The Secure Remote Update Protocol

A Specification.

Andrew John Poulter (a.j.poulter@soton.ac.uk)

July 28, 2017

Contents

1 Update Messages

The SRUP Update Messages are provided to initiate and trigger a *software* update operation. The Server must begin by sending the `INITIATE` message to the Device. On receipt of an `INITIATE` message the Device must attempt to retrieve the update data from the specified URL. The Device must then send a SRUP Response message to the Server to indicate the outcome of the retrieval operation. If the status `SRUP_UPATE_SUCCESS` is received the Server may then send an `ACTIVATE` message to signal that the Device should apply the retrieved update.

1.1 Update Initiate Message

On receiving an `INITIATE` message the Device must attempt to retrieve the data from the specified URL; and compare the digest of this data with the value specified. If the operation is successful it must send a SRUP Response message (see 2) — with a status code of `0x00` (`SRUP_UPATE_SUCCESS`). If the operation fails it must send a `RESPONSE` message with a status code indicating the source of the failure.

- `0xFD` (`SRUP_UPATE_FAIL_SERVER`) should be sent if the server cannot be reached
- `0xFE` (`SRUP_UPATE_FAIL_FILE`) should be used if the server cannot supply the file specified
- `0xFF` (`SRUP_UPATE_FAIL_DIGEST`) should be used if the digest of the retrieved file does not match the value specified in the `INITIATE` message
- Optionally `0xFC` (`SRUP_UPATE_FAIL_HTTP_ERROR`) may be sent in place of `0xFD` (`SRUP_UPATE_FAIL_SERVER`) to indicate that a more detailed HTTP error code is available

If the `0xFC` (`SRUP_UPATE_FAIL_DETAILED`) status code is sent, the Device must then send two SRUP Data messages (see Section 4) to communicate the HTTP response to the Server. The first `DATA` message must have a Data ID of `HTTP_STATUS` and contain the HTTP status code returned by the web-server in the Data field. The second `DATA` message must have a Data ID of `HTTP_RESPONSE` — and the received HTTP response must be contained within the Data field.

Full details of the `INITIATE` message are shown in Table 1.1.

1.2 Update Activate Message

On receiving an `ACTIVATE` message the Device must apply application or system specific procedures to apply to update previously received and specified by the `TOKEN`. The Device optionally may then send a further `RESPONSE` message — signalling the outcome of attempting to apply the update. A `STATUS` code of `0x10` — `SRUP_ACTIVATE_SUCCESS` should be used to signal that the activation was successful; and `0x1F` — `SRUP_ACTIVATE_FAIL` should be used if the activation failed.

Full details of the `INITIATE` message are shown in Table 1.2.

Field	**Type**	**Notes**
Message Type	`uint8_t`	`0x01` — `SRUP_MESSAGE_TYPE_INITIATE`
Version	`uint8_t`	
Sequence ID	`uint64_t`	
Signature	`uint8_t*`	Variable length
Signature Length	`uint16_t`	
Token	`uint8_t*`	Variable length
Token Length	`uint16_t`	
Sender ID	`uint64_t`	
UUID	`uint64_t`	Target ID for device to be updated
URL	`char*`	Variable length
URL Length	`uint16_t`	
Digest	`uint8_t*`	Variable length
Digest Length	`uint16_t`	

Table 1.1: The SRUP UPDATE INITIATE Message

Field	**Type**	**Notes**
Message Type	`uint8_t`	`0x03` — `SRUP_MESSAGE_TYPE_ACTIVATE`
Version	`uint8_t`	
Sequence ID	`uint64_t`	
Signature	`uint8_t*`	Variable length
Signature Length	`uint16_t`	
Token	`uint8_t*`	Variable length
Token Length	`uint16_t`	
Sender ID	`uint64_t`	

Table 1.2: The SRUP UPDATE ACTIVATE Message

2 Response Message

The SRUP Response message is used to signal the outcome of a number of operations within SRUP. Permissible status values are defined for each operation type which uses the `RESPONSE` message.

Full details of the `RESPONSE` message are shown in Table 2.1.

3 Action Message

The SRUP Action message is used to permit a Server to send an arbitrary command to a Device. The Action ID field denotes the action that is being requested. These values are not defined by the protocol — but are application or system defined values. The recipient may optionally send a `RESPONSE` message to signal the outcome of the

Field	**Type**	**Notes**
Message Type	uint8_t	0x02 — SRUP_MESSAGE_TYPE_RESPONSE
Version	uint8_t	
Sequence ID	uint64_t	
Signature	uint8_t*	Variable length
Signature Length	uint16_t	
Token	uint8_t*	Variable length
Token Length	uint16_t	
Sender ID	uint64_t	
Status	uint8_t	Values: 0x00 — SRUP_UPATE_SUCCESS 0x10 — SRUP_ACTIVATE_SUCCESS 0x1F — SRUP_ACTIVATE_FAIL 0x20 — SRUP_ACTION_SUCCESS 0x2E — SRUP_ACTION_UKNOWN 0x2F — SRUP_ACTION_FAIL 0x3F — SRUP_DATA_TYPE_UKNOWN 0x40 — SRUP_GROUP_ADD_SUCCESS 0x41 — SRUP_GROUP_DEL_SUCCESS 0x4C — SRUP_GROUP_DEL_INVALID 0x4D — SRUP_GROUP_DEL_FAIL 0x4E — SRUP_GROUP_ADD_FAIL_LIMIT 0x4F — SRUP_GROUP_ADD_FAIL 0x50 — SRUP_JOIN_SUCCESS 0x5E — SRUP_JOIN_REFUSED 0x5F — SRUP_JOIN_FAIL 0x60 — SRUP_OBSERVED_JOIN_VALID 0x6E — SRUP_OBSERVED_JOIN_INVALID 0x6F — SRUP_OBSERVED_JOIN_FAIL 0x70 — SRUP_RESIGN_SUCCESS 0x7F — SRUP_RESIGN_FAIL 0x80 — SRUP_DEREGISTER_SUCCESS 0x8F — SRUP_DEREGISTER_FAIL 0xFC — SRUP_UPATE_FAIL_HTTP_ERROR 0xFD — SRUP_UPATE_FAIL_SERVER 0xFE — SRUP_UPATE_FAIL_FILE 0xFF — SRUP_UPATE_FAIL_DIGEST

Table 2.1: The SURP RESPONSE Message

action (0x20 — SRUP_ACTION_SUCCESS or 0x2F — SRUP_ACTION_FAIL). Actions requiring parametric data can be communicated to the recipient by first sending one or more Data messages containing the parameter; before sending the ACTION message. Messages with an Action ID value unknown to the recipient should be ignored. The recipient may optionally send a `RESPONSE` message with a status value of `0x2E` — `SRUP_ACTION_UKNOWN`. Full details of the `DATA` message are shown in Table 3.1.

Field	Type	Notes
Message Type	`uint8_t`	`0x04` — `SRUP_MESSAGE_TYPE_ACTION`
Version	`uint8_t`	
Sequence ID	`uint64_t`	
Signature	`uint8_t*`	Variable length
Signature Length	`uint16_t`	
Token	`uint8_t*`	Variable length
Token Length	`uint16_t`	
Sender ID	`uint64_t`	
Action ID	`uint8_t*`	Variable length
Action ID Length	`uint16_t`	

Table 3.1: The SRUP DATA Message

4 Data Message

The SRUP Data message is designed to permit Servers and Devices to exchange arbitrary data. Each message may consist of a simple or complex data type, identified by means of the Data ID field. Messages with a Data ID value unknown to the recipient should be ignored. The recipient may send a `RESPONSE` message with a status value of `0x3F` — `SRUP_DATA_TYPE_UKNOWN`.

Note that although the values that can be taken by the Data ID field are not defined within the protocol, the values `HTTP_ERROR` & `HTTP_RESPONSE` are reserved for use in conjunction with the `SRUP_UPATE_FAIL_DETAILED` status of the `SRUP_RESPONSE` message. (See Section 1.2); and `IDENTIFICATION_RESPONSE` is reserved for use in connection with the Identification Request message (See Section 5).

Full details of the `DATA` message are shown in Table 4.1.

Field	**Type**	**Notes**
Message Type	`uint8_t`	`0x05` — `SRUP_MESSAGE_TYPE_DATA`
Version	`uint8_t`	
Sequence ID	`uint64_t`	
Signature	`uint8_t*`	Variable length
Signature Length	`uint16_t`	
Token	`uint8_t*`	Variable length
Token Length	`uint16_t`	
Sender ID	`uint64_t`	
Data ID	`uint8_t*`	Variable length
Data ID Length	`uint16_t`	
Data	`uint8_t*`	Variable length
Data Length	`uint16_t`	

Table 4.1: The SRUP DATA Message

5 Identification Request Message

There is a requirement for an operator of a device to be able to query that device to identify settings or parameters in use on that device. Whilst the details of these would be application specific example might include the specific version or build of the software running on the device, specific hardware version or other details, etc. The Identify Request Message is a message type used to initiate the exchange of Identification information. On receiving the message, the recipient must reply with an `DATA` message (see Section 4) with the Data ID field containing `IDENTIFICATION_RESPONSE`; and the identification information contained within the Data field.

Full details of the `IDENTIFICATION_REQUEST` message are shown in Table 5.1.

Field	**Type**	**Notes**
Message Type	`uint8_t`	`0x06` — `SRUP_MESSAGE_TYPE_ID_REQUEST`
Version	`uint8_t`	
Sequence ID	`uint64_t`	
Signature	`uint8_t*`	Variable length
Signature Length	`uint16_t`	
Token	`uint8_t*`	Variable length
Token Length	`uint16_t`	
Sender ID	`uint64_t`	

Table 5.1: The SRUP IDENTIFY REQUEST Message

6 Group Messages

Typically Devices using SRUP will subscribe only to the MQTT topic associated with their SRUP identity; however in some scenarios a Server may wish to instigate device groups in order to facilitate a single send operation to be used to communicate with number of Devices simultaneously. In order to enable this, SRUP provides a Group Add, Group Delete, and Group Destroy messages.

In all three cases the Server should keep track of which Groups it has created, and which Devices it has requested to join or leave those groups.

6.1 Group Add Message

The Group Add Message instructs a Device to subscribe to a specified group for group messages. On receiving a `GROUP ADD` message the Device must attempt to subscribe to the MQTT topic associated with that Group Identity. The Device should then reply with an `RESPONSE` message (see Section 2). The Status field should be set to `0x40` — `SRUP_GROUP_ADD_SUCCESS` if the Device was able to subscribe to the MQTT topic specified. `0x4F` — `SRUP_GROUP_ADD_FAIL` should be sent if the subscribe operation fails. If there is a constraint on the number of topics the Device is able to subscribe to, and if therefore the Device cannot add a new subscription, then `0x4E` — `SRUP_GROUP_ADD_FAIL_LIMIT` should be sent.

Full details of the `GROUP_ADD` message are shown in Table 6.1.

Field	Type	Notes
Message Type	`uint8_t`	`0x07` — `SRUP_MESSAGE_TYPE_GROUP_ADD`
Version	`uint8_t`	
Sequence ID	`uint64_t`	
Signature	`uint8_t*`	Variable length
Signature Length	`uint16_t`	
Token	`uint8_t*`	Variable length
Token Length	`uint16_t`	
Sender ID	`uint64_t`	
UUID	`uint64_t`	Device ID for device to that is to join the specified group
Group ID	`uint8_t*`	Variable length
Group ID Length	`uint16_t`	

Table 6.1: The SRUP GROUP ADD Message

6.2 Group Delete Message

The Group Delete Message instructs a Device to unsubscribe from a specified group for group messages. On receiving a `GROUP DELETE` message the Device must attempt to unsubscribe to the MQTT topic associated with that Group Identity. The Device should then reply with an RESPONSE message (see Section 2). The Status field should be set to 0x41 — SRUP_GROUP_DEL_SUCCESS if the Device was able to unsubscribe to the MQTT topic specified. 0x4D — SRUP_GROUP_DEL_FAIL should be sent if the unsubscribe operation fails. If the Device is not a member of the specified group then the RESPONSE message should have the Status set to 0x4C — SRUP_GROUP_DEL_INVALID.

Full details of the GROUP_DELETE message are shown in Table 6.2.

Field	Type	Notes
Message Type	uint8_t	0x08 — SRUP_MESSAGE_TYPE_GROUP_DEL
Version	uint8_t	
Sequence ID	uint64_t	
Signature	uint8_t*	Variable length
Signature Length	uint16_t	
Token	uint8_t*	Variable length
Token Length	uint16_t	
Sender ID	uint64_t	
UUID	uint64_t	Device ID for device to that should leave the specified group
Group ID	uint8_t*	Variable length
Group ID Length	uint16_t	

Table 6.2: The SRUP GROUP DELETE Message

6.3 Group Destroy Message

The Group Destroy Message instructs all Devices in a specified Group to unsubscribe it. On receiving a GROUP DESTROY message all Devices must attempt to unsubscribe to the MQTT topic associated with that Group Identity. The Devices should then reply with an RESPONSE message (see Section 2). The Status field should be set to 0x41 — SRUP_GROUP_DEL_SUCCESS if the Device was able to unsubscribe to the MQTT topic specified. 0x4D — SRUP_GROUP_DEL_FAIL should be sent if the unsubscribe operation fails. If the Device is not a member of the specified group then the RESPONSE message should have the Status set to 0x4C — SRUP_GROUP_DEL_INVALID.

The structure of the GROUP_DESTROY is identical to the GROUP_DELETE message — except that no Device UUID is specified.

Full details of the GROUP_DESTROY message are shown in Table 6.3.

7 Join Messages

The Join Message family is used to negotiate Devices joining the control of (and thus becoming subordinate to) a specified C2 Server. A number of different message types

Field	Type	Notes
Message Type	`uint8_t`	`0x08` — `SRUP_MESSAGE_TYPE_GROUP_DEL`
Version	`uint8_t`	
Sequence ID	`uint64_t`	
Signature	`uint8_t*`	Variable length
Signature Length	`uint16_t`	
Token	`uint8_t*`	Variable length
Token Length	`uint16_t`	
Sender ID	`uint64_t`	
Group ID	`uint8_t*`	Variable length
Group ID Length	`uint16_t`	

Table 6.3: The SRUP GROUP DESTROY Message

are specified in order to provide unmediated Joining; and both human-mediated, & machine-mediated Joining.

7.1 Unmediated Join Messages

7.1.1 Join Request

A Device may send a Join Request message to a given Server requesting to subordinate itself to that Server. Note that unlike other SRUP messages sent from a Device this message must be sent on the MQTT Topic associated with the Server's identity (since, by definition, the Server is not yet subscribed to the Device's topic).

On receiving a Join Request message the Server must either accepting the Join — or refuse it.

If accepting it must attempt to subscribe to a topic associated with the Sender ID of the `JOIN REQUEST` message, and send a `RESPONSE` message with the Status set to `0x50` — `SRUP_JOIN_SUCCESS` if successful; or a `0x5F` — `SRUP_JOIN_FAIL` if the subscribe fails.

If refusing the request the Server must send a `RESPONSE` message with the Status set to `0x5E` — `SRUP_JOIN_REFUSED`.

Full details of the `UNMEDIATED JOIN REQUEST` message are shown in Table 7.1.

7.1.2 Join Command

A Server may send a Join Command to a Device: such as in the scenario where the user initiates the Join process by providing the Server with the identity of the Device directly via the Server's user-interface.

On receiving a Join Command message the Device must signal acceptance of this operation by sending a `RESPONSE` message with the Status set to `0x41` — `SRUP_GROUP_DEL_SUCCESS`. Since the Server controls to which Devices it sends through the use of the MQTT topic no further action is required by the Device. Optionally the Device may

Field	Type	Notes
Message Type	`uint8_t`	`0x09` — `SRUP_MESSAGE_TYPE_JOIN_REQ`
Version	`uint8_t`	
Sequence ID	`uint64_t`	
Signature	`uint8_t*`	Variable length
Signature Length	`uint16_t`	
Token	`uint8_t*`	Variable length
Token Length	`uint16_t`	
Sender ID	`uint64_t`	

Table 7.1: The SRUP UNMEDIATED JOIN REQUEST Message

decline the command (subject to system specific implementation) — which may be signalled via a `RESPONSE` message with the Status set to `0x5E` — `SRUP_JOIN_REFUSED`. Note that the Device must respond. The Server must be prepared not to receive a response (such as may be the case if the Device in question is off-line): in which case it should assume that the Join has failed.

Full details of the `UNMEDIATED JOIN COMMAND` message are shown in Table 7.2.

Field	Type	Notes
Message Type	`uint8_t`	`0x0A` — `SRUP_MESSAGE_TYPE_JOIN_COMMAND`
Version	`uint8_t`	
Sequence ID	`uint64_t`	
Signature	`uint8_t*`	Variable length
Signature Length	`uint16_t`	
Token	`uint8_t*`	Variable length
Token Length	`uint16_t`	
Sender ID	`uint64_t`	
UUID	`uint64_t`	Device ID for device to that is being instructed to become subordinate to the C2 Server

Table 7.2: The SRUP UNMEDIATED JOIN COMMAND Message

7.2 Human-Mediated Join Messages

7.2.1 Human-Mediated Join Request Message

If the Server requires additional confirmation of the identity of the Device before accepting the Join, then a human-mediated Join operation can be used. In this scenario the Device initiates the process by sending a Human-Mediated Join Request Message to the Server. (Note as described in Section 7.1.1 the Device must send this message using the MQTT topic corresponding to the Server's identity). On receiving this message the Server must then send a Mediated Join Response message to the Device. (See Section 7.2.2).

Full details of the `HUMAN-MEDIATED JOIN REQUEST` message are shown in Table 7.3.

Field	**Type**	**Notes**
Message Type	`uint8_t`	`0x0B` — `SRUP_MESSAGE_TYPE_HM_JOIN_REQ`
Version	`uint8_t`	
Sequence ID	`uint64_t`	
Signature	`uint8_t*`	Variable length
Signature Length	`uint16_t`	
Token	`uint8_t*`	Variable length
Token Length	`uint16_t`	
Sender ID	`uint64_t`	

Table 7.3: The SRUP HUMAN-MEDIATED JOIN REQUEST Message

7.2.2 Human-Mediated Join Response Message

The Human-Mediated Join Response Message consists of an encrypted random *confirmation* value sent to the Device (and separately by the C2 system to the user — e.g. over a HTTPS web connection) which can be used by the Device to prove to the human observer that the physical Device presenting the information corresponds to the Device that's requesting the Join operation within SRUP.

In order to practically implement this, the Server must generate a 128-bit random confirmation value, which is then encrypted using the intended recipient's public key (using RSA) and then this value is then sent within the Human-Mediated Join Response Message. The mechanism for the human observer to check the confirmation value, and then accept / reject is not specified within SRUP.

Full details of the `HUMAN-MEDIATED JOIN RESPONSE` message are shown in Table 7.4.

7.3 Machine-Mediated Join Messages

In scenarios where no human-observer may be present a machine-mediated version of the Join process is provided. This process requires a trusted third-party (already subordinate to the C2 Server) to take the place of the human observer — with the observation taking-place over a short-range point-to-point communication channel outside of SRUP. The protocol does not specify the mechanism for this observation to take place.

7.3.1 Observed Join Request Message

The Device must first identify details of the C2 Server and the Observer outside of the protocol. Once this has been done, the Device must send a Observed Join Request

Field	Type	Notes
Message Type	`uint8_t`	`0x0C` — `SRUP_MESSAGE_TYPE_HM_JOIN_RESP`
Version	`uint8_t`	
Sequence ID	`uint64_t`	
Signature	`uint8_t*`	Variable length
Signature Length	`uint16_t`	
Token	`uint8_t*`	Variable length
Token Length	`uint16_t`	
Sender ID	`uint64_t`	
Encrypted Conf.	`uint8_t*`	Length will be RSA key-length / 8 e.g. 256 bytes for a 2048-bit key. Unencrypted confirmation value is 128-bits implemented as `uint8_t[8]`
Conf. Length	`uint16_t`	

Table 7.4: The SRUP HUMAN-MEDIATED JOIN RESPONSE Message

Message to the Server. (Note as described in Section 7.1.1 the Device must send this message using the MQTT topic corresponding to the Server's identity).

The Observed Join Request Message contains the identity of the Observer node in the Observer ID field.

On receiving a valid Observed Join Request message the Server must then send an Observed Join Response message back to the Device, and a Observation Request message to the Observer Node. (See Sections 7.3.2 & 7.3.3).

Full details of the `OBSERVED JOIN REQUEST` message are shown in Table 7.5.

Field	Type	Notes
Message Type	`uint8_t`	`0x0D` — `SRUP_MESSAGE_TYPE_OBS_JOIN_REQ`
Version	`uint8_t`	
Sequence ID	`uint64_t`	
Signature	`uint8_t*`	Variable length
Signature Length	`uint16_t`	
Token	`uint8_t*`	Variable length
Token Length	`uint16_t`	
Sender ID	`uint64_t`	
Observer ID	`uint64_t`	

Table 7.5: The SRUP OBSERVED JOIN REQUEST Message

7.3.2 Observed Join Response Message

The Observed Join Response message is identical in content to the Human-Mediated Join Response Message (Section 7.2.1); however it is included as a discrete message type to simplify the implementation of systems in which both human- and machine-mediated join operations may occur.

Upon receiving a Observed Join Response Message a Device must transmit the unencrypted Confirmation value to the observer externally to the protocol.

Full details of the `OBSERVED JOIN RESPONSE` message are shown in Table 7.6.

Field	Type	Notes
Message Type	`uint8_t`	`0x0E` — `SRUP_MESSAGE_TYPE_OBS_JOIN_RESP`
Version	`uint8_t`	
Sequence ID	`uint64_t`	
Signature	`uint8_t*`	Variable length
Signature Length	`uint16_t`	
Token	`uint8_t*`	Variable length
Token Length	`uint16_t`	
Sender ID	`uint64_t`	
Encrypted Conf.	`uint8_t*`	Length will be RSA key-length / 8 e.g. 256 bytes for a 2048-bit key. Unencrypted confirmation value is 128-bits implemented as `uint8_t[8]`
Conf. Length	`uint16_t`	

Table 7.6: The SRUP OBSERVED JOIN RESPONSE Message

7.3.3 Observation Request Message

The Observation Request Message is used to communicate the Confirmation value to the Observer node. The format is similar to the `OBSERVED JOIN RESPONSE` Message (Section 7.3.2) — though the Confirmation value is encrypted using the Observation Node's public key.

On receiving a `OBSERVATION REQUEST` message the Observer Node must prepare to receive the Confirmation value from the Device requesting the Join, externally to the protocol; and then to compare the received value to the value contained within the `OBSERVATION REQUEST` message. The Observer must then send a `RESPONSE` message back to the Server to signal the outcome of the comparison. This `RESPONSE` message should have a Status of `0x60` — `SRUP_OBSERVED_JOIN_VALID` if the two values match; or a Status of `0x6E` — `SRUP_OBSERVED_JOIN_INVALID` if they do not. A Status of `0x6F` — `SRUP_OBSERVED_JOIN_FAIL` may be used if the operation fails (such as no data is received by the Observer).

Full details of the `OBSERVATION REQUEST` message are shown in Table 7.6.

Field	Type	Notes
Message Type	`uint8_t`	`0x0F` — `SRUP_MESSAGE_TYPE_OBSERVE_REQ`
Version	`uint8_t`	
Sequence ID	`uint64_t`	
Signature	`uint8_t*`	Variable length
Signature Length	`uint16_t`	
Token	`uint8_t*`	Variable length
Token Length	`uint16_t`	
Sender ID	`uint64_t`	
Encrypted Conf.	`uint8_t*`	Length will be RSA key-length / 8 e.g. 256 bytes for a 2048-bit key. Unencrypted confirmation value is 128-bits implemented as `uint8_t[8]`
Conf. Length	`uint16_t`	

Table 7.7: The SRUP OBSERVATION REQUEST Message

8 Resignation & Termination Messages

In order to remove a Device from the control of a Server, SRUP provides Resignation & Termination message types.

8.1 Resign Request

If a Device wishes to resign from the control of Server is may send a Resign Request Message to the Server. Upon receiving a `RESIGN REQUEST` message the Server may either accept or reject the request; and then it must send a `RESPONSE` message — with a Status of `0x70` — `SRUP_RESIGN_SUCCESS`, or `0x7F` — `SRUP_RESIGN_FAIL` if the resignation request is not accepted.

Full details of the `RESIGN REQUEST` message are shown in Table 8.1.

Field	Type	Notes
Message Type	`uint8_t`	`0x10` — `SRUP_MESSAGE_TYPE_RESIGN_REQ`
Version	`uint8_t`	
Sequence ID	`uint64_t`	
Signature	`uint8_t*`	Variable length
Signature Length	`uint16_t`	
Token	`uint8_t*`	Variable length
Token Length	`uint16_t`	
Sender ID	`uint64_t`	

Table 8.1: The RESIGN REQUEST Message

8.2 Termination Command

A Server may send a Termination Command message to any subordinate Device to instruct it that it is no-longer subject to control by the Server. On receiving a Termination Command message the Device may optionally send a RESPONSE message — with a Status of 0x70 — SRUP_RESIGN_SUCCESS to the Server (via the MQTT topic associated with the Server's identity. The Server must not send any further messages to the Device without another Join operation (see Section 7) first taking place.

Full details of the TERMINATION COMMAND message are shown in Table 8.2.

Field	Type	Notes
Message Type	uint8_t	0x11 — SRUP_MESSAGE_TYPE_TERMINATE_COM
Version	uint8_t	
Sequence ID	uint64_t	
Signature	uint8_t*	Variable length
Signature Length	uint16_t	
Token	uint8_t*	Variable length
Token Length	uint16_t	
Sender ID	uint64_t	

Table 8.2: The TERMINATION COMMAND Message

9 Deregistration Messages

Devices may be permanently removed from the system via a Deregistration message. Deregistration causes the the Device's public key to be deleted from the System. Devices should also remove the Server public keys that they hold, upon notification of deregistration.

9.1 Deregister Request

A Device wishing to be permanently deregistered from a given system my send a Deregister Request message to any Server to which it is subordinate. The Server must then send a final RESPONSE message to the Device with a status of 0x80 — SRUP_DEREGISTER_SUCCESS; or in the event that the Deregistration cannot be processed, a status of 0x8F — SRUP_DEREGISTER_FAIL may be sent. This should be used exceptionally however as Deregistration requests should always be honoured unless they cannot be processed. The Device must participate in Registration (see Section 10) and Join (Section 7) operations before it is able to receive further SRUP messages.

Full details of the DEREGISTER REQUEST message are shown in Table 8.2.

Field	Type	Notes
Message Type	`uint8_t`	`0x12` — `SRUP_MESSAGE_TYPE_DEREGISTER_REG`
Version	`uint8_t`	
Sequence ID	`uint64_t`	
Signature	`uint8_t*`	Variable length
Signature Length	`uint16_t`	
Token	`uint8_t*`	Variable length
Token Length	`uint16_t`	
Sender ID	`uint64_t`	

Table 9.1: The DEREGISTER REQUEST Message

9.2 Deregister Command

A Server may send a Deregister Command message to any subordinate Device to instruct it that it is no-longer Registered within the system and that it's access have been revoked. A Device receiving a Deregister Command should not attempt to send any further SRUP messages — and it should disconnect from the MQTT Broker being used for SRUP messages. Full details of the `DEREGISTER COMMAND` message are shown in Table 9.2.

Field	Type	Notes
Message Type	`uint8_t`	`0x13` — `SRUP_MESSAGE_TYPE_DEREGISTER_COM`
Version	`uint8_t`	
Sequence ID	`uint64_t`	
Signature	`uint8_t*`	Variable length
Signature Length	`uint16_t`	
Token	`uint8_t*`	Variable length
Token Length	`uint16_t`	
Sender ID	`uint64_t`	

Table 9.2: The DEREGISTER COMMAND Message

10 Registration

The process of a Device performing initial Registration to facilitate registration of the Device's public key is performed entirely outside of the SRUP protocol. An explanation of the process is included for completeness — and an outline of a reference implementation is described.

10.1 Registration Requirements

SRUP requires that key exchange has taken place before the first Join operation takes place. It is therefore assumed that the Device will already have a copy of the public key for any Server that it wishes to communicate with (or for any which would wish to communicate with it); and that all Server's have a copy of the public key for any Devices that they will communicate.

10.2 Example Reference Registration scheme

It is expected that implementations will use an HTTPS web API running over TLS. Using this approach the Device would send its Identity and public key to the web-server via a HTTPS `POST` request (together with any additional system implementation specific information required); and receive a response from the web-service containing the address of the MQTT broker & the Server public key (or a HTTPS URL from which the key or keys may be retrieved — e.g. by using a `GET` request against an end-point corresponding to the Server ID; though again this is on an implementation-specific basis).

Once registration is complete — the C2 server(s) need to receive a copy of the keys. Similarly a mechanism should be provided for a Server (on receiving a Deregister Request (See Section 9.1) to propagate the removal of the key to all other Servers.