SILENT CIRCLE ENGINEERING

Silent Circle Instant Messaging Protocol

libscimp API guide

Author: Vinnie Moscaritolo

Date: October 23, 2012

Version: Preliminary 0.11

Table of Contents

Implementation

The protocol is implemented in libscimp, currently an XCODE project that builds for both OSX and IOS.

Crypto is implemented with LibTomCrypt and LibTomMath, by Tom St Denis.

We added the following:

- Skein and SkeinMAC using Doug Whiting's public code.
- SHA-512/256 using vectors from NIST
- wrapper code defined in cryptowrappers.h and implemented in tomcryptwrappers.c that abstract the crypto layer enough that if we want to replace the crypto later, it won't be a big deal.

There is also a operation test (scimpTest.c) that puts the SCIMP library through it's paces and verifies each of the entry points and callbacks.

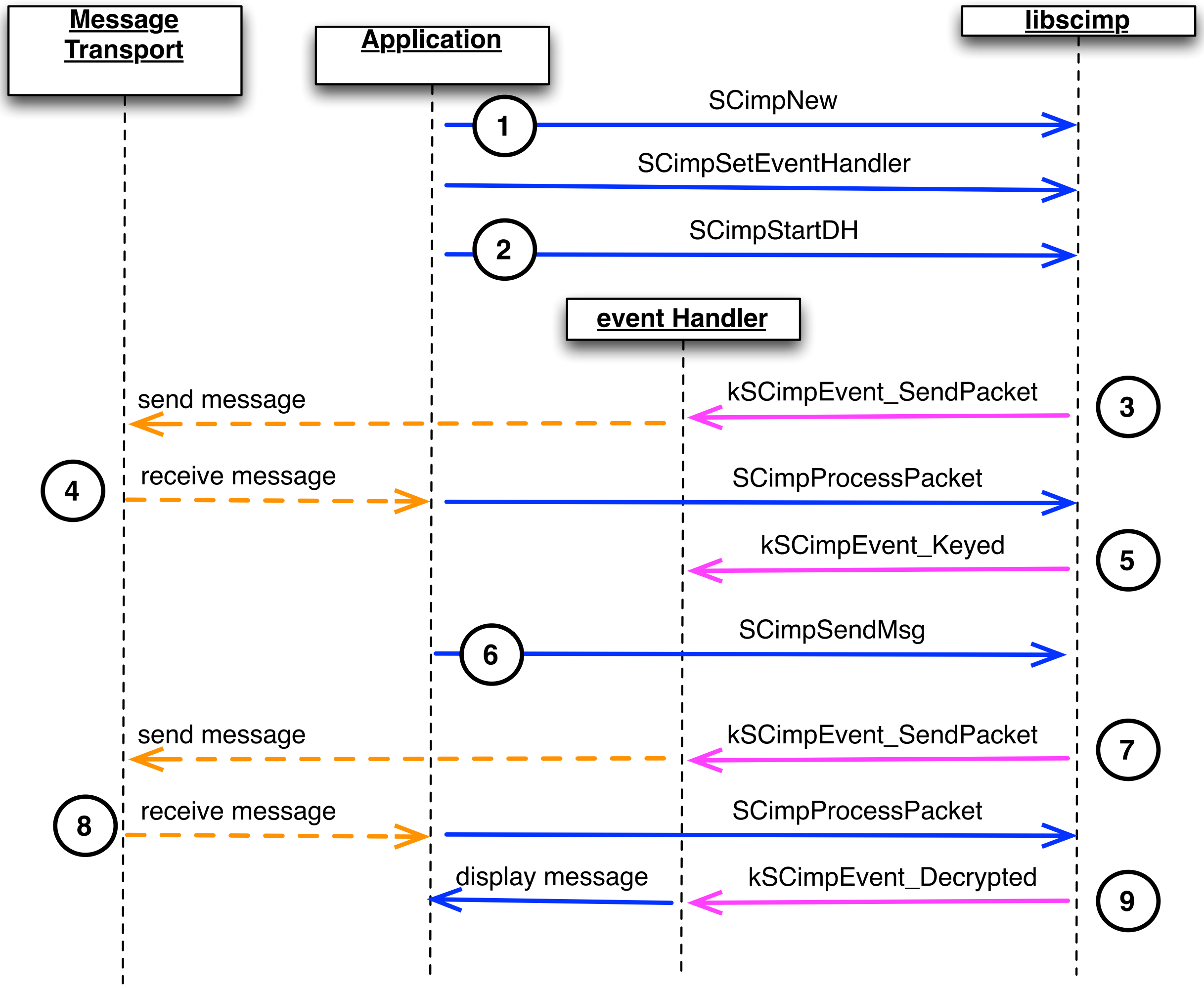
Using the libscimp API

The libscimp library has a number of entry points that abstracts the cryptography that is performed with it. The SCimp.h header file defines all the entry-points and constants.

Each conversation is managed by a separate SCIMP context object. This object can be created by the SCimpNew() API and freed by the SCimpFree().

Specific properties of a SCIMP context can be queried and modified by the various GET and SET calls depending on the item's type.

Once a SCIMP object is created, scimplib uses a callback method to communicate events to both the application and the network that will transport the messages.



scimplib message flow

1. Application creates a SCimp Context SCimpNew() and sets up an event handler with SCimpSetEventHandler.
2. Application starts the key establishment process by calling SCimpStartDH()
3. libscimp sends a kSCimpEvent_SendPacket event to event handler, which relays packet to the appropriate message transport (XMPP) .
4. Messages are received from XMPP and passed to scimplib through SCimpProcessPacket().

5. Once the key establishment process is handled, the event handler will notify the application with a kSCimpEvent_Keyed event. At which point the application can commence with secure traffic.
6. The application initiates a message to the other party by calling libscimp with SCimpSendMsg().
7. The message is encrypted with the appropriate key and sent to the event handler through the kSCimpEvent_SendPacket event, which once again relays the packet to the appropriate message transport (XMPP).
8. When the other party replies with a message, it is sent to libscimp through the SCimpProcessPacket() call.
9. The message is decrypted by libscimp and the event handler is notified with a kSCimpEvent_Decrypted event. The event handler should pass this decrypted message to the proper entry point in the application that displays this message to the user.

SCimpNew

Create new SCIMP object.

Syntax

```
      SCLError SCimpNew(
          char*                  meStr,
          char*                  youStr,
          SCimpContextRef *      outScimp );
```

Parameters

Parameters	Description
meStr	C string of this party's JID
youStr	C string of other party's JID
outScimp	pointer new SCIMP context

SCimpFree

Free existing SCIMP object.

Syntax

```
      void SCimpFree(SCimpContextRef scimp);;
```

Parameters

Parameters	Description
scimp	scimp context to free

SCimpSetEventHandler

Set event handler for SCIMP object.

Syntax

```
SCLError SCimpSetEventHandler(
         SCimpContextRef scimp,
         ScimpEventHandler handler, void* userValue);
```

Parameters

Parameters	Description
scimp	scimp context
handler	pointer to handler code
userValue	object to pass to handler

Notes:

scimplib event handler

```
SCLError sEventHandler(SCimpContextRef ctx,
                       SCimpEvent* event, void* uservalue)
{
    SCLError     err  = kSCLError_NoErr;
      switch(event->type)
      {
        // handle SCIMP events

      }

      return err;
}
```

Events sent to ScimpEventHandler

Send Packet Event

Data that needs to be transmitted to other party

data (uint8_t*)	pointer to data packet
length (size_t)	length in bytes of data

Decrypted Event

data arrived from other party

data (uint8_t*)	pointer to data packet
length (size_t)	length in bytes of data

ClearText Event

data arrived from other party unencrypted

data (uint8_t*)	pointer to data packet
length (size_t)	length in bytes of data

Keyed Event

scimplib finished the key negotiation process

version (uint8_t)	protocol version we are using
ciphersuite (enum)	SCimpCipherSuite
sasMethod (enum)	SCimpsas
isInitiator (bool)	we are initiator
hasCS (bool)	we have existing shared secret
csMatches (bool)	existing shared secrets matched with other party

Re-Keying Event

other party forced a rekey -

same data as Keyed Event

Error Event

A fatal error occurred while operating
error (SCLError) Error Number

kSCLError_BadIntegrity - during keying this means that the other party's PK hash or confirmation code did not match the key they sent, indicates a error or a possible MITM attack, during message transfer this could indicate that the message integrity did not match

kSCLError_CorruptData - indicates that the protocol received something it could not process.

Warning Event

A non fatal error occurred while operating
warning (SCLError) Error Number

kSCLError_SecretsMismatch - Shared secrets did not match

Shutdown Event

The other party requested that they are done communicating and possibly shutting down,

SCimpSendMsg

Send an encrypted message to the other party

Syntax

```
SCLError SCimpSendMsg(SCimpContextRef  scimp,
                      void*            data,
                      size_t           dataLen);
```

Parameters

Parameters	Description
scimp	scimp context
data	pointer to data to send to other party
dataLen	length in bytes of data to encrypt

SCimpProcessPacket

Process this message received from other party

Syntax

```
SCLError SCimpProcessPacket(SCimpContextRef    scimp,
                        void*          data,
                        size_t         dataLen,
                         uint8_t*      msgId,
                        size_t         msgIdLen);
```

Parameters

Parameters	Description
scimp	scimp context
data	pointer to data from other party
dataLen	length in bytes of data to process
msgID	optional Message ID to pass to event handler
msgIdLen	length in bytes of message ID

SCimpStartDH

Start the keying process

Syntax

```
      SCLError SCimpStartDH(SCimpContextRef scimp);
```

Parameters

Parameters	Description
scimp	scimp context

SCimpAcceptSecret

Accept new shared secret

Syntax

```
      SCLError SCimpAcceptSecret(SCimpContextRef scimp);
```

Parameters

Parameters	Description
scimp	scimp context

SCimpSaveState

Save current conversation state information

Syntax

```
SCLError SCimpSaveState(SCimpContextRef scimp,
                        uint8_t         *key,
                        size_t          keyLen,
                        void            **outBlob,
                        size_t          *blobSize);
```

Parameters

Parameters	Description
scimp	scimp context
key	pointer to 256 or 512 bit blob encryption key
keyLen	length of key
outBlob	pointer to where to store malloced state information
blobSize	pointer to where to store length of state information

Notes:

The information generated by the save state call consists of critical security parameters, such as the secret keys used to manage the conversation. In order to make this more secure, the application must pass in a key to encrypt the state before storing this information.

The app should call free() with a pointer returned in outBlob when done..

On IOS this call can be made in response to an `applicationWillTerminate` event.

SCimpRestoreSCIMP

restore previously saved conversation state information.

Syntax

```
    SCLError SCimpRestoreState      (void         *blob,
                                   uint8_t        *key,
                                   size_t         keyLen,
                                   size_t          blobSize,
                                   SCimpContextRef *outscimp);
```

Parameters

Parameters	Description
blob	pointer to state information
key	pointer to 256 or 512 bit blob encryption key
keyLen	length of key
blobSize	length of state information
outscimp	pointer to where to store scimp context

Other APIs

need to document these

```
SCLError SCimpGetNumericProperty( SCimpContextRef scimp,
                             SCimpProperty whichProperty,
                             uint32_t *prop);

SCLError SCimpSetNumericProperty( SCimpContextRef scimp,
                             SCimpProperty whichProperty,
                             uint32_t prop);

SCLError SCimpGetDataProperty( SCimpContextRef scimp,
                          SCimpProperty whichProperty,
                          void *buffer, size_t bufSize, size_t *datSize);

SCLError SCimpGetAllocatedDataProperty( SCimpContextRef scimp,
                                   SCimpProperty whichProperty,
                                   void **outData, size_t *datSize);

SCLError SCimpSCimpDataProperty( SCimpContextRef scimp,
                          SCimpProperty whichProperty,
                          void *data,  size_t  datSize);

SCLError SCimpGetInfo( SCimpContextRef scimp, SCimpInfo* info);


SCLError SCimpEnableTransitionEvents(SCimpContextRef  scimp, bool enable);

SCLError  SCimpGetVersionString(size_t bufSize, char *outString);
```

Appendix A: Document History

Date	Rev	Author	Change
10/19/12	0.9	vin	cleanup and split API into separate doc
10/22/12	0.10	vin	typos
10/23/12	0.11	jim	more typos