XEP-0217: Simplified Encrypted Session Negotiation

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

2007-05-30
Version 0.1

Status	Type	Short Name
Deferred	Standards Track	TO BE ASSIGNED

This document specifies a minimal subset of the Encrypted Session Negotiation protocol sufficent for negotiating an end-to-end encrypted session.

Legal

Copyright

Permissions

Warranty

Liability

Conformance

Contents

1 Introduction

Encrypted Session Negotiation (XEP-0116) [1] is a fully-fledged protocol that supports multiple different end-to-end encryption functionalities and scenarios. The protocol is as simple as possible given its feature set. However, the work involved to implement it may be reduced by removing support for several of the optional features, including alternative algorithms, 3-message exchange, public keys, repudiation and key re-exchange.
The minimal subset of the protocol defined in this document is designed to be relatively simple to implement while offering full compatibility with implementations of the fully-fledged protocol. The existence of this subset enables developers to produce working code before they have finished implementing the full protocol.
The requirements and the consequent cryptographic design that underpin this protocol are described in Requirements for Encrypted Sessions (XEP-0210) [2] and Cryptographic Design of Encrypted Sessions (XEP-0188) [3]. The basic concept is that of an encrypted session which acts as a secure tunnel between two endpoints. The protocol specified in Stanza Session Negotiation (XEP-0155) [4] and in this document is used to negotiate the encryption keys and establish the tunnel. Thereafter the content of each one-to-one XML stanza exchanged between the endpoints during the session will be encrypted and transmitted within a "wrapper" stanza using Stanza Encryption (XEP-0200) [5].
The cut-down protocol described here is a 4-message key exchange (see useful summary of 4-message negotiation) with short-authentication-string (SAS), hash commitment and optional retained secrets. It avoids using public keys - thus protecting the identity of *both* participants against active attacks from third parties.
Note: This protocol requires that both entities are online. An entity MAY use the protocol specified in Offline Encrypted Sessions (XEP-0187) [6] if it believes the other entity is offline.

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>.

[1] XEP-0116: Encrypted Session Negotiation <https://xmpp.org/extensions/xep-0116.html>.
[2] XEP-0210: Requirements for Encrypted Sessions <https://xmpp.org/extensions/xep-0210.html>.
[3] XEP-0188: Cryptographic Design of Encrypted Sessions <https://xmpp.org/extensions/xep-0188.html>.
[4] XEP-0155: Stanza Session Negotiation <https://xmpp.org/extensions/xep-0155.html>.
[5] XEP-0200: Stanza Encryption <https://xmpp.org/extensions/xep-0200.html>.
[6] XEP-0187: Offline Encrypted Sessions <https://xmpp.org/extensions/xep-0187.html>.

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) [7] or the presence-based profile of XEP-0030 specified in Entity Capabilities (XEP-0115) [8].
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>
```

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”.

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>
```

[7] XEP-0030: Service Discovery <https://xmpp.org/extensions/xep-0030.html>.
[8] XEP-0115: Entity Capabilities <https://xmpp.org/extensions/xep-0115.html>.

4 Online ESession Negotiation

4.1 ESession Request (Alice)

In addition to the ”accept”, ”security”, ”otr” 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.

- The list of Modular Exponential (MODP) group numbers (as specified in RFC 2409 [9] or RFC 3526 [10]) 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) [11]
- The list of stanza types that MAY be encrypted and decrypted in a ”stanzas” field (message, presence, iq)
- The different versions of the Encrypted Session Negotiation protocol that are supported in a ”ver” field

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 128 - i.e. the number of bits per cipher block for the AES-128 block cipher algorithm):

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 [12])

Alice MUST send all her calculated values of ’He’ 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 [13]). She MUST also specify a randomly generated Base64 encoded value of NA (her ESession ID in a ”my_nonce” field).

[9] RFC 2409: The Internet Key Exchange (IKE) <http://tools.ietf.org/html/rfc2409>.
[10] RFC 3526: More Modular Exponential (MODP) Diffie-Hellman Groups <http://tools.ietf.org/html/rfc3526>.
[11] 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”.
[12] Secure Hash Standard: Federal Information Processing Standards Publication 180-2 <http://csrc.nist.gov/publications/fips/fips180-2/fips186-2withchangenotice.pdf>.
[13] RFC 4648: The Base16, Base32, and Base64 Data Encodings <http://tools.ietf.org/html/rfc4648>.

The form SHOULD NOT include a ”sign_algs” field. However, to ensure compatibility with entities that support the full Encrypted Session Negotiation protocol, the form SHOULD include the following fixed values in hidden fields:

Field	Value
crypt_algs	”aes128-ctr”
hash_algs	”sha256”
compress	”none”
init_pubkey	”none”
resp_pubkey	”none”
rekey_freq	”4294967295”
sas_algs	”sas28x5”

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='otr'>
        <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'>
        <option><value>5</value></option>
```

```
        <option><value>14</value></option>
        <option><value>2</value></option>
        <option><value>1</value></option>
      </field>
      <field type='hidden' var='crypt_algs'>
        <value>aes128-ctr</value>
      </field>
      <field type='hidden' var='hash_algs'>
        <value>sha256</value>
      </field>
      <field type='hidden' var='compress'>
        <value>none</value>
      </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='hidden' var='init_pubkey'>
        <value>none</value>
      </field>
      <field type='hidden' var='resp_pubkey'>
        <value>none</value>
      </field>
      <field type='list-single' var='ver'>
        <option><value>1.3</value></option>
        <option><value>1.2</value></option>
      </field>
      <field type='hidden' var='rekey_freq'>
        <value>4294967295</value>
      </field>
      <field type='hidden' var='my_nonce'>
        <value> ** Alice's␣Base64␣encoded␣ESession␣ID␣**␣</value>
␣␣␣␣␣␣</field>
␣␣␣␣␣␣<field␣type='hidden'␣var='sas_algs'>
␣␣␣␣␣␣␣␣<value>sas28x5</value>
␣␣␣␣␣␣</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>
```

4.2 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 Bob finds that one or more of the fields (other than the "rekey_freq" field) listed in the Fixed Parameters table (see ESession Request) does not include the value included in the table (or an <option/> element containing the value), or if Bob supports *none* of the options for one or more of the negotiable ESession fields ("modp", "stanzas", "ver"), then he SHOULD also return a <not-acceptable/> error specifying the field(s) with unsupported options:

Listing 4: 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 5: 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='otr'><value>true</value></field>
```

```
      <field var='disclosure'><value>never</value></field>
      <field var='security'><value>c2s</value></field>
    </x>
  </feature>
</message>
```

4.3 ESession Response (Bob)

4.3.1 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 negotiable ESession fields ("modp", "stanzas", "ver") he received from Alice, including one of the MODP groups and Alice's corresponding value of 'He'. Note: MODP group 14, with its 2048-bit modulus, could be considered a good match for AES-128, however CPU-constrained implementations MAY select a smaller group.
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).
Bob MUST then perform the following computations (where n is 128, the number of bits per cipher block for AES-128):

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

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

4.3.2 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. The form SHOULD include the fields and associated values listed in the Fixed Parameters table (see ESession Request).
He MUST place his Base64 encoded values of NB and d in the 'my_nonce' and 'dhkeys' fields.
Note: Bob MUST NOT return Alice's value of NA in the 'my_nonce' field.

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.
Bob SHOULD respond to Alice by sending her the form (formB).

Listing 6: Bob Responds to Alice

```
<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='otr'><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>aes128-ctr</value></field>
      <field var='hash_algs'><value>sha256</value></field>
      <field var='compress'><value>none</value></field>
      <field var='stanzas'><value>message</value></field>
      <field var='init_pubkey'><value>none</value></field>
      <field var='resp_pubkey'><value>none</value></field>
      <field var='ver'><value>1.3</value></field>
      <field var='rekey_freq'><value>4294967295</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>
```

4.4 ESession Accept (Alice)

4.4.1 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, and n is 128 - the number of bits per cipher block):

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 = SHA256(d^x mod p) (the shared secret)
6. Generate provisory session keys *only* for the messages Alice sends to Bob (KCA, KMA, KSA) - see the next section, Generating Session Keys.

4.4.2 Generating Session Keys

Alice MUST use HMAC with SHA256 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*(SHA256, K, "Initiator Cipher Key")
2. Integrity key KMA = *HMAC*(SHA256, K, "Initiator MAC Key")
3. SIGMA key KSA = *HMAC*(SHA256, K, "Initiator SIGMA Key")

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

1. Encryption key KCB = *HMAC*(SHA256, K, "Responder Cipher Key")

2. Integrity key KMB = *HMAC*(SHA256, K, "Responder MAC Key")

3. SIGMA key KSB = *HMAC*(SHA256, K, "Responder SIGMA Key")

Note: Only the 128 *least* significant bits of the HMAC output must be used for each key.
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).

4.4.3 Hiding Alice's Identity

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

1. Set formA to be the full Normalized *content* of the ESession Request data form that Alice sent to Bob at the start of the negotiation.

2. Set formA2 to be the full Normalized *content* of Alice's session negotiation completion form *excluding* the 'identity' and 'mac' fields (see Sending Alice's Identity below).

3. Concatenate Bob's ESession ID, Alice's ESession ID, e, formA and formA2, and calculate the HMAC of the resulting byte string using SHA256 and the key KSA.

```
macA = HMAC(SHA256, KSA, {NB, NA, e, formA, formA2})
```

4. Encrypt the HMAC result with AES-128 in counter mode (see Recommendation for Block Cipher Modes of Operation [14]), using the encryption key KCA and block counter CA. 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 128 - the number of bits per cipher block).

```
IDA = CIPHER(KCA, CA, macA)
```

5. Calculate the HMAC of the encrypted identity (IDA) and the value of Bob's block cipher counter CA *before* the encryption above using SHA256 and the integrity key KMA.

[14] 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>.

```
MA = HMAC(SHA256, KMA, CA, IDA)
```

4.4.4 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.
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 SHA256 and the key NA [15]) of each secret (if any) 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 7: Alice Sends Bob Her Identity

```
<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>
```

[15] 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).

4.5 ESession Accept (Bob)

4.5.1 Generating Provisory Session Keys (Bob)

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.

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 = SHA256(e^y mod p)

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

4.5.2 Verifying Alice's Identity

Bob MUST also perform the following steps:

1. Calculate the HMAC of the encrypted identity (IDA) and the value of Alice's block cipher counter using SHA256 and the integrity key KMA.

```
MA = HMAC(SHA256, KMA, CA, IDA)
```

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

3. Obtain macB by decrypting IDA with the AES-128 symmetric block cipher algorithm ("DECIPHER") in counter mode, using the encryption key KCA and block counter CA. 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 128 - the number of bits per cipher block).

```
macA = DECIPHER(KCA, CA, IDA)
```

4. Set the value of formA to be the full Normalized *content* of the ESession Request data form that Alice sent to Bob at the start of the negotiation.

5. Set the value of formA2 to be the full Normalized *content* of Alice's session negotiation completion form *excluding* the 'identity' and 'mac' fields (see Sending Alice's Identity).

6. Concatenate Bob's ESession ID, Alice's ESession ID, e, formA and formA2, and calculate the HMAC of the resulting byte string using SHA256 and the key KSA.

```
macA = HMAC(SHA256, KSA, {NB, NA, e, formA, formA2})
```

7. Return a <feature-not-implemented/> error to Alice if the two values of macA he calculated in the steps above do not match.

4.5.3 Short Authentication String

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, 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.

4.5.4 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 (if any) 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 SHA256 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 (if any) 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 SHA256 result of the concatenated string of bytes:

```
K = SHA256(K | SRS | OSS)
```

Bob MUST now use the new value of K to generate the new session keys (KCA, KMA, KCB, KMB and KSB) - 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.

4.5.5 Sending Bob's Identity

Bob MUST now prove his identity to Alice while protecting it from third parties. He MUST perform the steps equivalent to those Alice performed above (see Hiding Alice's Identity for a more detailed description). Bob's calculations are summarised below. Note: When calculating macB pay attention to the order of NA and NB.
Note: formB is the full Normalized *content* of the reponse data form he generated above (see Response Form), and formB2 is the full Normalized *content* of Bob's session negotiation completion form *excluding* the 'identity' and 'mac' fields (see below).

```
macB = HMAC(SHA256, KSB, {NA, NB, d, formB, formB2})
```

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

```
MB = HMAC(SHA256, KMB, CB, IDB)
```

Bob MUST send Alice the Base64 encoded value of the HMAC (using SHA256 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. [16]

```
HMAC(SHA256, 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.

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

Listing 8: 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>
  <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>
```

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(SHA256, 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.

4.6 Final Steps (Alice)

4.6.1 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 SHA256) 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 SHA256 result of the concatenated string of bytes:

```
K = SHA256(K | SRS | OSS)
```

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

```
HMAC(SHA256, K, "New Retained Secret")
```

Alice MUST *securely* store the new value along with the retained secrets her client shares with Bob's other clients.
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.

4.6.2 Verifying Bob's Identity

Finally, Alice MUST verify the identity she received from Bob. She does this by performing steps equivalent to those performed by Bob above (see Verifying Alice's Identity for a more detailed description).
Alice's calculations are summarised below. Note: formB is the full Normalized *content* of the initial reponse data form Alice received from Bob (see Response Form), and formB2 is the full Normalized *content* of the session negotiation completion form she received from Bob *excluding* the 'identity' and 'mac' fields (see Sending Bob's Identity). Note: When calculating macB pay attention to the order of NA and NB.

```
MB = HMAC(SHA256, KMB, CB, IDB)
```

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

```
macB = HMAC(SHA256, KSB, {NA, NB, d, formB, formB2})
```

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). She MUST then securely destroy all keys associated with the ESession.

Listing 9: 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>
    <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. He MUST then securely destroy all keys associated with the ESession.

Listing 10: 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>
    <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 [17]).
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 [18]).

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.

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

7.3 Unverified SAS

Since very few people bother to (consistently) verify SAS, entities SHOULD protect against 'man-in-the-middle' attacks using retained secrets (and/or other 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.

7.4 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).

7.5 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 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 this protocol MUST support the following algorithms:

- Diffie-Hellman Key Agreement
- The block cipher algorithm ”aes128-ctr” (see AES [19])
- The hash algorithm ”sha256” (see Secure Hash Standard)
- HMAC (see Section 2 of RFC 2104 [20])
- The Short Authentication String generation algorithm ”sas28x5” (see The sas28x5 SAS Algorithm)

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 SHA256, 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 SHA256 of the string
3. Convert the 24-bit integer into a base-28 [21] 5-character string using the following ”digits”: acdefghikmopqruvwxy123456789 (the digits have values 0-27)

10 IANA Considerations

This document requires no interaction with the Internet Assigned Numbers Authority (IANA) [22].

[19] Advanced Encryption Standard: Federal Information Processing Standards Publication 197 <http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf>.

[20] RFC 2104: HMAC: Keyed-Hashing for Message Authentication <http://tools.ietf.org/html/rfc2104>.

[21] 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'.

[22] 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/>.

11 XMPP Registrar Considerations

See Encrypted Session Negotiation.