

A man with a beard and a small robot on his head is looking at a tablet. The background is a blurred office setting. The word 'arm' is written in white lowercase letters on the left side of the image.

arm

Securing IoT applications with Mbed TLS

Hannes Tschofenig

Agenda

Theory

- Threats
- Security services
- TLS Protocol
- Performance
- DTLS

Hands-on with Arm Keil MDK

- Pre-shared secret-based authentication
(covered in webinar #1)
- Public-key based authentication
(covered in webinar #2)

Speaker's bio – Hannes Tschofenig



- Employed by Arm Ltd working mostly on IoT (security) standards.
- Previous employers include European Data Protection Supervisor, Nokia Siemens Networks/Nokia and Siemens.
- Contributed to different standardization bodies, such as the IETF (see [IETF data tracker page](#)) and OMA, and EEMBC.
- Contact email address: Hannes.Tschofenig@arm.com

Threats

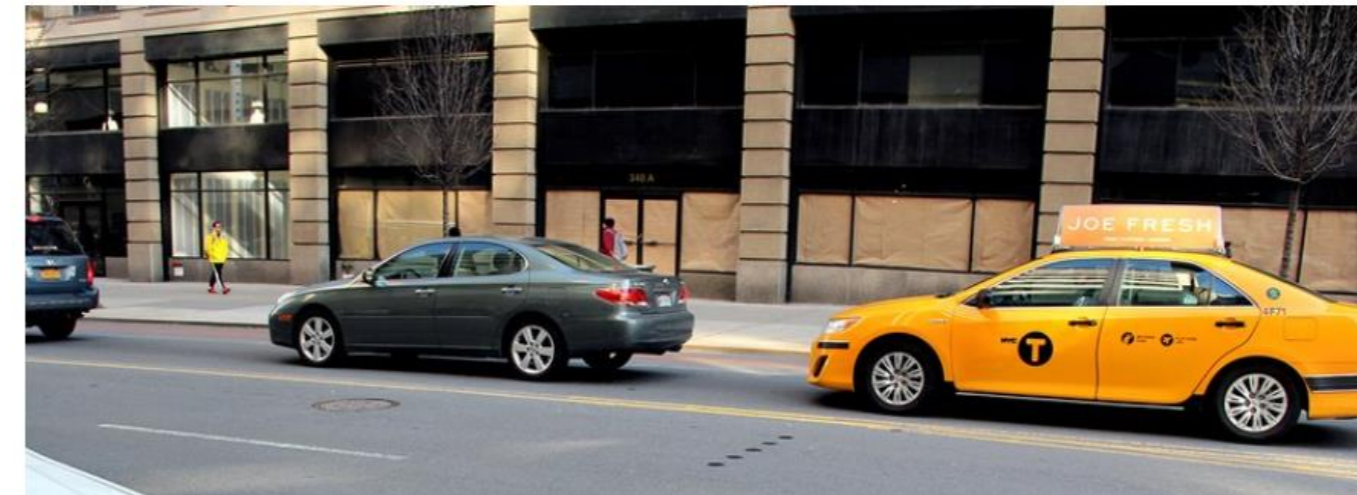
Examples of breaches due to missing communication security

Car Hack



Traffic Lights Hack

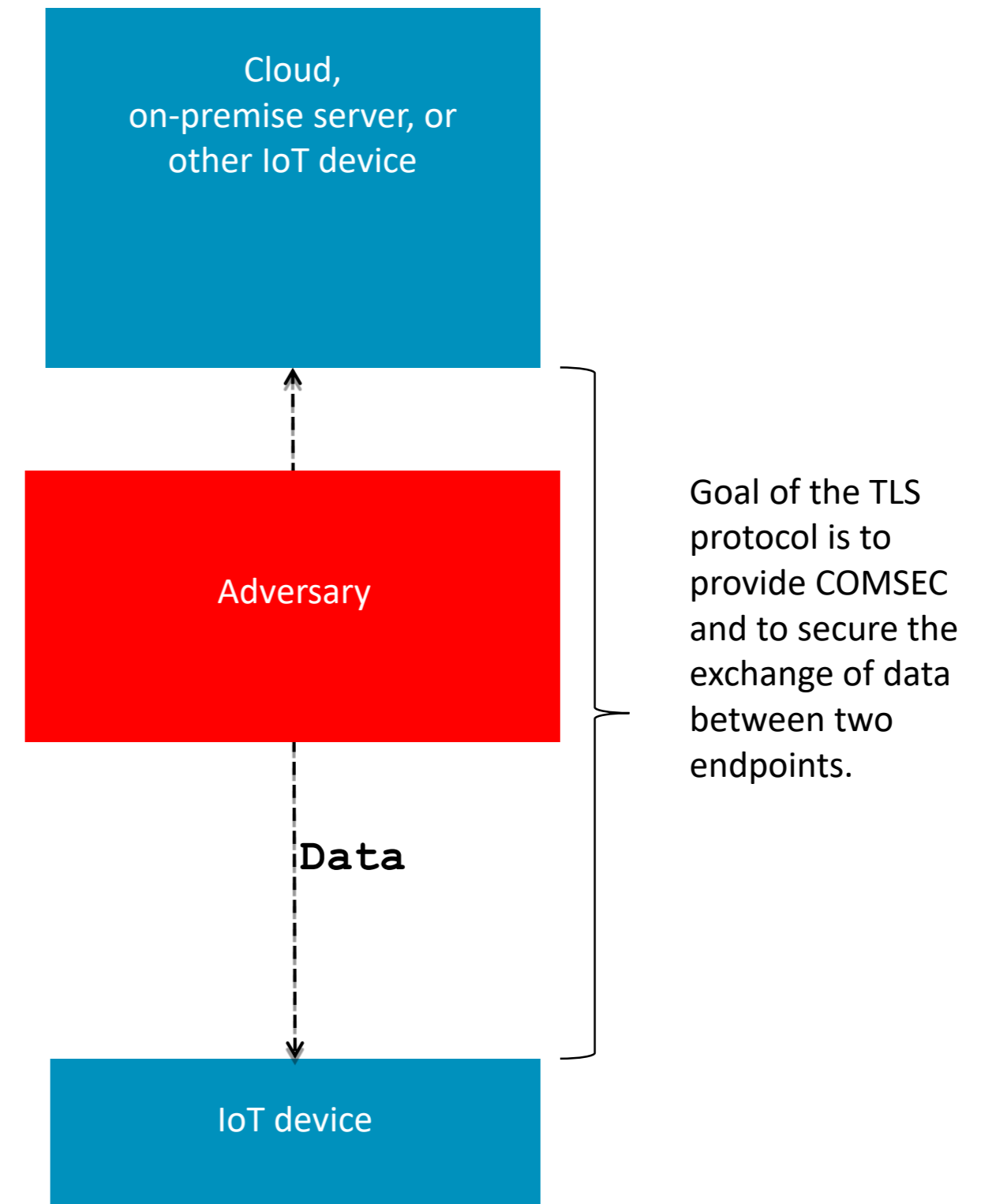
HACKERS CAN MESS WITH TRAFFIC LIGHTS TO JAM ROADS AND REROUTE CARS



Security services

What are we trying to protect?

- Internet threat model is documented in [RFC 3552](#)
 - Attacker has nearly complete control of the communications channel.
 - End systems engaged in a protocol exchange have not themselves been compromised.
- Focus of the IETF standardization activities has been on communication security (COMSEC) and providing security services, such as confidentiality, integrity and authentication.
- Lack of COMSEC is one of the top 5 problems with IoT security.



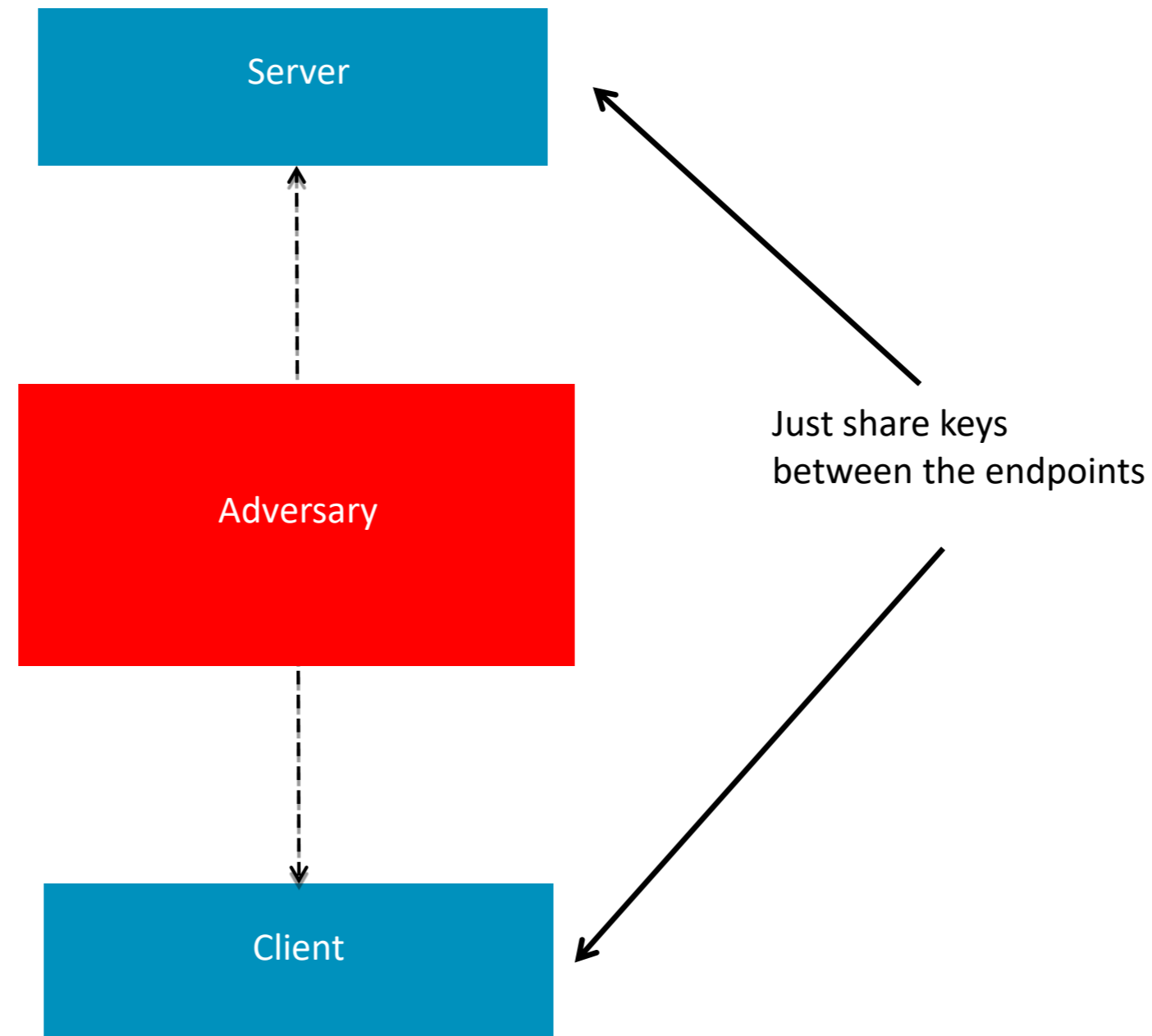
TLS Protocol

History

- TLS has been around for some time
 - **SSL 1.0, 2.0 and 3.0** was developed by Netscape. Version 1.0 was never publicly released because of serious security flaws in the protocol; version 2.0, released in February 1995, "contained a number of security flaws which ultimately led to the design of SSL version 3.0". SSL version 3.0, released in 1996.
 - **TLS 1.0** was published as [RFC 2246](#) in January 1999.
 - **TLS 1.1** was published as [RFC 4346](#) in April 2006.
 - **TLS 1.2** was published as [RFC 5246](#) in August 2008.
- Beside the work on the major TLS versions DTLS 1.1 was published as [RFC 4347](#) in April 2006 and DTLS 1.2 was published as [RFC 6347](#) in January 2012.
- Equally important is the work on numerous extensions to TLS/DTLS, which happened throughout the years.
- The work on **TLS 1.3** started late 2013 and is still ongoing in the [IETF TLS working group](#).
- TLS is widely deployed and has been excessively analyzed by researchers.

Naïve approach to COMSEC

- Ideally, we just need keys, algorithms and parameters shared between the endpoints to protect application data.



Protecting application data

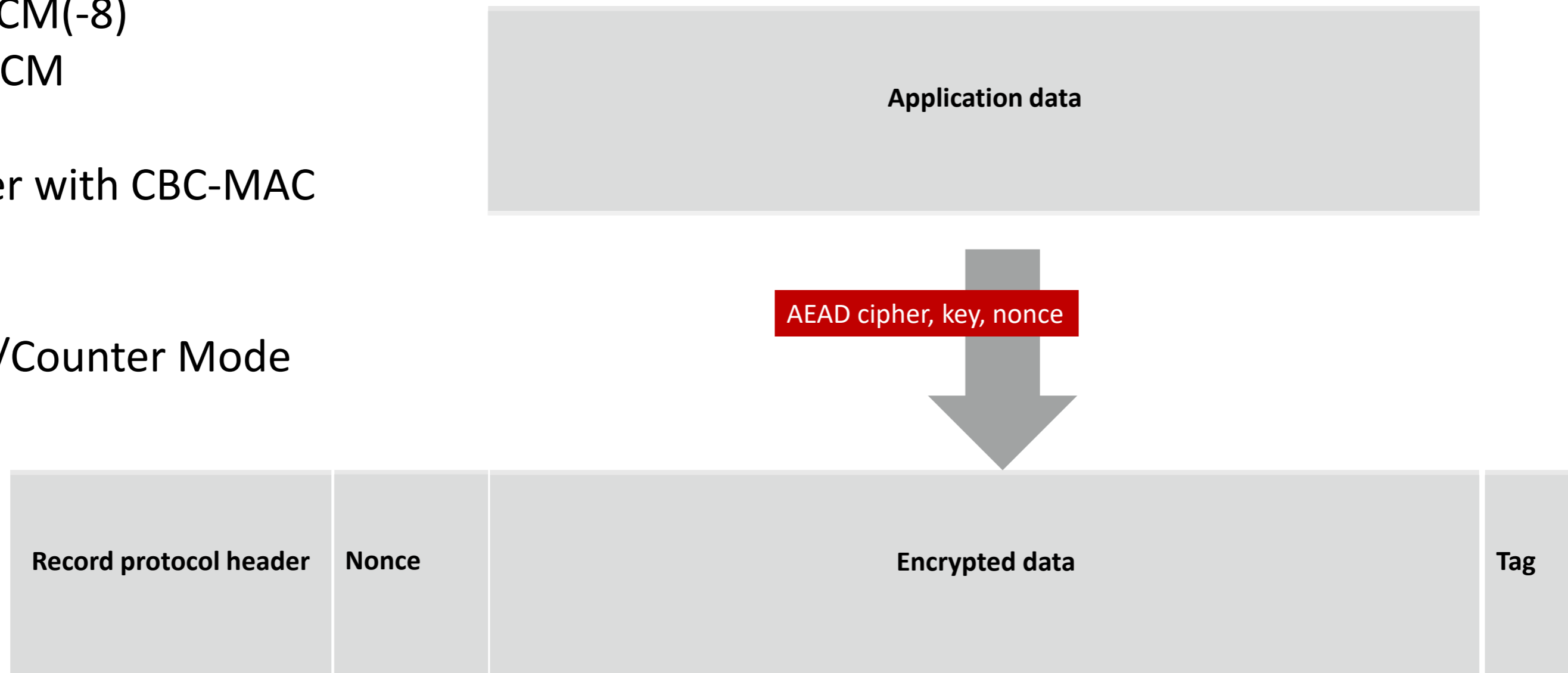
Using Authenticated Encryption with Associated Data (AEAD)

Examples of AEAD ciphers are:

- AES-128-CCM(-8)
- AES-256-GCM

CCM –Counter with CBC-MAC Mode (CCM)

GCM - Galois/Counter Mode



Tag = Authentication Tag = Integrity Check Value (ICV)

Practical challenges

- There are additional requirements:
 - Algorithms, parameters and features need to be negotiated. With dynamic negotiation downgrade protection is needed.
 - Endpoints need to be authenticated.
 - Key management needs to be addressed.
 - Re-keying is needed from time-to-time.
 - Different types of credentials are often used (e.g., public key crypto, strong password-based credentials).
 - Exchange needs to be protected against Denial of Service.
 - Protection against traffic analysis and other privacy-related threats may be needed.
- In a nutshell, there is more to the protection of application data in flight than the use of keys with an algorithm.
- This is where an authentication and key agreement protocol comes into the picture.

Design Idea: Two Phase Protocol Exchange

Phase 1

- Endpoint authentication that may involve asymmetric cryptography
- Multi-roundtrip handshake with algorithm and feature negotiation
- Added DoS protection capabilities
- End result: Symmetric keys for use with negotiated ciphers.

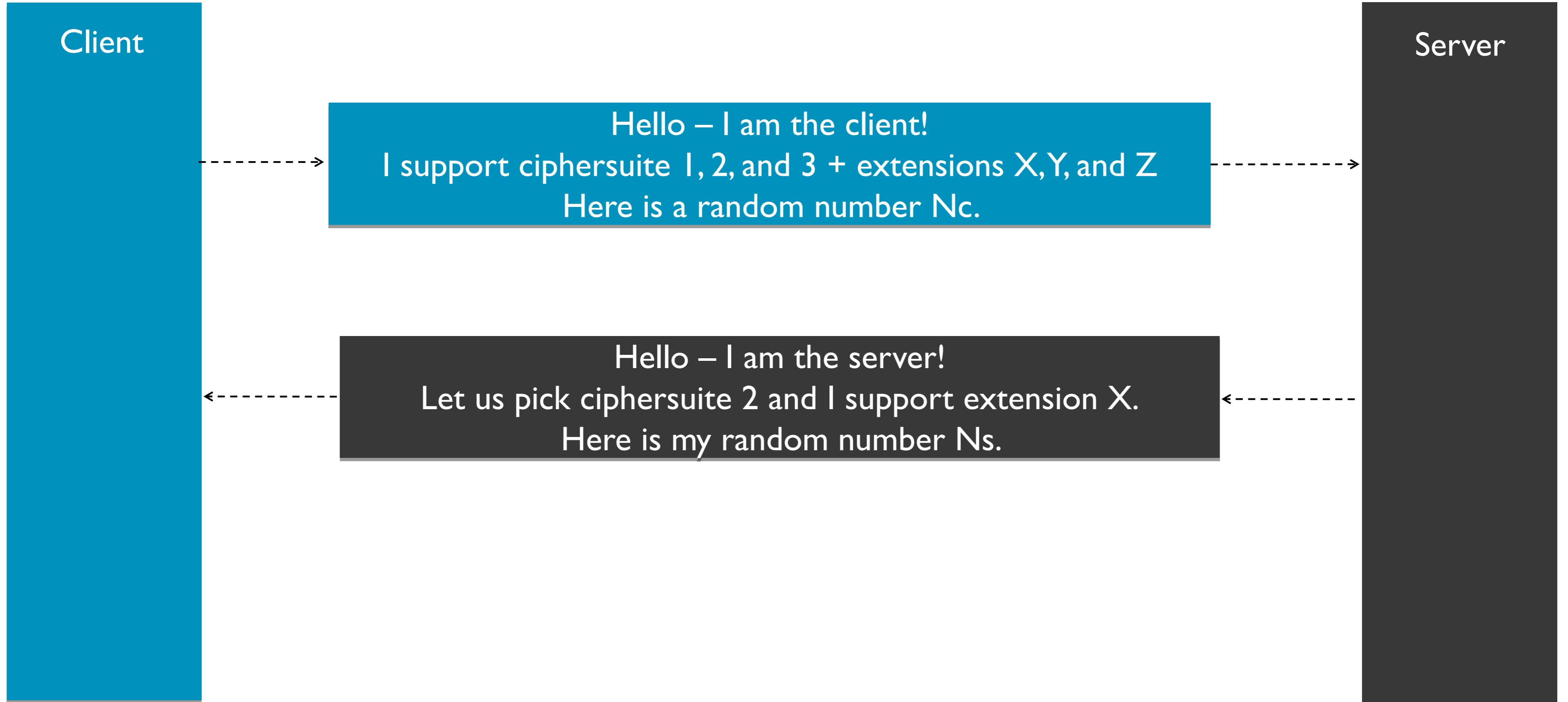
The TLS spec calls this the “Handshaking Protocols”.

Phase 2

- Symmetric crypto to offer data-origin authentication, integrity and confidentiality protection
- Great performance with low on-the-wire overhead.

The TLS spec calls this the “Record Protocol”.

Abstract TLS handshake exchange (Part 1)




The ciphersuite concept

- Up till TLS 1.2 a ciphersuite is a combination of
 - **Authentication and key exchange algorithm** (e.g., PSK)
 - **Cipher and key length** (e.g., Advanced Encryption Standard (AES) with 128 bit keys [[AES](#)])
 - **Mode of operation** (e.g., Counter with Cipher Block Chaining - Message Authentication Code (CBC-MAC) Mode (CCM) for AES) [[RFC3610](#)]
 - **Hash algorithm for integrity protection**, such as the Secure Hash Algorithm (SHA) in combination with Keyed-Hashing for Message Authentication (HMAC) (see [[RFC2104](#)] and [[RFC4634](#)])
 - **Hash algorithm for use with the pseudorandom function** (e.g., HMAC with the SHA-256)
 - Misc information (e.g., length of authentication tags)
 - Information whether the ciphersuite is suitable for DTLS or only for TLS.
- Examples are TLS_PSK_WITH_AES_128_CCM_8 and TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8.
- The full list of standardized ciphersuites can be found [here](#).

TLS/DTLS extensions

- **The TLS/DTLS protocol is an authentication framework rather than a single authentication protocol.**
- Various extensions have been standardized over time to improve functionality and performance.
- The TLS/DTLS 1.2 Profiles for the Internet of Things ([RFC 7925](#)) offers guidance for the IoT sector.
- RFC 7925 covers the following profiles and explains what extensions are useful for
 - PSK-based ciphersuite
 - Raw public key-based ciphersuite
 - Certificate-based ciphersuite.

Randomness

 [About Debian](#) [Getting Debian](#) [Support](#) [Developers' Corner](#)

debian / [security information](#) / 2008 / security information -- dsa-1571-1 openssl

Debian Security Advisory

DSA-1571-1 openssl -- predictable random number generator

Date Reported:
13 May 2008

Affected Packages:
[openssl](#)

Vulnerable:
Yes



RSA warns developers not to use RSA products

In today's news of the weird, RSA (a division of EMC) has recommended that developers desist from using the (allegedly) 'backdoored' Dual_EC_DRBG random number generator -- which happens to be the *default* in RSA's BSafe cryptographic toolkit. Youch.

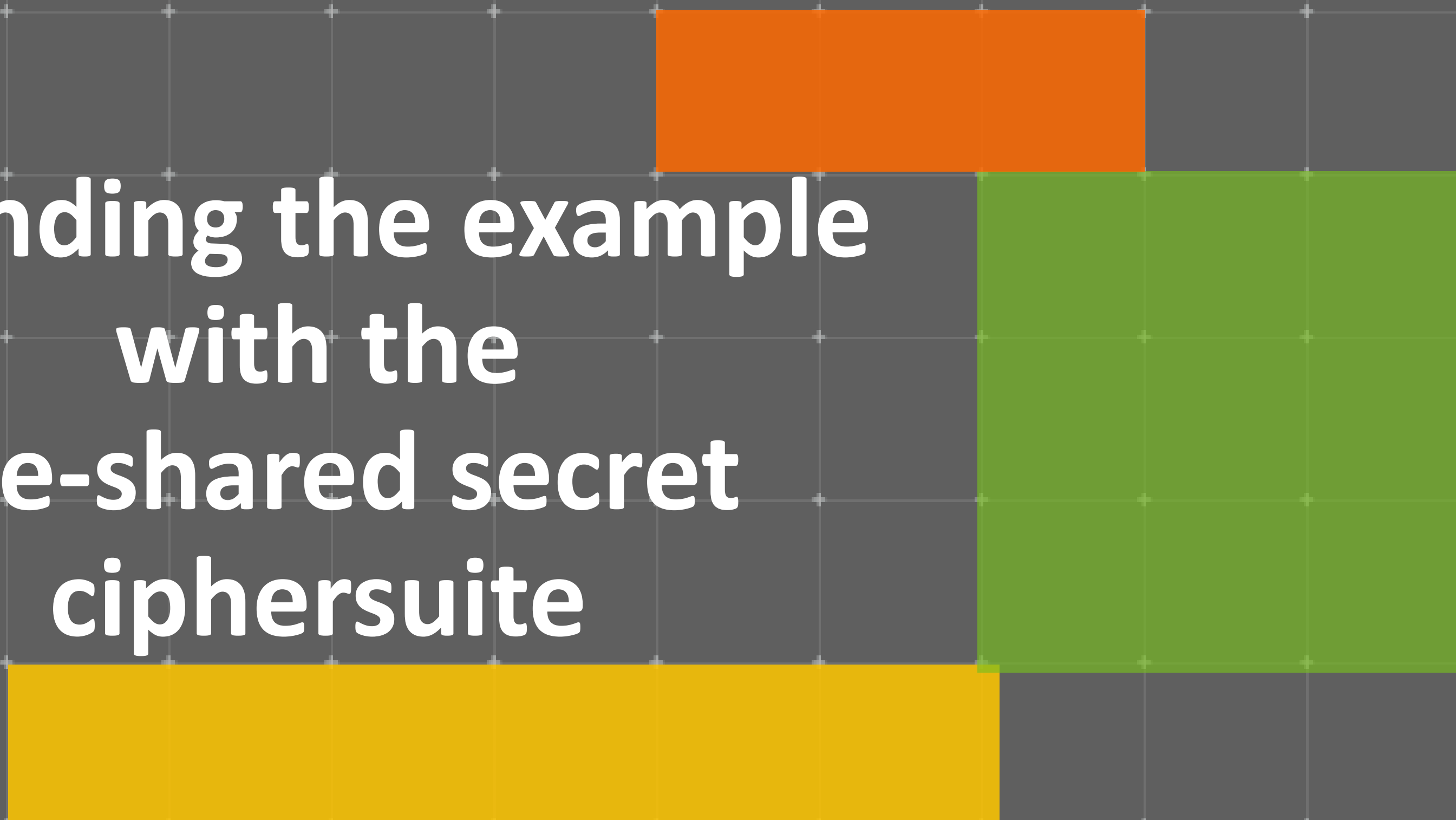


Widespread Weak Keys in Network Devices

We performed a large-scale study of RSA and DSA cryptographic keys in use on the Internet and discovered that significant numbers of keys are insecure due to insufficient randomness. These keys are being used to secure TLS (HTTPS) and SSH connections for hundreds of thousands of hosts.

Randomness: Why do we care?

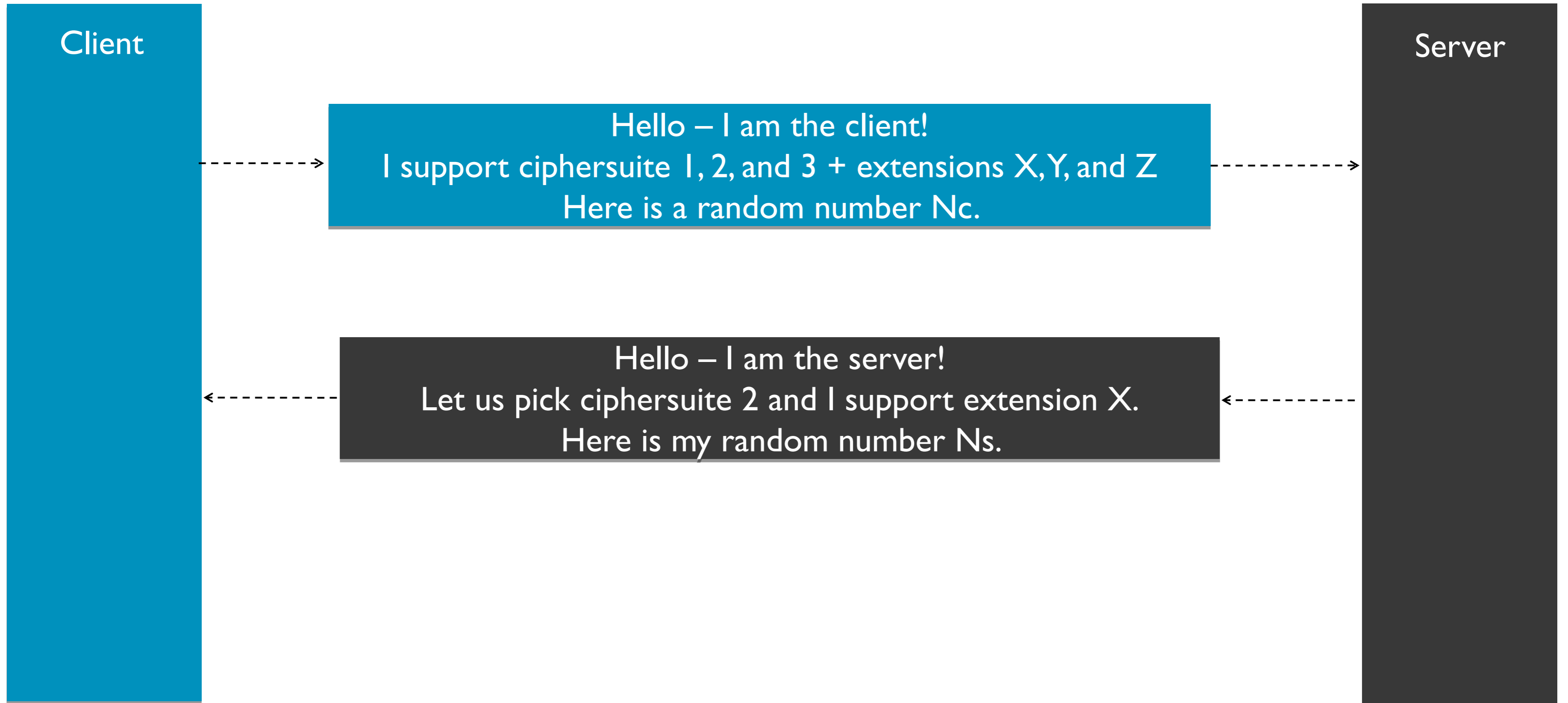
- Cryptographic protocols depend on good entropy source.
- High entropy \approx true random data
 - true random data = the perfect 50/50 coin-flip
- Examples:
 - Public/Private key pairs
 - Digital signatures (based on El Gamal)
 - Diffie-Hellman exchanges
 - Unique AES keys
 - Unique IVs
 - Nonces
- TLS for use in your project/product requires an entropy source.

The slide features three overlapping colored rectangles: an orange rectangle at the top right, a green rectangle in the middle right, and a yellow rectangle at the bottom left. The text is centered over these shapes.

Extending the example with the pre-shared secret ciphersuite

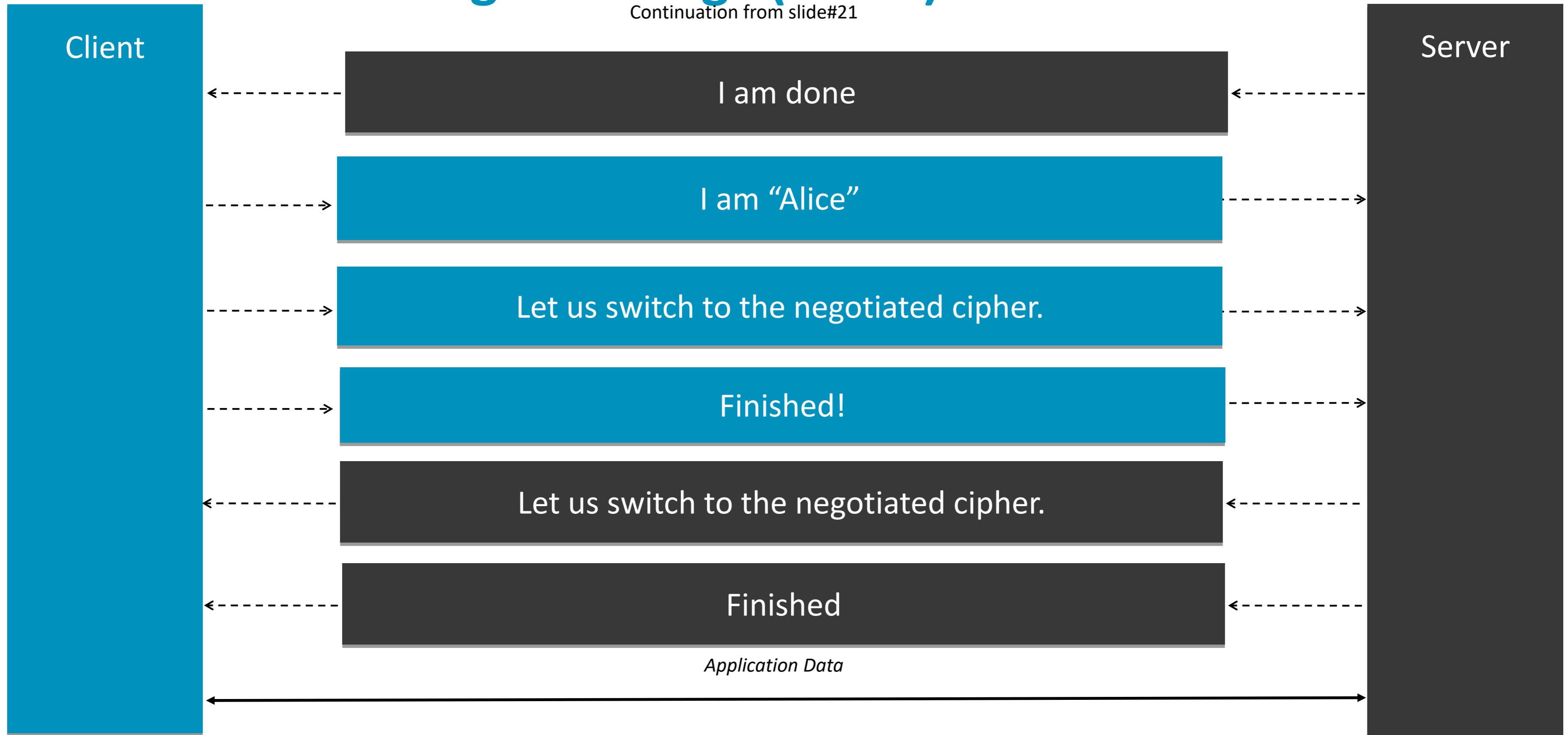
Note: Client and Server share a PSK Identity and a PSK Secret.

Recap: Abstract TLS handshake exchange (Part 1)

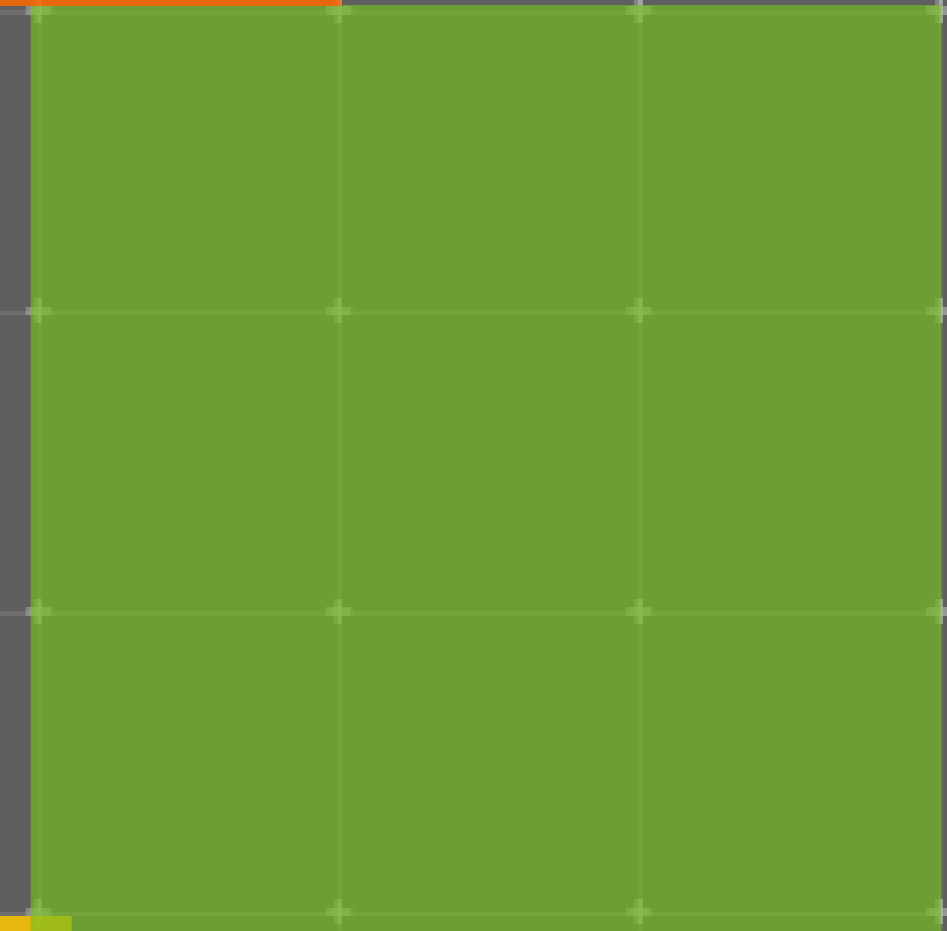


Abstract TLS message exchange (Part 2)

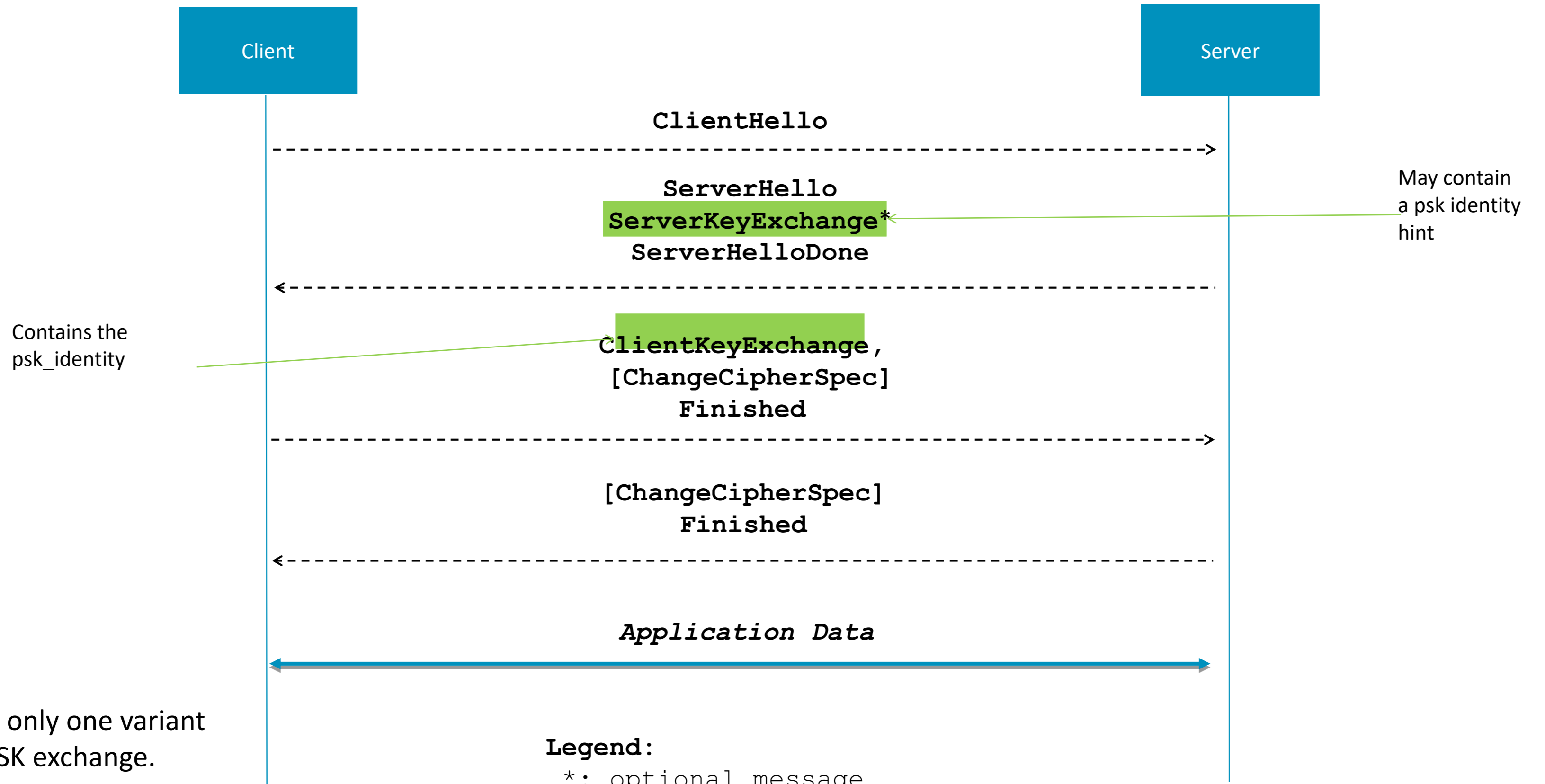
Continuation from slide#21



TLS-PSK handshake (with correct terminology)



TLS-PSK exchange

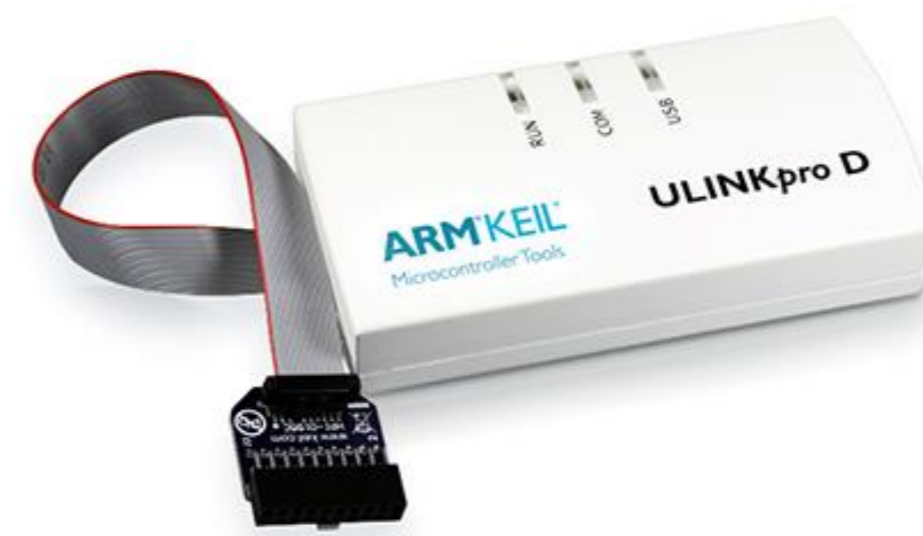


Note: This is only one variant of the TLS-PSK exchange.

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.



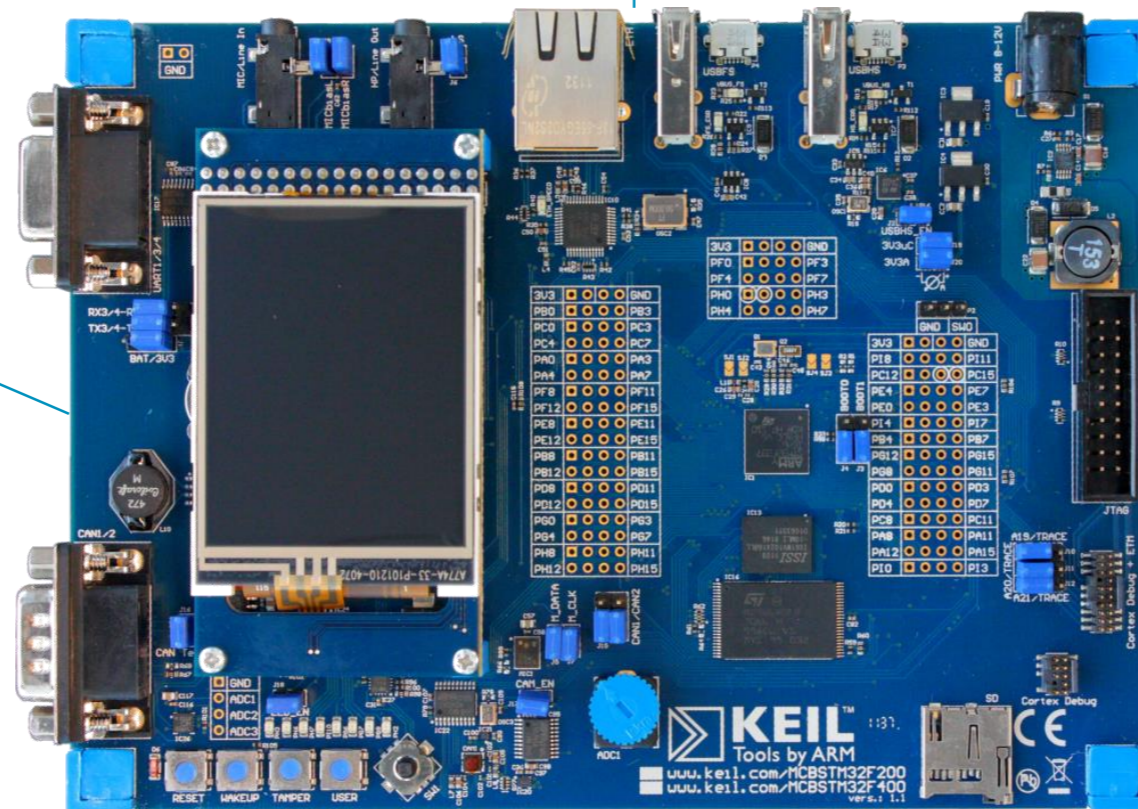
Demo setup



Development laptop



TLS server



Keil
MCBSTM32F400

TLS client

Mbed TLS

Mbed TLS source code: <https://github.com/ARMmbed/mbedtls>

Client

- Modified version of the `ssl_client1.c/ssl_client2.c` example code running on Keil MCBSTM32F400.
- Corresponds to the following command line execution:

```
> ssl_client2 server_addr=<IP_ADDR> debug_level=5  
psk_identity="client" psk=0102030405  
force_version=tls1_2 force_ciphersuite=TLS-PSK-WITH-  
AES-128-CCM-8
```

Server

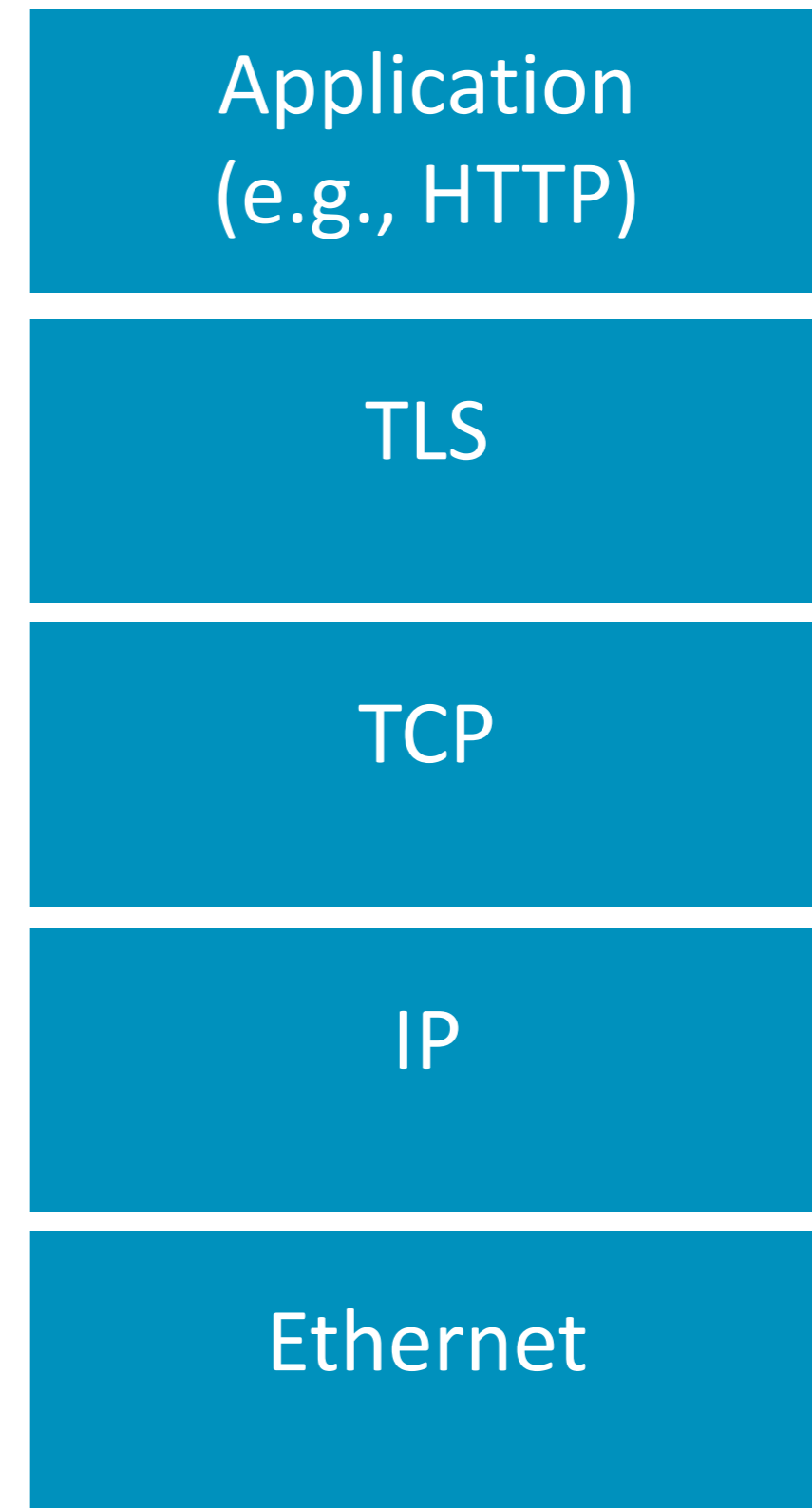
- Test server provided with Mbed TLS source code – `ssl_server2`
- Command

```
> ssl_server2 debug_level=5 psk_identity="client"  
psk=0102030405 force_version=tls1_2
```

Purpose of this hands-on example

For TLS

1. Configure Mbed TLS to execute a PSK-based ciphersuite
2. Become familiar with the Mbed TLS API
3. Understand main steps in establishing a TLS connection to protect HTTP data



Example Protocol
Stack

Config.h

- C Preprocessor Directives for including/excluding functionality into the Mbed TLS library.
- Located in `include/mbedtls/config.h`
- For the Keil IDE an additional file, `mbedtls_config.h` is used.
- All configuration settings with documentation can be found at https://tls.mbed.org/api/config_8h.html

TLS-PSK config.h settings

- According to RFC 7925 we use
 - TLS_PSK_WITH_AES_128_CCM_8 as a ciphersuite, which requires MBEDTLS_KEY_EXCHANGE_PSK_ENABLED
 - TLS 1.2: MBEDTLS_SSL_PROTO_TLS1_2
 - AES, CCM and SHA256 support (MBEDTLS_AES_C, MBEDTLS_CCM_C, MBEDTLS_SHA256_C)
- Debugging features
- System support
- IoT features (e.g., MBEDTLS_SSL_MAX_FRAGMENT_LENGTH)
- [Performance enhancements]

Mbed TLS Application Code (TLS)

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

Performance

Size of the TLS exchange: PSK example

175 bytes of TLS payload with

- 6 byte identity length
- no extensions
- only a single ciphersuite being proposed by client.

Client	Server	Size
ClientHello		49 bytes
	ServerHello	44 bytes
	Server Hello Done	4 bytes
Client Key Exchange		12 bytes
Change Cipher Spec Protocol		1 byte
TLS Finished		32 bytes
	Change Cipher Spec	1 byte
	TLS Finished	32 bytes

Key length

The chosen key length impacts security and performance.

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

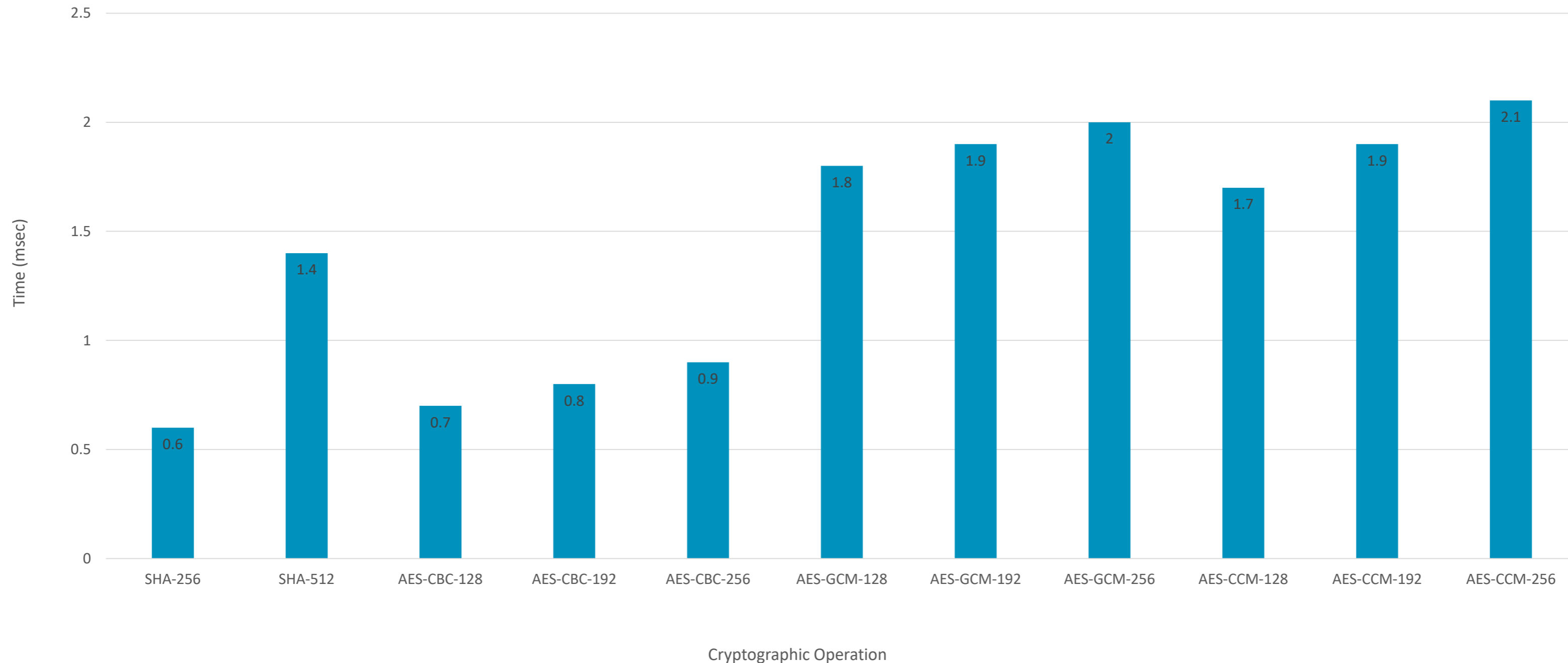
- [RFC 7925](#) recommends to use at least 112 bits symmetric keys for state-of-the-art applications.
- [A 2013 ENISA report](#) states that an 80-bit symmetric key is sufficient for legacy applications but recommends 128 bits for new systems. The lifetime of a device has to be taken into account as well!

Values in table above are based on recommendations from [RFC 4492](#).

See also [RFC 3766](#) for "Determining Strengths For Public Keys Used For Exchanging Symmetric Keys".

Symmetric key crypto performance

LPC 1768 / ARM Cortex-M3 core running at 96 MHz



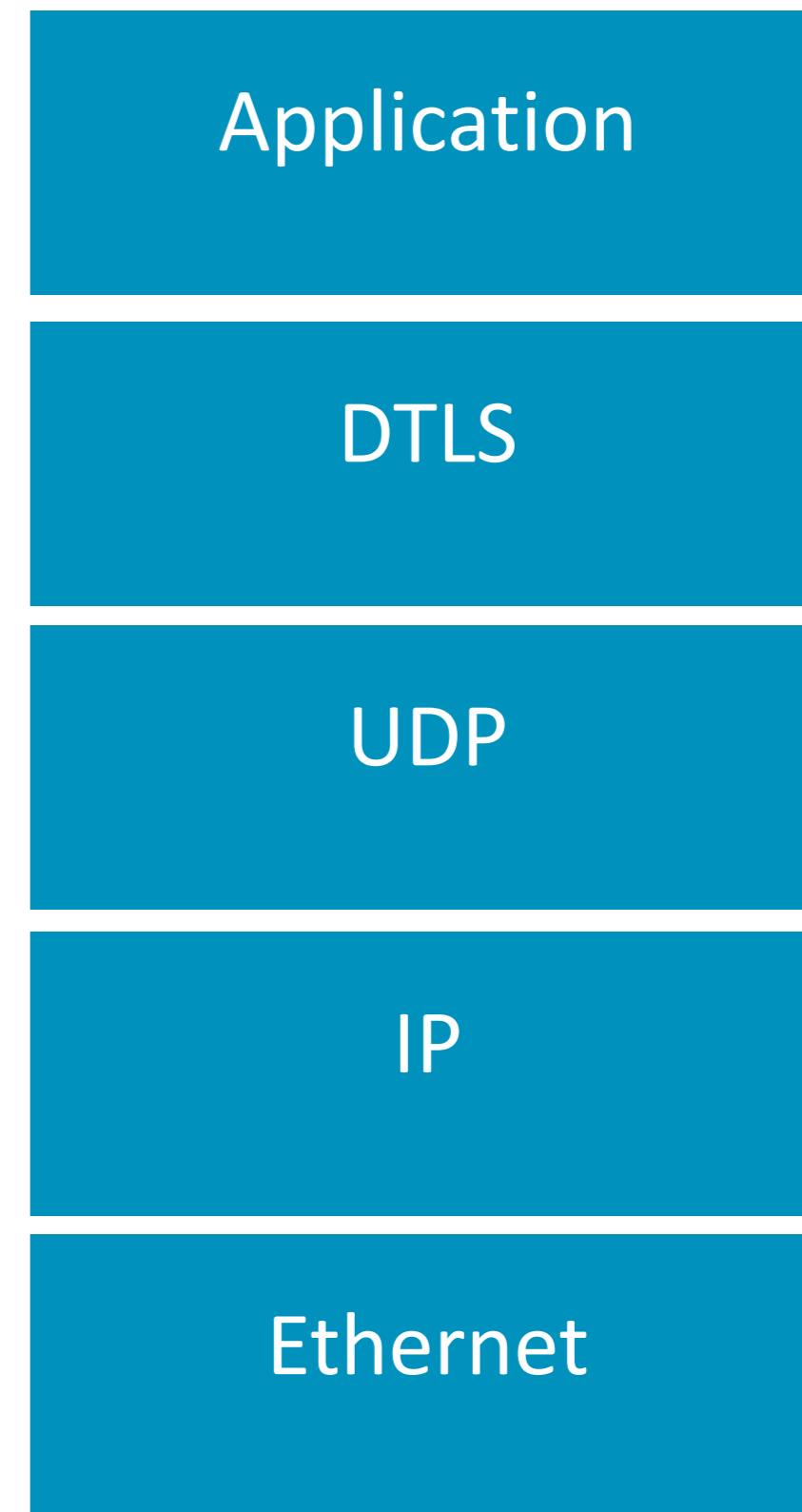
Symmetric key cryptography performance explained

- **SHA** computes a hash over a buffer with a length of 1024 bytes.
- **AES-CBC**: 1024 input bytes are encrypted. No integrity protection is used. IV size is 16 bytes.
- **AES-CCM** and **AES-GCM**: 1024 input bytes are encrypted and integrity protected. No additional data is used. In this version of the test a 12 bytes nonce value is used together with the input data. In addition to the encrypted data a 16 byte tag value is produced.

DTLS

DTLS

- The design of DTLS is intentionally similar to TLS. DTLS 1.2 is documented in [RFC 6347](#).
- DTLS operates on top of an unreliable datagram transport; most importantly over UDP.
 - CoAP is an example of a protocol that runs over UDP.
- But since TLS cannot be used directly over unreliable transports enhancements are needed.



Example Protocol Stack

Conclusion

Summary

- TLS is the most successful Internet security protocol and has helped to secure web and smart phone traffic. It is now being used in the IoT environment.
- TLS is a flexible protocol with various options.
- In part #1 of this webinar we started with TLS-PSK exchange. At the next webinar we look into how to use public key based crypto with TLS and what the performance implications are.

The Arm trademarks featured in this presentation are registered trademarks or trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. All rights reserved. All other marks featured may be trademarks of their respective owners.

www.arm.com/company/policies/trademarks

