XEP-0188: Cryptographic Design of Encrypted Sessions

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

2007-05-30
Version 0.6

Status	Type	Short Name
Deferred	Informational	N/A

This document describes the cryptographic design that underpins the XMPP protocol extensions Encrypted Session Negotiation, Offline Encrypted Sessions and Stanza Encryption.

Legal

Copyright

Permissions

Warranty

Liability

Conformance

Contents

1 Introduction

Note: The protocols developed according to the cryptographic design described in this document are described in Encrypted Session Negotiation (XEP-0116) [1], Simplified Encrypted Session Negotiation (XEP-0217) [2], Offline Encrypted Sessions (XEP-0187) [3] and Stanza Encryption (XEP-0200) [4]. The information in those documents should be sufficient for implementors. This purely informative document is primarily for people interested in the design and analysis of those protocols.

As specified in RFC 3920 [5], XMPP is an XML streaming protocol that enables the near-real-time exchange of XML fragments between any two (or more) network endpoints. To date, the main application built on top of the core XML streaming layer is instant messaging (IM) and presence, the base extensions for which are specified in RFC 3921 [6]. There are three first-level elements of XML streams (<message/>, <presence/>, and <iq/>); each of these "XML stanza" types has different semantics, which can complicate the task of defining a generalized approach to end-to-end encryption for XMPP. In addition, XML stanzas can be extended (via properly-namespaced child elements) for a wide variety of functionality.

XMPP is a session-oriented communication technology: normally, a client authenticates with a server and maintains a long-lived connection that defines the client's XMPP session. Such stream-level sessions may be secured via channel encryption using Transport Level Security (RFC 2246 [7]), as specified in Section 5 of RFC 3920 [8]. However, there is no guarantee that all hops will implement or enforce channel encryption (or that intermediate servers are trustworthy), which makes end-to-end encryption desirable.

This document specifies a method for encrypted sessions ("ESessions") that takes advantage of the inherent possibilities and strengths of session encryption as opposed to object encryption. The detailed requirements for encrypted sessions are defined in Requirements for Encrypted Sessions (XEP-0210) [9].

The conceptual model for the approach specified in this document was inspired by "off-the-record" (OTR) communication, as implemented in the Gaim encryption plugin and described in Off-the-Record Communication [10]. 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 all one-to-one XML stanzas exchanged between the endpoints will be encrypted and then transmitted within a "wrapper" protocol element.

Note: In order to gain a thorough understanding of this document, it is recommended that the Off-the-Record Communication paper and RFC 6189 [11] are read first.

[1] XEP-0116: Encrypted Session Negotiation <https://xmpp.org/extensions/xep-0116.html>.
[2] XEP-0217: Simplified Encrypted Session Negotiation <https://xmpp.org/extensions/xep-0217.html>.
[3] XEP-0187: Offline Encrypted Sessions <https://xmpp.org/extensions/xep-0187.html>.
[4] XEP-0200: Stanza Encryption <https://xmpp.org/extensions/xep-0200.html>.
[5] RFC 3920: Extensible Messaging and Presence Protocol (XMPP): Core <http://tools.ietf.org/html/rfc3920>.
[6] RFC 3921: Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence <http://tools.ietf.org/html/rfc3921>.
[7] RFC 2246: The TLS Protocol Version 1.0 <http://tools.ietf.org/html/rfc2246>.
[8] RFC 3920: Extensible Messaging and Presence Protocol (XMPP): Core <http://tools.ietf.org/html/rfc3920>.
[9] XEP-0210: Requirements for Encrypted Sessions <https://xmpp.org/extensions/xep-0210.html>.
[10] Off-the-Record Communication, or, Why Not to Use PGP <http://www.cypherpunks.ca/otr/otr-codecon.pdf> <http://www.cypherpunks.ca/otr/otr-wpes.pdf>.
[11] RFC 6189: ZRTP: Media Path Key Agreement for Unicast Secure RTP <http://tools.ietf.org/html/rfc6189>.

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.
2. "Bob" is the name of the other participant in the ESession started by Alice.

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 Cryptographic Origins

3.1 Introduction

Authenticated key-exchange is the most challenging part of the design of any secure communication protocol. The ESessions key exchange essentially translates the SIGMA [12][13] key-exchange protocol into the syntax of XMPP. The SIGMA approach to Diffie-Hellman Key Agreement (see RFC 2631 [15]) underpins several standard key-exchange protocols including the Internet Key Exchange (IKE) protocol versions 1 and 2 (see RFC 2409 [16] and RFC 4306 [17]).
Note: Although this section provides an overview of SIGMA, it is strongly recommended that the SIGMA paper is read first in order to gain a thorough understanding of this document.
The 3-message SIGMA-I-based key exchange 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 two 4-message SIGMA-R-based key exchanges with hash commitment defend the *responder's* identity against active attacks and facilitate detection of a Man in the Middle attack. They SHOULD be used to establish client to client sessions.
Note: The block cipher function, *cipher*, uses CTR mode.

3.2 SIGMA Parameter Descriptions

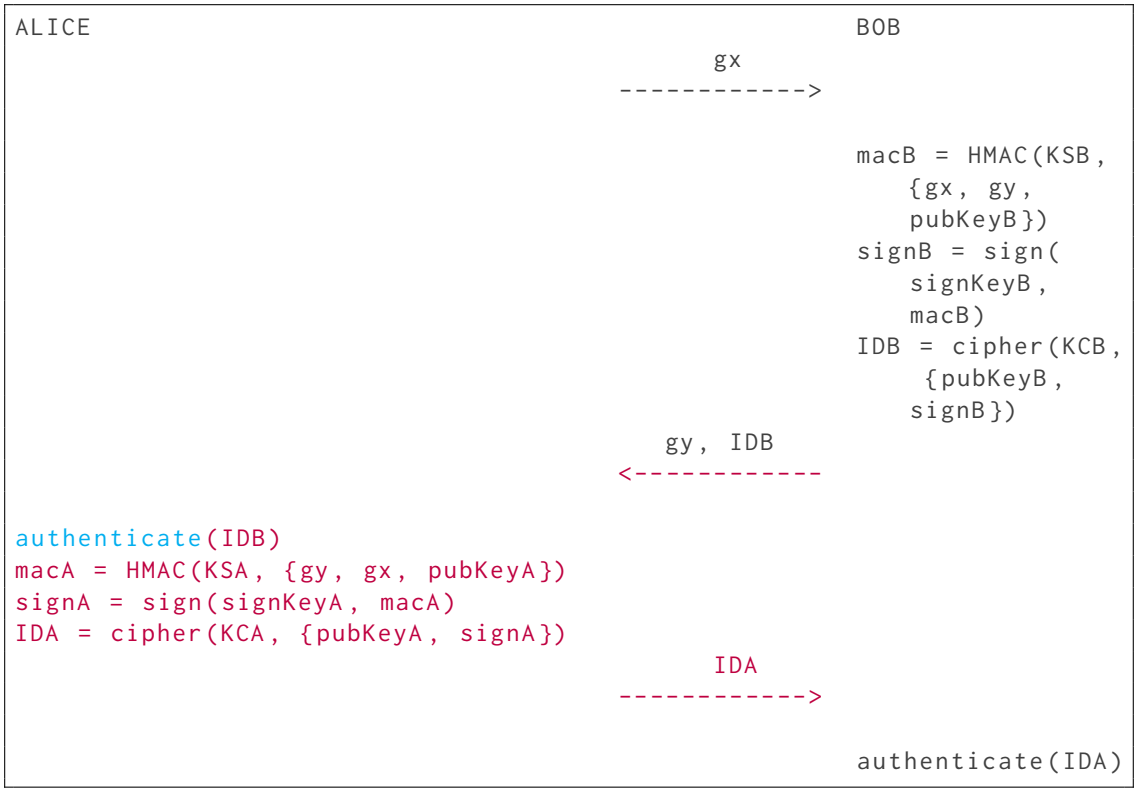
[12] SIGMA: the 'SIGn-and-MAc' Approach to Authenticated Diffie-Hellman and its Use in the IKE Protocols (Hugo Krawczyk, June 12 2003) <http://www.ee.technion.ac.il/~{}hugo/sigma.ps>.
[13] Like RFC 2409 [14], this protocol uses *variant (ii)*, as described in Secion 5.4 of the SIGMA paper.
[15] RFC 2631: Diffie-Hellman Key Agreement Method <http://tools.ietf.org/html/rfc2631>.
[16] RFC 2409: The Internet Key Exchange (IKE) <http://tools.ietf.org/html/rfc2409>.
[17] RFC 4306: Internet Key Exchange (IKEv2) Protocol <http://tools.ietf.org/html/rfc4306>.

Parameter	Description
g	Diffie-Hellman generator
x, y	Alice and Bob's private Diffie-Hellman keys
gx, gy	Alice and Bob's public Diffie-Hellman keys
Hgx	Hash of Alice's public Diffie-Hellman key
KSA, KSB	The MAC keys (derived from K) that Alice and Bob use to calculate macA and macB
pubKeyA, pubKeyB	The public keys that represent the identity of Alice and Bob, and are used to verify their signatures
macA, macB	The MAC values that associate the shared secret with the identity of Alice or Bob
signKeyA, signKeyB	The private keys that Alice and Bob use to sign
signA, signB	Alice's and Bob's signatures of the shared secret
KCA, KCB	The cipher keys (derived from K) that Alice and Bob use to encrypt
IDA, IDB	The encrypted parameters that identify Alice and Bob to each other
SAS	Short Authentication String

Parameter	Description
p	Diffie-Hellman prime
e, d	Alice and Bob's public Diffie-Hellman keys (the same as gx, gy)
He	Hash of Alice's public Diffie-Hellman key
K	Shared secret (derived by Alice from gy and x, or by Bob from gx and y)
HASH	Selected hash algorithm
NA, NB	Alice and Bob's session freshness nonces (ESession IDs)
CA, CB	Block cipher initial counter value for blocks sent by Alice and Bob
n	Block size of selected cipher algorithm in bits
KMA, KMB	The MAC keys (derived from K) that Alice and Bob use to protect the integrity of encrypted data
MA, MB	The MAC values that Alice and Bob use to confirm the integrity of encrypted data
SRS	Shared retained secret (derived from K in previous session between the clients)
RS1A...RSZA	Retained secrets Alice shares with Bob (one for each client he uses)
RS1B...RSZB	Retained secrets Bob shares with Alice (one for each client she uses)
RSH1A...RSHZA	HMACs of retained secrets Alice shares with Bob
SRSH	Bob's HMAC of SRS
OSS	Other shared secret of Alice and Bob (e.g. a shared password) defaults to "secret"
isPKA, isPKB	Whether or not Alice and Bob prefer to receive a public key (booleans)

3.3 SIGMA-I Overview

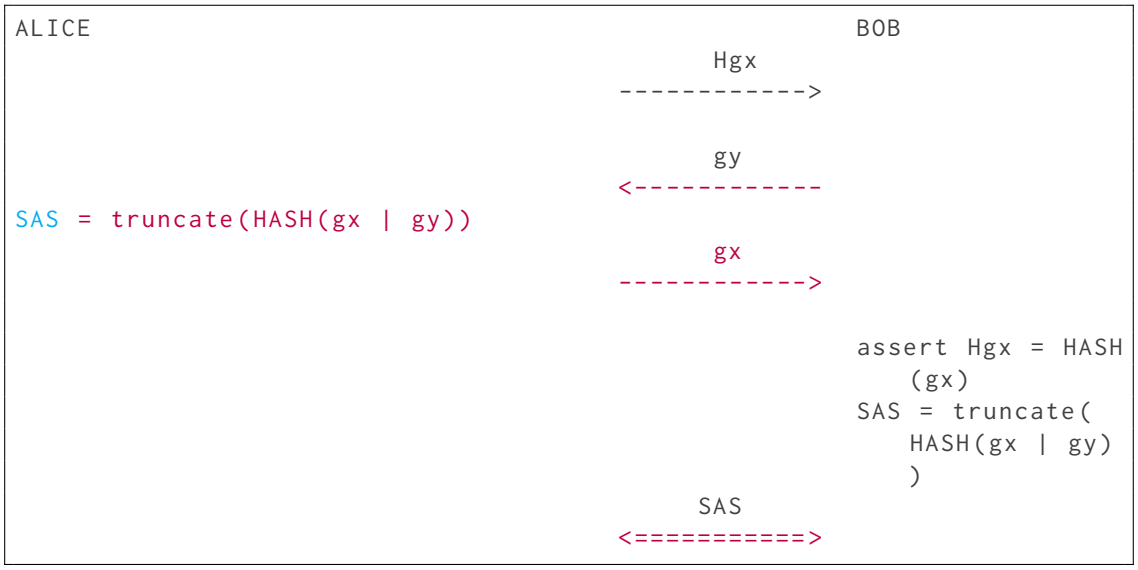
The diagram below demonstrates the barest cryptographic skeleton of the SIGMA-I key exchange protocol. Here Bob allows Alice to protect her identity from active attacks, by allowing her to authenticate him before she communicates her identity. Note: The cipher keys (KCA and KCB) are different in each direction, making this exchange slightly more conservative than SIGMA.



3.4 SAS-Only Overview

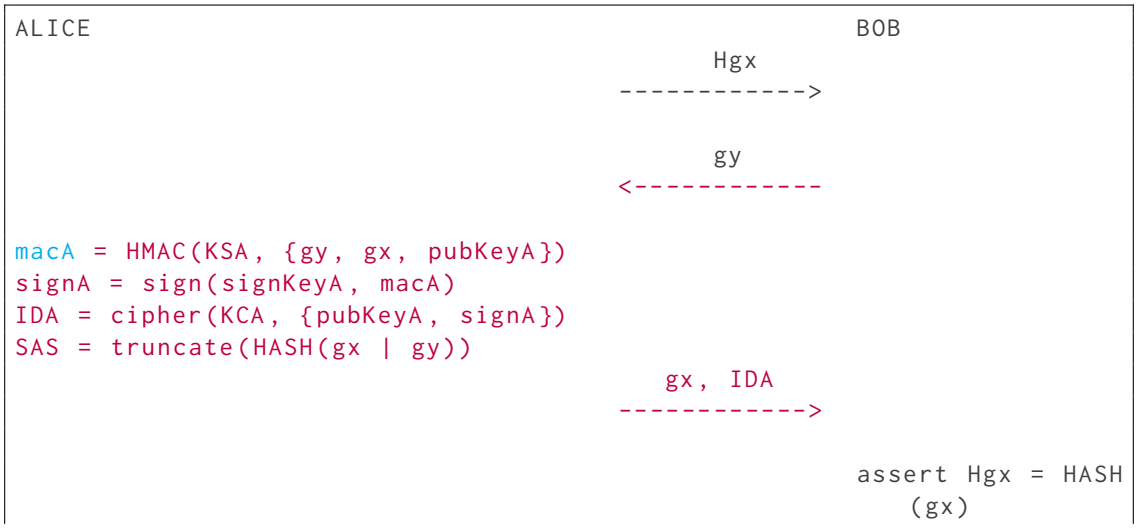
The diagram below demonstrates the skeleton of the Diffie-Hellman key exchange that employs out-of-band Short Authentication String (SAS) verification. If Alice and Bob's public keys are not yet trusted, or if their private keys have been compromised, then the hash commitment sent in the first step enables Alice and Bob to verify their copies of each other's Diffie-Hellman (and public) keys and detect a Man in the Middle more easily.
If a Man in the Middle changes the public Diffie-Hellman keys that Alice and Bob receive, then he could potentially use his knowledge of the SAS that Bob will eventually calculate when choosing the key he will send to Alice in the second step. However, the fact that the value he received in the first step is only a hash means the Man in the Middle must choose the key he sends to Alice before he can predict the SAS that she will calculate with it. Therefore, even if the SAS is very short, he is unable to use his resources to choose a key that will (have a better

than random chance to) result in a SAS that matches Bob's. So only a truncated version of the HASH of Alice and Bob's keys needs to be verified out-of-band in the final step.



3.5 SIGMA-R with SAS Overview

The logic of the four-step SIGMA-R protocol is similar to the three-step SIGMA-I protocol. The difference being that Bob protects his identity from active attacks by by delaying communicating his identity to Alice until he has authenticated her. The diagram below demonstrates the skeleton of the key exchange. Note that it also takes advantage of the extra step required for SIGMA-R to incorporate a hash commitment, thus enabling *optional* out-of-band SAS authentication.



```
                                                    SAS = truncate(
                                                        HASH(gx | gy)
                                                        )
                                        SAS
                                    <===========>

                                                    authenticate(IDA)
                                                    macB = HMAC(KSB,
                                                        {gx, gy,
                                                        pubKeyB})
                                                    signB = sign(
                                                        signKeyB,
                                                        macB)
                                                    IDB = cipher(KCB,
                                                         {pubKeyB,
                                                        signB})
                                        IDB
                                    <------------
authenticate(IDB)
```

3.6 SIGMA-I Key Exchange

The diagram below describes exactly the same SIGMA-I key exchange protocol as the SIGMA-I Overview above. It provides much more detail, without specifying any ESession-specific details. The differences between it and the SIGMA-R with SAS Key Exchange are highlighted.

```
ALICE                                               BOB

NA = random()
x = random()
e = gx mod p
                                    e, NA
                                ------------>
                                                    NB = random()
                                                    CA = random()
                                                    CB = CA XOR 2n-1
                                                    y = random()
                                                    d = gy mod p
                                                    assert 1 < e < p-1
                                                    K = HASH(ey mod p)
                                                    KCA = HMAC(HASH, K, "
                                                        Initiator␣Cipher␣Key"
                                                        )
                                                    KCB = HMAC(HASH, K, "
                                                        Responder␣Cipher␣Key"
                                                        )
                                                    KMA = HMAC(HASH, K, "
                                                        Initiator␣MAC␣Key")
```

```
                                                    KMB = HMAC(HASH, K, ”
                                                        Responder␣MAC␣Key”)
                                                    KSA = HMAC(HASH, K, ”
                                                        Initiator␣SIGMA␣Key”)
                                                    KSB = HMAC(HASH, K, ”
                                                        Responder␣SIGMA␣Key”)
                                                    macB = HMAC(HASH, KSB, {
                                                        NA, NB, d, pubKeyB,
                                                        CA})
                                                    signB = sign(signKeyB,
                                                        macB)
                                                    IDB = cipher(KCB, CB, {
                                                        pubKeyB, signB})
                                                    MB = HMAC(HASH, KMB, CB,
                                                        IDB)
                                  d, CA, NB
                                <------------
                                   IDB, MB
CB = CA XOR 2n-1
assert 1 < d < p-1
K = HASH(dx mod p)
KCA = HMAC(HASH, K, ”Initiator␣Cipher␣Key”)
KCB = HMAC(HASH, K, ”Responder␣Cipher␣Key”)
KMA = HMAC(HASH, K, ”Initiator␣MAC␣Key”)
KMB = HMAC(HASH, K, ”Responder␣MAC␣Key”)
KSA = HMAC(HASH, K, ”Initiator␣SIGMA␣Key”)
KSB = HMAC(HASH, K, ”Responder␣SIGMA␣Key”)
assert MB = HMAC(HASH, KMB, CB, IDB)
{pubKeyB, signB} = decipher(KCB, CB, IDB)
macB = HMAC(HASH, KSB, {NA, NB, d, pubKeyB, CA})
verify(signB, pubKeyB, macB)
macA = HMAC(HASH, KSA, {NB, NA, e, pubKeyA})
signA = sign(signKeyA, macA)
IDA = cipher(KCA, CA, {pubKeyA, signA})
MA = HMAC(HASH, KMA, CA, IDA)
                                     IDA
                                ------------>
                                      MA
                                                    assert MA = HMAC(HASH,
                                                        KMA, CA, IDA)
                                                    {pubKeyA, signA} =
                                                        decipher(KCA, CA, IDA
                                                        )
                                                    macA = HMAC(HASH, KSA, {
                                                        NB, NA, e, pubKeyA})
                                                    verify(signA, pubKeyA,
                                                        macA)
```

3.7 SIGMA-R with SAS Key Exchange

The Short Authentication String technique enables protection against a Man in the Middle without the need to generate, distribute or authenticate any public keys. As long as a hash commitment is used at the start of the key exchange then only a short human-friendly string needs to be verified out-of-band (e.g. by recognizable voice communication).
Furthermore, if retained secrets associated with a client/user combination are employed *consistently* during key exchanges, then the Man in the Middle would need to be present for every session, including the first, and the out-of-band verification would only need to be performed once to verify the absence of a Man in the Middle for all sessions between the parties (past, present and future). [18]
Public keys are optional in the diagram below. It describes the same SIGMA-R with SAS key exchange protocol as the SIGMA-R Overview. It provides much more detail including the use of retained secrets and other secrets. The use of public keys is negotiated in the first two messages. Note: These *optional* security enhancements are especially important when the protocol is being used without public keys.
The diagram does not specify any ESession-specific details. The differences between it and the SIGMA-I Key Exchange are highlighted.

```
ALICE                                              BOB

NA = random()
x = random()
e = gx mod p
He = SHA256(e)
                                 He, isPKA
                               ------------>
                                 isPKB, NA
                                                   NB = random()
                                                   CA = random()
                                                   CB = CA XOR 2n-1
                                                   y = random()
                                                   d = gy mod p
                                 d, CA, NB
                               <------------
                                isPKA, isPKB
CB = CA XOR 2n-1
assert 1 < d < p-1
K = HASH(dx mod p)
KCA = HMAC(HASH, K, "Initiator Cipher Key")
KMA = HMAC(HASH, K, "Initiator MAC Key")
KSA = HMAC(HASH, K, "Initiator SIGMA Key")
RSH1A...RSHZA = HMAC(HASH, NA, RS1A...RSZA)
if isPKB equals false then:
    macA = HMAC(HASH, KSA, {NB, NA, e, RSH1A...RSHZA})
```

[18] This combination of techniques underpins the ZRTP key agreement protocol.

```
    IDA = cipher(KCA, CA, macA)
else:
    macA = HMAC(HASH, KSA, {NB, NA, e, pubKeyA, RSH1A...RSHZA})
    signA = sign(signKeyA, macA)
    IDA = cipher(KCA, CA, {pubKeyA, signA})
MA = HMAC(HASH, KMA, CA, IDA)
SAS = truncate(HASH(MA | d | "Short Authentication String"))
```

```
                                IDA, MA
                              ------------>
                              e, RSH1A...RSHZA

                                                assert He = SHA256(e)
                                                SAS = truncate(HASH(MA |
                                                    d | "Short
                                                    Authentication String
                                                    "))
                                    SAS
                              <============>

                                                assert 1 < e < p-1
                                                K = HASH(ey mod p)
                                                KCA = HMAC(HASH, K, "
                                                    Initiator Cipher Key"
                                                    )
                                                KMA = HMAC(HASH, K, "
                                                    Initiator MAC Key")
                                                KSA = HMAC(HASH, K, "
                                                    Initiator SIGMA Key")

                                                assert MA = HMAC(HASH,
                                                    KMA, CA, IDA)
                                                if isPKB equals false
                                                    then:
                                                     macA = decipher(KCA,
                                                         CA, IDA)
                                                     assert macA = HMAC(
                                                         HASH, KSA, {NB,
                                                         NA, e, RSH1A...
                                                         RSHZA})
                                                else:
                                                    {pubKeyA, signA} =
                                                        decipher(KCA, CA,
                                                         IDA)
                                                    macA = HMAC(HASH, KSA
                                                        , {NB, NA, e,
                                                        pubKeyA, RSH1A...
                                                        RSHZA})
                                                    verify(signA, pubKeyA
```

```
                                                        , macA)
                                                SRS = choose(RS1B...RSZB,
                                                     RSH1A...RSHZA, NA)
                                                K = HASH(K | SRS | OSS)
                                                KCA = HMAC(HASH, K, "
                                                    Initiator␣Cipher␣Key"
                                                    )
                                                KCB = HMAC(HASH, K, "
                                                    Responder␣Cipher␣Key"
                                                    )
                                                KMA = HMAC(HASH, K, "
                                                    Initiator␣MAC␣Key")
                                                KMB = HMAC(HASH, K, "
                                                    Responder␣MAC␣Key")
                                                KSB = HMAC(HASH, K, "
                                                    Responder␣SIGMA␣Key")
                                                SRSH = HMAC(HASH, SRS, "
                                                    Shared␣Retained␣
                                                    Secret")
                                                retain(HMAC(HASH, K, "New
                                                    ␣Retained␣Secret"))
                                                if isPKA equals false
                                                    then:
                                                     macB = HMAC(HASH, KSB
                                                         , {NA, NB, d, CA
                                                         })
                                                     IDB = cipher(KCB, CB,
                                                          macB)
                                                else:
                                                     macB = HMAC(HASH, KSB
                                                         , {NA, NB, d,
                                                         pubKeyB, CA})
                                                     signB = sign(signKeyB
                                                         , macB)
                                                     IDB = cipher(KCB, CB,
                                                          {pubKeyB, signB
                                                         })
                                                MB = HMAC(HASH, KMB, CB,
                                                    IDB)
                                  IDB
                               <------------
                                  MB, SRSH

SRS = choose(RS1A...RSZA, SRSH)
K = HASH(K | SRS | OSS)
KCA = HMAC(HASH, K, "Initiator␣Cipher␣Key")
KCB = HMAC(HASH, K, "Responder␣Cipher␣Key")
KMA = HMAC(HASH, K, "Initiator␣MAC␣Key")
KMB = HMAC(HASH, K, "Responder␣MAC␣Key")
```

```
KSB = HMAC(HASH, K, "Responder_SIGMA_Key")
retain(HMAC(HASH, K, "New_Retained_Secret"))
assert MB = HMAC(HASH, KMB, CB, IDB)
if isPKA equals false then:
    macB = decipher(KCB, CB, IDB)
    assert macB = HMAC(HASH, KSB, {NA, NB, d, CA})
else:
    {pubKeyB, signB} = decipher(KCB, CB, IDB)
    macB = HMAC(HASH, KSB, {NA, NB, d, pubKeyB, CA})
    verify(signB, pubKeyB, macB)
```

4 Cryptographic Design

This section provides an overview of the full ESession key-exchange protocol from a cryptographic point of view. This protocol is based on the *full fledge* protocol, as described in Appendix B of the SIGMA paper. It also uses *variant (ii)*, as described in Secion 5.4 of the same paper.

4.1 ESession Parameter Descriptions

The table below describes the parameters that are not found in the Parameter Descriptions tables above.

Parameter	Description
options	Includes a set of possible values for each and every ESession parameter (see the ESession Request sub-section in Encrypted Session Negotiation), including sets of possible values for p, g, HASH, CIPHER, SIGN
chosen	Includes a chosen value for each ESession parameter
CIPHER	Selected CTR-mode block cipher algorithm
DECIPHER	Selected CTR-mode block decipher algorithm (corresponds to CIPHER)
SIGN	Selected signature algorithm
VERIFY	The selected signature verification algorithm (corresponds to SIGN)
SASGEN	The selected SAS generation algorithm
x1...xZ	Alice's private Diffie-Hellman keys - each value corresponds to one of Z different DH groups
e1...eZ	The choice of public Diffie-Hellman keys that Alice offers Bob - each value corresponds to one of Z different DH groups (and a different value of x)
He1...HeZ	The list of hash commitments that Alice sends to Bob (hashes of e1...eZ)
signKeysA	All the private keys that Alice is able to use to create signatures
signsB	The set of signatures of formB (one for each of Bob's private keys)
pubKeysA	All of Alice's public keys that Bob has access to

* Offline negotiation only

4.2 Online ESession-I Negotiation

Alice uses this protocol when Bob is Online. In addition to the key exchange described in the SIGMA-I Key Exchange protocol above, she offers Bob a choice of Diffie-Hellman groups with her corresponding values of e, various algorithms and other parameters. The differences between this protocol and Online ESession-R Negotiation are highlighted.

```
ALICE                                          BOB

NA = random()
for g,p   options
    x = random()
    e = gx mod p
formA = {e1...eZ, options, NA}
                                    formA
                                 --------->

                                               chosen = {p,g,HASH,CIPHER,
                                                   SIGN...} = choose(options
                                                   )
                                               e = choose(e1...eZ, p)
                                               NB = random()
                                               CA = random()
                                               CB = CA XOR 2n-1
                                               y = random()
                                               d = gy mod p
                                               formB = {CA, chosen, d, NA,
                                                   NB}
                                               assert 1 < e < p-1
                                               K = HASH(ey mod p)
                                               KCA = HMAC(HASH, K, "
                                                   Initiator␣Cipher␣Key")
                                               KCB = HMAC(HASH, K, "
                                                   Responder␣Cipher␣Key")
                                               KMA = HMAC(HASH, K, "
                                                   Initiator␣MAC␣Key")
                                               KMB = HMAC(HASH, K, "
                                                   Responder␣MAC␣Key")
                                               KSA = HMAC(HASH, K, "
                                                   Initiator␣SIGMA␣Key")
                                               KSB = HMAC(HASH, K, "
                                                   Responder␣SIGMA␣Key")
                                               macB = HMAC(HASH, KSB, {NA,
                                                   NB, d, pubKeyB, formB})
```

```
                                          signB = SIGN(signKeyB, macB)
                                          IDB = CIPHER(KCB, CB, {
                                              pubKeyB, signB})
                                          MB = HMAC(HASH, KMB, CB, IDB)

                                 formB
                               <---------
                                 IDB, MB
assert chosen   options
x = choose(x1...xZ, p)
e = gx mod p
CB = CA XOR 2n-1
assert 1 < d < p-1
K = HASH(dx mod p)
KCA = HMAC(HASH, K, "Initiator Cipher Key")
KCB = HMAC(HASH, K, "Responder Cipher Key")
KMA = HMAC(HASH, K, "Initiator MAC Key")
KMB = HMAC(HASH, K, "Responder MAC Key")
KSA = HMAC(HASH, K, "Initiator SIGMA Key")
KSB = HMAC(HASH, K, "Responder SIGMA Key")
assert MB = HMAC(HASH, KMB, CB, IDB)
{pubKeyB, signB} = DECIPHER(KCB, CB, IDB)
macB = HMAC(HASH, KSB, {NA, NB, d, pubKeyB, formB})
VERIFY(signB, pubKeyB, macB)
macA = HMAC(HASH, KSA, {NB, NA, e, pubKeyA, formA})
signA = SIGN(signKeyA, macA)
IDA = CIPHER(KCA, CA, {pubKeyA, signA})
MA = HMAC(HASH, KMA, CA, IDA)
                                   IDA
                                 --------->
                                   MA, NB
                                          assert MA = HMAC(HASH, KMA,
                                              CA, IDA)
                                          {pubKeyA, signA} = DECIPHER(
                                              KCA, CA, IDA)
                                          macA = HMAC(HASH, KSA, {NB,
                                              NA, e, pubKeyA, formA})
                                          VERIFY(signA, pubKeyA, macA)
```

4.3 Online ESession-R Negotiation

This protocol is similar to the Online ESession-I Negotiation above, except that Bob's identity is protected from active attacks (by by delaying communicating his identity to Alice until he has authenticated her). The optional use of SAS, retained secrets and other secrets means the protocol may be used without any public keys. The differences between this protocol and Online ESession-I Negotiation are highlighted.

```
ALICE                                                BOB

NA = random()
for g,p   options
    x = random()
    e = gx mod p
    He = SHA256(e)
formA = {He1...HeZ, options, NA}

                                    formA
                                ------------>

                                                chosen = {p,g,HASH,CIPHER,
                                                    SIGN,SASGEN,isPKA,isPKB
                                                    ...} = choose(options)
                                                He = choose(He1...HeZ, p)
                                                NB = random()
                                                CA = random()
                                                CB = CA XOR 2n-1
                                                y = random()
                                                d = gy mod p
                                                formB = {CA, chosen, d, NA,
                                                     NB}
                                    formB
                                <------------
assert chosen   options
x = choose(x1...xZ, p)
e = choose(e1...eZ, p)
CB = CA XOR 2n-1
assert 1 < d < p-1
K = HASH(dx mod p)
KCA = HMAC(HASH, K, "Initiator Cipher Key")
KMA = HMAC(HASH, K, "Initiator MAC Key")
KSA = HMAC(HASH, K, "Initiator SIGMA Key")
RSH1A...RSHZA = HMAC(HASH, NA, RS1A...RSZA)
formA2 = {RSH1A...RSHZA, e, NB}
if isPKB equals false then:
    macA = HMAC(HASH, KSA, {NB, NA, e, formA, formA2})
    IDA = CIPHER(KCA, CA, macA)
else:
    macA = HMAC(HASH, KSA, {NB, NA, e, pubKeyA, formA, formA2})
    signA = SIGN(signKeyA, macA)
    IDA = CIPHER(KCA, CA, {pubKeyA, signA})
MA = HMAC(HASH, KMA, CA, IDA)
SAS = SASGEN(MA, formB)
                                   IDA, MA
                                ------------>
                                    formA2
```

```
                                        assert He = SHA256(e)
                                        SAS = SASGEN(MA, formB)
                          SAS
                <===========>

                                        assert 1 < e < p-1
                                        K = HASH(ey mod p)
                                        KCA = HMAC(HASH, K, "
                                            Initiator Cipher Key")
                                        KMA = HMAC(HASH, K, "
                                            Initiator MAC Key")
                                        KSA = HMAC(HASH, K, "
                                            Initiator SIGMA Key")
                                        assert MA = HMAC(HASH, KMA,
                                             CA, IDA)
                                        if isPKB equals false
                                            then:
                                             macA = DECIPHER(KCA, CA
                                                 , IDA)
                                             assert macA = HMAC(HASH
                                                 , KSA, {NB, NA, e,
                                                 formA, formA2})
                                        else:
                                             {pubKeyA, signA} =
                                                 DECIPHER(KCA, CA,
                                                 IDA)
                                             macA = HMAC(HASH, KSA,
                                                 {NB, NA, e, pubKeyA
                                                 , formA, formA2})
                                             VERIFY(signA, pubKeyA,
                                                 macA)
                                        SRS = choose(RS1B...RSZB,
                                            RSH1A...RSHZA, NA)
                                        K = HASH(K | SRS | OSS)
                                        KCA = HMAC(HASH, K, "
                                            Initiator Cipher Key")
                                        KCB = HMAC(HASH, K, "
                                            Responder Cipher Key")
                                        KMA = HMAC(HASH, K, "
                                            Initiator MAC Key")
                                        KMB = HMAC(HASH, K, "
                                            Responder MAC Key")
                                        KSB = HMAC(HASH, K, "
                                            Responder SIGMA Key")
                                        if SRS equals false then:
                                             SRS = random()
                                        SRSH = HMAC(HASH, SRS, "
                                            Shared Retained Secret"
                                            )
```

```
                                                retain(HMAC(HASH, K, "New␣
                                                    Retained␣Secret"))
                                                formB2 = {NA, SRSH}
                                                if isPKA equals false
                                                    then:
                                                    macB = HMAC(HASH, KSB,
                                                        {NA, NB, d, formB,
                                                        formB2})
                                                    IDB = CIPHER(KCB, CB,
                                                        macB)
                                                else:
                                                    macB = HMAC(HASH, KSB,
                                                        {NA, NB, d, pubKeyB
                                                        , formB, formB2})
                                                    signB = SIGN(signKeyB,
                                                        macB)
                                                    IDB = CIPHER(KCB, CB, {
                                                        pubKeyB, signB})
                                                MB = HMAC(HASH, KMB, CB,
                                                    IDB)

                                  IDB, MB
                                <------------
                                    formB2

SRS = choose(RS1A...RSZA, SRSH)
K = HASH(K | SRS | OSS)
KCA = HMAC(HASH, K, "Initiator␣Cipher␣Key")
KCB = HMAC(HASH, K, "Responder␣Cipher␣Key")
KMA = HMAC(HASH, K, "Initiator␣MAC␣Key")
KMB = HMAC(HASH, K, "Responder␣MAC␣Key")
KSB = HMAC(HASH, K, "Responder␣SIGMA␣Key")
retain(HMAC(HASH, K, "New␣Retained␣Secret"))
assert MB = HMAC(HASH, KMB, CB, IDB)
if isPKA equals false then:
    macB = DECIPHER(KCB, CB, IDB)
    assert macB = HMAC(HASH, KSB, {NA, NB, d, formB, formB2})
else:
    {pubKeyB, signB} = DECIPHER(KCB, CB, IDB)
    macB = HMAC(HASH, KSB, {NA, NB, d, pubKeyB, formB, formB2})
    VERIFY(signB, pubKeyB, macB)
```

4.4 Offline ESession Negotiation

Bob uses this protocol to send stanzas to Alice when she is Offline. Note: Since the full SIGMA protocol cannot be used if Alice is offline, her identity is not protected at all.
The diagram is split into three phases. First Alice publishes her ESession options before going offline. Later Bob completes the key exchange (and sends her encrypted stanzas that are not

shown below) these are all stored by Alice's server. Finally when Alice comes online again she verifies and calculates the decryption key.
The differences between this offline protocol and the Online ESession-I Negotiation protocol above are highlighted in the diagram below.

```
ALICE                    ALICE'S SERVER              BOB

NA = random()
for g,p  options
    x = random()
    e = gx mod p
formA = {e1...eZ, options, NA}
signsA = multi_sign(signKeysA, formA)
retain(NA, x1...xZ, expireTime)

                    formA
                --------->
                    signsA

                          retain(formA, signsA)
-------------------------------------------------------------------------------------

                          retrieve(formA, signsA)

                                                  formA
                                              --------->
                                                  signsA

                                                          verify_one(signsA
    , pubKeysA, formA)
                                                          chosen = {p,g,
    HASH,CIPHER,SIGN...} = choose(options)
                                                          e = choose(e1...
    eZ, p)
                                                          CA = random()
                                                          y = random()
                                                          d = gy mod p
                                                          CB = CA XOR 2n-1
                                                          NB = random()
                                                          formB = {CA,
    chosen, d, NA, NB}
                                                          assert 1 < e < p
    -1
                                                          K = HASH(ey mod p
    )
                                                          KCA = HMAC(HASH,
    K, "Initiator Cipher Key")
                                                          KCB = HMAC(HASH,
```

```
    K, "Responder Cipher Key")
                                                        KMA = HMAC(HASH,
    K, "Initiator MAC Key")
                                                        KMB = HMAC(HASH,
    K, "Responder MAC Key")
                                                        KSA = HMAC(HASH,
    K, "Initiator SIGMA Key")
                                                        KSB = HMAC(HASH,
    K, "Responder SIGMA Key")
                                                        macB = HMAC(HASH,
    KSB, {NA, NB, d, pubKeyB, formB})
                                                        signB = SIGN(
    signKeyB, macB)
                                                        IDB = CIPHER(KCB,
    CB, {pubKeyB, signB})
                                                        MB = HMAC(HASH,
    KMB, CB, IDB)

                                                formB
                                        <--------
                                                IDB, MB

                            retain(formB,IDB,MB)
-------------------------------------------------------------------------------------------

                            retrieve(formB,IDB,MB)(formB,IDB,MB)
-------------------------------------------------------------------------------------------

                            retrieve(formB,IDB,MB)(formB,IDB,MB)
-------------------------------------------------------------------------------------------

                            retrieve(formB,IDB,MB)(formB,IDB,MB)
-------------------------------------------------------------------------------------------

                            retrieve(formB,IDB,MB)
                        formB
                <--------
                        IDB, MB

retrieve(NA, x1...xZ, expireTime)
assert now < expireTime
assert chosen  options
x = choose(x1...xZ, p)
e = gx mod p
CB = CA XOR 2n-1
assert 1 < d < p-1
K = HASH(dx mod p)
KCA = HMAC(HASH, K, "Initiator Cipher Key")
KCB = HMAC(HASH, K, "Responder Cipher Key")
```

```
KMA = HMAC(HASH, K, "Initiator MAC Key")
KMB = HMAC(HASH, K, "Responder MAC Key")
KSA = HMAC(HASH, K, "Initiator SIGMA Key")
KSB = HMAC(HASH, K, "Responder SIGMA Key")
assert MB = HMAC(HASH, KMB, CB, IDB)
{pubKeyB, signB} = DECIPHER(KCB, CB, IDB)
macB = HMAC(HASH, KSB, {NA, NB, d, pubKeyB, formB})
VERIFY(signB, pubKeyB, macB)
```

Note: KMB is necessary only to allow Bob to terminate the ESession if he comes online before Alice terminates it. The calculation of KCB and KSB is not strictly necessary.

5 Security Considerations

The security considerations are described in Encrypted Session Negotiation and Offline Encrypted Sessions.

6 IANA Considerations

This document requires no interaction with the Internet Assigned Numbers Authority (IANA) [19].

7 XMPP Registrar Considerations

This document requires no interaction with the XMPP Registrar [20].

8 Acknowledgments

The author would like to thank: Ian Goldberg for the time he spent reviewing an early version of this protocol and for his invaluable suggestions and comments; and Hugo Krawczyk for his general advice and encouragment. The author of this document is entirely responsable for any errors it contains.

[19] 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/>.

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