XEP-0200: Stanza Encryption

Ian Paterson
mailto:ian.paterson@clientside.co.uk
xmpp:ian@zoofy.com

2007-05-30
Version 0.2

Status	Type	Short Name
Deferred	Standards Track	TO BE ASSIGNED

This document specifies an XMPP protocol extension for session-based stanza encryption.

Legal

Copyright

Permissions

Warranty

Liability

Conformance

Contents

1 Introduction

End-to-end encryption is a desirable feature for any communication technology. Ideally, such a technology would design encryption in from the beginning and would forbid unencrypted communications. Realistically, most communication technologies have not been designed in that manner, and Jabber/XMPP technologies are no exception. In particular, the original Jabber technologies developed in 1999 did not include end-to-end encryption by default. PGP-based encryption of message bodies and signing of presence information was added as an extension to the core protocols in the year 2000; this extension is documented in Current Jabber OpenPGP Usage (XEP-0027) [1]. When the core protocols were formalized within the Internet Standards Process by the IETF's XMPP Working Group in 2003 (see RFC 3920 [2] and RFC 3921 [3]), a different extension was defined using S/MIME-based signing and encryption of CPIM-formatted messages (see RFC 3862 [4]) and PIDF-formatted presence information (see RFC 3863 [5]); this extension is specified in RFC 3923 [6].

For reasons described in Requirements for Encrypted Sessions (XEP-0210) [7], the foregoing proposals (and others not mentioned) have not been widely implemented and deployed. This is unfortunate, since an open communication protocol needs to enable end-to-end encryption in order to be seriously considered for deployment by a broad range of users.

This document describes a different session-based approach to the end-to-end encryption of the full content of XMPP stanzas sent between two entities. The protocol assumes that the encrypted session parameters (initial keys, counters and algorithms etc.) have already been agreed, typically through a negotiation protocol such as Encrypted Session Negotiation (XEP-0116) [8], Simplified Encrypted Session Negotiation (XEP-0217) [9] or Offline Encrypted Sessions (XEP-0187) [10]. The session approach when combined with short-lived keys offers many important advantages over the existing "Object Encryption" proposals, including Perfect Forward Secrecy and Identity Protection.

2 Requirements

The requirements and the consequent cryptographic design that underpin this protocol and its associated protocols are described in Requirements for Encrypted Sessions (XEP-0210)

[1] XEP-0027: Current Jabber OpenPGP Usage <https://xmpp.org/extensions/xep-0027.html>.
[2] RFC 3920: Extensible Messaging and Presence Protocol (XMPP): Core <http://tools.ietf.org/html/rfc3920>.
[3] RFC 3921: Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence <http://tools.ietf.org/html/rfc3921>.
[4] RFC 3862: Common Presence and Instant Messaging (CPIM): Message Format <http://tools.ietf.org/html/rfc3862>.
[5] RFC 3863: Presence Information Data Format (PIDF) <http://tools.ietf.org/html/rfc3863>.
[6] RFC 3923: End-to-End Signing and Object Encryption for the Extensible Messaging and Presence Protocol (XMPP) <http://tools.ietf.org/html/rfc3923>.
[7] XEP-0210: Requirements for Encrypted Sessions <https://xmpp.org/extensions/xep-0210.html>.
[8] XEP-0116: Encrypted Session Negotiation <https://xmpp.org/extensions/xep-0116.html>.
[9] XEP-0217: Simplified Encrypted Session Negotiation <https://xmpp.org/extensions/xep-0217.html>.
[10] XEP-0187: Offline Encrypted Sessions <https://xmpp.org/extensions/xep-0187.html>.

[11] and Cryptographic Design of Encrypted Sessions. The specific objectives of this protocol include:

- Encryption of the full stanza content (except that which is required for stanza routing)
- Minimise the Perfect Forward Secrecy window by enabling light-weight renegotiation of the short-term keys without requiring a full session renegotiation.
- Minimise bandwidth overhead
- No dependence on XML canonicalization

3 Dramatis Personae

This document introduces two characters to help the reader follow the necessary exchanges:

1. ”Alice” is the name of the entity sending the encrypted stanza or initiating a re-key. Within the scope of this document, we stipulate that her fully-qualified JID is: <alice@example.org/pda>.
2. ”Bob” is the name of the entity receiving the encrypted stanza or accepting a re-key. Within the scope of this document, his fully-qualified JID is: <bob@example.com/laptop>.

While Alice and Bob are introduced as ”end users”, they are simply meant to be examples of XMPP entities. Any directly addressable XMPP entity may send or receive encrypted stanzas within an encrypted session or initiate a re-key.
Here we assume that Alice and Bob have already established an encrypted session. Either Alice or Bob MAY send encrypted stanzas within the encrypted session or initiate a re-key. The following sections describe the process where Alice sends Bob an encrypted stanza and initiates a re-key.

4 Assumptions

The following sections assume that the parameters in the tables below have already been agreed. For more details refer to an encrypted session negotiation protocol such as Encrypted Session Negotiation.

[11] XEP-0210: Requirements for Encrypted Sessions <https://xmpp.org/extensions/xep-0210.html>.

Parameter	Description
CA, CB	The initial block cipher counter value for blocks sent by Alice and Bob
KCA, KCB	The initial secret cipher keys that Alice and Bob use to encrypt
KMA, KMB	The initial secret MAC keys that Alice and Bob use to protect the integrity of encrypted data
HASH	Agreed hash algorithm
CIPHER, DECIPHER	Agreed CTR-mode block cipher algorithm
COMPRESS, DECOMPRESS	Agreed compression algorithm

Parameter	Description
g	Diffie-Hellman generator
p	Diffie-Hellman prime
e, d	Alice and Bob's initial public Diffie-Hellman keys
x, y	*Alice and Bob's initial private Diffie-Hellman keys

* x and y MUST be known only to Alice and Bob respectively, all other parameters MUST be known by both parties
All parameters except the algorithms are multi-precision integers, so implementations will need a Big Integer Math library to perform the necessary modular arithmetic. Note: A simple HMAC function and a cryptographic-strength pseudo-random number generator are also required, but no other cryptographic code is necessary.

5 Encryptable Content

Alice MAY use this protocol to encrypt only that part of the *content* of one-to-one <message/>, <presence/> and <iq/> stanzas that would normally be ignored by the intermediate servers. She MUST NOT encrypt:

- Stanza wrapper element tags (only stanza content)
- <error/> elements [12]

[12] RFC 6120 requires that stanzas of type 'error' contain an <error/> child element.

- <defined-condition xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/> child elements of <error/> elements. [13]

- <thread/> elements [14]

- <amp/> elements (see Advanced Message Processing (XEP-0079) [15])

A stanza MUST NOT contain more than one <c xmlns='http://www.xmpp.org/extensions/xep-0200.html#ns'/> element, and it MUST be an immediate child of the stanza wrapper element. There is only one exception to those two rules, if the stanza is type 'error' then its <error/> child element MAY also contain a <c xmlns='http://www.xmpp.org/extensions/xep-0200.html#ns'/> element.

Listing 1: Plain Message Stanza

```
<message from='alice@example.org/pda'
         to='bob@example.com/laptop'
         type='chat'>
  <thread>ffd7076498744578d10edabfe7f4a866</thread>
  <body>Hello, Bob!</body>
  <amp xmlns='http://jabber.org/protocol/amp'>
    <rule action='error' condition='match-resource' value='exact'/>
  </amp>
  <active xmlns='http://jabber.org/protocol/chatstates'/>
</message>
```

Listing 2: Message Content to be Encrypted

```
<body>Hello, Bob!</body>
<active xmlns='http://jabber.org/protocol/chatstates'/>
```

Listing 3: Plain Presence Stanza

```
<presence from='alice@example.org/pda'
          to='bob@example.com/laptop'>
  <show>dnd</show>
  <status>Working</status>
  <c xmlns='http://jabber.org/protocol/caps'
     node='http://exodus.jabberstudio.org/caps'
     ver='0.9'
```

[13] RFC 6120 requires that <error/> elements contain a <defined-condition/> child element.

[14] Applications typically use <thread/> elements internally to route stanzas to the process handling a session. The content of thread elements MUST be opaque with no semantic meaning and only exact comparisons MAY be made against it.

[15] XEP-0079: Advanced Message Processing <https://xmpp.org/extensions/xep-0079.html>.

```
      ext='jingle_ftrans_xhtml'/>
  <geoloc xmlns='http://jabber.org/protocol/geoloc'>
    <alt>1609</alt>
    <description>Jabber, Inc.</description>
    <error>10</error>
    <lat>39.75477</lat>
    <lon>-104.99768</lon>
    <timestamp>2004-02-19T21:12Z</timestamp>
  </geoloc>
</presence>
```

Listing 4: Presence with Encrypted Content

```
<presence from='alice@example.org/pda'
          to='bob@example.com/laptop'>
  <c xmlns='http://www.xmpp.org/extensions/xep-0200.html#ns'>
    <data> ** Base64 encoded m_final ** </data>
    <mac> ** Base64 encoded a_mac ** </mac>
  </c>
</presence>
```

Listing 5: Plain IQ Error Stanza

```
<iq from='alice@example.org/pda'
    to='bob@example.com/laptop'
    id='publish1'
    type='error'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <publish node='princely_musings'>
      <item id='ae890ac52d0df67ed7cfdf51b644e901'>
        ...
      </item>
    </publish>
  </pubsub>
  <error type='modify'>
    <not-acceptable xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
    <payload-too-big xmlns='http://jabber.org/protocol/pubsub#errors'/
       >
  </error>
</iq>
```

Listing 6: Encrypted IQ Error Stanza

```
<iq from='alice@example.org/pda'
    to='bob@example.com/laptop'
    id='publish1'
    type='error'>
  <c xmlns='http://www.xmpp.org/extensions/xep-0200.html#ns'>
    <data> ** Base64 encoded m_final ** </data>
    <mac> ** Base64 encoded a_mac ** </mac>
```

```
  </c>
  <error type='modify'>
    <not-acceptable xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
    <c xmlns='http://www.xmpp.org/extensions/xep-0200.html#ns'>
      <data> ** Base64 encoded m_final ** </data>
      <mac> ** Base64 encoded a_mac ** </mac>
    </c>
  </error>
</iq>
```

6 Encrypting a Stanza

Alice MUST perform the following steps to encrypt the XML content. Note: if there is no XML content to be encrypted (e.g. if this is an empty Re-Key Exchange stanza), then CA MUST be incremented by 1 (see below), and only the last two steps (normalization and MAC calculation) should be performed.

1. Serialize the XML content she wishes to send into an array of UTF-8 bytes, m. [16]

2. Compress m using the negotiated algorithm. If a compression algorithm other than 'none' was agreed, the compression context is typically initialized after key exchange and passed from one stanza to the next, with only a partial flush at the end of each stanza. [17]

```
m_compressed = COMPRESS(m)
```

3. Encrypt the data with the agreed algorithm in counter mode, using the encryption key KCA. Note: CA MUST be incremented by 1 for each encrypted block or partial block (i.e. CA = (CA + 1) mod 2^n, where n is the number of bits per cipher block for the agreed block cipher algorithm). Note: if the block cipher algorithm 'none' was agreed then encryption MUST NOT be performed and CA MUST be incremented by 1 (for replay protection).

```
m_final = CIPHER(KCA, CA, m_compressed)
```

4. Generate the whole serialized *content* of the <c/> element:
   If there is encrypted XML content, the XML MUST include the Base64 encoded (even if

[16] Although counter mode encryption requires no padding, implementations MAY still disguise the length of m by appending a random number of white-space characters.

[17] If Bob were to receive a stanza out-of-order, then he would fail to decrypt the stanza and be forced to terminate the encrypted session.

the block cipher algorithm 'none' was agreed [18], in accordance with Section 4 of RFC 4648 [19]) value of m_final wrapped in a <data/> element.
Only if Alice has received stanzas containing a <key/> element (see Re-Key Exchange) from Bob since she sent her last stanza then the XML MUST include the (positive, non-zero) number of such stanzas she has received (since she sent her last stanza) wrapped in a <new/> element.
The content may also contain one <key/> element (see Re-Key Exchange) and one or more <old/> elements (see Publishing Old MAC Values).
Alice MUST normalize the content by removing any whitespace from the serialized content (i.e. remove all character data from *between* all elements). Note: <c/> elements are so simple that there should *never* be a need to convert the XML to canonical form.
For example:

```
m_content = '<data>␣**␣Base64␣encoded␣m_final␣**␣</data><new>1</
    new>'
```

5. Process the XML content, concatenated with the value of Alice's block cipher counter CA *before* the data was encrypted, through the HMAC algorithm (as defined in Section 2 of RFC 2104 [20]), along with the agreed hash algorithm ("HASH") and the integrity key KMA.

```
a_mac = HMAC(HASH, KMA, m_content | CA)
```

7 Sending an Encrypted Stanza

Before sending the stanza to Bob, Alice MUST wrap the (unnormalized) content and the Base64 encoded value of a_mac (wrapped in a <mac/> element) inside an <c/> element and insert it into the stanza in place of the original content.

Listing 7: Message Stanza with Encrypted Content

```
<message from='alice@example.org/pda'
         to='bob@example.com/laptop'
         type='chat'>
  <thread>ffd7076498744578d10edabfe7f4a866</thread>
  <c xmlns='http://www.xmpp.org/extensions/xep-0200.html#ns'>
    <data> ** Base64 encoded m_final ** </data>
    <new>1</new>
    <mac> ** Base64 encoded a_mac ** </mac>
```

[18] The content is encoded even if no encryption is used to avoid triggering namespace errors when entities parse the XML.
[19] RFC 4648: The Base16, Base32, and Base64 Data Encodings <http://tools.ietf.org/html/rfc4648>.
[20] RFC 2104: HMAC: Keyed-Hashing for Message Authentication <http://tools.ietf.org/html/rfc2104>.

```
  </c>
  <amp xmlns='http://jabber.org/protocol/amp' per-hop='true'>
    <rule action='error' condition='match-resource' value='exact'/>
  </amp>
</message>
```

8 Decrypting a Stanza

When Bob receives the stanza from Alice, he MUST extract and Base64 decode the values of m_final and a_mac from the content and perform the following steps.

1. Use any <new/> element the stanza may (or may not) contain to determine which of his <key/> elements (see Re-Key Initiation) Alice had received before she sent him the stanza. He MUST use this information to determine which of his stored sets of values {KCA KMA y} he should use to decrypt and verify the stanza. Bob MUST then securely destroy any of his sets of values that are older than the selected set.

2. Remove the <mac/> element from the serialized content of the <c/> element, and normalize the remaining content by removing all whitespace. Calculate the Message Authentication Code (MAC) for the *content* concatenated with Alice's block cipher counter using the agreed hash algorithm ("HASH") and the integrity key KMA.

```
b_mac = HMAC(HASH, KMA, m_content | CA)
```

3. Verify that b_mac and a_mac match. If they are not identical, the content has been tampered with and Bob MUST terminate the encrypted session, he MAY send a <not-acceptable/> error to Alice. [21]

4. Decrypt m_final using the agreed algorithm, KCA and CA. Note: CA MUST be incremented by 1 for each decrypted block (see Encrypting a Stanza). Note: if the block cipher algorithm 'none' was agreed decryption MUST NOT be performed and CA MUST be incremented by 1.

```
m_compressed = DECIPHER(KCA, CA, m_final)
```

5. Decompress m_compressed using the negotiated algorithm (usually 'none').

[21] If Bob were to receive a stanza out-of-order, then the MACs would not match because the values of CA would not be synchronized.

```
m = DECOMPRESS(m_compressed)
```

6. Replace the <c/> element in the serialized XML stanza with m and feed the stanza into an XML parser. If the parser returns an XML format error then Bob MUST terminate the encrypted session, he MAY send a <not-acceptable/> error to Alice. [22]

9 Re-Key Exchange

9.1 Introduction

Once an attacker has discovered an encryption key it could be used to decrypt all stanzas within a session, including stanzas that were intercepted *before* the key was discovered. To reduce the window of vulnerability, both Alice and Bob SHOULD use the Diffie-Hellman key exchange described below to agree a new encryption key as regularly as possible. They MUST also destroy all copies of keys as soon as they are no longer needed.
Note: Although most entities are capable of exchanging new keys after every stanza, clients running in constrained runtime environments may require a few seconds of full-load CPU time in order to re-key. During encrypted session negotiation (when using Encrypted Session Negotiation for example) these clients MAY negotiate the minimum number of stanzas to be exchanged between re-keys at the cost of a larger window of vulnerability. Entities MUST NOT initiate key re-exchanges more frequently than the agreed limit.
Either Alice or Bob MAY initiate a key re-exchange. Here we describe the process initiated by Alice.

9.2 Re-Key Initiation

First Alice MUST calculate new values for the encryption parameters:

1. Generate: a secret random number x (where 2^{2n-1} < x < p - 1, where n is the number of bits per cipher block for CIPHER)

2. Calculate: e = g^x mod p

3. Calculate: K = d^x mod p (the new shared secret)

4. Alice's Encryption key: KCA = HMAC(HASH, K, "Rekey Initiator Crypt")

[22] Bob MUST NOT send a stream error to his server since intermediate entities are not responsible for encoded content.

5. Bob's Encryption key: KCB = HMAC(HASH, K, "Rekey Acceptor Crypt")

6. Alice's Integrity key: KMA = HMAC(HASH, K, "Rekey Initiator MAC")

7. Bob's Integrity key: KMB = HMAC(HASH, K, "Rekey Acceptor MAC")

Note: Once KCA, KMA, KCB and KMB have been calculated the value of K MUST be securely destroyed. When calculating those keys, as many bits of key data as are needed for each key MUST be taken from the *least* significant bits of the output of HMAC. For algorithms with variable-length keys the maximum length (up to the output length of HMAC) SHOULD be used.
The new value of e SHOULD be wrapped in a <key/> element and sent to Bob. To avoid unnecessary network traffic, it SHOULD be sent together with encrypted content (see Encrypting a Stanza). Alice MUST use her *old* KCA and KMA to encrypt and calculate the MAC of this stanza, after which she MUST securely destroy all copies of the old value of KCA. She MUST use her *new* KCA and KMA when sending *subsequent* stanzas.
Note: There is no need for Alice to provide a signature because the calculation of the MAC includes the new value of e (see Encrypting a Stanza).

Listing 8: Alice Sends Re-Key Stanza

```
<message from='alice@example.org/pda' to='bob@example.com/laptop'>
  <thread>ffd7076498744578d10edabfe7f4a866</thread>
  <c xmlns='http://www.xmpp.org/extensions/xep-0200.html#ns'>
    <data> ** Base64 encoded m_final ** </data>
    <key> ** Base64 encoded value of new e ** </key>
    <mac> ** Base64 encoded a_mac ** </mac>
  </c>
</message>
```

Note: Bob may send one of more stanzas before he receives Alice's <key/> element (i.e. the stanzas may be in transit at the same time). So, before destroying her old values of KCB and KMB, Alice MUST wait until either she receives a stanza encrypted with her new key, or a reasonable time has passed (60 seconds should cover a network round-trip and calculations by a constrained client). Similarly she MUST wait before destroying her old value of x, in case Bob sends one or more stanzas including a <key/> element before he receives Alice's new key. Consequently, if Alice sends several <key/> elements to Bob within a reasonable time without receiving a stanza from him, then she MUST remember several "sets" of the three values: {KCB KMB x}.
Note: Alice never remembers more than one copy of KCA and KMA.

9.3 Re-Key Acceptance

After receiving and decrypting a stanza that contains a <key/> element Bob MUST extract the new value of e and confirm that it is greater than one. Then he MUST calculate K using the new value of e and the value of y from his oldest stored set of values {KCA KMA y} (i.e. the set that also contains the value of KCA used to decrypt the stanza):

```
K = ey mod p
```

Bob MUST replace the values of KCA and KMA in *all* his stored sets of values {KCA KMA y} with values that are derived from K in exactly the same way as Alice did (see Re-Key Initiation).
Only if Bob is storing exactly one set of values {KCA KMA y} then he MUST also replace his values of KCB and KMB with values that are derived from K in exactly the same way as Alice did.

10 Publishing Old MAC Values

Once the expired MAC keys have been published, anyone could create valid arbitrary stanzas with them. This prevents anyone being able to prove the authenticity of a transcript of the encrypted session in the future.
Either entity MAY publish old values of KMA and/or KMB within any encrypted stanza as long as it knows that all the stanzas that MAY use the old values have been received and validated.
Note: A 'man-in-the-middle' could delay the delivery of stanzas indefinitely. So, before Alice publishes KMA (and KMB), she MUST wait until she has *both* sent a re-key to Bob *and* received a stanza from Bob encrypted with her new key. (She MAY also publish KMB after she has received a re-key from Bob.)

Listing 9: Publishing Expired MAC Keys

```
<message from='alice@example.org/pda' to='bob@example.com/laptop'>
  <thread>ffd7076498744578d10edabfe7f4a866</thread>
  <c xmlns='http://www.xmpp.org/extensions/xep-0200.html#ns'>
    <data> ** Base64 encoded m_final ** </data>
    <old> ** Base64 encoded old MAC key ** </old>
    <old> ** Base64 encoded old MAC key ** </old>
    <mac> ** Base64 encoded a_mac ** </mac>
  </c>
</message>
```

Entities SHOULD ignore any <old/> elements they receive.

11 Security Considerations

11.1 Random Numbers

Weak pseudo-random number generators (PRNG) enable successful attacks. Implementors MUST use a cryptographically strong PRNG to generate all random numbers (see RFC 1750 [23]).

11.2 Storage

If either entity stores a (re-encrypted) transcript of an encrypted session for future consultation then the Perfect Forward Secrecy offered by this protocol is lost. If the negotiated value of the 'otr' Stanza Session Negotiation field is 'true' the entities MUST NOT store any part of the encrypted session content (not even in encrypted form).

11.3 Replay Attacks

The block cipher counters maintained implicitly by Alice and Bob (CA and CB) prevent stanzas being replayed within any encrypted session. They ensure that the MAC will be different for all stanzas, even if the HMAC key and the content of the stanza are identical.
Alice and Bob MUST ensure that every value of x and y (and therefore e and d) they generate is unique. This prevents complete online encrypted sessions being replayed.

11.4 Maximum Key Life

After each key exchange an entity MUST NOT exchange a total of 2^{32} encrypted *blocks* (not stanzas) before it initiates a key re-exchange (see RFC 4344 [24]). Note: This limitation also ensures the same key and counter values are never used to encrypt two different blocks using counter mode (thus preventing simple attacks).
In order to reduce the Perfect Forward Secrecy window of vulnerability, after an extended period of inactivity entities SHOULD re-key (or terminate the encrypted session).

11.5 Extra Responsabilities of Implementors

Cryptography plays only a small part in an entity's security. Even if it implements this protocol perfectly it may still be vulnerable to other attacks. For examples, an implementation might store encrypted session keys on swap space or save private keys to a file in cleartext! Implementors MUST take very great care when developing applications with secure technologies.

[23] RFC 1750: Randomness Recommendations for Security <http://tools.ietf.org/html/rfc1750>.
[24] RFC 4344: SSH Transport Layer Encryption Modes <http://tools.ietf.org/html/rfc4344>.

12 IANA Considerations

This document requires no interaction with the Internet Assigned Numbers Authority (IANA) [25].

13 XMPP Registrar Considerations

13.1 Protocol Namespaces

Until this specification advances to a status of Draft, its associated namespace shall be "http://www.xmpp.org/extensions/xep-0200.html#ns"; upon advancement of this specification, the XMPP Registrar [26] shall issue a permanent namespace in accordance with the process defined in Section 4 of XMPP Registrar Function (XEP-0053) [27].

14 XML Schemas

```
<?xml version='1.0' encoding='UTF-8'?>

<xs:schema
    xmlns:xs='http://www.w3.org/2001/XMLSchema'
    targetNamespace='http://www.xmpp.org/extensions/xep-0200.html#ns'
    xmlns='http://www.xmpp.org/extensions/xep-0200.html#ns'
    elementFormDefault='qualified'>

  <xs:element name='data' type='xs:string'/>

  <xs:element name='c'>
    <xs:complexType>
      <xs:sequence>
        <xs:element ref='data' minOccurs='0' maxOccurs='1'/>
        <xs:element ref='key' minOccurs='0' maxOccurs='1'/>
        <xs:element ref='mac' minOccurs='0' maxOccurs='1'/>
        <xs:element ref='new' minOccurs='0' maxOccurs='1'/>
        <xs:element ref='old' minOccurs='0' maxOccurs='unbounded'/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
```

[25] The Internet Assigned Numbers Authority (IANA) is the central coordinator for the assignment of unique parameter values for Internet protocols, such as port numbers and URI schemes. For further information, see <http://www.iana.org/>.

[26] The XMPP Registrar maintains a list of reserved protocol namespaces as well as registries of parameters used in the context of XMPP extension protocols approved by the XMPP Standards Foundation. For further information, see <https://xmpp.org/registrar/>.

[27] XEP-0053: XMPP Registrar Function <https://xmpp.org/extensions/xep-0053.html>.

```
  <xs:element name='key' type='xs:string'/>
  <xs:element name='mac' type='xs:string'/>
  <xs:element name='new' type='xs:positiveInteger'/>
  <xs:element name='old' type='xs:string'/>
</xs:schema>
```