Improved robustness in DTLS

Marco Tiloca

(SICS Swedish ICT)

Hamid Rahmouni
Maarten Hoeve

(ENCS)

Improved robustness in DTLS

- **Communication with RTUs**
  - Inter and intra domain
  - Data, control info, commands
- **Secure communication channels**
  - Authentication, integrity, secrecy
  - Typically achieved at the transport layer
- **TLS and DTLS are mostly adopted**
  - TLS for TCP communication
  - DTLS for UDP communication

Improved robustness in DTLS

DTLS overview (1/2)

- **Transport-level security**
  - IETF standard since 2012
  - In short, “TLS over UDP”

- **Same TLS assurances**
  - Hop-by-hop security
  - Prevent eavesdropping
  - Prevent message forgery

- **Wide adoption**
  - De-facto choice for datagram application
  - De-facto security protocol in M2M and IoT

SEGRID

DTLS overview (2/2)

- **Alternative choice to TLS for**
  - Datagram-based communication
  - Interaction with embedded systems
  - CoAP as application level protocol

I E T F

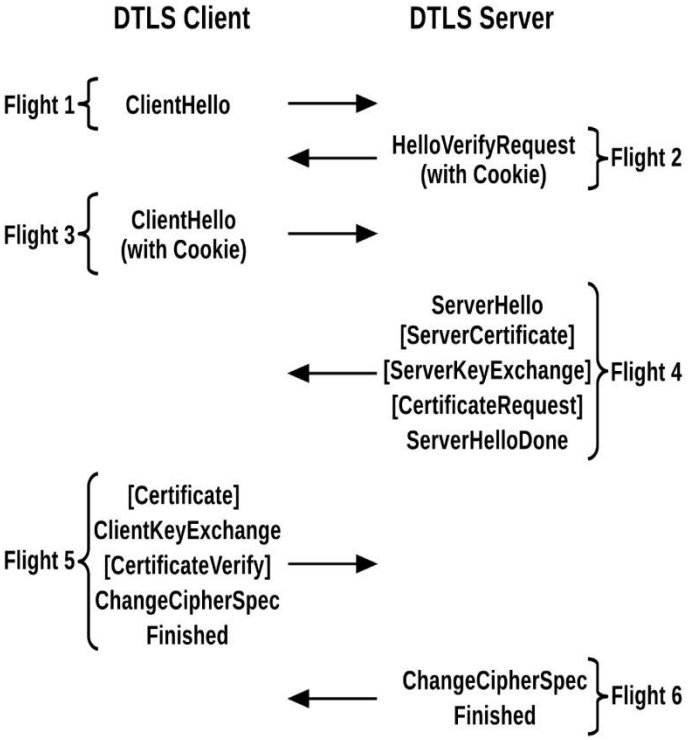
- **Designed with TLS in mind**
  - Session established through a *handshake*
  - Messages transmitted as secure records
  - Same security services and cyphersuites

Type	Version	Epoch	Sequence Number	Length	Fragment
1 Byte	2 Bytes	2 Bytes	6 Bytes	2 Bytes	Variable

Improved robustness in DTLS

Handshake overview (1/2)

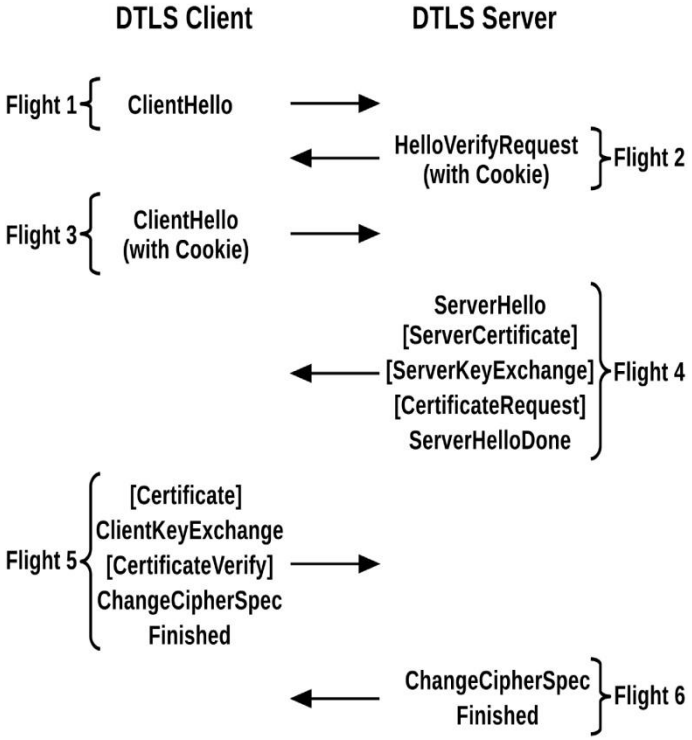
- **Between a client and a server**
  - Agree on algorithms and parameters
  - Establish secrets and key material
  - Mutual authentication in different ways
- **Complex process**
  - Time consuming
  - Resource demand
  - Three message rounds
- **Eventual session establishment**
  - Support for session resumption



Improved robustness in DTLS

Handshake overview (2/2)

- **Flight 1 – Flight 3**
  - Announcement as *ClientHello* message
  - DoS protection, with Cookie exchange
- **Flight 4 – Flight 5**
  - Mutual authentication
  - Establishment of session key material
- **Flight 6**
  - Confirmation and session start

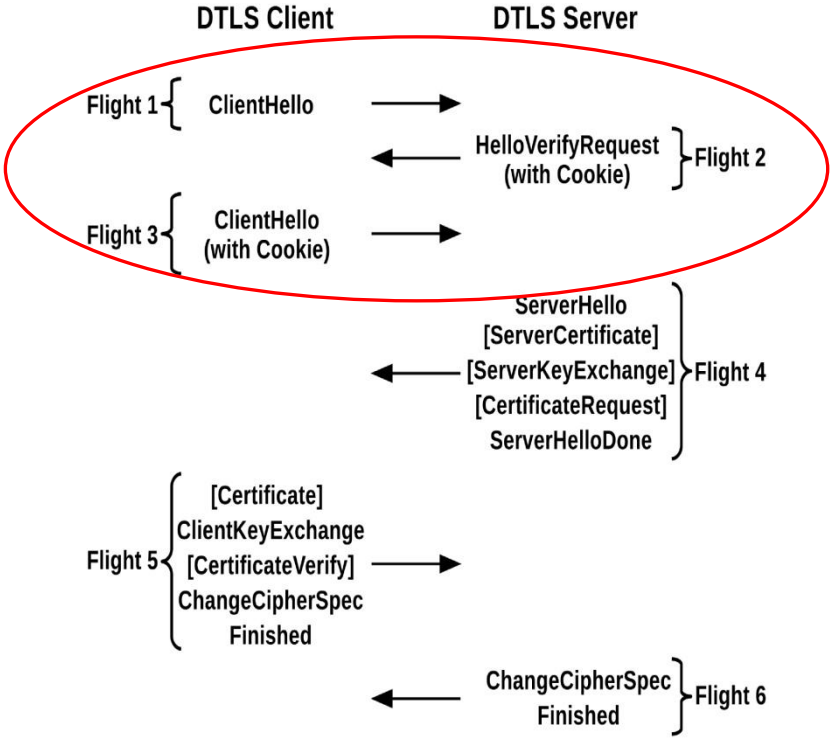


- **The handshake suffers of two main issues!**

Improved robustness in DTLS

Denial of Service attack (1/2)

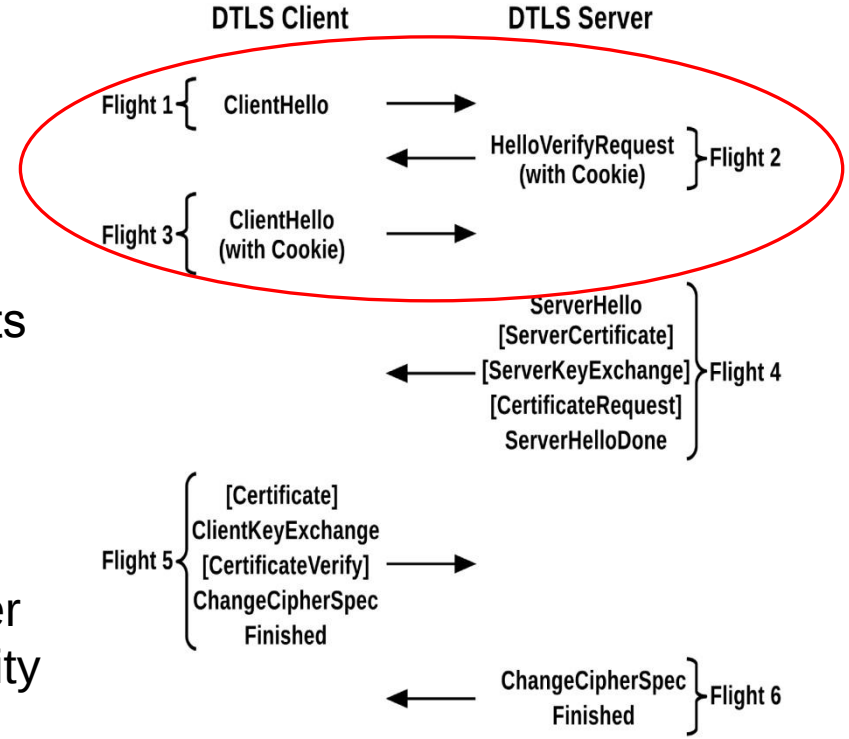
- **Cookie exchange**
  - The server generates a Cookie
  - The client replies with the same Cookie
- **Cookie effectiveness**
  - Only simple attacks are prevented
  - More complex attacks are still possible
- **Actual flaw:**
  - The cookie is not authenticated
  - The cookie is sent as plaintext



Improved robustness in DTLS

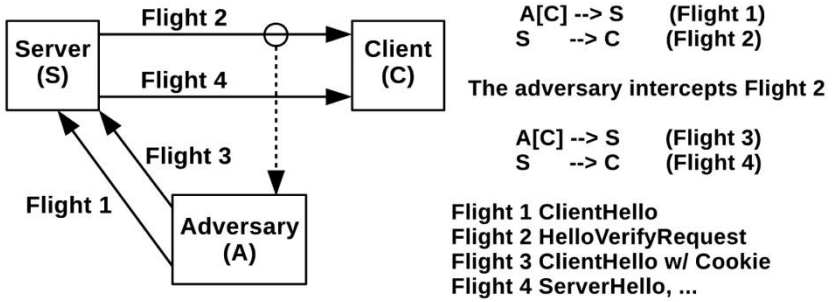
Denial of Service attack (2/2)

- **Smarter DoS attack**
  - Cookie interception and re-usage
  - The handshake continues until Flight 4
  - “Reflections” on innocent unaware clients



- **Attack impact**
  - Invalid, half-open, sessions on the server
  - Reduced server resources and availability

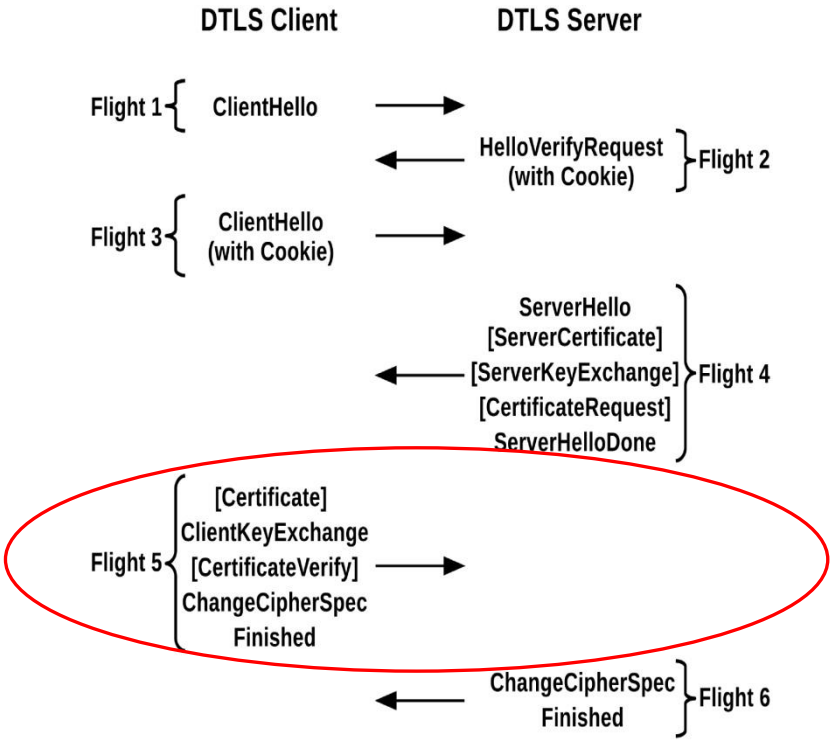
- **Same for TLS, although:**
  - No Cookie exchange is performed
  - Need to first establish a TCP session

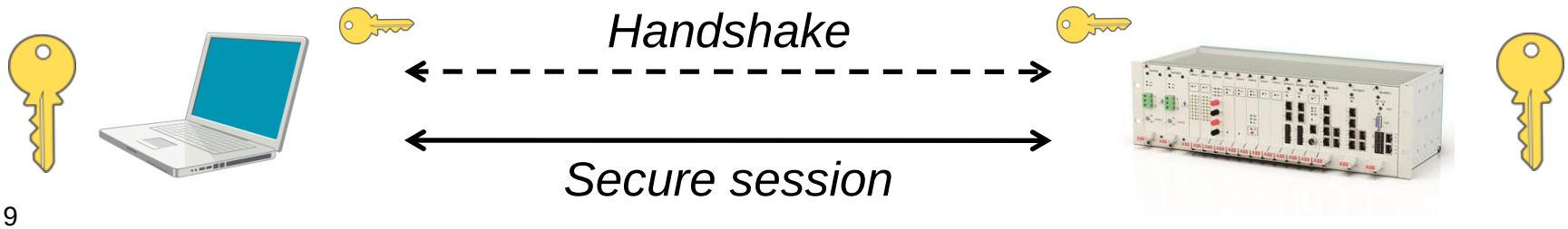


Improved robustness in DTLS

Storage of pre-shared keys (1/2)

- **Pre-shared key establishment**
  - Convenient and often adopted
  - Client authentication with no certificate
- **One symmetric key**
  - Pre-shared between client and server
  - Used during the handshake process

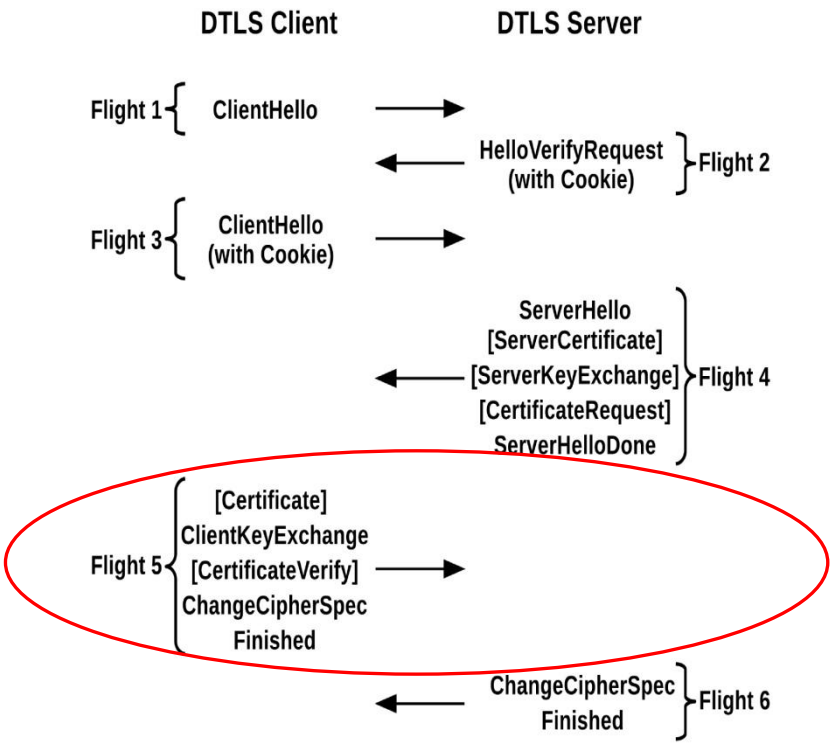




Improved robustness in DTLS

Storage of pre-shared keys (2/2)

- **Key storage on the server**
  - One pre-shared key per client
  - Retrieved during the handshake
  - Used to establish security material
- **Poor scalability**
  - Storage overhead grows linearly
- **Key provisioning and management**
  - Complicated in dynamic environments



Improved robustness in DTLS

Solved both issues

- **Adopted approach (DoS)**
  - Identify invalid *ClientHello* messages
  - Recognize invalid handshakes
  - Early abort of invalid handshakes

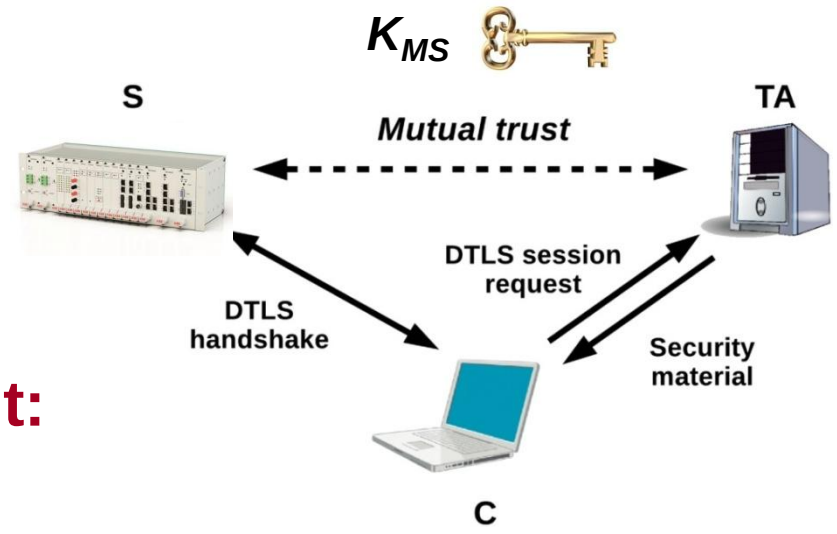
- **Adopted approach (key storage)**
  - Decouple server stored material from clients
  - Massively reduce the number of stored keys

- **Main features**
  - Single architecture addressing both issues
  - Compliant minor changes in the handshake
  - Involvement of a Trusted Third Party

Improved robustness in DTLS

Solved both issues (*)

- **The server:**
  - Shares a key with a Trust Anchor (TA)
  - Locally stores only that key (K_{MS})



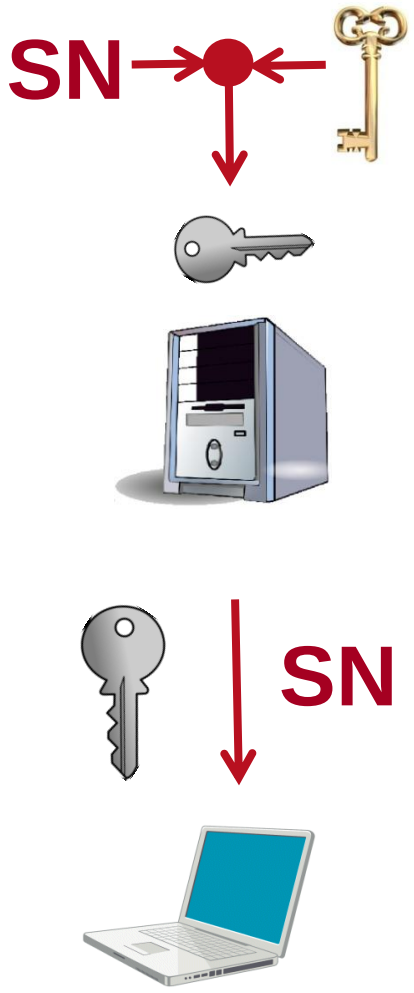
- **Before the handshake, the client:**
  - Contacts the Trust Anchor
  - Receives two additional support keys

- **During the handshake, the client:**
  - Authenticates the *ClientHello* message
  - Suggests the server how to derive a shared key

(*) M. Tiloca, C. Gehrmann and L. Seitz, *On Improving Resistance to Denial of Service and Key Provisioning Scalability of the DTLS Handshake*, International journal of Information Security, Springer, 2016 (To appear)

Improved robustness in DTLS

Solution against DoS (1/2)

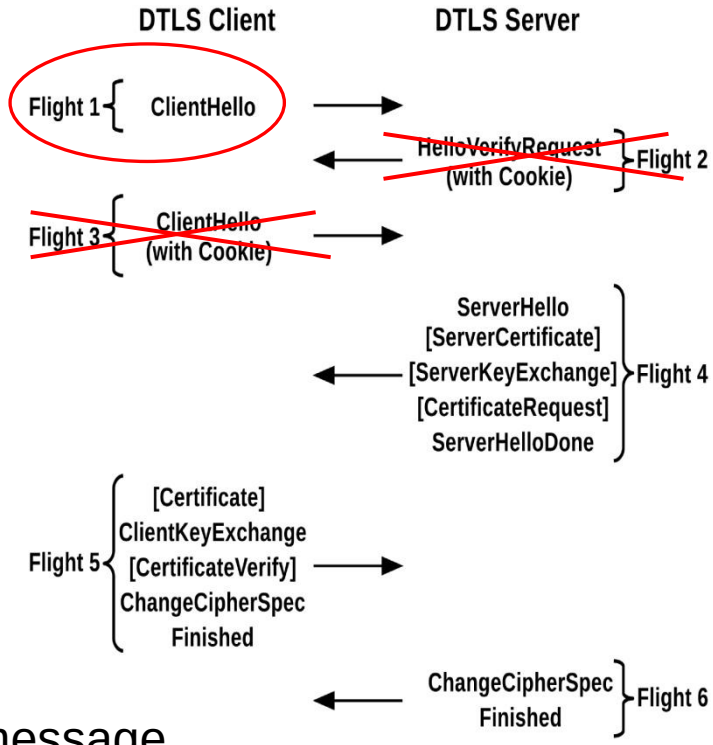


- **Step 1 (Client)**
  - Receive a **MAC key** and a **sequence number** from the TA
  - The key is derived from K_{MS} shared between server and TA
- **Step 2 (Client)**
  - Sequence number included in the *ClientHello* message
  - MAC key used to authenticate the *ClientHello* message
  - Many techniques are possible (HMAC, Galois-Field, …)
- **Step 3 (Client)**
  - Resulting MAC included in the *ClientHello* message

Improved robustness in DTLS

Solution against DoS (2/2)

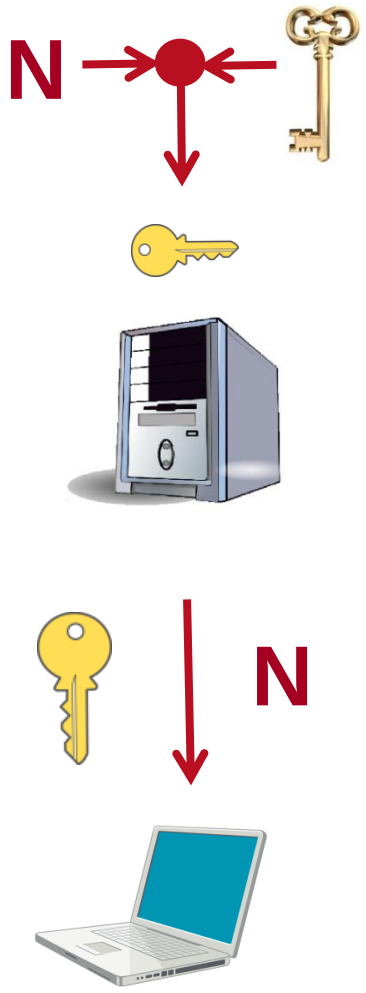
- **Step 4 (Server)**
  - Receive the *ClientHello* message
  - Retrieve the conveyed sequence number
- **Step 5 (Server)**
  - Consider the sequence number and K_{MS}
  - Derive the correct MAC key
- **Step 6 (Server)**
  - Use the MAC key to verify the *ClientHello* message
  - The computed MAC is compared with the conveyed one



The Cookie exchange is not necessary anymore

Improved robustness in DTLS

Scalable key storage (1/2)

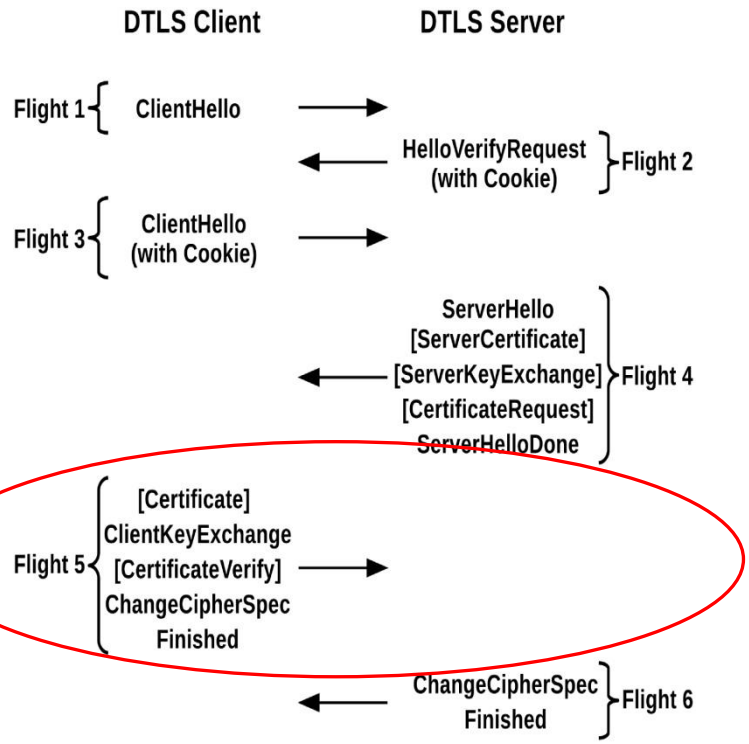


- **Step 1 (Client)**
  - Receive a **pre-shared key** and a **nonce** from the TA
  - The key is derived from K_{MS} shared between server and TA
- **Step 2 (Client)**
  - Nonce included in the *ClientKeyExchange* message
- **Step 3 (Client)**
  - The received key is hereafter used as DTLS pre-shared key

Improved robustness in DTLS

Scalable key storage (2/2)

- **Step 4 (Server)**
  - Receive the *ClientKeyExchange* message
  - Retrieve the conveyed nonce
- **Step 5 (Server)**
  - Consider the nonce and K_{MS}
  - Derive the pre-shared key
- **Step 6 (Server)**
  - The derived key is hereafter used as DTLS pre-shared key



SEGRID

Conclusions

- **Achieved goals**
  - The Denial of Service attack is neutralized
  - The server does not start invalid DTLS sessions
  - The server preserves resource and availability
  - The server stores only one key K_{MS}
- **Side benefits**
  - No additional communication with clients
  - The Cookie exchange is not required anymore
  - We do not break the current standards
  - Same approach reusable also in TLS
- **Proof-of-concept implementation**
  - Californium/Scandium Java libraries
  - First tests in simple local settings

SWEDISH ICT SICS

Improved robustness in DTLS

Demo!

- **SEGRID case study**
  - Communication between RTUs
  - First Java implementation by SICS
  - Porting adaptation by ENCS
- **Up and running on ENCS equipment**
  - Improved DTLS handshake
  - Proof of correct session establishment
- **Tested under DoS**
  - Counteraction of a real attack instance

ENCS

Thanks for your attention!

Marco Tiloca
marco@sics.se
http://www.sics.se/~marco