XEP-0102: Security Extensions

Jean-Louis Seguineau
mailto:jean-louis.seguineau@antepo.com
xmpp:jlseguineau@im.antepo.com

2003-06-25
Version 0.1

Status	Type	Short Name
Deferred	Standards Track	Not yet assigned

Security extensions for Jabber/XMPP.

Legal

Copyright

This XMPP Extension Protocol is copyright © 1999 – 2024 by the XMPP Standards Foundation (XSF).

Permissions

Permission is hereby granted, free of charge, to any person obtaining a copy of this specification (the ”Specification”), to make use of the Specification without restriction, including without limitation the rights to implement the Specification in a software program, deploy the Specification in a network service, and copy, modify, merge, publish, translate, distribute, sublicense, or sell copies of the Specification, and to permit persons to whom the Specification is furnished to do so, subject to the condition that the foregoing copyright notice and this permission notice shall be included in all copies or substantial portions of the Specification. Unless separate permission is granted, modified works that are redistributed shall not contain misleading information regarding the authors, title, number, or publisher of the Specification, and shall not claim endorsement of the modified works by the authors, any organization or project to which the authors belong, or the XMPP Standards Foundation.

Warranty

NOTE WELL: This Specification is provided on an ”AS IS” BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE.

Liability

In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall the XMPP Standards Foundation or any author of this Specification be liable for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising from, out of, or in connection with the Specification or the implementation, deployment, or other use of the Specification (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if the XMPP Standards Foundation or such author has been advised of the possibility of such damages.

Conformance

This XMPP Extension Protocol has been contributed in full conformance with the XSF’s Intellectual Property Rights Policy (a copy of which can be found at <https://xmpp.org/about/xsf/ipr-policy> or obtained by writing to XMPP Standards Foundation, P.O. Box 787, Parker, CO 80134 USA).

Contents

1 Introduction **1**

2 Terms and Definitions **1**

3 Requirements And Considerations **3**
3.1 Security Requirements . . . 3
3.1.1 Data Protection . . . 3
3.1.2 Data Classification . . . 3
3.1.3 End To End Protection . . . 4
3.1.4 Trust Issues . . . 4
3.1.5 Cryptosystem Design Considerations . . . 4
3.2 2.2 Environmental Considerations . . . 5
3.3 Usability . . . 5
3.4 Development And Deployment . . . 5
3.5 XML Processing . . . 6
3.5.1 Transporting Binary Content . . . 6
3.5.2 Transporting Encrypted Content . . . 6
3.5.3 Performing HMAC Computation . . . 6
3.5.4 Performing Cryptographic Operations . . . 7

4 xmpp:sec namespace **7**
4.1 Elements within the extension . . . 7
4.2 Attributes . . . 8
4.3 Elements . . . 8
4.4 Attributes values . . . 10

5 Base Key Agreement **11**
5.1 Overview . . . 11
5.1.1 Secure password registration . . . 12

6 Authenticated Key Agreement **13**
6.1 Introduction . . . 13
6.2 Key Exchange Protocol . . . 15
6.2.1 Aggressive Mode Key Exchange . . . 16
6.2.2 Main Mode Key Exchange . . . 16
6.2.3 Deriving key material for Cryptographic Transforms . . . 17
6.3 Authenticated Key Exchange Application . . . 18
6.3.1 Main Mode Key Exchange . . . 18
6.3.2 Aggressive Mode Key Exchange . . . 26

7 Key Transport **30**
7.1 Conversation Key Transport . . . 30
7.1.1 Key transport exchange . . . 31

7.1.2 Generating And Sending a Conversation Key Transport PDU 31
7.1.3 Receiving and Processing the Conversation Key Transport PDU 33
7.2 Public Key Transport 33
7.2.1 Certificate transport 34
7.2.2 Other Public Keys Transport 34

8 Message Protection **35**
8.1 Overview 35
8.2 Message Protection Mechanism 35
8.3 Generating And Sending the Protected Message PDU 36
8.4 Receiving and Processing the Protected Message PDU 37

9 Algorithms **38**

10 PKCS #3: Diffie-Hellman Key-Agreement Standard **39**
10.1 Scope 39
10.2 References 39
10.3 Definitions 39
10.4 Symbols and abbreviations 40
10.5 General overview 40
10.6 Parameter generation 40
10.6.1 Notes 41
10.7 Phase I 41
10.7.1 Private-value generation 41
10.7.2 Exponentiation 41
10.7.3 Integer-to-octet-string conversion 41
10.8 Phase II 42
10.8.1 Octet-string-to-integer conversion 42
10.8.2 Exponentiation 42
10.8.3 Integer-to-octet-string conversion 42
10.9 Object identifier 43
10.10 Revision history 43
10.10.1 Versions 1.0-1.2 43
10.10.2 Version 1.3 43
10.10.3 Version 1.4 43
10.11 Author's address 44

11 IKE Diffie-Hellman Groups **44**
11.1 768-bit MODP - modp1 44
11.2 1024-bit MODP Group - modp2 44
11.3 1536-bit MODP Group - modp5 45
11.4 2048-bit MODP Group 45
11.5 3072-bit MODP Group 45
11.6 4096-bit MODP Group 46

11.7 6144-bit MODP Group 47
11.8 8192-bit MODP Group 47

1 Introduction

While the benefits of IM are clear and compelling, the risks associated with sharing sensitive information in an IM environment are often overlooked. We need a mechanism that permits communities of users to protect their IM conversations. This document presents an extension protocol that can be incorporated into the existing XMPP protocol to provide such a mechanism.
In addition to its ability to protect instant message data, the proposed protocol may also serve as a foundation for securing other data transported via XMPP extensions.

2 Terms and Definitions

Term	Definition
User	A user is simply any XMPP user. Users are uniquely identified by a JID; they connect to XMPP hosts using a XMPP node. Users produce and consume information, and we wish to provide them with mechanisms that can be used to protect this information.
Community	A community is a collection of users who wish to communicate via XMPP. No restrictions or assumptions are made about the size of communities or the geographical, organizational, or national attributes of the members. Communities are assumed to be dynamic and ad-hoc. Users typically join communities by the simple act of invitation. All members of a community are assumed to be peers. The members of communities share information among themselves, and we wish to provide them with mechanisms that can permit information to only be shared by community members.
Conversation	A conversation is the set of messages that flows among the members of a community via some network. Conversations consist of both the actual conversation data produced and consumed by the various users as well as the XMPP protocol elements that transport it. Members participate in a conversation when they are the source or destination of this traffic.
Initiator	The initiator is the user who requested a security session negotiation. Initiator's are identified by their JID.
Responder	The responder is the user who responded to a security session negotiation request. Responder's are identified by their JID.
Concatentation operator	The '|' character is used in character or octet string expressions to indicate concatenation.
PFS	Perfect Forward Secrecy. In cryptography, is said of a key-establishment protocol in which the compromise of a session key or long-term private key after a given session does not cause the compromise of any earlier session.

Term	Definition
GRP	The definition of a Diffie-Hellman group length
DHx	The Diffie-Hellman ephemeral public keys for the initiator (x=i) and the responder (x=r)
KEY	The Diffie-Hellman ephemeral session secret that is agreed to during a key exchange negotiation.
CKYx	A 64 bits pseudo-random number or cookie generated by the initiator (x=i) and responder (x=r) in the authenticated key exchange.
KEYID	The concatenation of CKI-I and CKI-r and the domain of interpretation. It is the name of the keying material.
sKEYID	This is the keying material named by KEYID. It is never transmitted but is used in the various calculations made by the exchanging parties.
EHAo	A list of encryption/hash/authentication algorithms choices.
EHAs	The selected reference encryption/hash/authentication choice.
Nx	The nonces selected by the initiator (x=i) and the responder (x=r)
JIDx	The identities of the initiator (x=i) and the responder (x=r)
E{value}Kx	The encryption of value with the public key of the initiator (x=i) and the responder (x=r). Encryption is done using the algorithm associated with the authentication method. Usually this will be RSA
D{value}Kx	The decryption of value with the public key of the initiator (x=i) and the responder (x=r). Decryption is done using the algorithm associated with the authentication method. Usually this will be RSA
S{value}Kx	The signature of value with the private key of the initiator (x=i) and the responder (x=r). Signing is done using the algorithm associated with the authentication method. Usually this will be RSA or DSS
prf(a, b)	The result of applying pseudo-random function ”a” to data ”b”. One may think of ”a” as a key or as a value that characterizes the function prf; in the latter case it is the index into a family of functions. Each function in the family provides a ”hash” or one-way mixing of the input.
prf(0, b)	The application of a one-way function to data ”b”. The similarity with the previous notation is deliberate and indicates that a single algorithm, e.g. MD5, might will used for both purposes. In the first case a ”keyed” MD5 transform would be used with key ”a”; in the second case the transform would have the fixed key value zero, resulting in a one-way function.
hmac(a, b)	This indicates the HMAC algorithm. pseudo-random function ”a” to data ”b”.

3 Requirements And Considerations

The proposed protocol is designed to address the specific requirements and considerations presented in this section.

3.1 Security Requirements

3.1.1 Data Protection

A secure IM system must permit conversation participants to preserve the following properties of their conversation data:

Property	Description
confidentiality	Conversation data must only be disclosed to authorized recipients
integrity	Conversation data must not be altered
data origin authentication	Recipients must be able to determine the identity of the sender and trust that the message did, in fact, come from the sender. It is important to note that this requirement does not include the requirement of a durable digital signature on conversation data.
replay protection	Recipients must be able to detect and ignore duplicate conversation data.

These are established, traditional goals of information security applied to the conversation data. In the IM environment, these goals protect against the following attacks:

- eavesdropping, snooping, etc.
- masquerading as a conversation participant
- forging messages

Preserving the availability of conversation data is not addressed by this protocol.
Finally, note that this protocol does not concern any authentication between an XMPP node and an XMPP host.

3.1.2 Data Classification

A secure IM system must support a data classification feature through the use of security labeling. Conversation participants must be able to associate a security label with each piece

of conversation data. This label may be used to specify a data classification level for the conversation data.

3.1.3 End To End Protection

It is easy to imagine XMPP systems in which the servers play active, fundamental roles in the protection of conversation data. Such systems could offer many advantages, like:

- allowing the servers to function as credential issuing authorities,
- allowing the servers to function as policy enforcement points.

Unfortunately, such systems have significant disadvantages when one considers the nature of instant messaging:

- Many servers may be un-trusted, public servers.
- In many conversation communities, decisions of trust and membership can only be adequately defined by the members themselves.
- In many conversation communities, membership in the community changes in real time based upon the dynamics of the conversation.
- In many conversation communities, the data classification of the conversation changes in real time based upon the dynamics of the conversation.

Furthermore, the use of gateways to external IM systems is a further complication.
Based on this analysis, we propose that security be entirely controlled in an end to end fashion by the conversation participants themselves via their user agent software.

3.1.4 Trust Issues

Similarly, we believe that trust decisions are in the hands of the conversation participants. A security protocol and appropriate user agents must provide a mechanism for them to make informed decisions.

3.1.5 Cryptosystem Design Considerations

One of the accepted axioms of security is that people must avoid the temptation to start from scratch and produce new, untested algorithms and protocols. History has demonstrated that such approaches are likely to contain flaws and that considerable time and effort are required to identify and address all of these flaws. Any new security protocol should be based on existing, established algorithms and protocols.

3.2 2.2 Environmental Considerations

Any new IM security protocol must integrate smoothly into the existing IM environment, and it must also recognize the nature of the transactions performed by conversation participants. These considerations are especially important:

- dynamic communities. The members of a community are defined in near real time by the existing members.
- dynamic conversations. Conversations may involve any possible subset of the entire set of community members.

Addressing these considerations becomes especially crucial when selecting a conference keying mechanism.

3.3 Usability

Given the requirement to place the responsibility for the protection of conversation data in the hands of the participants, it is imperative to address some fundamental usability issues:

- Overall ease of use is a requirement. For protocol purposes, one implication is that some form of authentication via passphrases is necessary. While we recognize that this can have appalling consequences, especially when we realize that a passphrase may be shared by all of the community members, we also recognize its utility.
- PKIs are well established in many large organizations, and some communities will prefer to rely on credentials issued from these authorities. We must allow the use of existing PKI credentials and trust models rather than impose closed, XMPP-specific credentials.
- Performance must not be negatively impacted. This is particularly true if we consider that most communities are composed of human users conversing in real time. For protocol purposes, one obvious implication is the desire to minimize computationally expensive public key operations.

3.4 Development And Deployment

To successfully integrate into the existing XMPP environment, an extension protocol for security must satisfy the following:

- It must be an optional extension of the existing XMPP protocol.
- It must be transparent to existing XMPP servers.
- It must function gracefully in cases where some community members are not running a user agent that supports the protocol.

- It must make good use of XML.
- It must avoid encumbered algorithms.
- It must be straightforward to implement using widely available cryptographic toolkits.
- It must not require a PKI.

3.5 XML Processing

Since cryptographic operations are applied to data that is transported within an XML stream, the protocol defines a set of rules to ensure a consistent interpretation by all conversation participants.

3.5.1 Transporting Binary Content

Binary data, such as the result of an HMAC, is always transported in an encoded form; the only supported encoding scheme is base64.
Senders MAY include arbitrary white space within the character stream. Senders SHOULD NOT include any other characters outside of the encoding set.
Receivers MUST ignore all characters not in the encoding set.

3.5.2 Transporting Encrypted Content

Encrypted data is always transported in an encoded form; the only supported encoding scheme is base64.
Senders MAY include arbitrary white space within the character stream. Senders SHOULD NOT include any other characters outside of the encoding set.
Receivers MUST ignore all characters not in the encoding set.

3.5.3 Performing HMAC Computation

HMACs are computed over a specific collection of attribute values and character data; when computing an HMAC the following rules apply:

- All characters MUST be HMACed in their pure Unicode form encoded in UTF-16.
- The octets in each character MUST be processed in network byte order.
- For a given element, the attribute values that are HMACed MUST be processed in the specified order regardless of the order in which they appear in the element tag.

- For each attribute value, the computation MUST only include characters from the anticipated set defined in this specification; in particular, white space MUST always be ignored.

- For character data that is represented in a base64 encoded form, the computation MUST only include valid characters from the encoding set.

3.5.4 Performing Cryptographic Operations

The following algorithm is used to encrypt a character string:

- The character string MUST be represented in Unicode encoded in UTF-16.

- The octets in each character MUST be processed in network byte order.

- Appropriate cryptographic algorithm parameters, such as an IV for a block cipher, are generated.

- The octet string derived from the character string is padded with up to 256 octets of arbitrary padding data. There MUST be at least one padding octet. The last octet of the padding MUST indicate the number of preceeding octets in the stream. All padding octets except the last octet SHOULD be randomly generated. When block ciphers are used, the padding MUST result in a stream of octets that is a multiple of the cipher's block size.

4 xmpp:sec namespace

4.1 Elements within the extension

When used to extend existing XMPP construct, the container element is an <x/> element. Each <x/> element could have one <SecurityAssociation/> to refer to a particular security session, one <KeyAgreement/> element which would contain the information for an an exchange of keys. The <x/> element could have its content authenticated by one <Signature/> element which contains the information about signature of information exchanged between two nodes. The <x/> element may contains one <KeyTransport/> element which contains the information about keys to be securely exchanged between two nodes.
When used in an IQ XMPP construct, the container element is a <query/> element. Each <query/> element could have one <SecurityAssociation/> to refer to a particular security session, one <KeyAgreement/> element which would contain the information for an an exchange of keys. The <query/> element could have its content authenticated by one <Signature/> element which contains the information about signature of information exchanged between two nodes. The <query/> element may contains one <KeyTransport/> element which contains the information about keys to be securely exchanged between two nodes.
Each <SecurityAssociation/> element may have <DigestMethod/>, <EncryptionMethod/> and

<SignatureMethod/> elements to specify the actual algorithms set that will be used in a key exchange.
Each <KeyAgreement/> element may have a <DHKeyValue/> and a < DHParamters/> elements to specify the actual data and parameters used in the key exchange. It may also contain a <KA-Nonce/> element to specify a nonce to be used in a key exchange.

4.2 Attributes

Attribute	Meaning
id	The id attribute hold the agreement or security association ID, when present.
length	The length attribute hold the require number of bits in the prime number used to generate the DH key pair.

4.3 Elements

Element	Meaning
SecurityAssociation	The <SecurityAssociation/> tag is used to encapsulate EncryptionMethod, DigestMethod, SignatureMethod data. It is used as a container for the different algorithm definition that are negotiated for the session.
AgreementMethod	The <AgreementMethod/> tag is an optional element that identifies the key agreement algorithm to be applied to an object.
DigestMethod	The <DigestMethod/> tag is an optional element that identifies the digest algorithm to be applied to an object.
DigestValue	The <DigestValue/> tag is an optional element that contains the encoded value of a digest.
EncryptionMethod	The <EncryptionMethod/> tag is an optional element that describes the encryption algorithm applied to the cipher data. If the element is absent, the encryption algorithm must be known by the recipient or the decryption will fail.
Signature	The <Signature/> tag is used to encapsulate signature data. It is used as a container of other XML structures that could come from any namespace.
SignatureMethod	The <SignatureMethod/> tag is an optional element that specifies the algorithm used for signature generation and validation.
SignatureValue	The <SignatureValue/> tag is an optional element that contains the encoded value of a signature.

Element	Meaning
SignedInfo	The <SignedInfo/> tag includes the canonicalization algorithm, a signature algorithm, and one or more references. The SignedInfo element may contain an optional ID attribute that will allow it to be referenced by other signatures and objects. It is in the http://www.w3.org/2000/09/xmldsig# namespace.
KA-Nonce	The <KA-Nonce/> tag is an optional element under <KeyAgreement/> to assure that different keying material is generated even for repeated agreements using the same sender and recipient public keys.
KeyAgreement	The <KeyAgreement/> tag is used to encapsulate key agreement data. It is used as a container of other XML structures that could come from external namespace.
KeyInfo	The <KeyInfo/> tag is used to encapsulate key information data. It enables the recipient to obtain the key needed to validate a signature. <KeyInfo/> may contain keys, names, certificates and other public key management information, such as in-band key distribution or key agreement data. It is used as a container of other XML structures that could come from external namespace.
OriginatorKeyInfo	The <OriginatorKeyInfo/> tag is used to encapsulate originator key information data in a key agreement. It is of type <KeyInfo/> and used as a container of other XML structures that could come from external namespace.
RecipientKeyInfo	The <RecipientKeyInfo/> tag is used to encapsulate recipient key information data in a key agreement. It is It is of type <KeyInfo/> and used as a container of other XML structures that could come from external namespace.
KeyName	The <KeyName/> tag is an optional element of <KeyInfo/> and contains a string value (in which white space is significant) which may be used to communicate a key identifiert to the recipient.
KeyValue	The <KeyValue/> tag contains a single public key that may be useful in validating a signature. The KeyValue element may include externally defined public keys values represented as PCDATA or element types from an external namespace
KeyTransport	The <KeyTransport/> tag is used to encapsulate transported key data. It is used as a container of other XML structures that could come from any namespace.
CarriedKeyName	The <CarriedKeyName/> tag is optional and used to specified the name of the transported key.
DHKeyValue	The <DHKeyValue/> tag is used to encapsulate a Diffie-Hellman key agreement content. It is designed to follow the XML digital signature standard.
DHParameters	The <DHParameters/> tag is used to to encapsulate a Diffie-Hellman key exchange parameters.
Public	The <Public/> tag is holding the actual content of a Diffie-Hellman public key.

Element	Meaning
X509Data	The <X509Data/> tag is an optional element holding one or more identifiers of keys or X509 certificates, or certificates' identifiers or a revocation list. It is in the http://www.w3.org/2000/09/xmldsig# namespace.
PGPData	The <PGPData/> tag is an optional element used to convey information related to PGP public key pairs and signatures on such keys. It is in the http://www.w3.org/2000/09/xmldsig# namespace.
DSAKeyValue	The <DSAKeyValue/> tag is optional and defines a DSA public key inside a <KeyInfo/> element. It is in the http://www.w3.org/2000/09/xmldsig# namespace.
RSAKeyValue	The <RSAKeyValue/> tag is optional and defines a RSA public key inside a <KeyInfo/> element. It is in the http://www.w3.org/2000/09/xmldsig# namespace.

4.4 Attributes values

Element	Attribute	Value	Meaning
KeyAgreement	id	CDATA	The agreement ID
length	null	CDATA	The length of the prime number to be used by default is 768 bits. The length of the prime number to be usedas defined in the IKE Diffie-Hellman groups.
SecurityAssocitation	id	CDATA	The security association ID or cookie for a party in the negotiation.
AgreementMethod	Algorithm	CDATA	The algorythm URI for the key agreement.
DigestMethod	Algorithm	CDATA	The algorythm URI for the digest.
EncryptionMethod	Algorithm	CDATA	The algorythm URI for the encryption.
SignatureMethod	Algorithm	CDATA	The algorythm URI for the signature.

5 Base Key Agreement

5.1 Overview

The base key agreement (BKE) is an implementation of the "Diffie-Hellman Method For Key Agreement" (DH). It allows two nodes to create and share a secret key.
DH is not an encryption mechanism as we normally think of them, in that we do not typically use it to encrypt data. Instead, it is a method to securely exchange the keys that encrypt data. DH accomplishes this secure exchange by creating a "shared secret", sometimes called a "key encryption key", between two nodes. The shared secret then encrypts the symmetric key, or "data encryption key" - DES, Triple DES, CAST, IDEA, Blowfish, etc, for secure transmission.
Two nodes intending to agree on a secret key shall employ the first phase of the agreement independently to produce the public values outputs PV and PV'. The nodes shall exchange the outputs.
The nodes shall then employ the second phase independently with the other nodes's public value as input. The mathematics of Diffie-Hellman key agreement ensure that the resulting outputs SK of the second phase are the same for both entities.
1) First the nodes must get the "Diffie-Hellman parameters". A prime number, 'p' (larger than 2) and "base", 'g', an integer that is smaller than 'p'. They can either be hard coded or fetched from a server.
Diffie-Hellman groups are used to determine the length of the base prime numbers used during the key exchange. The strength of any key derived depends in part on the strength of the Diffie-Hellman group the prime numbers are based on:

- Group 2 (medium) is stronger than Group 1 (low). Group 1 will provide 768 bits of keying material, while Group 2 will provide 1,024 bits. If mismatched groups specified on each peer, negotiation will fail. The group cannot be switched during the negotiation.
- A larger group results in more entropy and therefore a key which is harder to break.

2) The nodes each secretly generate a private number called 'x', which is less than "p - 1".
3) The nodes next generate the ephemeral public keys, 'y'. They are created with the function:
y = gˆx mod p
4) The two nodes now exchange the public keys ('y') and the exchanged numbers are converted into a secret key, 'z'.
z = yˆx mod p
'z' can now be used as the key for whatever encryption method is used to transfer information between the two nodes. Mathematically, the two nodes should have generated the same value for 'z'.
z = (gˆx mod p)ˆx' mod p = (gˆx' mod p)ˆx mod p
All of these numbers are positve integers
xˆy means: x is raised to the y power

xmody means: x is divided by y and the remainder is returned
Suppose two nodes want to agree on a shared secret key to exchange information securely, they will exchange their public keys in order to encrypt that information. To this goal, the transport XMPP packet SHOULD include an extension of the form:

Listing 1: Key agreement Application

```
<x xmlns="xmpp:sec">
<KeyAgreement length="1024">
<DHKeyValue>
<Public&gt;...&lt;/Public>
</DHKeyValue>
</KeyAgreement>
</x>
```

In this extension, the only negotiable parameter is the key length that is passed in the length attribute of the <KeyAgreement/> tag. The length attribute is used to retrieve the DH parameter group and the associated prime and generator values. We are using DH groups derived from the Internet Key Exchange protocol (IKE) which is used by IPSec. A summary of these groups and the associated parameters are described later in this document.

5.1.1 Secure password registration

An example of using this agreement is to send encrypted password on the wire when registering a new user. Registration is the only time a password needs to be exchanged between an XMPP server and a client. Once that has been carried out, then every authentication can be done through digest.
The client uses an empty <x/> element in the request to signal that it supports the XMPP security extension.
The flow between client and server will look like:

Listing 2: Client requests register parameters

```
<iq to="domain" type="get" id="req-0">
<x xmlns="jabber:iq:register">
<x xmlns="xmpp:sec">
<KeyAgreement length="1024"/>
</x>
</query>
</iq>
```

The server will reply to the request by sending out its own ephemeral public key inside the <x/> extension.

Listing 3: Server respond with register parameters

```
<iq from="domain" type="result" id="req-0">
<x xmlns="jabber:iq:register">
<username/>
<password/>
<x xmlns="xmpp:sec">
<KeyAgreement>
<DHKeyValue>
<Public>encoded server public key</Public>
</DHKeyValue>
</KeyAgreement>
</x>
</query>
</iq>
```

The client then generate its own public key, calcultate the shared secret according to the DH method and uses it to encrypt the password accordingly. It includes its own ephemeral public key into the reply to the server inside the <x/> extension.

Listing 4: Client sends register parameters

```
<iq to="domain" type="set" id="req-1">
<x xmlns="jabber:iq:register">
<username>username</username>
<password>encrypted password</password>
<x xmlns="xmpp:sec">
<KeyAgreement>
<DHKeyValue>
<Public>encoded client public key</Public>
</DHKeyValue>
</KeyAgreement>
</x>
</query>
</iq>
```

The server now calculates the shared secret according to the DH method and uses its private key to decrypt the password.

Listing 5: Server acknowledge register

```
<iq to="domain" type="result" id="req-1"/>
```

6 Authenticated Key Agreement

6.1 Introduction

The Diffie-Hellman key agreement algorithm provides a mechanism to allow key establishment in a scalable and secure way. It allows two parties to agree on a shared value without

requiring encryption. An Authenticated Key Agreement (AKE) is a secure protocol ensuring that in addition to securely sharing a secret, the two parties can be certain of each other's identities, even when an active attacker exists.
This AKE uses a hybrid protocol derived from the Internet Key Exchange (IKE) and the OAKLEY key determination protocol. The purpose is to negotiate and provide authenticated key material for security association (SA) in a protected manner. The basic mechanism is the Diffie-Hellman Key Exchange. It provides the following addition to base key agreement:

- it uses weak address validation mechanism (cookies) to avoid denial of service attacks.
- it provides negotiation of mutually agreeable supporting algorithm for the protocol, such as the encryption method, the key derivation method and the authentication method.
- the authentication does not depend on encryption using the DH exponentials, but instead validates the binding of the exponential to the identities of the parties.
- it does not require the computation of the shared exponential before the authentication.
- it provides additional security to the derivation of encryption keys, as it is made to depend not only of the DH algorithm but also on the cryptographic method used to securely authenticate the parties to each other.

This key agreement protocol is used to establish a shared key with an assigned identifier and associated identities for two parties. The resulting common keying information state comprise a key name, secret keying material, the identification of the two parties, and three algorithms for use during authentication:

- encryption for privacy,
- hashing for protecting the integrity of message and for authentication of message fields
- authentication to mutually authenticate the parties

The anti clogging tokens, or cookies, provide a weak form of source address identification for both parties. The cookies exchange can be completed before they perform the expensive computations later in the protocol. The cookies are used also for key naming.

- The construction of the cookies is implementation dependent. It is recommended to make them the result of a one-way function applied to a secret value (changed periodically), and the local and remote addresses. In this way, the cookies remain stateless and expire periodically. Note that this would cause the KEYID's derived from the secret value to also expire, necessitating the removal of any state information associated with it.

- The encryption functions must be cryptographic transforms which guarantee privacy and integrity for the message data. They include any that satisfy this criteria and are defined for use with RFC 2406 [1].
- The one-way hash functions must be cryptographic transform which can be used as either keyed hash (pseudo-random) or non keyed transforms. They include any that are defined for use with RFC2406.
- Where nonces are indicated they will be variable precision integers with an entropy value that match the strength attribute of the DH group used in the exchange.

6.2 Key Exchange Protocol

The main exchange has three optional features:

- stateless cookie exchange,
- perfect forward secrecy for the keying material,
- use of signatures (for non-repudiation).

The two parties can use any combination of these features. The general outline of processing is that the Initiator of the exchange begins by specifying as much information as he wishes in his first message. The Responder replies, supplying as much information as he wishes. The two sides exchange messages, supplying more information each time, until their requirements are satisfied.
The choice of how much information to include in each message depends on which options are desirable. For example, if stateless cookies are not a requirement, and perfect forward secrecy for the keying material are not requirements, and if non- repudiatable signatures are acceptable, then the exchange can be completed in three messages. Additional features may increase the number of roundtrips needed for the keying material determination.
The three components of the key determination are:

- Cookies exchange
- DH half key exchange
- Authentication

The initiator can supply as little information as a bare exchange request, carrying no additional information. On the other hand the initiator can begin by supplying all the necessary information for the responder to authenticate the request and complete the key determination quickly, if the responder choose to accept this method. If not the responder can reply with a minimum amount of information.

[1] RFC 2406: IP Encapsulating Security Payload (ESP) <http://tools.ietf.org/html/rfc2406>.

6.2.1 Aggressive Mode Key Exchange

The following example indicates how two parties can complete a key exchange in three messages. The identities are not secret, the derived keying material is protected by PFS.
By using digital signatures, the two parties will have a proof of communication that can be recorded and presented later to a third party.
The keying material implied by the group exponentials is not needed for completing the exchange. If it is desirable to defer the computation, the implementation can save the ”x” and ”gˆy” values and mark the keying material as ”uncomputed”. It can be computed from this information later.

Initiator	Message content	Responder
□	GRP, CKYi, DHi, EHAo, JIDi, JIDr, Ni, S{JIDi | JIDr | CKYi | 0 | Ni | 0 | GRP | DHi | 0 | EHAo}Ki	□
□	GRP, CKYr, DHr, EHAs, JIDi, JIDr, Nr, S{JIDr | JIDi| | CKYr | CKYi | Nr | Ni | GRP | DHr | DHi | EHAs}Kr	□
□	GRP, CKYi, CKYr, DHi, EHAs, JIDi, JIDr, Ni, Nr, S{JIDi | JIDr | CKYr | CKYi | Ni | Nr | GRP | DHi | DHr | EHAs}KEY	□

The result of this exchange is a key with :

- KEYID = CKYi | CKYr
- sKEYID = prf(Ni | Nr, KEY | CKYi | CKYr).

The Aggressive Mode example is written to suggest that public key technology is used for the signatures. However, a pseudorandom function can be used, if the parties have previously agreed to such a scheme and have a shared key.
If the first proposal in the EHAo list is an ”existing key” method, then the KEYID named in that proposal will supply the keying material for the ”signature” which is computed using the ”H” algorithm associated with the KEYID.

6.2.2 Main Mode Key Exchange

In this exchage the two parties are minimally aggressive; they use the cookie exchange to delay creation of state, and they use perfect forward secrecy to protect the identities.
They use public key encryption for authentication; digital signatures or pre-shared keys can also be used. The Main mode does not change the use of nonces, prf’s, etc., but it does change

how much information is transmitted in each message.
The responder considers the ability of the initiator to repeat CKYr as weak evidence that the message originates from a ”live” correspondent on the network and the correspondent is associated with the initiator’s network address.
The initiator makes similar assumptions when CKYi is repeated to the initiator. All messages must have valid cookies or at least one zero cookie. If both cookies are zero, this indicates a request for a cookie; if only the initiator cookie is zero, it is a response to a cookie request.
Information in messages violating the cookie rules cannot be used for any operations. Note that the Initiator and Responder must agree on one set of EHA algorithms; there is not one set for the Responder and one for the Initiator. The Initiator must include at least MD5 and DES in the initial offer.

Initiator	Message content	Responder
□	CKYi, DHi, EHAo, JIDi, JIDr	□
□	CKYr, DHr, EHAs, JIDi, JIDr	□
□	GRP, CKYi, CKYr, DHi, EHAs, JIDi, JIDr, E{Ni}KEY	□
□	GRP, CKYi, CKYr, DHr, JIDi, JIDr, E{Ni | Nr}KEY, prf(Kir, JIDr | JIDi | GRP | DHr | DHi | EHAs)	□
□	GRP, CKYi, CKYr, DHi, JIDi, JIDr, prf(Kir, JIDi | JIDr | GRP | DHi | DHr | EHAs)	□

Where Kir = prf(0, Ni | Nr)
The result of this exchange is a key with :

- KEYID = CKYi | CKYr
- sKEYID = prf(Kir, KEY | CKYi | CKYr).

6.2.3 Deriving key material for Cryptographic Transforms

The keying material computed by the key exchange should have at least 90 bits of entropy, which means that it must be at least 90 bits in length. This may be more or less than is required for keying the encryption and/or pseudorandom function transforms.
The transforms used should have auxiliary algorithms which take a variable precision integer and turn it into keying material of the appropriate length. The result of either Main Mode or Aggressive Mode is three groups of authenticated keying material:

Context	Keying Material
Digest	sKEYID_d = prf(sKEYID, KEY | CKYi | CKYi | 0)
Authentication	sKEYID_a = prf(sKEYID, SKEYID_d | KEY | CKYi | CKYr | 1)
Encryption	sKEYID_e = prf(sKEYID, SKEYID_a | KEY | CKYi | CKYr | 2)

and agreed upon policy to protect further communications. The values of 0, 1, and 2 above are represented by a single octet. The key used for encryption is derived from sKEYID_e in an algorithm-specific manner.
Encryption keys used to protect the SA are derived from sKEYID_e in an algorithm-specific manner. When SKEYID_e is not long enough to supply all the necessary keying material an algorithm requires, the key is derived from feeding the results of a pseudo-random function into itself, concatenating the results, and taking the highest necessary bits.
For example, if the (ficticious) algorithm MYALGO requires 320-bits of key, and the prf used to generate sKEYID_e only generates 120 bits of material, the key for MYALGO, would be the first 320-bits of Ka, where:
Ka = K1 | K2 | K3 | ...

And

- K1 = prf(sKEYID_e, 0)
- K2 = prf(sKEYID_e, K1)
- K3 = prf(sKEYID_e, K2)
- ...

prf is the HMAC version of the negotiated hash function and 0 is represented by a single octet. Each result of the prf provides 120 bits of material for a total of 360 bits. MYALGO would use the first 320 bits of that 360 bit string.

6.3 Authenticated Key Exchange Application

6.3.1 Main Mode Key Exchange

The intitiator uses a <SecurityAssociation/> element in the request to list all the EHA algorithms that it supports. In addition it provides its own DH ephemeral public key.

- The values of initiator and responder MUST be the JIDs of the two participants, respectively.

- The initiator cookie is prepared by generating a string of 32 random octets (64 random bits). The cookie resulting octets are then encoded into a string of hex characters. The generated value is used as the originator key name for the security association.

- The available set of confidentiality and HMAC cryptographic algorithms is selected. The manner in which these algorithms are selected and all related policy issues are outside the scope of this specification.

- The available set of authentication algorithms is selected. The manner in which these algorithms are selected and all related policy issues are outside the scope of this specification. When the digital signature form of authentication is selected, the relevant end-entity certificate and, optionally, a chain of CA certificates representing a validation path, is assembled and encoded. A set of trusted CA certificates MAY optionally be included via caCertificate elements; if so, the set MUST include the issuer of the initiator's end-entity certificate.

These values are then used to prepare the XML element; this element is transmitted via the existing XMPP iq mechanism:

```
<iq from="initiator@domain" to="responder@domain" type="get" id="req-0
    ">
<query xmlns="xmpp:sec">
<SecurityAssociation id="SA@domain">
<OriginatorKeyInfo>
<KeyName>A32F...245A</KeyName>
</OriginatorKeyInfo>
<EncryptionMethod Algorithm="des"/>
<EncryptionMethod Algorithm="tripledes-cbc"/>
<EncryptionMethod Algorithm="aes128"/>
<EncryptionMethod Algorithm="aes129"/>
<EncryptionMethod Algorithm="aes256"/>
<DigestMethod Algorithm="hmac-md5"/>
<DigestMethod Algorithm="hmac-sha1"/>
<DigestMethod Algorithm="hmac-sha128"/>
<DigestMethod Algorithm="hmac-sha256"/>
<DigestMethod Algorithm="hmac-ripemd128"/>
<DigestMethod Algorithm="hmac-ripemd160"/>
<SignatureMethod Algorithm="dsa-sha1"/>
<SignatureMethod Algorithm="rsa-sha1"/>
</SecurityAssociation>
</query>
</iq>
```

The responder will reply to the request by sending out its own selcted EHA algorithms that will be used in the remainign transaction.

- The values of initiator and responder MUST be the JIDs of the two participants, respectively.
- The responder cookie is prepared by generating a string of 32 random octets (64 random bits). The cookie resulting octets are then encoded into a string of hex characters. The generated value is used as the recipient key name for the security association..
- The algorithms attributes are checked against the values supported by the user agent. If the receiver is not able to select one set out of the proposed algorithms, an error code 406-Unacceptable is returned.
- The desired confidentiality and HMAC cryptographic algorithms are selected from the proposed set. The manner in which these algorithms are selected and all related policy issues are outside the scope of this specification.
- The desired authentication algorithm is selected from the proposed set. The manner in which this algorithm is selected and all related policy issues are outside the scope of this specification. In the digital signature case, the responder's end-entity certificate MUST be issued by one of the trusted CAs listed in the session1 PDU or by the same issuer as the initiator's end-entity certificate. If the responder does not have acceptable credentials, an error code of 401-Unuthorized occurs.

```
<iq from="responder@domain" to="initiator@domain" type="result" id="
   req-0">
<query xmlns="xmpp:sec">
<SecurityAssociation id="SA@domain">
<OriginatorKeyInfo>
<KeyName>A32F...245A</KeyName>
</OriginatorKeyInfo>
<RecipientKeyInfo>
<KeyName>324A...BF24</KeyName>
</RecipientKeyInfo>
<EncryptionMethod Algorithm="tripledes-cbc"/>
<DigestMethod Algorithm="hmac-sha1"/>
<SignatureMethod Algorithm="rsa-sha1"/>
</SecurityAssociation>
</query>
</iq>
```

The intitiator provides its own DH ephemeral public key.

- The values of initiator and responder MUST be the JIDs of the two participants, respectively.
- The initator and responder cookies are used as the originator key name and the recipient key name for the security association..

- A Diffie-Hellman group is selected. The appropriate values for g and p will be used to generate the initiator's public key.
- An ephemeral private key, x, is generated using g and p for the selected group. This key MUST be generated using an appropriate random number source. The corresponding public key, g^x, is generated and encoded.

```
<iq from="initiator@domain" to="responder@domain" type="get" id="req-1
   ">
<query xmlns="xmpp:sec">
<SecurityAssociation id="SA@domain">
<OriginatorKeyInfo>
<KeyName>A32F...245A</KeyName>
</OriginatorKeyInfo>
<RecipientKeyInfo>
<KeyName>324A...BF24</KeyName>
</RecipientKeyInfo>
</SecurityAssociation>
<KeyAgreement length="1024">
<DHKeyValue>
<Public>
... encoded initiator public key
</Public>
</DHKeyValue>
</KeyAgreement>
</query>
</iq>
```

The responder check the validity of the parameters and eventualy replies with its own DH ephemeral public key.

- The values of initiator and responder MUST be the JIDs of the two participants, respectively.
- The initator and responder cookies are checked; a mismatch results in an error code of 406 - Unacceptable .
- The Diffie-Hellman group is checked against the values supported by the user agent. An unsupported group results in an error code of 406 - Unacceptable
- An ephemeral private key, y, is generated using g and p for the group indicated by the PDU. This key MUST be generated using an appropriate random number source. The corresponding public key, g^y, is generated and encoded.

```
<iq from="responder@domain" to="initiator@domain" type="result" id="
   req-1">
<query xmlns="xmpp:sec">
```

```
<SecurityAssociation id="SA@domain">
<OriginatorKeyInfo>
<KeyName>A32F...245A</KeyName>
</OriginatorKeyInfo>
<RecipientKeyInfo>
<KeyName>324A...BF24</KeyName>
</RecipientKeyInfo>
</SecurityAssociation>
<KeyAgreement length="1024">
<DHKeyValue>
<Public>
... encoded initiator public key
</Public>
</DHKeyValue>
</KeyAgreement>
</query>
</iq>
```

The intitiator provides its nonce encrypted with the agreed algorithm and the public key of the responder.

- The values of initiator and responder MUST be the JIDs of the two participants, respectively.
- The initator and responder cookies are checked; a mismatch results in the procedure being aborted.
- The initiator nonce is prepared by first generating a string of 20 random octets (160 random bits). The nonce is then encrypted using the selected encryption algorithm and the shared secret key. The resulting octets are then encoded into a string of base64 characters.

```
<iq from="initiator@domain" to="responder@domain" type="get" id="req-1
   ">
<query xmlns="xmpp:sec">
<SecurityAssociation id="SA@domain">
<OriginatorKeyInfo>
<KeyName>A32F...245A</KeyName>
</OriginatorKeyInfo>
<RecipientKeyInfo>
<KeyName>324A...BF24</KeyName>
</RecipientKeyInfo>
</SecurityAssociation>
<KeyAgreement>
<KA-Nonce>
<EncryptedData xmlns="http://www.w3.org/2001/04/xmlenc#"
    Type="http://www.w3.org/2001/04/xmlenc#Element">
<CipherData>
```

```
<CipherValue>
... encoded encrypted initiator nonce
</CipherValue>
</CipherData>
</EncryptedData>
</KA-Nonce>
</KeyAgreement>
</query>
</iq>
```

The responder replies with the concatenation of its own nonce and the initiator nonce encrypted with the agreed algorithm and the public key of the initiator. The packet is authenticated using the agreed signature algorithm.

- The values of initiator and responder MUST be the JIDs of the two participants, respectively.
- The initator and responder cookies are checked; a mismatch results in an error code of 401 - Unauthorized.
- The initiator nonce is decrypted using the responder private key.
- The responder nonce is prepared by first generating a string of 20 random octets (160 random bits). It is then apended to the initiator nonce and the result encrypted using the selected encryption algorithm and the shared secret key. The resulting octets are then encoded into a string of base64 characters.
- Based on the selected authentication algorithm, the responder's authenticator is constructed. A digital signature requires calculating:

1. Kir = hmac (0, initiator's nonce | responder's nonce)
2. EHAs = (Encryption algorithm URI | Digest algorithm URI | Signature algorithm URI)
3. HASH_R = hmac (Kir, JID responder | JID initiator | length of DH group | responder DH public key | initiator DH public key | EHAs)

- HASH_R is encoded in base64.

```
<iq from="responder@domain" to="initiator@domain" type="result" id="
    req-1">
<query xmlns="xmpp:sec">
<SecurityAssociation id="SA@domain">
<OriginatorKeyInfo>
<KeyName>A32F...245A</KeyName>
</OriginatorKeyInfo>
<RecipientKeyInfo>
<KeyName>324A...BF24</KeyName>
```

```
</RecipientKeyInfo>
</SecurityAssociation>
<KeyAgreement>
<KA-Nonce>
<EncryptedData xmlns="http://www.w3.org/2001/04/xmlenc#"
    Type="http://www.w3.org/2001/04/xmlenc#Element">
<CipherData>
<CipherValue>
... encoded encrypted responder nonce
</CipherValue>
</CipherData>
</EncryptedData>
</KA-Nonce>
</KeyAgreement>
<Signature
    xmlns="http://www.w3.org/2000/09/xmldsig#"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.w3.org/2000/09/xmldsig#xmldsig-core
       -schema.xsd">
<SignaturetValue>
... encoded signature value
</SignatureValue>
</Signature>
</query>
</iq>
```

The initiator authenticate the keying material using the agreed signature algorithm.

- The values of initiator and responder MUST be the JIDs of the two participants, respectively.
- The initator and responder cookies are checked; a mismatch results in the procedure being aborted.
- The concatenation of the responder and initiator nonce is decrypted using the initiator private key. The original initiator nonce is compared to the result. An invalid nonce results in aborting the procedure. Otherwise the result is used to generate Kir
- Based on the selected authentication algorithm, the responder's authenticator is constructed. A digital signature requires calculating:

1. Kir = hmac (0, initiator's nonce | responder's nonce)
2. EHAs = (Encryption algorithm URI | Digest algorithm URI | Signature algorithm URI)
3. HASH_R = hmac (Kir, JID responder | JID initiator | length of DH group | responder DH public key | initiator DH public key | EHAs)

- The authenticator is verified. A failure results in aborting the procedure.

- Based on the selected authentication algorithm, the initiator's authenticator is constructed. A digital signature requires calculating:

1. HASH_I = hmac (Kir, JID initiator | JID responder | length of DH group | initiator DH public key | responder DH public key | EHAs)

```
<iq from="initiator@domain" to="responder@domain" type="set" id="req-2
   ">
<query xmlns="xmpp:sec">
<SecurityAssociation id="SA@domain">
<OriginatorKeyInfo>
<KeyName>A32F...245A</KeyName>
</OriginatorKeyInfo>
<RecipientKeyInfo>
<KeyName>324A...BF24</KeyName>
</RecipientKeyInfo>
</SecurityAssociation>
<Signature
    xmlns="http://www.w3.org/2000/09/xmldsig#"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.w3.org/2000/09/xmldsig#
xmldsig-core-schema.xsd">
<SignaturetValue>
<p>... encoded signature value</p>
</SignatureValue>
</Signature>
</query>
</iq>
```

The responder acknowledge the keying material.

- The values of initiator and responder MUST be the JIDs of the two participants, respectively.
- The initator and responder cookies are checked; a mismatch results in an error code of 401 - Unauthorized.
- Based on the selected authentication algorithm, the initiator's authenticator is constructed. A digital signature requires calculating:

1. HASH_I = hmac (Kir, JID initiator | JID responder | length of DH group | initiator DH public key | responder DH public key | EHAs)

- The authenticator is verified. A failure results in an error code of 406 - Unacceptable.

```
<iq from="responder@domain" to="initiator@domain" type="result" id="
   req-2"/>
```

6.3.2 Aggressive Mode Key Exchange

The intitiator uses <SecurityAssociation/> element in the request to list all the EHA algorithms that it supports. In addition it provides its own DH ephemeral public key. The message is signed with its own private key.

- The values of initiator and responder MUST be the JIDs of the two participants, respectively.
- The initiator cookie is prepared by generating a string of 32 random octets (64 random bits). The cookie resulting octets are then encoded into a string of hex characters. The generated value will be used as identifier for the initiator leg of the security association.
- The available set of confidentiality and HMAC cryptographic algorithms is selected. The manner in which these algorithms are selected and all related policy issues are outside the scope of this specification.
- The available set of authentication algorithms is selected. The manner in which these algorithms are selected and all related policy issues are outside the scope of this specification. When the digital signature form of authentication is selected, the relevant end-entity certificate and, optionally, a chain of CA certificates representing a validation path, is assembled and encoded. A set of trusted CA certificates MAY optionally be included via caCertificate elements; if so, the set MUST include the issuer of the initiator's end-entity certificate.
- A Diffie-Hellman group is selected. The appropriate values for g and p will be used to generate the initiator's public key.
- An ephemeral private key, x, is generated using g and p for the selected group. This key MUST be generated using an appropriate random number source. The corresponding public key, gˆx, is generated and encoded.
- The initiator nonce is prepared by first generating a string of 20 random octets (160 random bits). The resulting octets are then encoded into a string of base64 characters.
- Based on the selected authentication algorithm, the initiator's authenticator is constructed. A digital signature requires calculating:

1. EHAs = (Encryption algorithm URI | Digest algorithm URI | Signature algorithm URI)
2. SIGN_I = S (JID initiator | JID responder | initiator cookie | 0 | initiator nonce | 0 | length of DH group | initiator DH public key | 0 | EHAs) initiator private key

- SIGN_I is encoded in base64.

```
<iq from="initiator@domain" to="responder@domain" type="get" id="req-0
   ">
<query xmlns="xmpp:sec">
<SecurityAssociation id="SA@domain">
<OriginatorKeyInfo>
<KeyName>A32F...245A</KeyName>
<RSAKeyValue>
<p>... encoded initiator public key value </p>
</RSAKeyValue>
</OriginatorKeyInfo>
<EncryptionMethod Algorithm="tripledes-cbc"/>
<DigestMethod Algorithm="hmac-sha1"/>
<SignatureMethod Algorithm="rsa-sha1"/>
</SecurityAssociation>
<KeyAgreement length="1024">
<DHKeyValue>
<Public>
<p>... encoded initiator public key</p>
</Public>
</DHKeyValue>
<KA-Nonce>
<p>... encoded initiator nonce value>
</KA-Nonce>
</KeyAgreement>
<Signature
    xmlns="http://www.w3.org/2000/09/xmldsig#"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.w3.org/2000/09/xmldsig#xmldsig-core
       -schema.xsd">
<SignatureValue>
<p>... encoded initiator signature value</p>
</SignatureValue>
</Signature>
</query>
</iq>
```

The responder will reply to the request by acknowledging the selected EHA algorithms. In addition, it provides its own DH ephemeral public key. The message is signed with its own private key.

- The values of initiator and responder MUST be the JIDs of the two participants, respectively.
- The Diffie-Hellman group is checked against the values supported by the user agent. An unsupported group results in an error code of 406 - Unacceptable
- Based on the selected authentication algorithm, the initiator's authenticator is constructed. A digital signature requires calculating:

1. EHAs = (Encryption algorithm URI | Digest algorithm URI | Signature algorithm URI)
2. SIGN_I = S (JID initiator | JID responder | initiator cookie | 0 | initiator nonce | 0 | length of DH group | initiator DH public key | 0 | EHAs) initiator public key

- The authenticator is verified. A failure results in an error code of 401 - Unauthorized.
- The responder cookie is prepared by generating a string of 32 random octets (64 random bits). The cookie resulting octets are then encoded into a string of hex characters. The generated value will be used as identifier for the responder leg of the security association.
- An ephemeral private key, y, is generated using g and p for the group indicated by the PDU. This key MUST be generated using an appropriate random number source. The corresponding public key, g^y, is generated and encoded.
- The responder nonce is prepared by first generating a string of 20 random octets (160 random bits). The resulting octets are then encoded into a string of base64 characters.
- Based on the selected authentication algorithm, the responder's authenticator is constructed. A digital signature requires calculating:

1. SIGN_R = S (JID responder | JID initiator | responder cookie | initiator cookie | responder nonce | initiator nonce | length of DH group | responder DH public key | initiator DH public key | EHAs) responder private key

- SIGN_R is encoded in base64.

```
<iq from="responder@domain" to="initiator@domain" type="result" id="
    req-0">
<query xmlns="xmpp:sec">
<SecurityAssociation id="SA@domain">
<OriginatorKeyInfo>
<KeyName>A32F...245A</KeyName>
</OriginatorKeyInfo>
<RecipientKeyInfo>
<KeyName>324A...BF24</KeyName>
<RSAKeyValue>
<p>... encoded responder public key value </p>
</RSAKeyValue>
</RecipientKeyInfo>
</SecurityAssociation>
<KeyAgreement length="1024">
<DHKeyValue>
<Public>
<p>... encoded responder public key</p>
</Public>
</DHKeyValue>
```

```
<KA-Nonce>
<p>... encoded responder nonce value>
</KA-Nonce>
</KeyAgreement>
<Signature
    xmlns="http://www.w3.org/2000/09/xmldsig#"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.w3.org/2000/09/xmldsig#
xmldsig-core-schema.xsd">
<SignatureValue>
<p>... encoded responder signature value</p>
</SignatureValue>
</Signature>
</query>
</iq>
```

The initiator authenticate the keying material using the agreed signature algorithm.

- The values of initiator and responder MUST be the JIDs of the two participants, respectively.
- Based on the selected authentication algorithm, the initiator's authenticator is constructed. A digital signature requires calculating:

1. EHAs = (Encryption algorithm URI | Digest algorithm URI | Signature algorithm URI)
2. SIGN_R = S (JID responder | JID initiator | responder cookie | initiator cookie | responder nonce | initiator nonce | length of DH group | responder DH public key | initiator DH public key | EHAs) responder public key

- The authenticator is verified. A failure results in the procedure being aborted.
- Based on the selected authentication algorithm, the authenticator is constructed. A digital signature requires calculating:

1. SIGN_I = S (JID initiator | JID responder | initiator cookie | responder cookie | initiator nonce | responder nonce | length of DH group | initiator DH public key | responder DH public key | EHAs) shared secret key

- SIGN_I is encoded in base64.

```
<iq from="initiator@domain" to="responder@domain" type="set" id="req-2
    ">
<query xmlns="xmpp:sec">
<SecurityAssociation id="SA@domain">
<OriginatorKeyInfo>
<KeyName>A32F...245A</KeyName>
```

```
</OriginatorKeyInfo>
<RecipientKeyInfo>
<KeyName>324A...BF24</KeyName>
</RecipientKeyInfo>
</SecurityAssociation>
<Signature
    xmlns="http://www.w3.org/2000/09/xmldsig#"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.w3.org/2000/09/xmldsig#
xmldsig-core-schema.xsd">
<SignaturetValue>
<p>... encoded signature value</p>
</SignatureValue>
</Signature&gt;&lt;/query>
</iq>
```

The responder acknowledge the keying material.

- The values of initiator and responder MUST be the JIDs of the two participants, respectively.
- Based on the selected authentication algorithm, the initiator's authenticator is constructed. A digital signature requires calculating:

1. EHAs = (Encryption algorithm URI | Digest algorithm URI | Signature algorithm URI)
2. SIGN_I = S (JID initiator | JID responder | initiator cookie | responder cookie | initiator nonce | responder nonce | length of DH group | initiator DH public key | responder DH public key | EHAs) shared secret key

- The authenticator is verified. A failure results in an error code of 406 - Unacceptable.

```
<iq from="responder@domain" to="initiator@domain" type="result" id="
   req-2"/>
```

7 Key Transport

7.1 Conversation Key Transport

Conversation keys are transported using the symmetric key wrap feature of XML Encryption embedded in the KeyTransport PDU.

7.1.1 Key transport exchange

Key transport follow a previous Security Association establishment and the generation of a shared secret key through a key agreement.

Initiator	Message content	Responder
□	JIDi, JIDr, CKYe, sKEYID_e, CKYa, sKEYID_a, CKYd, sKEYID_d, S{JIDi | JIDr | Ni | Nr | CKYe | sKEYID_e | CKYa | sKEYID_a | CKYd | sKEYID_d }KEY	□

7.1.2 Generating And Sending a Conversation Key Transport PDU

The Key Transport assumes that a security association be negotiated for the purpose of securely transporting conversation keys. The sender's user agent employs the following algorithm to generate the keyTransport PDU:

- The values of initiator and responder MUST be the JIDs of the two participants who negotiated the security association, respectively.
- The security association identifier is assembled.
- The payload, which consists of the confidentiality key sKEYID_e, digest key sKEYID_d and the integrity key sKEYID_a , is wrapped in instances of xenc:EncryptedKey as follows:

1. The Type attribute of the xenc:EncryptedKey element MUST indicate 'content'.
2. The Id, MimeType and Encoding attributes of the xenc:EncryptedKey element MUST NOT be present.
3. The xenc:EncryptionMethod element MUST be present, and the Algorithm attribute MUST indicate a valid symmetric key wrap algorithm. Furthermore, the algorithm MUST be the same as was negotiated for the security association.
4. The ds:KeyInfo element MUST NOT be present. The key to use is the shared secret KEY of the negotiated security association.
5. The xenc:ContainedKeyName element MUST be present.
6. The xenc:CipherData element MUST be present, and it MUST use the CipherValue choice.

- The HMAC is computed using KEY of the negotiated security association. A digital signature requires calculating:

1. HMAC = prf(KEY, JIDi | JIDr | Ni | Nr | sKEYID_e | key name | sKEYID_e | sKEYID_a key name | sKEYID_a | sKEYID_d key name | sKEYID_d)

These values are then used to prepare the XML KeyTransport element; this element is transmitted via the existing XMPP iq mechanism. The order in which the keys are in the payload is significant. The first mandatory key is sKEYID_e. The second optional key is sKEYID_a. And the last optional key is sKEYID_d.

```
<iq from="initiator@domain" to="responder@domain" type="set" id="req-0
    ">
<query xmlns="xmpp:sec">
<SecurityAssociation id="negotiated␣SA␣id"/>
<KeyTransport>
<EncryptedKey xmlns='http://www.w3.org/2001/04/xmlenc#'
    Type='http://www.w3.org/2001/04/xmlenc#Content'>
<ContainedKeyName>A32F...245A324A...BF24-enc&lt;/ContainedKeyName>
<EncryptionMethod Algorithm="kw-tripledes"/>
<CipherData>
<CipherValue>
<p>... encoded encrypted confidentiality key</p>
</CipherValue>
</CipherData>
</EncryptedKey>
<EncryptedKey xmlns='http://www.w3.org/2001/04/xmlenc#'
    Type='http://www.w3.org/2001/04/xmlenc#Content'>
<ContainedKeyName>A32F...245A324A...BF24-auth&lt;/ContainedKeyName>
<EncryptionMethod Algorithm="kw-tripledes"/>
<CipherData>
<CipherValue>
<p>... encoded encrypted confidentiality key</p>
</CipherValue>
</CipherData>
</EncryptedKey>
<EncryptedKey xmlns='http://www.w3.org/2001/04/xmlenc#'
    Type='http://www.w3.org/2001/04/xmlenc#Content'>
<ContainedKeyName>A32F...245A324A...BF24-dig&lt;/ContainedKeyName>
<EncryptionMethod Algorithm="kw-tripledes"/>
<CipherData>
<CipherValue>
<p>... encoded encrypted confidentiality key</p>
</CipherValue>
</CipherData>
</EncryptedKey>
</KeyTransport>
<Signature
    xmlns="http://www.w3.org/2000/09/xmldsig#"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.w3.org/2000/09/xmldsig#
```

```
xmldsig-core-schema.xsd">
<SignedInfo>
<SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#dsa"/>
<DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
</SignedInfo>
<SignaturetValue>
<p>... encoded signature value</p>
</SignatureValue>
</Signature>
</query>
</iq>
```

7.1.3 Receiving and Processing the Conversation Key Transport PDU

The receiver's user agent employs the following algorithm to process each KeyTransport PDU:

- The values of initiator, responder, and security association id MUST indicate an existing security association. An invalid security association results in an error of 401 - Unauthorized.
- The payload, which consists of the confidentiality key sKEYID_e, digest key sKEYID_d and the intergrity key sKEYID_a, is unwrapped. Any failures result in an error code of 406-Unacceptable.
- The body of the HMAC element is decoded into the actual HMAC octet string.
- The HMAC is computed using KEY of the security association. A digital signature requires calculating:

1. HMAC = prf(KEY, JIDi | JIDr | Ni | Nr | sKEYID_e key name | sKEYID_e | sKEYID_a key name | sKEYID_a | sKEYID_d key name | sKEYID_d)

- The HMAC is validated. An invalid HMAC results in an error code of 406-Unacceptable.
- The keys are added to the user agent's key store.

If any errors occur during processing, the error is communicated via the existing XMPP mechanism:

```
<iq from="responder@domain" to="initiator@domain" type="result" id="
    req-0"/>
```

7.2 Public Key Transport

Public keys are transported embedded in the KeyTransport PDU.

7.2.1 Certificate transport

X509 certificates can also be transported in existing XMPP message. The following example uses a presence subscription packet as the vehicle PDU. The subscribee public key and certificate are sent to the initiator of a presence subscription.

```
<presence from="responder@domain" to="initiator@domain" type="
    subscribed">
<x xmlns="xmpp:sec">
<KeyTransport>
<KeyInfo>
<X509Data>
<X509IssuerSerial>
<X509IssuerName>
    CN=TAMURA Kent, OU=TRL, O=IBM, L=Yamato-shi, ST=Kanagawa, C=JP
</X509IssuerName>
<X509SerialNumber&gt;12345678&lt;/X509SerialNumber>
</X509IssuerSerial>
<X509SKI&gt;31d97bd7&lt;/X509SKI>
</X509Data>
<X509Data>
<X509SubjectName&gt;Subject of Certificate B&lt;/X509SubjectName>
</X509Data>
<X509Data>
<X509Certificate&gt;MIICXTCCA..&lt;/X509Certificate>
<X509Certificate&gt;MIICPzCCA...&lt;/X509Certificate>
<X509Certificate&gt;MIICSTCCA...&lt;/X509Certificate>
</X509Data>
</KeyInfo>
</KeyTransport>
</x>
</presence>
```

7.2.2 Other Public Keys Transport

- The values of initiator and responder MUST be the JIDs of the two participants in the exchange, respectively.
- The payload, which consists of the public key of the responder is assembled.
- The SIGN is computed using the private key of the responder. A digital signature requires calculating:

1. SIGN = S (JIRi | JIDR | Kr name | Kr) responder private key

These values are then used to prepare the XML KeyTransport element; this element is transmitted via an existing XMPP mechanism. In the following example, the responder public

key is sent to the initiator of a presence subscription.

```
<presence from="responder@domain" to="initiator@domain" type="
    subscribed">
<x xmlns="xmpp:sec">
<KeyTransport>
<KeyInfo>
<KeyName>responder@domain key name</KeyName>
<RSAKeyValue>
<p>... encoded responder public key value </p>
</RSAKeyValue>
</KeyInfo>
</KeyTransport>
<Signature
    xmlns="http://www.w3.org/2000/09/xmldsig#"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.w3.org/2000/09/xmldsig#xmldsig-core
       -schema.xsd">
<SignedInfo>
<SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#dsa"/>
</SignedInfo>
<SignaturetValue>
<p>... encoded signature value</p>
</SignatureValue>
</Signature>
</x>
</presence>
```

8 Message Protection

8.1 Overview

The ultimate goal is the protection of conversation data. The protocol exchanges described above allow the conversation participants to cryptographically protect their conversation data using the conversation keys that they share.

8.2 Message Protection Mechanism

A protected message is defined as a traditional XMPP message whose body content is extended to include the transport of a cryptographically protected message body. The two key features are:

- the usual body element contains some arbitrary text.

- the message contains a XMPP x element defining the xmpp:sec namespace; this element transports the protected message.

This mechanism has the advantages of allowing transparent integration with existing XMPP servers and existing XMPP clients.

8.3 Generating And Sending the Protected Message PDU

The sender's user agent employs the following algorithm to generate the protected Message PDU:

- The security association identifier is assembled.
- The actual message body is encoded into a character string corresponding to a XMPP message body element. This character string is then wrapped in an instance of xenc:EncryptedData as follows:

1. The Type attribute of the xenc:EncryptedData element MUST indicate 'element'.
2. The Id, MimeType and Encoding attributes of the xenc:EncryptedData element MUST NOT be present.
3. The xenc:EncryptionMethod element MUST be present, and the Algorithm attribute MUST indicate a valid block encryption algorithm.
4. The ds:KeyInfo element MUST NOT be present. The key to be used is the confidentiality key indicated by the convId attribute.
5. The xenc:CipherData element MUST be present, and it MUST use the CipherValue choice.

- Using the HMAC key indicated by the security association, the HMAC is computed. A digital signature requires calculating:

1. HMAC = prf(sKEYID_d, key name | JID from | JID to | message id | message type | message thread | message subject | message body)

These values are then used to prepare the XML protected element; this element is transmitted via the existing XMPP message mechanism:

```
<message from="initiator@domain" to="responder@_domain" id="msg-0">
<body>
The real body is protected.
</body>
<x xmlns="xmpp:security">
<EncryptedData xmlns="http://www.w3.org/2001/04/xmlenc#"
```

```
    Type="http://www.w3.org/2001/04/xmlenc#Element">
<KeyInfo>
<KeyName>A32F2...45A324A...BF24-enc</KeyName>
</KeyInfo>
<EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#
   tripledes-cbc"/>
<CipherData>
<CipherValue>
... encoded encrypted message content
</CipherValue>
</CipherData>
</EncryptedData>
<Signature
    xmlns="http://www.w3.org/2000/09/xmldsig#"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.w3.org/2000/09/xmldsig#xmldsig-core
       -schema.xsd">
<SignedInfo>
<SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#dsa"/>
</SignedInfo>
<KeyInfo>
<KeyName>A32F2...45A324A...BF24-auth</KeyName>
</KeyInfo>
<SignatureValue>
... encoded signature value
</SignatureValue>
</Signature>
</x>
</message>
```

8.4 Receiving and Processing the Protected Message PDU

The receiver's user agent employs the following algorithm to process each protectedMessage PDU:

- The values of initiator, responder, and key name MUST indicate an existing security association. An invalid security association results in an error of 406-Unacceptable.
- The payload, which consists of the actual message body, is unwrapped. Any failures result in an error code of 406-Unacceptable.
- The body of the HMAC element is decoded into the actual HMAC octet string.
- Using the HMAC key indicated by the security association, the HMAC is computed. A digital signature requires calculating:

1. HMAC = prf(sKEYID_d, key name | JID from | JID to | message id | message type | message thread | message subject | message body)

- The HMAC is validated. An invalid HMAC results in an error code of 406-Unacceptable.

If any errors occur during processing, the error is communicated via the existing XMPP mechanism:

9 Algorithms

This section discusses algorithms used with the XMPP security specification. Entries contain the identifier to be used as the value of the Algorithm attribute of the EncryptionMethod element or other element representing the role of the algorithm, a reference to the formal specification, definitions for the representation of keys and the results of cryptographic operations where applicable, and general applicability comments.
The table below lists the categories of algorithms. Within each category, a brief name, the level of implementation requirement, and an identifying URI are given for each algorithm.

Category	Algorithm	URI
Block Encryption	TRIPLEDES	tripledes-cbc
AES-128	aes128-cbc	
AES-192	aes192-cbc	
AES-256	aes256-cbc	
Key Transport	RSA-v1.5	rsa-1_5
RSA-OAEP	rsa-oaep-mgf1p	
Symmetric Key Wrap	TRIPLEDES KeyWrap	kw-tripledes
AES-128 KeyWrap	kw-aes128	
AES-256 KeyWrap	kw-aes256	
AES-192 KeyWrap	kw-aes192	
Message Digest	MD5	md5
SHA1	sha1	
SHA256	sha256	
SHA512	sha512	
RIPEMD-160	ripemd160	
HMAC-MD5	hmac-md5	
HMAC-SHA1	hmac-sha1	
HMAC-SHA128	hmac-sha128	
HMAC-SHA256	hmac-sha256	
Signature	DSAwithSHA1 (DSS)	dsa-sha1
RSAwithSHA1	rsa-sha1	

10 PKCS #3: Diffie-Hellman Key-Agreement Standard

An RSA Laboratories Technical Note Version 1.4 Revised November 1, 1993[2]

10.1 Scope

This standard describes a method for implementing Diffie-Hellman key agreement, whereby two parties, without any prior arrangements, can agree upon a secret key that is known only to them (and, in particular, is not known to an eavesdropper listening to the dialogue by which the parties agree on the key). This secret key can then be used, for example, to encrypt further communications between the parties.
The intended application of this standard is in protocols for establishing secure connections, such as those proposed for OSI's transport and network layers [ISO90a][ISO90b].
Details on the interpretation of the agreed-upon secret key are outside the scope of this standard, as are details on sources of the pseudorandom bits required by this standard.

10.2 References

X.208 CCITT. Recommendation X.208: Specification of Abstract Syntax Notation One (ASN.1). 1988.

X.509 CCITT. Recommendation X.509: The Directory—Authentication Framework. 1988.

DH76
W. Diffie and M.E. Hellman. New directions in cryptography. IEEE Transactions on Information Theory, IT-22:644-654, 1976.

Sch90
C.P. Schnorr. Efficient identification and signatures for smart cards. In G. Brassard, editor, Advances in Cryptology—CRYPTO '89 Proceedings, volume 435 of Lecture Notes in Computer Science, pages 239-251. Springer-Verlag, New York, 1990.

ISO90a
ISO. JTC1/SC6/N6285: Draft Transport Layer Security Protocol. Draft, November 1990.

ISO90b
ISO. JTC1/SC6/N2559: Draft Network Layer Security Protocol. Draft, September 1990.

10.3 Definitions

For the purposes of this standard, the following definitions apply.
AlgorithmIdentifier: A type that identifies an algorithm (by object identifier) and any associated parameters. This type is defined in X.509.
ASN.1: Abstract Syntax Notation One, as defined in X.208.
Diffie-Hellman parameters: Prime and base.
Diffie-Hellman: The Diffie-Hellman key-agreement protocol, elsewhere called "exponential

[2] upersedes June 3, 1991 version, which was also published as NIST/OSI Implementors' Workshop document SEC-SIG-91-19. PKCS documents are available by electronic mail to <pkcs@rsa.com>.

key agreement," as defined in [DH76].

10.4 Symbols and abbreviations

Upper-case italic symbols (e.g., PV) denote octet strings; lower-case italic symbols (e.g., g) denote integers.

PV public value	p prime
PV' other's public value	x private value
SK secret key	x' other's private value
G base	y integer public value
K length of prime in octets	y' other's integer public value
L length of private value in bits	z integer secret key
mod n modulo n	

10.5 General overview

The next four sections specify parameter generation, two phases of Diffie-Hellman key agreement, and an object identifier.
A central authority shall generate Diffie-Hellman parameters, and the two phases of key agreement shall be performed with these parameters. It is possible that more than one instance of parameters may be generated by a given central authority, and that there may be more than one central authority. Indeed, each entity may be its own central authority, with different entities having different parameters. The algorithm identifier for Diffie-Hellman key agreement specifies which Diffie-Hellman parameters are employed.
Two entities intending to agree on a secret key shall employ the first phase independently to produce outputs PV and PV', the public values. The entities shall exchange the outputs.
The entities shall then employ the second phase independently with the other entity's public value as input. The mathematics of Diffie-Hellman key agreement ensure that the outputs SK of the second phase are the same for both entities.

10.6 Parameter generation

This section describes Diffie-Hellman parameter generation.
A central authority shall select an odd prime p. The central authority shall also select an integer g, the base, that satisfies $0 < g < p$. The central authority may optionally select an integer l, the private-value length in bits, that satisfies $2^{l-1} \leq p$.
The length of the prime p in octets is the integer k satisfying

$28(k-1) \le p < 28k$.

10.6.1 Notes

1. The cost of some methods for computing discrete logarithms depends on the the length of the prime, while the cost of others depends on the length of the private value. The intention of selecting a private-value length is to reduce the computation time for key agreement, while maintaining a given level of security. A similar optimization is suggested by Schnorr [Sch90].
2. Some additional conditions on the choice of prime, base, and private-value length may well be taken into account in order to deter discrete logarithm computation. These security conditions fall outside the scope of this standard.

10.7 Phase I

This section describes the first phase of Diffie-Hellman key agreement.
The first phase consists of three steps: private-value generation, exponentiation, and integer-to-octet-string conversion. The input to the first phase shall be the Diffie-Hellman parameters. The output from the first phase shall be an octet string PV, the public value; and an integer x, the private value.
This phase is performed independently by the two parties intending to agree on a secret key.

10.7.1 Private-value generation

An integer x, the private value, shall be generated privately and randomly. This integer shall satisfy $0 < x < p-1$, unless the central authority specifies a private-value length l, in which case the integer shall satisfy $2^{l-1} \le x < 2^l$.

10.7.2 Exponentiation

The base g shall be raised to the private value x modulo p to give an integer y, the integer public value.
$y = g^x \bmod p$, $0 < y < p$.
This is the classic discrete-exponentiation computation.

10.7.3 Integer-to-octet-string conversion

The integer public value y shall be converted to an octet string PV of length k, the public value. The public value PV shall satisfy
y = , (1)
where PV1, ..., PVk are the octets of PV from first to last.

In other words, the first octet of PV has the most significance in the integer and the last octet of PV has the least significance.

10.8 Phase II

This section describes the second phase of Diffie-Hellman key agreement.
The second phase consists of three steps: octet-string-to-integer conversion, exponentiation, and integer-to-octet-string conversion. The input to the second phase shall be the Diffie-Hellman parameters; an octet string PV', the other entity's public value; and the private value x. The output from the second phase shall be an octet string SK, the agreed-upon secret key.
This phase is performed independently by the two parties intending to agree on a secret key, after the parties have exchanged public values resulting from the first phase.

10.8.1 Octet-string-to-integer conversion

The other entity's public value PV' shall be converted to an integer y', the other entity's integer public value. Let PV'1, ..., PV'k be the octets of PV' from first to last. Then the other entity's integer public value y' shall satisfy
y' = .
In other words, the first octet of PV' has the most significance in the integer and the last octet of PV' has the least significance.

10.8.2 Exponentiation

The other entity's integer public value y' shall be raised to the private integer x modulo p to give an integer z, the integer secret key.
z = (y')x mod p, 0 < z < p .
This is the classic discrete-exponentiation computation.
Note. The integer secret key z satisfies
z = (y')x = (gx')x = (gx)x' = yx' mod p ,
where x' is the other entity's private value. This mathematical relationship is the reason the two entities arrive at the same key.

10.8.3 Integer-to-octet-string conversion

The integer secret key z shall be converted to an octet string SK, the secret key, of length k. The secret key SK shall satisfy
z = ,
where SK1, ..., SKk are the octets of SK from first to last.
In other words, the first octet of SK has the most significance in the integer and the last octet

of SK has the least significance.

10.9 Object identifier

This standard defines two object identifiers: pkcs-3 and DHKeyValue.
The object identifier pkcs-3 identifies this standard.
pkcs-3 OBJECT IDENTIFIER ::= { iso(1) member-body(2) US(840) rsadsi(113549) pkcs(1) 3 }
The object identifier DHKeyValue identifies the Diffie-Hellman key agreement method defined in Sections 7 and 8.
DHKeyValue OBJECT IDENTIFIER ::= { pkcs-3 1 }
The DHKeyValue object identifier is intended to be used in the algorithm field of a value of type AlgorithmIdentifier. The parameters field of that type, which has the algorithm-specific syntax ANY DEFINED BY algorithm, would have ASN.1 type DHParameter for this algorithm.
DHParameter ::= SEQUENCE { prime INTEGER, - p base INTEGER, - g privateValueLength INTEGER OPTIONAL }
The fields of type DHParameter have the following meanings:

- prime is the prime p.
- base is the base g.
- privateValueLength is the optional private-value length l.

10.10 Revision history

10.10.1 Versions 1.0-1.2

Versions 1.0-1.2 were distributed to participants in RSA Data Security, Inc.'s Public-Key Cryptography Standards meetings in February and March 1991.

10.10.2 Version 1.3

Version 1.3 is part of the June 3, 1991 initial public release of PKCS. Version 1.3 was published as NIST/OSI Implementors' Workshop document SEC-SIG-91-19.

10.10.3 Version 1.4

Version 1.4 incorporates several editorial changes, including updates to the references and the addition of a revision history. The following substantive changes were made:

- Section 6: Parameter generation is modified to allow central authority to select private-value length in bits.

- Section 7.1: Private-value generation is modified to handle private-value length.
- Section 9: Optional privateValueLength field is added to DHParameter type.

10.11 Author's address

RSA Laboratories (415) 595-7703 100 Marine Parkway (415) 595-4126 (fax) Redwood City, CA 94065 USA pkcs-editor@rsa.com

11 IKE Diffie-Hellman Groups

Different Diffie-Hellman groups are defined for use in IKE. These groups were generated by Richard Schroeppel at the University of Arizona. Properties of these primes are described in [Orm96].

11.1 768-bit MODP - modp1

IKE implementations MAY support a MODP group with the following prime and generator. This group is assigned id 1 (one).
The prime is: 2^768 - 2 ^704 - 1 + 2^64 * { [2^638 pi] + 149686 } Its hexadecimal value is:

```
FFFFFFFF FFFFFFFF C90FDAA2 2168C234 C4C6628B 80DC1CD1 29024E08
8A67CC74 020BBEA6 3B139B22 514A0879 8E3404DD EF9519B3 CD3A431B
302B0A6D F25F1437 4FE1356D 6D51C245 E485B576 625E7EC6 F44C42E9
A63A3620 FFFFFFFF FFFFFFFF
```

The generator is: 2.

11.2 1024-bit MODP Group - modp2

IKE implementations SHOULD support a MODP group with the following prime and generator. This group is assigned id 2 (two).
The prime is 2^1024 - 2^960 - 1 + 2^64 * { [2^894 pi] + 129093 }. Its hexadecimal value is:

```
FFFFFFFF FFFFFFFF C90FDAA2 2168C234 C4C6628B 80DC1CD1 29024E08
8A67CC74 020BBEA6 3B139B22 514A0879 8E3404DD EF9519B3 CD3A431B
302B0A6D F25F1437 4FE1356D 6D51C245 E485B576 625E7EC6 F44C42E9
A637ED6B 0BFF5CB6 F406B7ED EE386BFB 5A899FA5 AE9F2411 7C4B1FE6
49286651 ECE65381 FFFFFFFF FFFFFFFF
```

The generator is 2 (decimal)

11.3 1536-bit MODP Group - modp5

IKE implementations MUST support a MODP group with the following prime and generator. This group is assigned id 5 (five). The 1536 bit MODP group has been used for the implementations for quite a long time, but it has not been documented in the current RFCs or drafts. The prime is 2^1536 - 2^1472 - 1 + 2^64 * {[2^1406 pi] + 741804}. Its hexadecimal value is

```
FFFFFFFF FFFFFFFF C90FDAA2 2168C234 C4C6628B 80DC1CD1 29024E08
8A67CC74 020BBEA6 3B139B22 514A0879 8E3404DD EF9519B3 CD3A431B
302B0A6D F25F1437 4FE1356D 6D51C245 E485B576 625E7EC6 F44C42E9
A637ED6B 0BFF5CB6 F406B7ED EE386BFB 5A899FA5 AE9F2411 7C4B1FE6
49286651 ECE45B3D C2007CB8 A163BF05 98DA4836 1C55D39A 69163FA8
FD24CF5F 83655D23 DCA3AD96 1C62F356 208552BB 9ED52907 7096966D
670C354E 4ABC9804 F1746C08 CA237327 FFFFFFFF FFFFFFFF
```

The generator is 2.

11.4 2048-bit MODP Group

This prime is: 2^2048 - 2^1984 - 1 + 2^64 * { [2^1918 pi] + 124476 } Its hexadecimal value is

```
FFFFFFFF FFFFFFFF C90FDAA2 2168C234 C4C6628B 80DC1CD1
29024E08 8A67CC74 020BBEA6 3B139B22 514A0879 8E3404DD
EF9519B3 CD3A431B 302B0A6D F25F1437 4FE1356D 6D51C245
E485B576 625E7EC6 F44C42E9 A637ED6B 0BFF5CB6 F406B7ED
EE386BFB 5A899FA5 AE9F2411 7C4B1FE6 49286651 ECE45B3D
C2007CB8 A163BF05 98DA4836 1C55D39A 69163FA8 FD24CF5F
83655D23 DCA3AD96 1C62F356 208552BB 9ED52907 7096966D
670C354E 4ABC9804 F1746C08 CA18217C 32905E46 2E36CE3B
E39E772C 180E8603 9B2783A2 EC07A28F B5C55DF0 6F4C52C9
DE2BCBF6 95581718 3995497C EA956AE5 15D22618 98FA0510
15728E5A 8AACAA68 FFFFFFFF FFFFFFFF
```

The generator is: 2.

11.5 3072-bit MODP Group

This prime is: 2^3072 - 2^3008 - 1 + 2^64 * { [2^2942 pi] + 1690314 }
Its hexadecimal value is:

```
FFFFFFFF FFFFFFFF C90FDAA2 2168C234 C4C6628B 80DC1CD1
29024E08 8A67CC74 020BBEA6 3B139B22 514A0879 8E3404DD
EF9519B3 CD3A431B 302B0A6D F25F1437 4FE1356D 6D51C245
E485B576 625E7EC6 F44C42E9 A637ED6B 0BFF5CB6 F406B7ED
```

```
EE386BFB 5A899FA5 AE9F2411 7C4B1FE6 49286651 ECE45B3D
C2007CB8 A163BF05 98DA4836 1C55D39A 69163FA8 FD24CF5F
83655D23 DCA3AD96 1C62F356 208552BB 9ED52907 7096966D
670C354E 4ABC9804 F1746C08 CA18217C 32905E46 2E36CE3B
E39E772C 180E8603 9B2783A2 EC07A28F B5C55DF0 6F4C52C9
DE2BCBF6 95581718 3995497C EA956AE5 15D22618 98FA0510
15728E5A 8AAAC42D AD33170D 04507A33 A85521AB DF1CBA64
ECFB8504 58DBEF0A 8AEA7157 5D060C7D B3970F85 A6E1E4C7
ABF5AE8C DB0933D7 1E8C94E0 4A25619D CEE3D226 1AD2EE6B
F12FFA06 D98A0864 D8760273 3EC86A64 521F2B18 177B200C
BBE11757 7A615D6C 770988C0 BAD946E2 08E24FA0 74E5AB31
43DB5BFC E0FD108E 4B82D120 A93AD2CA FFFFFFFF FFFFFFFF
```

The generator is: 2.

11.6 4096-bit MODP Group

This prime is: 2^4096 - 2^4032 - 1 + 2^64 * { [2^3966 pi] + 240904 }
Its hexadecimal value is :

```
FFFFFFFF FFFFFFFF C90FDAA2 2168C234 C4C6628B 80DC1CD1
29024E08 8A67CC74 020BBEA6 3B139B22 514A0879 8E3404DD
EF9519B3 CD3A431B 302B0A6D F25F1437 4FE1356D 6D51C245
E485B576 625E7EC6 F44C42E9 A637ED6B 0BFF5CB6 F406B7ED
EE386BFB 5A899FA5 AE9F2411 7C4B1FE6 49286651 ECE45B3D
C2007CB8 A163BF05 98DA4836 1C55D39A 69163FA8 FD24CF5F
83655D23 DCA3AD96 1C62F356 208552BB 9ED52907 7096966D
670C354E 4ABC9804 F1746C08 CA18217C 32905E46 2E36CE3B
E39E772C 180E8603 9B2783A2 EC07A28F B5C55DF0 6F4C52C9
DE2BCBF6 95581718 3995497C EA956AE5 15D22618 98FA0510
15728E5A 8AAAC42D AD33170D 04507A33 A85521AB DF1CBA64
ECFB8504 58DBEF0A 8AEA7157 5D060C7D B3970F85 A6E1E4C7
ABF5AE8C DB0933D7 1E8C94E0 4A25619D CEE3D226 1AD2EE6B
F12FFA06 D98A0864 D8760273 3EC86A64 521F2B18 177B200C
BBE11757 7A615D6C 770988C0 BAD946E2 08E24FA0 74E5AB31
43DB5BFC E0FD108E 4B82D120 A9210801 1A723C12 A787E6D7
88719A10 BDBA5B26 99C32718 6AF4E23C 1A946834 B6150BDA
2583E9CA 2AD44CE8 DBBBC2DB 04DE8EF9 2E8EFC14 1FBECAA6
287C5947 4E6BC05D 99B2964F A090C3A2 233BA186 515BE7ED
1F612970 CEE2D7AF B81BDD76 2170481C D0069127 D5B05AA9
93B4EA98 8D8FDDC1 86FFB7DC 90A6C08F 4DF435C9 34063199
FFFFFFFF FFFFFFFF
```

The generator is: 2.

11.7 6144-bit MODP Group

This prime is: 2^6144 - 2^6080 - 1 + 2^64 * { [2^6014 pi] + 929484 }
Its hexadecimal value is :

```
FFFFFFFF FFFFFFFF C90FDAA2 2168C234 C4C6628B 80DC1CD1
29024E08 8A67CC74 020BBEA6 3B139B22 514A0879 8E3404DD
EF9519B3 CD3A431B 302B0A6D F25F1437 4FE1356D 6D51C245
E485B576 625E7EC6 F44C42E9 A637ED6B 0BFF5CB6 F406B7ED
EE386BFB 5A899FA5 AE9F2411 7C4B1FE6 49286651 ECE45B3D
C2007CB8 A163BF05 98DA4836 1C55D39A 69163FA8 FD24CF5F
83655D23 DCA3AD96 1C62F356 208552BB 9ED52907 7096966D
670C354E 4ABC9804 F1746C08 CA18217C 32905E46 2E36CE3B
E39E772C 180E8603 9B2783A2 EC07A28F B5C55DF0 6F4C52C9
DE2BCBF6 95581718 3995497C EA956AE5 15D22618 98FA0510
15728E5A 8AAAC42D AD33170D 04507A33 A85521AB DF1CBA64
ECFB8504 58DBEF0A 8AEA7157 5D060C7D B3970F85 A6E1E4C7
ABF5AE8C DB0933D7 1E8C94E0 4A25619D CEE3D226 1AD2EE6B
F12FFA06 D98A0864 D8760273 3EC86A64 521F2B18 177B200C
BBE11757 7A615D6C 770988C0 BAD946E2 08E24FA0 74E5AB31
43DB5BFC E0FD108E 4B82D120 A9210801 1A723C12 A787E6D7
88719A10 BDBA5B26 99C32718 6AF4E23C 1A946834 B6150BDA
2583E9CA 2AD44CE8 DBBBC2DB 04DE8EF9 2E8EFC14 1FBECAA6
287C5947 4E6BC05D 99B2964F A090C3A2 233BA186 515BE7ED
1F612970 CEE2D7AF B81BDD76 2170481C D0069127 D5B05AA9
93B4EA98 8D8FDDC1 86FFB7DC 90A6C08F 4DF435C9 34028492
36C3FAB4 D27C7026 C1D4DCB2 602646DE C9751E76 3DBA37BD
F8FF9406 AD9E530E E5DB382F 413001AE B06A53ED 9027D831
179727B0 865A8918 DA3EDBEB CF9B14ED 44CE6CBA CED4BB1B
DB7F1447 E6CC254B 33205151 2BD7AF42 6FB8F401 378CD2BF
5983CA01 C64B92EC F032EA15 D1721D03 F482D7CE 6E74FEF6
D55E702F 46980C82 B5A84031 900B1C9E 59E7C97F BEC7E8F3
23A97A7E 36CC88BE 0F1D45B7 FF585AC5 4BD407B2 2B4154AA
CC8F6D7E BF48E1D8 14CC5ED2 0F8037E0 A79715EE F29BE328
06A1D58B B7C5DA76 F550AA3D 8A1FBFF0 EB19CCB1 A313D55C
DA56C9EC 2EF29632 387FE8D7 6E3C0468 043E8F66 3F4860EE
12BF2D5B 0B7474D6 E694F91E 6DCC4024 FFFFFFFF FFFFFFFF
```

The generator is: 2.

11.8 8192-bit MODP Group

This prime is: 2^8192 - 2^8128 - 1 + 2^64 * { [2^8062 pi] + 4743158 }
Its hexadecimal value is :

```
FFFFFFFF FFFFFFFF C90FDAA2 2168C234 C4C6628B 80DC1CD1
29024E08 8A67CC74 020BBEA6 3B139B22 514A0879 8E3404DD
```

```
EF9519B3 CD3A431B 302B0A6D F25F1437 4FE1356D 6D51C245
E485B576 625E7EC6 F44C42E9 A637ED6B 0BFF5CB6 F406B7ED
EE386BFB 5A899FA5 AE9F2411 7C4B1FE6 49286651 ECE45B3D
C2007CB8 A163BF05 98DA4836 1C55D39A 69163FA8 FD24CF5F
83655D23 DCA3AD96 1C62F356 208552BB 9ED52907 7096966D
670C354E 4ABC9804 F1746C08 CA18217C 32905E46 2E36CE3B
E39E772C 180E8603 9B2783A2 EC07A28F B5C55DF0 6F4C52C9
DE2BCBF6 95581718 3995497C EA956AE5 15D22618 98FA0510
15728E5A 8AAAC42D AD33170D 04507A33 A85521AB DF1CBA64
ECFB8504 58DBEF0A 8AEA7157 5D060C7D B3970F85 A6E1E4C7
ABF5AE8C DB0933D7 1E8C94E0 4A25619D CEE3D226 1AD2EE6B
F12FFA06 D98A0864 D8760273 3EC86A64 521F2B18 177B200C
BBE11757 7A615D6C 770988C0 BAD946E2 08E24FA0 74E5AB31
43DB5BFC E0FD108E 4B82D120 A9210801 1A723C12 A787E6D7
88719A10 BDBA5B26 99C32718 6AF4E23C 1A946834 B6150BDA
2583E9CA 2AD44CE8 DBBBC2DB 04DE8EF9 2E8EFC14 1FBECAA6
287C5947 4E6BC05D 99B2964F A090C3A2 233BA186 515BE7ED
1F612970 CEE2D7AF B81BDD76 2170481C D0069127 D5B05AA9
93B4EA98 8D8FDDC1 86FFB7DC 90A6C08F 4DF435C9 34028492
36C3FAB4 D27C7026 C1D4DCB2 602646DE C9751E76 3DBA37BD
F8FF9406 AD9E530E E5DB382F 413001AE B06A53ED 9027D831
179727B0 865A8918 DA3EDBEB CF9B14ED 44CE6CBA CED4BB1B
DB7F1447 E6CC254B 33205151 2BD7AF42 6FB8F401 378CD2BF
5983CA01 C64B92EC F032EA15 D1721D03 F482D7CE 6E74FEF6
D55E702F 46980C82 B5A84031 900B1C9E 59E7C97F BEC7E8F3
23A97A7E 36CC88BE 0F1D45B7 FF585AC5 4BD407B2 2B4154AA
CC8F6D7E BF48E1D8 14CC5ED2 0F8037E0 A79715EE F29BE328
06A1D58B B7C5DA76 F550AA3D 8A1FBFF0 EB19CCB1 A313D55C
DA56C9EC 2EF29632 387FE8D7 6E3C0468 043E8F66 3F4860EE
12BF2D5B 0B7474D6 E694F91E 6DBE1159 74A3926F 12FEE5E4
38777CB6 A932DF8C D8BEC4D0 73B931BA 3BC832B6 8D9DD300
741FA7BF 8AFC47ED 2576F693 6BA42466 3AAB639C 5AE4F568
3423B474 2BF1C978 238F16CB E39D652D E3FDB8BE FC848AD9
22222E04 A4037C07 13EB57A8 1A23F0C7 3473FC64 6CEA306B
4BCBC886 2F8385DD FA9D4B7F A2C087E8 79683303 ED5BDD3A
062B3CF5 B3A278A6 6D2A13F8 3F44F82D DF310EE0 74AB6A36
4597E899 A0255DC1 64F31CC5 0846851D F9AB4819 5DED7EA1
B1D510BD 7EE74D73 FAF36BC3 1ECFA268 359046F4 EB879F92
4009438B 481C6CD7 889A002E D5EE382B C9190DA6 FC026E47
9558E447 5677E9AA 9E3050E2 765694DF C81F56E8 80B96E71
60C980DD 98EDD3DF FFFFFFFF FFFFFFFF
```

The generator is: 2.