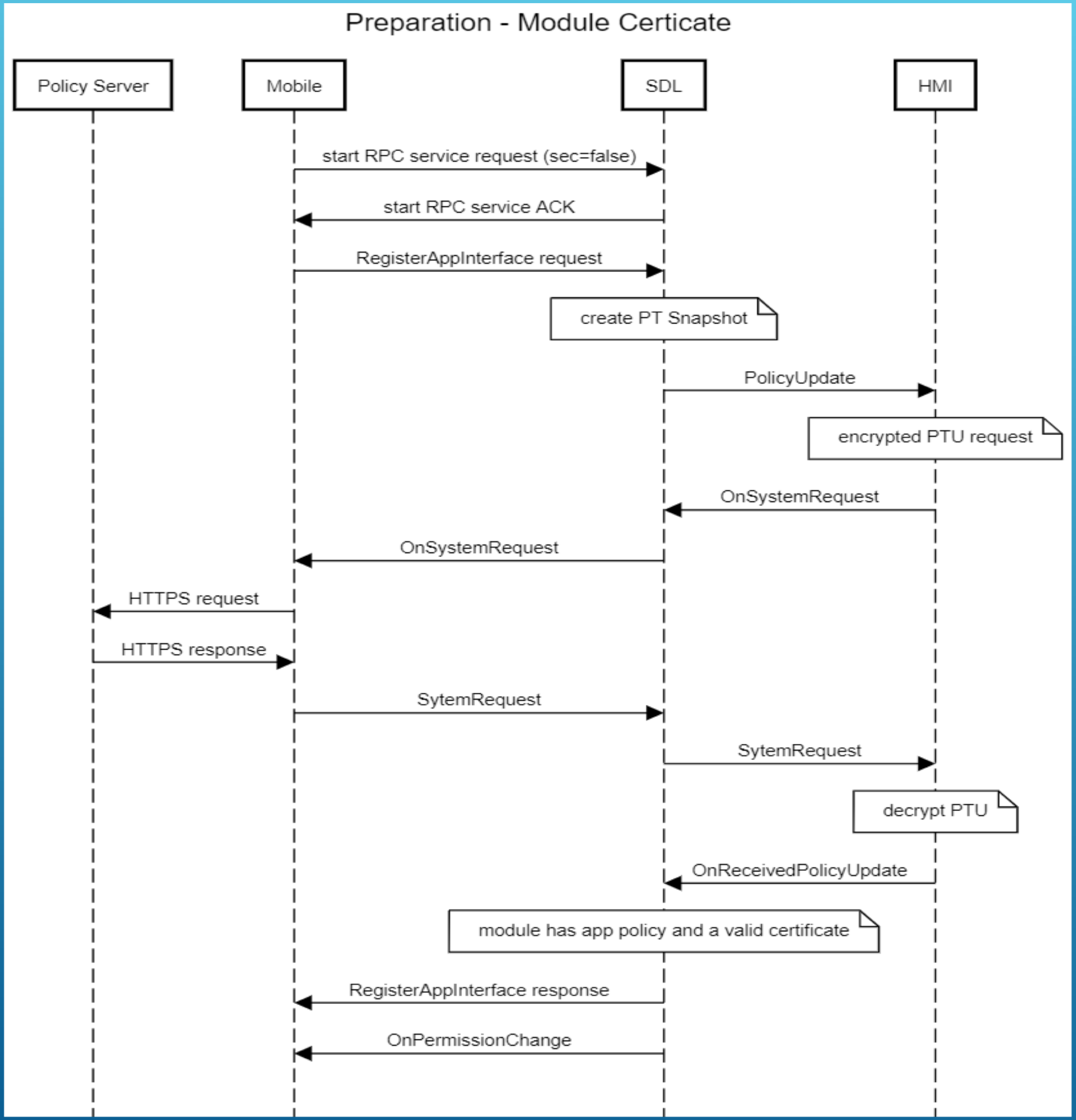
RPC MESSAGE PROTECTION AND SECURE SERVICES IN SDL

Zhimin Yang

Currently – How secure video and audio services work

- Step1: Prepare module certificate (valid for a long time)
- Step2: Get app certificate (short time)
- Step3: TLS or DTLS handshake
- Step4: Start/Enable secure service
- Step5: Start sending encrypted video / audio data

Module Cert.



Preparation - Module Certicate

Policy Server

Mobile

SDL

HMI

start RPC service request (sec=false)

start RPC service ACK

RegisterAppInterface request

create PT Snapshot

PolicyUpdate

encrypted PTU request

OnSystemRequest

OnSystemRequest

HTTPS request

HTTPS response

SytemRequest

SytemRequest

decrypt PTU

OnReceivedPolicyUpdate

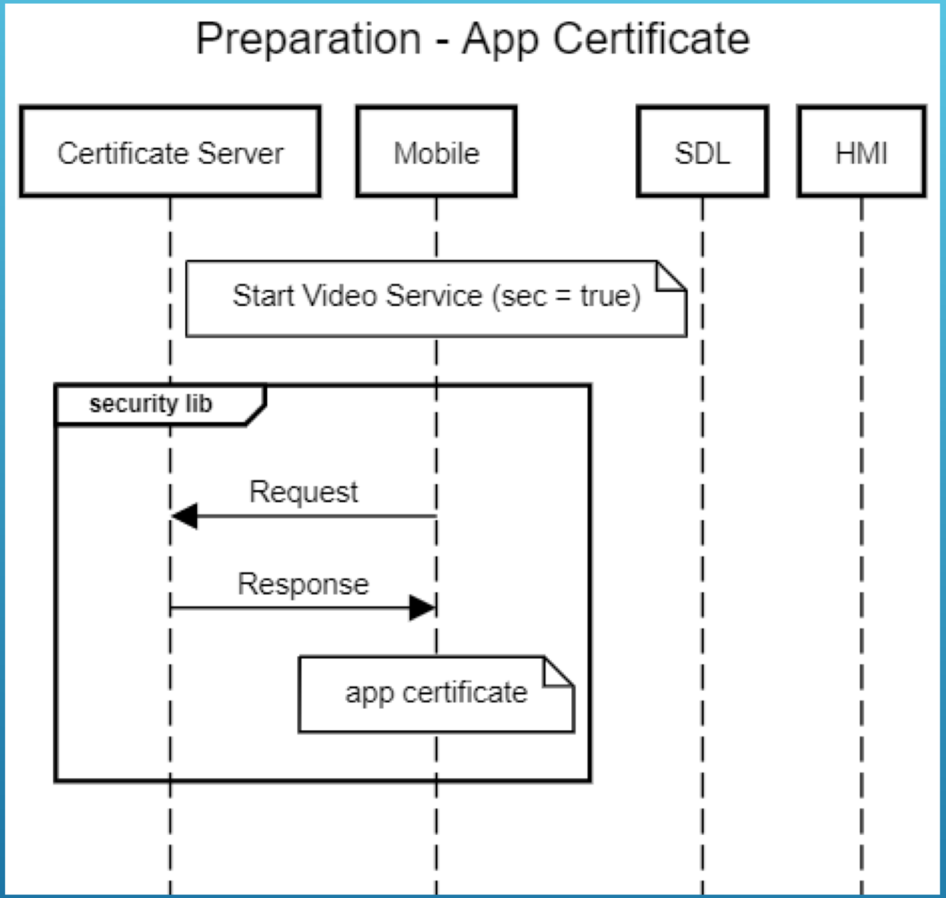
module has app policy and a valid certificate

RegisterAppInterface response

OnPermissionChange

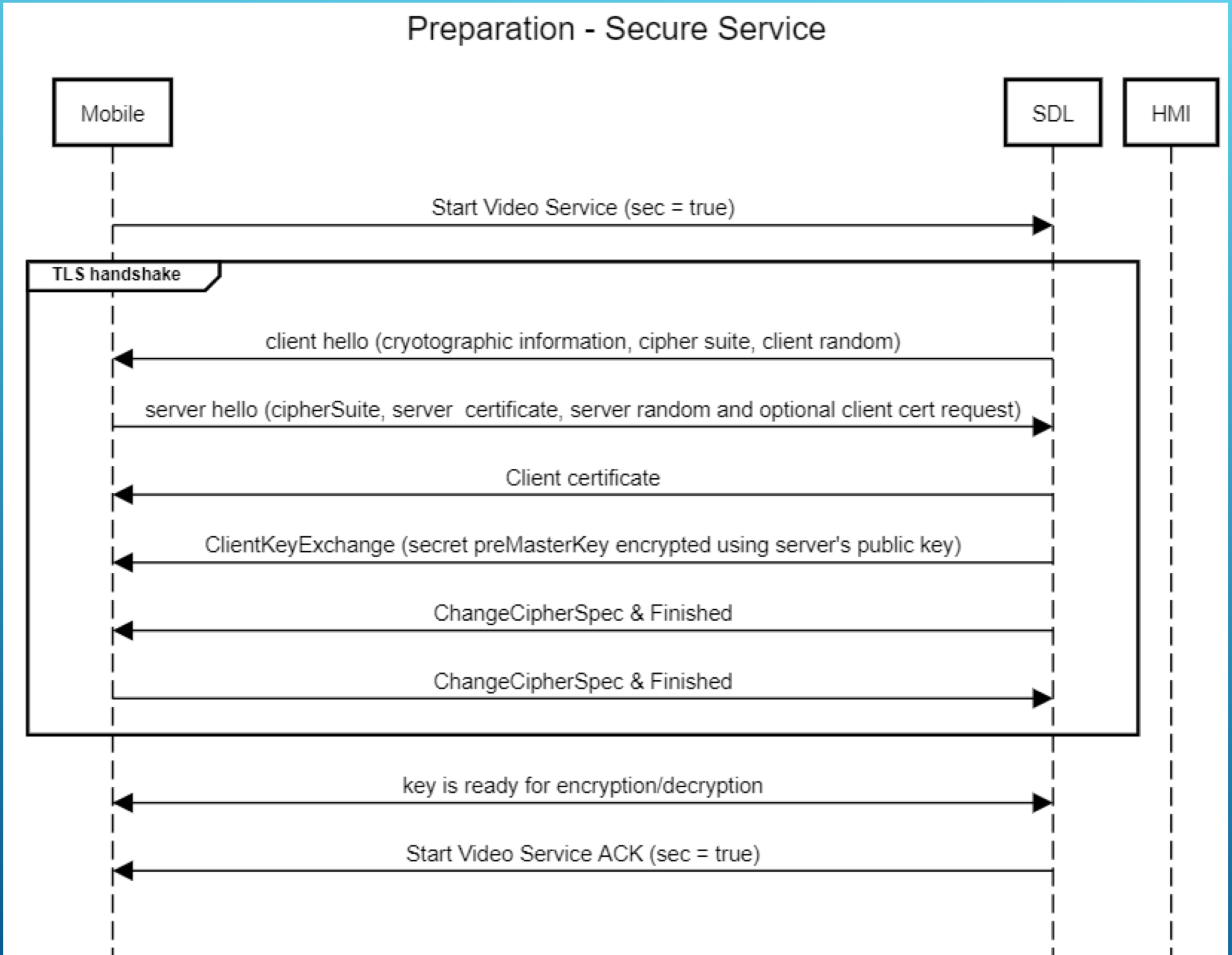
- Start control service 0 is not illustrated in the flow
- Start service request is on control service
- RPCs are on RPC service
- Module cert valid for a long time
- PTU does not happen very time an app registers

App Cert.



- App has the function call to proxy with secure flag turned on
- Everything else is handled by security lib : download an app cert, TLS handshake, encryption and decryption
- App cert is valid for a few days

Secure Service.



Preparation - Secure Service

Mobile

SDL

HMI

Start Video Service (sec = true)

TLS handshake

client hello (cryotographic information, cipher suite, client random)

server hello (cipherSuite, server certificate, server random and optional client cert request)

Client certificate

ClientKeyExchange (secret preMasterKey encrypted using server's public key)

ChangeCipherSpec & Finished

ChangeCipherSpec & Finished

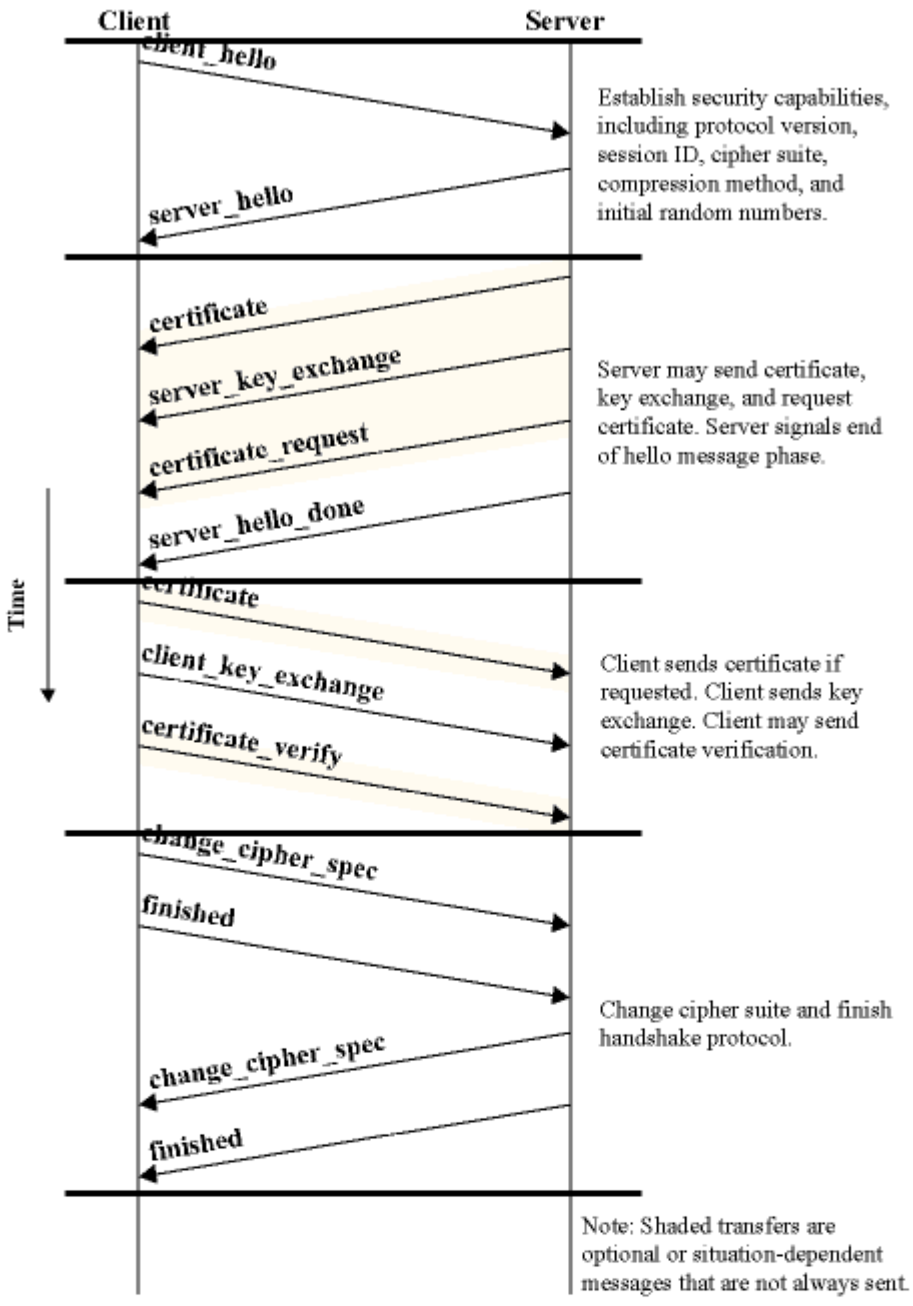
key is ready for encryption/decryption

Start Video Service ACK (sec = true)

TLS handshake happens in the control service

SDL authenticates the app id

TLS handshake.



Establish security capabilities, including protocol version, session ID, cipher suite, compression method, and initial random numbers.

Server may send certificate, key exchange, and request certificate. Server signals end of hello message phase.

Client sends certificate if requested. Client sends key exchange. Client may send certificate verification.

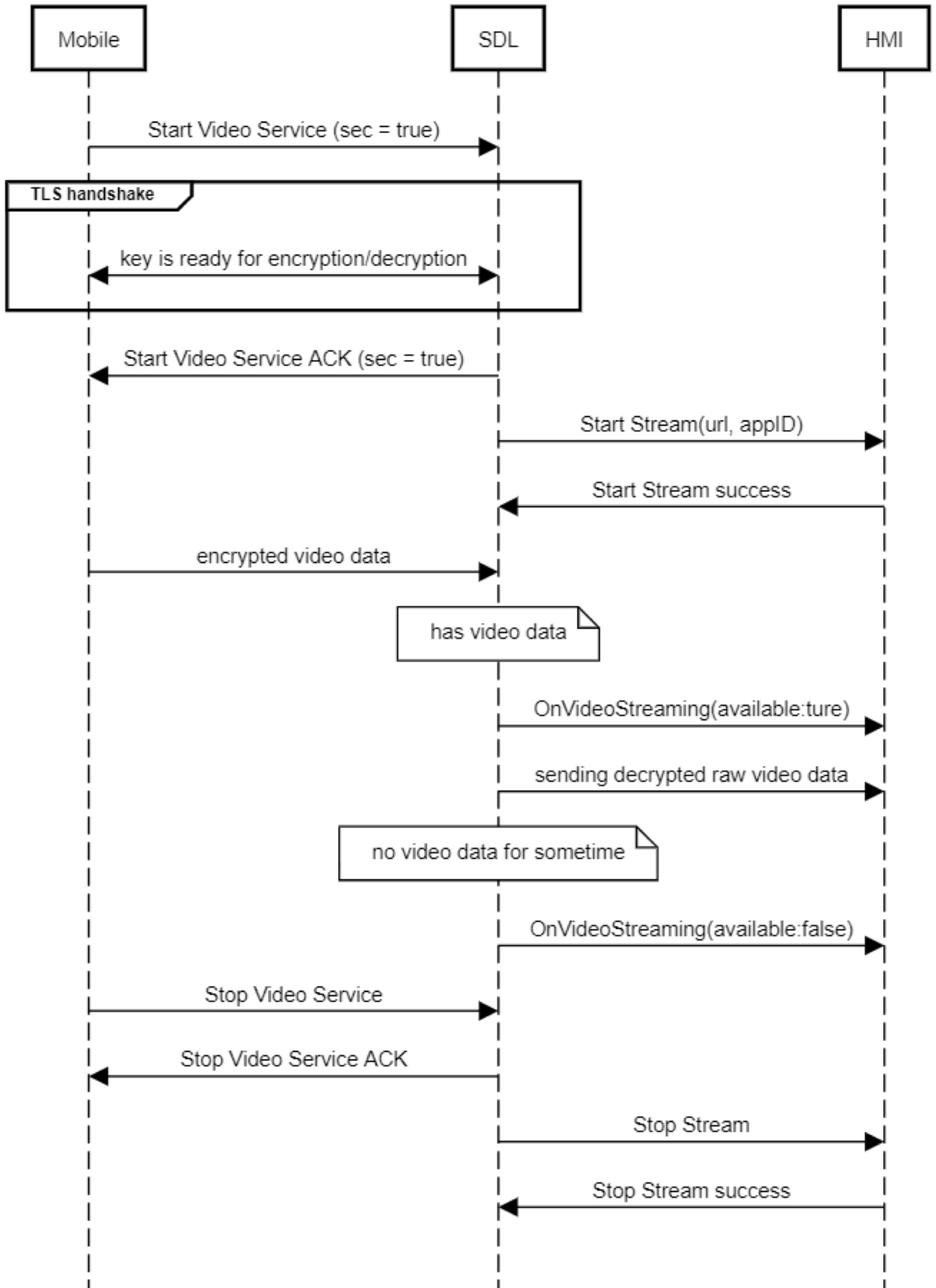
Change cipher suite and finish handshake protocol.

Note: Shaded transfers are optional or situation-dependent messages that are not always sent.

Figure 14.6 Handshake Protocol Action

Secure Video Service.

Secure Video Service



Mobile

SDL

HMI

Start Video Service (sec = true)

TLS handshake

key is ready for encryption/decryption

Start Video Service ACK (sec = true)

Start Stream(url, appID)

Start Stream success

encrypted video data

has video data

OnVideoStreaming(available:ture)

sending decrypted raw video data

no video data for sometime

OnVideoStreaming(available:false)

Stop Video Service

Stop Video Service ACK

Stop Stream

Stop Stream success

The Secure RPC Service in SDL

Functionality already exists and is implemented in SDL Core

- Utilizes the same common security model as Audio and Video services (previously discussed)
- Security model is already in production and utilized by existing navigation partners.
- Current SDL security model requires some RPC’s to be exchanged in unencrypted manner.
- Secure RPC service can be opened after an unsecure RPC service is established.

The Secure RPC Service in SDL Continued

Proposed Enhancements to Secure RPC service handling

- Current model allows an app to freely send encrypted RPC's after the secure RPC service has been opened
- SDL implementation does not provide capability to allow an OEM to require encryption of specific RPC's.
- OEM's need a way to notify apps that specific RPC's require encryption.
- OEM's also need a way to enforce specific RPC's to be sent and received with encryption on.
- Recommended proposed solution is an incremental enhancement over the existing security model.

Our goal or problem to solve

- Protect certain high risk RPC messages so that they cannot be forged
  - From an authenticated app
  - Not modified

Messages	Risk
Messages that change vehicle state (Remote control)	High
Messages that transmit read-only Personal Identifiable Information	Medium
Messages that transmit read-only vehicle data	Medium
Other Messages	Low

Security/Encryption – necessary or not

- Q: Some transports already provide encryption, for example BT and MFi chip, does SDL still need encryption?
- Yes for security (boarder than just encryption)
- Transport security cannot be assumed. TCP is clearly not encrypted for example. BT is not that secure as we thinks, there are some interesting reading about BT security
  - https://www.macrumors.com/2018/07/24/apple-fixed-bluetooth-security-vulnerability/
  - https://www.schneier.com/blog/archives/2017/09/bluetooth_vulne.html
  - https://armis.com/blueborne/
- Even Transport level is secure, that's only transport-to-SDL, still need end(app)-to-end(SDL) encryption.

It is not just encryption – It is also Authentication

- Mutual authentication

  - App authentication

  Modules only accept a request from a known and authenticated app, and only send information to an authenticated app

  - Module authentication

  App only send information to an authenticated module
- TLS handshake does certificate verification to check the identity of app and/or module

Preferred approach : Re-use existing security workflow, extend policies providing dynamic flexibility to OEM's

- Policy update tells SDL which RPCs for which app need protection
- SDL forward the info to Apps (proxy) via OnPermissionChanged
- Proxy starts encryption

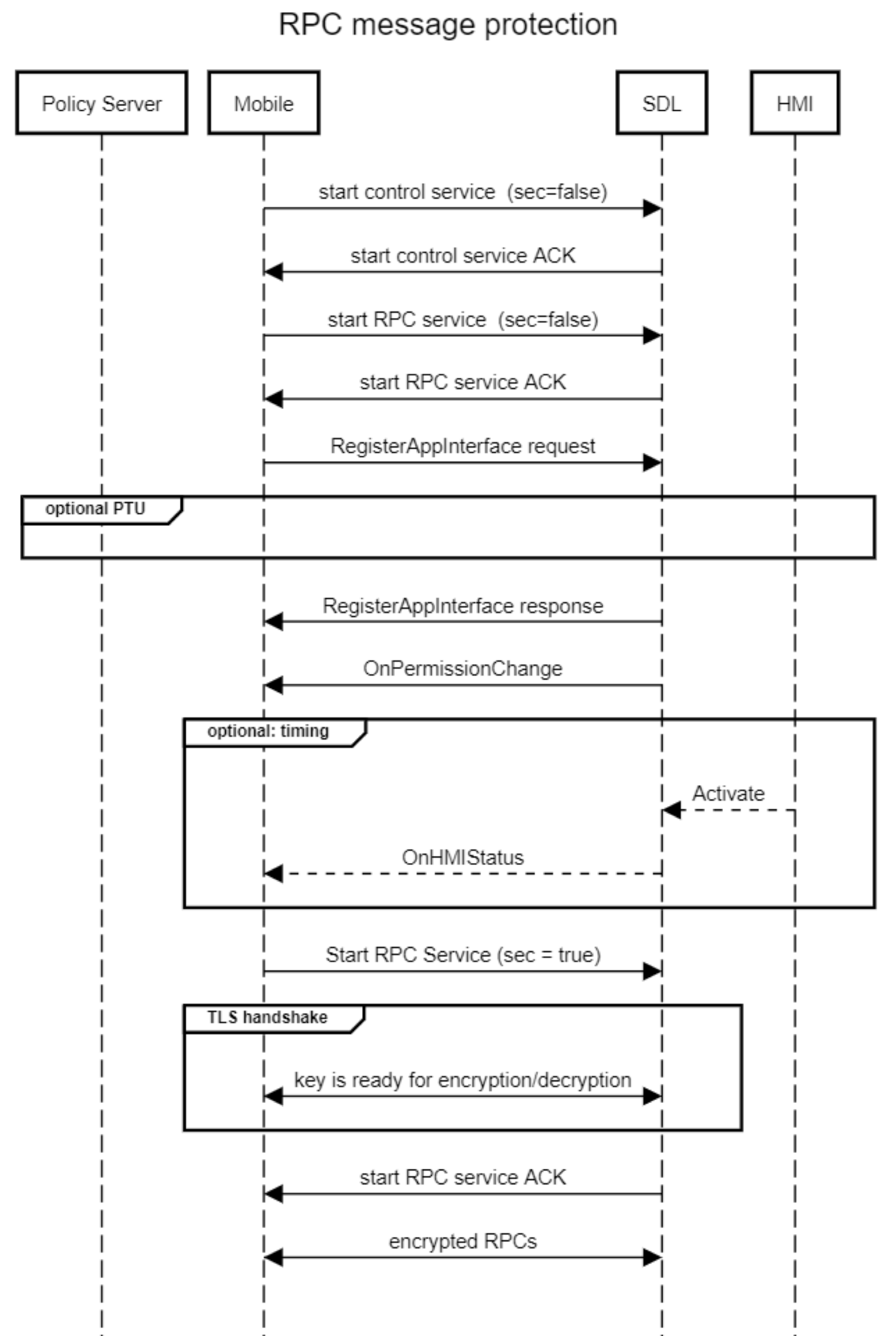
Ground up redesign SDL security	Reuse secure video workflow
All traffic (except control messages) can be encrypted All RPC messages can be encrypted from the beginning	RegisterAppInterface, OnSystemRequest, SystemRequest cannot be encrypted from the beginning, encryption enabled only after certain point in the flow
Hard to design, prove and make sure the new design is secure	Is in Production already Small incremental change needed
Time consuming to design, implementation and verify	Ready to start
Lots of work	Less work
Not backward compatible with existing mobile Navigation apps	backward compatible

Another acceptable approach : Re-use existing security workflow, utilize the SDL Core INI file

The current security model allows an OEM to specify which services (NAV,PCM) require protection, allowing OEM's to have flexibility to choose what services require encryption.

- This concept can be extended one step further to provide an OEM with the flexibility to choose which RPC's require encryption via the SDL Core INI file.
- This model is not as flexible as the policies recommendation, however it is a sufficient method to ensure specific RPC's are encrypted and continues to follow the existing security model implemented in SDL.
- This model has the benefit of not requiring any policies side changes.

Our design.


RPC message protection
Policy Server
Mobile
SDL
HMI
start control service (sec=false)
start control service ACK
start RPC service (sec=false)
start RPC service ACK
RegisterAppInterface request
optional PTU
RegisterAppInterface response
OnPermissionChange
optional: timing
Activate
OnHMIStatus
Start RPC Service (sec = true)
TLS handshake
key is ready for encryption/decryption
start RPC service ACK
encrypted RPCs

Our design for RPC protection.

	Secure video	Secure RPC
Step1: Prepare module certificate (secure by OEM encryption)	same	same
Step2: Get app certificate (secure by OEM security lib)	same	same
Step3: TLS or DTLS handshake (standard)	same	same implementation (RPC need TLS)
Step4: Start/Enable secure service	Service 11	Service 7
Step5: Start sending encrypted video data	Video data	RPC data

- TLS handshake guarantees that the app is authenticated
- Encryption add additional security – but may be optional if risk accepted
- The only difference that matters is the data to be encrypted/decrypted
- Policy update tells which RPCs for which app need protection

What need to be protected/encrypted

- Option 1: Most RPCs (as some exceptions exist) of an app – the so called ALL approach – in reality Most.
  - All RPC's cannot be encrypted with existing SDL Security Model (RAI, SystemRequest).
- Option 2: OEM selected RPCs of an app – our approach recommended.
  - OEM select RPC's to encrypt based on policies or ini config.
- Similar extensions to the policy
  - Option1: one Boolean flag per app
  - Option2: one Boolean flag per RPC per app, thus multiple flags per app

Our proposal: – OEM' Decide what RPC's to Encrypt

Advantages

- SDL core has logic to check each RPC message against HMI level, policy allowance, mobile API version, etc. it is easy to add an additional check for RPC protection
- It gives OEM flexibility to select messages to protect, OEM can gradually increase their protected RPC list, OEM controls what and when
- It includes "ALL" capability, It allows OEM to transit to All approach smoothly, when OEM is ready, OEM control the pace
- It includes "Zero" capability, Compatible to existing system and apps, good for OEMs that do not have security implemented
- May Have the option to fall back to authentication only

Disadvantages

- Proxy needs to maintain protected RPC list and check against the list, (proxy already check mobile API versioning of each RPC?)
- Complexity in policy server, however not required in INI approach
- OEM disparity

All Approach – Most RPC’s are encrypted, OEM’s lose flexibility to choose

Advantages

- It is still a policy configuration extension per app as our approach
- Compared to dynamic policies approach, it is less complex: One flag per app vs multiple flags per app
- It includes “Zero” capability, Compatible to existing system and apps, good for OEMs that do not have security implemented

Disadvantages

- Many OEM’s may not be ready to encrypt ALL RPC’s.
- No flexibility to just protect what’s needed by the OEM, force OEM to choose none or all
- Not possible to do authentication only
- Is not more secure than our recommended approach (in terms of how to attack, or security analysis)
- OEM disparity still in in app level, not RPC level

Production Requirements

- Some RPC messages have more risk than other RPCs.
- Ford has the emergent need to encrypt RC related RPC messages for the production
- Ford may encrypt all RPC messages when it is ready
- not ready yet, lessons learned from previous SYNC3 experience
  - We turn off video packet encryption for iOS mobile-navigation applications due to the low performance of encryption (with app authentication still on)
  - Problems met when enable RPC encryption in the past?
  - It is not the performance to encrypt/decrypt, it is the overall impact to the system

Conclusion

- We support the flexibility (at the cost of complexity)