XEP-0116: Encrypted Session Negotiation

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

Peter Saint-Andre
mailto:peter@andyet.net
xmpp:stpeter@stpeter.im
https://stpeter.im/

Dave Smith
mailto:dizzyd@jabber.org
xmpp:dizzyd@jabber.org

2007-05-30
Version 0.16

Status	Type	Short Name
Deferred	Standards Track	TO BE ASSIGNED

This document specifies an XMPP protocol extension for negotiating an end-to-end encrypted session.

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 proposal describes a different approach to end-to-end encryption for use by entities that communicate using XMPP. The requirements and the consequent cryptographic design that underpin this protocol are described in Requirements for Encrypted Sessions and Cryptographic Design of Encrypted Sessions (XEP-0188) [8]. The basic concept is that of an encrypted session which acts as a secure tunnel between two endpoints. Once the tunnel is established, the content of each one-to-one XML stanza exchanged between the endpoints will be encrypted and then transmitted within a "wrapper" stanza using Stanza Encryption (XEP-0200) [9].

Simplified Encrypted Session Negotiation (XEP-0217) [10] describes a minimal subset of this protocol chosen to enable developers to produce working code before they have finished implementing the optional parts of this protocol.

[1] XEP-0027: Current Jabber OpenPGP Usage <http://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 <http://xmpp.org/extensions/xep-0210.html>.
[8] XEP-0188: Cryptographic Design of Encrypted Sessions <http://xmpp.org/extensions/xep-0188.html>.
[9] XEP-0200: Stanza Encryption <http://xmpp.org/extensions/xep-0200.html>.
[10] XEP-0217: Simplified Encrypted Session Negotiation <http://xmpp.org/extensions/xep-0217.html>.

2 Dramatis Personae

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

1. "Alice" is the name of the initiator of the ESession. 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 other participant in the ESession started by Alice. Within the scope of this document, his fully-qualified JID is: <bob@example.com/laptop>.
3. "Aunt Tillie" the archetypal typical user (i.e. non-technical, with only very limited knowledge of how to use a computer, and averse to performing any procedures that are not familiar).

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 participate in an ESession.

3 Discovering Support

Before attempting to engage in an ESession with Bob, Alice MAY discover whether he supports this protocol, using either Service Discovery (XEP-0030) [11] or the presence-based profile of XEP-0030 specified in Entity Capabilities (XEP-0115) [12].
The normal course of events is for Alice to authenticate with her server, retrieve her roster (see RFC 3921), send initial presence to her server, and then receive presence information from all the contacts in her roster. If the presence information she receives from some contacts does not include capabilities data (per XEP-0115), Alice SHOULD then send a service discovery information ("disco#info") request to each of those contacts (in accordance with XEP-0030). Such initial service discovery stanzas MUST NOT be considered part of encrypted communication sessions for the purposes of this document, since they perform a "bootstrapping" function that is a prerequisite to encrypted communications. The disco#info request sent from Alice to Bob might look as follows:

Listing 1: Alice Queries Bob for ESession Support via Disco

```
<iq type='get'
    from='alice@example.org/pda'
    to='bob@example.com/laptop'
    id='disco1'>
  <query xmlns='http://jabber.org/protocol/disco#info'/>
</iq>
```

[11] XEP-0030: Service Discovery <http://xmpp.org/extensions/xep-0030.html>.
[12] XEP-0115: Entity Capabilities <http://xmpp.org/extensions/xep-0115.html>.

If Bob sends a disco#info reply and he supports the protocol defined herein, then he MUST include a service discovery feature variable of "http://www.xmpp.org/extensions/xep-0116.html#ns" (see Protocol Namespaces regarding issuance of one or more permanent namespaces).

Listing 2: Bob Returns disco#info Data

```
<iq type='result'
    from='bob@example.com/laptop'
    to='alice@example.org/pda'
    id='disco1'>
  <query xmlns='http://jabber.org/protocol/disco#info'>
    <identity category='client' type='pc'/>
    ...
    <feature var='http://www.xmpp.org/extensions/xep-0116.html#ns'/>
    ...
  </query>
</iq>
```

4 Online ESession Negotiation

4.1 Introduction

The process for establishing a secure session over an insecure transport is essentially a negotiation of various ESession algorithms and other parameters, combined with a translation into XMPP syntax of the SIGMA [13] approach to key exchange (see Cryptographic Design of Encrypted Sessions).
If Alice believes Bob may be online then she SHOULD use the protocol specified in Stanza Session Negotiation (XEP-0155) [14] and in this section to negotiate the ESession options and the keys.
Note: If Alice believes Bob is offline then she SHOULD NOT use this negotiation protocol. However, she MAY use the protocol specified in Offline Encrypted Sessions (XEP-0187) [15] to establish the ESession options and keys. Alternatively, she MAY send stanzas without encryption - in which case her client MUST make absolutely clear to her that the stanzas will not be protected and give her the option not to send the stanzas.

4.2 Three- or Four-Message Negotiation?

This protocol supports both 3- and 4-message key negotiations.
The 3-message SIGMA-I-based key exchange (see useful summary of 3-message negotiation)

[13] SIGMA: the 'SIGn-and-MAc' Approach to Authenticated Diffie-Hellman and its Use in the IKE Protocols (Hugo Krawczyk, June 12 2003) <http://www.ee.technion.ac.il/~{}hugo/sigma.ps>.
[14] XEP-0155: Stanza Session Negotiation <http://xmpp.org/extensions/xep-0155.html>.
[15] XEP-0187: Offline Encrypted Sessions <http://xmpp.org/extensions/xep-0187.html>.

protects the identity of the *initiator* against active attacks. This SHOULD NOT be used to establish client-to-client sessions since the *responder's* identity is not protected against active attacks. However, it SHOULD be used to establish client-to-service (server) sessions, especially where the identity of the service is well known to third parties.
The 4-message SIGMA-R-based key exchange (see useful summary of 4-message negotiation) with hash commitment defends the *responder's* identity against active attacks and facilitates detection of a Man in the Middle attack. It SHOULD be used to establish client-to-client sessions. The 4-message key exchange also includes the following optional security enhancements:

- "Secret Retention": If retained secrets are employed *consistently* during key exchanges, then the Man in the Middle would need to be present for every session, including the first. Sessions remain secure even if a long-lived private signing key is compromised at some time *after* the first session.

- "Short-Authentication-String": Alice and Bob can use SAS once to quickly authenticate each other's public keys. Only a very short human-friendly string needs to be verified out-of-band (e.g. by recognizable voice communication).
  Alternatively, thanks to its protection against Man-in-the-Middle attacks, SAS can be used to eliminate the need to generate, distribute or authenticate any public keys. Note: When this protocol is being used without public keys Alice and Bob SHOULD employ Secret Retention, then the out-of-band verification only needs to be performed once to verify the absence of a Man in the Middle for all sessions (past, present and future). [16]

- "Other Secret": Alice and Bob agree a password out-of-band and their clients use it to authenticate each other every time a session is negotiated.

4.3 ESession Request (Alice)

In addition to the "accept", "security", "logging" and "disclosure" fields (see Back Doors) specified in Stanza Session Negotiation, Alice MUST send to Bob each of the ESession options (see list below) that she is willing to use (see Mandatory to Implement Technologies).

1. The list of Modular Exponential (MODP) group numbers (as specified in RFC 2409 [17] or RFC 3526 [18]) that MAY be used for Diffie-Hellman key exchange in a "modp" field (valid group numbers include 1,2,3,4,5,14,15,16,17 and 18) [19]

[16] This combination of techniques underpins the ZRTP key agreement protocol.
[17] RFC 2409: The Internet Key Exchange (IKE) <http://tools.ietf.org/html/rfc2409>.
[18] RFC 3526: More Modular Exponential (MODP) Diffie-Hellman Groups <http://tools.ietf.org/html/rfc3526>.
[19] Entities SHOULD offer even the lowest MODP groups since some entities are CPU-constrained, and security experts tend to agree that "longer keys do not protect against the most realistic security threats".

2. Symmetric block cipher algorithm names in a ”crypt_algs” field

3. Hash algorithm names in a ”hash_algs” field

4. Compression algorithm names in a ”compress” field

5. Short Authentication String generation algorithm names in a ”sas_algs” field

6. The list of stanza types that MAY be encrypted and decrypted in a ”stanzas” field (message, presence, iq)

7. The different versions of this protocol that are supported in a ”ver” field [20]

8. The minimum number of stanzas that MUST be exchanged before an entity MAY initiate a key re-exchange in a ”rekey_freq” field (1 - every stanza, 100 - every hundred stanzas). Note: This value MUST be less than 2^{32} (see Re-Keying Limits)

9. The methods of identification of the other entity that would be acceptable, and the methods of identification that this entity is prepared to offer (”init_pubkey” and ”resp_pubkey” parameters). The values of these two parameters MUST be either ’key’ (a public signature-verification key wrapped in a <KeyValue/> element as specified in XML Signature [21]), or ’hash’ (a fingerprint of the public key - the result of processing the Normalized <KeyValue/> element with the selected hash algorithm, ”HASH”) [22], or ’none’ (no authentication via public keys). Note: ’none’ MUST NOT be specified with 3-message negotiation.

10. Signature algorithm names (unless the values of the ”init_pubkey” and ”resp_pubkey” parameters are fixed to ’none’)

Each MODP group has at least two well known constants: a large prime number p, and a generator g for a subgroup of GF(p). For *each* MODP group that Alice specifies she MUST perform the following computations to calculate her Diffie-Hellman keys (where n is the number of bits per cipher block for the block cipher algorithm with the largest block size out of those she specified):

[20] This version of this document describes version 1.0 of this protocol.
[21] XML Signature Syntax and Processing <http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/>.
[22] If the entity already possesses one of the other entity’s public keys then it is RECOMMENDED that the fingerprint is requested from the other entity instead of the public key - since this saves bandwidth.

1. Generate: a secret random number x (where 2^{2n-1} < x < p - 1)

2. Calculate: e = g^x mod p

3. Calculate: He = SHA256(e) (see SHA [23])

Note: The last step is not necessary for 3-message negotiations.
Alice MUST send all her calculated values of 'He' (for 4-message negotiations) or 'e' (for 3-message negotiations) to Bob (in a "dhhashes" field in the same order as the associated MODP groups are being sent) Base64 encoded (in accordance with Section 4 of RFC 4648 [24]). She MUST also specify a randomly generated Base64 encoded value of NA (her ESession ID in a "my_nonce" field).
The options in each field MUST appear in Alice's order of preference.

Listing 3: Initiates a 4-message ESession Negotiation

```
<message from='alice@example.org/pda' to='bob@example.com'>
  <thread>ffd7076498744578d10edabfe7f4a866</thread>
  <feature xmlns='http://jabber.org/protocol/feature-neg'>
    <x type='form' xmlns='jabber:x:data'>
      <field type='hidden' var='FORM_TYPE'>
        <value>urn:xmpp:ssn</value>
      </field>
      <field type='boolean' var='accept'>
        <value>1</value>
        <required/>
      </field>
      <field type='list-single' var='logging'>
        <option><value>false</value></option>
        <option><value>true</value></option>
        <required/>
      </field>
      <field type='list-single' var='disclosure'>
        <option><value>never</value></option>
        <required/>
      </field>
      <field type='list-single' var='security'>
        <option><value>e2e</value></option>
        <option><value>c2s</value></option>
        <required/>
      </field>
      <field type='list-single' var='modp'>
```

[23] Secure Hash Standard: Federal Information Processing Standards Publication 180-2 <http://csrc.nist.gov/publications/fips/fips180-2/fips186-2withchangenotice.pdf>.
[24] RFC 4648: The Base16, Base32, and Base64 Data Encodings <http://tools.ietf.org/html/rfc4648>.

```
        <option><value>5</value></option>
        <option><value>14</value></option>
        <option><value>2</value></option>
        <option><value>1</value></option>
      </field>
      <field type='list-single' var='crypt_algs'>
        <option><value>aes256-ctr</value></option>
        <option><value>twofish256-ctr</value></option>
        <option><value>aes128-ctr</value></option>
      </field>
      <field type='list-single' var='hash_algs'>
        <option><value>whirlpool</value></option>
        <option><value>sha256</value></option>
      </field>
      <field type='list-single' var='sign_algs'>
        <option><value>http://www.w3.org/2000/09/xmldsig#rsa-sha256</
            value></option>
        <option><value>http://www.w3.org/2000/09/xmldsig#dsa-sha256</
            value></option>
      </field>
      <field type='list-single' var='compress'>
        <option><value>none</value></option>
      </field>
      <field type='list-multi' var='stanzas'>
        <option><value>message</value></option>
        <option><value>iq</value></option>
        <option><value>presence</value></option>
      </field>
      <field type='list-single' var='init_pubkey'>
        <option><value>key</value></option>
        <option><value>hash</value></option>
        <option><value>none</value></option>
      </field>
      <field type='list-single' var='resp_pubkey'>
        <option><value>key</value></option>
        <option><value>hash</value></option>
        <option><value>none</value></option>
      </field>
      <field type='list-single' var='ver'>
        <option><value>1.3</value></option>
        <option><value>1.2</value></option>
      </field>
      <field type='text-single' var='rekey_freq'>
        <value>1</value>
      </field>
      <field type='hidden' var='my_nonce'>
        <value> ** Alice's␣Base64␣encoded␣ESession␣ID␣**␣</value>
␣␣␣␣␣␣</field>
␣␣␣␣␣␣<field␣type='list-single'␣var='sas_algs'>
```

```
␣␣␣␣␣␣␣␣<option><value>sas28x5</value></option>
␣␣␣␣␣␣</field>
␣␣␣␣␣␣<field␣type='hidden'␣var='dhhashes'>
␣␣␣␣␣␣␣␣<value>␣**␣Base64␣encoded␣value␣of␣He5␣**␣</value>
␣␣␣␣␣␣␣␣<value>␣**␣Base64␣encoded␣value␣of␣He14␣**␣</value>
␣␣␣␣␣␣␣␣<value>␣**␣Base64␣encoded␣value␣of␣He2␣**␣</value>
␣␣␣␣␣␣␣␣<value>␣**␣Base64␣encoded␣value␣of␣He1␣**␣</value>
␣␣␣␣␣␣</field>
␣␣␣␣</x>
␣␣</feature>
␣␣<amp␣xmlns='http://jabber.org/protocol/amp'␣per-hop='true'>
␣␣␣␣<rule␣action='drop'␣condition='deliver'␣value='stored'/>
␣␣</amp>
</message>
```

The first message of a 3-message negotiation is identical except there MUST be no 'sas_algs' field and a 'dhkeys' field MUST be included instead of the 'dhhashes' field:

Listing 4: Alice Initiates a 3-message ESession Negotiation

```
<message from='alice@example.org/pda' to='bob@example.com'>
  <thread>ffd7076498744578d10edabfe7f4a866</thread>
  <feature xmlns='http://jabber.org/protocol/feature-neg'>
    <x type='form' xmlns='jabber:x:data'>
      <field type='hidden' var='FORM_TYPE'>
        <value>urn:xmpp:ssn</value>
      </field>
      ...
      ...
      ...
      <field type='hidden' var='my_nonce'>
        <value> ** Alice's␣Base64␣encoded␣ESession␣ID␣**␣</value>
␣␣␣␣␣␣</field>
␣␣␣␣␣␣<field␣type='hidden'␣var='dhkeys'>
␣␣␣␣␣␣␣␣<value>␣**␣Base64␣encoded␣value␣of␣e5␣**␣</value>
␣␣␣␣␣␣␣␣<value>␣**␣Base64␣encoded␣value␣of␣e14␣**␣</value>
␣␣␣␣␣␣␣␣<value>␣**␣Base64␣encoded␣value␣of␣e2␣**␣</value>
␣␣␣␣␣␣</field>
␣␣␣␣</x>
␣␣</feature>
␣␣<amp␣xmlns='http://jabber.org/protocol/amp'␣per-hop='true'>
␣␣␣␣<rule␣action='drop'␣condition='deliver'␣value='stored'/>
␣␣</amp>
</message>
```

4.4 ESession Rejection (Bob)

If Bob does not want to reveal presence to Alice for whatever reason then Bob SHOULD return no response or error.
If Alice initiated a 3-message negotiation but Bob only supports 4-message negotiations (with Alice) then he SHOULD return a <feature-not-implemented/> error specifying the 'dhkeys' field:

Listing 5: Bob Informs Alice that 3-message Negotiation is Not Supported

```
<message type='error'
         from='bob@example.com/laptop'
         to='alice@example.org/pda'>
  <thread>ffd7076498744578d10edabfe7f4a866</thread>
  <feature xmlns='http://jabber.org/protocol/feature-neg'>
    ...
  </feature>
  <error type='cancel'>
    <feature-not-implemented xmlns='urn:ietf:params:xml:ns:xmpp-
        stanzas'/>
    <feature xmlns='http://jabber.org/protocol/feature-neg'>
      <field var='dhkeys'/>
    </feature>
  </error>
</message>
```

If Bob supports *none* of the options for one or more ESession fields, then he SHOULD return a <not-acceptable/> error specifying the field(s) with unsupported options:

Listing 6: Bob Informs Alice that Her Options are Not Supported

```
<message type='error'
         from='bob@example.com/laptop'
         to='alice@example.org/pda'>
  <thread>ffd7076498744578d10edabfe7f4a866</thread>
  <feature xmlns='http://jabber.org/protocol/feature-neg'>
    ...
  </feature>
  <error type='cancel'>
    <not-acceptable xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
    <feature xmlns='http://jabber.org/protocol/feature-neg'>
      <field var='modp'/>
      <field var='ver'/>
    </feature>
  </error>
</message>
```

Either Bob or Alice MAY attempt to initiate a new ESession after any error during the negotiation process. However, both MUST consider the previous negotiation to have failed and

MUST discard any information learned through the previous negotiation.
If Bob is unwilling to start an ESession, but he *is* ready to initiate a one-to-one stanza session with Alice (see Stanza Session Negotiation), and if Alice included an option for the "security" field with the value "none" or "c2s", then Bob SHOULD accept the stanza session and terminate the ESession negotiation by specifying "none" or "c2s" for the value of the "security" field in his response.

Listing 7: Bob Accepts Stanza Session

```
<message from='bob@example.com/laptop' to='alice@example.org/pda'>
  <thread>ffd7076498744578d10edabfe7f4a866</thread>
  <feature xmlns='http://jabber.org/protocol/feature-neg'>
    <x type='submit' xmlns='jabber:x:data'>
      <field var='FORM_TYPE'>
        <value>urn:xmpp:ssn</value>
      </field>
      <field var='accept'><value>1</value></field>
      <field var='logging'><value>true</value></field>
      <field var='disclosure'><value>never</value></field>
      <field var='security'><value>c2s</value></field>
    </x>
  </feature>
</message>
```

4.5 ESession Response (Bob)

4.5.1 Diffie-Hellman Preparation (Bob)

If Bob supports one or more of each of Alice's ESession options and is willing to start an ESession with Alice, then he MUST select one of the options from each of the ESession fields he received from Alice including one hash algorithm ("HASH"), and one of the MODP groups (see RFC 3766 [25] or RFC 3526 for recommendations regarding balancing the sizes of symmetric cipher blocks and Diffie-Hellman moduli) and Alice's corresponding value of 'He' (for 4-message negotiations) or 'e' (for 3-message negotiations).
Note: Each MODP group has at least two well known constants: a large prime number p, and a generator g for a subgroup of GF(p).
For 3-message negotiations, Bob SHOULD return a <feature-not-implemented/> error unless: 1 < e < p - 1
Bob MUST then perform the following computations (where n is the number of bits per cipher block for the selected block cipher algorithm):

1. Generate a random number NB (his ESession ID)

[25] RFC 3766: Determining Strengths For Public Keys Used For Exchanging Symmetric Keys <http://tools.ietf.org/html/rfc3766>.

2. Generate an n-bit random number CA (the block cipher counter for stanzas sent from Alice to Bob)

3. Set CB = CA XOR 2^{n-1} (where CB is the block counter for stanzas sent from Bob to Alice)

4. Generate a secret random number y (where 2^{2n-1} < y < p - 1)

5. Calculate d = g^y mod p

6. Calculate K = HASH(e^y mod p) (the Diffie-Hellman shared secret)

If this is a 4-message negotiation Bob MUST skip the last step above.

4.5.2 Response Form

Bob SHOULD generate the form that he will send back to Alice, including his responses for all the fields Alice sent him except that he MUST NOT include a 'dhhashes' field.
He MUST set the 'init_pubkey' field to specify what sort of identification he requires from Alice (see ESession Request). He MUST set the value of the 'rekey_freq' field to be less than 2^{32} and greater than or equal to the value specified by Alice. Bob MUST place his Base64 encoded values of NB and d in the 'my_nonce' and 'dhkeys' fields. Note: Bob MUST NOT return Alice's values of NA and e in these fields.
Bob MUST encapsulate the Base64 encoded values of CA and Alice's NA in two new 'counter' and 'nonce' fields and append them to the form.
If this is a 4-message negotiation Bob SHOULD respond to Alice by sending her the form (formB) immediately - there is nothing more for him to do until he receives Alice's next message (i.e. he can skip the following sections). If this is a 3-message negotiation Bob MUST NOT send the form until he has completed the steps in the following sections.

Listing 8: Bob Responds to Alice (4-Message Negotiation only)

```
<message from='bob@example.com/laptop' to='alice@example.org/pda'>
  <thread>ffd7076498744578d10edabfe7f4a866</thread>
  <feature xmlns='http://jabber.org/protocol/feature-neg'>
    <x type='submit' xmlns='jabber:x:data'>
      <field var='FORM_TYPE'>
        <value>urn:xmpp:ssn</value>
      </field>
      <field var='accept'><value>1</value></field>
      <field var='logging'><value>true</value></field>
      <field var='disclosure'><value>never</value></field>
```

```
      <field var='security'><value>e2e</value></field>
      <field var='modp'><value>5</value></field>
      <field var='crypt_algs'><value>aes256-ctr</value></field>
      <field var='hash_algs'><value>sha256</value></field>
      <field var='sign_algs'><value>http://www.w3.org/2000/09/xmldsig#
         rsa-sha256</value></field>
      <field var='compress'><value>none</value></field>
      <field var='stanzas'><value>message</value></field>
      <field var='init_pubkey'><value>hash</value></field>
      <field var='resp_pubkey'><value>hash</value></field>
      <field var='ver'><value>1.3</value></field>
      <field var='rekey_freq'><value>50</value></field>
      <field var='my_nonce'>
        <value> ** Bob's Base64 encoded ESession ID ** </value>
      </field>
      <field var='sas_algs'><value>sas28x5</value></field>
      <field var='dhkeys'>
        <value> ** Base64 encoded value of d ** </value>
      </field>
      <field var='nonce'>
        <value> ** Alice's Base64 encoded ESession ID ** </value>
      </field>
      <field var='counter'>
        <value> ** Base64 encoded block counter ** </value>
      </field>
    </x>
  </feature>
</message>
```

4.5.3 Generating Session Keys

Bob MUST use HMAC with the selected hash algorithm (”HASH”) and the shared secret (”K”) to generate two sets of three keys, one set for each direction of the ESession.
For stanzas that Alice will send to Bob, the keys are calculated as:

1. Encryption key KCA = *HMAC*(HASH, K, ”Initiator Cipher Key”)

2. Integrity key KMA = *HMAC*(HASH, K, ”Initiator MAC Key”)

3. SIGMA key KSA = *HMAC*(HASH, K, ”Initiator SIGMA Key”)

For stanzas that Bob will send to Alice the keys are calculated as:

1. Encryption key KCB = *HMAC*(HASH, K, ”Responder Cipher Key”)

2. Integrity key KMB = *HMAC*(HASH, K, ”Responder MAC Key”)

3. SIGMA key KSB = *HMAC*(HASH, K, ”Responder SIGMA Key”)

Note: As many bits of key data as are needed for each key MUST be taken from the least significant bits of the HMAC output. When negotiating a hash, entities MUST ensure that the hash output is no shorter than the required key data. For algorithms with variable-length keys the maximum length (up to the hash output length) SHOULD be used.
Once the sets of keys have been calculated the value of K MUST be securely destroyed, unless it will be used later to generate the final shared secret (see Generating Bob’s Final Session Keys.

4.5.4 Hiding Bob’s Identity

Bob MUST perform the following steps before he can prove his identity to Alice while protecting it from third parties.

1. Bob MUST select one method of identification from the values in the ’resp_pubkey’ field he received from Alice. If he selects the ’none’ method of identification then he MUST set pubKeyB to a zero length string of characters. Otherwise Bob SHOULD select pubKeyB, which MUST be a Normalized <KeyValue/> element (as specified in XML Signature). This is the public key Alice will use to authenticate his signature with the signature algorithm he selected (”SIGN”).

2. Set formB to be the full Normalized *content* of the reponse data form he generated above (see Response Form). Note: this MUST NOT include ’identity’ or ’mac’ fields.

3. Concatenate Alice’s ESession ID, Bob’s ESession ID, d, pubKeyB and formB, and calculate the HMAC of the resulting byte string using the selected hash algorithm (”HASH”) and the key KSB.

```
macB = HMAC(HASH, KSB, {NA, NB, d, pubKeyB, formB})
```

4. If the value of the ’resp_pubkey’ field that Alice sent Bob was *not* ’none’ then Bob MUST calculate signB, the signature (using his private signature key that corresponds to pubKeyB) of the HMAC result wrapped in a <SignatureValue/> element. Note: signB MUST be calculated as specified in XML Signature except that it is signature of the HMAC result, *not* of a <SignedInfo/> element.

```
if (resp_pubkey != 'none') signB = SIGN(signKeyB, macB)
```

5. If the value of the 'resp_pubkey' field that Alice sent Bob was 'hash' then Bob SHOULD set pubKeyB to the key's fingerprint wrapped in a <fingerprint/> element

```
if (resp_pubkey == 'hash') pubKeyB = '<fingerprint>' + HASH(
    pubKeyB) + '</fingerprint>'
```

6. Encrypt the byte string resulting from the concatenation of pubKeyB and signB (or, if the value of the 'resp_pubkey' field that Alice sent Bob was 'none', encrypt just the HMAC result) with the agreed algorithm ("CIPHER") in counter mode (see Recommendation for Block Cipher Modes of Operation [26]), using the encryption key KCB and block counter CB. Note: CB MUST be incremented by 1 for each encrypted block or partial block (i.e. CB = (CB + 1) mod 2^n, where n is the number of bits per cipher block for the agreed block cipher algorithm).

```
IDB = CIPHER(KCB, CB, {pubKeyB, signB})
```

or

```
IDB = CIPHER(KCB, CB, macB)
```

7. Calculate the HMAC of the encrypted identity (IDB) and the value of Bob's block cipher counter CB *before* the encryption above using the selected hash algorithm ("HASH") and the integrity key KMB.

```
MB = HMAC(HASH, KMB, CB, IDB)
```

4.5.5 Sending the Response (3-message negotiations)

For 3-message negotiations Bob should append the Base64 encoded values of IDB and MB to formB wrapped in 'identity' and 'mac' fields, and send the resulting form to Alice:

Listing 9: Bob Responds to Alice (3-Message Negotiation)

```
<message from='bob@example.com/laptop' to='alice@example.org/pda'>
  <thread>ffd7076498744578d10edabfe7f4a866</thread>
  <feature xmlns='http://jabber.org/protocol/feature-neg'>
```

[26] Recommendation for Block Cipher Modes of Operation: Federal Information Processing Standards Publication 800-38a <http://csrc.nist.gov/publications/nistpubs/800-38a/sp800-38a.pdf>.

```
    <x type='submit' xmlns='jabber:x:data'>
      <field var='FORM_TYPE'>
        <value>urn:xmpp:ssn</value>
      </field>
      ...
      ...
      ...
      <field var='my_nonce'>
        <value> ** Bob's Base64 encoded ESession ID ** </value>
      </field>
      <field var='dhkeys'>
        <value> ** Base64 encoded value of d ** </value>
      </field>
      <field var='nonce'>
        <value> ** Alice's Base64 encoded ESession ID ** </value>
      </field>
      <field var='counter'>
        <value> ** Base64 encoded block counter ** </value>
      </field>
      <field var='identity'>
        <value> ** Encrypted identity ** </value>
      </field>
      <field var='mac'>
        <value> ** Integrity of identity ** </value>
      </field>
    </x>
  </feature>
</message>
```

4.6 ESession Accept (Alice)

4.6.1 Diffie-Hellman Preparation (Alice)

After Alice receives Bob's response, she MUST use the value of d and the ESession options specified in Bob's response to perform the following steps (where p and g are the constants associated with the selected MODP group, HASH is the selected hash algorithm, and n is the number of bits per cipher block for the agreed block cipher algorithm):

1. Verify that the ESession options selected by Bob are acceptable

2. Return a <not-acceptable/> error to Bob unless: 1 < d < p - 1

3. Set CB = CA XOR 2^{n-1} (where CB is the block counter for stanzas sent from Bob to Alice)

4. Select her values of x and e that correspond to the selected MODP group (from all the values of x and e she calculated previously - see ESession Request)

5. Calculate K = HASH(d^x mod p) (the shared secret)

6. Generate the session keys (KCA, KMA, KSA, KCB, KMB and KSB) in exactly the same way as Bob did (see Generating Session Keys). Note: In the case of 4-message negotiation it is only necessary to generate provisory keys for the messages Alice sends to Bob (KCA, KMA, KSA).

If this is a 4-message negotiation then Alice MUST skip the next section and proceed by executing the steps in the Hiding Alice's Identity section.

4.6.2 Verifying Bob's Identity

If this is a 3-message negotiation then Alice MUST also perform the following steps:

1. Calculate the HMAC of the encrypted identity (IDB) and the value of Bob's block cipher counter using HASH and the integrity key KMB.

```
MB = HMAC(HASH, KMB, CB, IDB)
```

2. Return a <feature-not-implemented/> error to Bob unless the value of MB she calculated matches the one she received in the 'mac' field

3. Obtain macB (if the value of the 'resp_pubkey' field she sent to Bob in her ESession Request was 'none') or pubKeyB and signB (otherwise) by decrypting IDB with the agreed symmetric block cipher algorithm ("DECIPHER") in counter mode, using the encryption key KCB and block counter CB. Note: CB MUST be incremented by 1 for each encrypted block or partial block (i.e. CB = (CB + 1) mod 2^n, where n is the number of bits per cipher block for the agreed block cipher algorithm).

```
macB = DECIPHER(KCB, CB, IDB)
```

or

```
{pubKeyB, signB} = DECIPHER(KCB, CB, IDB)
```

4. If the value of the 'resp_pubkey' field that Alice sent Bob was 'none' then Alice MUST set pubKeyB to be a zero length string of characters. Otherwise, if the value was 'hash', then Alice SHOULD change the value of pubKeyB to be her copy of the public key (a Normalized <KeyValue/> element) whose HASH matches the value wrapped in the pubKeyB <fingerprint/> element that she received from Bob.
   Note: If she cannot find a copy of the public key then Alice MUST terminate the ESession. She MAY then request a new ESession with the 'resp_pubkey' field set to 'key' or 'none'.

5. Set the value of formB to be the Normalized *content* of the form she received from Bob without any 'identity' or 'mac' fields.

6. Concatenate Alice's ESession ID, Bob's ESession ID, d, pubKeyB and formB, and calculate the HMAC of the resulting byte string using HASH and the key KSB.

```
macB = HMAC(HASH, KSB, {NA, NB, d, pubKeyB, formB})
```

7. If the value of the 'resp_pubkey' field that Alice sent Bob was 'none' then return a <feature-not-implemented/> error to Bob if the two values of macB she calculated in the steps above do not match.
   If the value of the 'resp_pubkey' field was not 'none', return a <feature-not-implemented/> error unless she can confirm (or has previously confirmed) that pubKeyB really is Bob's public key (see Verifying Keys) and she can use pubKeyB with the selected signature verification algorithm ("VERIFY") to confirm that signB is the signature of the HMAC result (see XML Signature).

```
VERIFY(signB, pubKeyB, macB)
```

4.6.3 Hiding Alice's Identity

Alice MUST then prove her identity to Bob while protecting it from third parties. She MUST perform the steps equivalent to those Bob performed above (see Hiding Bob's Identity for a more detailed description). Alice's calculations are summarised below. Note: When calculating macA pay attention to the order of NB and NA and to the inclusion of formA2.
Note: formA is the full Normalized *content* of the ESession Request data form that Alice sent to Bob at the start of the negotiation, while formA2 is the full Normalized *content* of Alice's session negotiation completion form *excluding* the 'identity' and 'mac' fields (see Sending Alice's Identity below).

```
macA = HMAC(HASH, KSA, {NB, NA, e, pubKeyA, formA, formA2})
```

```
if (init_pubkey != 'none') signA = SIGN(signKeyA, macA)
```

```
if (init_pubkey == 'hash') pubKeyA = HASH(pubKeyA)
```

```
IDA = CIPHER(KCA, CA, {pubKeyA, signA})  OR  IDA = CIPHER(KCA, CA,
    macA)
```

```
MA = HMAC(HASH, KMA, CA, IDA)
```

4.6.4 Sending Alice's Identity

Alice MUST send the Base64 encoded values of NB (wrapped in a 'nonce' field), IDA (wrapped in an 'identity' field) and MA (wrapped in a 'mac' field) to Bob in her session negotiation completion message.
In the case of a 3-message negotiation Alice MAY also send encrypted content (see Stanza Encryption) in the same stanza as the proof of her identity:

Listing 10: Alice Sends Bob Her Identity (3-Message Negotiation)

```
<message from='alice@example.org/pda' to='bob@example.com/laptop'>
  <thread>ffd7076498744578d10edabfe7f4a866</thread>
  <feature xmlns='http://jabber.org/protocol/feature-neg'>
    <x type='result' xmlns='jabber:x:data'>
      <field var='FORM_TYPE'><value>urn:xmpp:ssn</value></field>
      <field var='accept'><value>1</value></field>
      <field var='nonce'><value> ** Bob's Base64 encoded ESession ID
          ** </value></field>
      <field var='identity'><value> ** Encrypted identity ** </value
    ></field>
      <field var='mac'><value> ** Integrity of identity ** </value></
    field>
    </x>
  </feature>
  <c xmlns='http://www.xmpp.org/extensions/xep-0200.html#ns'>
    <data> ** Base64 encoded m_final ** </data>
    <mac> ** Base64 encoded a_mac ** </mac>
  </c>
</message>
```

Note: If Alice also includes a 'terminate' field with its value set to "1" or "true" (see ESession Termination) within the form then the ESession is terminated immediately. Note: This special case, where a single stanza is encrypted and sent in isolation, is equivalent to object encryption (or object signing if no encryption is specified) and offers several significant advantages over non-session approaches - including perfect forward secrecy.
In the case of a 4-message negotiation Alice MUST also include in the data form her Base64

encoded values of e (wrapped in a 'dhkeys' field) and the Base64 encoded HMAC (using HASH and the key NA [27]) of each secret that Alice has retained from her previous session with each of Bob's clients (wrapped in a 'rshashes' field) - see Sending Bob's Identity. Note: Alice MUST also append a few random numbers to the 'rshashes' field to make it difficult for an active attacker to discover if she has communicated with Bob before or how many clients Bob has used to communicate with her.

Listing 11: Alice Sends Bob Her Identity (4-Message Negotiation)

```
<message from='alice@example.org/pda' to='bob@example.com/laptop'>
  <thread>ffd7076498744578d10edabfe7f4a866</thread>
  <feature xmlns='http://jabber.org/protocol/feature-neg'>
    <x type='result' xmlns='jabber:x:data'>
      <field var='FORM_TYPE'><value>urn:xmpp:ssn</value></field>
      <field var='accept'><value>1</value></field>
      <field var='nonce'><value> ** Bob's Base64 encoded ESession ID
          ** </value></field>
      <field type='hidden' var='dhkeys'>
        <value> ** Base64 encoded value of e5 ** </value>
      </field>
      <field type='hidden' var='rshashes'>
        <value> ** Base64 encoded hash of retained secret ** </value>
        <value> ** Base64 encoded hash of retained secret ** </value>
        <value> ** Base64 encoded random value ** </value>
        <value> ** Base64 encoded random value ** </value>
      </field>
      <field var='identity'><value> ** Encrypted identity ** </value
    ></field>
      <field var='mac'><value> ** Integrity of identity ** </value></
    field>
    </x>
  </feature>
</message>
```

4.7 ESession Accept (Bob)

4.7.1 Verifying Alice's Identity

In the case of a 4-message negotiation Bob MUST perform the following four steps:

1. Return a <feature-not-implemented/> error unless SHA256(e) equals 'He', the value he received from Alice in her original session request.

[27] The HMACs of the retained secrets are generated using Alice's unique session nonce to prevent her being identified by her retained secrets (only one secret changes each session, and some might not change very often).

2. Return a <feature-not-implemented/> error unless: 1 < e < p - 1

3. Use the value of e he received from Alice, his secret value of y and their agreed value of p to calculate the value of the Diffie-Hellman shared secret: K = HASH(e^y mod p)

4. Generate Alice's provisory session keys (KCA, KMA, KSA) in exactly the same way as specified for 3-message negotiations in the Generating Session Keys section.

After receiving Alice's identity Bob MUST verify it by performing steps equivalent to those performed by Alice above (see Verifying Bob's Identity for a more detailed description). Some of Bob's calculations are summarised below. Note: When calculating macA pay attention to the order of NB and NA and to the inclusion of formA2.
Note: formA is the full Normalized *content* of the ESession Request data form that Alice sent to Bob at the start of the negotiation, while formA2 is the full Normalized *content* of Alice's session negotiation completion form *excluding* the 'identity' and 'mac' fields (see Sending Alice's Identity).
Note: If Bob sends an error to Alice then he SHOULD ignore any encrypted content he received in the stanza.

```
MA = HMAC(HASH, KMA, CA, IDA)
```

```
macA = DECIPHER(KCA, CA, IDA)  OR  {pubKeyA, signA} = DECIPHER(KCA, CA
    , IDA)
```

```
macA = HMAC(HASH, KSA, {NB, NA, e, pubKeyA, formA, formA2})
```

```
VERIFY(signA, pubKeyA, macA)
```

In the case of a 3-message negotiation, the ESession negotiation is now complete.

4.7.2 Short Authentication String

Note: The steps in this and all the following Online ESession Negotiation sections are only necessary for 4-message negotiations.
Bob and Alice MAY confirm out-of-band that the Short Authentication Strings (SAS) their clients generate for them (using the SAS generation algorithm that they agreed on) are the same. This out-of-band step MAY be performed at any time. However, if either Bob or Alice has not provided a public key, or if either of their public keys has never been authenticated by the other party, then they SHOULD confirm out-of-band that their SAS match as soon as they realise that the two clients have no retained secret in common (see Generating Bob's Final Session Keys below, or Generating Alice's Final Session Keys). However, if it is inconvenient

for Bob and Alice to confirm the match immediately, both clients MAY remember (in a secure way) that a SAS match has not yet been confirmed and remind Bob and Alice at the start of each ESession that they should confirm the SAS match (even if they have a retained secret in common). Their clients should continue to remind them until they either confirm a SAS match, or indicate that security is not important enough for them to bother.

4.7.3 Generating Bob's Final Session Keys

Bob MUST identify the shared retained secret (SRS) by selecting from his client's list of the secrets it retained from previous sessions with Alice's clients (i.e., secrets from sessions where the bareJID was the same as the one Alice is currently using). Note: The list contains the most recent shared secret for each of Alice's clients that she has previously used to negotiate ESessions with the client Bob is currently using.
Bob does this by calculating the HMAC (using HASH and the key NA) of each secret in the list in turn and comparing it with each of the values in the 'rshashes' field he received from Alice (see Sending Alice's Identity). Once he finds a match, and has confirmed that the secret has not expired (because it is older than an implementation-defined period of time), then he has found the SRS.
If Bob cannot find a match, then he SHOULD search through all the retained secrets that have not expired for all the other JIDs his client has communicated with to try to find a match with one of the values in the 'rshashes' field he received from Alice (since she may simply be using a different JID, perhaps in order to protect her identity from third parties). Once he finds a match then he has found the SRS. Note: Resource-constrained implementations MAY make the performance of this second extended search an optional feature.
Bob MUST calculate the final session key by appending to K (the Diffie-Hellman shared secret) the SRS (only if one was found) and then the Other Shared Secret (only if one exists) and then setting K to be the HASH result of the concatenated string of bytes:

```
K = HASH(K | SRS | OSS)
```

Bob MUST now use the new value of K to generate the new session keys (KCA, KMA, KCB, KMB and KSB) in exactly the same way as he does for 3-message negotiations (see Generating Session Keys). These keys will be used to exchange encrypted stanzas. Note: Bob will still need the value of K in the next section.

4.7.4 Sending Bob's Identity

Bob MUST now prove his identity to Alice while protecting it from third parties. He does this in the same way as he does for 3-message negotiations (see Hiding Bob's Identity for a more detailed description) except that, when calculating macB, he MUST include formB2:

```
macB = HMAC(HASH, KSB, {NA, NB, d, pubKeyB, formB, formB2})
```

Note: formB2 is the full Normalized *content* of Bob's session negotiation completion form *excluding* the 'identity' and 'mac' fields (see below).
Bob MUST send Alice the Base64 encoded value of the HMAC (using HASH and the key SRS) of the string "Shared Retained Secret" (wrapped in an 'srshash' field). If no SRS was found then he MUST use a random number instead. [28]

```
HMAC(HASH, SRS, "Shared Retained Secret")
```

Bob MUST also include in the data form the Base64 encoded values of NA, and IDB and MB (that he just calculated). Note: He MAY also send encrypted content (see Stanza Encryption) in the same stanza.

Listing 12: Bob Sends Alice His Identity

```
<message from='bob@example.com/laptop' to='alice@example.org/pda'>
  <thread>ffd7076498744578d10edabfe7f4a866</thread>
  <init xmlns='http://www.xmpp.org/extensions/xep-0116.html#ns-init'>
    <x type='result' xmlns='jabber:x:data'>
      <field var='FORM_TYPE'><value>urn:xmpp:ssn</value></field>
      <field var='nonce'><value> ** Alice's Base64 encoded ESession ID
          ** </value></field>
      <field var='srshash'><value> ** HMAC with shared retained secret
        ** </value></field>
      <field var='identity'><value> ** Encrypted identity ** </value
        ></field>
      <field var='mac'><value> ** Integrity of identity ** </value></
        field>
    </x>
  </init>
</message>
```

Finally, Bob MUST destroy all his copies of the old retained secret (SRS) he was keeping for Alice's client, and calculate a new retained secret for this session:

```
HMAC(HASH, K, "New Retained Secret")
```

Bob MUST *securely* store the new value along with the retained secrets his client shares with Alice's other clients.
Bob's value of K MUST now be securely destroyed.

[28] Bob always sends a value in the 'srshash' field to prevent an attacker learning that the session is not protected by a retained secret.

4.8 Final Steps (Alice)

4.8.1 Generating Alice's Final Session Keys

Alice MUST identify the shared retained secret (SRS) by selecting from her client's list of the secrets it retained from sessions with Bob's clients (the most recent secret for each of the clients he has used to negotiate ESessions with Alice's client).
Alice does this by using each secret in the list in turn as the key to calculate the HMAC (with HASH) of the string "Shared Retained Secret", and comparing the calculated value with the value in the 'srshash' field she received from Bob (see Sending Bob's Identity). Once she finds a match, and has confirmed that the secret has not expired (because it is older than an implementation-defined period of time), then she has found the SRS.
Alice MUST calculate the final session key by appending to K (the Diffie-Hellman shared secret) the SRS (only if one was found) and then the Other Shared Secret (only if one exists) and then setting K to be the HASH result of the concatenated string of bytes:

```
K = HASH(K | SRS | OSS)
```

Alice MUST destroy all her copies of SRS (the retained secret she was keeping for Bob's client), calculate a new retained secret for this session (see below) and *securely* store the new value along with the other retained secrets her client shares with Bob's clients:

```
HMAC(HASH, K, "New Retained Secret")
```

Alice MUST now use the new value of K to generate the new session keys (KCA, KMA, KCB, KMB and KSB) in exactly the same way as Bob did (see Generating Session Keys). These keys will be used to exchange encrypted stanzas.

4.8.2 Verifying Bob's Identity

Finally, Alice MUST verify the identity she received from Bob. She does this in the same way as she does for 3-message negotiations Verifying Bob's Identity above. Note: If Alice discovers an error then she SHOULD ignore any encrypted content she received in the stanza.
Once ESession negotiation is complete, Alice and Bob MUST exchange only encrypted forms of the one-to-one stanza types they agreed upon (e.g., <message/> and <iq/> stanzas) within the session.

5 ESession Termination

Either entity MAY terminate an ESession at any time. Entities MUST terminate all open ESessions before they go offline. To terminate an ESession Alice MUST send an encrypted stanza (see Stanza Encryption) to Bob including within the encrypted XML of the <data/> element

a stanza session negotiation form with a ”terminate” field (as specified in the Termination section of Stanza Session Negotiation). Note: She MAY publish old values of KMA and/or KMB within her termination stanza as long as she is sure all the stanzas that MAY use the old values have been received and validated (see Stanza Encryption). She MUST then securely destroy all keys associated with the ESession.

Listing 13: Alice Terminates an ESession

```
<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 encrypted terminate form ** </data>
    <old> ** Base64 encoded old MAC key ** </old>
    <mac> ** Base64 encoded a_mac ** </mac>
  </c>
</message>
```

When Bob receives a termination stanza he MUST verify the MAC (to be sure he received all the stanzas Alice sent him during the ESession) and immediately send an encrypted termination acknowledgement form (as specified in the Termination section of Stanza Session Negotiation) back to Alice. Note: He MAY publish *any* old values of KMA or KMB within the acknowledgement stanza. He MUST then securely destroy all keys associated with the ESession.

Listing 14: Bob Acknowledges ESession Termination

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

When Alice receives the stanza she MUST verify the MAC to be sure she received all the stanzas Bob sent her during the ESession. Once an entity has sent a termination or termination acknowledgement stanza it MUST NOT send another stanza within the ESession.

6 Implementation Notes

6.1 Multiple-Precision Integers

Before Base-64 encoding, hashing or HMACing an arbitrary-length integer, the integer MUST first be converted to a ”big endian” bitstring. The bitstring MUST then be padded with leading

zero bits so that there are an integral number of octets. Finally, if the integer is not of fixed bit-length (i.e. not a hash or HMAC result) and the bitstring contains leading octets that are zero, these MUST be removed (so the high-order octet is non-zero).

6.2 XML Normalization

Before the signature or MAC of a block of XML is generated or verified, all character data *between* all elements MUST be removed and the XML MUST be converted to canonical form (see Canonical XML [29]).
All the XML this protocol requires to be signed or MACed is very simple, so in this case, canonicalization SHOULD only require the following changes:

- Set attribute value delimiters to single quotation marks (i.e. simply replace all single quotes in the serialized XML with double quotes)
- Impose lexicographic order on the attributes of ”field” elements (i.e. ensure ”type” is before ”var”)

Implementations MAY conceivably also need to make the following changes. Note: Empty elements and special characters SHOULD NOT appear in the signed or MACed XML specified in this protocol.

- Ensure there are no character references
- Convert empty elements to start-end tag pairs
- Ensure there is no whitespace except for single spaces before attributes
- Ensure there are no ”xmlns” attributes or namespace prefixes.

7 Security Considerations

7.1 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 [30]).

7.2 Replay Attacks

Alice and Bob MUST ensure that the value of e or d they provide when negotiating each online ESession is unique. This prevents complete online ESessions being replayed.

[29] Canonical XML 1.0 <http://www.w3.org/TR/xml-c14n>.
[30] RFC 1750: Randomness Recommendations for Security <http://tools.ietf.org/html/rfc1750>.

7.3 Verifying Keys

The trust system outlined in this document is based on Alice trusting that the public key presented by Bob (wrapped in a <KeyValue/> element) is *actually* Bob's key (and vice versa). Determining this trust may be done in a variety of ways depending on the entities' support for different public key (certificate) formats, signing algorithms and signing authorities. For instance, if Bob publishes a PGP/GPG public key, Alice MAY verify that his key is signed by another key that she knows to be good. Or, if Bob provides an X.509 certificate, she MAY check that his key has been signed by a Certificate Authority that she trusts.
When trust cannot be achieved automatically, methods that are not transparent to the users may be employed. The out-of-band Short Authentication String mechanism described in this document is an easy way for people to do that. Alternatively, Bob could communicate the *full* SHA-256 fingerprint of his public key to Alice via secure out-of-band communication (e.g. face-to-face). This would enable Alice to confirm that the public key she receives in-band is valid. Note: Since very few people bother to (consistently) verify SAS or fingerprints, entities SHOULD protect against 'man-in-the-middle' attacks using retained secrets and/or other secrets. In the case of retained secrets entities SHOULD remember whether or not the whole chain of retained secrets (and the associated sessions) has ever been validated by the user verifying a SAS.
Note: If no keys are acceptable to Alice (because Alice has never verified any of the keys, and because either the keys are not signed, or Alice does not support the signature algorithms of the keys, or she cannot parse the certificate formats, or she does not recognise the authorities that signed the keys) then, although the ESession can still be encrypted, she cannot be sure she is communicating with Bob.

7.4 Key Associations

An entity SHOULD remember the fingerprints of all public keys it receives, and remember whether or not they have been validated by the user (see Verifying Keys).
Entities MUST associate one or more JIDs with each public key fingerprint that they store, and alert their users immediately if another JID presents the same public key. This is necessary since if Bob already has fingerprints from Alice and Mallory, and Bob's client presents only the JID (or a name associated with the JID) to Bob, then Mallory could use his own public key (that is trusted by Bob) and pretend to be Alice simply by exchanging stanzas with Bob using Alice's JID.
If a JID for which a key has previously been stored attempts to establish an ESession using a public key with a different fingerprint (or no key at all) then the entity MUST alert its user.
Since Alice MAY use many different JIDs to talk to Bob, but always identify herself to him with the same public key, Entities SHOULD associate a "petname" with each public key fingerprint they store. Entities MUST present any public key petnames clearly to their users, and more prominently than any petname or nickname associated with the JID or the JID itself.

7.5 Unencrypted ESessions

Organisations with full disclosure policies may require entities to disable encryption (see Back Doors) to enable the logging of all messages on their server. Unencrypted ESessions meet all the Security Requirements (see Cryptographic Design of Encrypted Sessions) except for Confidentiality. Unencrypted ESessions enable Alice to to confirm *securely* with Bob that both client-server connections are secure. i.e. that the value of the 'security' option (as specified in Stanza Session Negotiation) has not been tampered with.

7.6 Back Doors

The authors and the XSF would like to discourage the deliberate inclusion of "back doors" in implementations of this protocol. However, we recognize that some organizations must monitor stanza sessions or record stanza sessions in decryptable form for legal compliance reasons, or may choose to monitor stanza sessions for quality assurance purposes. In these cases it is important to inform the other entity of the (potential for) disclosure before starting the ESession (if only to maintain public confidence in this protocol).
Both implementations MUST immediately and clearly inform their users if the negotiated value of the 'disclose' field is not 'never'.
Before disclosing any stanza session, an entity SHOULD either negotiate the value of the 'disclose' field to be 'enabled' or terminate the negotiation unsuccessfully. It MUST NOT negotiate the value of the 'disclose' field to be 'disabled' unless it would be illegal for it to divulge the disclosure to the other entity.
In any case an implementation MUST NOT negotiate the value of the 'disclose' field to be 'never' unless it implements no feature or mechanism (not even a disabled feature or mechanism) that could be used directly or indirectly to divulge to *any* third-party either the identites of the participants, or the keys, or the content of *any* ESession (or information that could be used to recover any of those items). If an implementation deliberately fails to observe this last point (or fails to correct an accidental back door) then it is not compliant with this protocol and MUST NOT either claim or imply any compliance with this protocol or any of the other protocols developed by the authors or the XSF. In this case the authors and the XSF reserve all rights regarding the names of the protocols.
The expectation is that this legal requirement will persuade many implementors either to tell the users of their products that a back door exists, or not to implement a back door at all (if, once informed, the market demands that).

7.7 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 ESession 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.

8 Mandatory to Implement Technologies

An implementation of ESession MUST support the Diffie-Hellman Key Agreement and HMAC (see Section 2 of RFC 2104 [31]) algorithms. Note: Some of the parameter names mentioned below are related to secure shell; see SSH Transport Layer Encryption Modes for block cipher algorithm details; see the IANA Secure Shell Protocol Parameters Registry [32] for some of the other names.

8.1 Block Cipher Algorithms

An implementation of ESession MUST support the following block cipher algorithm:

- aes128-ctr (see AES [33])

The block length of an block cipher algorithm's cipher SHOULD be at least 128 bits. An implementation of ESession MAY also support the following block cipher algorithms:

- aes256-ctr
- aes192-ctr
- twofish256-ctr (see Twofish [34])
- twofish192-ctr
- twofish128-ctr
- serpent256-ctr (see Serpent [35])
- serpent192-ctr
- serpent128-ctr
- none (no encryption, only signing)

[31] RFC 2104: HMAC: Keyed-Hashing for Message Authentication <http://tools.ietf.org/html/rfc2104>.
[32] IANA registry of parameters related to secure shell <http://www.iana.org/assignments/ssh-parameters>.
[33] Advanced Encryption Standard: Federal Information Processing Standards Publication 197 <http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf>.
[34] The Twofish Block Cipher <http://www.schneier.com/twofish.html>.
[35] The Serpent Block Cipher <http://www.cl.cam.ac.uk/~{}rja14/serpent.html>.

8.2 Key Signing Algorithms

An implementation of ESession MUST support the following signing algorithm:

- http://www.w3.org/2000/09/xmldsig#rsa-sha256
  (the same apath except that it uses SHA256 instead of SHA1, see XML Signature)

An implementation of ESession SHOULD also support at least the following signing algorithm:

- http://www.w3.org/2000/09/xmldsig#dsa-sha256
  (the same apath except that it uses SHA256 instead of SHA1, see XML Signature)

8.3 Public Signature-Verification-Key Formats

<KeyValue/> elements (as specified in XML Signature) may contain different public key formats. An implementation of ESession MUST support the following format:

- <RSAKeyValue/>

An implementation of ESession SHOULD also support the following public key format:

- <DSAKeyValue/>

8.4 Hash Algorithms

An implementation of ESession MUST support the following hash algorithm:

- sha256 (see Secure Hash Standard)

An implementation of ESession SHOULD also support at least the following hash algorithm (sha1 and md5 are broken and therefore NOT RECOMMENDED):

- whirlpool (see Whirlpool [36])

8.5 Short Authentication String Generation Algorithms

An implementation of ESession MUST support the following SAS generation algorithm:

- sas28x5 (see The sas28x5 SAS Algorithm)

[36] The Whirlpool Hash Function <http://paginas.terra.com.br/informatica/paulobarreto/WhirlpoolPage.html>.

8.6 Compression Algorithms

An implementation of ESession MUST support the following compression algorithm:

- none (no compression, the output MUST be the same as the input)

Support for other algorithms is NOT RECOMMENDED since compression partially defeats the Repudiability requirement of this document by making it more difficult for a third party (with some knowledge of the plaintext) to modify a transcript of an encrypted session in a meaningful way. However, encrypted content is pseudo-random and cannot be compressed, so, in those cases where bandwidth is severely constrained, an implementation of ESession MAY support the following algorithms to compress content before it is encrypted:

- lzw (see Standard ECMA-151 [37])
- zlib (see RFC 1950 [38])

9 The sas28x5 SAS Algorithm

Given the multi-precision integer MA (a big-endian byte array), the UTF-8 byte string formB (see Hiding Bob's Identity) and the hash function "HASH", the following steps can be used to calculate a 5-character SAS with over 16 million possible values that is easy to read and communicate verbally:

1. Concatenate MA, formB and the UTF-8 byte string "Short Authentication String" into a string of bytes

2. Calculate the least significant 24-bits of the HASH of the string

3. Convert the 24-bit integer into a base-28 [39] 5-character string using the following "digits": acdefghikmopqruvwxy123456789 (the digits have values 0-27)

[37] Standard ECMA-151: Data Compression for Information Interchange - Adaptive Coding with Embedded Dictionary - DLCZ Algorithm <http://www.ecma-international.org/publications/standards/Ecma-151.htm>.
[38] RFC 1950: ZLIB Compressed Data Format Specification version 3.3 <http://tools.ietf.org/html/rfc1950>.
[39] Base-28 was used instead of Base-36 because some characters are often confused when communicated verbally (n, s, b, t, z, j), and because zero is often read as the letter 'o', and the letter 'l' is often read as the number '1'.

10 IANA Considerations

This document requires no interaction with the Internet Assigned Numbers Authority (IANA) [40].

11 XMPP Registrar Considerations

11.1 Protocol Namespaces

Until this specification advances to a status of Draft, its associated namespaces shall be "http://www.xmpp.org/extensions/xep-0116.html#ns" and "http://www.xmpp.org/extensions/xep-0116.html#ns-init"; upon advancement of this specification, the XMPP Registrar [41] shall issue permanent namespaces in accordance with the process defined in Section 4 of XMPP Registrar Function (XEP-0053) [42].

11.2 Field Standardization

Field Standardization for Data Forms (XEP-0068) [43] defines a process for standardizing the fields used within Data Forms qualified by a particular namespace. The following fields shall be registered for use in *both* Encrypted Session Negotiation and Stanza Session Negotiation:

```
<form_type>
  <name>http://www.xmpp.org/extensions/xep-0116.html#ns</name>
  <doc>XEP-0116</doc>
  <desc>ESession negotiation forms</desc>
  <field
      var='compress'
      type='list-single'
      label='Compression algorithm options'/>
  <field
      var='counter'
      type='hidden'
      label='Initial block counter'/>
  <field
      var='crypt_algs'
      type='list-single'
```

[40] 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/>.

[41] 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 <http://xmpp.org/registrar/>.

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

[43] XEP-0068: Field Data Standardization for Data Forms <http://xmpp.org/extensions/xep-0068.html>.

```
      label='Symmetric block cipher options'/>
  <field
      var='dhhashes'
      type='hidden'
      label='Hashes of Diffie-Hellman public keys'/>
  <field
      var='dhkeys'
      type='hidden'
      label='Diffie-Hellman public keys'/>
  <field
      var='expires'
      type='hidden'
      label='Expiry time of offline ESession options'/>
  <field
      var='hash_algs'
      type='list-single'
      label='Hash algorithm options'/>
  <field
      var='match_resource'
      type='text-single'
      label='Target resource for offline ESessions'/>
  <field
      var='modp'
      type='list-single'
      label='MODP group number'/>
  <field
      var='my_nonce'
      type='hidden'
      label='ESession ID of Sender'/>
  <field
      var='nonce'
      type='hidden'
      label='ESession ID of Receiver'/>
  <field
      var='init_pubkey'
      type='list-single'
      label='Initiator public key required'>
    <option label='No Key'>
      <value>none</value>
    </option>
    <option label='Full Key'>
      <value>key</value>
    </option>
    <option label='Key Fingerprint'>
      <value>hash</value>
    </option>
  </field>
  <field
      var='resp_pubkey'
```

```
      type='list-single'
      label='Responder public key required'>
    <option label='No Key'>
      <value>none</value>
    </option>
    <option label='Full Key'>
      <value>key</value>
    </option>
    <option label='Key Fingerprint'>
      <value>hash</value>
    </option>
  </field>
  <field
      var='rekey_freq'
      type='text-single'
      label='Minimum number of stanzas between key exchanges'/>
  <field
      var='rshashes'
      type='hidden'
      label='Hashes of retained secrets'/>
  <field
      var='sas_algs'
      type='list-single'
      label='SAS generation algorithm options'/>
  <field
      var='sign_algs'
      type='list-single'
      label='Signature algorithm options'/>
  <field
      var='signs'
      type='list-single'
      label='Data form signatures'/>
  <field
      var='srshash'
      type='hidden'
      label='Hash of shared retained secret'/>
  <field
      var='stanzas'
      type='list-multi'
      label='Stanzas types to encrypt'/>
    <option>
      <value>message</value>
    </option>
    <option>
      <value>presence</value>
    </option>
    <option>
      <value>iq</value>
    </option>
```

```
  </field>
  <field
      var='ver'
      type='list-single'
      label='Supported versions of ESessions'>
    <option>
      <value>1.0</value>
    </option>
  </field>
</form_type>

<form_type>
  <name>urn:xmpp:ssn</name>
  <doc>XEP-0155</doc>
  ...
</form_type>
```

12 Open Issues

12.1 To Think About

1. What challenges exist to make the OTR Gaim Plugin use this protocol natively when talking to XMPP entities? Can these be mitigated by 'non-critical' protocol changes?

2. Would anything in this protocol (e.g., its dependency on in-order stanza delivery) prevent an XMPP entity using it to exchange encrypted messages and presence with a user of a non-XMPP messaging system, assuming that the gateway both supports this protocol and is compatible with a purpose-built security plugin on the other user's client (e.g. a Gaim plugin connects to the gateway via a non-XMPP network)?

3. Could use Flexible Offline Message Retrieval (XEP-0013) [44] (FOMR) instead of AMP to prevent any offline ESessions Bob can't decrypt being delivered to him. (Each <item/> that corresponds to an ESession message would have to contain a <ESessionID/> child, to allow Bob to discover which of his stored values of y was used to encrypt the message.)

12.2 To Do

1. Ask the authors of AMP to explain how to achieve the match_resource functionality specified in XEP-0187.

2. Add non-repudiable signing option

3. Perhaps the document needs to specify more carefully how block counters are handled between messages, especially in the event of partial blocks?

[44] XEP-0013: Flexible Offline Message Retrieval <http://xmpp.org/extensions/xep-0013.html>.

4. Give examples of specific errors and discuss error scenarios throughout document (e.g., what should Bob do if he is not offline and he receives an offline key exchange stanza?).

5. Define an *optional* protocol that would allow Alice to store values of NA and x (and the PKIDs she trusts) 'securely' on her own server (before she goes offline).