SPECIFICATION

UWB FiRa Protocol

Qorvo

Release R12.7.0-405-gb33c5c4272

Contents

1 Glossary

ADC
Analog Digital Converter

AEAD
Authenticated Encryption with Associated Data

AES
Advanced Encryption Standard

AoA
Angle of Arrival

AIDL
Android Interface Definition Language

AOSP
Android Open Source Project

APDU
Application Protocol Data Unit

API
Application Programming Interface

BPRF	Base Pulse Repetition Frequency
CAP	Contention Access Period
CCC	Car Connectivity Consortium
CCM	Counter with Cipher Block Chaining Message Authentication Code
CFO	Clock Frequency Offset
CFP	Contention Free Period
CIR	Channel Impulse Response
CM	Control Message
CMAC	Cipher-Based Message Authentication Code
CRUM	Control Update Message
DL-TDoA	Downlink TDoA
DPF	Data Packet Format
DRBG	Deterministic Random Bit Generator
DS	Device Specific
DS-TWR	Double-Sided Two-Way Ranging
DTM	Downlink TDoA Message
DUT	Device Under Test
ECB	Electronic Code Book
EVB	Evaluation Board
FBS	FiRa Based Session
FoM	Figure of Merit
FP	First Path

GID	Group IDentifier
GPIO	General Purpose Input Output
HAL	Hardware Abstraction Layer
HIE	Header Information Element
HPRF	High Pulse Repetition Frequency
I2C	Inter-Integrated Circuit
IE	Information Element
IFI	Inter-Frame-Interval
IFI_GT	Inter-Frame-Interval Guard Time
IFI_BGT	Inter-Frame-Interval Block Guard Time
KDF	Key Derivation Function
HUS	Hybrid UWB Scheduling
L1	Layer One
LNA	Low Noise Amplifier
LLHW	Low Level Hardware
LUT	Lookup Table
LO	Local Oscillator
IV	Initialization Vector
MAC	Medium Access Control
MCU	MicroController Unit
MHR	MAC Header
MRM	Measurement Report Message

MRP	Measurement Report Phase
MTI	Moving Target Indicator
NA	Not Applicable
NL	NetLink
NONCE	Number used Once
OID	Opcode IDentifier
OOB	Out-Of-Band
OUI	Organizationally Unique Identifier
OWR	One-Way Ranging
PA	Power Amplifier
PDoA	Phase Difference of Arrival
PHR	Physical Layer Header
PHY	Physical Layer
PIE	Payload Information Element
PRF	Pulse Repetition Frequency
PSDU	PHY Service Data Unit
PSR	Preamble Symbol Repetitions
RAM	Random Access Memory
RCP	Ranging Control Phase
RDS	Ranging Data Set
RFFE	Radio Frequency Front End
RFM	Ranging Final Message

RFP	Ranging Final Phase
RFRAME	Ranging Frame
RFU	Reserved for Future Use
RIM	Ranging Initiation Message
RIP	Ranging Initiation Phase
RP	Ranging Phase
RRM	Ranging Response Message
RRP	Ranging Response Phase
RRRM	Ranging Result Report Message
RSL	Received Signal Level
RSSI	Received Signal Strength Indicator
S1	ACPI power state corresponding to CPU Idle
S3	ACPI power state corresponding to suspend to RAM
Sample	One complex value from CIR array
SE	Secure Element
SHR	Synchronization Header
SIP	System In a Package
SNR	Signal to Noise Ratio
SOC	System On a Chip
SPI	Serial Peripheral Interface
SS-TWR	Single-Sided Two-Way Ranging
STS	Scrambled Timestamp Sequence

SUS	Secure UWB Service
SYNC	Synchronization Preamble Sequence
TDoA	Time Difference of Arrival
TLV	Type Length Value
ToA	Time of Arrival
ToF	Time of Flight
TWR	Two Way Ranging
UCI	UWB Subsystem Command Interface
UL-TDoA	Uplink TDoA
UTM	Uplink TDoA Message
UWB	Ultra-Wide Band
UWBS	Ultra Wide-Band Subsystem

2 One-Way Ranging (OWR)

2.1 UpLink Time Difference of Arrival (UL-TDoA)

2.1.1 Overview

OWR *UL-TDoA* is a system designed to accurately determine the position of moving objects. The main use case is to monitor in real time the location of physical assets such as company's equipment, machinery or vehicles.

As shown in Fig. 2.1, it is built upon three key components: UT-Tags, UT-Anchors and a Central Location Engine.

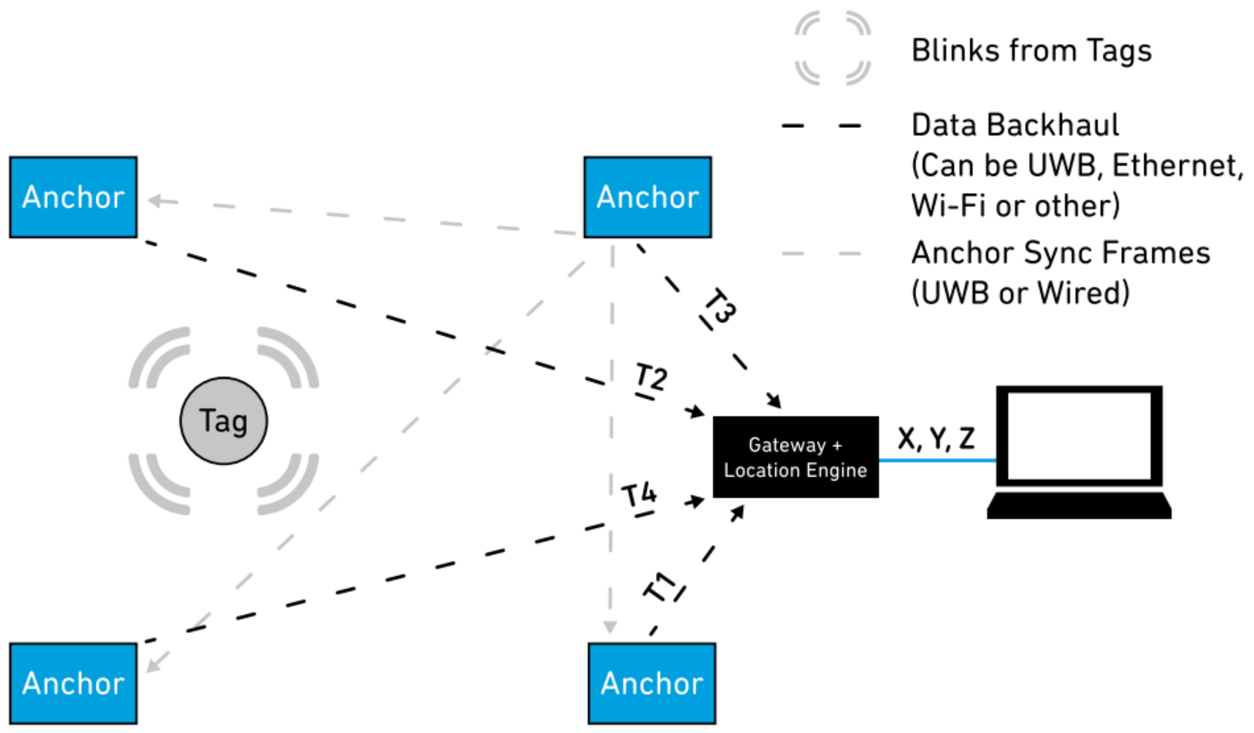
Blinks from Tags
Data Backhaul (Can be UWB, Ethernet, Wi-Fi or other)
Anchor Sync Frames (UWB or Wired)
Anchor
Anchor
Tag
T3
T2
T4
T1
Gateway + Location Engine
X, Y, Z
Anchor
Anchor

Fig. 2.1: UL-TDoA system overview

Note: Picture Fig. 2.1 is taken from *Ultra-Wideband For Dummies* e-book (https://www.qorvo.com/design-hub/ebooks/ultra-wideband-for-dummies).

UT-Tags are portable devices that periodically transmit Blink messages, enabling them to be located by the UT-Anchors infrastructure. These tags are designed to be highly energy-efficient, as they can remain in an idle state for the majority of the time. They only activate briefly to transmit their signal at a predetermined interval.

UT-Anchors are stationery devices that listen for Blink messages from UT-Tags and report their timestamps to the Central Location Engine.

The **Central Location Engine** calculates the position of the UT-Tags based on the data received from the UT-Anchors.

2.1.2 Functional Description

This chapter contains the functional description of the *UL-TDoA* system.

Note: *UL-TDoA* has not been integrated yet in the official version of the FIRA specification. The FIRA requirements for *UL-TDoA* are still a subject to change and so is this description. This is especially true when it comes to UT-Anchors which are not clearly defined in FIRA CRs so this chapter focuses only on UT-Tags.

2.1.2.1 UT-Tag configuration

To initialize and configure the device to act as a UT-Tag, the Host shall send *SESSION_INIT_CMD* and *SESSION_SET_APP_CONFIG_CMD* as illustrated in Fig. 2.2 (just like for all the other ranging methods). All the relevant session parameters are gathered in Table 2.1.

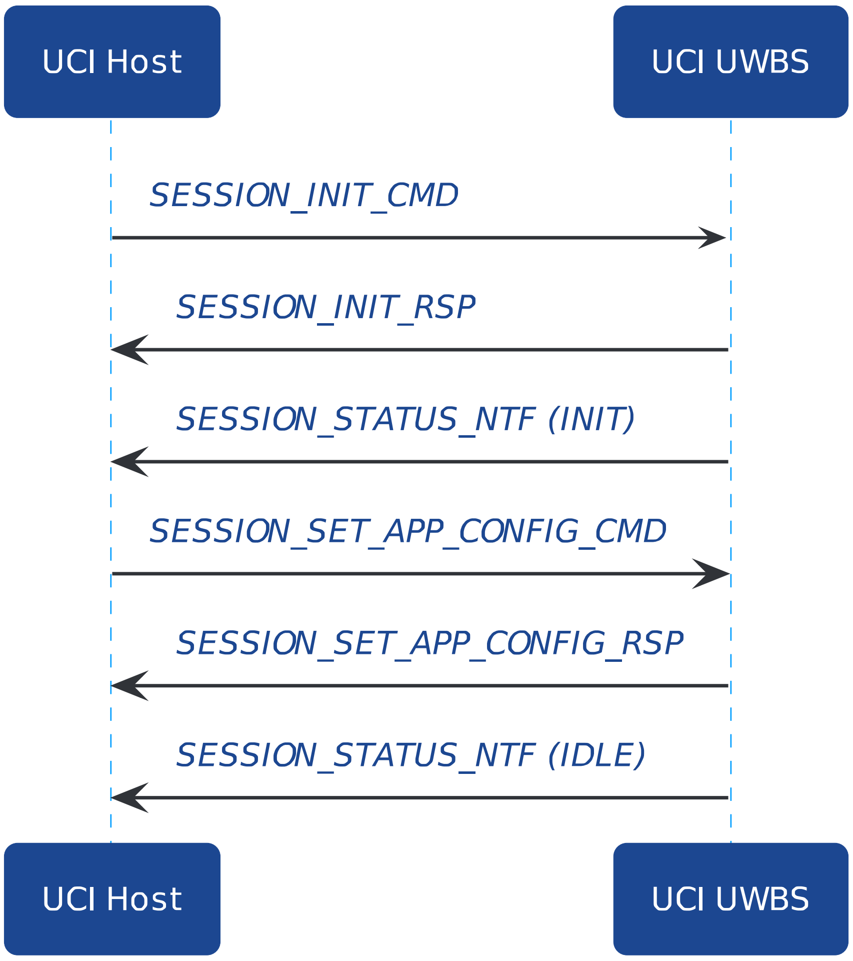
UCI Host
UCI UWBS
SESSION_INIT_CMD
SESSION_INIT_RSP
SESSION_STATUS_NTF (INIT)
SESSION_SET_APP_CONFIG_CMD
SESSION_SET_APP_CONFIG_RSP
SESSION_STATUS_NTF (IDLE)
UCI Host
UCI UWBS

Fig. 2.2: *UCI* messages for UT-Tag session configuration

Table 2.1: Relevant session parameters for UT-Tag

Tag	Name	Length	Default value	Required value
0x01	RANGING_ROUND_USAGE	1	*NA*	0x00 (One Way Ranging UL-TDoA)
0x02	STS_CONFIG	1	0x00 (Static STS)	0x00 (Static STS)
0x03	MULTI_NODE_MODE	1	*NA*	0x01 (One to Many)
0x04	CHANNEL_NUMBER	1	9	X
0x06	DEVICE_MAC_ADDRESS	1	*NA*	X
0x0B	MAC_FCS_TYPE	1	0x00 (CRC 16)	X
0x11	DEVICE_ROLE	1	*NA*	0x04 (UT-Tag)
0x12	RFRAME_CONFIG	1	0x03 (SP3)	0x01 (SP1)
0x14	PREAMBLE_CODE_INDEX	1	10	X
0x15	SFD_ID	1	2	X
0x16	PSDU_DATA_RATE	1	0x00 (6.81 Mbps)	X
0x17	PREAMBLE_DURATION	1	0x01 (64 symbols)	X
0x18	LINK_LAYER_MODE	1	0x00 (Bypass Mode)	0x00 (Bypass Mode)
0x1F	PRF_MODE	1	0x00 (BPRF mode)	X
0x22	SCHEDULE_MODE	1	*NA*	0x01 (Time-scheduled ranging)
0x25	SESSION_PRIORITY	1	50	X
0x26	MAC_ADDRESS_MODE	1	0x00 (2 bytes)	X
0x27	VENDOR_ID	2	*NA*	X
0x28	STATIC_STS_IV	6	*NA*	X
0x29	NUMBER_OF_STS_SEGMENTS	1	0x00	0x01-0x04
0x2B	UWB_INITIATION_TIME	8	0	X
0x31	BPRF_PHR_DATA_RATE	1	0x00 (850 kbps)	X
0x33	UL_TDOA_TX_INTERVAL	4	2000 (ms)	X
0x34	UL_TDOA_RANDOM_WINDOW	4	0 (no randomization)	X
0x35	STS_LENGTH	1	0x01 (64 symbols)	X
0x38	UL_TDOA_DEVICE_ID	1/3/5/9	0x00 (not included)	X
0x39	UL_TDOA_TX_TIMESTAMP	1	0x00 (not included)	X

2.1.2.2 UT-Tag specific behaviour

UT-Tags transmit Blink Uplink TDoA Messages (Blink UTMs) periodically with a period configured by *UL_TDOA_TX_INTERVAL* session parameter. Additionally a randomization window might be set at the beginning of this interval to reduce the likelihood of continuous collisions. The length of this randomization window is configured by *UL_TDOA_RANDOM_WINDOW* session parameter. This is illustrated in Fig. 2.3.

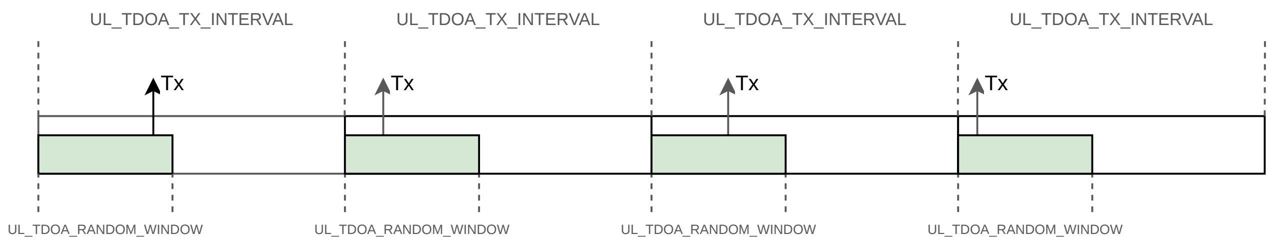

Fig. 2.3: UT-Tag timing structure

2.1.2.3 Blink Uplink TDoA Message description

Blink *UTM* always contains Frame Number field and may also contain some additional information such as UL-TDoA Device ID or Tx Timestamp in the local time domain. The presence of these fields is configurable with *UL_TDOA_DEVICE_ID* and *UL_TDOA_TX_TIMESTAMP* session parameters. The Frame Number is the local *UTM* number which shall be incremented by the sender after every *UTM* transmission. The UL-TDoA Device ID shall contain a unique ID assigned to the sender of the message. Its length and value are configurable with *UL_TDOA_DEVICE_ID* session parameter. Both fields are part of the Payload IE which is encrypted, so they can't be decoded without knowing the session key.

All UTMs comply with the following requirements:

- They shall be SP1 Multipurpose MAC Frames.
- They shall not contain a Header *IE*.
- The MAC Header shall be terminated with Header Termination *IE* and the Payload IEs shall be terminated with Payload Termination *IE*.
- They shall not include a Destination Address field in the MAC Header, i.e. they are considered as implicit broadcast frames.
- They shall include a Source Address field in the MAC Header, currently only short addressing is supported.
- Static *STS* shall be used for *STS* generation and key derivation. For the purpose of the Configuration Digest computation, the *SLOT_DURATION* parameter should be set to 0x0000.

3 STS configurations

3.1 Overview

[IEE20b] introduces the concept of *STS* to authenticate the emitter of a frame, but it does not specifies how to authenticate and encrypt the content of the frame itself and how to make it change frame to frame. FiRa attempts to improve those points, and thus the security of the transactions, by regrouping the following two elements when it uses the concept of *STS* configurations:

- It specifies how to build the *STS* located in the *SHR* of a frame as described in [IEE20b]
- It specifies different mechanisms to guarantee the authenticity and confidentiality of the *HIE*/*PIE* parts of a frame.

Fig. 3.1 illustrates the interactions among the end-user and the software systems involved in a basic FiRa *TWR*.

The addition of the FiRa to the security of the exchanges is shown in the figure with the external attacker located to the right which is trying to eavesdrop the FiRa exchanges between the two authenticated participants. The encryption mechanisms brought by the FiRa *STS* configuration prevents this case.

The attacker might also try to replicate and inject *UWB* frames to perform man in-the-middle attack (a very common attack in *UWB* systems) where the unauthorized device shortens the expected distance between authenticated participants.

For more details about general *UWB* security mechanisms, please refer to general_functionalities/security/sts. For more details about FiRa *TWR* or *OWR* mechanisms and the frame exchanges they induce, please refer to *Two-Way Ranging (TWR)*.

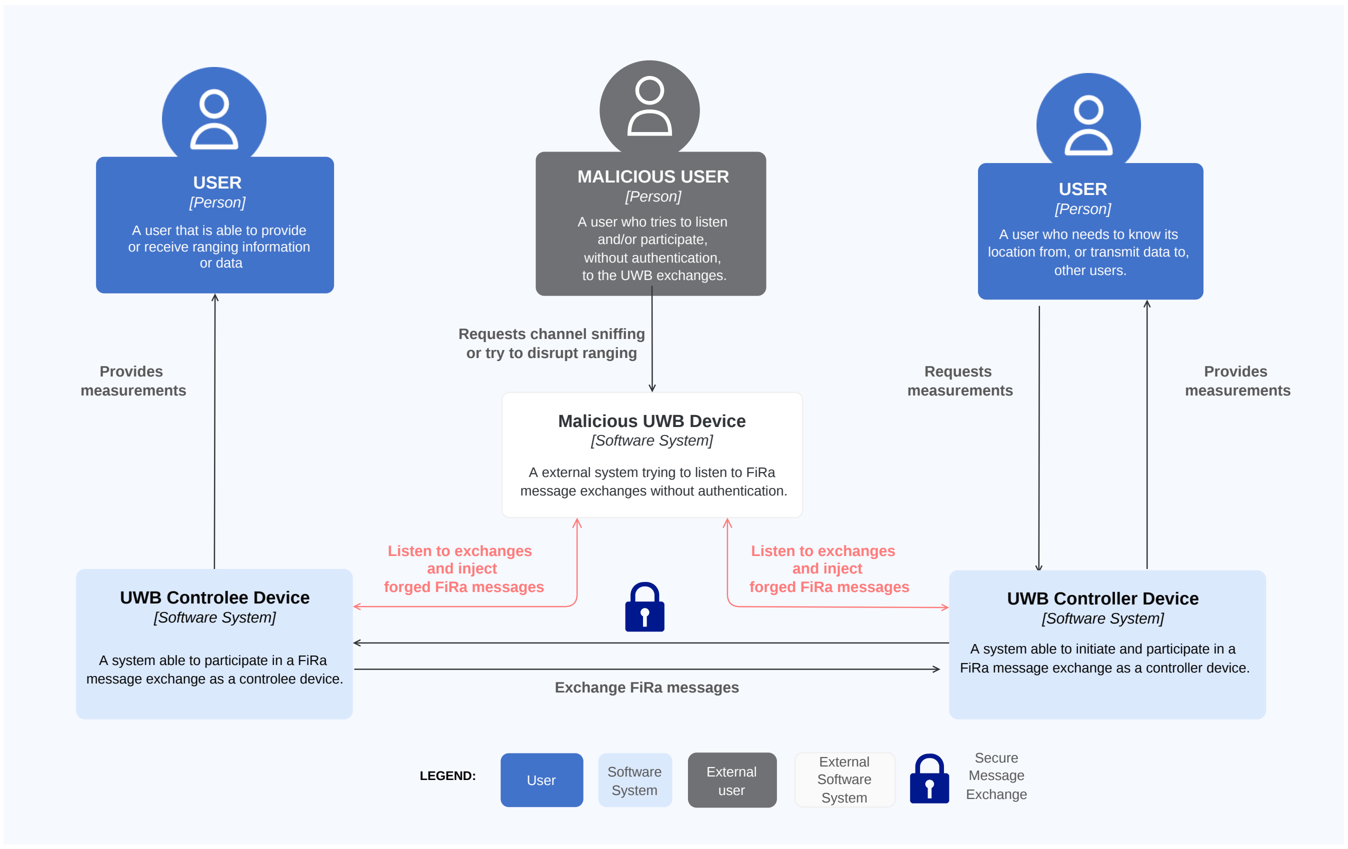
USER
[Person]
A user that is able to provide or receive ranging information or data
MALICIOUS USER
[Person]
A user who tries to listen and/or participate, without authentication, to the UWB exchanges.
USER
[Person]
A user who needs to know its location from, or transmit data to, other users.
Provides measurements
Requests channel sniffing or try to disrupt ranging
Requests measurements
Provides measurements
Malicious UWB Device
[Software System]
A external system trying to listen to FiRa message exchanges without authentication.
Listen to exchanges and inject forged FiRa messages
Listen to exchanges and inject forged FiRa messages
UWB Controlee Device
[Software System]
A system able to participate in a FiRa message exchange as a controlee device.
UWB Controller Device
[Software System]
A system able to initiate and participate in a FiRa message exchange as a controller device.
Exchange FiRa messages
LEGEND:
User
Software System
External user
External Software System
Secure Message Exchange

Fig. 3.1: FiRa *TWR* ranging system through context diagram

3.1.1 The different STS configurations specified by FiRa and supported by QM33.

FiRa has defined a set of standardized configurations that can be used to ensure different levels of protection against possible attacks. For *STS* generation, FiRa has defined a set of *AES* operations and crypto-derived materials used to provide *STS* variability by changing the 128b key and the 128b *IV* mentioned in general_functionalities/security/sts. The same set of cryptographic assets is used to perform *HIE*/*PIE* encryption and ensure slot synchronization between participants in a ranging session.

These specific *STS* configurations defined in [FiR23b] are:

Static *STS* (mandatory for any FiRa device)
: In this configuration, the system uses a pre-known static set of cryptographic assets for all participants and does not feature anti-rollback mechanisms neither strong encryption. This *STS* configuration is not considered secured and shall not be used for sensitive operations, but it allows a fast implementation and an immediate time synchronization.

Provisioned *STS*
: This configuration does not mandate the negotiation of cryptographic assets through a secured *OOB* channel, neither the direct connection between the *UWBS* and *SE*. The cryptographic keys are provided through *UCI* parameters. This *STS* configuration is considered as secure.

Provisioned *STS* with Responder-specific sub-session key
: This configuration shares the same principles as the Provisioned *STS* but it mandates the use of different cryptographic keys for each participant. This *STS* configuration is considered secured.

The *STS* configuration of a FiRa region shall be chosen depending on device capabilities and the desired level of security. This choice is usually made by an external system in charge of negotiating the session parameters. It performs this operation through *OOB* communication and takes into account the level of security required by the service and the *STS* related capabilities of all the session participants.

3.2 Functional Description

3.2.1 STS configurations principles

Amongst the different *STS* configurations defined by FiRa (see *Overview*), the *UWBS* supports the following:

- Static *STS*
- Provisioned *STS*
- Provisioned *STS* with sub-session keys

In the following sections, some details about how the *STS* is handled by the stack are given. They shall be read in conjunction with [FiR23b] and [IEE20b].

3.2.1.1 Crypto assets management

In order to build *STS*, the *UWBS* uses three parameters, as explained in [IEE20b], that are needed by the *PHY* to trigger *STS* generation:

- `phyHrpUwbStsVUpper96`
- `phyHrpUwbStsVCounter`
- `phyHrpUwbStsKey`

However, the *PHY* chapter of this specification does not provide any information on how to compute those parameters and make them vary over time, this is left to protocol implementation.

In FiRa, those derivation mechanisms vary depending on the *STS* configuration. They are based on several crypto assets (keys, counters, etc.) which are computed during operation (Fig. 3.6 and Fig. 3.4 illustrate the computation of those assets in Provisioned *STS* and Static *STS* respectively). All the processes illustrated in those diagrams are performed by the *MAC* layer during the session starting phase or during controlee addition into the session if a sub-session has been requested. When these crypto assets are available and configuration is properly validated, the session is able to start.

Nevertheless, as each asset matches a specific purpose, if one of them fails during calculation, the remaining computation pipeline is aborted, the session is not started and the Host *UWB* service is informed.

3.2.1.1.1 Crypto keys

Amongst the needed crypto assets to build *STS*, there is several cryptographic keys that are described in the following sections.

3.2.1.1.2 SecSessionKey

It is an *AES* 128 bits or 256 bits key used to secure a *UWB* Session. It might be retrieved from different sources depending on the *STS* configuration chosen and it serves as a root key for all other crypto assets, hence, its confidentiality is vital if a secure approach is required.

3.2.1.1.3 SecDataProtectionKey

`secSessionKey` is never directly used in the key derivation process. Instead, an intermediate `secDataProtectionKey` is derived from it. This allows, for example, in certain *STS* configurations, to avoid storing the `secSessionKey` in the *UWBS*.

`secDataProtectionKey`, as shown in Fig. 3.2, is derived using the `CMAC-AES(secSessionKey, Counter|Label|Context|OutputLength)` function, where `Counter` is set to `0x01` or `0x02` depending on the iteration, `Label` is set to `DataPrtK` and `OutputLength` is set to either 128 or 256. `Context` is directly obtained from `configDigest`.

Note: `secDataProtectionKey` has the same length as the `secSessionKey`. If a 256 bits `secSessionKey` key is used, then two *KDF* iterations are required and the results are concatenated into one output key.

3.2.1.1.4 SecDataPrivacyKey

`secDataPrivacyKey` is obtained using the `CMAC-AES(secSessionKey, Counter|Label|Context|OutputLength)` function, where `Counter` is set to `0x01`, `Label` is set to `PrivacyK` and `OutputLength` is set to 128. `Context` is the same as the one used for `secDataProtectionKey` derivation.

3.2.1.1.5 SecDerivedPayloadKey

`secDerivedPayloadKey` is derived using the `CMAC-AES(secDataProtectionKey, Counter|Label|Context|OutputLength)` function, where `Counter` is set to `0x01`, `Label` is set to `DerPaylK` and `OutputLength` is set to 128. `Context` is built by concatenating the 96 less significant bits of `configDigest` with `cryptoStsIndex`.

3.2.1.1.6 SecDerivedAuthenticationKey

`secDerivedAuthenticationKey` is derived using the `CMAC-AES(secDataProtectionKey, Counter|Label|Context|OutputLength)` function, where `Counter` is set to `0x01`, `Label` is set to `DataAuthK` and `OutputLength` is set to 128. `Context` is the same as the one used for `secDerivedPayloadKey`.

3.2.1.1.7 SecDerivedAuthenticationIV

`secDerivedAuthenticationIV` is derived using the `CMAC-AES(secDataProtectionKey, Counter|Label|Context|OutputLength)`, where `Counter` is set to `0x01`, `Label` is set to `DerAuthI` and `OutputLength` is set to 128. `Context` is the same as the one used for `secDerivedPayloadKey` and `secDerivedAuthenticationKey`.

Note: Derived keys are updated either in a per-session or per-rotation increment basis depending on the chosen *STS* configuration. The update basis and the element impacted can be seen in Fig. 3.6 Refer to *Key Rotation* for more details.

3.2.1.1.8 FiRa key derivation function

Diving into the key derivation process, the first step is to take a closer look at the *FiRa key derivation function* widely used in the calculation pipeline of both Fig. 3.6 and and Fig. 3.4. Fig. 3.2 shows an operation block where *CMAC* is used as the underlying cryptographic primitive ([NIS05]) with *AES* as the block cipher ([NIS01a]). It features:

- 32 bits Counter: Its starting value is set to `0x00000001` and it is incremented by one for each *CMAC-AES* operation. If a 256 bits derived key needs to be obtained as output, two iterations are needed and the results are concatenated.
- 64 bits Label: Identifies the purpose of the derived key.
- 128 bits Context: Links the derived key to a specific session.
- 32 bits Output length: It defines the length of the derived key.
- 128 bits or 256 bits Root key: The key for the *CMAC* algorithm.

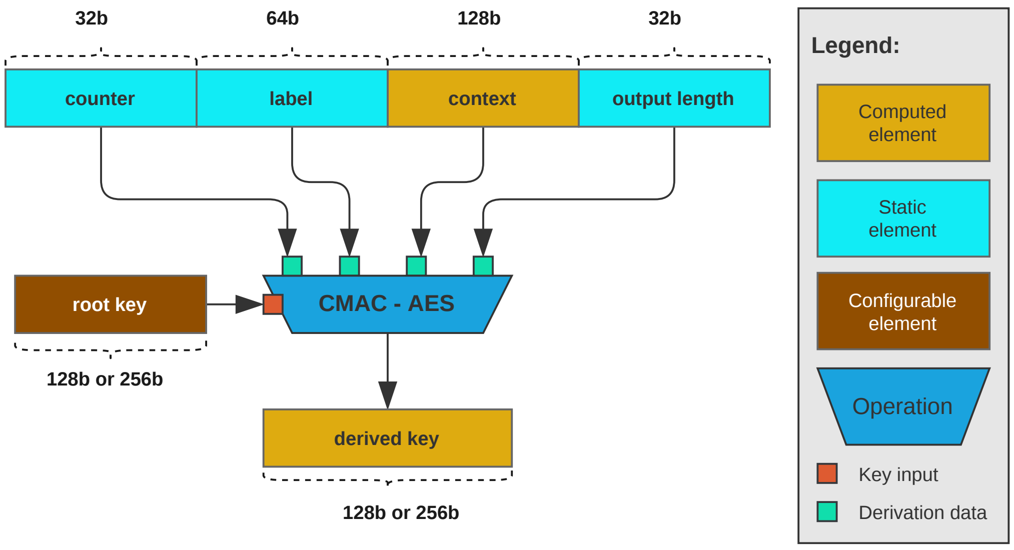

Fig. 3.2: FiRa Key Derivation Function

For more information about the key derivation process, please refer to [NIS09].

3.2.1.1.9 ConfigDigest

Disregarding the session, the *Host Application* shall provide a valid session configuration to the *UWBS* to enable its start. As mentioned earlier, a 128 bits context is needed to link the derived crypto assets to a given session. This context is named `configDigest` and it is built using different element of the session configuration. This mechanism forbids, even in Static *STS*, two different sessions to deduce the same crypto assets.

To build this crypto asset, a 17 bytes vector is built from the following list of concatenated configuration parameters. Strictly in the given order:

- `RANGING_ROUND_USAGE`: 1 byte
- `STS_CONFIG`: 1 byte
- `MULTI_NODE_MODE`: 1 byte
- `CHANNEL_NUMBER`: 1 byte

- `SLOT_DURATION`: 2 byte
- `MAC_FCS_TYPE`: 1 byte
- `RFRAME_CONFIG`: 1 byte
- `PREAMBLE_CODE_INDEX`: 1 byte
- `SFD_ID`: 1 byte
- `PSDU_DATA_RATE`: 1 byte
- `PREAMBLE_DURATION`: 1 byte
- `CONSTANT = 0x03`: 1 byte
- `ID`: 4 byte

Note: For non-slot-based ranging structures, `SLOT_DURATION` is set to `0x0000`. For all other configurations, `SLOT_DURATION` is set to its value in microseconds.

Note: `ID` might represent either the session or the sub-session identifier depending on the *STS* configuration chosen.

Note: For multi-byte elements in the list, the most significant byte is placed first. Section D.2.1 in [FiR22] provides an example on how to build such a vector.

The 128 bits `configDigest` is finally obtained by applying the `CMAC-AES(Key, Concatenated Parameters)` function, where the Key is a 128 bits zero byte-string. `configDigest` is used during all other *CMAC-AES* operations as part of the context input parameter in the *KDF*.

3.2.1.1.10 Counters

3.2.1.1.11 PhyStsIndexInit

Going further in execution of the pipeline illustrated in Fig. 3.6 and Fig. 3.4, the `phyStsIndexInit` is computed using the `CMAC-AES(secDataProtectionKey, Counter|Label|Context|OutputLength)` function, where `Counter` is set to `0x01`, `Label` is set to `StsIndIn` and `OutputLength` is set to 128.

3.2.1.1.12 PhyStsIndex

`phyStsIndex` is a 32 bits counter which initial value is initialized during the starting phase of the session with the result of the operation `phyStsIndexInit & 0x7FFFFFFF`. Then, during session execution, it is incremented by one after each slot and, as it is explained in *Slot Index Synchronization*, has a vital role in the slot time synchronization of controlees.

3.2.1.1.13 CryptoStsIndex

Similarly to `phyStsIndex`, `cryptoStsIndex` is another 32 bits counter used as part of the *NONCE* which is linking crypto assets to time during the life-span of a session providing replay protection. It mirrors `phyStsIndex` or `slotIndex` depending on the *STS* configuration.

Note: Derived counters are updated in a per-slot basis as shown by Fig. 3.6 and Fig. 3.4.

3.2.1.2 Frame protection mechanisms

Previous sections explained the computation of the different crypto assets needed to secure FiRa message exchanges. This section explains how to actually use them during an ongoing ranging session.

Explicit data contained in *UWB* frames shall be protected against eavesdropping. In the specific case of FiRa, the *MAC* layer is responsible for encrypting and authenticating different parts of the *PSDU*. In FiRa, while *PIE* encryption is mandatory, *HIE* encryption is optional and depends on the chosen *STS* configuration.

3.2.1.2.1 Authenticated Encryption with Associated Data

If a *PIE* is transmitted in the nested *IE*s field of the *UWB* frame, FiRa ensures confidentiality and authenticity by using *AES-CCM** as the underlying cryptographic primitive. *CCM** combines encryption, counter mode and authentication through variable-length authentication tags. After having encrypted the message, the *UWBS* concatenates both the cipher text and an encrypted 8-bytes authentication tag into the *PIE*s fields of the *MAC* payload.

As *AEAD* scheme is used, an Auxiliary Security Header (as described by section 9.4.2 in [IEE20a]) is included in the *MHR*. The required fields are set as follows:

- *Security Level* is set to `0b110` (`ENC-MIC-64`) as both encryption and 64 bits authentication tags are requested by FiRa.
- *Key Identifier Mode* is set to `0b00` as `secDerivedPayloadKey`, implicitly known by both parties, is used as an encryption key.
- *Frame Counter Suppression* is set to `0b1` as the frame counter is not directly carried in the *MHR* but it is set to the previously mentioned `cryptoStsIndex`.
- *ASN* is set to `0b0` as the *NONCE* is formed by concatenating `securityLevel`, `cryptoStsIndex` and `Source Address`:
  - `nonce[7:0]` is set to `0x06` (`ENC-MIC-64`).
  - `nonce[39:8]` is set to the `cryptoStsIndex` of the current frame.
  - `nonce[103:40]` is set to the corresponding extended-source-address of the device.

Note: If short-address is used instead of extended-address, most-significant-bytes of `nonce[103:40]` shall be padded with `0x00`.

Note: [IEE20a] and [NIS04] provide more details about *AES-CCM**.

3.2.1.2.2 Header IE Encryption/Decryption

To avoid a malicious agent to be able to track `phyStsIndex` as a simple incremental counter, some *STS* configurations require to encrypt FiRa *HIE*. When it is needed, this encryption is performed using an *AES-ECB* operation as described in [NIS01b] with `secPrivacyKey` as the encryption key and the concatenation of a known 64 bits padding field, `sessionId` and `phyStsIndex` as plaintext.

Note: To allow a non-synchronized device to decrypt *HIE* and retrieve `phyStsIndex`, `secPrivacyKey` does not change during the entire duration of the session.

3.2.1.2.3 MAC Payload Encryption bypass

While not compliant with FiRa protocol, *UWBS* provides a vendor-specific mechanism to avoid *AEAD* encryption on *PIE*s fields. Deactivating this encryption is achieved by setting the `UWB_STACK_MAC_ENCRYPTION_CONTROL` compilation flag.

3.2.1.2.4 Slot Index Synchronization

During FiRa ranging session using specific modes such as *TWR*, the Controller device is responsible for keeping track of time and to implement the necessary elements to enable the participants to synchronize with itself.

As explained in other chapters, FiRa divides the time into blocks, rounds, and slots, providing an incremental monotonic counter (also named `phyStsIndex`) to identify each slot during session's life-span.

Alongside FiRa's vendor *OUI* and session identifier, current `phyStsIndex` is broadcasted on each frame as part of FiRa *HIE*. Then, each controlee shall be able, using its own cryptographic secrets, to deduce `phyStsIndexInit` and, by using its session configuration, to calculate the current time. (3.1) illustrates this relation.

$$\begin{aligned}
BlockIndex_i &= \lfloor \frac{AbsoluteSlotIndex_i}{SlotsPerBlock} \rfloor \\
RoundIndex_i &= \lfloor \frac{SlotInCurrentBlock_i}{SlotsPerRound} \rfloor \\
SlotIndex_i &= SlotInCurrentBlock_i \quad \% \quad SlotsPerRound \\
&\text{Where:} \\
SlotsPerBlock &= \lfloor \frac{BlockDuration}{SlotDuration} \rfloor \\
SlotInCurrentBlock_i &= AbsoluteSlotIndex_i \quad \% \quad SlotsPerBlock \\
AbsoluteSlotIndex_i &= PhyStsIndex_i - InitialPhyStsIndex \\
InitialPhyStsIndex &= PhyStsIndexInit \quad \& \quad 0x7FFFFFFF
\end{aligned} \tag{3.1}$$

3.2.1.2.5 STS Generation

The lower part of Fig. 3.6 and Fig. 3.4 shows the mapping between FiRa defined crypto assets (`cryptoStsIndex`, `phyVUpper64`, `secDerivedAuthenticationKey`, `secDerivedAuthenticationIV`) and the elements required by [IEE20b] (`phyHrpUwbStsVUpper96`, `phyHrpUwbStsVCounter` and `phyHrpUwbStsKey`) to build the *STS* of a frame.

During each `TX` or `RX` with a `SP1` or `SP3` frame, the *PHY* obtains the necessary seeds to perform its own *DRBG* and generate the expected sequence.

Note: *STS* variability is a fundamental aspect of a secure ranging protocol. The proposed FiRa configurations ensure different levels of variability frame-to-frame depending on the complexity of the use-case served by the *UWB* device.

3.2.2 Static STS

This chapter contains information about elements that are specific to the Static *STS*. Fig. 3.3 shows the container diagram for a *TWR* ranging session in Static *STS* between one Controller and one Controlee.

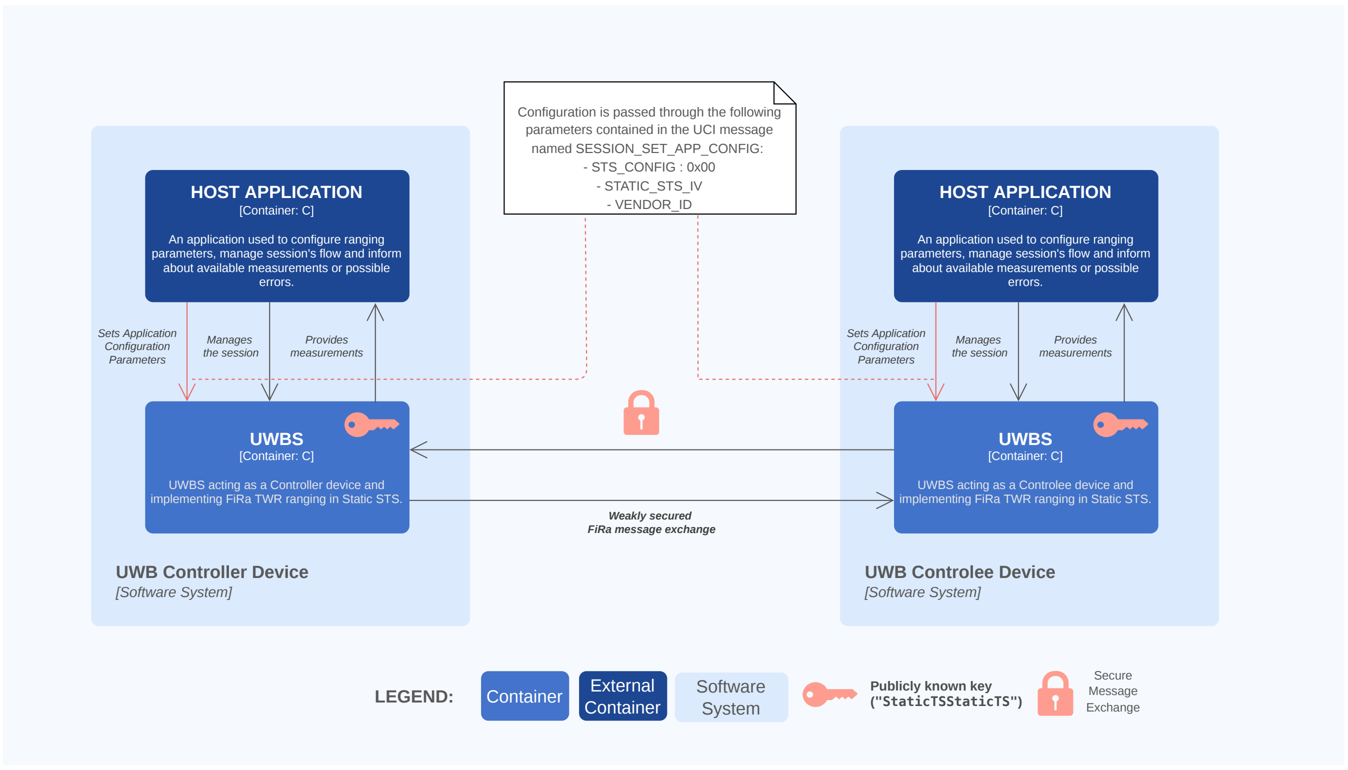
Configuration is passed through the following parameters contained in the UCI message named SESSION_SET_APP_CONFIG:
- STS_CONFIG : 0x00
- STATIC_STS_IV
- VENDOR_ID
HOST APPLICATION
[Container: C]
An application used to configure ranging parameters, manage session's flow and inform about available measurements or possible errors.
Sets Application Configuration Parameters
Manages the session
Provides measurements
UWBS
[Container: C]
UWBS acting as a Controller device and implementing FiRa TWR ranging in Static STS.
UWBS acting as a Controlee device and implementing FiRa TWR ranging in Static STS.
Weakly secured FiRa message exchange
UWB Controller Device
[Software System]
UWB Controlee Device
[Software System]
LEGEND:
Container
External Container
Software System
Publicly known key ("StaticTSStaticTS")
Secure Message Exchange

Fig. 3.3: Container Diagram for Static *STS*

Static is the simplest *STS* configuration, based on a publicly known `secSessionKey` and a group of crypto-derived materials that do not change during the whole session. This configuration is mandatory for those cases where prior knowledge of secrets is not possible or necessary. For example, in scenarios such as indoor navigation, scanning, information broadcast, or any other case where the exchanged data is not specifically considered sensitive.

Note: Some FiRa 2.x features, such as *DL-TDoA*, *UL-TDoA*, Contention-Based Ranging and *OWR* for *AoA* must use Static *STS*.

3.2.2.1 Crypto assets management

Static *STS* uses a simplified version of the key derivation pipeline previously described. These simplifications are illustrated in Fig. 3.4.

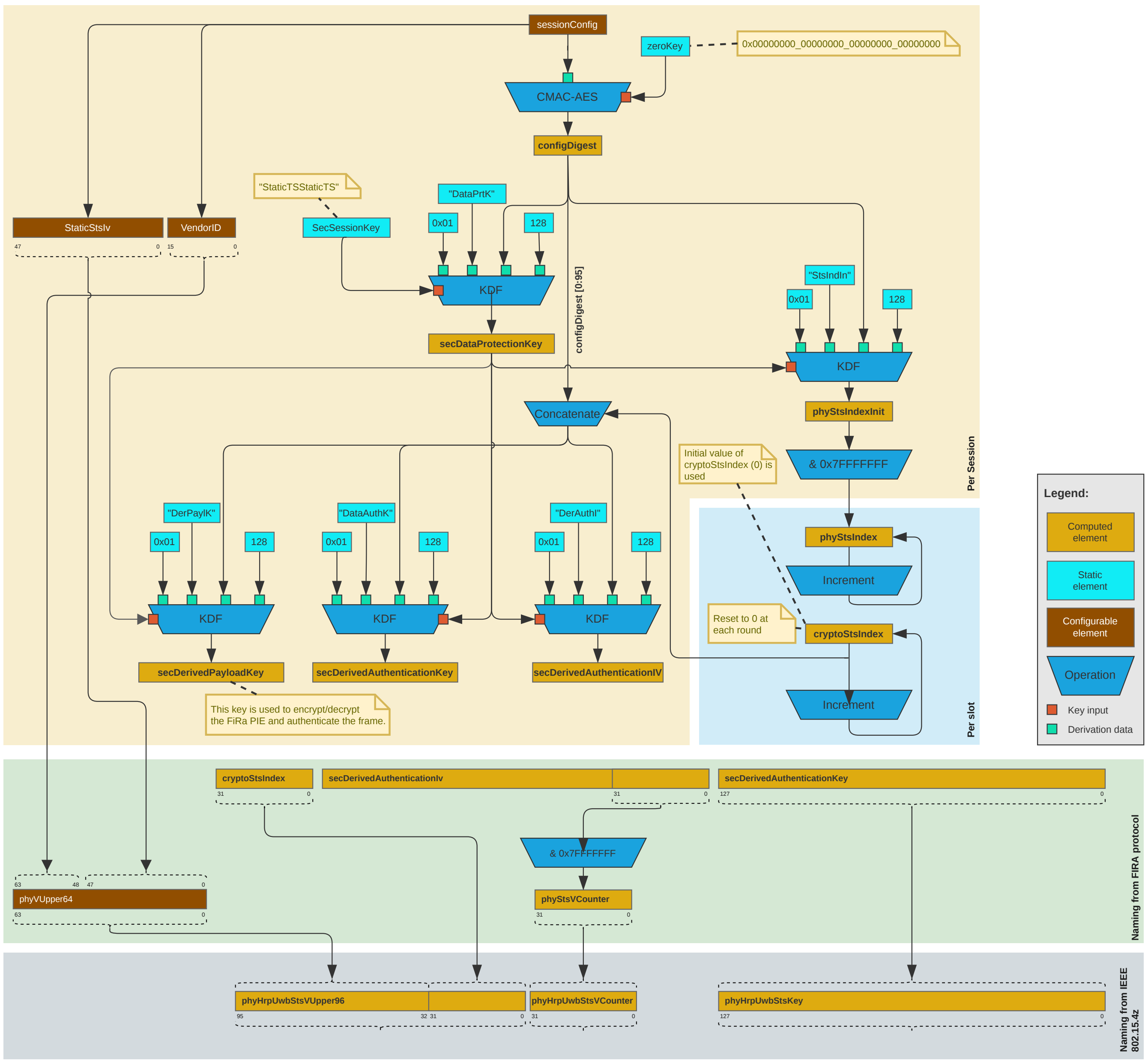

Fig. 3.4: Crypto assets computation pipeline for Static *STS*

3.2.2.1.1 Crypto keys

3.2.2.1.2 SecSessionKey

As shown in Fig. 3.4, the `SecSessionKey` in Static *STS* is publicly known and set to 128 bits string `StaticTSStaticTS`. This makes *Static* non-secure by design but allows for a simple and fast implementation.

3.2.2.1.3 Frame protection mechanisms

3.2.2.1.4 HIE and PIE Encryption/Decryption

In Static *STS*, only FiRa *PIE* is encrypted. *HIE* is transmitted as plaintext over the air. This allows any existing FiRa session to access current `phyStsIndex` and rapidly synchronize with the Controller device.

3.2.2.1.5 Slot Index Synchronization

Session slot synchronization is achieved by using the same mechanism explained in *Slot Index Synchronization*.

3.2.2.1.6 STS Generation

`phyVUpper64` is generated from the concatenation of `VendorId` and `StaticStsIv`, both set through *UCI* during session configuration. As these parameters are directly set by the application and agreed upon by *OOB*, they contribute to *STS* variability between two different sessions.

On the other hand, `cryptoStsIndex`, as part of `phyHrpUwbStsVUpper96`, takes its value from the current slot index; hence it is reset to `0x00` at the beginning of each round. This ensures that a different *STS* is generated inside each slot of the round, but not between different blocks of the same session.

3.2.2.2 Key Takeaways

To summarize, using a static *STS* has the following characteristics:

- The session key is publicly known.
- The same session key is used as a source to secure all the exchanges.
- Slot Index takes part in sequence generation and payload encryption, ensuring that a different *STS* can be generated at every slot of the round, but not providing block-to-block variability.
- Only *PIE* is encrypted, *HIE* is transmitted in plaintext.
- As key rotation is not supported, the same derived keys are used for the whole lifetime of the session, ensuring immediate resynchronization but compromising the overall confidentiality of crypto assets.
- Allows for low-cost and high-performance implementation.

3.2.3 Provisioned STS

This chapter contains information about elements that are specific to the Provisioned *STS*. Fig. 3.5 shows the container diagram for a *TWR* ranging session in Provisioned *STS* between one Controller and one Controlee.

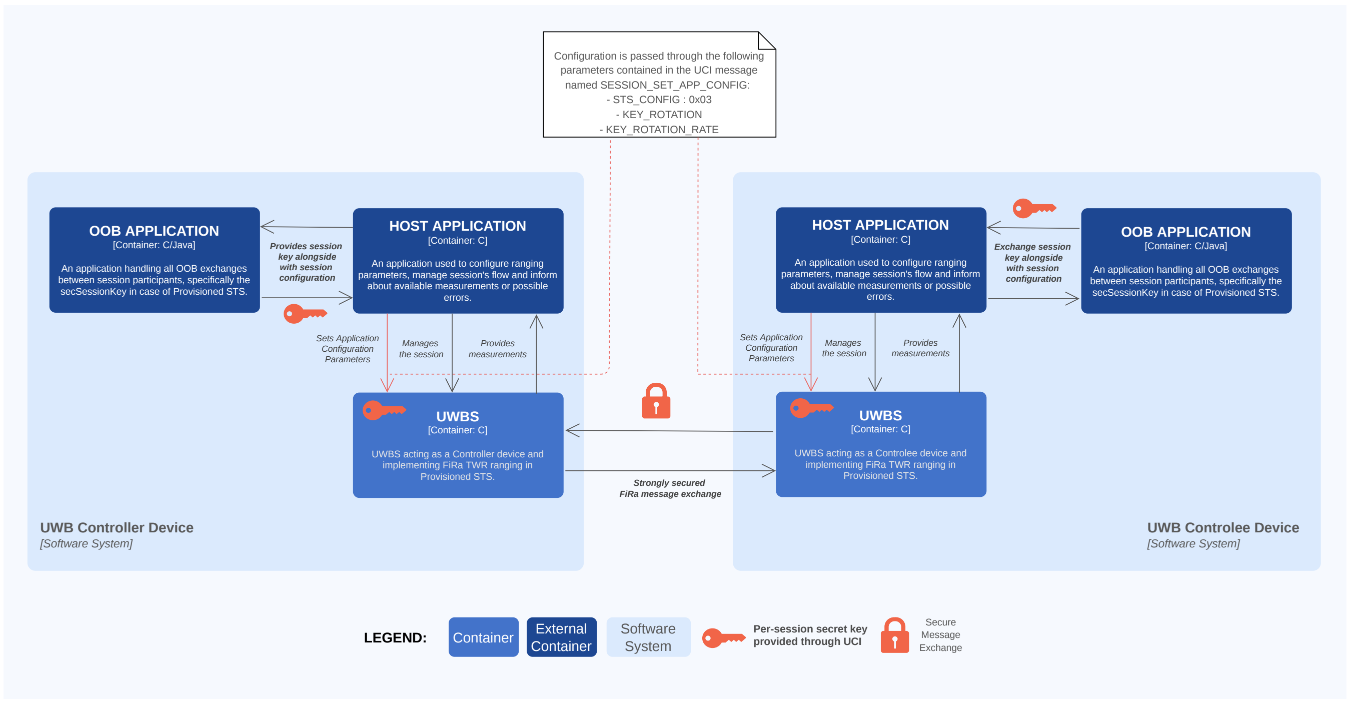
Configuration is passed through the following parameters contained in the UCI message named SESSION_SET_APP_CONFIG:
- STS_CONFIG : 0x03
- KEY_ROTATION
- KEY_ROTATION_RATE
OOB APPLICATION
[Container: C/Java]
An application handling all OOB exchanges between session participants, specifically the secSessionKey in case of Provisioned STS.
Provides session key alongside with session configuration
HOST APPLICATION
[Container: C]
An application used to configure ranging parameters, manage session's flow and inform about available measurements or possible errors.
Sets Application Configuration Parameters
Manages the session
Provides measurements
UWBS
[Container: C]
UWBS acting as a Controller device and implementing FiRa TWR ranging in Provisioned STS.
Strongly secured FiRa message exchange
UWBS acting as a Controlee device and implementing FiRa TWR ranging in Provisioned STS.
Exchange session key alongside with session configuration
UWB Controller Device
[Software System]
UWB Controlee Device
[Software System]
LEGEND:
Container
External Container
Software System
Per-session secret key provided through UCI
Secure Message Exchange

Fig. 3.5: Container Diagram for Provisioned *STS*

Apart from the *UWBS* device and the *Host Application* already mentioned in Static *STS* section, Fig. 3.5 also shows the *OOB* application handling *OOB* channels and providing session configuration. It also illustrates the session keys used to secure FiRa messages exchanges.

Note: *OOB* internal details and exchanges are out of the scope of this chapter so these are not shown in Fig. 3.5.

3.2.3.1 Crypto assets management

Provisioned *STS* uses a different version of the cryto assets computation pipeline previously described and is illustrated in Fig. 3.6.

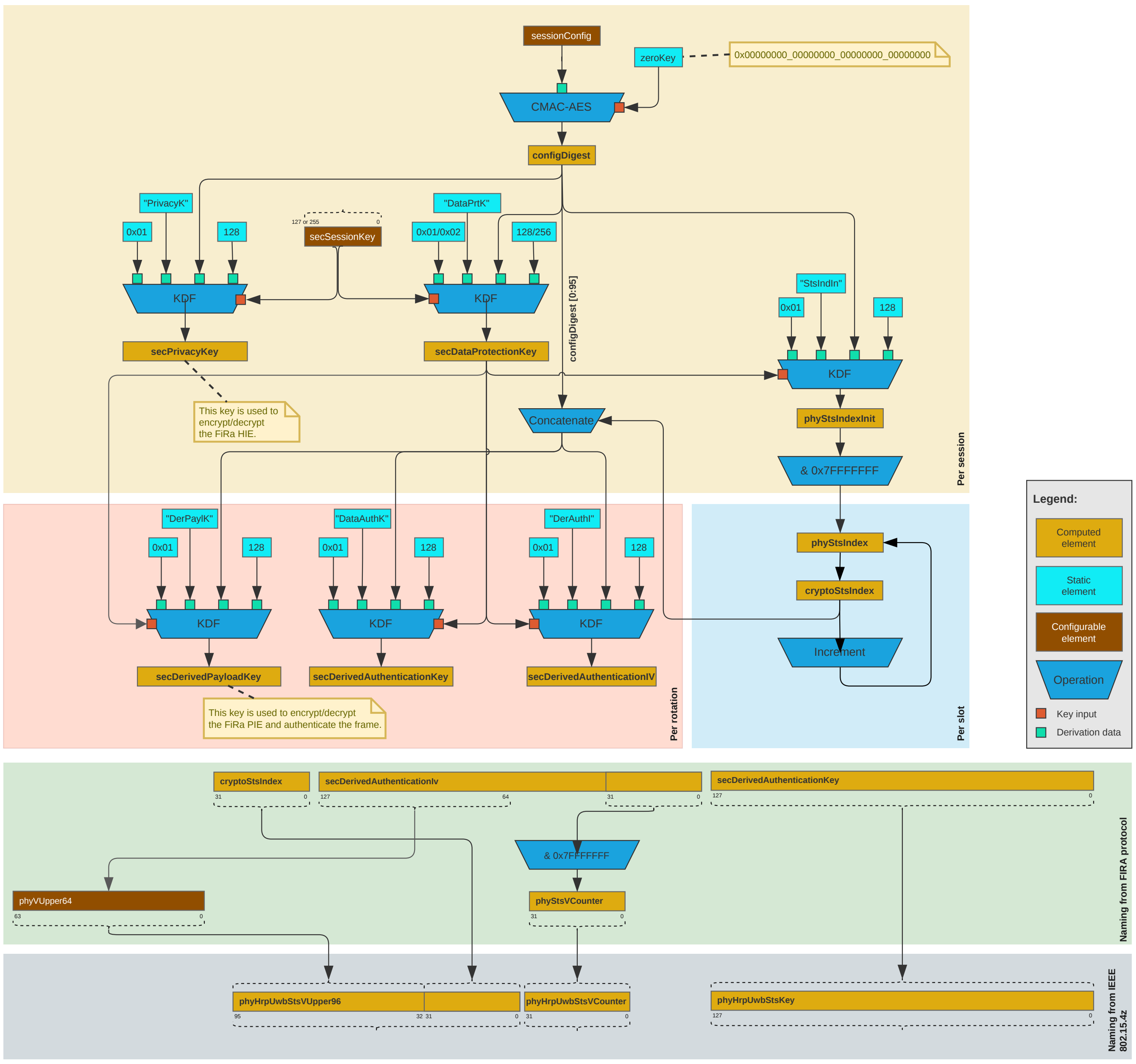

Fig. 3.6: Crypto assets computation pipeline for Provisioned *STS*

3.2.3.1.1 Crypto keys

3.2.3.1.2 SecSessionKey

`secSessionKey` is agreed upon during *OOB* exchanges and given to the *UWBS* in plaintext during session configuration phase. Two possible lengths are supported: 128 bits and 256 bits.

3.2.3.2 Frame Protection mechanisms

For all the following operations, `cryptoStsIndex` is taken from `phyStsIndex` instead from the current slot index as in Static *STS*.

3.2.3.2.1 Key Rotation

The set of cryptographic assets are expected to change at a given rate during the entire duration of the session, reducing the risk of side channel attacks. These assets are:

- `secDerivedPayloadKey`
- `secDerivedAuthenticationKey`
- `secDerivedAuthenticationIv`

Even though this key rotation procedure is required to be supported by Provisioned *STS*, its use is not mandatory. The session configuration allows to enable/disable it from application perspective and it allows to set the corresponding rotation rate:

- `KEY_ROTATION`, 1 byte: `0x00` to disable, `0x01` to enable [default: `0x00`].
- `KEY_ROTATION_RATE`, 1 byte: defining the `n` parameter [default: `0`].

A new set of keys are available every 2^n blocks and their *KDF*, according to Fig. 3.6, shall use `cryptoStsIndex` from the `phyStsIndex` value at the first slot of the block triggering the rotation.

3.2.3.2.2 HIE and PIE Encryption/Decryption

Alongside an encrypted *PIE*, Provisioned *STS* also encrypts *HIE*, ensuring `phyStsIndex` and `sessionId` are not transmitted in plaintext and are not easily identifiable over session lifetime as described in *Header IE Encryption/Decryption*.

Note: *HIE* encryption shall be performed before *AEAD* on *PIE*s fields, in other words, authentication tag shall be calculated on both encrypted *HIE* and *PIE*s.

3.2.3.2.3 STS Generation

STS generation in Provisioned *STS* follows exactly the same procedure as described in *STS Generation*. Instead of deducing `phyVUpper64` from session configuration, as in Static *STS*, its value is taken from the current state of the most significant 64 bits of `secDerivedAuthenticationIv`.

3.2.3.3 Key Takeaways

To summarize, using a provisioned *STS* has the following characteristics:

- `secSessionKey` is agreed upon during *OOB* exchanges.
- The same session key is used as a source to secure all the exchanges.
- `phyStsIndex` takes part in both *STS* generation and frame encryption.
- Both *PIE* and *HIE* are encrypted using different *AES* mechanisms.
- Key rotation of certain crypto assets is supported.

3.2.4 Provisioned STS with Sub-session Keys

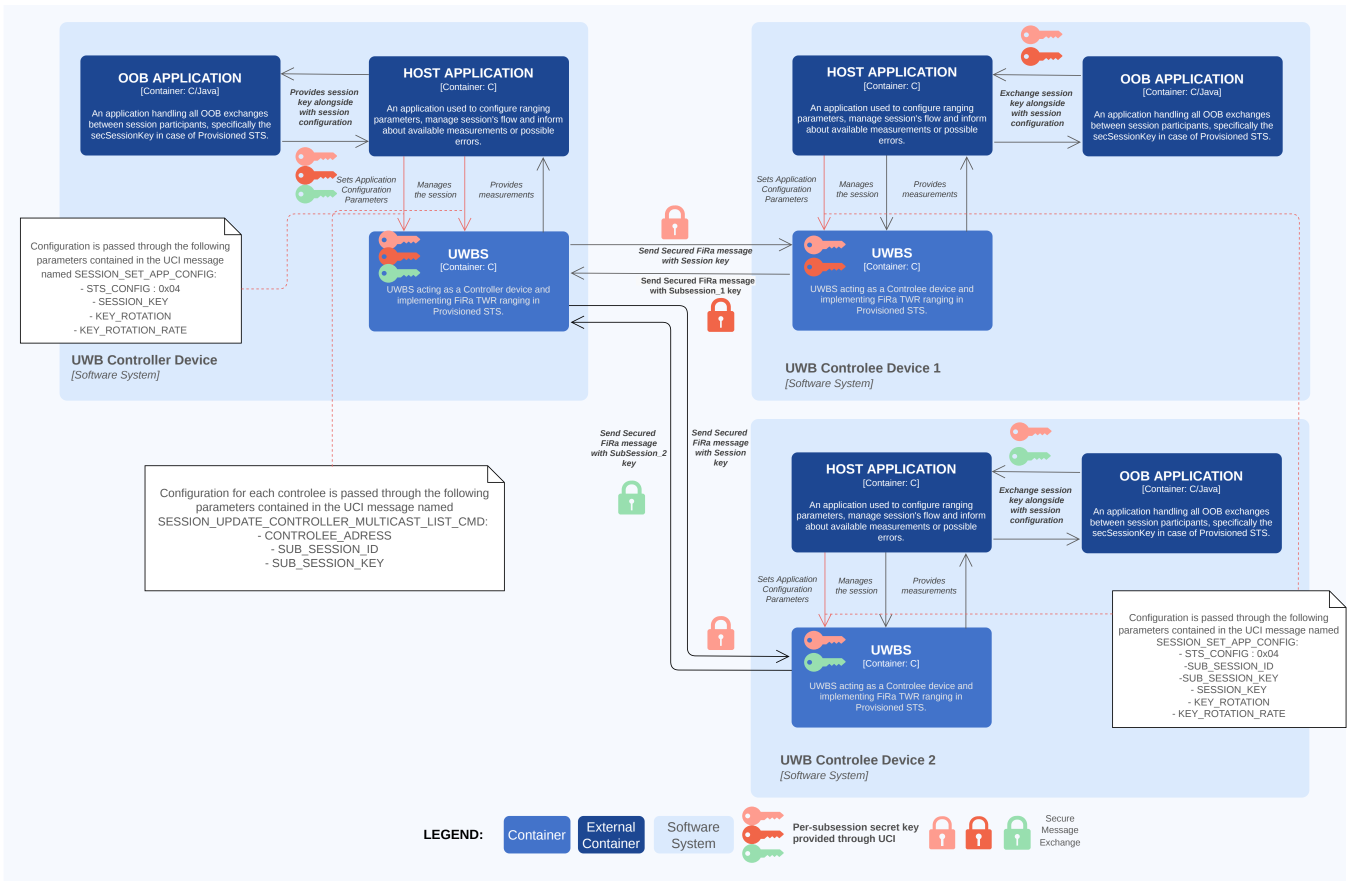
OOB APPLICATION
[Container: C/Java]
An application handling all OOB exchanges between session participants, specifically the secSessionKey in case of Provisioned STS.
Provides session key alongside with session configuration
HOST APPLICATION
[Container: C]
An application used to configure ranging parameters, manage session's flow and inform about available measurements or possible errors.
Sets Application Configuration Parameters
Manages the session
Provides measurements
UWBS
[Container: C]
UWBS acting as a Controller device and implementing FiRa TWR ranging in Provisioned STS.
Configuration is passed through the following parameters contained in the UCI message named SESSION_SET_APP_CONFIG:
- STS_CONFIG : 0x04
- SESSION_KEY
- KEY_ROTATION
- KEY_ROTATION_RATE
UWB Controller Device
[Software System]
Send Secured FiRa message with Session key
Send Secured FiRa message with Subsession_1 key
HOST APPLICATION
[Container: C]
An application used to configure ranging parameters, manage session's flow and inform about available measurements or possible errors.
Exchange session key alongside with session configuration
OOB APPLICATION
[Container: C/Java]
An application handling all OOB exchanges between session participants, specifically the secSessionKey in case of Provisioned STS.
Sets Application Configuration Parameters
Manages the session
Provides measurements
UWBS
[Container: C]
UWBS acting as a Controlee device and implementing FiRa TWR ranging in Provisioned STS.
UWB Controlee Device 1
[Software System]
Send Secured FiRa message with SubSession_2 key
Send Secured FiRa message with Session key
Configuration for each controlee is passed through the following parameters contained in the UCI message named SESSION_UPDATE_CONTROLLER_MULTICAST_LIST_CMD:
- CONTROLEE_ADRESS
- SUB_SESSION_ID
- SUB_SESSION_KEY
HOST APPLICATION
[Container: C]
An application used to configure ranging parameters, manage session's flow and inform about available measurements or possible errors.
Exchange session key alongside with session configuration
OOB APPLICATION
[Container: C/Java]
An application handling all OOB exchanges between session participants, specifically the secSessionKey in case of Provisioned STS.
Sets Application Configuration Parameters
Manages the session
Provides measurements
UWBS
[Container: C]
UWBS acting as a Controlee device and implementing FiRa TWR ranging in Provisioned STS.
Configuration is passed through the following parameters contained in the UCI message named SESSION_SET_APP_CONFIG:
- STS_CONFIG : 0x04
-SUB_SESSION_ID
-SUB_SESSION_KEY
- SESSION_KEY
- KEY_ROTATION
- KEY_ROTATION_RATE
UWB Controlee Device 2
[Software System]
LEGEND:
Container
External Container
Software System
Per-subsession secret key provided through UCI
Secure Message Exchange

Fig. 3.7: Container Diagram for Provisioned *STS* for Responder Specific Sub-session Key

This *STS* configuration relies on the concept of sub-sessions. A FiRa session contains a set of cryptographically derived assets, all calculated from a root key and a group configuration parameters, including the session identifier. A sub-session symmetrically calculates its own set of assets from a sub-session key and a sub-session identifier. These secondary groups of derived keys are uniquely known by the Controlee defining the corresponding sub-session and the Controller device, helping to secure each link between two *UWB* devices in a *TWR* multicast ranging session.

3.2.4.1 Crypto assets management

3.2.4.1.1 Crypto keys

3.2.4.1.2 SecSessionKey

Depending on the chosen *STS* configuration, the `secSessionKey` follows the same rules as in Provisioned *STS* described in *Crypto assets management*.

3.2.4.2 Frame Protection mechanisms

As in Provisioned *STS*, frame protection procedures described in *Frame Protection mechanisms* still apply. Following FiRa ranging round structure in a *TWR* multicast session, all exchanges having the Controller as a source are secured with cryptographic assets derived from the session key. On the other hand, all exchanges having a specific Controlee as a source are secured with cryptographic assets derived from the corresponding sub-session key. This avoid a unintended Controlee to interfere, replay or eavesdrop on exchanges from other peers in the session. Fig. 3.7 illustrates this concept by having different root key for each exchange link.

3.2.4.3 Key Takeaways

This *STS* configuration inherits all the properties from Provisioned *STS* mentioned in this *section*. In addition, it also handles the problem of intra-session security between participants and features the following additional characteristics:

- Several keys are used to secure exchanges, each of them can either be configured through *UCI*.
- Exchanges having the Controller device as a source are secured with the `secSessionKey` as root key. Nevertheless, each Controlee uses its own `sub_session_key` to secure messages having the Controller device as destination.
- `phyStsIndex`, used for slot time synchronization, is individually derived from the corresponding key and used accordingly depending on the slot.
- As in Provisioned *STS*, key rotation is supported. In addition to crypto-derived assets coming from `secSessionKey`, the Controller device must rotate the ones coming from each `sub_session_key`.

4 FiRa Diagnostics

4.1 FiRa Diagnostics Overview

During FiRa ranging operation, the Qorvo UWB stack is able to gather various information about the received frames. This information can be considered as supplemental compared to the content of the standard range notification as it is not intended to be provided to upper layers.

The stack offers a feature named *diagnostics* allowing to surface this information when its FiRa ranging capability is used. This document describes the content of those diagnostics and how it shall be used.

4.2 Functional description

4.2.1 General description

By default, the FiRa region produces reports at the end of each round containing a summary of the measures it has been able to compute during the round. Those reports do not contain details about the frames exchanged during the round. The FiRa diagnostics feature of the Qorvo UWB Stack has been designed to solve this issue and to provide details about a round after its completion.

When activated, the stack stores additional information it gathers during the round. When the latter has been completed, it uses the stored information to produce a diagnostic report it attaches to the round. The diagnostic report is created disregarding the round validity status.

The diagnostics produced by the stack are composed of the following fields:

- **frame reports**: a set of different elements related to each frame exchanged during the round.

The diagnostic report is sent at the end of the ranging round, just after the standard ranging round report (`SESSION_INFO_NTF` *UCI* message). The session ID and sequence number of the round are included in the report to ease the correlation with the round.

The diagnostics can be activated through a FiRa session parameter. Parts of the report are also configurable through specific parameters.

4.2.2 Diagnostics content

The following chapter describes in more detail the content of a frame reports.

4.2.3 *Frame reports* field content

The *Frame reports* field contains the detail of all frames exchanged during the round. More specifically it provides for each frame:

- The action performed on the frame: TX or RX
- The antenna set used to transmit or receive
- The FiRa message ID of the frame
- The Emitter Short Address:
  - For TX frames, it contains the local MAC short address.
  - For RX frames, it contains the MAC short address of the emitter when the reception is successful, otherwise, it contains `0xFFFF`.
- A set of status related to the frame containing:
  - The status of the frame (successfully processed or not)
  - The status of the WiFi activation state during the frame
  - The status of the Max grant duration (exceeded or not)
- The *CFO* of the frame (only applicable to RX frames): clock offset between the transmitter and the receiver
- A list of Segment Metrics reports (only applicable to RX frames): Their number depends on the antenna sets used to perform the reception and the elements that were measured, there will be:
  - One Segment Metrics report per segment used to compute the *ToA* (i.e. one per receiver used, in case of simultaneous RX it will be >1)
  - One Segment Metrics report per segments used to compute each *AoA* supported by the antenna set. When the segment used for *ToA* is also used for *AoA* only one Segment Metric is given).

  Each Segment Metrics report contains:
  - The associated receiver ID and received segment index
  - The RF noise floor value during reception
  - The *RSL* of the global segment
  - The index of the first path in the entire *CIR* window
  - The *RSL* of the first path
  - The timestamp of the first path
  - The index of the peak path in the entire *CIR* window
  - The *RSL* of the peak path
  - The timestamp of the peak path

Note: the RF noise floor is not attached to a specific RX path (First path or Peak path), but is applicable to the entire received segment. It allows the computation of the *SNR* values corresponding to their relative *RSL*s, thanks to the following formula:

$$SNR = RSL - noise\_value$$

- A list of *AoA*s (only applicable to RX frames): one for each *AoA* the selected antenna set can compute containing:
  - The local *TDoA*
  - The local *PDoA*
  - The local *AoA*
  - The local *FoM* of the *AoA*
  - The type of *AoA* (Axis X, Y or Z)
- A list of *CIR* reports (only applicable for RX frames): Their number follows the same logic as the one for *Segment Metrics reports*.

  Each *CIR* report contains:
  - The associated receiver ID and received segment index
  - The offset of the first path in the sample window
  - The sample window around the first path

The following parts of the frame reports can be activated through a session parameter:

- Segment Metrics
- *AoA*s
- *CIR*s
- *CFO*
- Emitter Short Address

References

[FiR22] FiRa. Uci test specification. Technical Report, FiRa, 2022.

FiRa. Uci test specification. Technical Report, FiRa, 2022.

[FiR23a] FiRa. Fira consortium uwb command interface generic technical specification. Technical Report, FiRa, 2023.

FiRa. Fira consortium uwb command interface generic technical specification. Technical Report, FiRa, 2023.

[FiR23b] FiRa. Fira consortium uwb mac technical requirements. Technical Report, FiRa, 2023.

FiRa. Fira consortium uwb mac technical requirements. Technical Report, FiRa, 2023.

[IEE20a] IEEE. Standard for low-rate wireless networks. Technical Report, IEEE, 2020.

IEEE. Standard for low-rate wireless networks. Technical Report, IEEE, 2020.

[IEE20b] IEEE. Standard for low-rate wireless networks - amendment 1: enhanced ultra wideband (uwb) physical layers (phys) and associated ranging techniques. Technical Report, IEEE, 2020.

IEEE. Standard for low-rate wireless networks - amendment 1: enhanced ultra wideband (uwb) physical layers (phys) and associated ranging techniques. Technical Report, IEEE, 2020.

[NIS01a] NIST. Fips 197: advanced encryption standard (aes). Technical Report, NIST, 2001.

NIST. Fips 197: advanced encryption standard (aes). Technical Report, NIST, 2001.

[NIS01b] NIST. Nist sp 800-38a: recommendation for block cipher modes of operation: methods and techniques. Technical Report, NIST, 2001.

NIST. Nist sp 800-38a: recommendation for block cipher modes of operation: methods and techniques. Technical Report, NIST, 2001.

[NIS04] NIST. Nist sp 800-38c: recommendation for block cipher modes of operation: the ccm mode for authentication and confidentiality. Technical Report, NIST, 2004.

NIST. Nist sp 800-38c: recommendation for block cipher modes of operation: the ccm mode for authentication and confidentiality. Technical Report, NIST, 2004.

[NIS05] NIST. Nist sp 800-38b: recommendation for block cipher modes of operation: the cmac mode for authentication. Technical Report, NIST, 2005.

NIST. Nist sp 800-38b: recommendation for block cipher modes of operation: the cmac mode for authentication. Technical Report, NIST, 2005.

[NIS09] NIST. Nist sp 800-108: recommendation for key derivation using pseudorandom functions. Technical Report, NIST, 2009.

NIST. Nist sp 800-108: recommendation for key derivation using pseudorandom functions. Technical Report, NIST, 2009.

Index