Trusted Sockets Layer: A TLS 1.3 based trusted channel protocol

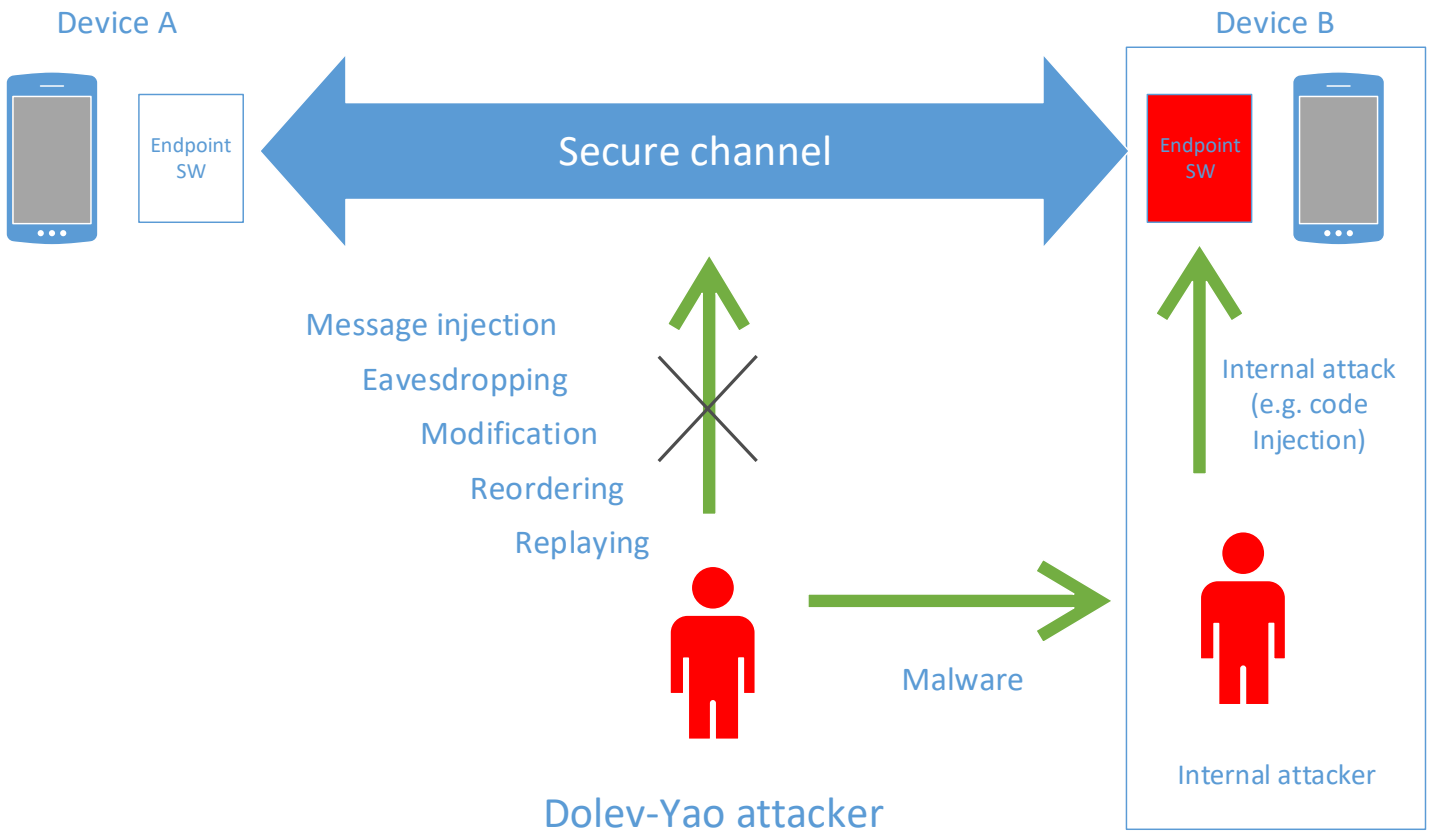
Arto Niemi
Senior Engineer
Helsinki System Security Laboratory (HSSL)
Huawei Technologies Oy (Finland) Co Ltd

NordSec 2021

Niemi A., Pop V.A.B., Ekberg JE. (2021) Trusted Sockets Layer: A TLS 1.3 Based Trusted Channel Protocol. In: Tuveri N., Michalas A., Brumley B.B. (eds) Secure IT Systems. NordSec 2021. Lecture Notes in Computer Science, vol 13115. Springer, Cham. https://doi.org/10.1007/978-3-030-91625-1_10

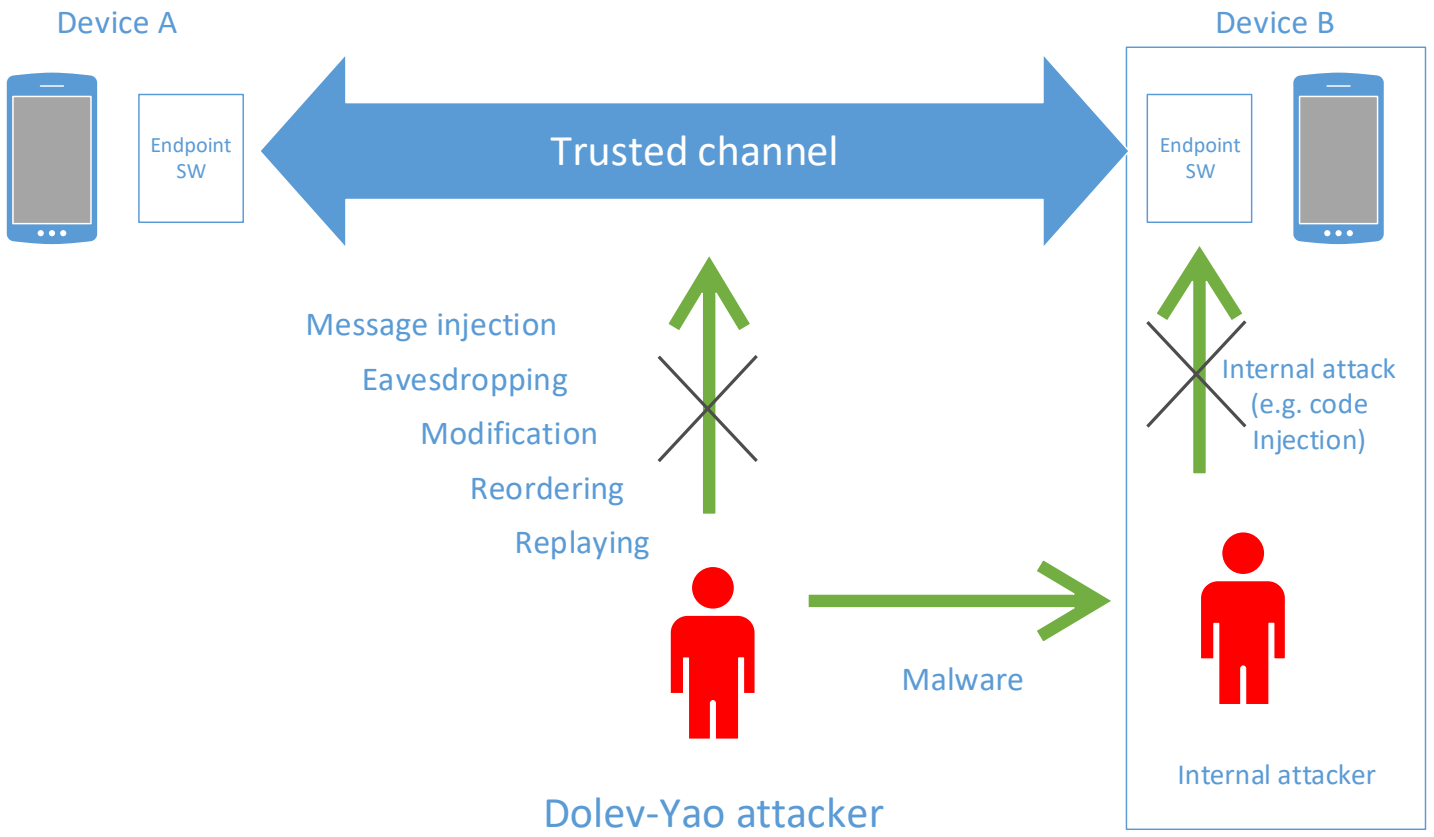
Why secure channels are not enough

- A secure channel protocol such as TLS provides no guarantees about integrity of the endpoint software
- *"Using encryption on the Internet is the equivalent of arranging an armored car to deliver credit card information from someone living in a cardboard box to someone living on a park bench"*
  --- Gene Stafford



Trusted channels

- A *trusted channel* provides:
  - Message confidentiality
  - Message integrity
  - Replay protection
  - Endpoint authentication
  - Endpoint integrity
- In contrast to secure channels (for which we have TLS), there are no standardized, widely used trusted channel protocols



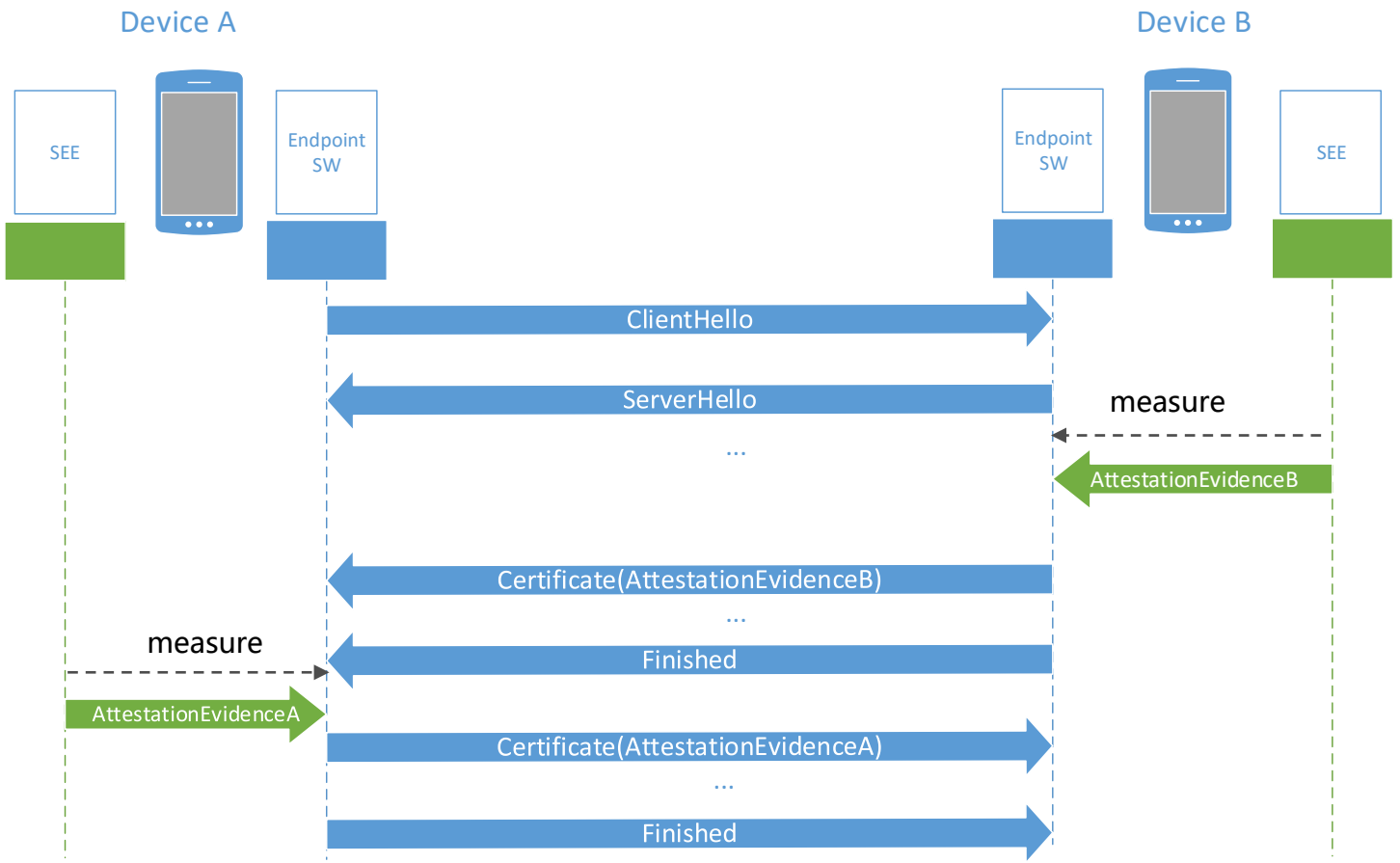
How to design a trusted channel protocol?

- Reminder:
  - trusted channel = secure channel + endpoint integrity
- First step: fulfill secure channel requirements
  - Option A: design a new protocol from scratch
    - Difficult, see timeline of TLS on the right!
  - Option B: use TLS 1.3 as the basis
    - A secure, widely-used and analyzed foundation
- Next step: ensure end-point integrity
  - Can be done with *remote attestation:*
    - A trusted module (e.g. a Secure Execution Environment, SEE) inspects (measures) the endpoint, collects attestation claims, and signs the claims to produce *attestation evidence*
    - Evidence is transmitted to the remote endpoint for verification

1994	SSL 1.0	Flawed design: e.g. no message integrity, no sequence numbers
1995	SSL 2.0	Only 1 cert / endpoint Insecure MD5 hash for integrity prot.
1995	SSL 3.0	RC4 weakness (Later: POODLE attack)
1999	TLS 1.0	IETF takes over from Netscape Attacks against RC4 ciphersuites BEAST attack against CBC padding
2006	TLS 1.1	
2008	TLS 1.2	ROBOT attack against PKCS #1.5 padding (Facebook privkey extracted) Renegotiation attack CRIME attack against compression Authentication privacy issues
2018	TLS 1.3	Concerns about some features like 0RTT data, but no major attacks
2021		IETF deprecates TLS 1.0 and 1.1 due to security issues

Integrating mutual attestation into TLS: a naïve attempt

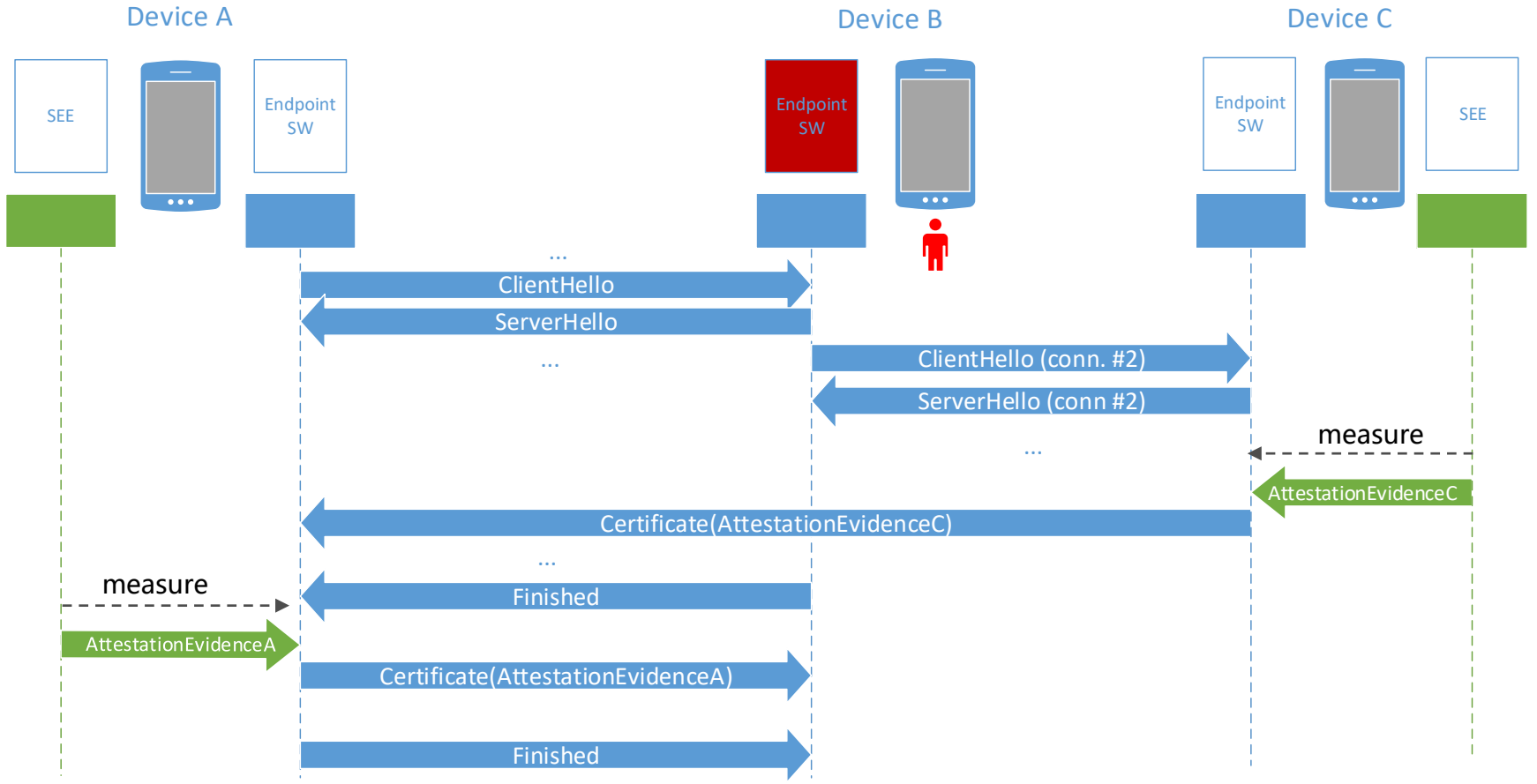
- Both devices generate attestation evidences during a TLS 1.3 handshake
- Evidences are transmitted inside Certificate messages
- Great, we are done!
  - … or are we?



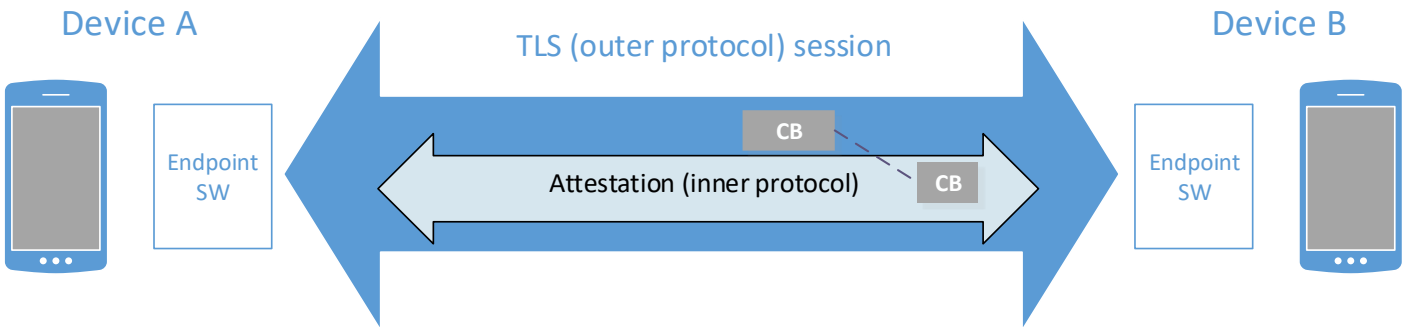
HUAWEI

Relay attack

- Attacker handshakes with an uncompromised device to get a valid-looking attestation evidence for his compromised device
- Possible because the attestation evidence was not bound to a specific TLS handshake or endpoint



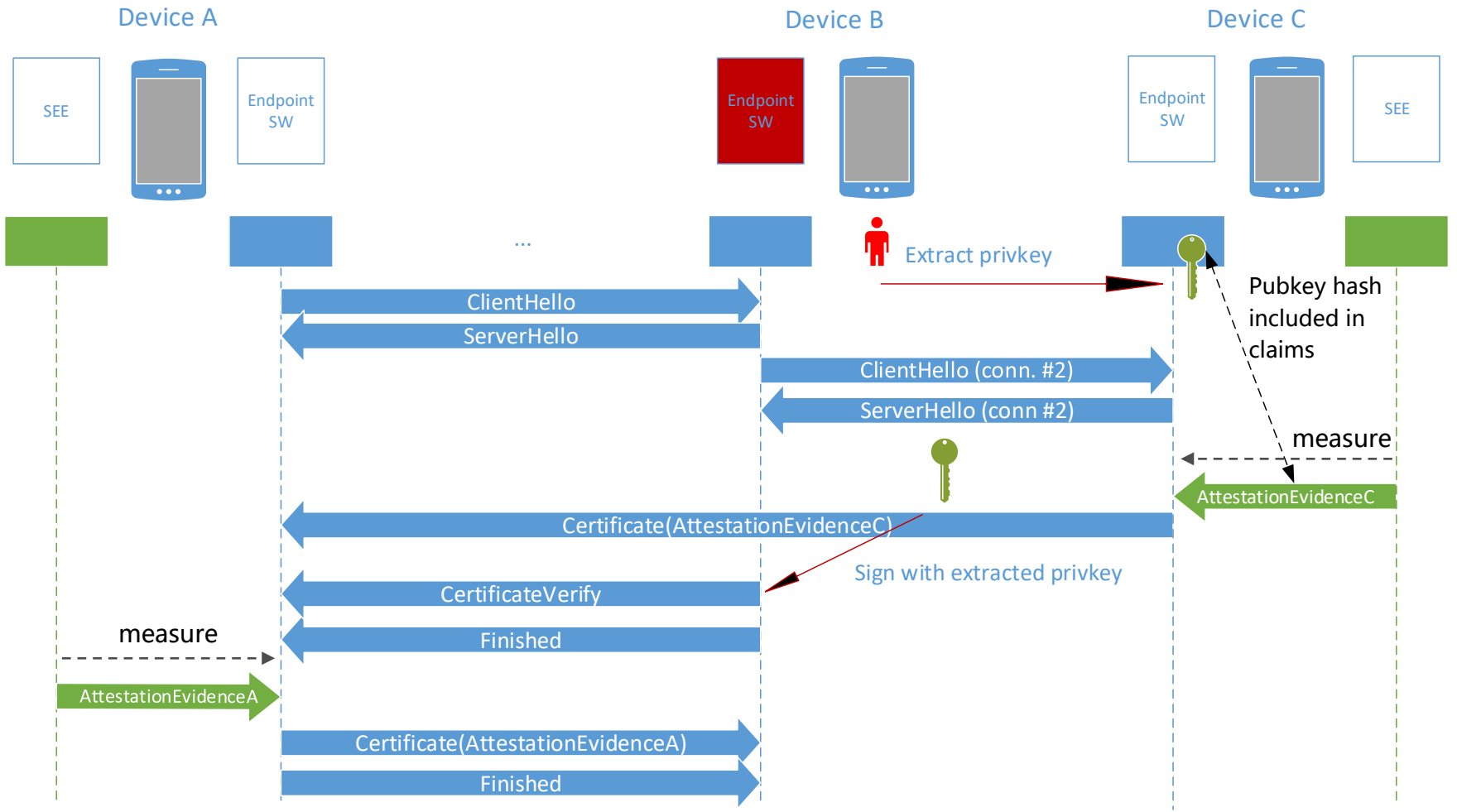
Channel binding



- *Channel binding:*
  - Establishing that no man-in-the-middle exists between two end-points that have attested/authenticated each other in one (inner) protocol, but are using a secure channel provided by another, (outer) protocol
- *Channel bindings* (CB)
  - A unique identifier for a protocol session or endpoint
- *Explicit* channel binding
  - Endpoints compute CBs, transmit them over the wire
  - Endpoints check that self-computed CB matches received CB
- *Implicit* channel binding
  - CB of inner protocol is used in the key derivation of the outer protocol

Collusion / insider attack

- Attestation evidence is bound to endpoint authentication keypair, but the key pair is stored in the REE
- Attacker extracts the private key belonging to an uncompromised endpoint, allowing him to pass endpoint authentication (in addition to attestation)
- Better: bind attestation evidence to a specific handshake



HUAWEI

Threat model

- **Standard Dolev-Yao capabilities:**
  - Eavesdropping
  - Message modification, replay, etc.
- **Insider attack capability on compromised devices:**
  - Arbitrary modification of endpoint software
- **Admin-level access to uncompromised endpoints:**
  - Allows relay attacks
  - Allows collusion attacks, e.g. extraction of long-term authentication privkey without changing endpoint code

- The attacker does **not** have:
  - Capability to extract live TLS session key material from the memory of an uncompromised endpoint[1]

1 Such an attack could be prevented by running the TLS endpoint in an SEE. This is addressed in our subsequent (current) work. In the present paper, we wanted to provide a more general solution for REE-based TLS endpoints.

Design choices

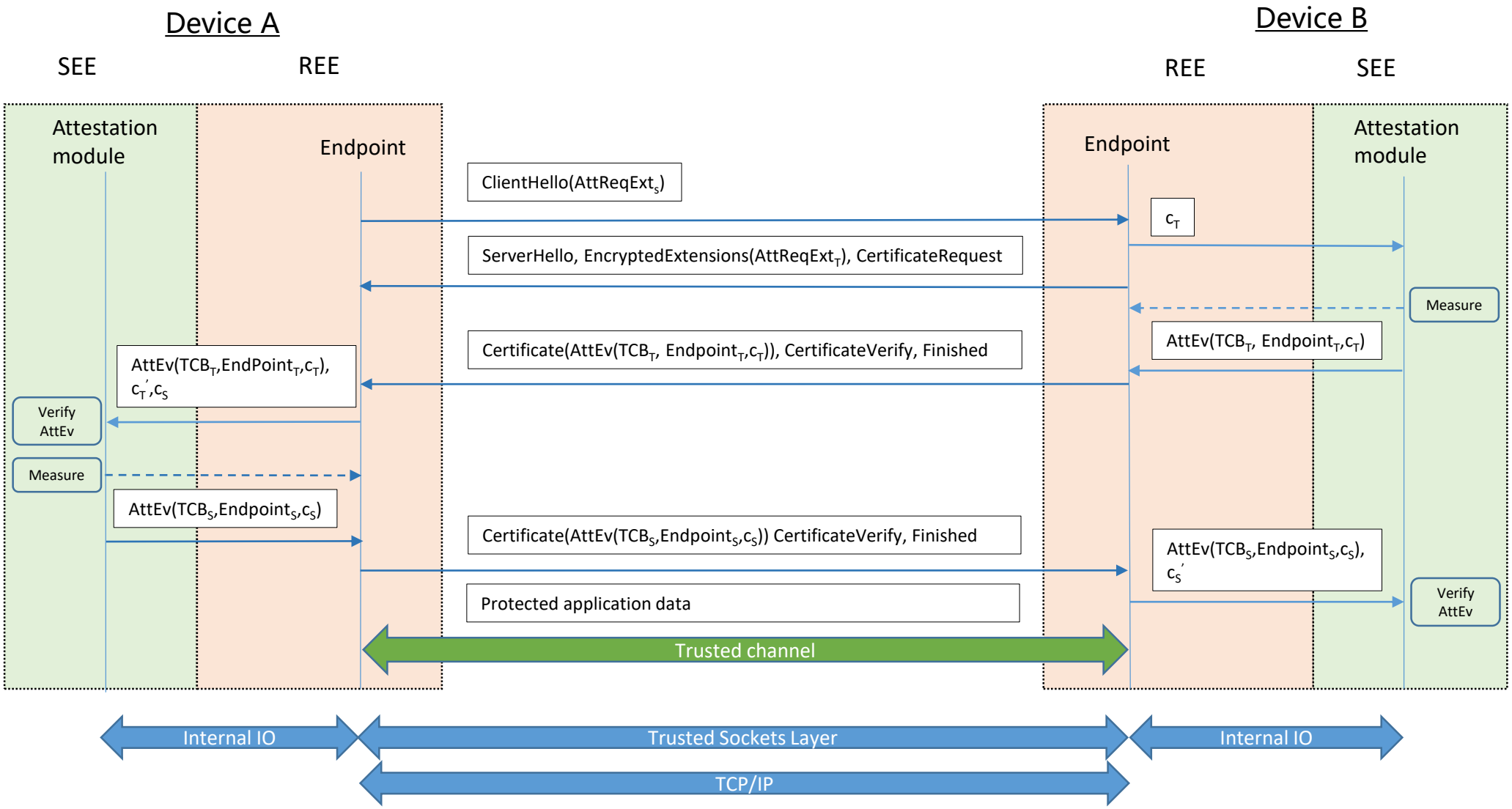
- **Which TLS protocol version to use?**
  - Only TLS 1.3 is secure by default (see timeline of attacks earlier)
  - Only TLS 1.3 encrypts authentication messages (privacy)
- **When to generate attestation evidence?**
  - Before the TLS handshake (*pre-handshake approach*)
    - Cannot bind evidence to a specific handshake → risk of replay attacks
  - After TLS session establishment (*post-handshake approach*)
    - Requires an extra round-trip to transmit attestation evidences
  - During the TLS handshake (*intra-handshake approach*)
    - Attestation evidence can be bound to handshake in progress
    - Requires no extra protocol round-trips on top of TLS HS
    - Converts TLS to a trusted channel protocol without requiring an extra protocol on top
- **Implicit or explicit channel binding?**
  - Explicit is better (we do not want to modify TLS 1.3 key derivation spec)

- **What to use as the channel bindings (CB)?**
  - Should be unique to the TLS handshake/session
  - Should be easily computable using standard TLS library APIs
  - TLS end-point authentication public key (or certificate)
    - Binds attestation to an endpoint identity, not to a handshake
    - Authentication key pair requires long-term secure storage
  - tls-unique (RFC 5929)
    - Not defined for TLS 1.3; vulnerable to renegotiation attacks
  - ECDHE public value
    - Better, but some TLS implementations cache ECDHE key pairs
  - ClientHello hash
    - Binds to both ECDHE public value and the current handshake
    - Relatively easy to extract using standard APIs
  - TLS-Exporter
    - Recommend channel bindings for TLS 1.3; standard APIs exist

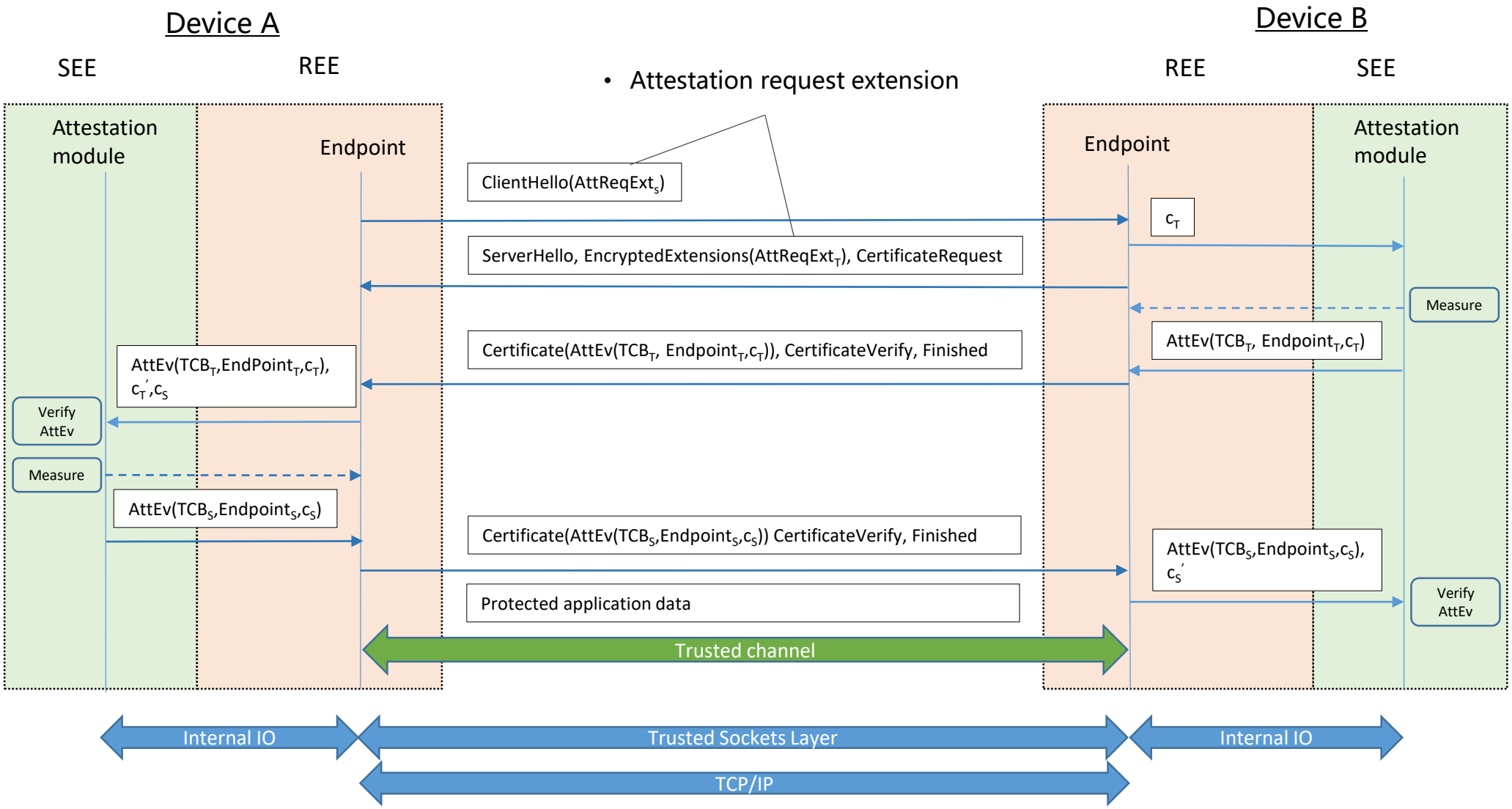
Comparison of existing TLS-based trusted channel protocols

	Goldman	Knauth	Walsh	Gasmi	PT-TLS	IDSCP	Aziz
TLS version	1.0	1.2	1.2	1.0	1.2	1.2	1.0
Changes TLS spec.	No	No	No	Yes	No	No	No
Attestation generation	Pre-HS	Pre-HS	Post-HS	Intra-HS	Post-HS	Post-HS	Post-HS
Channel bindings	Auth. public key	Auth. public key	(EC)DH public key	DH public key	tls-unique	Auth. cert	Auth. cert, hello nonce
Attestation privacy	No	No	Yes	Yes	Yes	Yes	Yes
Extra RTTs for attestation	0	0	1	0	2	2	1.5
Targeted TCB hardware	TPM	SGX	TPM	TPM	All	TPM	TPM
Relay or collusion attacks	Yes	Yes	See text	No	Yes	Yes	Yes
Key separation	Yes	Yes	Yes	Yes	Yes	Yes	No

Trusted Sockets Layer (TSL) protocol

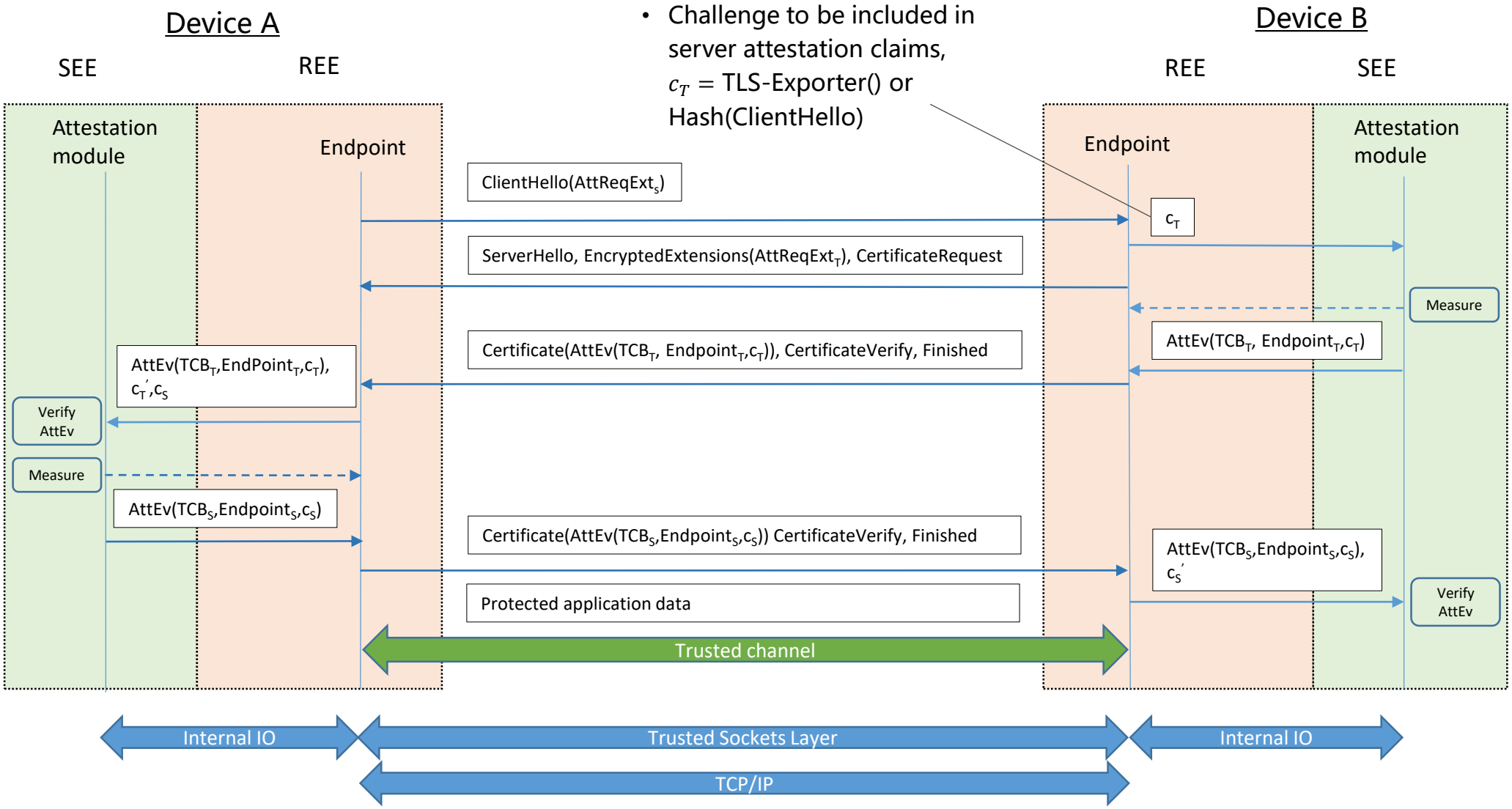


Trusted Sockets Layer (TSL) protocol



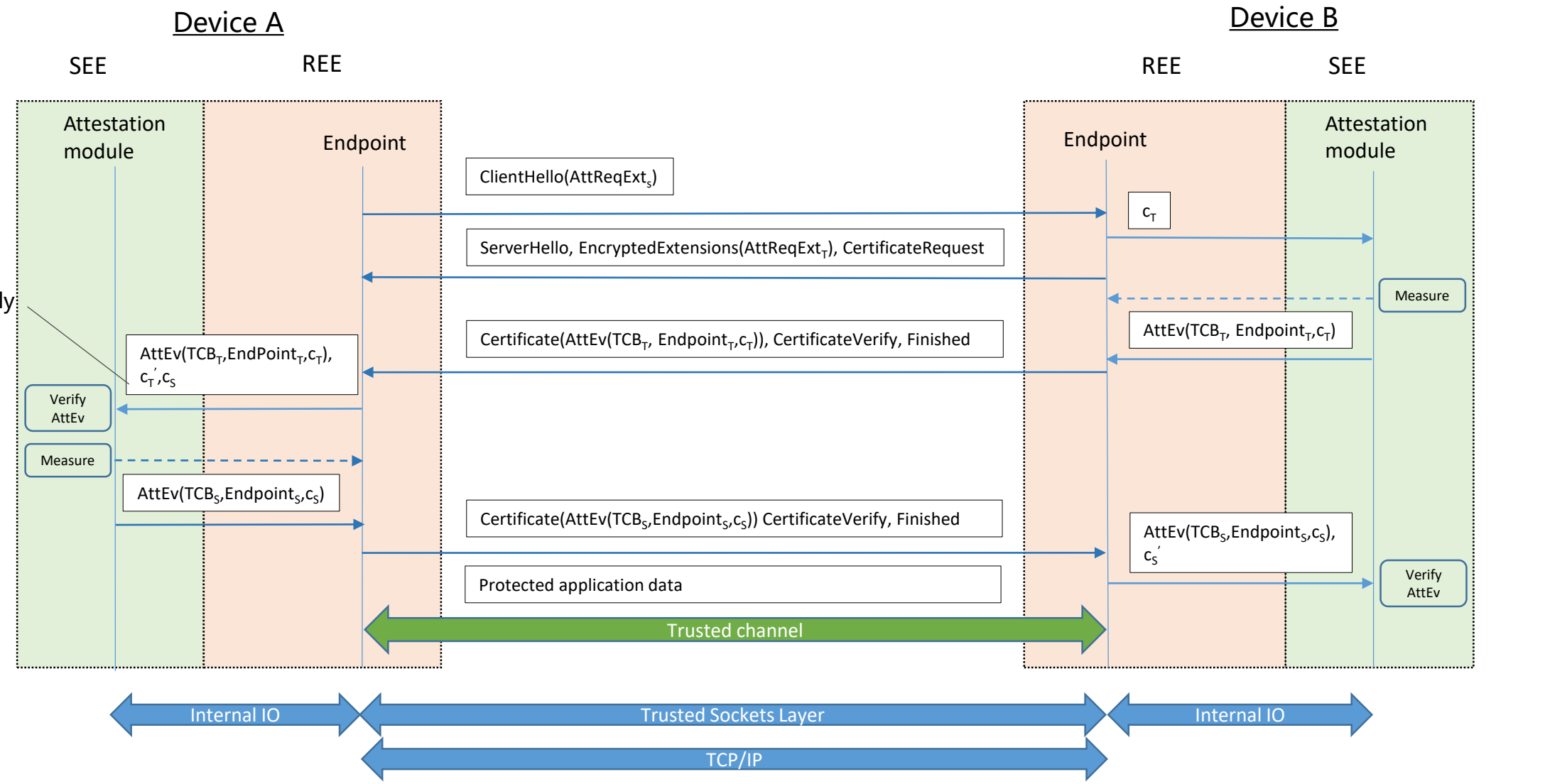
Trusted Sockets Layer (TSL) protocol

- Challenge to be included in server attestation claims, c_T = TLS-Exporter() or Hash(ClientHello)



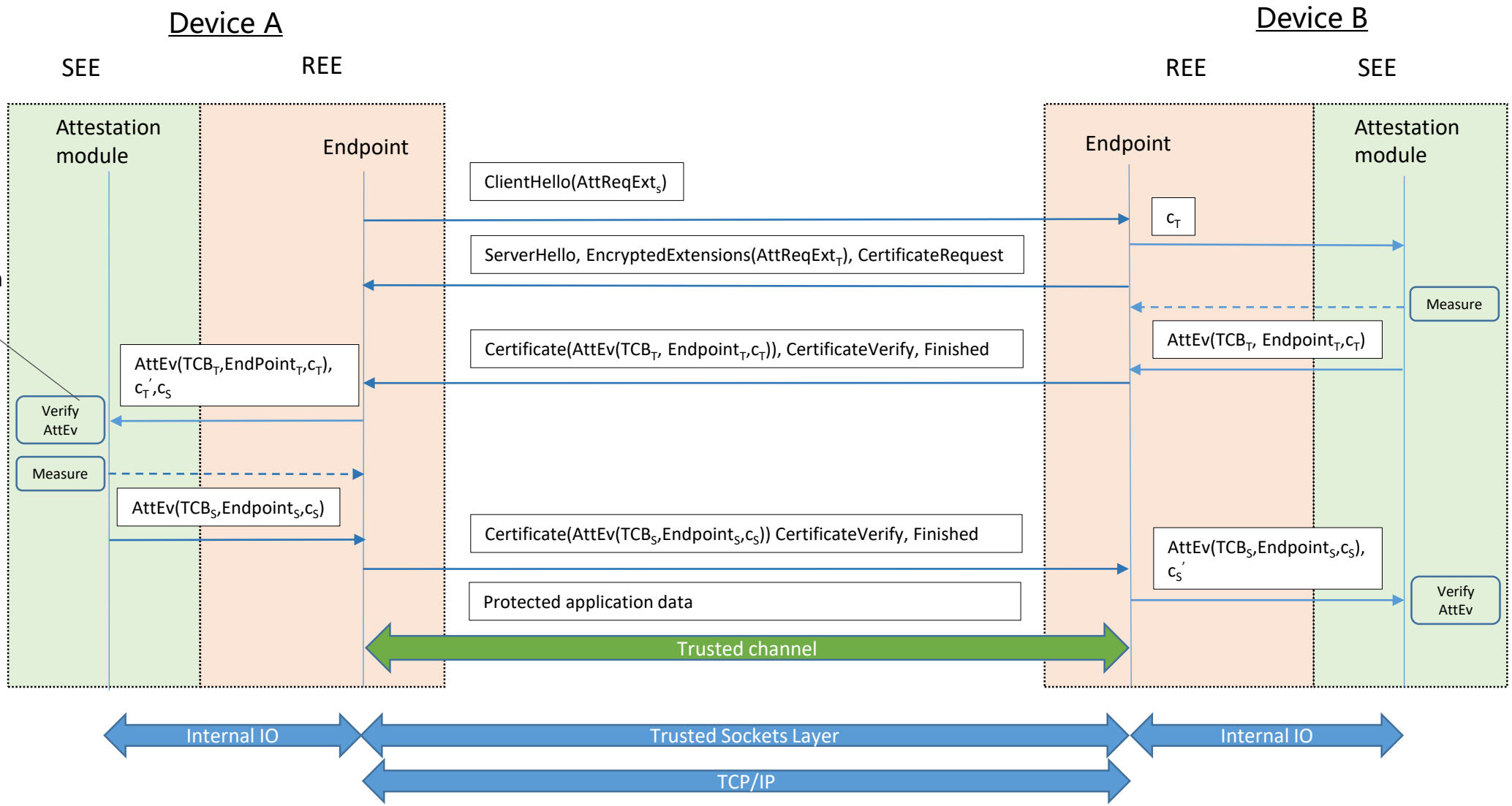
Trusted Sockets Layer (TSL) protocol

- Client computes reference c_T independently

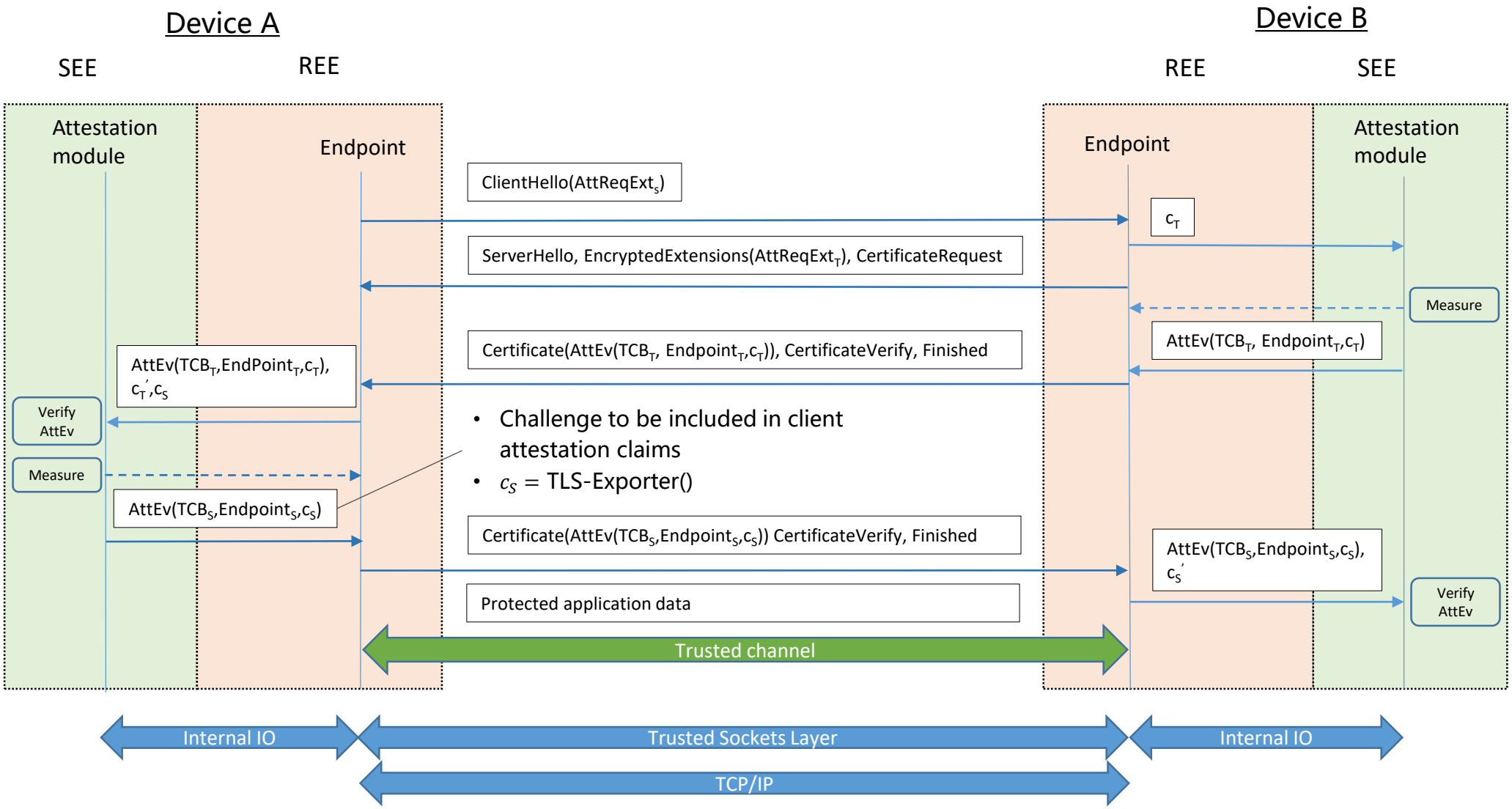


Trusted Sockets Layer (TSL) protocol

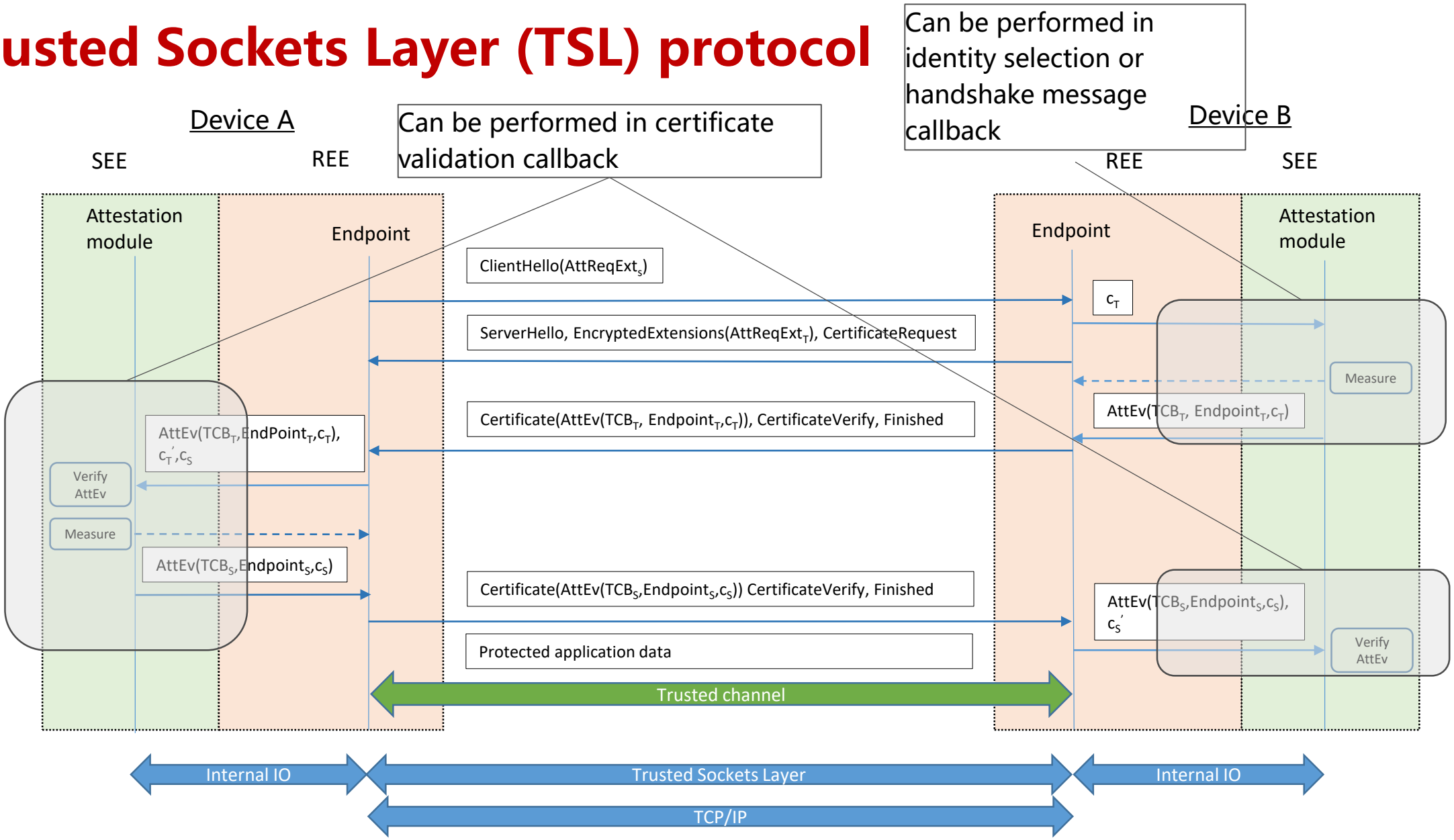
- Attestation evidence is verified within the SEE
- Challenge must match reference value



Trusted Sockets Layer (TSL) protocol



Trusted Sockets Layer (TSL) protocol



TSL vs prior art

	Goldman	Knauth	Walsh	Gasmi	PT-TLS	IDSCP	Aziz	TSL
TLS version	1.0	1.2	1.2	1.0	1.2	1.2	1.0	1.3
Changes TLS spec.	No	No	No	Yes	No	No	No	No
Attestation generation	Pre-HS	Pre-HS	Post-HS	Intra-HS	Post-HS	Post-HS	Post-HS	Intra-HS
Channel bindings	Auth. public key	Auth. public key	(EC)DH public key	DH public key	tls-unique	Auth. cert	Auth. cert, hello nonce	TLS-Exporter
Attestation privacy	No	No	Yes	Yes	Yes	Yes	Yes	Yes
Extra RTTs for attestation	0	0	1	0	2	2	1.5	0
Targeted TCB hardware	TPM	SGX	TPM	TPM	All	TPM	TPM	All
Relay or collusion attacks	Yes	Yes	See text	No	Yes	Yes	Yes	No
Key separation	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes

Summary

- **We surveyed a large number of TLS-based trusted channel protocols**
  - Many are vulnerable to relay and collusion attacks
  - Many require an extra protocol round-trip on top of TLS handshake
  - Most are designed for specific SEE hardware (e.g. TPM or SGX)
  - Some provide no privacy for attestation, or violate the key separation principle
  - All use obsolete TLS versions (1.2 and earlier)

- **Our Trusted Sockets Layer (TSL) protocol**
  - Uses strong channel bindings, computed with standard TLS-Exporter mechanism
  - Requires no extra protocol round-trips
  - Is hardware-agnostic
  - Uses the latest and most secure TLS version (1.3)
  - Is compliant with the TLS spec and RFC 5280 (extending Certification Path Validation Algorithm with attestation)
  - Can be easily implemented with existing TLS libraries via callback interfaces

- **And implemented two proof-of-concepts**
  - One using OpenSSL and another with our internal small-footprint TLS 1.3 implementation suitable for SEEs
  - Some of this work is currently in the process being open sourced

Conclusions and further work

- **Our current work on Trusted Sockets Layer (TSL) includes:**
  - Using the protocol to migrate enclaves between SEEs
  - A unified attestation framework to abstract HW-specific attestation mechanisms
  - Industry-strength implementation:
    - Simpler wrapper for existing APIs (e.g. `TSL_write` instead of `SSL_write`)
    - Comprehensive fuzz and defect testing framework

- **Want to help?**
  - Check out our internships!
    - https://jobs.workable.com/preview/3ab589e2-a99b-4dd3-b40c-2a928992056e
    - https://jobs.workable.com/preview/f4274fbc-573d-4b75-9ddb-00efe3781e42

Conclusions and further work

- **Closing thoughts**
  - In the beginning, there were TCP/IP sockets
    - But communication was vulnerable to attacks from the network
  - Then came SSL/TLS and made sockets *secure*
    - But endpoints were vulnerable to attacks from within the device
  - Then came TSL and made sockets ***t****rusted*
    - Perhaps in the future, we all use http**t**:// instead of https://

Questions?