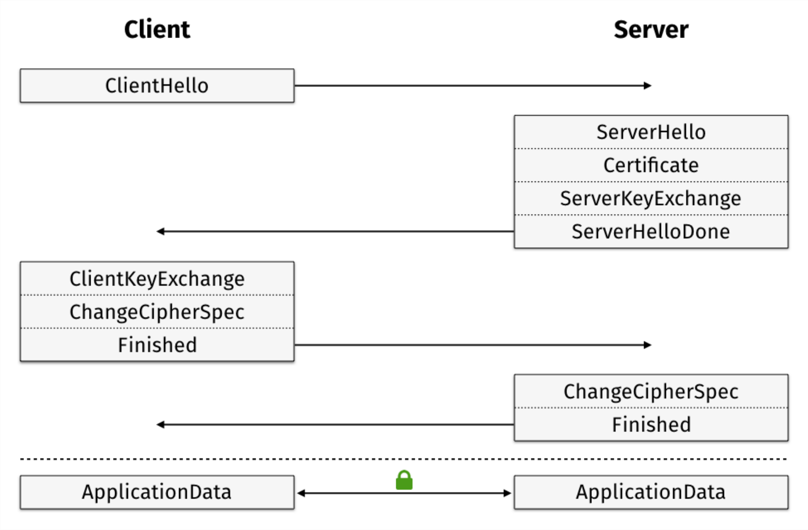
More complexity, more bugs

```
static OSStatus
SSLVerifySignedServerKeyExchange(SSLContext *ctx, bool isRsa, SSLBuffer signedParams,
                                 uint8_t *signature, UInt16 signatureLen)
{
    OSStatus        err;
    SSLBuffer       hashOut, hashCtx, clientRandom, serverRandom;
    uint8_t         hashes[SSL_SHA1_DIGEST_LEN + SSL_MD5_DIGEST_LEN];
    SSLBuffer       signedHashes;
    uint8_t         *dataToSign;
    size_t          dataToSignLen;

    ...
    if ((err = ReadyHash(&SSLHashSHA1, &hashCtx)) != 0)
        goto ↓fail;
    if ((err = SSLHashSHA1.update(&hashCtx, &clientRandom)) != 0)
        goto ↓fail;
    if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
        goto ↓fail;
    if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
        goto ↓fail;
        goto ↓fail;
    if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
        goto ↓fail;

    err = sslRawVerify(ctx,
                       ctx->peerPubKey,
                       dataToSign,              /* plaintext */
                       dataToSignLen,           /* plaintext length */
                       signature,
                       signatureLen);
    if(err) {
        sslErrorLog("SSLDecodeSignedServerKeyExchange: sslRawVerify "
                    "returned %d\n", (int)err);
        goto ↓fail;
    }
```

Security vs. performance



- Full key exchange: 1.5/2 round trips (not counting TCP handshake)
- 1-RTT shortcuts possible, but sacrifice forward secrecy

Design of TLS 1.3

Major improvements

- Remove (potentially dangerous) legacy features
- Simplify the protocol
- Encrypt more
- Reduce handshake round-trips

Less is more: features

- Unnecessary/dangerous features to be scrapped:
  - Key exchange methods without forward secrecy
  - Renegotiation mechanism
  - Custom Diffie-Hellman parameters
  - Compression

Less is more: ciphers

```
TLS_DH_anon_EXPORT_WITH_DES40_CBC_SHA
TLS_DH_anon_EXPORT_WITH_RC4_40_MD5
TLS_DH_anon_WITH_3DES_EDE_CBC_SHA
TLS_DH_anon_WITH_AES_128_CBC_SHA
TLS_DH_anon_WITH_AES_128_CBC_SHA256
TLS_DH_anon_WITH_AES_128_GCM_SHA256
TLS_DH_anon_WITH_AES_256_CBC_SHA
TLS_DH_anon_WITH_AES_256_CBC_SHA256
TLS_DH_anon_WITH_AES_256_GCM_SHA384
TLS_DH_anon_WITH_CAMELLIA_128_CBC_SHA
TLS_DH_anon_WITH_CAMELLIA_256_CBC_SHA
TLS_DH_anon_WITH_DES_CBC_SHA
TLS_DH_anon_WITH_RC4_128_MD5
TLS_DH_anon_WITH_SEED_CBC_SHA
TLS_DH_DSS_EXPORT_WITH_DES40_CBC_SHA
TLS_DH_DSS_WITH_3DES_EDE_CBC_SHA
TLS_DH_DSS_WITH_AES_128_CBC_SHA
TLS_DH_DSS_WITH_AES_128_CBC_SHA256
TLS_DH_DSS_WITH_AES_128_GCM_SHA256
TLS_DH_DSS_WITH_AES_256_CBC_SHA
TLS_DH_DSS_WITH_AES_256_CBC_SHA256
TLS_DH_DSS_WITH_AES_256_GCM_SHA384
TLS_DH_DSS_WITH_CAMELLIA_128_CBC_SHA
TLS_DH_DSS_WITH_CAMELLIA_256_CBC_SHA
TLS_DH_DSS_WITH_DES_CBC_SHA
TLS_DH_DSS_WITH_SEED_CBC_SHA
TLS_DH_RSA_EXPORT_WITH_DES40_CBC_SHA
TLS_DH_RSA_WITH_3DES_EDE_CBC_SHA
TLS_DH_RSA_WITH_AES_128_CBC_SHA
TLS_DH_RSA_WITH_AES_128_CBC_SHA256
TLS_DH_RSA_WITH_AES_128_GCM_SHA256
TLS_DH_RSA_WITH_AES_256_CBC_SHA
TLS_DH_RSA_WITH_AES_256_CBC_SHA256
TLS_DH_RSA_WITH_AES_256_GCM_SHA384
TLS_DH_RSA_WITH_CAMELLIA_128_CBC_SHA
TLS_DH_RSA_WITH_CAMELLIA_256_CBC_SHA
TLS_DH_RSA_WITH_DES_CBC_SHA
TLS_DH_RSA_WITH_SEED_CBC_SHA
TLS_DHE_DSS_EXPORT1024_WITH_DES_CBC_SHA
TLS_DHE_DSS_EXPORT1024_WITH_RC4_56_SHA
TLS_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA
TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA
TLS_DHE_DSS_WITH_AES_128_CBC_SHA
TLS_DHE_DSS_WITH_AES_128_CBC_SHA256
TLS_DHE_DSS_WITH_AES_128_GCM_SHA256
TLS_DHE_DSS_WITH_AES_256_CBC_SHA
TLS_DHE_DSS_WITH_AES_256_CBC_SHA256
TLS_DHE_DSS_WITH_AES_256_GCM_SHA384
TLS_DHE_DSS_WITH_CAMELLIA_128_CBC_SHA
TLS_DHE_DSS_WITH_CAMELLIA_256_CBC_SHA
TLS_DHE_DSS_WITH_DES_CBC_SHA
TLS_DHE_DSS_WITH_RC4_128_SHA
TLS_DHE_DSS_WITH_SEED_CBC_SHA
TLS_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA
TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA
TLS_DHE_RSA_WITH_AES_128_CBC_SHA
TLS_DHE_RSA_WITH_AES_128_CBC_SHA256
TLS_DHE_RSA_WITH_AES_128_GCM_SHA256
TLS_DHE_RSA_WITH_AES_256_CBC_SHA
TLS_DHE_RSA_WITH_AES_256_CBC_SHA256
TLS_DHE_RSA_WITH_AES_256_GCM_SHA384
TLS_DHE_RSA_WITH_CAMELLIA_128_CBC_SHA
TLS_DHE_RSA_WITH_CAMELLIA_256_CBC_SHA
TLS_DHE_RSA_WITH_DES_CBC_SHA
TLS_DHE_RSA_WITH_SEED_CBC_SHA
TLS_ECDH_anon_WITH_3DES_EDE_CBC_SHA
TLS_ECDH_anon_WITH_AES_128_CBC_SHA
TLS_ECDH_anon_WITH_AES_256_CBC_SHA
TLS_ECDH_anon_WITH_NULL_SHA
TLS_ECDH_anon_WITH_RC4_128_SHA
TLS_ECDH_ECDSA_WITH_3DES_EDE_CBC_SHA
TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA
TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA256
TLS_ECDH_ECDSA_WITH_AES_128_GCM_SHA256
TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA
TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA384
TLS_ECDH_ECDSA_WITH_AES_256_GCM_SHA384
TLS_ECDH_ECDSA_WITH_CAMELLIA_128_CBC_SHA256
TLS_ECDH_ECDSA_WITH_CAMELLIA_256_CBC_SHA384
TLS_ECDH_ECDSA_WITH_NULL_SHA
TLS_ECDH_ECDSA_WITH_RC4_128_SHA
TLS_ECDH_RSA_WITH_3DES_EDE_CBC_SHA
TLS_ECDH_RSA_WITH_AES_128_CBC_SHA
TLS_ECDH_RSA_WITH_AES_128_CBC_SHA256
TLS_ECDH_RSA_WITH_AES_128_GCM_SHA256
TLS_ECDH_RSA_WITH_AES_256_CBC_SHA
TLS_ECDH_RSA_WITH_AES_256_CBC_SHA384
TLS_ECDH_RSA_WITH_AES_256_GCM_SHA384
TLS_ECDH_RSA_WITH_CAMELLIA_128_CBC_SHA256
TLS_ECDH_RSA_WITH_CAMELLIA_256_CBC_SHA384
TLS_ECDH_RSA_WITH_NULL_SHA
TLS_ECDH_RSA_WITH_RC4_128_SHA
TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384
TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
TLS_ECDHE_ECDSA_WITH_CAMELLIA_128_CBC_SHA25
TLS_ECDHE_ECDSA_WITH_CAMELLIA_256_CBC_SHA38
TLS_ECDHE_ECDSA_WITH_NULL_SHA
TLS_ECDHE_ECDSA_WITH_RC4_128_SHA
TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
TLS_ECDHE_RSA_WITH_CAMELLIA_128_CBC_SHA256
TLS_ECDHE_RSA_WITH_CAMELLIA_256_CBC_SHA384
TLS_ECDHE_RSA_WITH_NULL_SHA
TLS_ECDHE_RSA_WITH_RC4_128_SHA
TLS_GOSTR341001_WITH_28147_CNT_IMIT
TLS_GOSTR341001_WITH_NULL_GOSTR3411
TLS_GOSTR341094_WITH_28147_CNT_IMIT
TLS_GOSTR341094_WITH_NULL_GOSTR3411
TLS_PSK_WITH_3DES_EDE_CBC_SHA
TLS_PSK_WITH_AES_128_CBC_SHA
TLS_PSK_WITH_AES_256_CBC_SHA
TLS_PSK_WITH_RC4_128_SHA
TLS_RSA_EXPORT1024_WITH_DES_CBC_SHA
TLS_RSA_EXPORT1024_WITH_RC4_56_SHA
TLS_RSA_EXPORT_WITH_DES40_CBC_SHA
TLS_RSA_EXPORT_WITH_RC2_CBC_40_MD5
TLS_RSA_EXPORT_WITH_RC4_40_MD5
TLS_RSA_WITH_3DES_EDE_CBC_SHA
TLS_RSA_WITH_AES_128_CBC_SHA
TLS_RSA_WITH_AES_128_CBC_SHA256
TLS_RSA_WITH_AES_128_GCM_SHA256
TLS_RSA_WITH_AES_256_CBC_SHA
TLS_RSA_WITH_AES_256_CBC_SHA256
TLS_RSA_WITH_AES_256_GCM_SHA384
TLS_RSA_WITH_CAMELLIA_128_CBC_SHA
TLS_RSA_WITH_CAMELLIA_256_CBC_SHA
TLS_RSA_WITH_DES_CBC_SHA
TLS_RSA_WITH_IDEA_CBC_SHA
TLS_RSA_WITH_NULL_MD5
TLS_RSA_WITH_NULL_SHA
TLS_RSA_WITH_NULL_SHA256
TLS_RSA_WITH_RC4_128_MD5
TLS_RSA_WITH_RC4_128_SHA
TLS_RSA_WITH_SEED_CBC_SHA
```

Less is more: ciphers

```
TLS_AES_128_GCM_SHA256
TLS_AES_256_GCM_SHA384
TLS_CHACHA20_POLY1305_SHA256
TLS_AES_128_CCM_SHA256
TLS_AES_128_CCM_8_SHA256
```

Encrypting the handshake

- All handshake messages after `ServerHello` are encrypted
- This includes extension info and server/client certificates

The new handshake

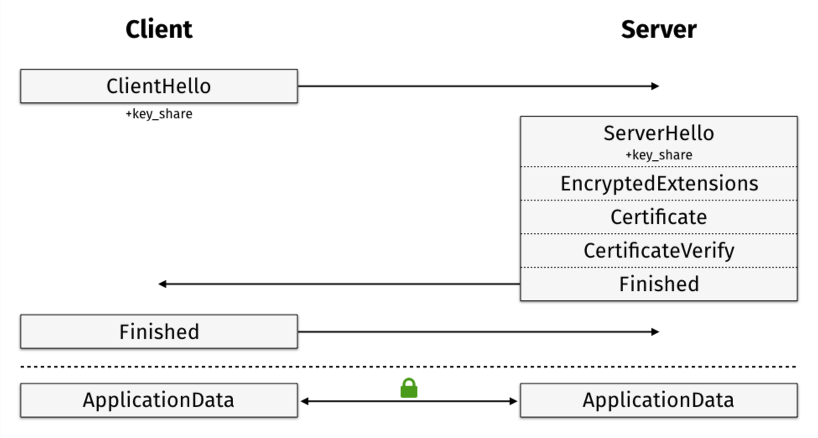
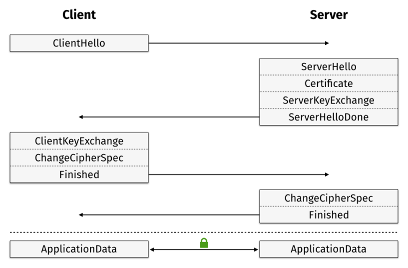


Image source: Tim Taubert

Handshake comparison



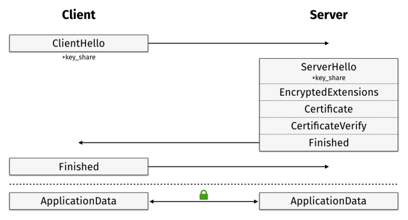


Image source: Tim Taubert

0-RTT handshake

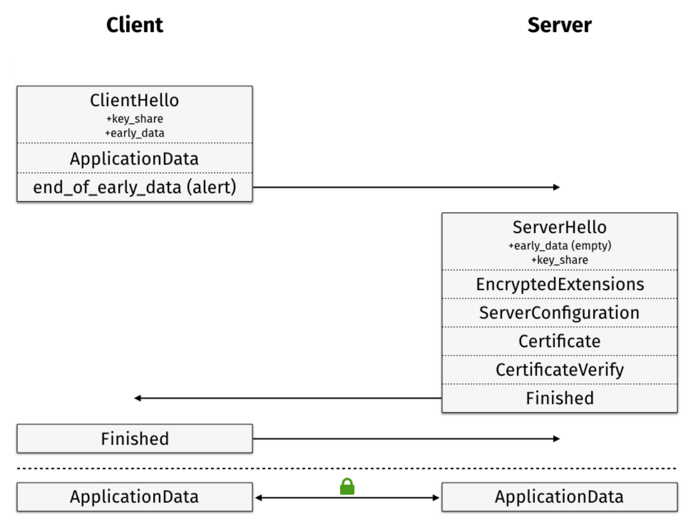


Image source: Tim Taubert

Remaining issues

0-RTT handshake: replay attacks

- `ClientHello` messages with *early data* can be replicated by an attacker
- If server accepts multiple messages encrypted with the same PSK, a replay attack is possible

- Typical usage: GET requests
- GET's shouldn't change state, but sometimes they do
  - New type of application bug

- Imperfect mitigations: timestamping, storing recent `ClientHello`'s

Quantum computers

- Shor's quantum algorithm: breaks RSA, Diffie-Hellman, ECC
- I.e. all asymmetric cryptography employed by TLS

- Solution: post-quantum cryptography
  - Experimental cipher suites
  - NIST's PQC competition: currently in round 1; standards expected around 2022-2024

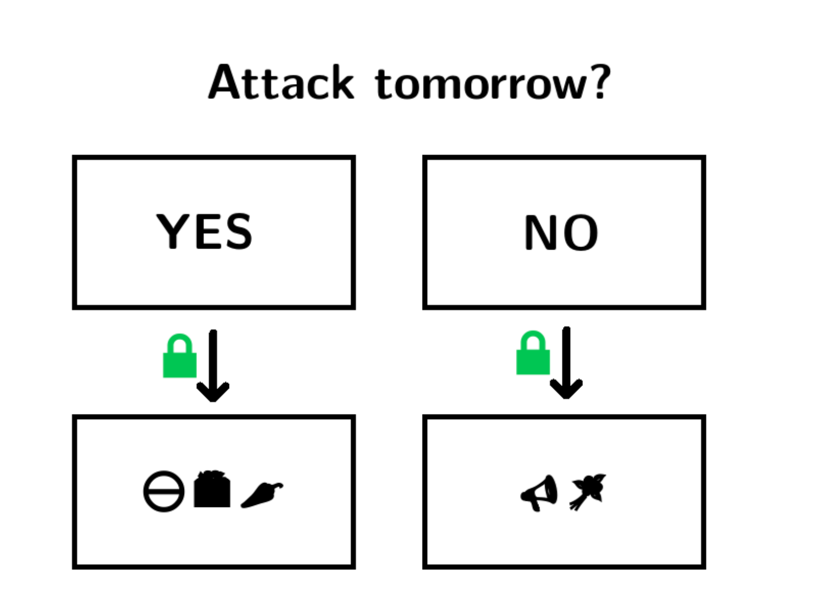
The global PKI

Image source: iotgadgets.com

The global PKI

- Compromise one root CA, MitM the whole web
- Proposed alternatives did not catch on, so far:
  - DANE: DNS(SEC)-based trust model
  - Convergence: more than one CA vouches for site authenticity
  - HPKP: key pinning in the browser
- Certificate Transparency project
  - Detection (but no prevention) of wrongly issued certificates
  - Caught Symantec issuing an unauthorized `google.com` certificate

Traffic analysis



Interoperability problems

- Upgrading an omnipresent internet protocol is hard
- Servers didn't implement TLS version negotiation correctly
- Middleboxes make assumptions based on previous protocol versions
- Upgrading is a little easier on the web:
  - Most people's browsers update automatically
  - Browser vendors can propel/force protocol adoption and deprecation
- E-mail, on the other hand...

Conclusion

- Lessons learned from SSL/TLS vulnerabilities
- TLS 1.3: simpler, faster and safer
- But: the problem of transport security has not been ‘solved’