Securing IoT applications with Mbed TLS

Hannes Tschofenig

Part#2: Public Key-based authentication

March 2018
Munich

Agenda

- For Part #2 of the webinar we are moving from Pre-Shared Secrets (PSKs) to certificated-based authentication.
- TLS-PSK ciphersuites have
  - great performance,
  - low overhead,
  - small code size.
- Drawback is the shared key concept.
- Public key cryptography was invented to deal with this drawback (but itself has drawbacks).

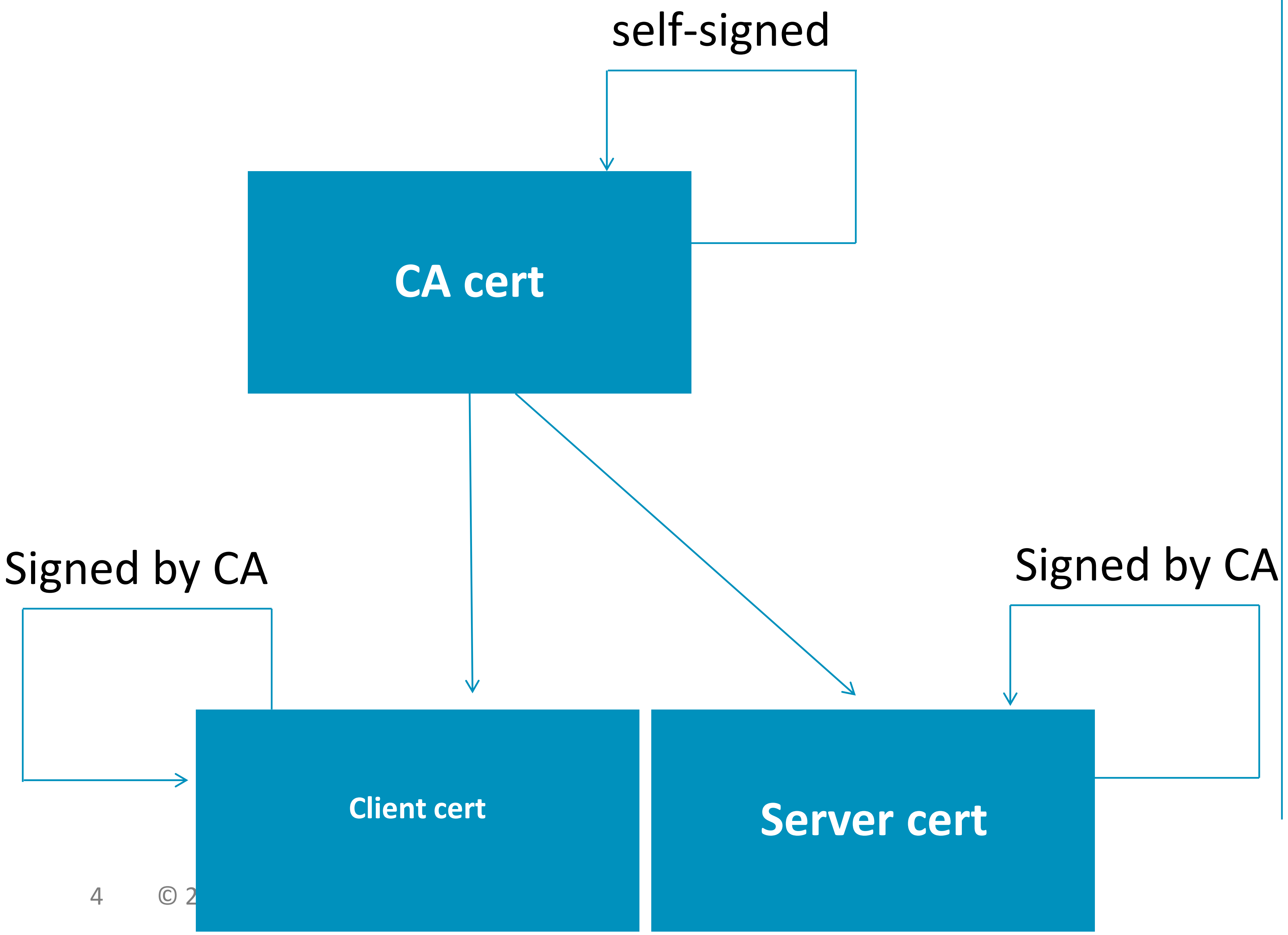
Public Key Infrastructure and certificate configuration

arm

Public Key Infrastructure

Various PKI deployments in existence

Structure of our PKI



The client has to store:

- Client certificate plus corresponding private key.
- CA certificate, which serves as the trust anchor.

The server has to store:

- Server certificate plus corresponding private key.

  (Some information for authenticating the client)

Generating certificates (using OpenSSL tools)

- When generating certificates you will be prompted to enter info.
- The CA cert will end up in the trust anchor store of the client.
- The Common Name used in the server cert needs to be resolvable via DNS UNLESS you use the server name indication extension.
- If the information in the Common Name does not match what is expected in the TLS handshake (based on configuration) then the exchange will (obviously) fail.

```
You are about to be asked to enter information that will be
incorporated into your certificate request.
What you are about to enter is what is called a Distinguished
Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:.
State or Province Name (full name) [Some-State]:.
Locality Name (eg, city) []:.
Organization Name (eg, company) [Internet Widgits Pty Ltd]:.
Organizational Unit Name (eg, section) []:.
Common Name (e.g. server FQDN or YOUR name) []:CA
Email Address []:.
```

Generating CA certificate

Listing supported curves

```
> openssl ecparam -list_curves
```

Self-signed CA Cert

```
> openssl ecparam -genkey -name secp256r1 -out ca.key
> openssl req -x509 -new -SHA256 -nodes -key ca.key -days 3650 -out ca.crt
```

Generating server certificate

Generate Server Private Key

```
> openssl ecparam -genkey -name secp256r1 -out server.key
```

Create CSR

```
> openssl req -new -SHA256 -key server.key -nodes -out server.csr
```

Print CSR:

```
> openssl req -in server.csr -noout -text
```

CA creates Server Cert

```
> openssl x509 -req -SHA256 -days 3650 -in server.csr -CA ca.crt
             -CAkey ca.key -CAcreateserial -out server.crt
```


arm

Generating client certificate

Generate Client Private Key

```
> openssl ecparam -genkey -name secp256r1 -out client.key
```

Create CSR

```
> openssl req -new -SHA256 -key client.key -nodes -out client.csr
```

CA creates Client Cert

```
> openssl x509 -req -SHA256 -days 3650 -in client.csr -CA ca.crt
               -CAkey ca.key -CAcreateserial -out client.crt
```

Operational PKI challenges worth mentioning

- Certificates contain an expiry date, which needs to be checked.
- Certificates may also get revoked.
- Certificates and trust anchors may need to be replaced.

- These topics are not covered in this webinar.

TLS protocol

arm

Public key crypto

- Two popular types of of asymmetric crypto systems emerged, namely RSA and Elliptic Curve Cryptography (ECC).
- The TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8 ciphersuite is recommended by many standards. It uses
  - Ephemeral Elliptic Curve Diffie-Hellman (ECDHE), and
  - The Elliptic Curve Digital Signature Algorithm (ECDSA).
- New to ECC?
  - Talk: "A gentle introduction to elliptic-curve cryptography" by Tanja Lange and Dan Bernstein.
  - Book: “Guide to Elliptic Curve Cryptography” by Vanstone, et al.

Recall: Key length

Preferred for IoT security

Symmetric	ECC	DH/DSA/RSA
80	163	1024
112	233	2048
128	283	3072
192	409	7680
256	571	15360

Two Phase Design of TLS

Phase 1 – "Handshaking Protocols"

TLS-PSK

- Used symmetric keys for authentication
- Covered in 1st webinar

TLS-ECDHE-ECDSA

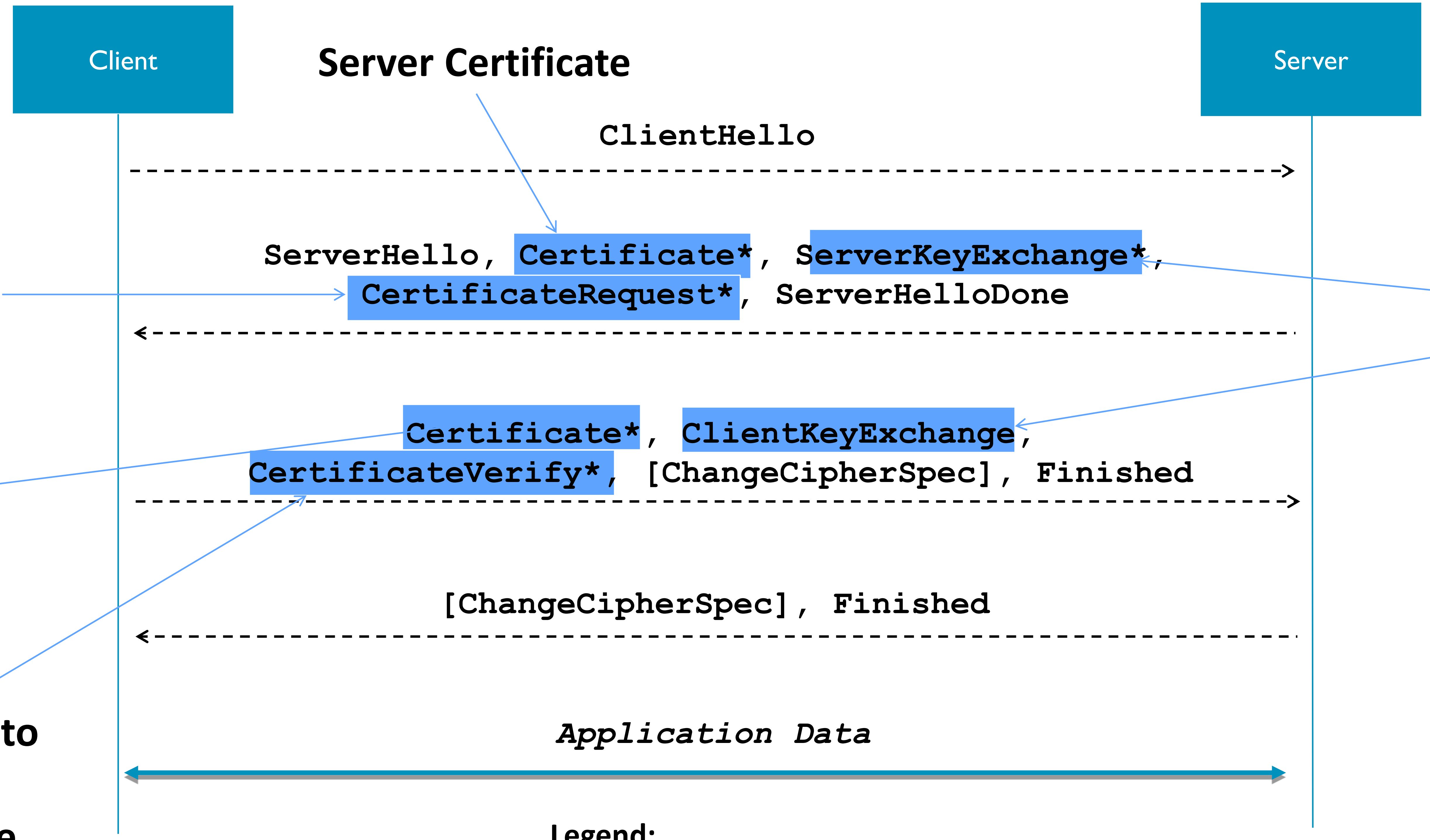
- Uses public key cryptography and (in our case) certificates for authentication.
- Covered in today's webinar.

Phase 2 – "Record Protocol"

AES-128-CCM-8 to protect HTTP

arm

Full TLS handshake



Certificate requested by the server and provided by the client (optional msgs)

Used when client provided certificate to demonstrate possession of private key.

Used by some ciphersuites to convey information to generate the premaster secret.

May need to be signed by the server.

Legend:
*: optional message
[]: Not a handshake message.

Note: Most Web deployments use server-to-client authentication only.

ECDHE-ECDSA Exchange

Client

Server

1

- Generate EC Diffie-Hellman key pair
- Ephemeral ECDH public key is put in ServerKeyExchange message.
- Sign ServerKeyExchange message with long term private key.

2

- Generate EC Diffie-Hellman key pair
- Place ephemeral ECDH public key in the ClientKeyExchange message
- CertificateVerify demonstrate possession of the long-term private key corresponding to the public key in the client's Certificate message.

3

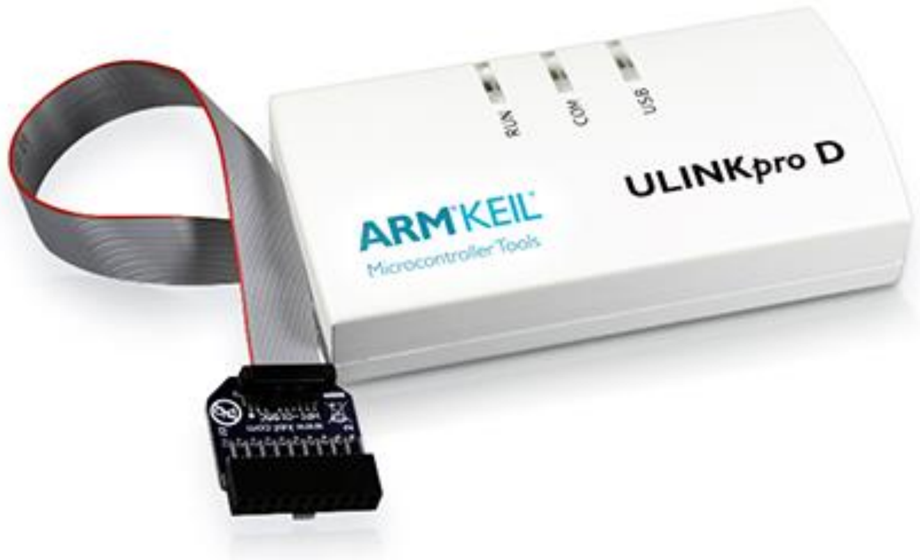
- ECDHE derived key becomes pre_master_secret, which is then used in master_secret calculation

arm

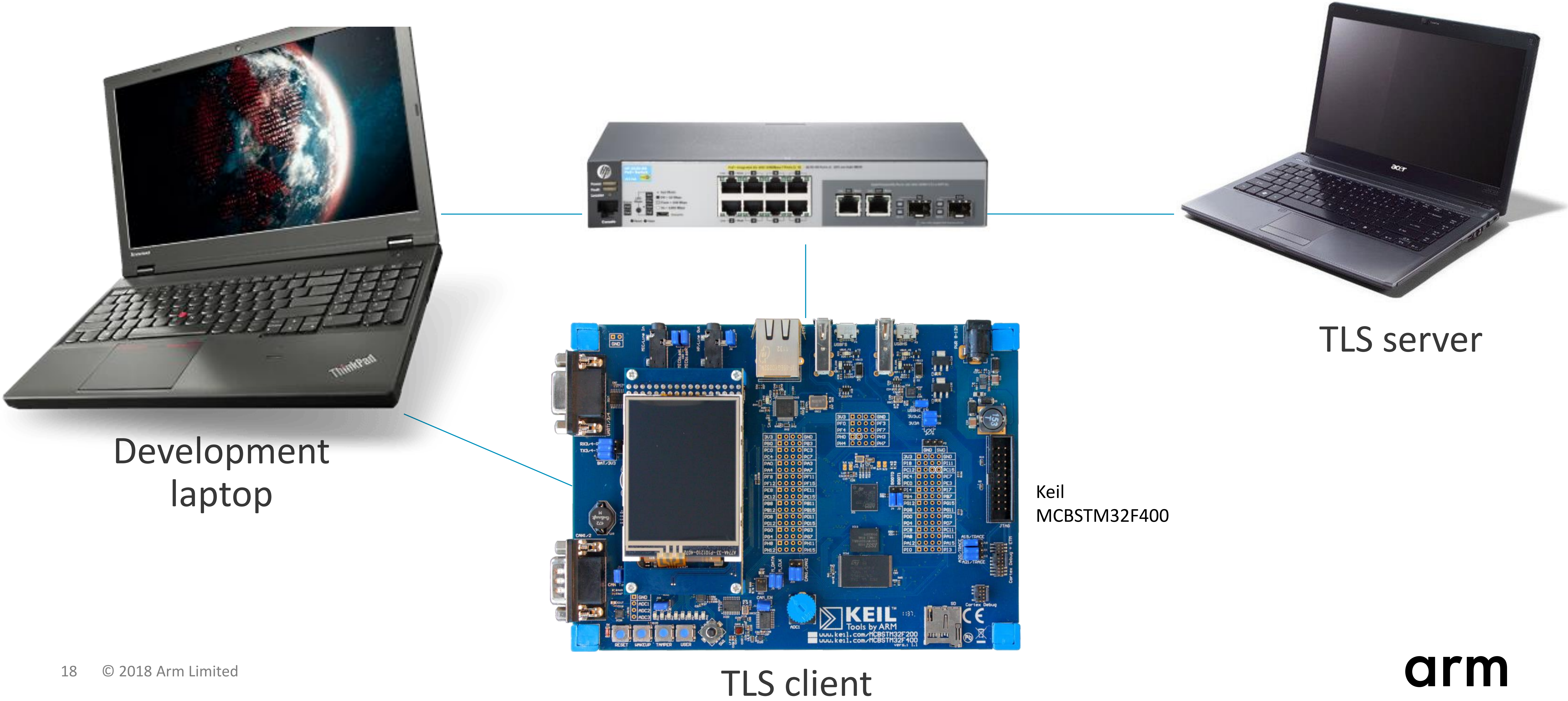
Hands-on

Platform

- For this hands-on session we are using the Keil MCBSTM32F400 Evaluation Board, which uses the STM32F407IG MCU.
- This MCU uses an Arm Cortex M4 processor. More information can be found in this datasheet.
- Keil RTX5 serves as the real-time OS. Mbed TLS and networking middleware.



Demo setup


Development laptop
TLS server
Keil MCBSTM32F400
TLS client

config.h settings for TLS-ECDHE-ECDSA

- According to RFC 7925 we use
  - TLS 1.2: **MBEDTLS_SSL_PROTO_TLS1_2**
  - TLS-ECDHE-ECDSA-WITH-AES-128-CCM-8 as a ciphersuite, which requires **MBEDTLS_KEY_EXCHANGE_ECDHE_ECDSA_ENABLED**
  - AES, CCM, and SHA256, (**MBEDTLS_AES_C, MBEDTLS_CCM_C, MBEDTLS_SHA256_C**)
  - ECC support: **MBEDTLS_ECDH_C, MBEDTLS_ECDSA_C MBEDTLS_ECP_C, MBEDTLS_BIGNUM_C**
  - ASN.1 and certificate parsing support
  - NIST Curve P256r1 (**MBEDTLS_ECP_DP_SECP256R1_ENABLED**)
  - Server Name Indication (SNI) extension (**MBEDTLS_SSL_SERVER_NAME_INDICATION**)
- We enable optimizations (**MBEDTLS_ECP_NIST_OPTIM**) and deterministic ECDSA (RFC 6979) with **MBEDTLS_ECDSA_DETERMINISTIC**

Mbed TLS client application code

1. Initialize TLS session data
2. Initialize the RNG
3. Establish TCP connection
4. Configure TLS
5. Run TLS handshake protocol
6. Exchange application data
7. Tear down communication and free state

Parse CA certificate, client certificate and private key of client.

- Load CA certificate
- Load client certificate and private key
- Configure SNI
- Configure curve(s)

Verify the server certificate

arm

Server-side command line

Demands mutual authentication

Server Certificate

Server Private Key

```
> programs/ssl/ssl_server2 auth_mode=required crt_file=server.crt key_file=server.key
ca_file=ca.crt
```

CA certificate

Parameters:

- **auth_mode** determines the behaviour of a missing client certificate or a failed client authentication. Allowed values are “none”, “optional” and “required”.
- **cert_file** indicates the file that contains the server certificate.
- **key_file** indicates the file that contains the private key of the server.
- **ca_file** indicates the file that contains the CA certificate.

arm

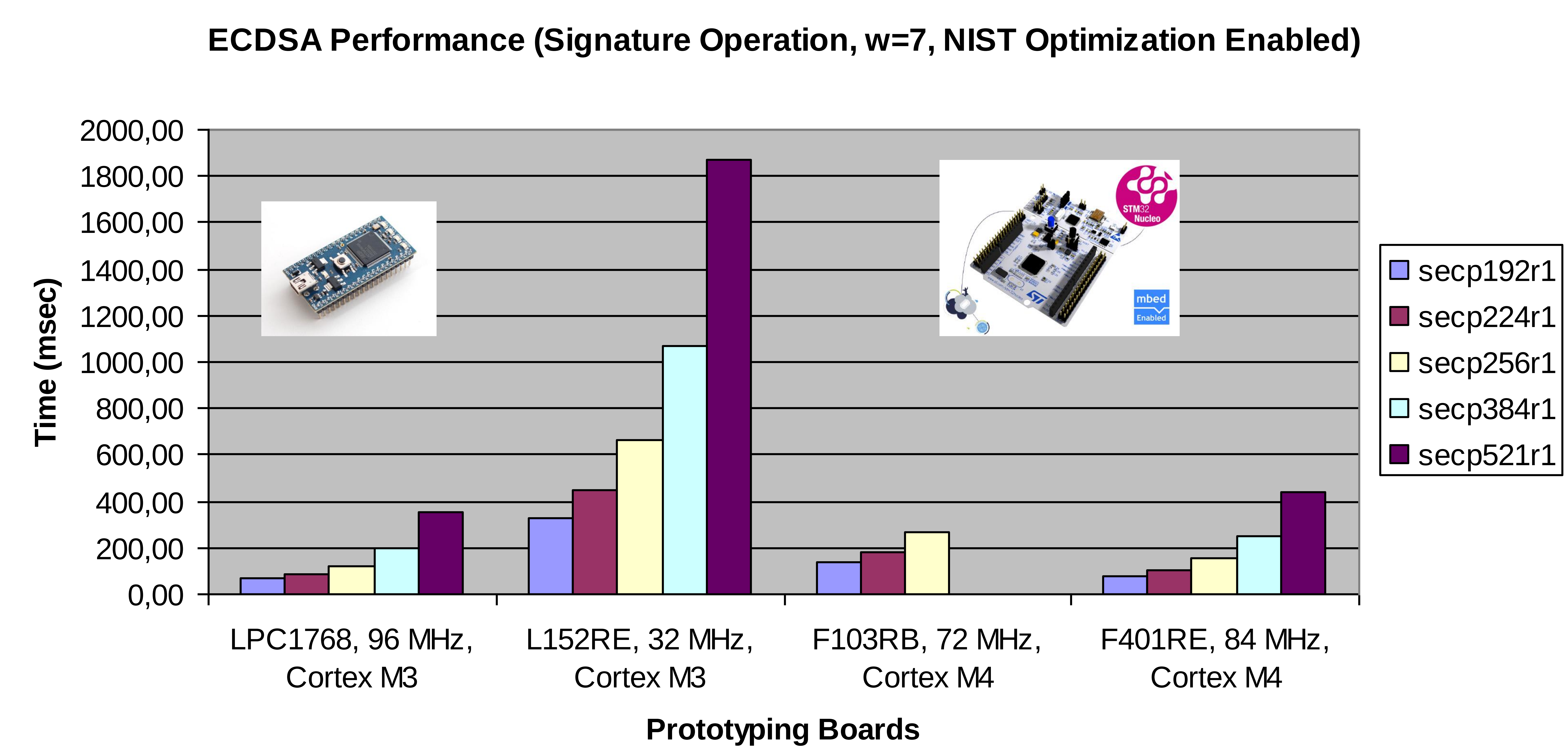
The cost of public key crypto

Handshake message size

Client	Server	Size
ClientHello		121 bytes
	ServerHello	87 bytes
	Certificate	557 bytes
	Server Key Exchange	215 bytes
	Certificate Request	78 bytes
	Server Hello Done	4 bytes
Certificate		570 bytes
Client Key Exchange		138 bytes
Certificate Verify		80 bytes
Change Cipher Spec Protocol		1 byte
TLS Finished		40 bytes
	Change Cipher Spec	1 byte
	TLS Finished	40 bytes

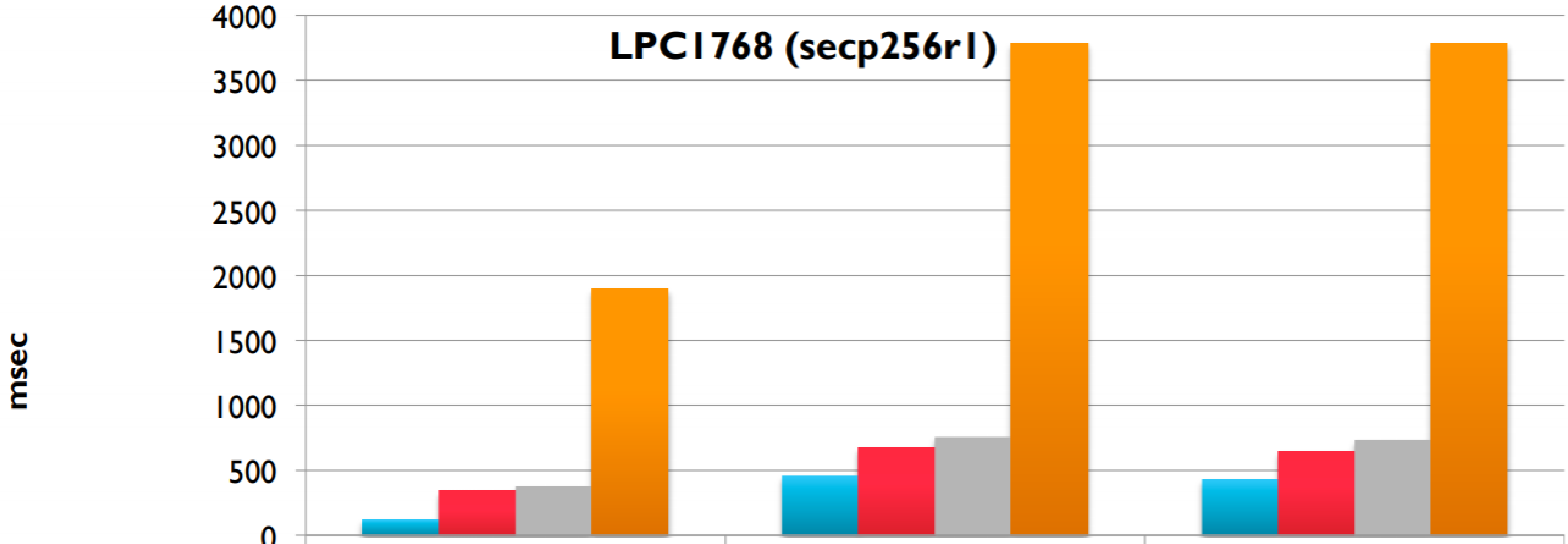
- Example assumes a ECC-based ciphersuite with a 256 bit curve.
- Only a single certificate is exchanged in the Certificate message.
- (But mutual authentication is used, i.e., client and server exchange certificates.)
- Result: 1932 bytes

Performance comparison: Signature generation



Performance data from a contribution to the NIST lightweight crypto workshop 2015.

Performance optimization impact



	ECDSA-Sign	ECDSA-Verify	ECDHE
w=6, FP, NIST	122	458	431
w=6, no FP, NIST	340	677	644
w=2, no FP, NIST	378	759	734
w=2, no FP, no NIST	1893	3788	3781

Using ~50 % more RAM increases the performance by a factor 8 or more.

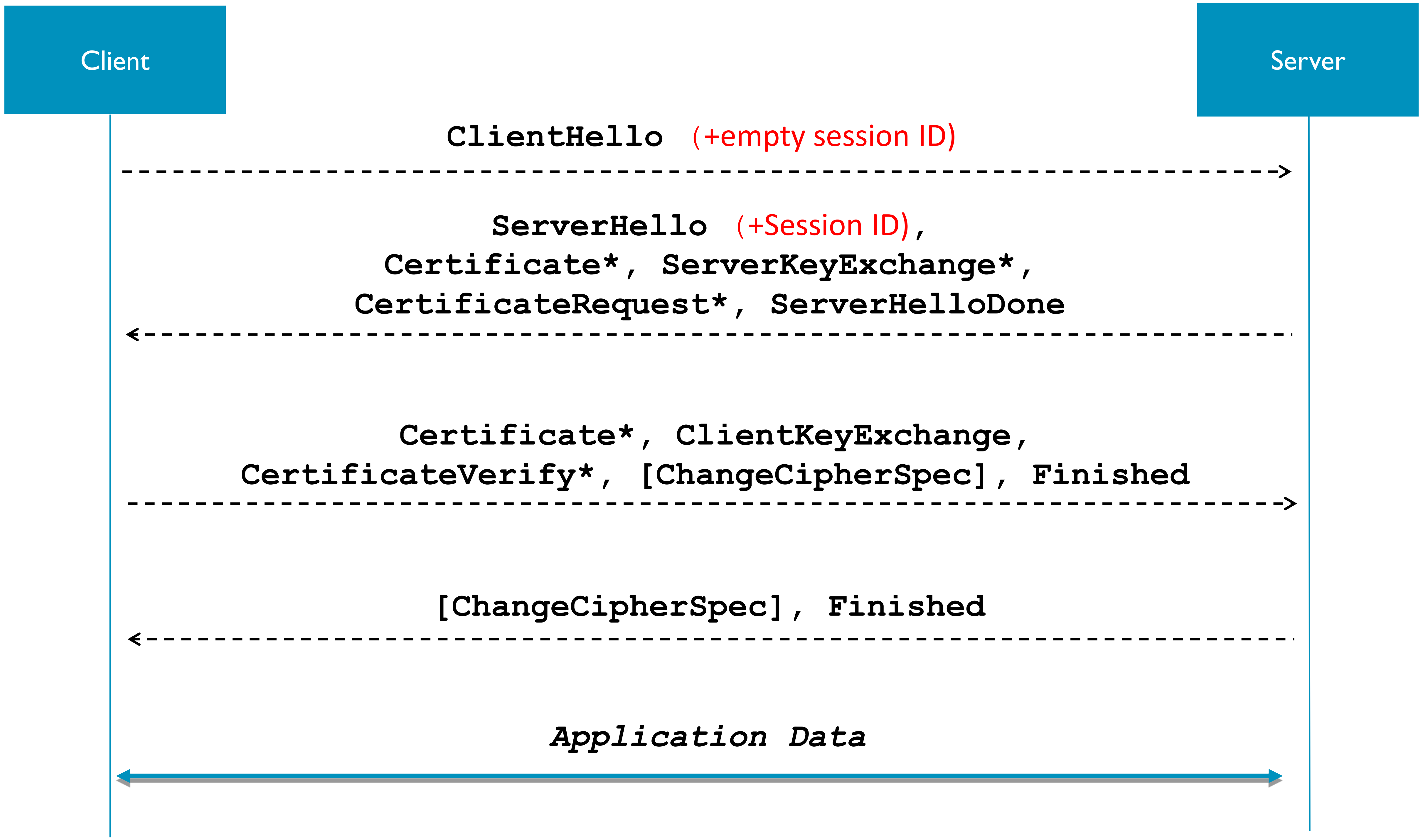
Performance data from a contribution to the NIST lightweight crypto workshop 2015.

arm

Improving performance with TLS extensions

arm

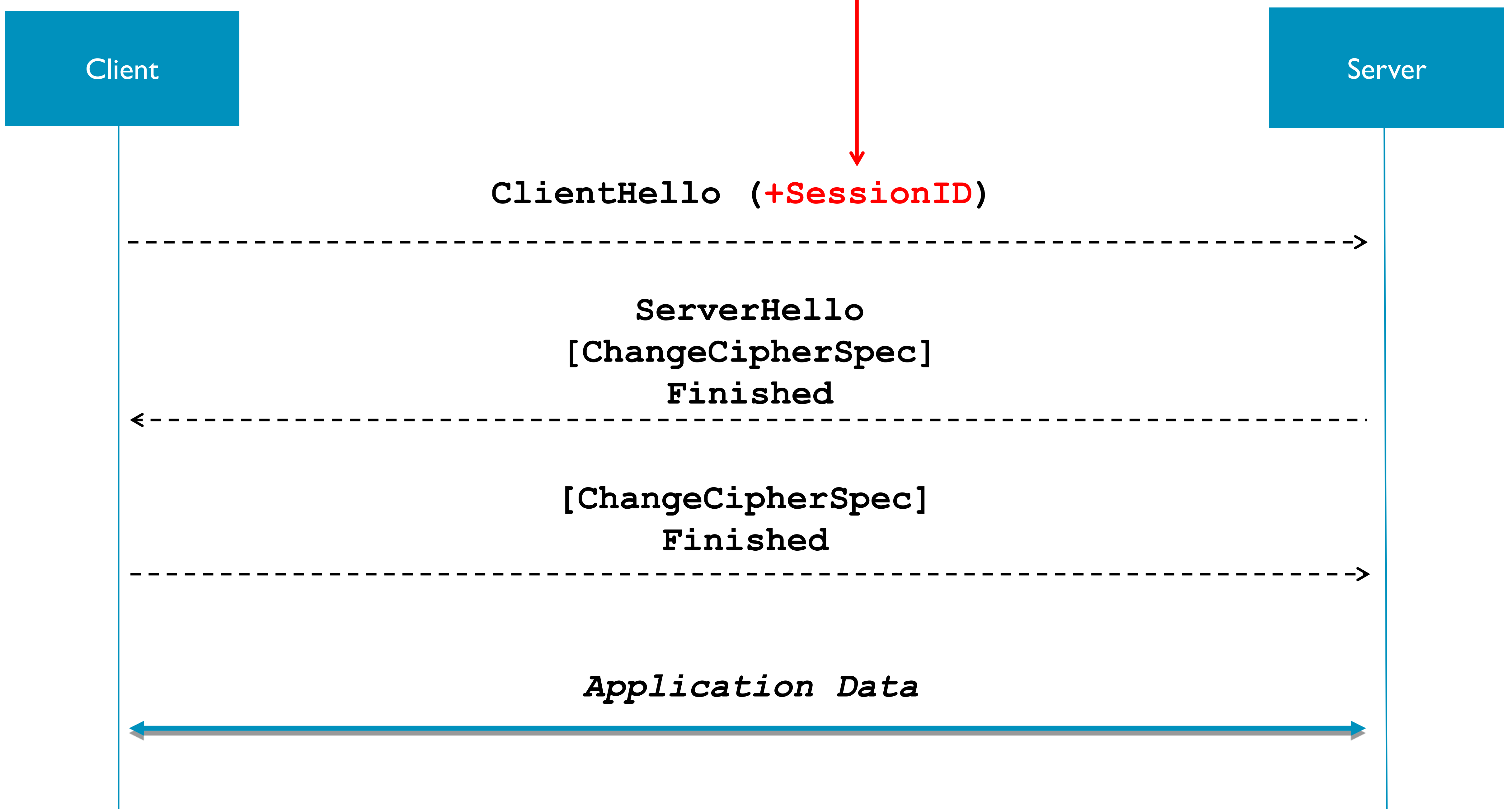
Session resumption exchange
First phase



A session ID is allocated by the server.

Session resumption exchange
Second phase

Security state established with full exchange and “indexed” with session identifier.

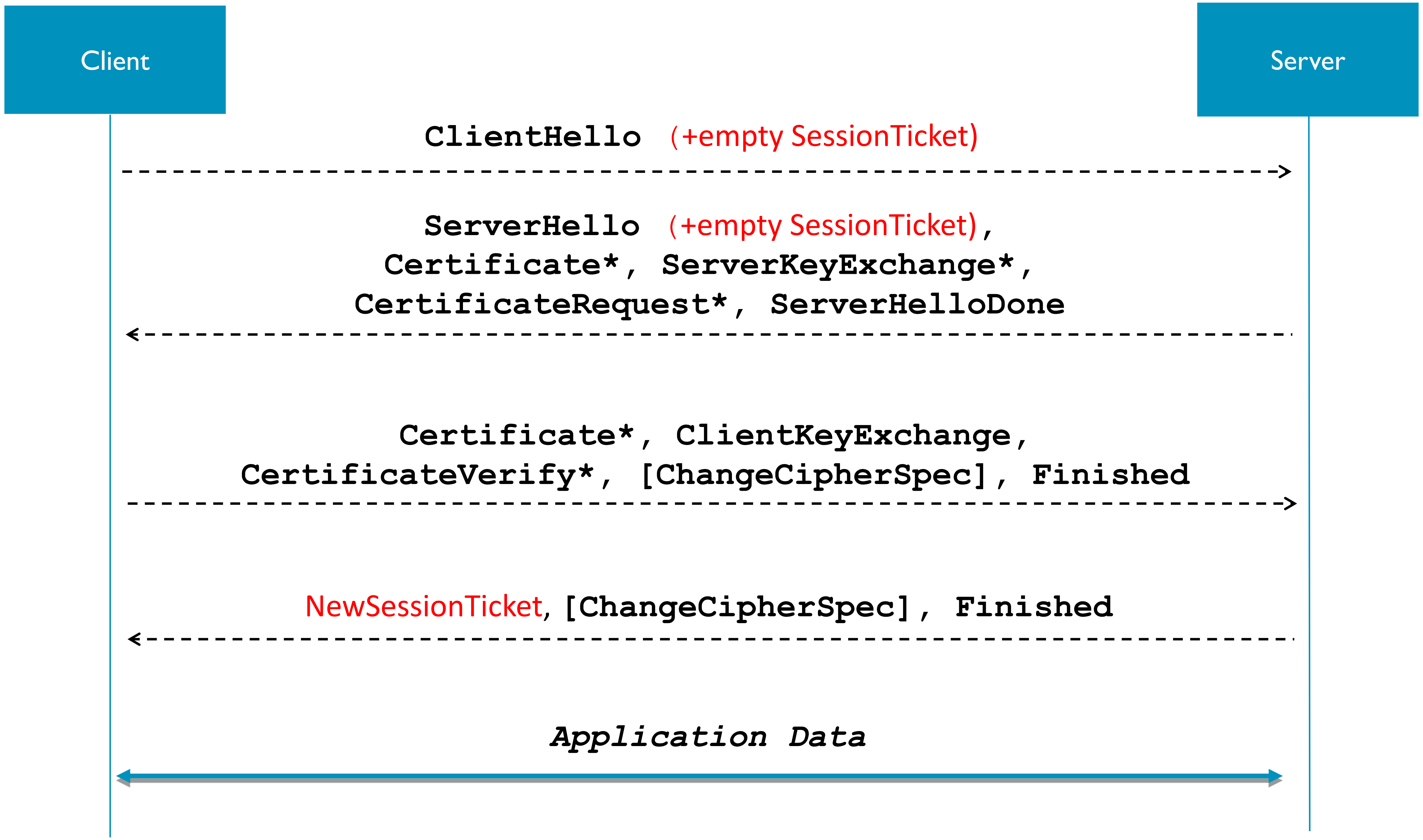


Benefits:

- Few message exchanged
- Less bandwidth consumed
- Lower computational overhead

arm

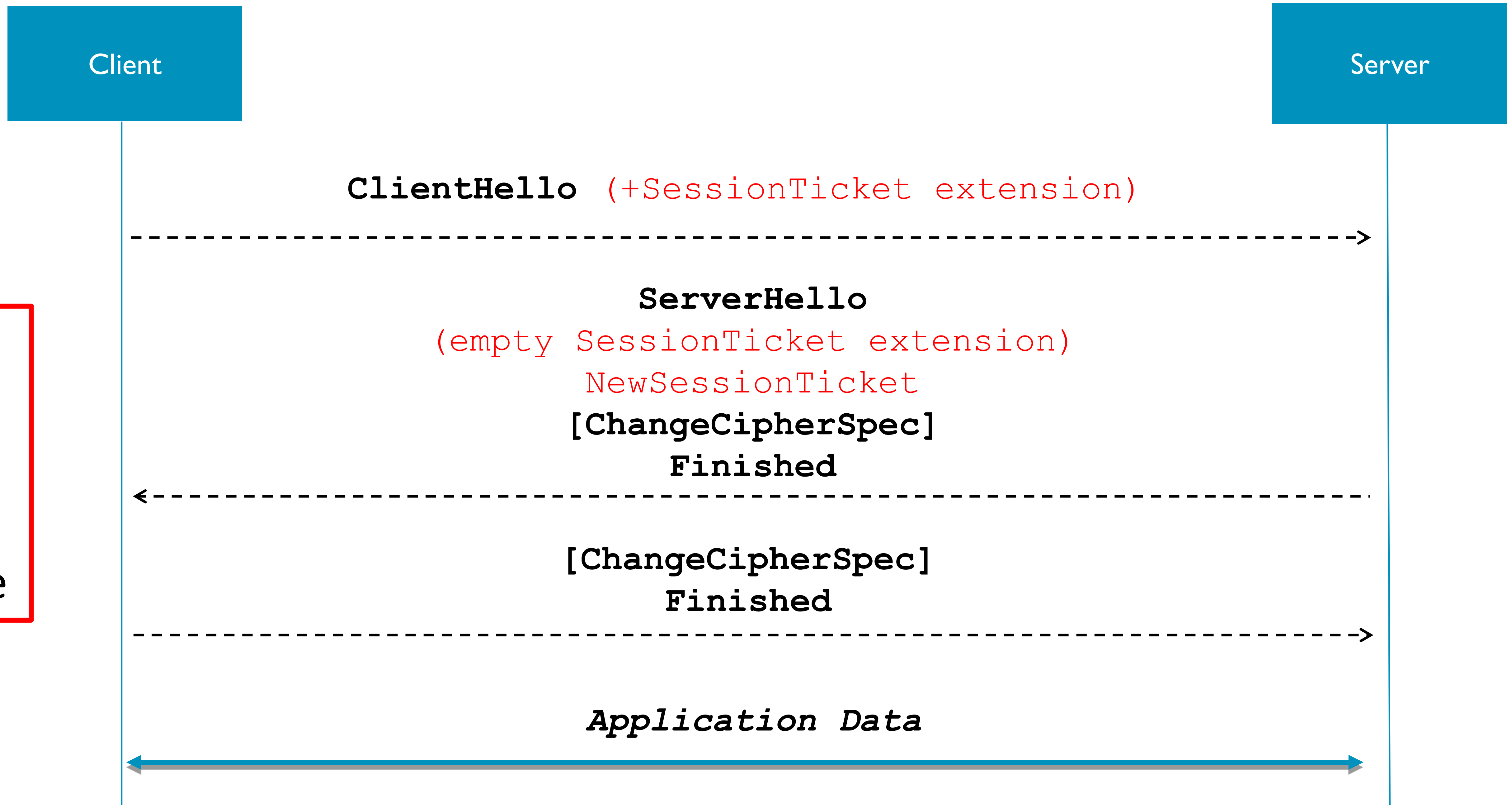
Session resumption without server-side state
First phase



Negotiating the SessionTicket extension and issuing a ticket with the NewSessionTicket message.

The client caches the ticket along with the session information.

Session resumption without server-side state
Second phase



Benefits:

- Same as session resumption.
- Increased scalability due to distributed session state storage

Ticket stores the session state including the master_secret, client authentication type, client identity, etc.

Specified in RFC 5077.

TLS cached info

Client	Server	Size
ClientHello		121 bytes
	ServerHello	87 bytes
	~~Certificate~~	~~557 bytes~~
	Server Key Exchange	215 bytes
	~~Certificate Request~~	~~78 bytes~~
	Server Hello Done	4 bytes
Certificate		570 bytes
Client Key Exchange		138 bytes
Certificate Verify		80 bytes
Change Cipher Spec Protocol		1 byte
TLS Finished		40 bytes
	Change Cipher Spec	1 byte
	TLS Finished	40 bytes

TLS exchanges lots of fairly static information.

- Certificates
- List of acceptable certification authorities

Idea: Cache information on the client and avoid sending it unless it changes.

TLS Cached Info specification is published in RFC 7924.

Allows to cache server certificate and certificate request.

Client-side certificate can be omitted by sending a Certificate URI extension instead, which is specified in RFC 6066.

arm

Further TLS extensions for performance improvement

Raw public key (RPK) extension (RFC 7250) re-uses the existing TLS Certificate message to convey the raw public key encoded in the SubjectPublicKeyInfo structure.

Maximum Fragment Length (MFL) extension (RFC 6066) allows the client to indicate to the server how much maximum memory buffers it uses for incoming messages.

Trusted CA Indication extension (RFC 6066) allows clients to indicate what trust anchor they support.

Note: Re-using TLS code at multiple layers helps to lower the overall code requirements.

Hands-on
(Session Resumption)

arm

config.h settings for session resumption

- No additions needed for plain session resumption.
- Only one parameter for RFC 5077 session resumption without server-side state:
  MBEDTLS_SSL_SESSION_TICKETS

Mbed TLS client application code

Initialize session resumption state

Configure session resumption without server-side state

Initial exchange

1. Initialize TLS session data
2. Initialize the RNG
3. Establish TCP connection
4. Configure TLS
5. Run full TLS handshake protocol
6. Exchange application data
7. Encounter error

Subsequent exchanges

8. Set session state
9. Establish TCP connection
10. Run TLS session resumption
11. Exchange application data
12. Tear down communication and free state

Free session resumption state

arm

Conclusion

arm

Summary

- PSK-based ciphersuites provide great performance.
- Certificate-based ciphersuites provide an alternative where the private key is not shared.
- Public key crypto is more challenging to performance.
- This performance impact can partially be mitigated using TLS extensions, such as session resumption.

arm

