Secure Communication Peer

1 Description

The Secure Communication Peer (SCP) is a collection of unknown nodes that exchange public keys upon request and perform basic services such as determining whether numbers are even or odd and storing small amounts of data.

The nodes are all contained within the Nucleo-F303RE, and they all communicate over the same unsecured UART channel. While the "services" offered are clearly of negligible value, this device provides an opportunity to use asymmetric cryptography to establish shared secrets and communicate securely with other nodes in a simulated network.

2 Communication

The UART interface provided by the Secure Communication Peer consists of a wired UART port for connection to a second device. The wired UART port has a baud rate of 115200 Hz. The SCP sends and receives data in messages as described in Section 3.

The UART interface is considered an unsecured channel. Assume that public keys and status messages are not falsified or tampered with during transit. This is a poor assumption for a general unsecured channel and is used here to limit the scope of the laboratory. Do not assume that any other messages are authentic.

The SCP sends and receives numbers (serial numbers, data lengths, etc.) in little-endian format. For example, the number 0x1234 is transmitted to the SCP as two bytes: first 0x34, then 0x12. This does not imply that general data (such as keys, nonces, auth tags, etc.) should be "reversed" when transmitting to the SCP.

3 Message Format

Each message sent or received by the Secure Communication Peer must conform to one of the formats listed in Table 1.

Table 1: Message Formats

Number	Name	Description	SCP Direction
1	Status Message	Status and error codes from SCP. See Table 5	TX
2	Public Key	Curve 25519 public key. See Table 6	TX, RX
3	Simple Message	Single-recipient message. See Table 7	TX, RX
4	Message	Multi-recipient message. See Table 8	TX, RX

3.1 Request Types

Messages and Simple Messages are general formats which can be used for multiple purposes by specifying one of the Request Types described in Table 2. The Request Type determines the expected contents of the Data field, as well as the behavior of the node(s) that receives the request. If a Request Type requires a response from the receiving node, the response will match the Message Format and Request Type of the request.

The Even or Odd request type is used to query a node asking whether a number is even or odd. The number is placed in the Data field. The response to this request also has type Even or Odd and consists of a single byte in the Data field, which is 1 if the number was odd and 0 if the number was even.

The Storage Write request type writes data to each specified recipient. The data is identified by a 2 byte non-zero storage ID, which should be chosen to be unique. The Data field consists of the 2 byte storage ID immediately followed by the data to be stored. The SCP can store a total of 10 sets of data, each with a unique storage ID.

The Storage Read request type requests that one of the specified recipients return a copy of data which was previously stored. The Data field consists of the storage ID corresponding to the desired data. One SCP node which previously received the data and which received the Storage Read request will respond. The response Data field consists of the storage ID followed by the requested data.

Table 2: Request Types

Number	Name	Description	Supported Message Formats
1	Even or Odd	Check if a value is even or odd	Simple Message (3), Message (4)
2	Storage Write	Write data to storage	Message (4)
3	Storage Read	Read data from storage	Message (4)

3.2 Status Messages

The SCP sends status messages to provide limited information about its internal state and aid in debugging when an error occurs. The possible statuses are described in Table 3 and the possible error codes are described in Table 4.

Table 3: SCP Statuses

Number	Name	Description
1	Not Ready	SCP is not ready to receive messages
2	Waiting for Pub Key	SCP is waiting for a Public Key Message
3	Waiting for Message	SCP has received a public key and is waiting for another message

3.3 Public Key Exchange

The SCP receives public keys in the format shown in Table 6. The number of recipients specified in the public key message determines how many nodes within the SCP will respond. Each node will respond using the required format. Responses from nodes will not overlap (collide) and there will be at least 5 ms separating the end of one transmission from the beginning of the next. SCP nodes support ECDH using Curve 25519.

The salt should be randomly generated; SCP nodes will also provide randomly generated salts in their response. The shared encryption key, which is used for all future messages between the two parties is derived from the ECDH pre-master secret using HKDF with SHA-256. The pre-master secret is the initial keying material, the output keying material has a length of 256 bits, and the salt (16 bytes) is computed as the XOR of the two salts provided in public key messages, one from the SCP node and one from the node initiating contact.

Table 4: SCP Error Codes

Number	Name	Description
1	Success	No error occurred
2	Bad Packet	Received packet length, but did not receive full packet
3	Bad Packet Length	Packet length is too large or too small
4	Invalid Format	Invalid message format
5	Shared Key Fail	Failed to derive shared key from public key message
6	Decryption Fail	Failed to decrypt and verify authentication tag
7	No Key	SCP doesn't have a key for the transmitting device
8	Invalid Data Length	Data Length is too large, too small, or not consistent with Message Length
9	Storage Full	All 10 storage slots are full
10	Data Not Found	Requested storage ID was not found
11	Invalid Recipient Count	Too many or too few recipients
12	Invalid Recipient	One or more recipients are not available/do not exist
13	Other Error	An internal error occurred and the SCP is resetting

Table 5: Status Message Format

Field Name	Length (bytes)	Description
Message Header	6	See Table 9
Status	1	Status of the SCP. See Table 3
Error	1	Error code for the most recent operation. See Table 4

Table 6: Public Key Message Format

Field Name	Length (bytes)	Description
Message Header	6	See Table 9
Public Key	32	Curve 25519 public key
Salt	16	Randomly generated salt for use in HKDF

3.4 Simple Messages

Simple Messages are encrypted with the shared symmetric encryption key as described in Section 3.3, using the format shown in Table 7. Simple messages have one recipient. The contents of the Data field vary depending on the Request Type—refer to Section 3.1.

The Data Length field is the length in bytes of the Data field. This is provided for convenience even though the Message Length field and knowledge of the required message structure is sufficient to determine the value of the Data Length field.

3.5 Messages

Messages support a varying number of recipients (up to 10). The contents of the message (the Request Type field through the end of the Data field) are encrypted with a random 256 bit key chosen by the sender. This key is encrypted with the shared symmetric encryption key(s) established via public key exchange, and an encrypted copy is included for each recipient. AES-GCM is used for encrypted the contents of the message

Table 7: Simple Message Format

Field Name	Length (bytes)	Description
Message Header	6	See Table 9
Encryption Header	28	See Table 10
Request Type	1	See Table 2
Data Length	2	Length in bytes of the `Data` field
Data	up to 256	Use depends on `Request Type`.

and for encrypting each copy of the message key. Note that the Encryption Header field immediately preceding the Request Type field contains the the nonce and authentication tag for the contents of the message (encrypted with the message key). The Encryption Header field inside each Encrypted Key field contains the nonce and authentication tag for that encrypted copy of the message key.

Table 8: Message Format

Field Name	Length (bytes)	Description
Message Header	6	See Table 9
Encrypted Keys	62 * No. Recipients	Encrypted copies of the key, 1 per recipient. See Table 11
Encryption Header	28	See Table 10
Request Type	1	See Table 2
Data Length	2	Length in bytes of the `Data` field
Data	up to 1024	Use depends on `Request Type`.

3.6 Common Message Components

Table 9: Message Header Format

Field Name	Length (bytes)	Description
Message Length	2	Length of the message, not including this field
Message Format	1	Format of the message. See Table 1
No. Recipients	1	Number of intended recipients
Sender Serial No.	2	Serial number of the node sending the message

Table 10: Encryption Header Format

Field Name	Length (bytes)	Description
Nonce	12	AES-GCM nonce
Auth Tag	16	AES-GCM authentication tag

Table 11: Encrypted Key Format

Field Name	Length (bytes)	Description
Recipient Serial No.	2	Serial number of the intended recipient
Encryption Header	28	See Table 10
Key	32	Encrypted AES-GCM key

4 Pinout

UART Signal Connections

Arduino Pin	Aux. Header Pin	Signal	Description
D6	CN10.25	TX	UART Transmit Line
N/A	CN10.18	RX	UART Receive Line

5 Firmware Installation

In order to make a Secure Communication Peer, a Nucleo-F303RE development board is needed. Firmware is provided on the Microprocessor Systems website.

To flash the firmware on the device on Windows:

1. Download and install the STSW-LINK004: ST-LINK Utility.
2. Run the STM32 ST-LINK Utility and connect to the Nucleo-F303RE board. This should be done without the DISCO board connected to avoid connection to the wrong target.
3. Flash the firmware by going to **Target →Program...**, selecting the firmware (.bin file) and clicking **Open**.
4. The default values on the next screen should be OK, click **Start** to complete the process.

To flash the firmware on the device on Linux:

1. Install the stlink-tools package. For Ubuntu based systems, run the command:
   ```
   $ sudo apt install stlink-tools
   ```
2. Flash the firmware through the command:
   ```
   $ st-flash write LAB-03_STaTS_Firmware.bin 0x8000000
   ```