



Realm Host Interface specification

Document number	DEN0148
Document quality	ALP
Document version	1.0-alp2
Document confidentiality	Non-confidential
Document build information	9e3736c6 doctool 0.56.1

Copyright © 2024 Arm Limited or its affiliates. All rights reserved.

DRAFT

Realm Host Interface specification

Release information

1.0-alp2 (31-10-2024)

New features

- Addition of RHI_FAL_CLOSE ABI

Clarifications

- Alter wording of parameter blocks to description, [value] form.
- Move to consistent 'sync' terminology in Appendix A (previously a mix of 'sync' and 'injection').
- Typo corrections.

Defects

- Allocate FIDs within the range reserved for RHI in SMCCC.
- Swap SessionID and connectionType parameters for RHI_SESSION_OPEN to be consistent with other calls in protocol.
- Types within Appendix A renamed to use BSB naming instead of BIB naming.

Relaxations

None

1.0-alp1 (19-09-2024)

- Added Device Assignment ABI chapter

DRAFT

Arm Non-Confidential Document License (“License”)

This License is a legal agreement between you and Arm Limited (“Arm”) for the use of Arm’s intellectual property (including, without limitation, any copyright) embodied in the document accompanying this License (“Document”). Arm licenses its intellectual property in the Document to you on condition that you agree to the terms of this License. By using or copying the Document you indicate that you agree to be bound by the terms of this License.

“Subsidiary” means any company the majority of whose voting shares is now or hereafter owned or controlled, directly or indirectly, by you. A company shall be a Subsidiary only for the period during which such control exists.

This Document is **NON-CONFIDENTIAL** and any use by you and your Subsidiaries (“Licensee”) is subject to the terms of this License between you and Arm.

Subject to the terms and conditions of this License, Arm hereby grants to Licensee under the intellectual property in the Document owned or controlled by Arm, a non-exclusive, non-transferable, non-sub-licensable, royalty-free, worldwide License to:

- (i) use and copy the Document for the purpose of designing and having designed products that comply with the Document;
- (ii) manufacture and have manufactured products which have been created under the License granted in (i) above; and
- (iii) sell, supply and distribute products which have been created under the License granted in (i) above.

Licensee hereby agrees that the Licenses granted above shall not extend to any portion or function of a product that is not itself compliant with part of the Document.

Except as expressly licensed above, Licensee acquires no right, title or interest in any Arm technology or any intellectual property embodied therein.

The content of this document is informational only. Any solutions presented herein are subject to changing conditions, information, scope, and data. This document was produced using reasonable efforts based on information available as of the date of issue of this document. The scope of information in this document may exceed that which Arm is required to provide, and such additional information is merely intended to further assist the recipient and does not represent Arm’s view of the scope of its obligations. You acknowledge and agree that you possess the necessary expertise in system security and functional safety and that you shall be solely responsible for compliance with all legal, regulatory, safety and security related requirements concerning your products, notwithstanding any information or support that may be provided by Arm herein. In addition, you are responsible for any applications which are used in conjunction with any Arm technology described in this document, and to minimize risks, adequate design and operating safeguards should be provided for by you.

Reference by Arm to any third party’s products or services within this document is not an express or implied approval or endorsement of the use thereof.

THE DOCUMENT IS PROVIDED “AS IS”. ARM PROVIDES NO REPRESENTATIONS AND NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY, SATISFACTORY QUALITY, NON-INFRINGEMENT OR FITNESS FOR A PARTICULAR PURPOSE WITH RESPECT TO THE DOCUMENT. Arm may make changes to the Document at any time and without notice. For the avoidance of doubt, Arm makes no representation with respect to, and has undertaken no analysis to identify or understand the scope and content of, third party patents, copyrights, trade secrets, or other rights.

NOTWITHSTANDING ANYTHING TO THE CONTRARY CONTAINED IN THIS LICENSE, TO THE FULLEST EXTENT PERMITTED BY LAW, IN NO EVENT WILL ARM BE LIABLE FOR ANY DAMAGES, IN CONTRACT, TORT OR OTHERWISE, IN CONNECTION WITH THE SUBJECT MATTER OF THIS LICENSE (INCLUDING WITHOUT LIMITATION) (I) LICENSEE’S USE OF THE DOCUMENT; AND (II) THE IMPLEMENTATION OF THE DOCUMENT IN ANY PRODUCT CREATED BY LICENSEE UNDER THIS LICENSE). THE EXISTENCE OF MORE THAN ONE CLAIM OR SUIT WILL NOT ENLARGE OR EXTEND THE LIMIT. LICENSEE RELEASES ARM FROM ALL OBLIGATIONS, LIABILITY, CLAIMS OR DEMANDS IN EXCESS OF THIS LIMITATION.

This License shall remain in force until terminated by Licensee or by Arm. Without prejudice to any of its other rights, if Licensee is in breach of any of the terms and conditions of this License then Arm may terminate this License immediately upon giving written notice to Licensee. Licensee may terminate this License at any time. Upon termination of this License by Licensee or by Arm, Licensee shall stop using the Document and destroy all copies of the Document in its possession. Upon termination of this License, all terms shall survive except for the License grants.

Any breach of this License by a Subsidiary shall entitle Arm to terminate this License as if you were the party in breach. Any termination of this License shall be effective in respect of all Subsidiaries. Any rights granted to any Subsidiary hereunder shall automatically terminate upon such Subsidiary ceasing to be a Subsidiary.

The Document consists solely of commercial items. Licensee shall be responsible for ensuring that any use, duplication or disclosure of the Document complies fully with any relevant export laws and regulations to assure that the Document or any portion thereof is not exported, directly or indirectly, in violation of such export laws.

This License may be translated into other languages for convenience, and Licensee agrees that if there is any conflict between the English version of this License and any translation, the terms of the English version of this License shall prevail.

The Arm corporate logo and words marked with ® or ™ are registered trademarks or trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. All rights reserved. Other brands and names mentioned in this document may be the trademarks of their respective owners. No license, express, implied or otherwise, is granted to Licensee under this License, to use the Arm trade marks in connection with the Document or any products based thereon. Visit Arm's website at <http://www.arm.com/company/policies/trademarks> for more information about Arm's trademarks.

The validity, construction and performance of this License shall be governed by English Law.

Copyright © 2024 Arm Limited (or its affiliates). All rights reserved.

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

Arm document reference: PRE-21585

version 5.0, March 2024

DRAFT

Contents

Realm Host Interface specification

Realm Host Interface specification	ii
Release information	ii
Arm Non-Confidential Document License (“License”)	iii

Preface

Conventions	vii
Typographical conventions	vii
Numbers	vii
Pseudocode descriptions	vii
Assembler syntax descriptions	vii
Rules-based writing	viii
Content item identifiers	viii
Content item rendering	viii
Content item classes	viii
Additional reading	ix
Feedback	x
Feedback on this book	x

Chapter 1

Realm Host Interface (RHI)	
1.1 Overview	11
1.2 Use cases	11
1.3 Transport	12
1.4 Protocols	12

Chapter 2

RHI Implementation Features	
2.1 RHI_IMPLEMENTATION_FEATURES	14

Chapter 3

Host Session	
3.1 RHI_SESSION_VERSION	17
3.1.1 Parameters	17
3.1.2 Return values	17
3.2 RHI_SESSION_FEATURES	18
3.2.1 Parameters	18
3.2.2 Return values	18
3.3 RHI_SESSION_OPEN	19
3.3.1 Parameters	19
3.3.2 Return values	19
3.3.3 Return conditions:	19
3.3.4 Protocol state on return:	19
3.4 RHI_SESSION_CLOSE	20
3.4.1 Parameters	20
3.4.2 Return values	20
3.4.3 Return Conditions:	20
3.4.4 Protocol State on return:	20
3.5 RHI_SESSION_SEND	21
3.5.1 Return values	21
3.5.2 Return Conditions:	21
3.5.3 Protocol State on return:	21
3.6 RHI_SESSION_RECEIVE	22

3.6.1	Parameters	22
3.6.2	Return values	22
3.6.3	Return Conditions:	22
3.6.4	Protocol State on return:	22

Chapter 4

Firmware Activity Log

4.1	Introduction	23
4.1.1	Security Considerations	23
4.2	RHI_FAL_VERSION	24
4.2.1	Parameters	24
4.2.2	Return values	24
4.3	RHI_FAL_FEATURES	25
4.3.1	Parameters	25
4.3.2	Return values	25
4.4	RHI_FAL_GET_SIZE	26
4.4.1	Parameters	26
4.4.2	Return values	26
4.5	RHI_FAL_READ	27
4.5.1	Parameters	27
4.5.2	Return values	27
4.6	RHI_FAL_CLOSE	28
4.6.1	Parameters	28
4.6.2	Return values	28

Chapter 5

Device Assignment

5.1	Introduction	29
5.2	DA Types	30
5.2.1	DA_OBJECT_TYPE	30
5.3	RHI_DA_VERSION	31
5.3.1	Parameters	31
5.3.2	Return values	31
5.4	RHI_DA_FEATURES	32
5.4.1	Parameters	32
5.4.2	Return values	32
5.5	RHI_DA_OBJECT_SIZE	33
5.5.1	Parameters	33
5.5.2	Return values	33
5.6	RHI_DA_OBJECT_READ	34
5.6.1	Parameters	34
5.6.2	Return values	34

Part A Appendix A

Boot Data Synchronisation	36
Introduction	36
System Overview	36
Security Considerations	38
System Message Data Structures	39

Preface

Conventions

Typographical conventions

The typographical conventions are:

italic

Introduces special terminology, and denotes citations.

bold

Denotes signal names, and is used for terms in descriptive lists, where appropriate.

`monospace`

Used for assembler syntax descriptions, pseudocode, and source code examples.

Also used in the main text for instruction mnemonics and for references to other items appearing in assembler syntax descriptions, pseudocode, and source code examples.

SMALL CAPITALS

Used for some common terms such as IMPLEMENTATION DEFINED.

Used for a few terms that have specific technical meanings, and are included in the Glossary.

Red text

Indicates an open issue.

Blue text

Indicates a link. This can be

- A cross-reference to another location within the document
- A URL, for example <http://developer.arm.com>

Numbers

Numbers are normally written in decimal. Binary numbers are preceded by 0b, and hexadecimal numbers by 0x. In both cases, the prefix and the associated value are written in a monospace font, for example `0xFFFF0000`. To improve readability, long numbers can be written with an underscore separator between every four characters, for example `0xFFFF_0000_0000_0000`. Ignore any underscores when interpreting the value of a number.

Pseudocode descriptions

This book uses a form of pseudocode to provide precise descriptions of the specified functionality. This pseudocode is written in a monospace font. The pseudocode language is described in the Arm Architecture Reference Manual.

Assembler syntax descriptions

This book contains numerous syntax descriptions for assembler instructions and for components of assembler instructions. These are shown in a `monospace` font.

Rules-based writing

This specification consists of a set of individual *content items*. A content item is classified as one of the following:

- Declaration
- Rule
- Goal
- Information
- Rationale
- Implementation note
- Software usage

Declarations and Rules are normative statements. An implementation that is compliant with this specification must conform to all Declarations and Rules in this specification that apply to that implementation.

Declarations and Rules must not be read in isolation. Where a particular feature is specified by multiple Declarations and Rules, these are generally grouped into sections and subsections that provide context. Where appropriate, these sections begin with a short introduction.

Arm strongly recommends that implementers read *all* chapters and sections of this document to ensure that an implementation is compliant.

Content items other than Declarations and Rules are informative statements. These are provided as an aid to understanding this specification.

Content item identifiers

A content item may have an associated identifier which is unique among content items in this specification.

After this specification reaches beta status, a given content item has the same identifier across subsequent versions of the specification.

Content item rendering

In this document, a content item is rendered with a token of the following format in the left margin: L_{iiii}

- L is a label that indicates the content class of the content item.
- $iiii$ is the identifier of the content item.

Content item classes

Declaration

A Declaration is a statement that does one or more of the following:

- Introduces a concept
- Introduces a term
- Describes the structure of data
- Describes the encoding of data

A Declaration does not describe behaviour.

A Declaration is rendered with the label D .

Rule

A Rule is a statement that describes the behaviour of a compliant implementation.

A Rule explains what happens in a particular situation.

A Rule does not define concepts or terminology.

A Rule is rendered with the label *R*.

Goal

A Goal is a statement about the purpose of a set of rules.

A Goal explains why a particular feature has been included in the specification.

A Goal is comparable to a “business requirement” or an “emergent property.”

A Goal is intended to be upheld by the logical conjunction of a set of rules.

A Goal is rendered with the label *G*.

Information

An Information statement provides information and guidance as an aid to understanding the specification.

An Information statement is rendered with the label *I*.

Rationale

A Rationale statement explains why the specification was specified in the way it was.

A Rationale statement is rendered with the label *X*.

Implementation note

An Implementation note provides guidance on implementation of the specification.

An Implementation note is rendered with the label *U*.

Software usage

A Software usage statement provides guidance on how software can make use of the features defined by the specification.

A Software usage statement is rendered with the label *S*.

Additional reading

This section lists publications by Arm and by third parties.

See Arm Developer (<http://developer.arm.com>) for access to Arm documentation.

[1] *Realm Management Monitor specification*. (ARM DEN 0137) Arm Limited.

[2] *Arm SMC Calling Convention*. (ARM DEN 0028 D) Arm Ltd.

[3] *Live Firmware Activation SMC Interface*. (ARM DEN 0147) Arm Ltd.

[4] *NIST Special Publication 800-56A. Recommendation for Pair-Wise Key-Establishment Schemes Using Discrete Logarithm Cryptography*. See <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-56Ar3.pdf>

[5] *Introducing Arm CCA*. (ARM DEN 0125) Arm Limited.

[6] *IANA Hash Function Textual Names*. See <https://www.iana.org/assignments/hash-function-text-names/hash-function-text-names.xhtml>

[7] *RME system architecture specification*. (ARM DEN 0129) Arm Ltd.

Feedback

Arm welcomes feedback on its documentation.

Feedback on this book

If you have comments on the content of this book, send an e-mail to errata@arm.com. Give:

- The title (Realm Host Interface specification).
- The number (DEN0148 1.0-alp2).
- The page numbers to which your comments apply.
- The rule identifiers to which your comments apply, if applicable.
- A concise explanation of your comments.

Arm also welcomes general suggestions for additions and improvements.

Note

Arm tests PDFs only in Adobe Acrobat and Acrobat Reader, and cannot guarantee the appearance or behavior of any document when viewed with any other PDF reader.

DRAFT

Chapter 1

Realm Host Interface (RHI)

1.1 Overview

I RHI enables software running within Realms to request specific services or data items from the non-secure host. This is in contrast to the Realm Services Interface (RSI) which provides access to services or data provided by the Realm Management Monitor (RMM) [1].

RHI supports sets of commands dedicated to specific access cases. This specification covers the usage and call patterns for these commands.

1.2 Use cases

I There are use cases where it may be necessary for Realm software to communicate with services executing within the context of the non-secure host responsible for the management of Realms. Use cases may fall into the following categories:

- To access data relevant to the state of the host platform
- To access a service that maps some state or resource controlled by the host to be accessed by the specific realm.

The inclusion of different sets of commands may depend upon the use cases addressed by a deployment. Some command sets have specific uses tied into the programming model required to access deployed features and support use of the Realm Services Interface. Others are more open ended and will require more usage explanation within the context of a given deployment.

Note

Data received from such host calls is provided from outside of the Trusted Computing Base for Realms. As such, it is necessary for a use case specific security mechanism to be available for any data received.

1.3 Transport

- R All RHI calls use the RSI_HOST_CALL command as a transport.
- I Within the RsiHostCall data structure, the gprs[0] value is an SMCCC Function Identifier (FID) which specifies the function to be called.
- I The FID range for RHI is reserved to be 0xC500_0040 – 0XC500_013F.
See also:
 - Realm Management Monitor (RMM) architecture specification [1]
 - Arm SMCCC calling convention [2]
- I In the description below of the RHI commands, parameters are listed corresponding to members of the RsiHostCall data structure.

1.4 Protocols

RHI commands are organised into a set of protocols, each identified by a numerical index. The table below summarises the protocols defined in this version of the specification.

RHI Protocol Set	Index
Host Session	0x1
FW Activity Log	0x2
Device Assignment	0x3

Chapter 2

RHI Implementation Features

- G The RHI_IMPLEMENTATION_FEATURES interface can be used to determine whether sets of RHI protocols supported by the current implementation.
- R The RHI_IMPLEMENTATION_FEATURES call must be supported by an implementation.

2.1 RHI_IMPLEMENTATION_FEATURES

I The RHI_IMPLEMENTATION_FEATURES call is used to test whether an RHI protocol set is supported by the host implementation. The RHI protocol set values to be tested are found in the [interface summary table](#)

Within each protocol set, there is a Features ABI to determine more information about the supported calls within the protocol.

Parameters

Argument	Type	Description
Imm	UInt16	Immediate, value 0
gprs[0]	UInt64	FID, value 0xC500_0040
gprs[1]	UInt64	RHI Protocol set value to query

Return values

Argument	Type	Value
gprs[0]	bool	if value is TRUE, the implementation supports this protocol set

Chapter 3

Host Session

G The following collection of calls are used to establish and use a communication channel between the Realm and an external entity. This is a general mechanism, which is specialised for a specific purpose by the data passed through the channel.

I The communication channel is created via a service running in the non-secure host. This service will determine the relevant external entity, for example a client of the host service managing the Realm.

Note that the channel does not provide any security mechanism. Security must be provided within any exchange of messages through the channel.

I For an example of a specific protocol using this communication channel, see [Appendix A: Boot Data Synchronisation](#)

I The communication channel can be created in one of two modes:

- BLOCKING: ABI calls do not return until communications are complete
- NON-BLOCKING: ABI calls return immediately and may need to be repeated until communications are complete.

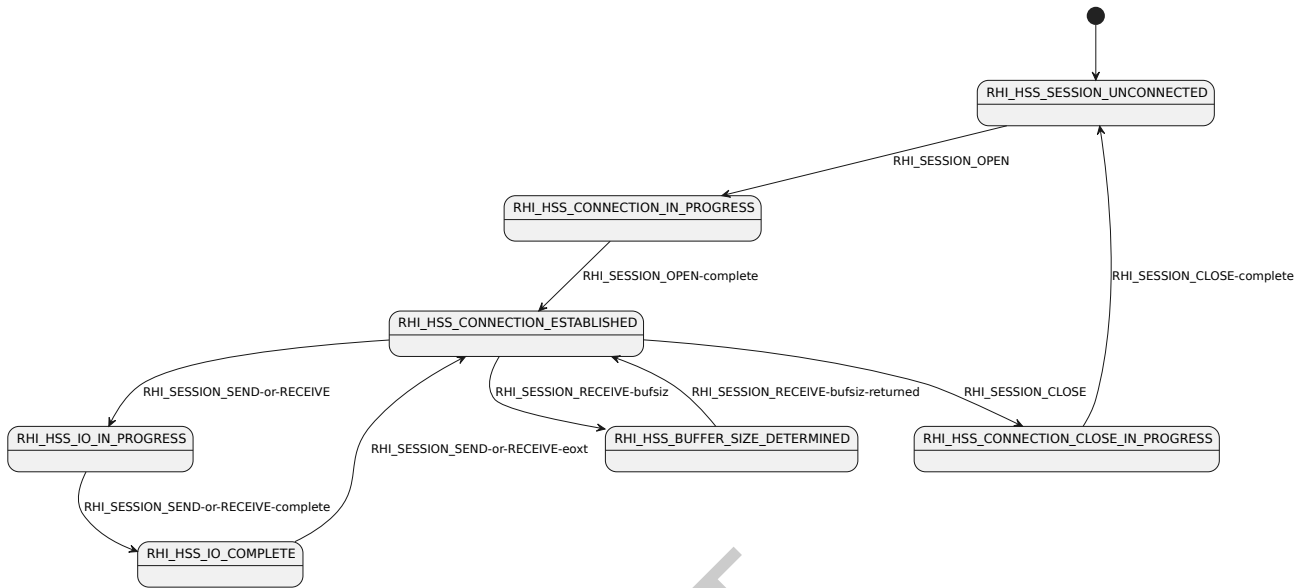
It is IMPLEMENTATION DEFINED whether either communication type is supported

I The Host Session ABIs cause an internal Host Session State (HSS) to be maintained for a connection. This state which can be used to determine when communications are complete. The states that a connection can be in are:

- RHI_HSS_SESSION_UNCONNECTED
- RHI_HSS_CONNECTION_IN_PROGRESS
- RHI_HSS_CONNECTION_ESTABLISHED
- RHI_HSS_IO_IN_PROGRESS
- RHI_HSS_IO_COMPLETE
- RHI_HSS_BUFFER_SIZE_DETERMINED

- RHI_HSS_CONNECTION_CLOSE_IN_PROGRESS

State Diagram:



U

The constants above can be implemented via an enumerated type or by assigning values which are IMPLEMENTATION DEFINED.

DRAFT

3.1 RHI_SESSION_VERSION

The RHI_SESSION_VERSION ABI returns the implemented numeric version of the RHI_SESSION calls within this protocol.

R If the protocol set is reported as supported via the [Supported Protocols ABI](#), the RHI_SESSION_VERSION ABI must be implemented.

3.1.1 Parameters

Argument	Type	Description
Imm	UInt16	Immediate, value 0
gprs[0]	UInt64	FID, value 0xC500_0041

3.1.2 Return values

Argument	Type	Description
gprs[0]	UInt64	Protocol Set version

DRAFT

3.2 RHI_SESSION_FEATURES

The features ABI provides implementation details for the RHI_SESSION calls within this protocol.

R If the protocol set is reported as supported via the [Supported Protocols ABI](#), the RHI_SESSION_FEATURES ABI must be implemented.

3.2.1 Parameters

Argument	Type	Description
Imm	UInt16	Immediate, value 0
gprs[0]	UInt64	FID, value 0xC500_0042

3.2.2 Return values

Argument	Type	Description
gprs[0]	UInt64	ABI calls supported bitmap
gprs[1]	UInt64	Connection modes supported bitmap

- ABI calls supported: each bit set in this bitmap indicates that the corresponding call is supported by the host implementation.
 - Bit 0: RHI_SESSION_OPEN supported
 - Bit 1: RHI_SESSION_CLOSE supported
 - Bit 2: RHI_SESSION_SEND supported
 - Bit 3: RHI_SESSION_RECEIVE supported
- Connection Modes supported: indicates which Connection Modes are supported by the host implementation.
 - Bit 0: BLOCKING connection mode supported
 - Bit 1: NON-BLOCKING connection mode supported

3.3 RHI_SESSION_OPEN

Call initiates a communication channel between the Realm and the host determined external entity

3.3.1 Parameters

If the `connectionType` is `NON-BLOCKING`, this behaviour also applies to `RHI_SESSION_OPEN` operations and the call will return immediately before the host has established any connection with the external entity. The protocol state will change to `RHI_HSS_CONNECTION_IN_PROGRESS` to indicate this state. `RHI_SESSION_OPEN` can then be called again until the protocol state changes to `RHI_HSS_CONNECTION_ESTABLISHED` (or error). For these subsequent calls, the `SessionID` returned from the first call must be specified. If the protocol state is `'RHI_HSS_SESSION_UNCONNECTED'` when the `RHI_SESSION_OPEN` call is made, then the `SessionID` parameter is ignored.

If the `connectionType` is `BLOCKING`, the call will not return until the host has established the connection with the external entity (or error).

Argument	Type	Description
<code>Imm</code>	<code>UInt16</code>	Immediate, value 0
<code>gprs[0]</code>	<code>UInt64</code>	FID, value 0xC500_0043
<code>gprs[1]</code>	<code>UInt64</code>	SessionID
<code>gprs[2]</code>	<code>connectionType</code>	<code>BLOCKING</code> or <code>NON-BLOCKING</code>

3.3.2 Return values

Argument	Type	Description
<code>gprs[0]</code>	<code>UInt64</code>	Return Code
<code>gprs[1]</code>	<code>UInt64</code>	SessionID for the new channel
<code>gprs[2]</code>	<code>UInt64</code>	Protocol state for the session

3.3.3 Return conditions:

- `RHI_SESS_SUCCESS`
- `RHI_SESS_PEER_NOT_AVAILABLE`: host could not establish a connection to external entity
- `RHI_SESS_INVALID_STATE_FOR_OPERATION`: state is not `RHI_HSS_SESSION_UNCONNECTED`
- `RHI_SESS_INVALID_SESSION_ID`: 0 not passed on initial call or unknown `SessionID` passed (`NON_BLOCKING`)
- `RHI_SESS_CONNECTION_TYPE_NOT_SUPPORTED`: the implementation does not support this connection type

3.3.4 Protocol state on return:

- `BLOCKING`: `RHI_HSS_CONNECTION_ESTABLISHED` or `connection_unconnected` (on error)
- `NON-BLOCKING`: `RHI_HSS_CONNECTION_IN_PROGRESS` on initial call, this state remains on subsequent calls until `RHI_HSS_CONNECTION_ESTABLISHED` or `connection_unconnected` (error)

3.4 RHI_SESSION_CLOSE

Call closes a previously opened communication channel.

3.4.1 Parameters

Argument	Type	Description
Imm	UInt16	Immediate, value 0
gprs[0]	UInt64	FID, value 0xC500_0044
gprs[1]	UInt64	Channel SessionID

3.4.2 Return values

Argument	Type	Description
gprs[0]	UInt64	Return Code
gprs[1]	UInt64	SessionID
gprs[2]	UInt64	Protocol state for the session

3.4.3 Return Conditions:

- RHI_SESS_SUCCESS
- RHI_SESS_INVALID_STATE_FOR_OPERATION: state is RHI_HSS_SESSION_UNCONNECTED
- RHI_SESS_PEER_NOT_AVAILABLE: host could not gracefully close session
- RHI_SESS_INVALID_SESSION_ID: unknown SessionID parameter

3.4.4 Protocol State on return:

- BLOCKING: connection_unconnected (RHI_SESS_SUCCESS) or RHI_HSS_CONNECTION_ESTABLISHED (on error)
- NON-BLOCKING: state changes to RHI_HSS_CONNECTION_CLOSE_IN_PROGRESS and the call returns immediately. This state remains on subsequent calls until state becomes connection_unconnected or an error is returned.

3.5 RHI_SESSION_SEND

Call transmits data on previously opened communication channel. Overlapped calls are not supported.

3.5.0.1 Parameters

Argument	Type	Description
Imm	UInt16	Immediate, value 0
gprs[0]	UInt64	FID, value 0xC500_0045
gprs[1]	UInt64	SessionID of channel
gprs[2]	Address	Realm IPA for buffer containing data, granule aligned
gprs[3]	UInt64	Length of data to send in bytes
gprs[4]	Offset	Offset in buffer from which to send data

3.5.1 Return values

Argument	Type	Description
gprs[0]	UInt64	Return Code
gprs[1]	UInt64	SessionID
gprs[2]	UInt64	Protocol state for the session
gprs[3]	UInt64	Length of data transmitted in bytes

3.5.2 Return Conditions:

- RHI_SESS_SUCCESS: if BLOCKING, operation ended. If NON-BLOCKING, see Protocol State
- RHI_SESS_INVALID_STATE_FOR_OPERATION: valid states are RHI_HSS_CONNECTION_ESTABLISHED or RHI_HSS_IO_COMPLETE, for first call, or in RHI_HSS_IO_IN_PROGRESS state (NON-BLOCKING)
- RHI_SESS_INVALID_SESSION_ID: unknown SessionID parameter
- RHI_SESS_PEER_NOT_AVAILABLE: communication error to external entity
- RHI_SESS_ACCESS_FAILED: buffer is not readable or is not granule aligned

3.5.3 Protocol State on return:

- BLOCKING: RHI_HSS_IO_COMPLETE (RHI_SESS_SUCCESS) or RHI_HSS_CONNECTION_ESTABLISHED (on error)
- NON-BLOCKING: RHI_HSS_IO_IN_PROGRESS on initial call, this state remains on subsequent calls until RHI_HSS_IO_COMPLETE (whether RHI_SESS_SUCCESS or error condition)

3.6 RHI_SESSION_RECEIVE

Call reads data from Communication channel.

3.6.1 Parameters

Argument	Type	Description
Imm	UInt16	Immediate, value 0
gprs[0]	UInt64	FID, value 0xC500_0046
gprs[1]	UInt64	SessionID of target channel
gprs[2]	Address	Realm IPA for buffer used to receive data, granule aligned
gprs[3]	UInt64	Size of receiving data buffer
gprs[4]	Offset	Offset in buffer where received data is to be written

If gprs[3] (buffer size) and gprs[2] (buffer address) parameters are both set to 0, then the call is intended to determine the size of data buffer required, which will be returned in gprs[2].

If only gprs[3] (buffer size) is 0, then no data will be read.

3.6.2 Return values

Argument	Type	Description
gprs[0]	UInt64	Return Code
gprs[0]	UInt64	SessionID
gprs[1]	UInt64	Protocol state for the session
gprs[2]	UInt64	Data length received (or buffer size required)

3.6.3 Return Conditions:

- RHI_SESS_SUCCESS: if BLOCKING, operation ended. If NON-BLOCKING, see Protocol State
- RHI_SESS_INVALID_STATE_FOR_OPERATION: valid states are RHI_HSS_CONNECTION_ESTABLISHED or RHI_HSS_IO_COMPLETE, for first call, or in RHI_HSS_IO_IN_PROGRESS state (NON-BLOCKING)
- RHI_SESS_INVALID_SESSION_ID: unknown SessionID parameter
- RHI_SESS_PEER_NOT_AVAILABLE: communication error to external entity
- RHI_SESS_ACCESS_FAILED: buffer is not writeable or is not granule aligned

3.6.4 Protocol State on return:

- RHI_HSS_BUFFER_SIZE_DETERMINED if 0 values passed for buffer address and size
- BLOCKING: RHI_HSS_IO_COMPLETE (RHI_SESS_SUCCESS) or RHI_HSS_CONNECTION_ESTABLISHED (on error)
- NON-BLOCKING: RHI_HSS_IO_IN_PROGRESS on initial call, this state remains on subsequent calls until RHI_HSS_IO_COMPLETE (whether RHI_SESS_SUCCESS or error condition)

Chapter 4

Firmware Activity Log

4.1 Introduction

G These interfaces are used to obtain the Firmware Activity Log (FAL). This log reflects the Firmware changes made to the CCA Platform. For further details see the Live Firmware Activation specification [3]

I The Firmware Activity Log contains a series of entries that describe changes applied to the firmware that makes up the CCA Platform. The initial state of the firmware deployed on the CCA Platform is captured at boot time. Updates to individual components may be made post boot, which will be captured as additional entries in the Firmware Activity Log. The Firmware Activity Log is maintained within the non-secure host system. The format and contents of the log is IMPLEMENTATION DEFINED and is intended to be passed to an enlightened verifier component to support the trustworthiness appraisal of the CCA Platform token. A typical entry within the log would include at least the following information about a firmware component:

- Component Identity
- Cryptographic measurement of the component in memory
- Identity of the signing authority public key
- The Security Version of the component maintained by Live Firmware Activation

I 4.1.1 Security Considerations

The Firmware Activity Log is an important part of the evidence used by a Relying Party to established the trustworthiness of a CCA Platform instance. The RHI interface does not provide any security guarantees for integrity of the data read from the non-secure Host. Data integrity for the log is established by the verifier, using a measurement entry within the CCA Platform attestation token [1]. This measurement is a compound value, established by extending a hash measurement of each entry within the Log. The verifier can recompute this value from the Log entries it receives, and match the attestation report entry to ensure that the log contents are as expected.

4.2 RHI_FAL_VERSION

The RHI_FAL_VERSION ABI returns the implemented numeric version of the RHI_FAL calls within this protocol.

R If the protocol set is reported as supported via the [Supported Protocols ABI](#), the RHI_FAL_VERSION ABI must be implemented.

4.2.1 Parameters

Argument	Type	Description
Imm	UInt16	Immediate, value 0
gprs[0]	UInt64	FID, value 0xC500_0047

4.2.2 Return values

Argument	Type	Description
gprs[0]	UInt64	Protocol Set version

DRAFT

4.3 RHI_FAL_FEATURES

The features ABI provides implementation details for the FAL calls within this protocol.

- R If the protocol set is reported as supported via the [Supported Protocols ABI](#), the FAL_FEATURES ABI must be implemented.

4.3.1 Parameters

Argument	Type	Description
Imm	UInt16	Immediate, value 0
gprs[0]	UInt64	FID, value 0xC500_0048

4.3.2 Return values

Argument	Type	Description
gprs[0]	UInt64	ABI calls supported bitmap

- ABI calls supported: each bit set in this bitmap indicates that the corresponding call is supported by the host implementation.
 - Bit 0: RHI_FAL_GET_SIZE supported
 - Bit 1: RHI_FAL_READ supported
 - Bit 2: RHI_FAL_CLOSE supported

4.4 RHI_FAL_GET_SIZE

The RHI_FAL_GET_SIZE ABI returns the overall size of the FW Activity Log.

4.4.1 Parameters

Argument	Type	Description
Imm	UInt16	Immediate, value 0
gprs[0]	UInt64	FID, value 0xC500_0049

4.4.2 Return values

Argument	Type	Description
gprs[0]	UInt64	Size in bytes of the FW Activity Log

DRAFT

4.5 RHI_FAL_READ

The RHI_FAL_READ ABI fetches the contents of the Firmware Activity Log

4.5.1 Parameters

Argument	Type	Description
Imm	UInt16	Immediate, value 0
gprs[0]	UInt64	FID, value 0xC500_004A
gprs[1]	Address	IPA for Receiving Buffer, granule aligned
gprs[2]	UInt64	Offset within buffer where data copy starts
gprs[3]	UInt64	Max length of buffer memory

4.5.2 Return values

Argument	Type	Description
gprs[0]	UInt64	Return code
gprs[1]	UInt64	Size in bytes written to buffer
gprs[2]	UInt64	Remaining size of FAL in bytes to be read

- Return Code:
 - RHI_FAL_SUCCESS
 - RHI_FAL_ACCESS_FAILED: buffer is not writable or not granule aligned

The ABI returns the number of bytes written to the buffer in gprs[1]. This may be fewer than the buffer size if the write is the last portion of the log to be transferred. If the number of bytes written is 0, this indicates that the full log has been transferred.

4.6 RHI_FAL_CLOSE

The RHI_FAL_CLOSE ABI indicates to the host that the FAL has been fully read

4.6.1 Parameters

Argument	Type	Description
Imm	UInt16	Immediate, value 0
gprs[0]	UInt64	FID, value 0xC500_004B

4.6.2 Return values

Argument	Type	Description
gprs[0]	UInt64	Return code, value RHI_FAL_SUCCESS

DRAFT

Chapter 5

Device Assignment

5.1 Introduction

- G Device Assignment (DA) This protocol set supplies ABI calls used to retrieve Device evidence cached within the Non Secure Host.
- I Before these RHI calls can be used to retrieve cached data, relevant RSI calls need to be made to prime the cache. Failure to make these RSI calls can result in error returns from the RHI calls below. For more details, see the ‘Realm Device Assignment’ chapter of [1].

5.2 DA Types

5.2.1 DA_OBJECT_TYPE

- I The Device Assignment ABIs are used to access multiple object types. The supported types of object are:
- RHI_DA_OBJECT_CERTIFICATE
 - RHI_DA_OBJECT_MEASUREMENT
 - RHI_DA_OBJECT_INTERFACE_REPORT
- RHI_DA_OBJECT_CERTIFICATE objects have PDEV scope.
- RHI_DA_OBJECT_MEASUREMENT objects have either PDEV or VDEV scope.
- RHI_DA_OBJECT_INTERFACE_REPORT objects have VDEV scope.
- U The constants above can be implemented via an enumerated type or by assigning values which are TBD.

DRAFT

5.3 RHI_DA_VERSION

The RHI_DA_VERSION ABI returns the implemented numeric version of the RHI_DA calls within this protocol.

R If the protocol set is reported as supported via the [Supported Protocols ABI](#), the RHI_DA_VERSION ABI must be implemented.

5.3.1 Parameters

Argument	Type	Description
Imm	UInt16	Immediate, value 0
gprs[0]	UInt64	FID, value 0xC500_004C

5.3.2 Return values

Argument	Type	Description
gprs[0]	UInt64	Protocol Set version

DRAFT

5.4 RHI_DA_FEATURES

The features ABI provides implementation details for the DA calls within this protocol.

- R If the protocol set is reported as supported via the [Supported Protocols ABI](#), the DA_FEATURES ABI must be implemented.

5.4.1 Parameters

Argument	Type	Description
Imm	UInt16	Immediate, value 0
gprs[0]	UInt64	FID, value 0xC500_004D

5.4.2 Return values

Argument	Type	Description
gprs[0]	UInt64	ABI calls supported bitmap

- ABI calls supported: each bit set in this bitmap indicates that the corresponding call is supported by the host implementation.
 - Bit 0: RHI_DA_OBJECT_SIZE supported.
 - Bit 1: RHI_DA_OBJECT_READ supported.

5.5 RHI_DA_OBJECT_SIZE

The RHI_DA_OBJECT_SIZE ABI returns the size of the relevant object for the requested VDEV_ID. VDEV_ID is the ID the host used for VDEV_CREATE.

5.5.1 Parameters

Argument	Type	Description
Imm	UInt16	Immediate, value 0
gprs[0]	UInt64	FID, value 0xC500_004E
gprs[1]	UInt64	VDEV_ID
gprs[2]	DA_OBJECT_TYPE	Type of the target object

5.5.2 Return values

Argument	Type	Description
gprs[0]	UInt64	Return code
gprs[1]	UInt64	Size in bytes of the object

- Return Code:
 - RHI_DA_SUCCESS.
 - RHI_ERROR_INVALID_VDEV_ID: VDEV_ID parameter not found.
 - RHI_ERROR_INVALID_DA_OBJECT_TYPE: invalid DA_OBJECT_TYPE.
 - RHI_ERROR_DATA_NOT_AVAILABLE: required priming RSI call for object has not been made.

5.6 RHI_DA_OBJECT_READ

The RHI_DA_OBJECT_READ ABI fetches a relevant object for a requested VDEV_ID and writes it to a buffer in NS PAS. VDEV_ID is the ID the host used for VDEV_CREATE.

5.6.1 Parameters

VDEV_ID is the ID the host used for VDEV_CREATE.

Argument	Type	Description
Imm	UInt16	Immediate, value 0
gprs[0]	UInt64	FID, value 0xC500_004F
gprs[1]	UInt64	VDEV_ID
gprs[2]	DA_OBJECT_TYPE	Type of the target object
gprs[3]	UInt64	Offset within buffer where data copy starts
gprs[4]	UInt64	Max length of buffer memory
gprs[5]	UInt64	IPA of buffer for read object

5.6.2 Return values

Argument	Type	Description
gprs[0]	UInt64	Return code
gprs[1]	UInt64	Size in bytes of data read

- Return Code:
 - RHI_DA_SUCCESS.
 - RHI_ERROR_INVALID_VDEV_ID: VDEV_ID parameter not found.
 - RHI_ERROR_INVALID_DA_OBJECT_TYPE: invalid DA_OBJECT_TYPE.
 - RHI_ERROR_DATA_NOT_AVAILABLE: required priming RSI call for object has not been made.
 - RHI_ERROR_INVALID_OFFSET: invalid offset for object size.
 - RHI_ERROR_INVALID_ADDR: invalid address for PA.

DRAFT

Part A
Appendix A

Boot Data Synchronisation

Introduction

This section describes a protocol for synchronising Realm boot with communications required to obtain parameters or secret blocks that can be injected into the boot sequence. This ‘Boot Sync Blocks’ Protocol (BSB Protocol) uses the [RHI_SESSION_* calls](#) as a transport.

The main use case for this BSB protocol is to inject values used in the early boot sequence of Realm firmware. Such values could be UEFI parameters or secrets used to protect user data. Use of the BSB protocol may be required because:

- there is no network stack available in the early stages of boot
- in order to maintain confidential computing guarantees, these values need to be provided by the User Context of the Realm initiator, which is known by the host service context, but difficult to establish for the Realm code at this early stage of boot

In the BSB protocol, messages are exchanged between Realm code and a User Context using a transport provided by the Host Session RHI calls. The messages have a common encoding, which includes an identifier for the type of message defined within the BSB protocol and a message specific data block. The BSB protocol does not identify or restrict the types of values transferred, this can be use case specific. As the Host Interface provides no security guarantees with respect to any data passed, much of the BSB protocol described here bootstraps an encrypted channel between the endpoints to protect the confidentiality and integrity of data transferred through the BSB protocol.

System Overview

There are multiple phases to the communication through the BSB protocol.

- In the first phase, the two communicating parties calculate the secrets required to create a Diffie-Hellman (DH) [4] key exchange and share them with each other.
- In the second phase, once shared keys have been calculated, the Realm sends the User Context an Attestation Report that the User Context can validate to confirm that communication is really to a Realm.
- After this point, the channel can be used to supply the relevant data required to specialise the boot sequence, using the shared key to protect the data.

Phase 1 - protected channel establishment

Within the Realm code, appropriate parameters are chosen for the Diffie-Hellman exchange. The Realm then generates a random value and uses the chosen DH parameters to create a public value R . The Realm code uses the [RHI_SESSION_OPEN ABI](#) to open a Host controlled session to the User Context. The host service is responsible for routing the message connection to a relevant service.

In the communication flow description below ‘*Sent*’ refers to the the Realm code using the [RHI_SESSION_SEND ABI](#) and ‘*Receive state*’ refers to the Realm code using the [RHI_SESSION_RECEIVE ABI](#), both referencing the SessionID received from the [RHI_SESSION_OPEN](#). All messages within the BSB protocol use data structures with a common header to encode the message type.

The initial message *Sent* uses a `BOOT_SYNC_KEY_XCHG_REQ` data structure. This contains the DH parameters and the value of R . The User Context service uses the key exchange request parameters provided, along with another generated random value, to create a public value U . The public values U and R are used to compute a shared secret K . That K is used to derive an encryption key, K_E and a binding key, K_B . The User Context prepares a `BOOT_SYNC_KEY_XCHG_RESP` data structure containing the value of U , and responds with this data to the non-secure service. The non-secure service forwards on this data to the Realm. In the meantime, the Realm is in *Receive state* until the `BOOT_SYNC_KEY_XCHG_RESP` data structure is delivered. Once available, U is extracted and the Realm performs the same cryptographic operations as the User Context (using U and R to compute K , then deriving K_E and K_B). Both end parties in the communication should now have identical keys

which can be used to protect communication in further phases.

Phase 2 - establishment of trust

Once the communication pathway is established, the User Context can establish the trustworthiness of the Realm. The Realm code uses the `RSI_ATTESTATION_TOKEN_INIT` and `RSI_ATTESTATION_TOKEN_CONTINUE` ABIs to obtain an Attestation Report (AR). The first of these, `RSI_ATTESTATION_TOKEN_INIT`, requires a `Challenge` parameter that establishes the context for the Report request. The `Challenge` used in this BSB protocol is a hash of the computed binding key (`KB`). The hash algorithm used here is fixed for the overall BSB protocol as it will also be used within the User Context. The resultant report is encrypted using the computed encryption key (`KE`) and a fixed algorithm for the BSB protocol. The encrypted data is packaged into a `BOOT_SYNC_ATTESTATION_REQUEST` data structure and *Sent* to the User Context. The User Context extracts the data from the `BOOT_SYNC_ATTESTATION_REQUEST` data structure and uses its computed encryption key, `KE`, to decrypt AR. AR is then verified for correctness using an Attestation Service. As well as overall correctness of AR the verification operation should check:

- that AR was produced by a CCA Platform determined to be trustworthy by an applicable policy
- that code measurements within AR match those for the expected (guest firmware) code in the Realm
- that `Challenge` claim within AR matches a hash of the binding key, `KB`, that the User Context computed.

If all of the above checks pass, the User Context can determine that the Realm is trustworthy to received the (confidential) Boot Data. The verification status, and optionally data from the verification result are packaged in a `BOOT_SYNC_ATTESTATION_RESPONSE` data structure. The Realm code waits in *Receive State* for the confirmation that the trust exchange has been approved.

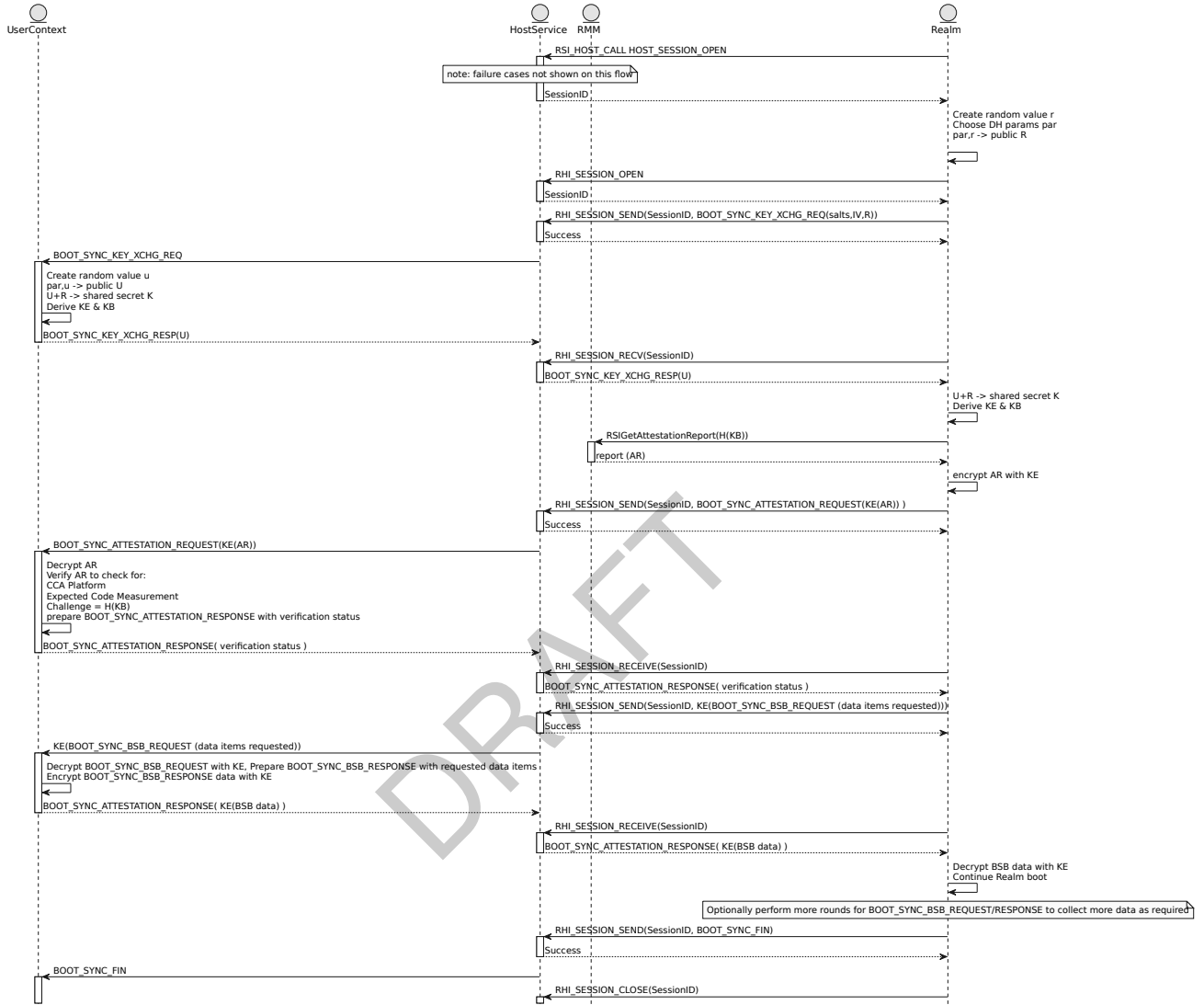
Phase 3 - transfer of boot information

The Realm can now make requests for the data required to continue the boot process (e.g UEFI variable data or secrets required to access encrypted content). Data requests are made by the Realm having *Sent* a `BOOT_SYNC_BSB_REQUEST` data structure and waiting in *Receive State* for a `BOOT_SYNC_BSB_RESPONSE` data structure to be sent from the User Context. The `BOOT_SYNC_BSB_REQUEST` data structure can have a variable number of elements, depending upon the data required to continue the Realm boot. This data can be supplied in a single communication round or in multiple communication rounds.

Once all communications are complete, a `BOOT_SYNC_FIN` data structure can be *Sent* by the Realm to signal

that no more data is required. No reply is expected to this message.

Flow Diagram



Security Considerations

The data path established by the RHI HOST_SESSION... ABI calls provides no security guarantees. Any message on this path could be inspected or modified by the host infrastructure that creates the communication channel. To mitigate this, the first phase of the BSB protocol establishes a secure channel, using a Diffie-Hellman exchange such that the two ends of the channel both compute protection keys without the intermediate party knowing these.

Diffie-Hellman exchanges are susceptible to person-in-the-middle attacks, where a component between the endpoints establishes separate secure connections with each party but then has access to all the communications traffic. To mitigate this possibility, the Realm supplies an Attestation Report, with bound data obtained from a binding key computed from the DH shared secret. Verification of the Attestation Report will confirm that it has not been tampered with in transmission and the bound data can be checked to ensure it matches the User Context value for the binding key. This mitigation allows detection of the above attack as it would result in different binding keys for each of the end parties.

The threat model for the BSB protocol may include data injection from an in-the-middle attacker that could insert malfaesant code in order to reveal later supplied data. To mitigate this, the BSB protocol can work in multiple

rounds. In this mode, the Realm state is updated with measurements of data supplied (apparently) from the User Context and further attestation reports are supplied in sequence. This allows the User Context to further validate the state of the Realm before supplying sensitive data.

Verification of the Attestation Report also allows the User Context to confirm that the other end of the connection is running under a CCA Platform and with expected software content in the Realm.

System Message Data Structures

Cryptography Algorithm considerations

The data structures below are structured to support a BSB protocol using the following cryptographic algorithms. If other algorithms are used to construct a similar BSB protocol then it is likely that some of the fields would change. The likely candidates for alteration have the variable names below annotated with (*). Array sizes are as appropriate for the algorithms used.

Algorithms:

- ECDH key using the ECC Curve-P384 for key exchange
- AEAD AES-GCM for encryption
- SHA512 HMAC-based Extract-and-Expand Key Derivation Function (HKDF) for key Derivation
- SHA256 for IV rolling hash

EFI_GUID

The BSB protocol reuses the EFI_GUID structure to define a 128-bit unique identifier value. EFI_GUID is commonly defined in several code bases.

Type	Variable
UINT32	Data1
UINT16	Data2
UINT16	Data3
UINT8[8]	Data4

BOOT_SYNC_GUID_BLOB

Common data structure prefix for all BSB protocol messages

Type	Variable	Notes
EFI_GUID	Name	GUID identifying message type
UINT32	Length	Length in bytes of full data message

GUID definitions for BSB protocol messages

Implementation GUIDs used to for message identities within the following data structures.

GUID Name	Value
gArmBootSyncKeyEncData	EAB79650-5746-4E46-9EC4-0BDF3D148A1E
gArmBootSyncKeyXchgReqGuid	BBD5E1D8-C8E9-48CB-A850-A30B15D08A22

GUID Name	Value
gArmBootSyncKeyXchgRespGuid	D83BF2F8-6B49-4238-859D-7E4C29150995
gArmBootSyncAttReqGuid	A31E8A1B-5D80-4336-8C0D-6F653B0CC8D1
gArmBootSyncAttRespGuid	1B3C3C27-51E1-4D1B-9C44-189EAEA48263
gArmBootSyncAttReport	59FC4FCE-B2B2-4DB6-A0CD-3053D9F49738
gArmBootSyncAttResult	FE4A5C90-FEC5-4029-B515-699A872E3B01
gArmBootSyncBsbReqGuid	60E65392-591A-43A4-98E8-257985B9FEB0
gArmBootSyncBsbRespGuid	0DA1DE44-D38D-40E2-9D0B-A4BAD90B1A5A
gArmBootSyncRequestOptions	54E1D918-311F-4F3F-B775-9A74A039C438
gArmBootSyncVarData	1DB974DF-3F49-44EB-B324-3CB7BA00F589
gArmBootSyncSecretData	9755286D-E064-41B4-8FC2-54101280525C
gArmBootSyncFinGuid	AD96854E-794C-43CA-B91D-61DE98313E45
gArmBootSyncNackGuid	7731492A-093E-49ED-8A54-8AD6B8CEC450

BOOT_SYNC_ENCRYPTED_DATA

Common data structure prefix for all BSB protocol messages that contain an encrypted payload. This data structure will be followed by EncDataLength bytes of encrypted data that must be decoded with the BSB protocol KE before being interpreted as the relevant payload data structure.

Type	Variable	Notes
BOOT_SYNC_GUID_BLOB		GUID: gArmBootSyncKeyEncData
UINT8[TAG_SIZE]	Tag*	AES GCM tag
UINT32	EncDataLength	(encrypted) data size beyond this header

BOOT_SYNC_BSB_HEADER

Common data structure used to define the structure of a multi element data block used in the BSB protocol. The data structure identifies the type and the number of elements contained within the message. This data structure will be followed by ElementCount number of data structures which will need interpreting per identified type.

Type	Variable	Notes
BOOT_SYNC_GUID_BLOB		GUID: identifies request or response message
UINT32	ElementCount	Number of following BOOT_SYNC_BSB_ELEMENT structures

BOOT_SYNC_BSB_ELEMENT

Common data structure used to identify and package a data packet within a BSB protocol message. This data structure will be followed by the data for the payload (length of message is within GUID_BLOB).

Type	Variable	Notes
	BOOT_SYNC_GUID_BLOB	GUID identifies the relevant payload

BOOT_SYNC_KEY_XCHG_REQ

Datastructure sent from Realm to User Context to initiate the BSB protocol Key Exchange

Type	Variable	Notes
	BOOT_SYNC_GUID_BLOB	GUID: gArmBootSyncKeyXchgReqGuid
UINT32	Version	Protocol version in Realm
UINT8[SALT_SIZE]	SaltKeyBinding*	Salt for Binding Key
UINT8[SALT_SIZE]	SaltKeyEncryption*	Salt for Encryption Key
UINT8[IV_SIZE]	IV*	Initialisation vector
UINT32	PEMDataLen	Size of Realm DH Public value (PEM format)
UINT8[]	PEMData	The fixed fields of this structure are followed by PEMDataLen bytes of PEM data

BOOT_SYNC_KEY_XCHG_RESP

Data structure sent from User Context to Realm to finalise the BSB protocol Key Exchange

Type	Variable	Notes
	BOOT_SYNC_GUID_BLOB	GUID: gArmBootSyncKeyXchgRespGuid
UINT32	Version	Protocol version in User Context
UINT32	PEMDataLen	Size of User Context DH Public value (PEM format)
UINT8[]	PEMData	The fixed fields of this structure are followed by PEMDataLen bytes of PEM data

BOOT_SYNC_ATTESTATION_REQUEST

Datastructure sent from Realm to User Context, containing the Attestation Report that can be verified to perform the trustworthiness step.

Type	Variable	Notes
	BOOT_SYNC_ENCRYPTED_DATA	EncDataHeader
	BOOT_SYNC_BSB_HEADER	BSBHeader GUID: gArmBootSyncAttReqGuid
	BOOT_SYNC_BSB_ELEMENT	Element[0] GUID: gArmBootSyncAttReport

Note: additional elements would be supplied if extra information is requires to supplement the Attestation Report

BOOT_SYNC_ATTESTATION_RESPONSE

Datastructure sent from User Context to Realm, containing the results of verifying the Attestation Report supplied in the corresponding BOOT_SYNC_ATTESTATION_REQUEST structure. This response could just be a Boolean SUCCESS / FAILURE or may contain a more detailed attestation result analysis.

Type	Variable	Notes
BOOT_SYNC_ENCRYPTED_DATA	EncDataHeader	
BOOT_SYNC_BSB_HEADER	BSBHeader	GUID: gArmBootSyncAttRespGuid
BOOT_SYNC_BSB_ELEMENT	Element[0]	GUID: gArmBootSyncAttResult

BOOT_SYNC_BSB_REQUEST

Datastructure sent from Realm to User Context to request the config data items required to continue boot.

Type	Variable	Notes
BOOT_SYNC_ENCRYPTED_DATA	EncDataHeader	
BOOT_SYNC_BSB_HEADER	BSBHeader	GUID: gArmBootSyncBsbReqGuid
BOOT_SYNC_BSB_ELEMENT	Element[0]	GUID: gArmBootSyncRequestOptions
UNIT64	Options	Identifies the data items required

BOOT_SYNC_BSB_RESPONSE

Datastructure sent from User Context to Realm in response to the BOOT_SYNC_BSB_REQUEST message and containing the requested items. (Typical element content shown as example).

Type	Variable	Notes
BOOT_SYNC_ENCRYPTED_DATA	EncDataHeader	
BOOT_SYNC_BSB_HEADER	BSBHeader	GUID: gArmBootSyncBsbRespGuid
BOOT_SYNC_BSB_ELEMENT	Element[0]	GUID: gArmBootSyncVarData
UINT8[Element[0].DataLength]	data	UEFI Variables data
BOOT_SYNC_BSB_ELEMENT	Element[1]	GUID: gArmBootSyncSecretData
UINT8[Element[1].DataLength]	data	Secrets data

BOOT_SYNC_FIN

Data structure sent from either party in the BSB protocol requesting termination of communications.

Type	Variable	Notes
		gArmBootSyncFinGuid
UINT64	Reason	Connection termination reason

Expected termination values for the Reason variable: SUCCESS - the protocol is terminating gracefully.

BOOT_SYNC_NACK

Data structure sent in response to any Request message indicating that the protocol state is incorrect. For example, if a BOOT_SYNC_BSB_REQUEST is sent by the Realm prior to trust being established, then a BOOT_SYNC_NACK would be returned.

Type	Variable	Notes
BOOT_SYNC_GUID_BLOB		gArmBootSyncNackGuid
UINT64	Reason	NACK reason

Expected termination values are:

- ECONNABORTED - the protocol is in an invalid state e.g. BOOT_SYNC_BSB_REQUEST sent before BOOT_SYNC_ATTESTATION_RESPONSE received
- ENOTCONN - the connection is unavailable

DRAFT