PhoneX messaging protocol draft
(confidential, subject to NDA)

Dušan Klinec, Miroslav Svítok

October 9, 2014

1 Message protocol v1 - S/MIME

Message protocol v1 uses S/MIME message format. Plaintex message format entering the encryption is:

```
s1 = ||1||source-sip||date-of-creation||nonce||text-to-send
```

- `1` is the version of the protocol.
- `source-sip` denotes creator of the message.
- `date-of-creation` denotes date when message was created, i.e., it is a number of milliseconds since Jan. 1, 1970, midnight GMT.
- `nonce` is a random integer. Supposed to serve as a protection against a replay attack. Moreover, it helps to detect duplicate messages due to lost ACK packets.
- `text-to-send` is a plaintext message entered by user in the application interface.

1.1 Message build algorithm

1. A compression of the $s1$ message to reduce a message size.
   `s2 = compress(s1)`
2. Compressed message is signed with the private key of the sender. Scheme used is `SHA1withRSA`. Sender certificate is added to the message so recipient is able to verify the signature. This step is required. Disadvantege is that extra certificate increases size of the sent message.
   `s3 = sign(s2, SHA1withRSA)`
3. Signed message is then encrypted with `AES128-CBC`, using an unique symmetric key encrypted with the recipient's public key.
   `s4 = encrypt(s3, RSA, AES128-CBC)`
4. Message is base64 encoded because SIP messaging protocol supports only text messages to be transmitted.
   `s5 = base64_encode(s4)`

2 Message protocol v2 - based on protocol buffers

Design of the next messaging protocol was inspired by problems encountered with messaging protocol v1. Namely it should solve the following issues:

- Allow for a better and modular implementation. Aim for forward compatibility to support future protocol extensions. Both secure transport protocol and message protocol can have different versions and separate dispatchers in the implementation handling the protocol type.

- Message protocol: Add support for a different type of messages, e.g., command messages, push notification messages, ACK messages. Each can have a different message format and a different dispatcher handling them.
- Reduce the message size: adding a certificate should be an optional step especially in the interactive communication. Attaching a certificate is ineffective in interactive communication. Simple message takes 1kB of data (base64 encoded). Sending such a big message with the mobile internet connection proved to be non-reliable due to high packet loss / transmission errors.
- Reduce the message code handling footprint. Currently, *BouncyCastle S/MIME* library is used for handling the message processing, i.e., encryption, decryption, signatures, etc... It has non-trivial code footprint and complexity. S/MIME protocol might be too complicated for some purposes (i.e., signaling messages) and cannot be used as a general purpose secure transport protocol.

2.1 Message protocol ideas

- The core idea is to clearly separate a *secure transport protocol (STP)*, responsible for a secure authenticated message transport to the destination and a *application message protocol (AMP)* , responsible for IM, signaling, etc... The design is inspired by ISO/OSI encapsulation/port multiplexing. The idea is to have multiple simpler protocols rather than one big, complex and full-fledged protocol.
- Both STP and AMP contains version number (like IP), in order to support future protocol extensions.
- There may be several distinct STPs defined, each providing different security properties. For example signed STPs provides non-repudiation while STPs featuring deniable authentication provide privacy. There may be a plain STP providing only authentication, or using only simple symmetric encryption using shared secret keys. Plain STP may be used e.g., for ACK messages.
- STP is level 1 protocol, AMP is level 2 protocol (AMP is encapsulated in / carried by STP).
- One of the STP can be S/MIME protocol, already used as v1 message protocol.
- Whole message protocol should be independent on the transport protocol used to deliver messages to the endpoints. It can be a text-only transport protocol, such as SIP SIMPLE or XMPP message, thus the whole message can be base64 encoded before sending.

2.2 Message protocol design

```
message MessageProtocolEnvelope {
  optional int protocolType=1;
  optional int protocolVersion=2;
  optional int sequenceNumber=3; // Optional.
  optional bytes payload=4;
}
```

Payload field contains particular STP. Mapping of the `protocolType` number to the particular STP:

```
1 = S/MIME, protocol v1.
2 = Lightweight STP.
3 = Group STP.
4 = FileTransfer protocol.
5 = OTR protocol.
6 = Shared secret key protocol.
7 = Plain protocol.
```

2.3 STPSimple, v1

Lightweight STP, using hybrid encryption, digital signature. Protocol does not provide PFC (perfect forward secrecy).

Application message passed from upper layer (AMP) is denoted as `messagePayload`.

```
message STPSimple {
  // Application message protocol type (e.g., IM, push, ...)
  // The message is routed to the upper layer dispatcher depending on this number.
  optional int32 ampType=1;

  // AMP version number. Supports extensions.
  optional int32 ampVersion=2;

  // OPT. Application message sequence number. For ACK purposes, message addressing.
  optional uint32 sequenceNumber=3;

  // Timestamp of the message. Replay attack, timestamping.
  optional uint64 messageSentMilliUTC=4;

  // Random integer for the message. Replay attack.
  optional uint32 randomNonce=5;

  // Sender address identification.
  optional string sender=6;

  // Message destination.
  optional string destination=7;

  // OPT. Message destination type (address/hash(nonce:address))
  optional int32 destinationType=8;

  // Digital signature of the message.
  optional bytes signature=9;
  // OPT.
  optional int32 signatureVersion=10;

  // OPT. Certificate version used for the signature.
  optional string certificateVersion=11;

  // Initialization vector for symmetric encryption.
  optional bytes iv=12;

  // RSA encrypted block. RSA/ECB/OAEPWithSHA1AndMGF1Padding
  optional bytes easymBlock=13;
  // OPT.
  optional int easymBlockVersion=14;

  // Symmetric block, ciphertext
  optional bytes esymBlock=15;
  // OPT.
  optional int esymBlockVersion=16;

  // HMAC on the message fields
  optional bytes hmac=17;
  // OPT.
  optional int hmacVersion=18;

  // Helper fields, used only for signature & HMAC.
  optional bytes encKey=19;
  optional bytes macKey=20;
  optional bytes payload=21;

  // Helper fields, used only for MAC computation.
  optional int32 protocolType=22;
  optional int32 protocolVersion=23;
}
```

Fields discussion.

- `Sequence` number is optional. Intended purpose is to be able to ACKnowledge some messages or answer on some messages.
- `randomNonce` is required as an replay attack protection.
- `destination` address i.e., the recipient address can be either plaintext, or hashed. By default, `destinationType` can be omitted. Default is plaintext recipient address.

  Intended purpose is to hide application message destination (can differ from underlying protocol pseudonym) from server or other recipients. Intended recipient can easily verify if the message is addressed for him, others has to perform costly computation. But it is not encryption, attacker still can test the destination.
- `signatureVersion` contains signature scheme used for the digital signature. This field can be omitted, default is RSA signature.
- `signature` is a RSA digital signature protects key message fields and the plaintext message. Signing plaintext is intended to bind plaintext message to the sender identity. Signature binds following fields together:

  – `protocolType`
  – `protocolVersion`
  – `ampType`
  – `ampVersion`
  – `sequenceNumber`
  – `randomNonce`
  – `sender`
  – `destination`
  – `destinationType`
  – `iv`, a initialization vector for the symmetric encryption.
  – `encKey01`, a symmetric encryption key for the symmetric encryption block.
  – `macKey01`, a HMAC symmetric key.
  – `messagePayload`

  $sig = SIGN_{privKey}(hash(protocolType : protocolVersion : ampType : ampVersion : sequenceNumber : randomNonce : sender : destination : destinationType : iv : encKey01 : macKey01 : messagePayload))$
- `iv` is an initialization vector for a symmetric encryption. If the encryption algorithm does not require IV (e.g., ECB. Note: do not use ECB!) this field is omitted.

- `easymBlockVersion` contains asymmetric encryption scheme version, field can be omitted, default is `RSA/ECB/OAEPWithSHA1AndMGF1Padding`.
- `easymBlock` contains asymmetrically encrypted symmetric keys. Asymmetric block size depends on the key size, but should be 2048 bits at least if using RSA encryption, thus several encryption keys fits this block. $easymBlock = AENCRYPT(encKey01 : macKey01)$
- `esymBlockVersion` contains symmetric encryption scheme version, field can be omitted, default is `AES/GCM/NoPadding`, 256-bit key. AES Galois Counter Mode is a preferred way of encryption. It is authenticated encryption (simultaneously encrypts and authenticates the plaintext data). In GCM mode, no padding is used (not a bug).
- `esymBlock` contains symmetrically encrypted `messagePayload`. Initialization vector is concatenated with the ciphertext. $esymBlock = IV||ENCRYPT(iv, encKey01, messagePayload)$
- `hmacVersion` contains hmac scheme used, field can be omitted, default is `HmacSHA1`.
- `hmac` HMAC on the message fields. Protects same fields as signature + signature field, asymmetric encryption block and symmetric encryption block. Signature protects plaintext version of the message while HMAC protects whole message (ciphertext version). Helps to avoid chosen ciphertext attack. HMAC is easy to generate & verify, has a small footprint while added security benefit is non-trivial. Signature protects plaintext version of the message.

  Note that it is a symmetric technique so attacker can still create packets with valid HMAC. Digital signature has to be verified (mac key is signed as well).

  - `protocolType`
  - `protocolVersion`
  - `ampType`
  - `ampVersion`
  - `sequenceNumber`
  - `randomNonce`
  - `sender`
  - `destination`
  - `destinationType`
  - `iv`, initialization vector for symmetric encryption.
  - `encKey01`, symmetric encryption key for the symmetric encryption block.
  - `macKey01`, hmac symmetric key.
  - `easymBlock`
  - `easymBlockVersion`
  - `esymBlock`
  - `esymBlockVersion`

  $hmac = HMAC(macKey01, protocolType : protocolVersion : ampType : ampVersion : sequenceNumber : randomNonce : sender : destination : destinationType : iv : encKey01 : macKey01 : easymBlock : easymBlockVersion : esymBlock : esymBlockVersion))$
- `encKey` [VIRTUAL] is symmetric encryption key extracted from asymmetric encryption block.
- `macKey` [VIRTUAL] is symmetric HMAC key extracted from asymmetric encryption block.
- `payload` [VIRTUAL] is decrypted symmetric block.
- `protocolType` [VIRTUAL] is protocol type - taken from lower protocol. It is a type of this protocol.
- `protocolVersion` [VIRTUAL] is protocol version - taken from lower protocol. It is a type of this protocol.

Virtual message fields. Fields marked as [VIRTUAL] are helper fields used only in offline phase, during message creation or message decryption. It helps to generate/verify a digital signature and HMAC on additional fields in a following way: these fields are filled in, signature and/or HMAC is generated/verified, virtual fields are erased and signature and HMAC is added to the message. Virtual messages must not be filled in during message transfer.

Version number discussion. There are several ways how to encapsulate two different protocols. Internet Protocol (IP) has version defined inside, while type of the protocol is defined in an underlying layer. We decided to add version number to the underlying layer so there are no further requirements on the upper message format. E.g., parser does not have to assume that the message is still ProtocolBuffers encoded with version number as a first field in the message, so the different version of the protocol message can change its format significantly. Note that this affects only a message dispatcher handling particular message type.

2.4 STPSimpleGroup

This section proposes an extension for the STPSimple protocol for group messaging. Idea: instead of flat key structure, concept of *keyslots* is proposed.

The AMP payload is encrypted with master symmetric encryption key, MK. For each recipient the STP message contains one key slot. Each key slot encrypts MK with public key of the recipient.

Benefit: single message can be sent to multiple recipients (e.g., by the server) without modification. Some broadcasting feature of the underlying transport protocol can be used to optimize group messages in bandwidth constrained environments.

It can be used for group messaging or for one user account enabled on different user devices (multiple instances), for each instance, a different key/certificate is used. Assumption: instances are distinguishable from each other by some ID/pseudonym, as in XMPP case.

This is proposed as a separate protocol in order to simplify protocol design and use of the protocol. In many cases the simple version suffices.

2.5 Further variants of STPs

There may be different multiple STPs suitable for different purposes.

- FileTransfer protocol. We may use our FT protocol already employed in the PhoneX architecture. It provides Perfect Forward Secrecy and has good anti-SPAM properties.

- OTR[1] protocol. It provides Perfect Forward Secrecy and is de-facto standard for a secure communication in instant messages. Benefit is that the protocol is well-tested by community and provides *deniable encryption.* Disadvantage is that initial authenticated key exchange takes 2 round trips what can be problematic in some network limited setups.
  Proposed trade-off: Perform re-keying / authenticated key exchange once per time window (e.g., once a week), when both clients are connected to WiFi or other wide bandwidth network.

- Symmetric encryption / authentication protocol. When using OTR, both sides posses shared secret symmetric keys that can be used for symmetric encryption and HMAC. This message has smaller footprint since digital signature and asymmetric encryption blocks are not used. Moreover, it is computationally effective and does not provide non-repudiation, which is sometimes undesirable.

- Plain STP. For some message types (ACKs), encryption is not necessary.

2.6 AMPIMSimple, v1

This section briefly describes application message protocol for instant messaging. This messaging protocol is very simple, it contains just one field with text message from the user.

[1] https://otr.cypherpunks.ca/Protocol-v3-4.0.0.html

```
message STPSimple {
  // Message sent by user. message=GZIP(plainMessage);
  optional bytes message=1;
}
```

Field message contains a GZIPed version of a plaintext message entered by user in UI. GZIP compression reduces overall message size.