Expert Transport Layer Security

Welcome

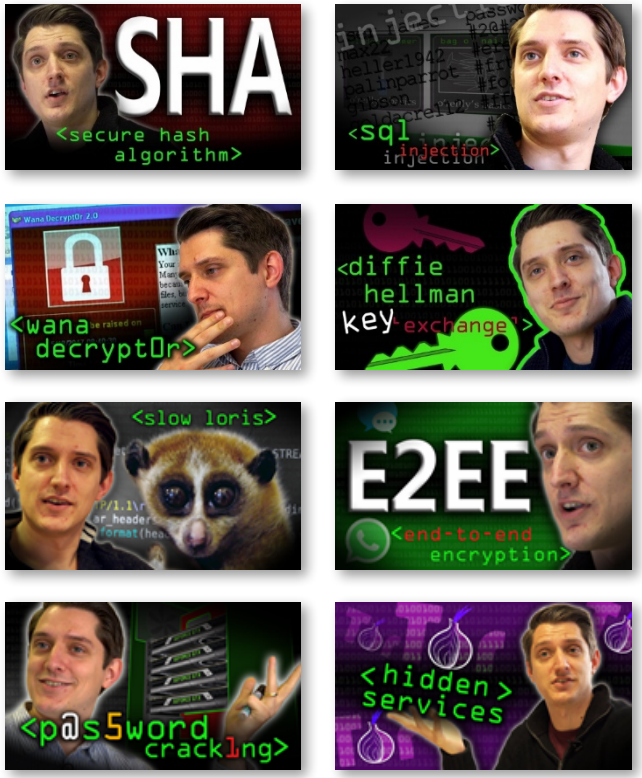
Michael Pound

- Assistant Professor at the University of Nottingham, UK
- Teach the final year Security module on our degrees

Email: michael.pound@nottingham.ac.uk

Twitter: @_mikepound

Videos: YouTube.com/computerphile

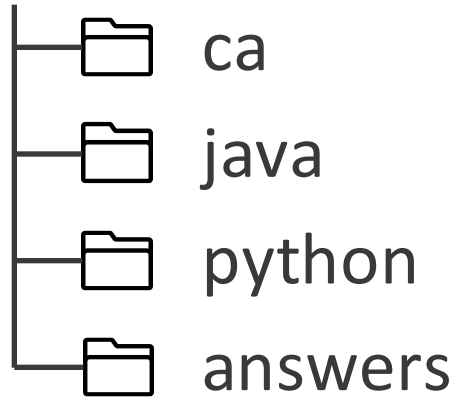


About This Course

- A detailed look at arguably the world's most important protocol – Transport Layer Security
- Not focused on HTTP: inter-application communication
- A look into Public Key Infrastructure
- Certificate pinning and mutual authentication
- Exercises in establishing and configuring secure sessions
- Knowledge of good practice for TLS and PKI

The Exercises

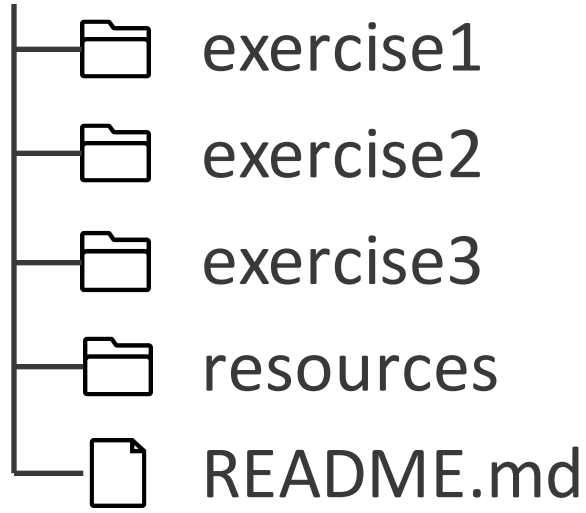
- Exercises in either Java or Python



- Don’t worry if you don’t finish them in the time, and by all means do more after the course!

https://github.com/mikepound/tls-exercises

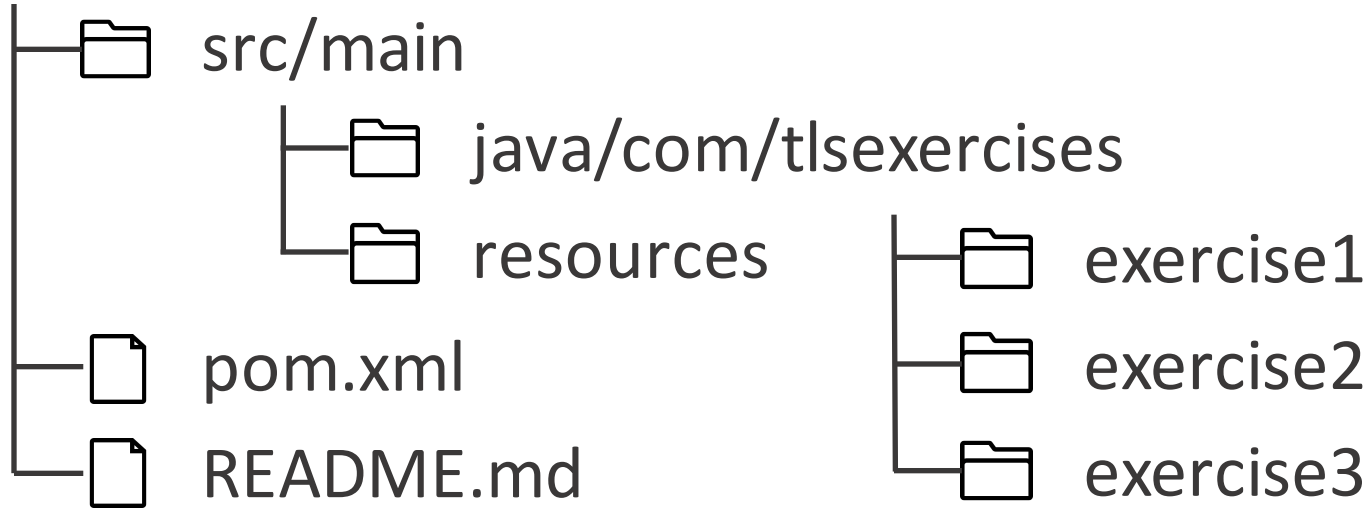
Python Exercises



- You can run from the command line, but the I recommend PyCharm
- Uses the standard *ssl* library
- Tested on Python 3.6
  - If you need Python 2 support, consider *pyOpenSSL* not *ssl*

https://github.com/mikepound/tls-exercises/tree/master/python

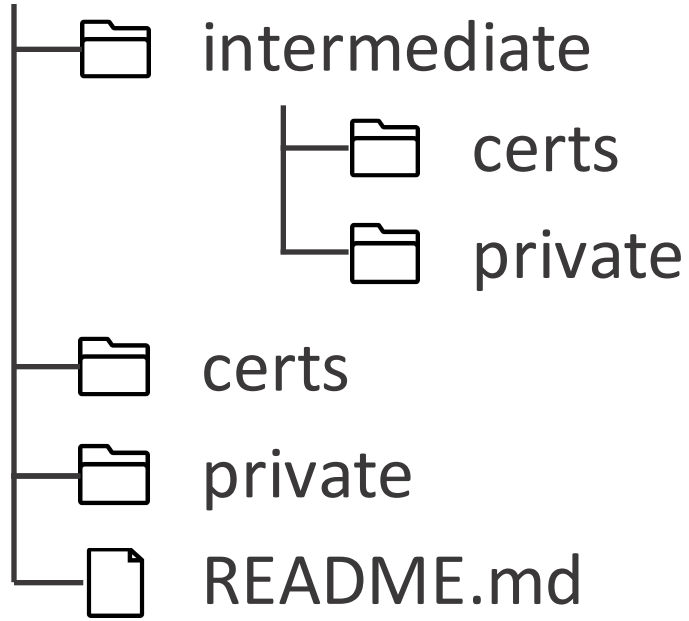
Java Exercises



- Standard java package structure including a maven pom
- Works well in IntelliJ
- Uses the SunJSSE implementation
- Tested on Java 8

https://github.com/mikepound/tls-exercises/tree/master/java

Certification Authority



- Example OpenSSL CA – Uses OpenSSL v1.1.1b
- Was also used to generate all certificates in use in the exercises

https://github.com/mikepound/tls-exercises/tree/master/ca

Glossary

- Glossary of common definitions from Cryptography, TLS and PKI
  - If anything is missing – please let me know!

https://github.com/mikepound/tls-exercises/blob/master/glossary.pdf

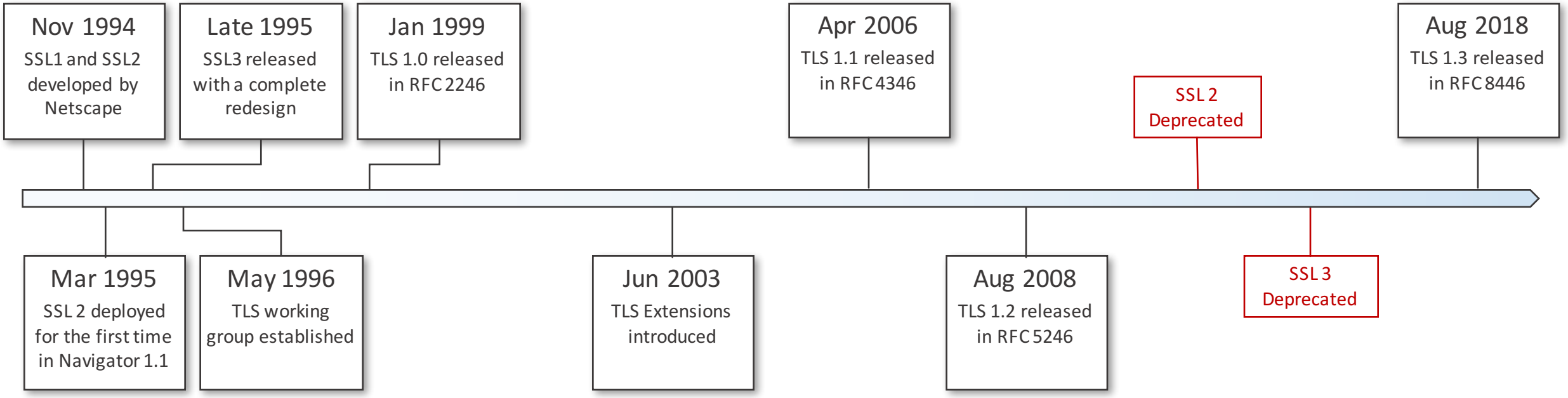
TLS Introduction

What is TLS?

- A cryptographic protocol for secure internet communication
- An agreed message and handshake structure for parties to:
  - Agree cryptographic parameters
  - Establish secret keys
  - Authenticate identities
  - Transmit messages

- Is an IETF proposed standard, currently at v1.3
- Used for HTTPS, but also many other systems using end-to-end encryption

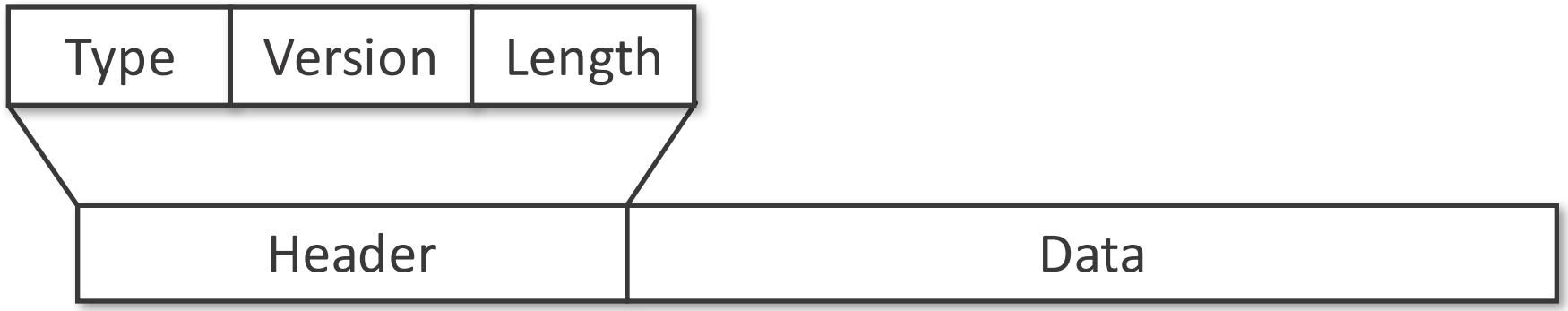
A brief history of SSL/TLS



The TLS Protocol

The Record Protocol

- All TLS packets are sent using the record protocol



Type	Subprotocol of {20, 21, 22, 23}
Version	Major and minor TLS version e.g. 03 01 for TLS 1.0
Length	The length of the data (max 16,384 bytes)

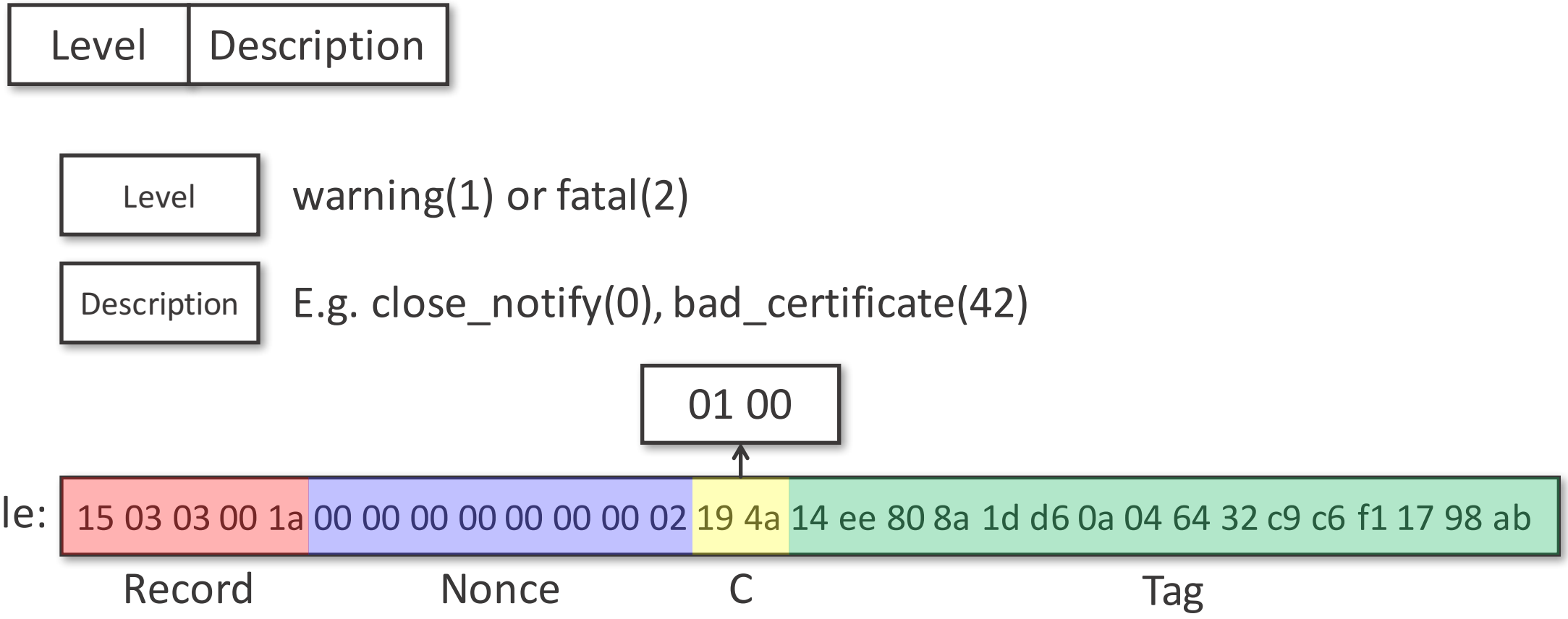
Subprotocols

- Data is sent using one of four subprotocols within the record layer

20 ChangeCipherSpec	Switching to a new encryption suite
21 Alert	Alert mechanism
22 Handshake	Handshake messages
23 Application Data	Opaque application data

TLS Alerts

- Perhaps the simplest of the subprotocols

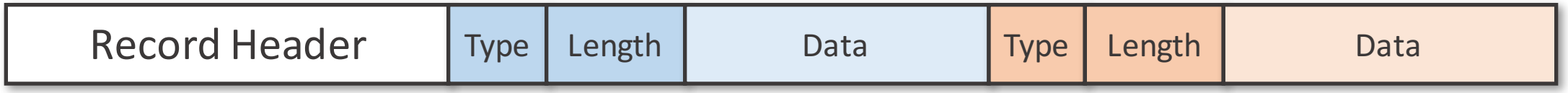


The TLS Handshake

- The TLS handshake protocol is used to establish parameters for the remainder of the session. It must:

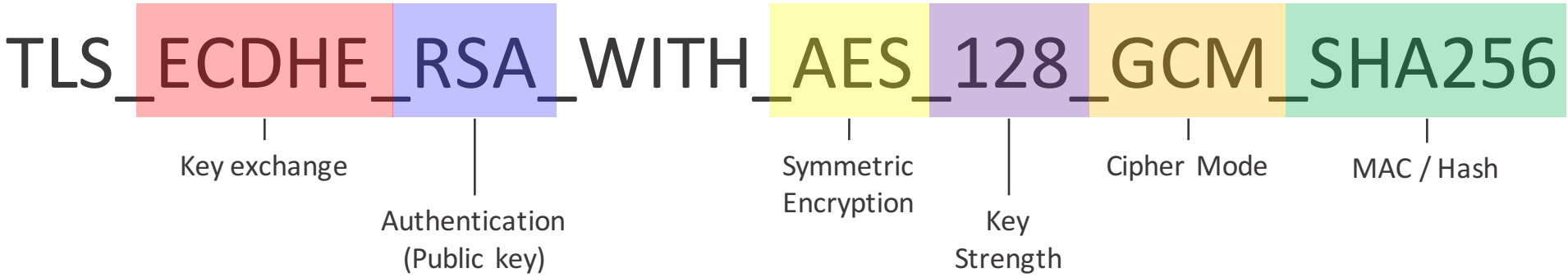
Agree ciphers and protocols	Establish shared secrets
Authenticate the server [and client]	Be robust to tampering and attack

- The handshake contains its own header:



Cipher Suites

- TLS defines cipher suites that describe the cryptographic primitives and other security parameters that will be used in that session:

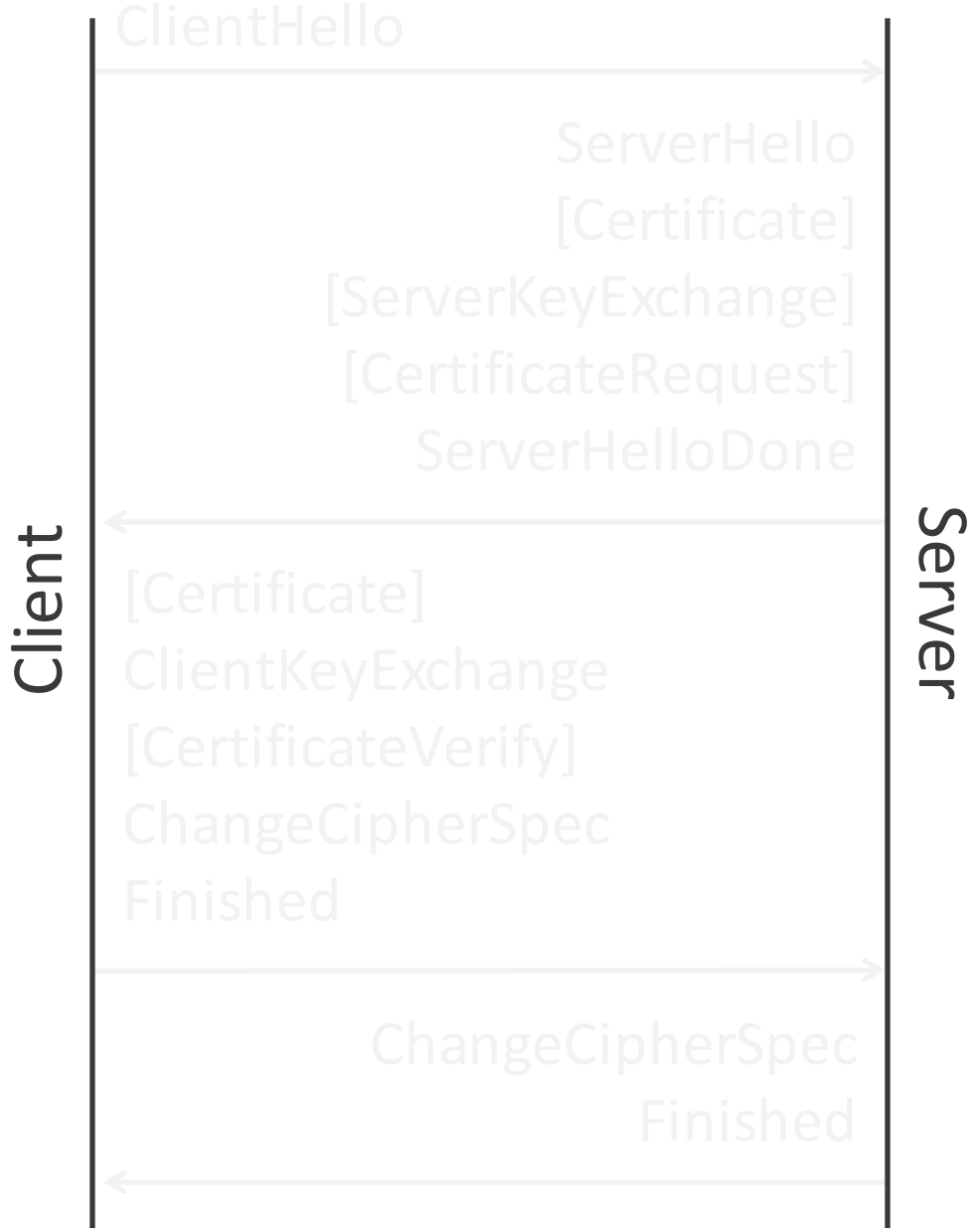


TLS Handshake

- We are aiming to:

  - Agree on encryption protocols
  - Establish a shared secret
  - Authenticate the server [and client]
  - Verify no tampering or attack

- This example is for ECDHE_RSA



TLS Handshake

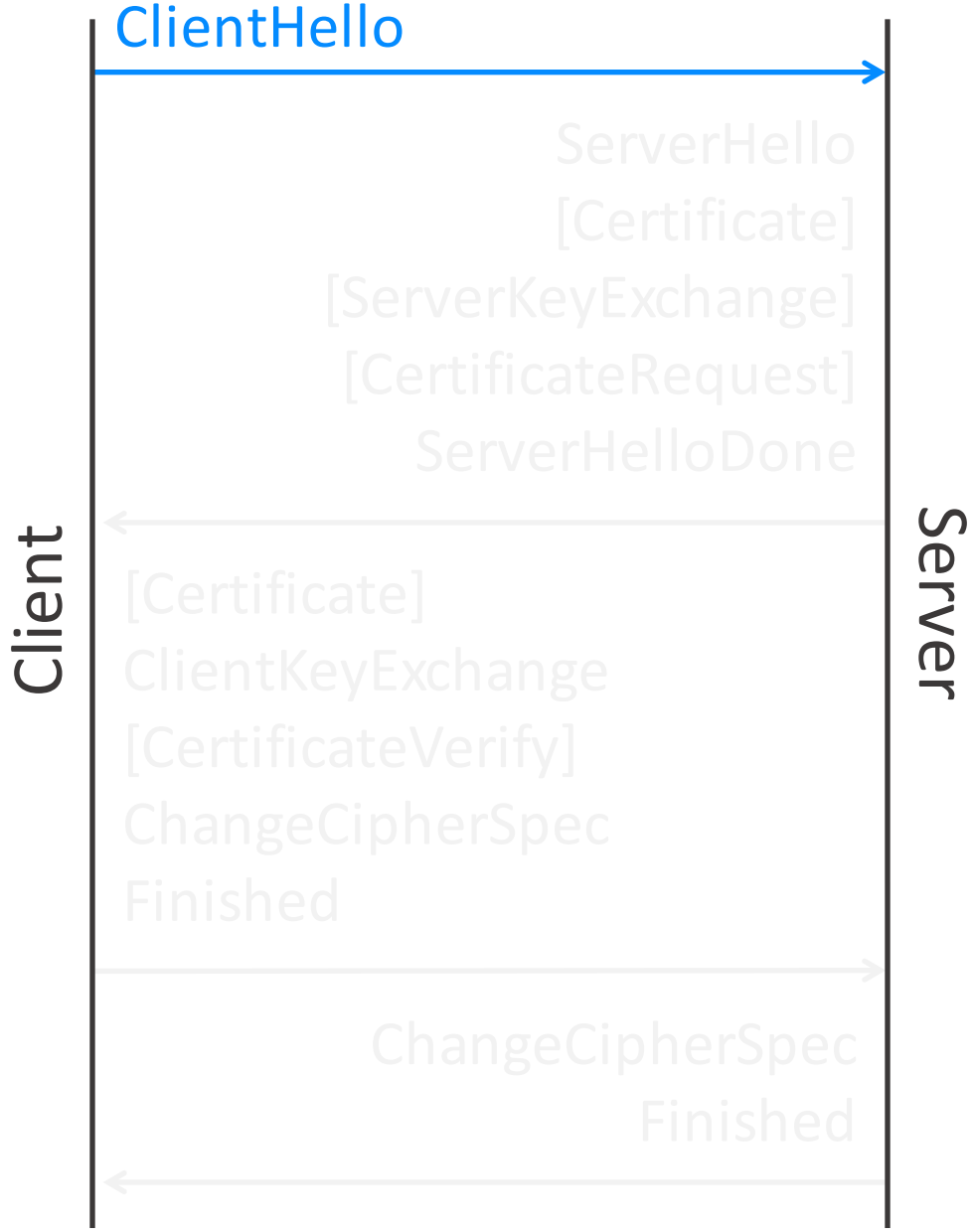
ClientHello

Max supported TLS version

Random number

List of supported cipher suites

What does the client support?



TLS Handshake

ServerHello

Chosen TLS version

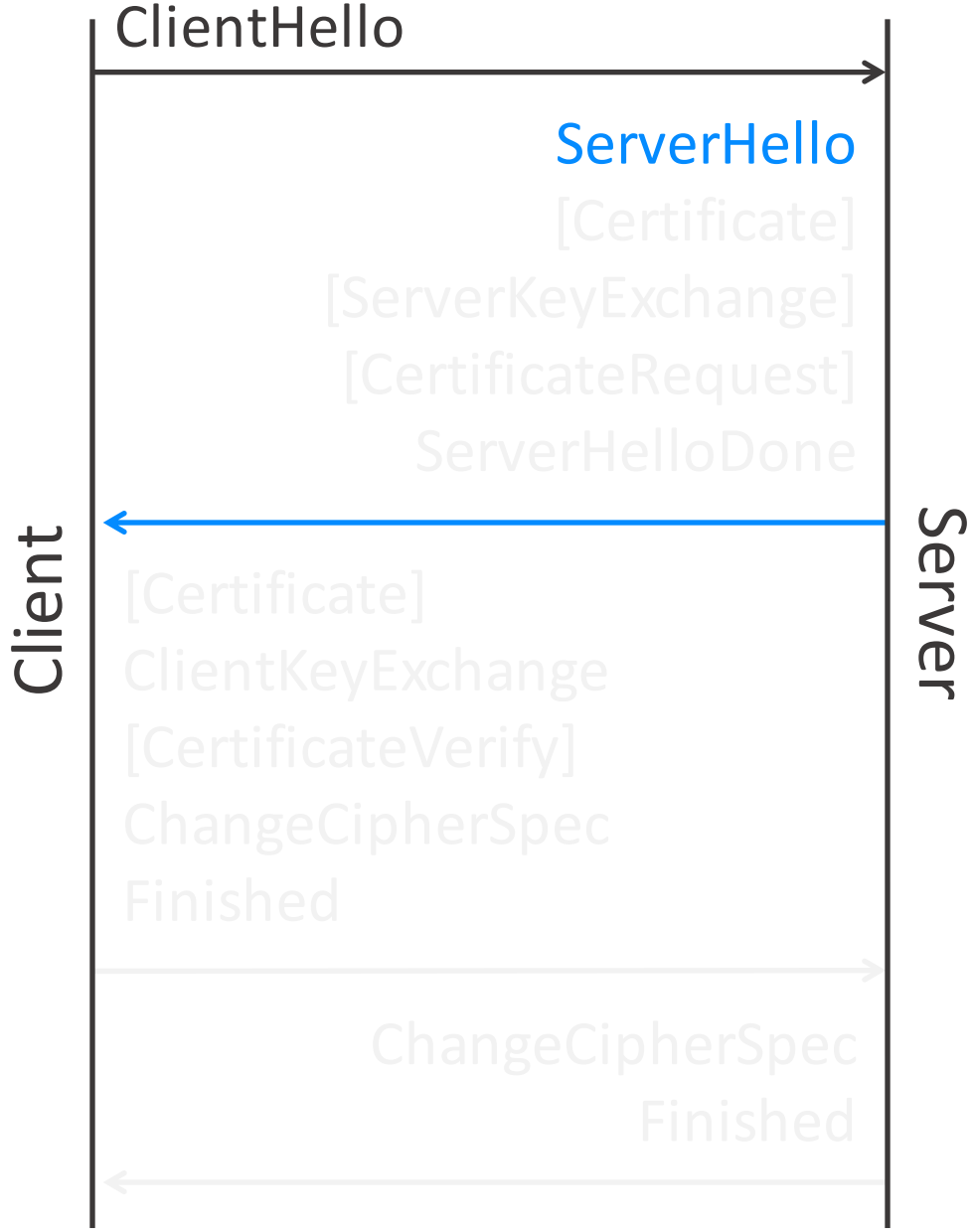
Random number

Chosen cipher suite

What symmetric cipher?

What key size?

What public key algorithm?



TLS Handshake

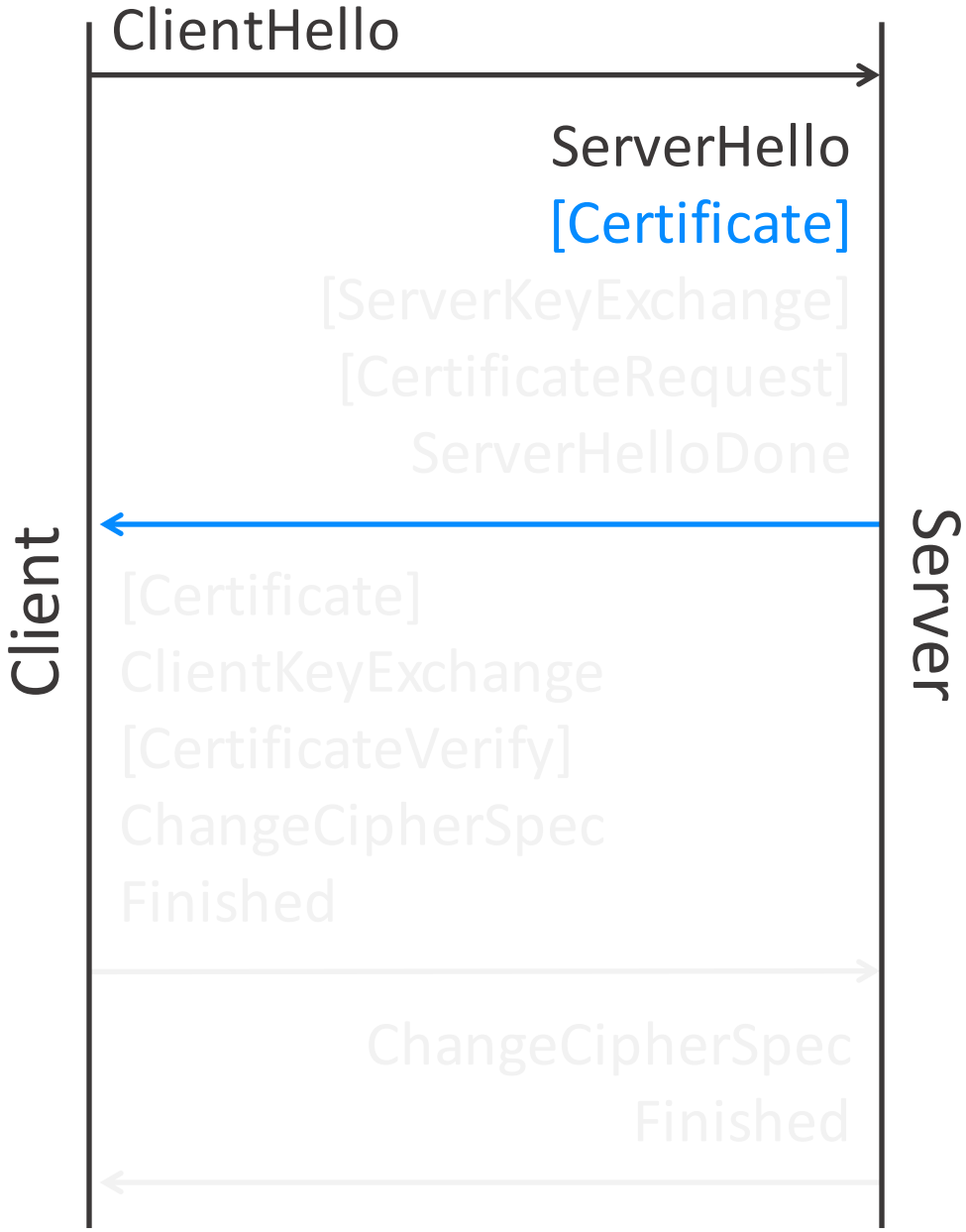
Certificate

The servers public key certificate

PKI

Certificate validation

The client checks that the public key certificate is valid using a root certificate



TLS Handshake

ServerKeyExchange

Key exchange parameters

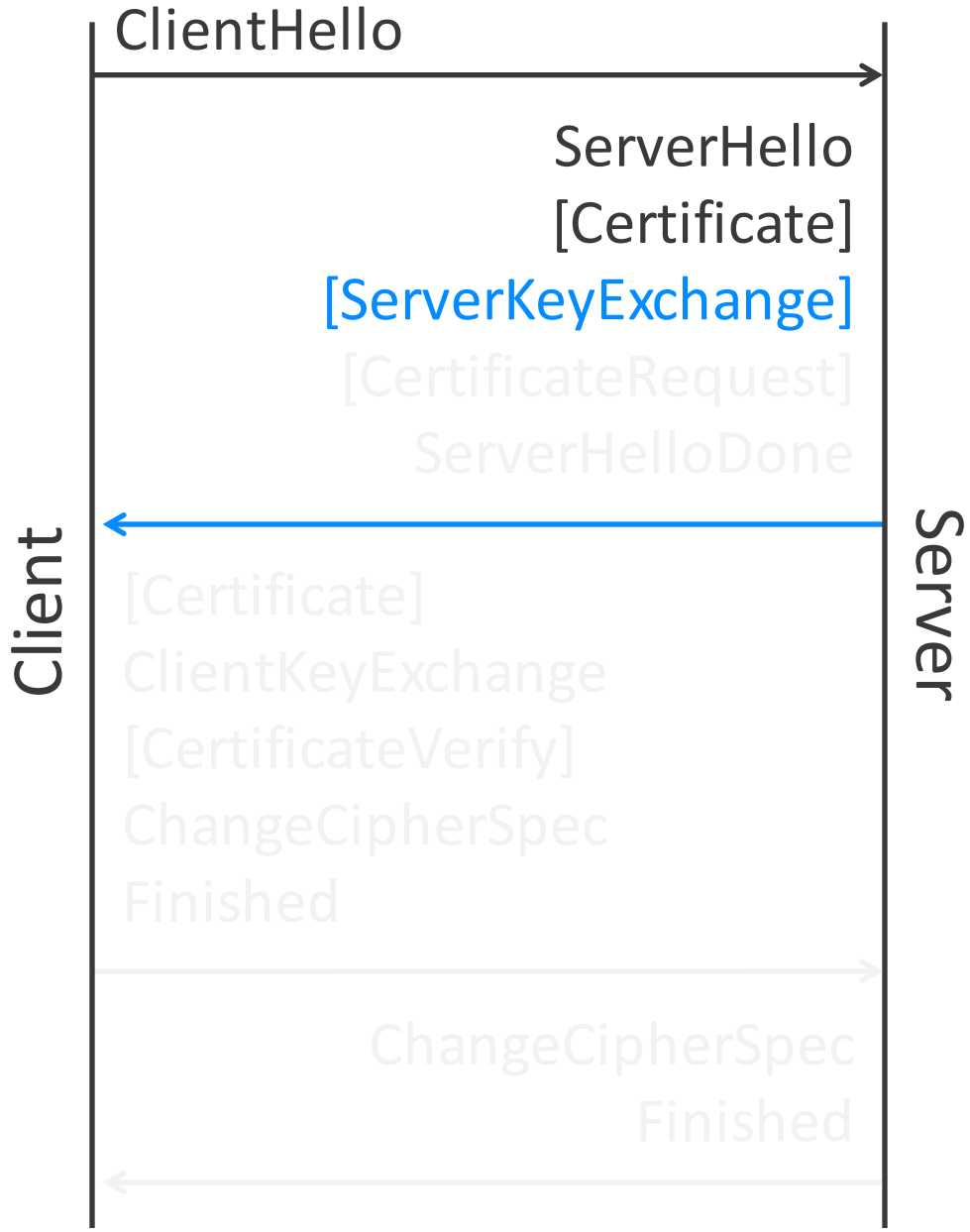
Key exchange public value

Digital signature

PKI

Authentication

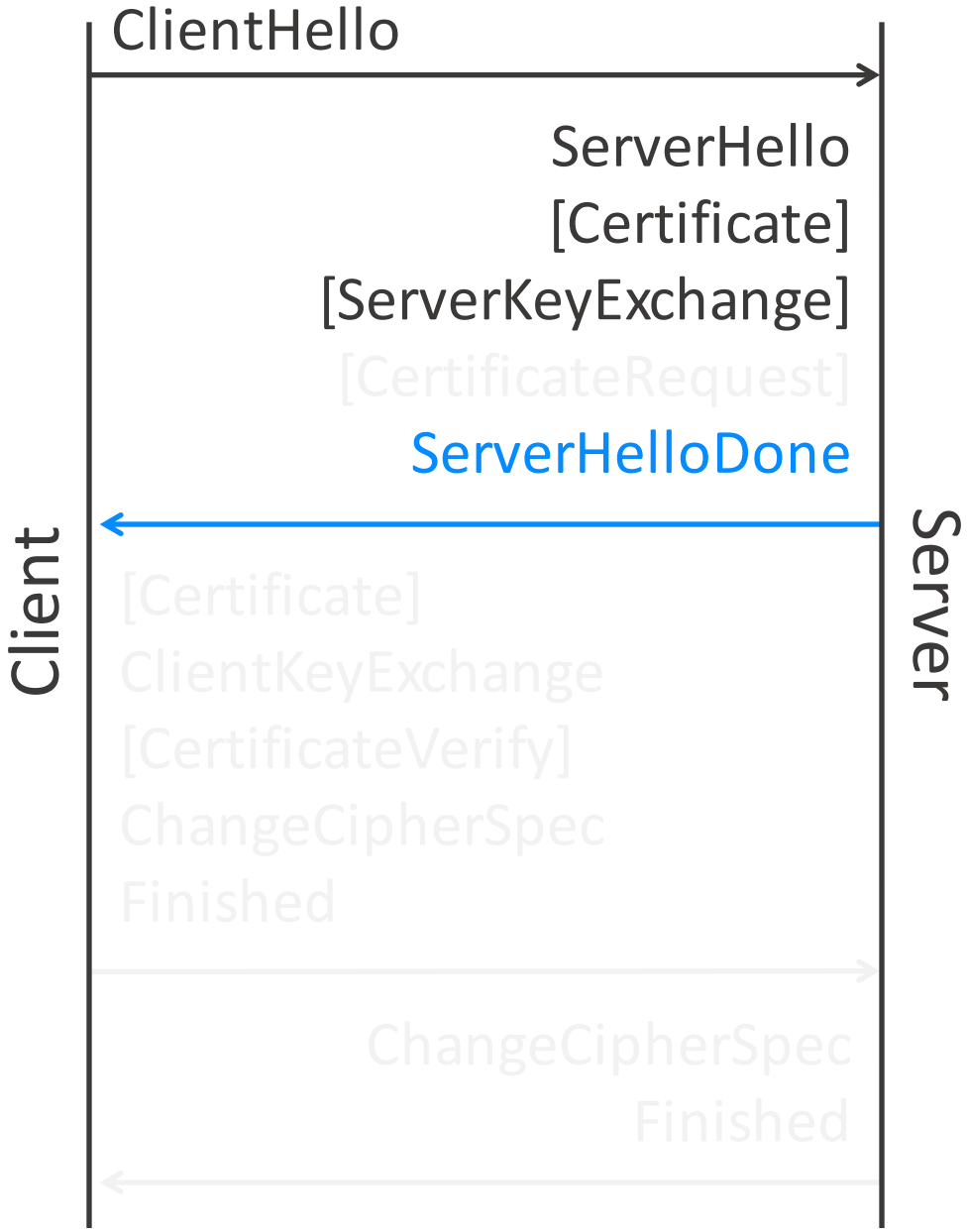
The client checks that the digital signature is valid



TLS Handshake

ServerHelloDone

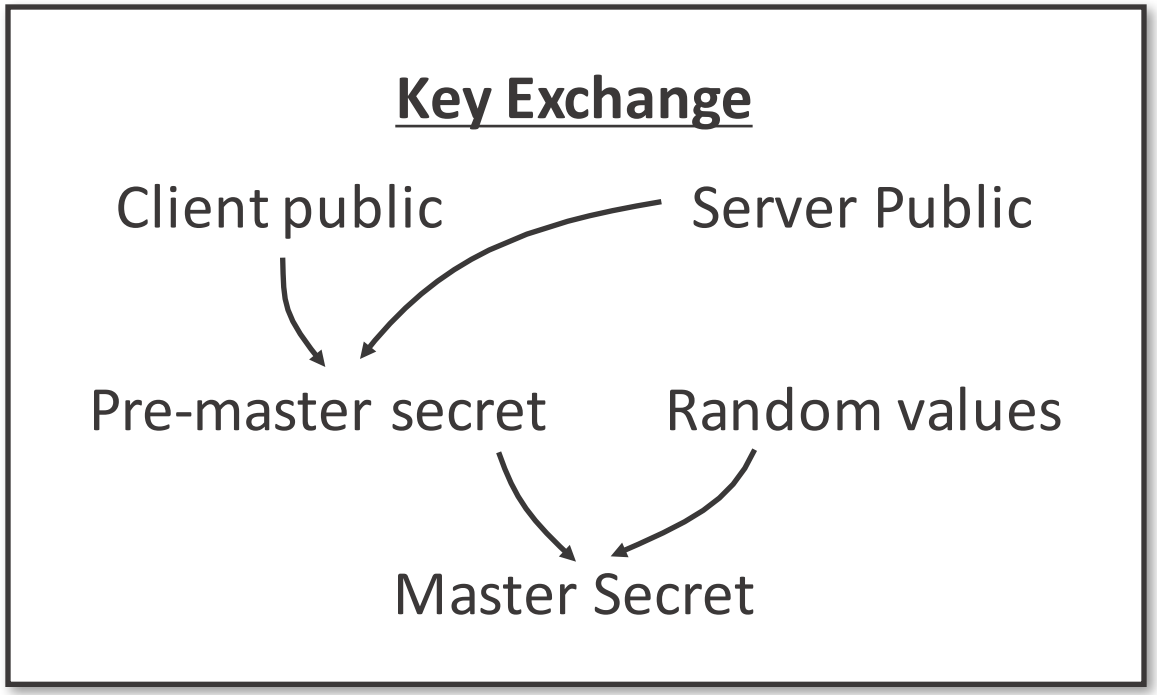
No further handshake messages to send

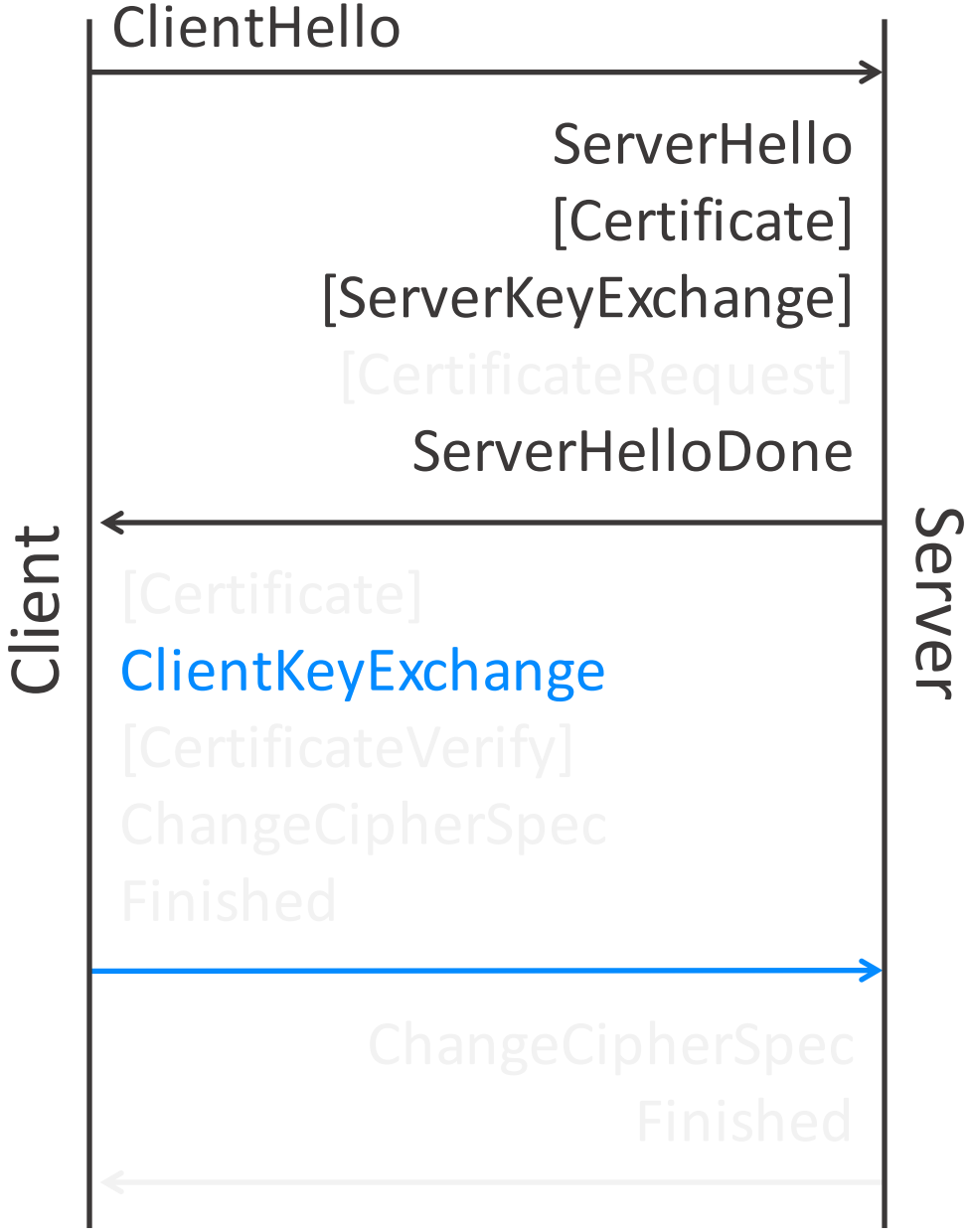


TLS Handshake

ClientKeyExchange

Key exchange public value



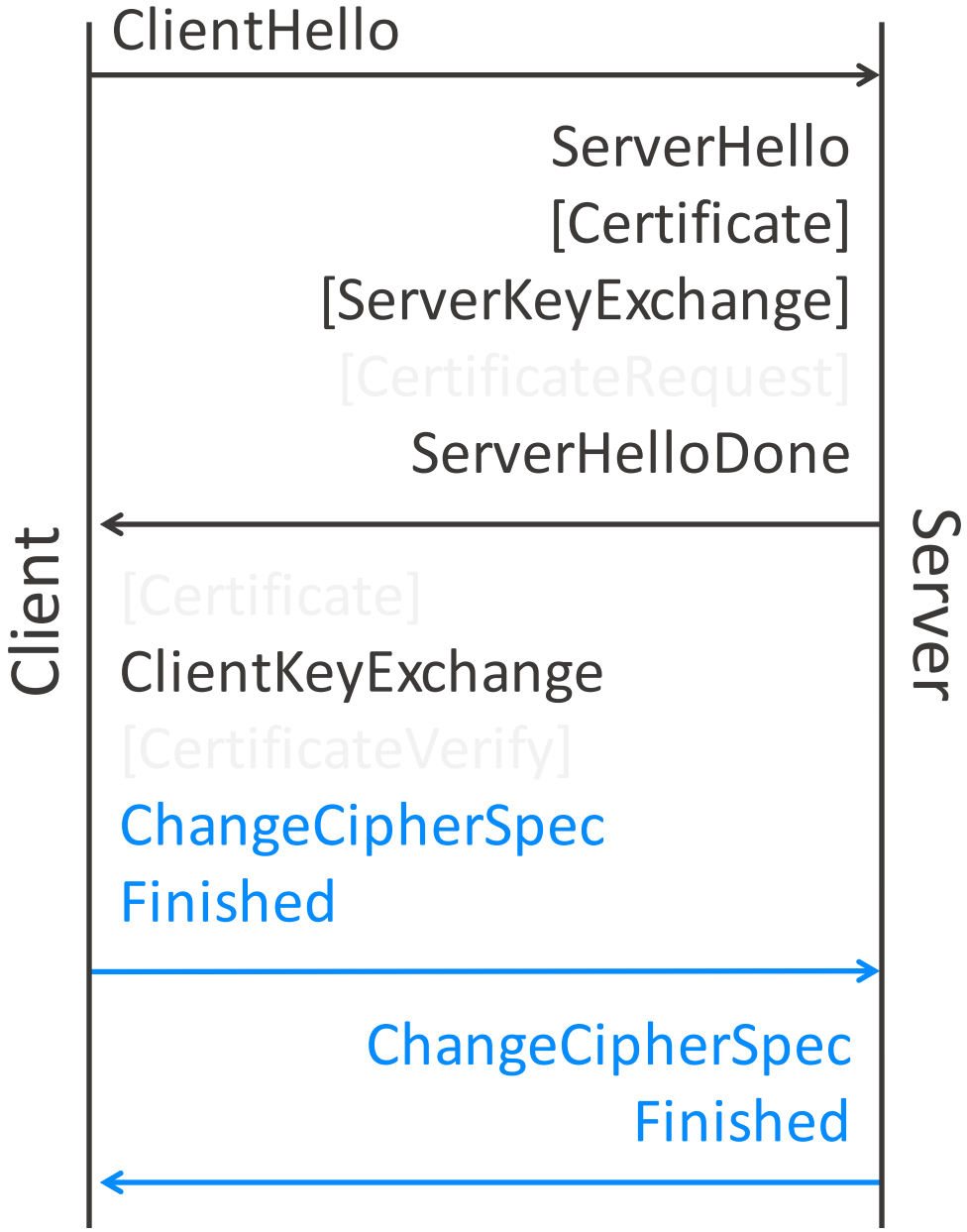


TLS Handshake

ChangeCipherSpec

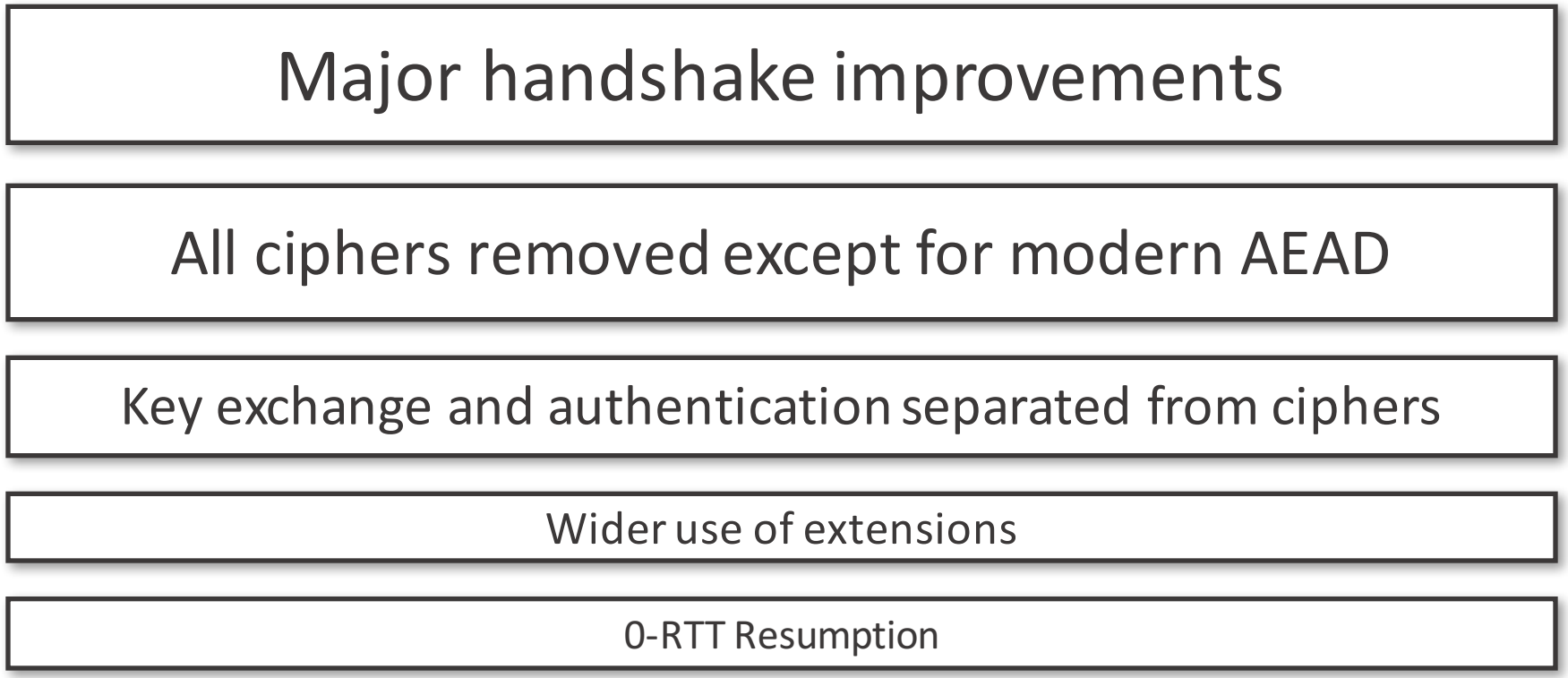
Finished

MAC (summary) of all previous messages



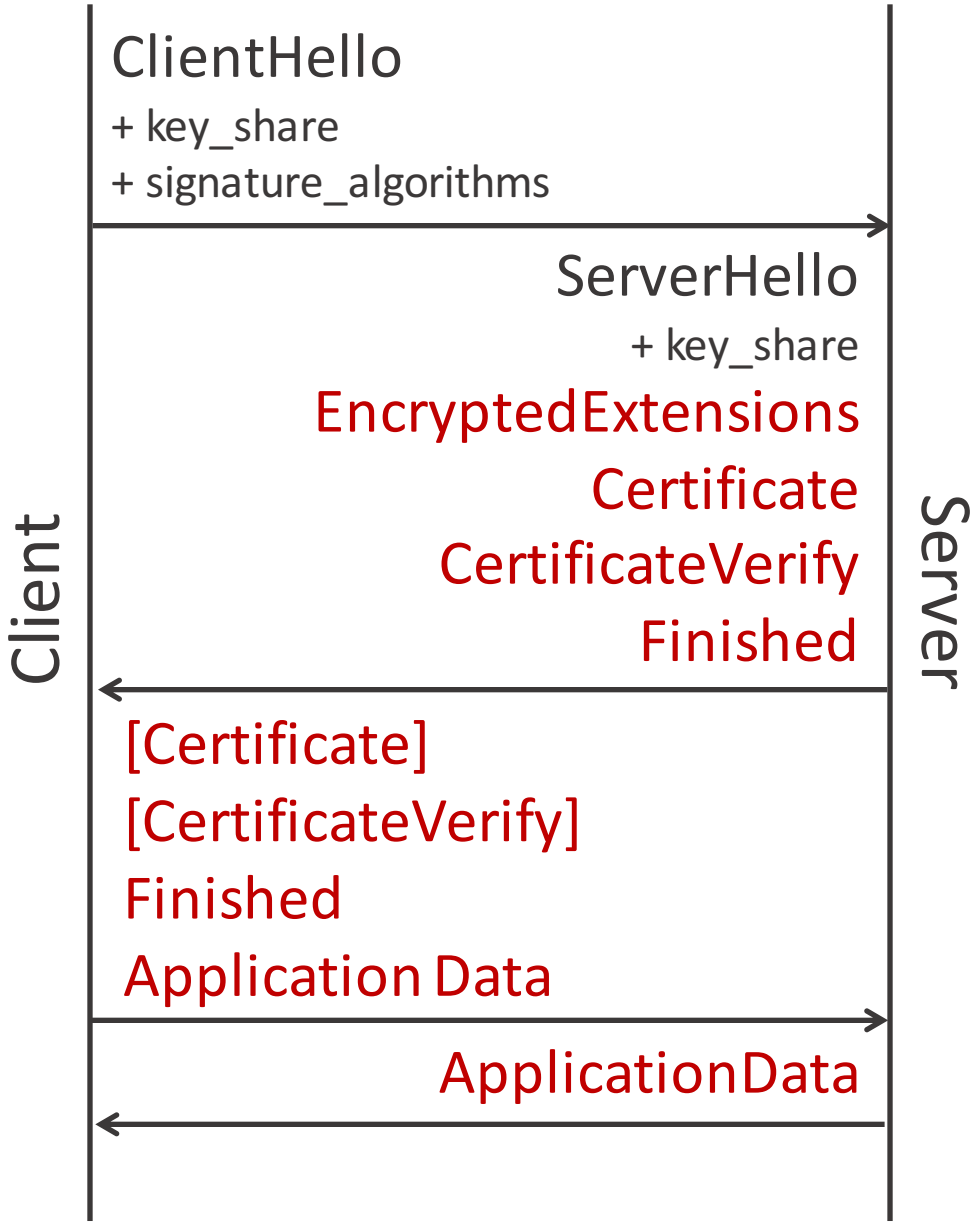
TLS 1.3

- TLS 1.3 is a major change – not just incremental improvements



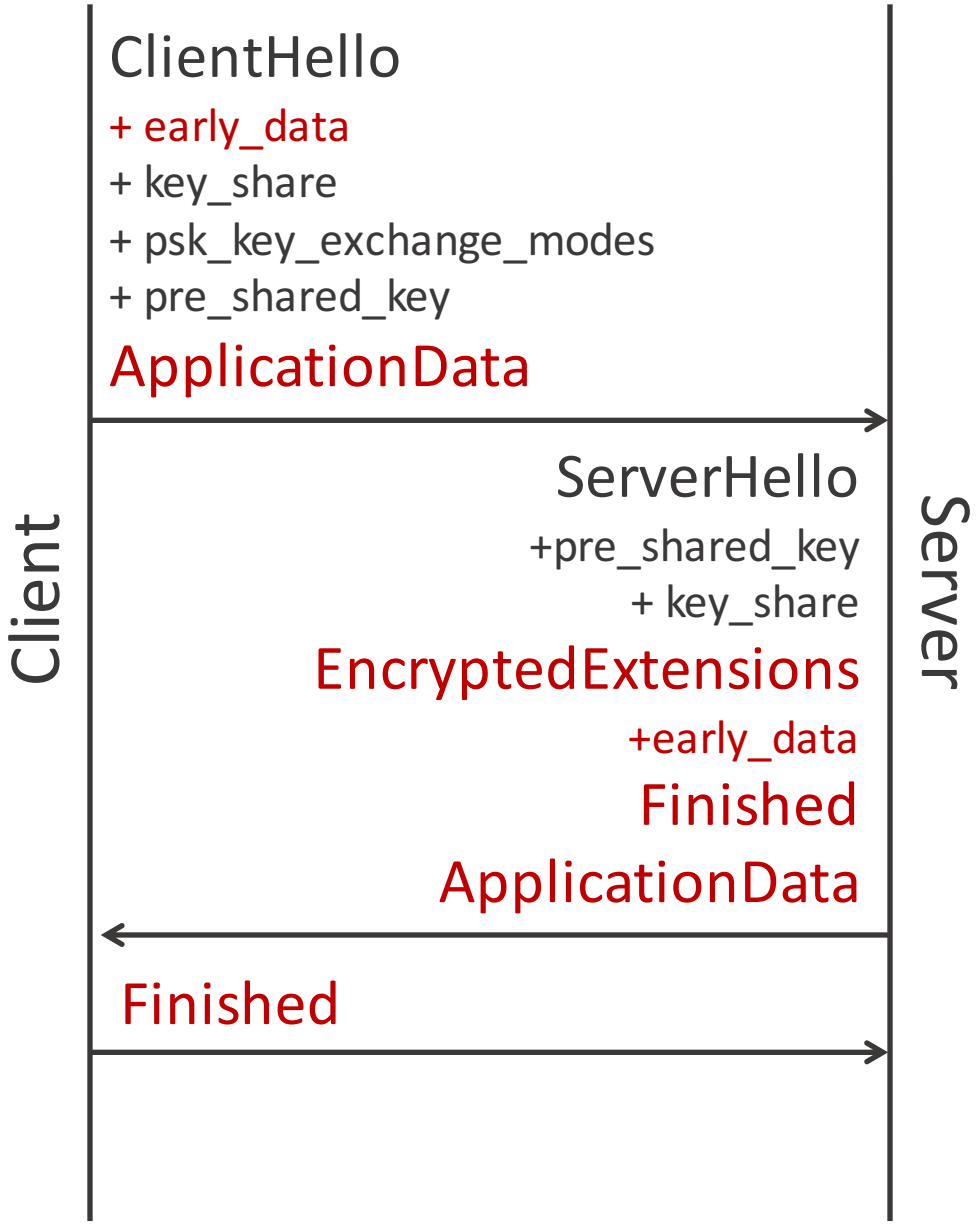
TLS 1.3 Protocol Differences

- The 1.3 handshake embeds much of the key exchange into the hello messages
- The client guesses a key sharing algorithm within ClientHello
- If the key share isn't supported, server sends a HelloRetryRequest
- Encryption begins during the second message



0-RTT Resumption

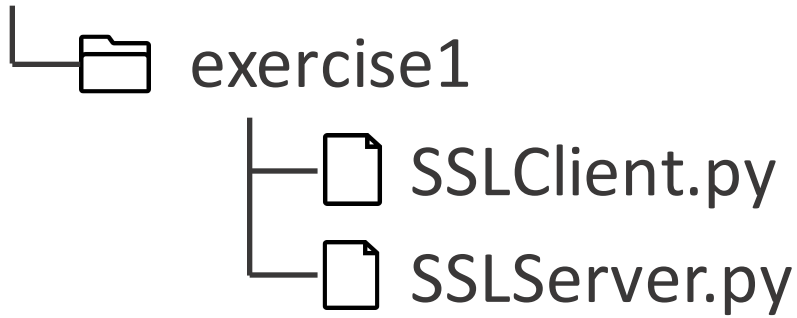
- During application data, the server can send a pre shared key for use next time
- Note this doesn't always supply perfect forward secrecy
- 1-RTT session resumption is closer to original 1.2 session keys



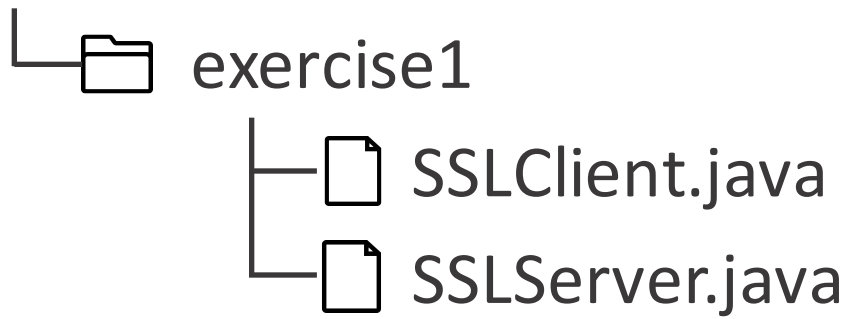
Exercise 1: Setting up a secure connection

Add code to the client to establish a working TLS 1.2 or 1.3 connection

Python



Java



1. Start the server
2. Alter the client to establish a connection
3. Test various cipher suites and protocol parameters

Public Key Infrastructure

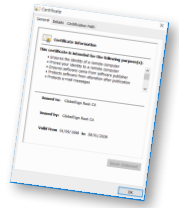
Why do we need PKI?

Server

Here's a digital signature that only I could have signed.

Signature

Here's a certificate you can use to verify it with the public key

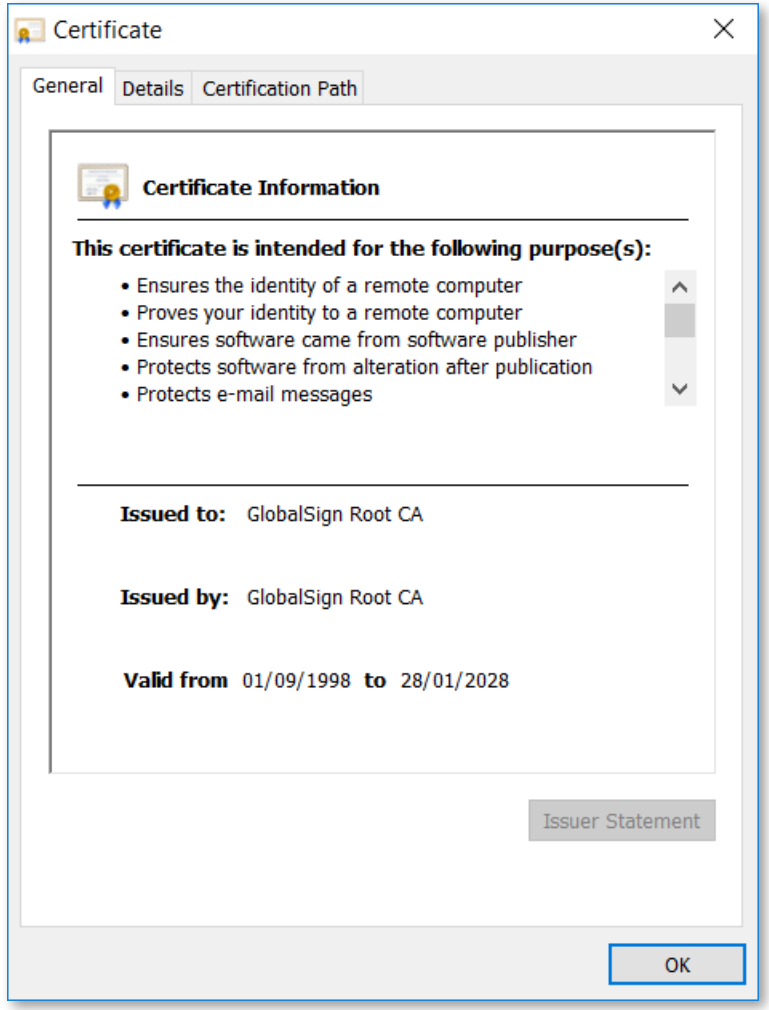


The key problem:

1. We have no real reason to trust this certificate
2. So we have no real reason to trust the signature

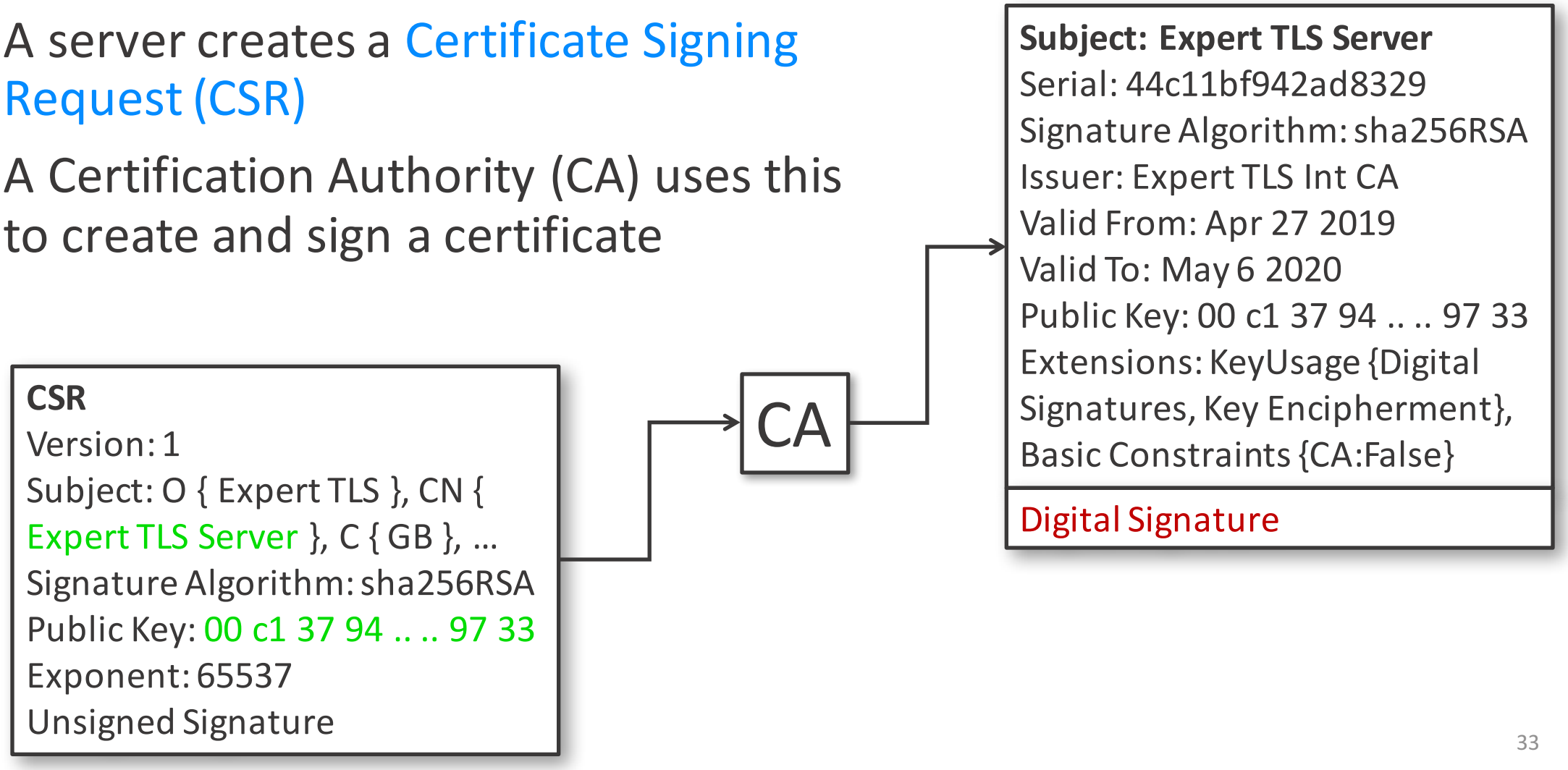
Digital Certificates

- If we want to use public key cryptography, we need trust
- We can use a trusted third party in order to verify the ownership of a public key
- Primarily managed through Public Key Infrastructure (PKI)
- Certificates usually held in X509 format



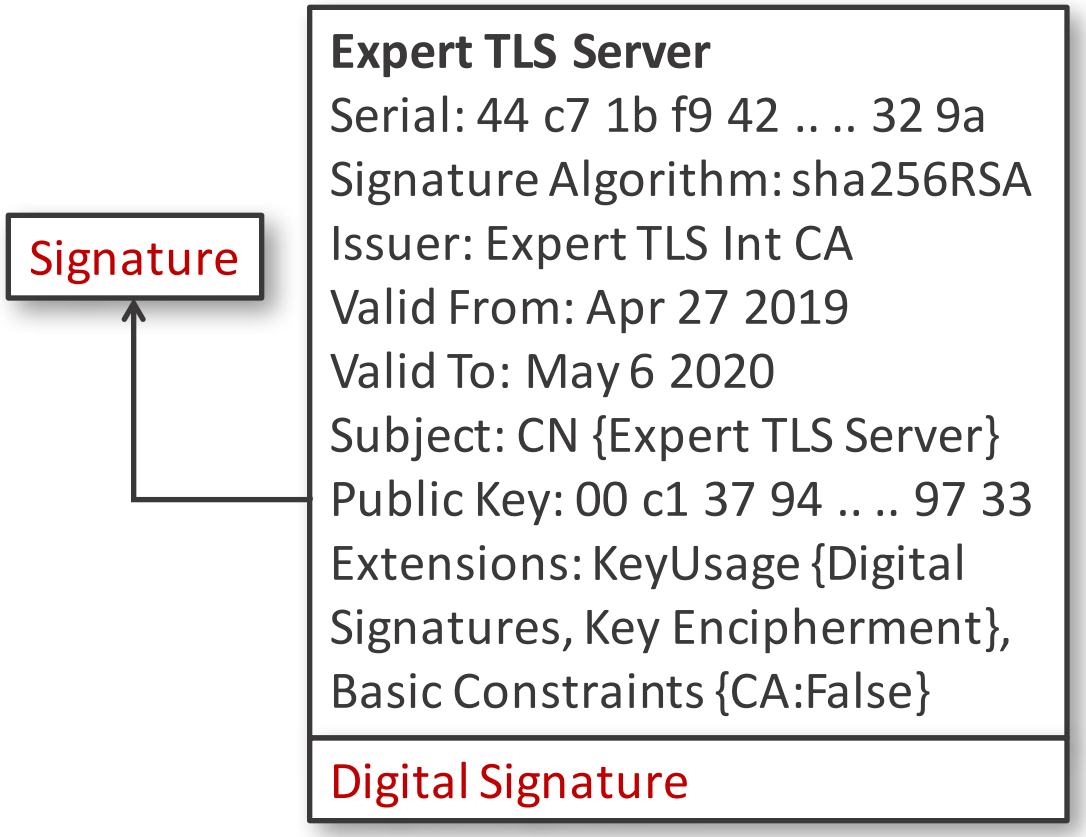
Certificate Issuance

- A server creates a Certificate Signing Request (CSR)
- A Certification Authority (CA) uses this to create and sign a certificate



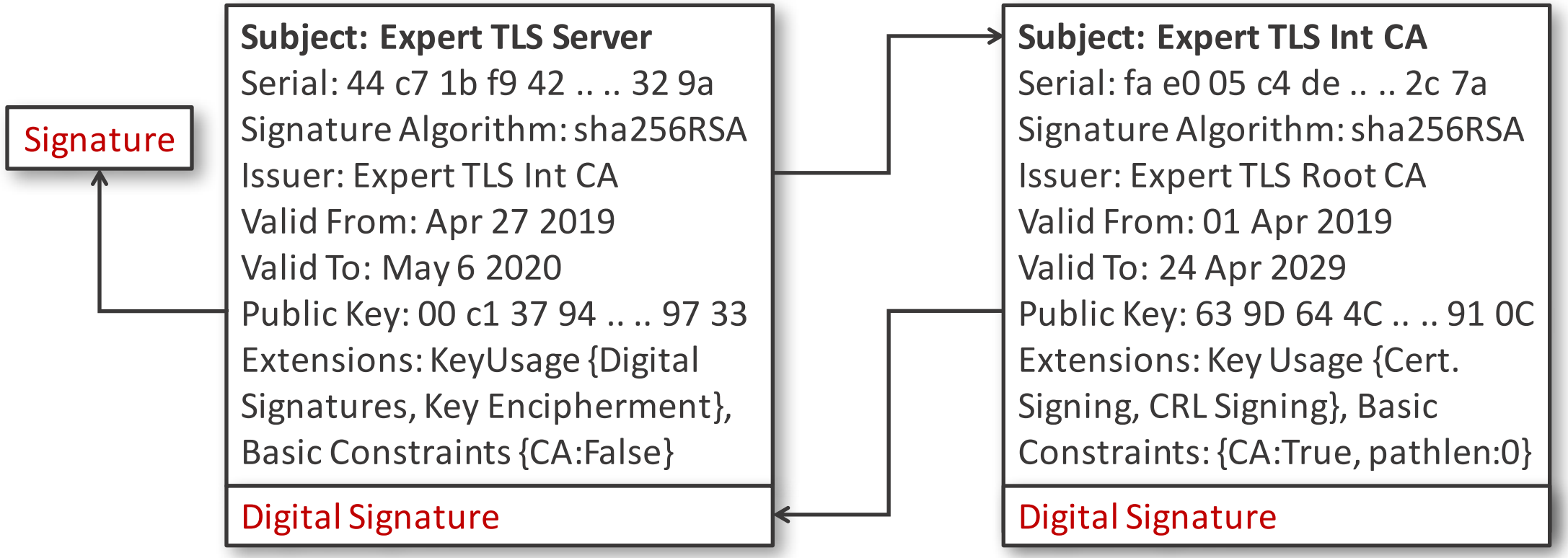
Certificate Use

- The server can supply digital signatures using their key pair, backed by the certificate when requested, e.g. during a TLS handshake



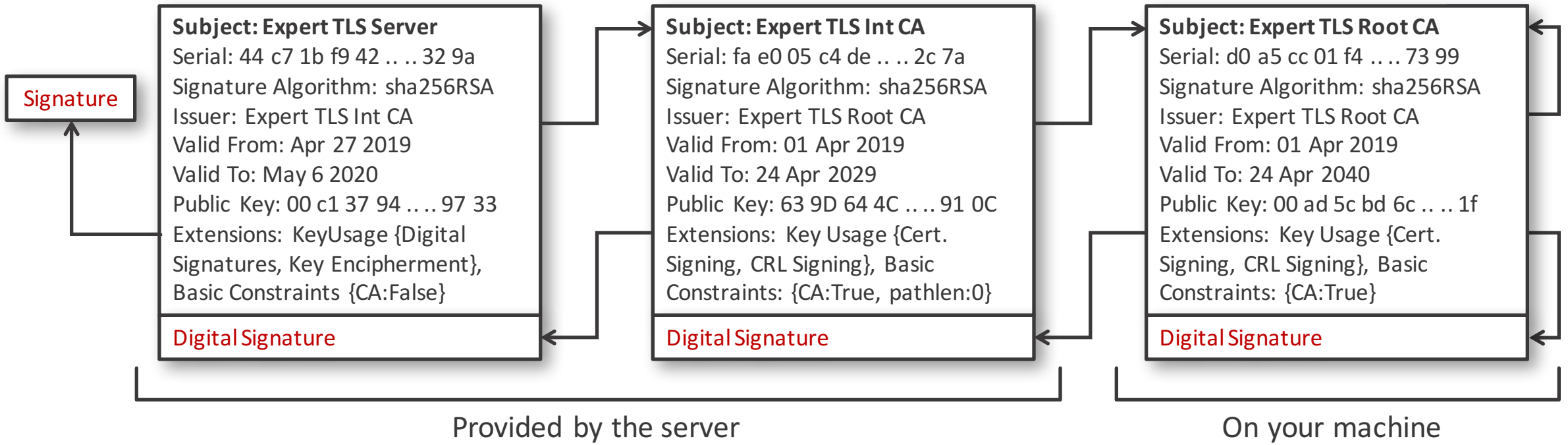
Chains of Trust

- To verify the trust in the server certificate, we need to examine the signing certificate



Chains of Trust

- In many cases, the chain involves multiple certificates
- Chains always end in a root certificate, located on your machine



Who Manages the Root Certificates?

- Major OS vendors operate root certificate programs
  - Apple for iOS and OS X
  - Microsoft for Windows
- Mozilla maintains root certificate store
  - Used in Linux, firefox
- Chrome tends to use the OS store, with additions of its own
  - EV CA List
  - Untrusted blacklist
- Android, Java etc. also ship with root stores
  - Often interfered with by vendors!

Advanced Transport Layer Security

Mutual Authentication

- Client authentication is rare on the web, but useful in some circumstances
  - Protected areas on web servers may require additional authentication
- Non-web systems use mutual authentication much more often
  - Limited numbers of clients
  - Useful for connecting systems over networks securely

Mutual Authentication

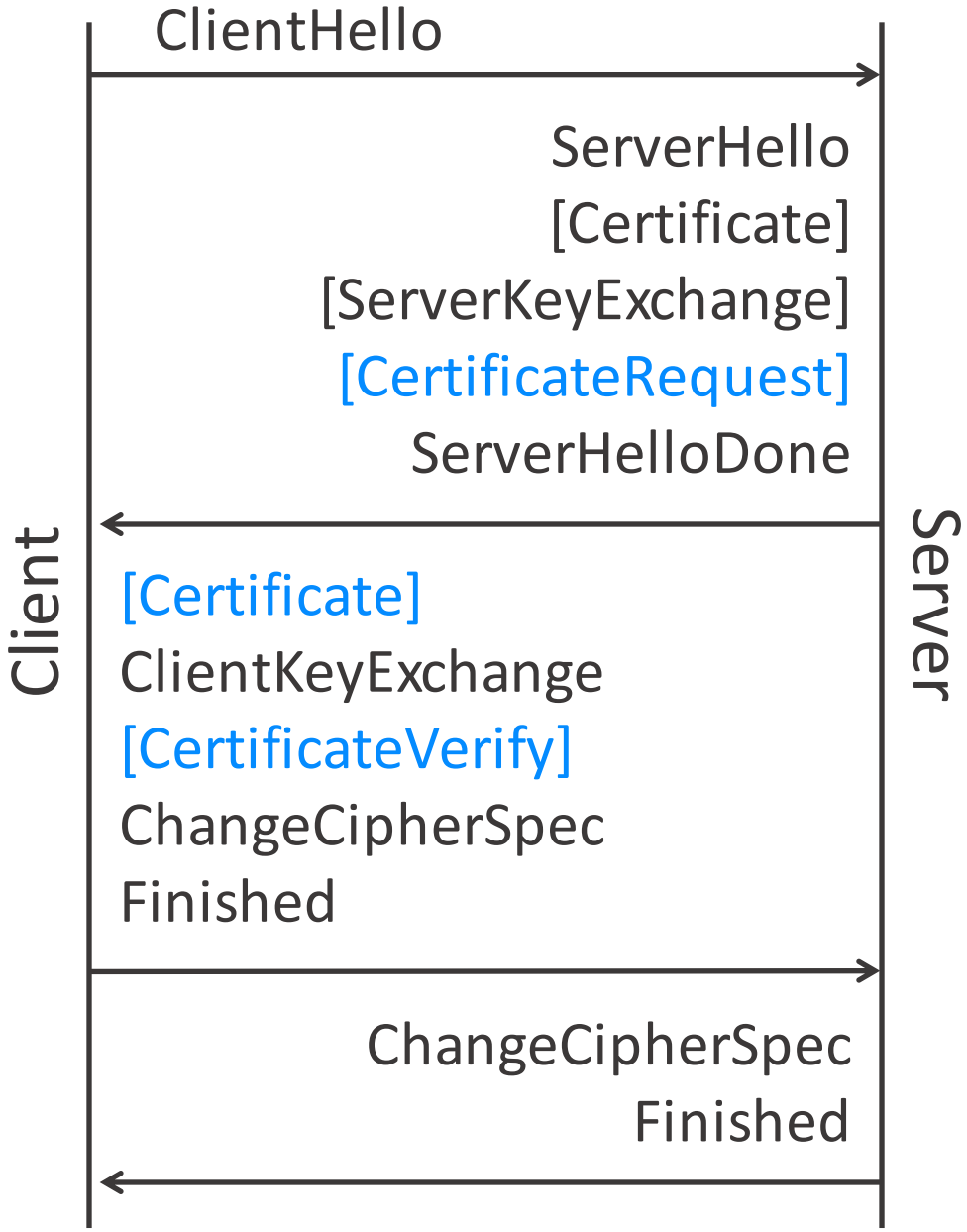
CertificateRequest

Includes supported algorithms and optionally CAs

Certificate

CertificateVerify

The client computes a signature over all previous handshake messages



Renegotiation

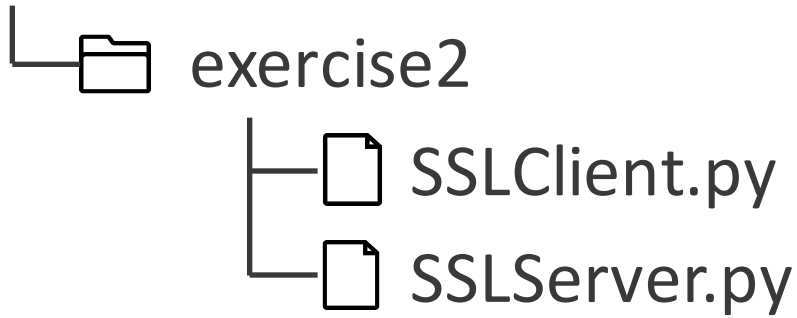
- **TLS v1.2** allows for renegotiation using a new handshake
- Useful for:
  - Upgrading encryption, information hiding, client authentication, renewing keys
- Initiated via a new ClientHello or a HelloRequest

- **TLS v1.3** doesn't support renegotiation:
  - Clients uses the post_handshake_auth extension, after which servers can send a CertificateRequest message at any time.
  - Either party can send a KeyUpdate handshake message at any time, deriving a new set of keys

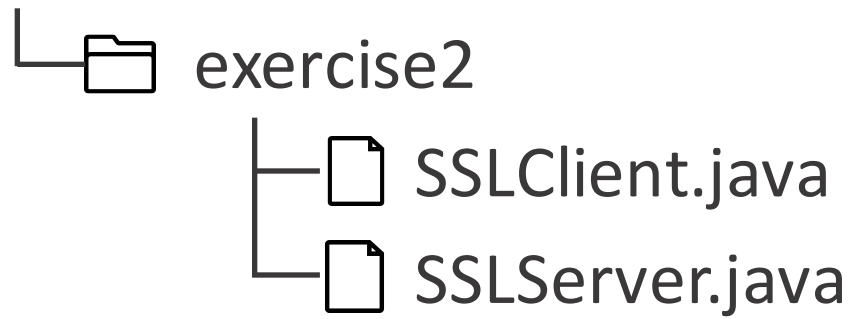
Exercise 2: Mutual Authentication

The TLS connection already works, now add client certificate authentication

Python



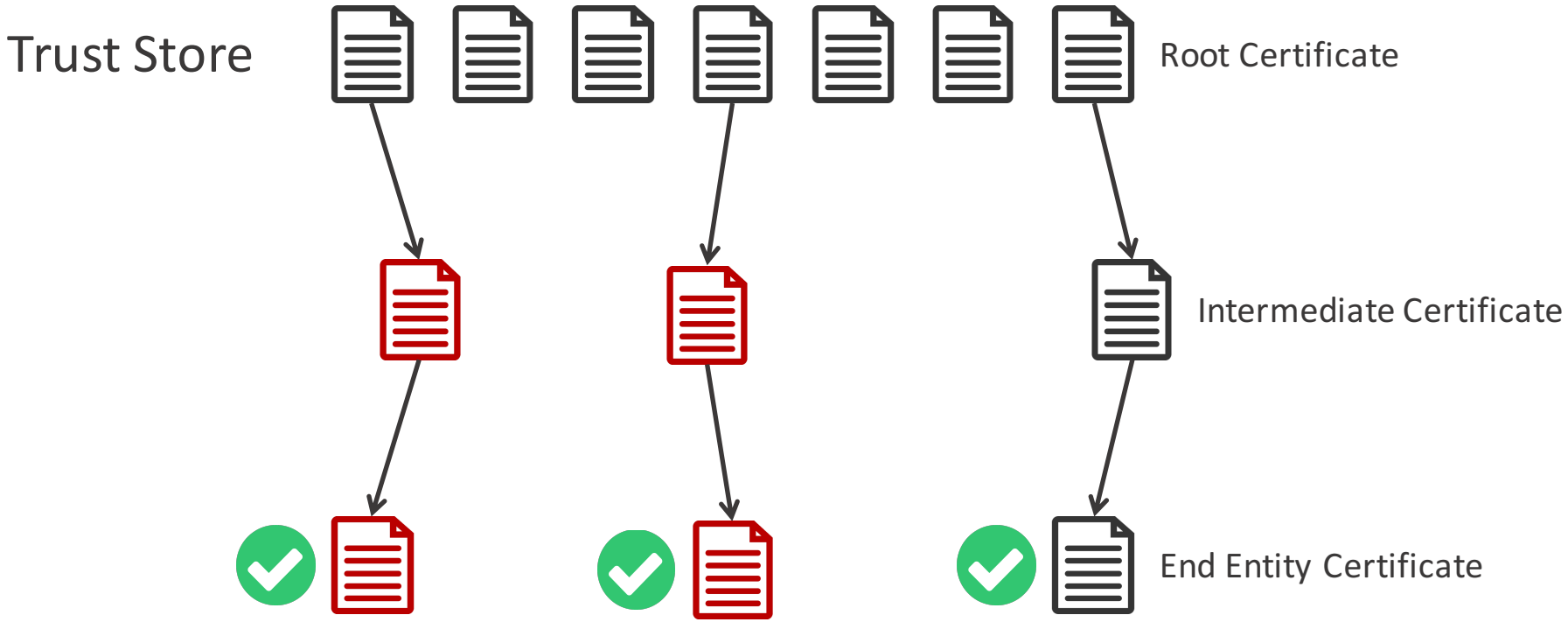
Java



1. Instruct the server to request a client cert
2. Provide a root certificate for the server
3. Provide a certificate chain for the client

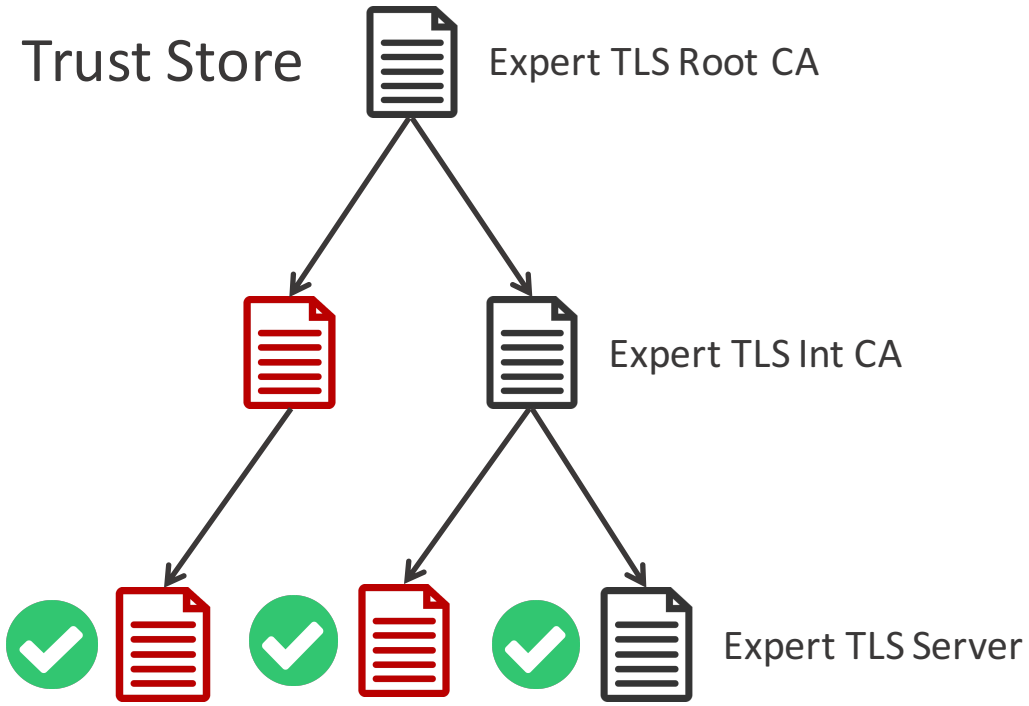
Certificate Pinning

- The main problem with PKI is that once you've trusted one or more CAs, you trust every certificate signed by them



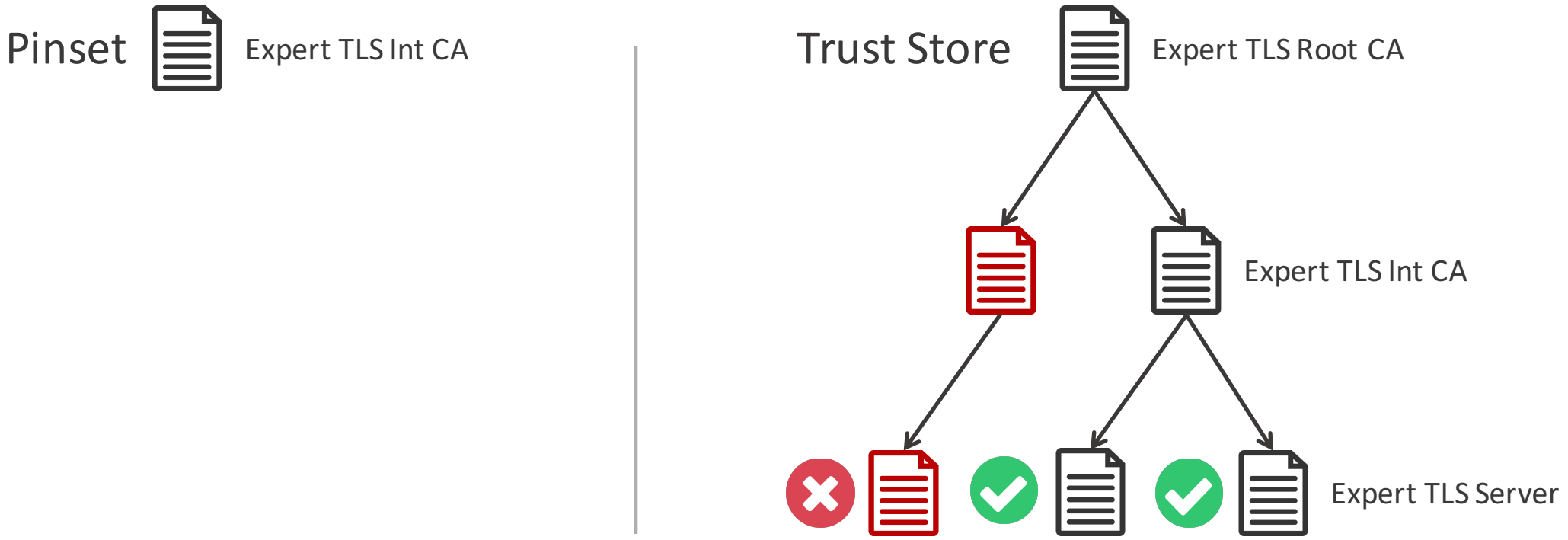
Certificate Pinning

- In our exercise we only trust a single root



Certificate Pinning

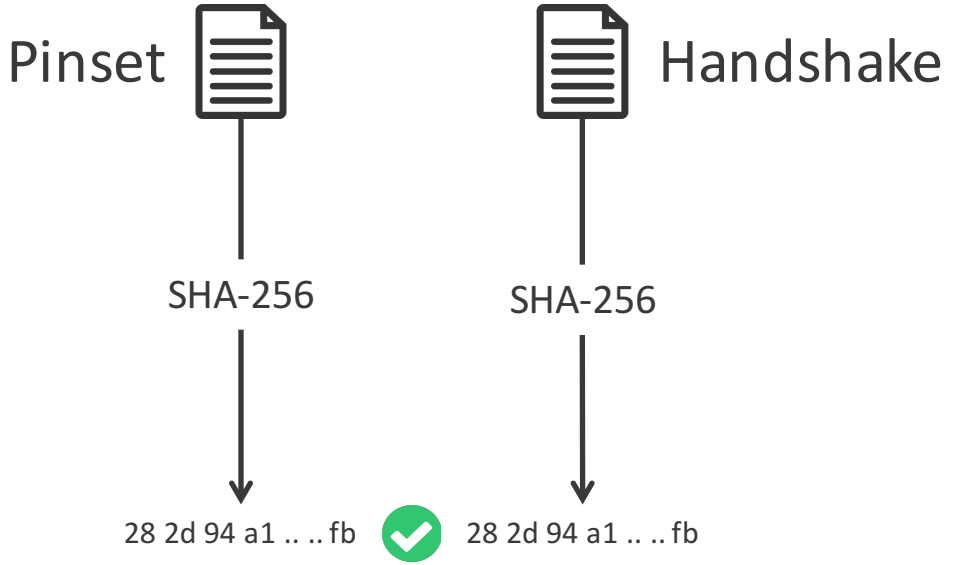
- Certificate pinning is the permanent association of a service with one or more identities such as public keys or certificates



How to Pin?

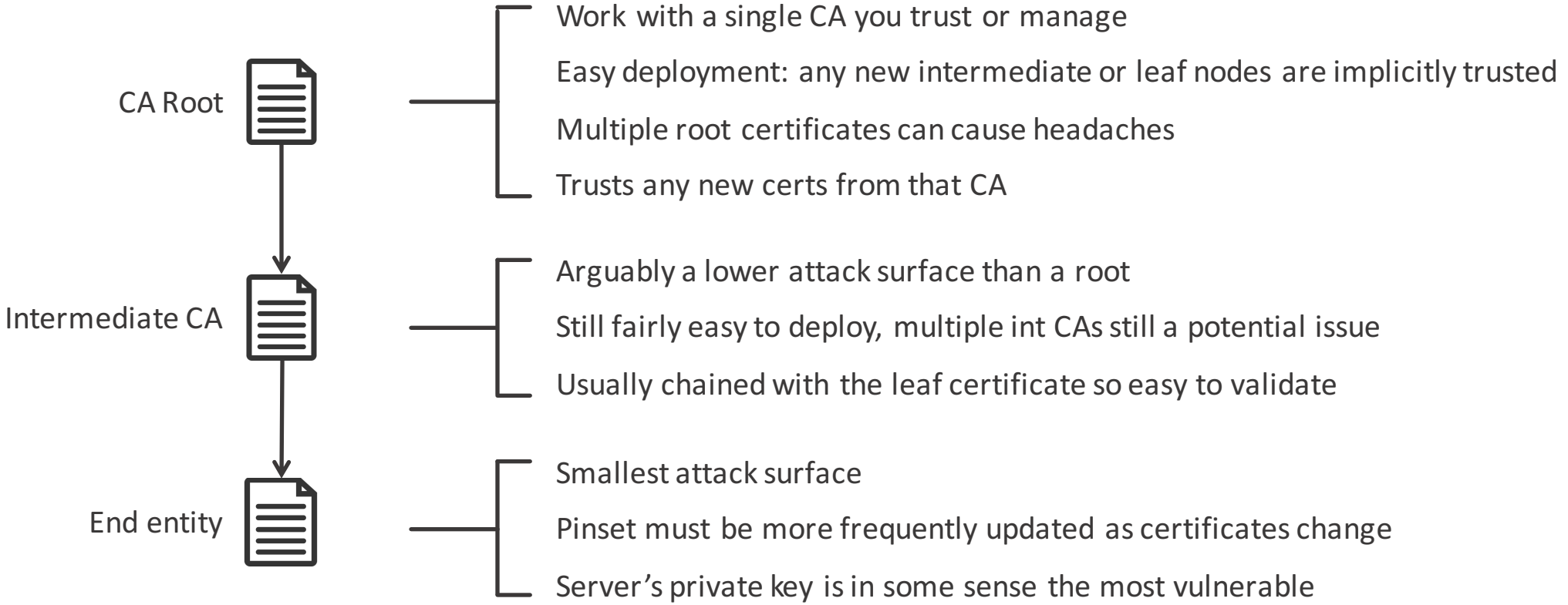
- Usually the best approach is to store a hash (SHA-256) of either the certificate or public key for any trusted certificates
- These are stored in a pinset

- During certificate verification, also check certificate against the pinset
- Standard verification such as validity period and signatures is also still important!



Where and what to Pin?

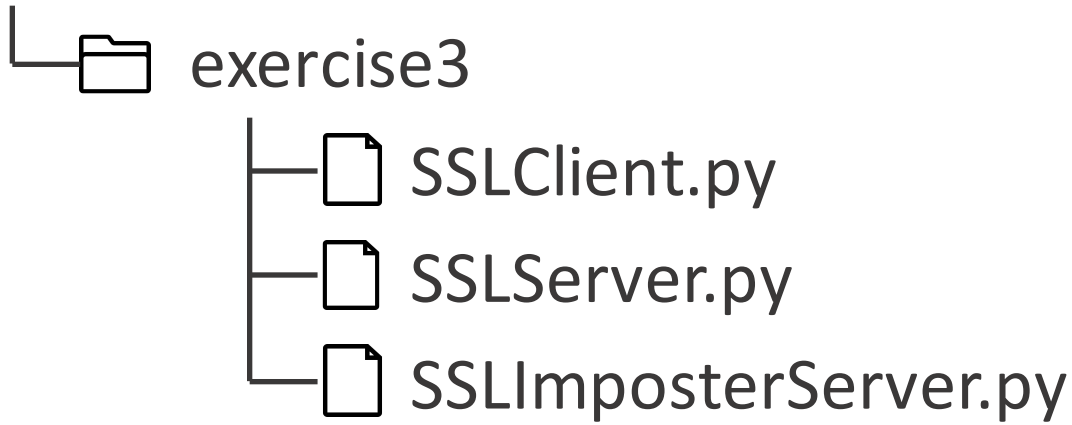
- In native applications pinning massively reduces your attack surface
- Can make deployment slightly more complex – usually worthwile



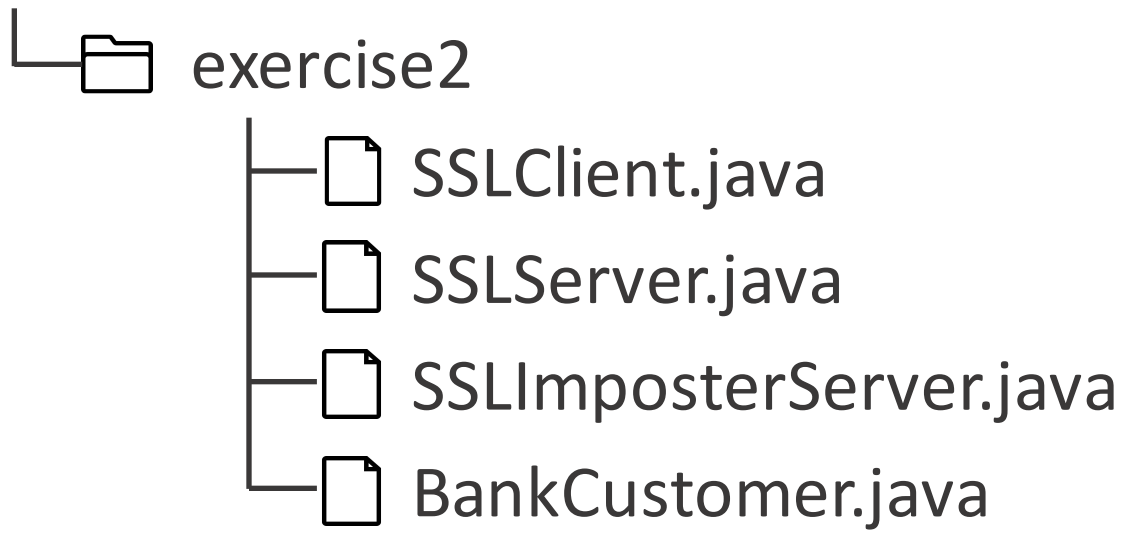
Exercise 3: Certificate Pinning

Pin the server certificate so that the client doesn't accept an imposter

Python



Java



1. Obtain the server's certificate in the client during or after the handshake
2. Hash the DER certificate and compare with the pinned version

Good Practice and Guidance

Protocols and Suites

Protocol Support

- If you're running a public web server, you may need to support as low as TLS 1.0 to serve the majority of clients
- In a non-public system, you can happily disable everything below TLS 1.2

Cipher Support

- TLS 1.3 disables most ciphers by default. In 1.2, if you know your client and server support them, restrict to a select few

Key Share

- Always stick to ephemeral key shares

Key Management

Protect private keys

- Avoid CAs that generate private keys for you
- If you run a CA – keep the root key in a locked box!
- Use passwords – unless absolutely necessary for deployment
- Consider using a hardware security module

Don’t share keys

- If servers are unrelated, they shouldn’t use the same keys

Rotate Keys

- Usually yearly depending on security requirements

Certificate Issuance

- Which is best?

Public CA

- A must for public websites
- Chose a CA that supports what you need

Self-signed

- Harder to deploy
- May encourage bad practice

Private CA

- More upfront work
- Easier to deploy
- As secure as PKI

Certificate Management

Use intermediates

- Don’t sign server and client certs with a root key
- Provides a level of indirection, and easier revocation

Certificate sharing

- Is it ok to share www.server.com, news.server.com, *.server.com?
- This is quite common, but remember sharing certs means sharing keys – be careful

Support OSCP or CRLs for your private CA

- Depending on your use case, it might be important to be able to revoke

Use chains correctly

- Send certificates in the correct order, and don’t send the root certificate

Diagnosing Issues

Hostname verification issues

- Most clients verify the host name unless told otherwise
- For a website, this would usually be the domain name
- You can supply multiple in a certificate using the subjectAltName extension

Handshake errors

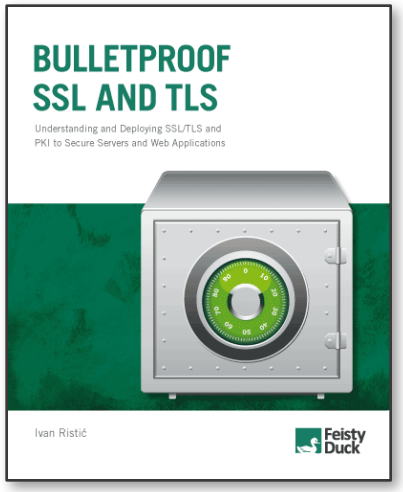
- Don’t forget you need at least one protocol and one cipher suite in common between the client and the server
- If it’s not working, enable everything before removing things slowly

Where Now?

- Use OpenSSL – experiment with the CA that's part of the course exercises
- Use Wireshark to examine TLS handshakes and communication
- Further reading:
  - Bulletproof SSL and TLS: Ivan Ristić

OpenSSL

WIRESHARK



- TLS 1.2
  - https://tools.ietf.org/html/rfc5246
- TLS 1.3
  - https://tools.ietf.org/html/rfc8446
- PKI
  - https://cabforum.org
- OpenSSL
  - https://www.openssl.org/
- Mozilla server config
  - https://ssl-config.mozilla.org/