SCC.439 Network and Systems Security

A Secure Messaging Protocol Proposal

Ben Goldsworthy, 32098584
b.goldsworthy@lancaster.ac.uk

I. Introduction

Communication protocols serve to enable the interactions between devices that form the very basis of the Internet. Over time, the desires for properties such as authenticiation, integrity and confidentiality, amongst others, has led to the development of new, more secure protocols. This paper details one such protocol.

The key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL', 'SHALL NOT', 'SHOULD', 'SHOULD NOT', 'RECOMMENDED', 'MAY', and 'OPTIONAL' in this document are to be interpreted as described in RFC 2119 <https://tools.ietf.org/html/rfc2119>

II. Specification

The specification explicitly states that this protocol:

- should allow for mutual authentication between client and server;
- should ensure the protection of message integrity and authenticity via a message authentication code (MAC);
- should ensure confidentiality of transmitted data via encryption; and
- should utilise individual session keys; and
- should refresh the session keys after the sending and receipt of 10 data responses from the server.

No mention is made of any performance requirements, or of the need for non-repudiation, so these are assumed to be outside of scope, as is any mechanism for the downgrading of cryptographic algorithms in order to enable wider support for the protocol. No explicit mention is made of the need for perfect forward secrecy, but this has been inferred as a requirement.
The specification also makes the assumption that both the server and the client have an existing shared secret, hereafter referred to as s_0.
The speciftication states that only 'all data in $DATA\_RESPONSE$ messages must be [encrypted]', but does not preclude further encryption.
No mention is made of resumable or continuable sessions, so session identifiers are considered out-of-scope.
Crucially, as no description is given as to the nature of the data being exchanged, this has been inferred from the rest of the specification. As no mention is made of the client needing to specify the data they are requesting, it has been assumed that server returns n bytes of some data d to each and every data request, and that the protocol specifcation does not intend to handle situations in which the client requires *specific* data from the server, only *immediate* data. An example situation in which this might be the case include that of a sensor that returns the last m recordings when prompted. The impact of this assumption is highlighted in part IV(A).
Finally, no mention is made of the need for user to be able to choose specific values, such as the number of messages that can be sent before a key re-negotiation is triggered.

III. The Protocol

A. Protocol Requirements

The proposed protocol consists of 10 message types: $HELLO$; $HELLO\_ACK$; ACK; $DATA\_REQUEST$; $DATA\_RESPONSE$; $DATA\_ACK$; $RESEND\_LAST$; $CLOSE$; $CLOSE\_ACK$; and $CLOSED\_WITHOUT\_ACK$.

The initial $HELLO$ and $HELLO\_ACK$ messages MUST be signed using personal RSA public keys, which MUST be authenticated with a CA certificate, via Public Key Infrastructure (PKI). X.509 certificates MUST NOT be used, as these can be forged.

Once mutual authentication is achieved, Elliptic Curve Diffie-Hellman Ephemeral (ECDHE) SHALL be used by each party to determine a shared session key, which MUST then be used to encrypt further messages via AES-GCM. This key generation SHOULD use the Curve25519 elliptic curve, as evidence exists that many other commonly-used curves may have been intentionally weakened by US intelligence agencies.

New keys MUST be re-negotiated after every 10 $DATA\_RESPONSE$ messages have been sent, and their receipt has been confirmed by matched $DATA\_ACK$ messages. This key re-negotation SHALL be initiated by the Server.

Message sequence numbers MUST be randomly generated for the initial $HELLO$ message, and thereafter incremented by nonces sent in preceding messages.

MACs MUST be generated for every message. This process SHOULD use HMAC-SHA512, MUST be encrypt-then-MAC and SHALL send the MAC along with the message. The hashed initial shared secret SHA-512(s_0) SHOULD be used as the key for the initial $HELLO$ and $HELLO\_ACK$ messages, and the subsequent shared secrets $s_1 \dots s_n$ (also hashed) after each session key re-negotiation.

Messages received more than 25 seconds after their send time MUST be discarded, and the connection SHOULD be closed. Alternatively, in the interests of usability (albeit at the expense of security), the receiver MAY choose to send a $RESEND\_LAST$ in order to prompt the retransmission of the offending message. If three $RESEND_LAST$s have not produced a timely enough response, the connection MUST be closed.

Either party SHOULD close the connection upon receiving any messages that are corrupted (e.g. wrong `msg-type` for the packet structure, incorrect `sequence-number`, etc.). Alternatively, the receiver MAY choose to send a $RESEND\_LAST$ in order to prompt the retransmission of the offending message. If three $RESEND_LAST$s have not produced a valid message, the connection MUST be closed.

If no expected message has been received after 45 seconds from the previous message send, the connection SHOULD be closed. Alternatively, the receiver MAY choose to send a $RESEND\_LAST$ in order to prompt the retransmission of any lost message. If three $RESEND_LAST$s have not produced a message, the connection MUST be closed.

If no $CLOSE\_ACK$ has followed a $CLOSE$ message after 45 seconds, a $CLOSED\_WITHOUT\_ACK$ message MAY be sent and the connection closed on the sender's end. Alternatively, another $CLOSE$ may be sent. If three $CLOSE$s have not produced a $CLOSE\_ACK$ message, a $CLOSED\_WITHOUT\_ACK$ message MUST be sent and the connection closed on the sender's end.

B. Message Formats

All messages are identified by the value in their `msg-type` field (e.g. `"hello"` for a $HELLO$ message). This means that, if messages were padded to the produce equal-length packets, it should be impossible for an eavesdropper to identify what message each packet contains (short of traffic and timing analysis)—this is considered to be out of scope for the current protocol, however.

1) $HELLO$, $HELLO\_ACK$ & $DATA\_RESPONSE$: For the initial $HELLO$ message, C includes a random integer n in the `sequence-number` field. S includes this in their $HELLO\_ACK$ in order to pair the messages together. For the following key-renegotation $HELLO$s, the sequence number field simply contains

the applicable sequence number.
The `ttl` field contains the Unix time value for the time the message was sent, plus 25 seconds. If the recipient finds `ttl` to be higher than the current system time, the connection will be closed.
In $HELLO$ and $HELLO\_ACK$ messages, the `data` field contains the sender's Curve25519 public key. In a $DATA\_RESPONSE$ message, it contains the data requested by C in their preceding $DATA\_REQUEST$ message.

```
{ "mac",
  "payload" {
    "msg-type",
    "data",
    "sequence-number"
    "ttl",
  }
}
```

2) ACK and $DATA\_ACK$: The `nonce` field contains an randomly-generated integer, which both C and S add to the current `sequence_number` value to produce the next sequence number.

```
{ "mac",
  "payload" {
    "msg-type",
    "sequence-number",
    "ttl",
    "nonce"
  }
}
```

3) $DATA\_REQUEST$, $RESEND\_LAST$, $CLOSE$ & $CLOSE\_ACK$: As other messages. There is no need for a `nonce` field in a $CLOSE\_ACK$ message, as in other $*ACK$s, as there should be no future sequence numbers for this connection.

```
{ "mac",
  "payload" {
    "msg-type",
    "sequence-number"
    "ttl",
  }
}
```

4) $CLOSED\_WITHOUT\_ACK$: As other messages, except `ttl` is ignored as the message being sent implies delays on the transmission medium, and that the sender has already closed the connection on their end.

```
{ "mac",
  "payload" {
    "msg-type",
    "sequence-number"
  }
}
```

C. Protocol Description

The protocol is used as follows, for communication between the Client C and the Server S. First, C and S both generate 32-bit Curve25519 private and public keys. Then, they share these public keys in $HELLO$ and $HELLO\_ACK$ messages, each signed using their personal RSA (*not* Curve25519) public keys.

$$C \rightarrow S : \{HELLO\}$$

$$S \rightarrow C : \{HELLO\_ACK\}$$

At this point, S and C can each use the `curve25519()` function with their own Curve25519 private key and their partner's Curve25519 public key, producing another shared secret s_1. Each performs SHA-512 (s_1) to produces a shared symmetric key K_{C,S_0}, which is used for the initial temporary session key.

$$C \rightarrow S : \{ACK\}_{K_{C,S_0}}$$
$$C \rightarrow S : \{DATA\_REQUEST\}_{K_{C,S_0}}$$
$$S \rightarrow C : \{DATA\_RESPONSE\}_{K_{C,S_0}}$$
$$C \rightarrow S : \{DATA\_ACK\}_{K_{C,S_0}}$$
$$\vdots$$
$$C \rightarrow S : \{DATA\_ACK\}_{K_{C,S_0}}$$

After 10 $DATA\_RESPONSE$ messages have been sent (and receipt has be verified by the return of a $DATA\_ACK$ message), S generates a new Curve25519 public-private keypair and the three-way handshake is repeated, this time instigated by S. After C has generated their own Curve25519 public-private keypair, both can generates a new shared symmetric key K_{C,S_1} (and so on).

$$S \rightarrow C : \{HELLO\}_{K_{C,S_0}}$$
$$C \rightarrow S : \{HELLO\_ACK\}_{K_{C,S_0}}$$
$$S \rightarrow C : \{ACK\}_{K_{C,S_1}}$$
$$C \rightarrow S : \{DATA\_REQUEST\}_{K_{C,S_1}}$$
$$\vdots$$

At a certain point, C or S (here S) attempts to close the connection mutually.

$$S \rightarrow C : \{CLOSE\}_{K_{C,S_n}}$$
$$C \rightarrow S : \{CLOSE\_ACK\}_{K_{C,S_n}}$$

However, if no $CLOSE\_ACK$ is forthcoming (say S has frozen temporarily), the sender of $CLOSE$ can send a $CLOSED\_WITHOUT\_ACK$ message to close without waiting for confirmation. This can be riskier, for example leaving C's connection open on S.

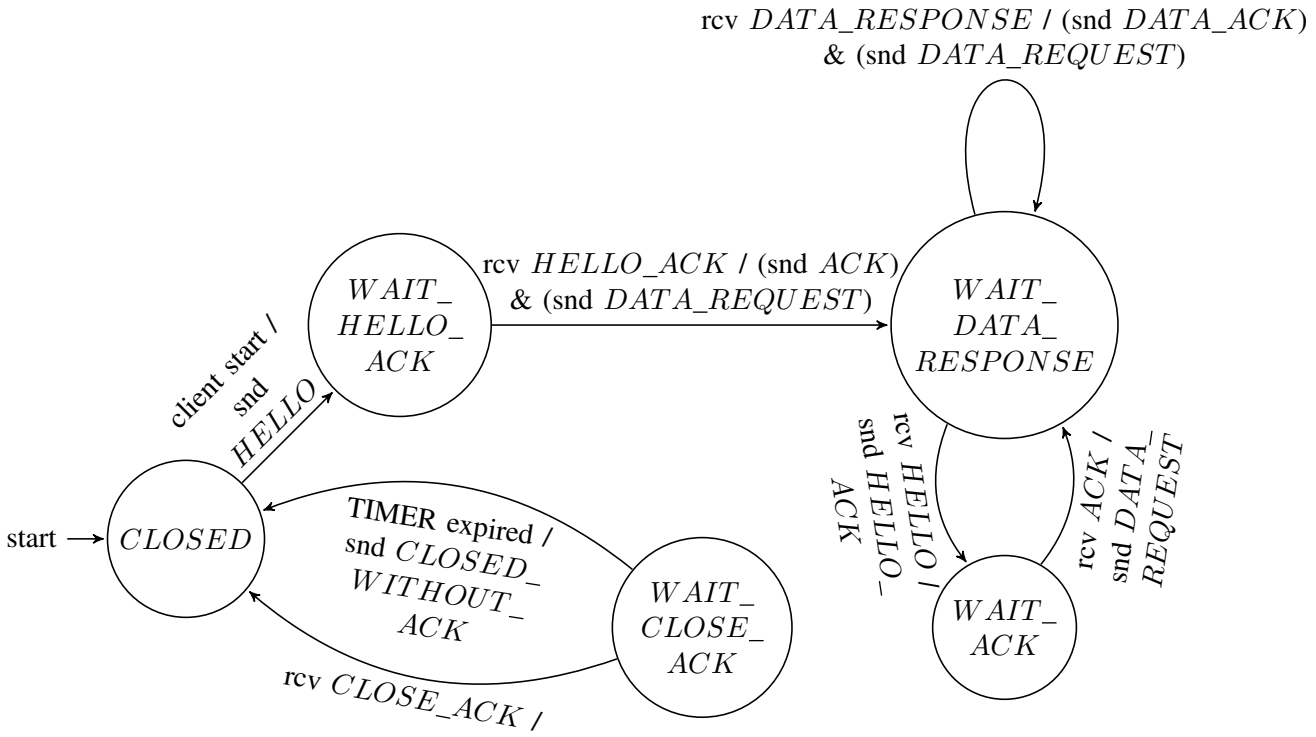
$$C \rightarrow S : \{CLOSED\_WITHOUT\_ACK\}_{K_{C,S_n}}$$

D. State Machines

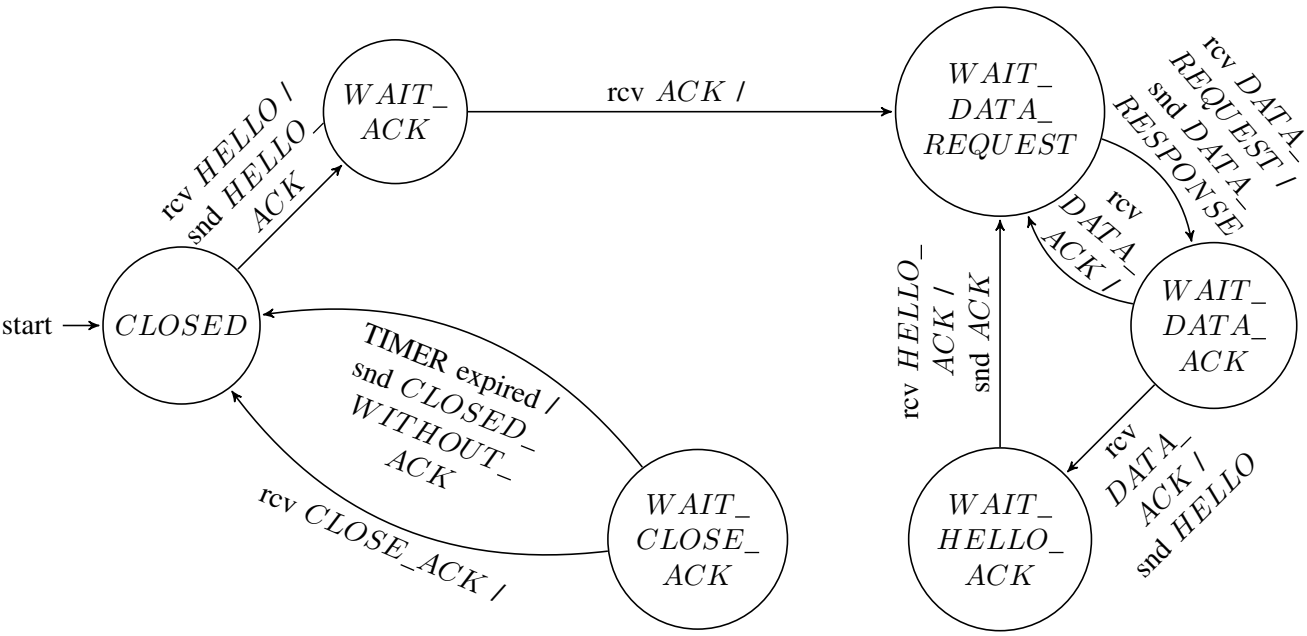
These state machines represent the ideal operation of the protocol. Requirements previously stated as 'SHOULD's and 'SHOULD NOT's are here treated as 'MUST's and 'MUST NOT's. 'MAY's are ignored.

1) Client: In all states:

- 'rcv $CLOSE$' will lead to state $CLOSED$ with the action 'snd $CLOSE\_ACK$';
- receiving any message (except $CLOSE\_ACK$) will also perform the 'set TIMER' action; and
- the 'TIMER expired' or 'MESSAGE invalid' actions occuring at any state (except $WAIT\_CLOSE\_ACK$) will lead to state $WAIT\_CLOSE\_ACK$ with the actions 'snd $CLOSE$ & set TIMER'.



2) Server: As Client.



IV. Protocol Analyis

A. Security

Confidentiality is ensured by encrypting all messages (after the initial $HELLO$ and $HELLO\_ACK$, which must be sent in the clear) using ephemeral session keys. Their ephemarality also ensures perfect forward secrecy, as any attacker gaining access to a given session private key would be able to decrypt at most 10 messages.

Integrity is assured via the use of a MAC for every message, initially using the shared secret s_0 and

subsequently using each renegotiated secret. This means that a tampered message, unless one that by some incredible coincidence hashed to the same MAC as the legitimate message, would be detectable via checking the MAC.

Authentication is provided by the use of personal public key signing on the initial two messages, which should serve to tie them to their purported senders' identities, and with the use of shared, re-negotiated ephemeral session keys to encrypt and generate MACs for each subsequent message. An attacker could only impersonate a given party when first connecting to the server if they had access to their personal private key, which would be a problem rather greater than the powers of one protocol to fix.

Replay attacks involve the capture of legitimate packets, which can then be sent by the attacker in such a way as to abuse features of the protocol design. The use of initially-random sequence numbers in all of the messages, subsequently incremented by unpredictable values, ensures that captured packets rapidly become ineffectual, and their use will result in a closed connection.

Delay attacks involve an attacker holding legitimate packets back, to be allowed through when needed to abuse protocol features. The use of the 25-second time-to-live period limits the risks of such an attack, and the failure of any party to receive a response within 45 seconds should assist in the detection of packet hold-up, indicating a man-in-the-middle attempting to perform a delay attack.

A denial-of-service attack involves blocking a legitimate user from being able to access a given service. A man-in-the-middle could act to drop all of C's messages before they reached S, for example, which would result in a dropped connection, but there is no security implication here. Such an attack could be avoided by finding a different communication channel to send over, and is thus outside of the scope of this protocol.

Padding oracle attacks are avoided by the use of AES-GCM, rather than AES-CBC, as well as the use of encrypt-then-MAC rather than vice versa.

Other attacks, such as timing and speculative execution attacks, are not considered to be relevant to this protocol.

The MACs for the first two messages ($HELLO$ and $HELLO\_ACK$) use the initial shared secret s_0 as their key every time. However, these first two messages are sent in the clear, and the MAC used only for authentication and integrity-checking, so this should pose no risk to perfect forward secrecy. Additionally, if it was considered an issue, new initial shared secrets could presumably be shared via the same mechanism as the current one.

There are situations in which one C sends two subsequent messages, without waiting for an ACK from S for the first. These situations occur whenever C sends a $DATA\_REQUEST$ message, such as after they send their ACK in the initial three-way handshake or after they send their $DATA\_ACK$ to indicate a succesfully-received $DATA\_RESPONSE$. This could lead to a situation in which the $DATA\_REQUEST$ messages arrives before the ACK/$DATA\_ACK$, which may cause S to close the connection. However, this can be mitigated with a time delay on C's end between the sending of both messages. Other aspects of the protocol design keep this from being a security issue.

Finally, as mentioned previously, the interpretation of the specification has relied on the inference that data is released by S in response to requests from C, but without the ability to specify the exact data requested. From that, the assumption has been made that specific data have no semantic meaning. This is important in the case of the key re-negotiation. As S sends their $HELLO$ after receiving a $DATA\_ACK$, and C sends their next $DATA\_REQUEST$ after sending their $DATA\_ACK$, it is possible that the key re-negotiation will be initiated and C's request for data ignored. Upon completion of the re-negotiation, C sends another

$DATA\_REQUEST$. Obviously, if it mattered exactly *what* data C was requesting, this would be an imperfect solution, and one in which S stored unfulfilled $DATA_REQUEST$s during the key-negotiation to action later would be preferable. However, there seems to be no security implications of this.

B. Test Cases

For these test cases, an implementation is assumed to have followed the 'SHOULD's and 'SHOULD NOT's, and ignored the 'MAY's.

- A Client or Server could send messages to a Server in an incorrect order to see how the other party handles receiving unexpected input. The expected response would be a $CLOSE$ message.
- A Client could send an initial $HELLO$, but send no ACK in response to the returned $HELLO\_ACK$. The expected response would be a $CLOSE$ message.
- A Server could not respond to an initial $HELLO$. The expected response would be a $CLOSE$ message.
- A Client or Server could send a message encrypted under an old session key. The expected response would be a $CLOSE$ message.
- A Client or Server could send a message with an incorrect sequence number (either incremented incorrectly from the previous number, or not matching a relevant message's number). The expected response would be a $CLOSE$ message.
- A Client or Server could not respond after receiving a $CLOSE$ message. The expected response would be a $CLOSED\_WITHOUT\_ACK$ message.
- A Client or Server could send a message with an incorrect MAC. The expected response would be a $CLOSE$ message.