XEP-0373: OpenPGP for XMPP

Florian Schmaus
mailto:flo@geekplace.eu
xmpp:flo@geekplace.eu

Dominik Schürmann
mailto:dominik@dominikschuermann.de
xmpp:dominik@dominikschuermann.de

Vincent Breitmoser
mailto:look@my.amazin.horse
xmpp:valodim@stratum0.org

2021-05-04
Version 0.7.0

Status	Type	Short Name
Experimental	Standards Track	ox

Specifies end-to-end encryption and authentication of data with the help of OpenPGP, announcement, discovery and retrieval of public keys and a mechanism to synchronize secret keys over multiple devices.

Legal

Copyright

This XMPP Extension Protocol is copyright © 1999 – 2020 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 Glossary **1**

3 OpenPGP Encrypted and Signed Data **2**
- 3.1 Exchanging OpenPGP Encrypted and Signed Data . . . 2
- 3.2 Verification of <openpgp/> Content . . . 3

4 Announcing and Discovering Public Keys via PEP **4**
- 4.1 The OpenPGP Public-Key Data Node . . . 4
- 4.2 The OpenPGP Public Key Metadata Node . . . 5
- 4.3 Discovering Public Keys of a User . . . 6
- 4.4 Requesting Public Keys . . . 7
- 4.5 Receiving notifications about key changes . . . 8

5 Synchronizing the Secret Key with a Private PEP Node **8**
- 5.1 Required PEP features . . . 8
  - 5.1.1 Discovering support . . . 8
- 5.2 Requesting Information About the Secret Key PEP Node . . . 9
  - 5.2.1 Client Sends Request . . . 9
  - 5.2.2 PEP Service Success Response . . . 10
  - 5.2.3 PEP Node Does Not Exist Response . . . 10
  - 5.2.4 PEP Not Supported . . . 10
- 5.3 Creating the Secret Key PEP Node . . . 11
- 5.4 Encrypting the Secret Key Backup . . . 11

6 Business Rules **12**
- 6.1 OpenPGP Packet Format Version Restriction . . . 12
- 6.2 PubSub Node Configuration . . . 12
- 6.3 Key Enforcement . . . 13

7 Implementors Advice **13**
- 7.1 Design Principles and Techniques . . . 13
- 7.2 Stanza Size . . . 13
- 7.3 XMPP Address Normalization . . . 14

8 Rationale **14**
- 8.1 Key Handling . . . 14
- 8.2 OpenPGP Element and Content Element Design . . . 14
- 8.3 Addressing the Issues and Problems of XEP-0027 . . . 15
- 8.4 Not using OpenPGP ASCII Armor . . . 15
- 8.5 OpenPGP User IDs . . . 15

9 Security Considerations **15**

10 IANA Considerations **16**

11 XMPP Registrar Considerations **16**
11.1 Protocol Namespaces 16

12 XML Schema **16**

13 Acknowledgements **16**

1 Introduction

This XMPP extension protocol specifies the foundations of end-to-end encryption and authentication, based on digital signatures, of data with the help of OpenPGP. Additional XEPs will use this extension protocol as building block when specifying their own OpenPGP profile suiting their use case. One such profile is the Instant Messaging Profile specified in OpenPGP for XMPP Instant Messaging (XEP-0374) [1].
XMPP provides the mechanisms to solve a lot of issues that come with modern day OpenPGP usage. For example, based on Personal Eventing Protocol (XEP-0163) [2] this specification describes a standardized way to discover OpenPGP public keys of other entities. But unlike the OpenPGP keyservers, this process establishes a strong relation between the key and the key's owning entity (usually a human user). A similar mechanism described herein allows to synchronize the secret key(s) across multiple devices.
OpenPGP in return allows for end-to-end encrypted data to be exchanged between one, two or even multiple entities (multi-end-to-multi-end encryption). Therefore this XEP can be used for example to implement end-to-end encrypted Multi-User Chat (XEP-0045) [3].

2 Glossary

OpenPGP element An XMPP extension element: <openpgp/> qualified by the 'urn:xmpp:openpgp:0' namespace

OpenPGP content element An element embedded via OpenPGP in a <openpgp/> element. Either one of <signcrypt/>, <sign/> or <crypt/>, qualified by the 'urn:xmpp:openpgp:0' namespace.

PEP Personal Eventing Protocol (XEP-0163) XEP-0163: Personal Eventing Protocol <https://xmpp.org/extensions/xep-0163.html>.

Public-Key metadata node ("metadata node") A PEP node containing metadata of the entity's public OpenPGP key.

Public-Key data node ("data node") A PEP node containing an entity's public OpenPGP key.

Secret-Key node A PEP node containing an entity's encrypted secret OpenPGP key.

OpenPGP v4 Fingerprint String A String representing the OpenPGP v4 fingerprint of a key. If the key consists of a primary key and subkeys, this is the fingerprint of the primary key.

[1] XEP-0374: OpenPGP for XMPP Instant Messaging <https://xmpp.org/extensions/xep-0374.html>.
[2] XEP-0163: Personal Eventing Protocol <https://xmpp.org/extensions/xep-0163.html>.
[3] XEP-0045: Multi-User Chat <https://xmpp.org/extensions/xep-0045.html>.

3 OpenPGP Encrypted and Signed Data

3.1 Exchanging OpenPGP Encrypted and Signed Data

The <openpgp/> extension element qualified by the 'urn:xmpp:openpgp:0' namespace is used in order to exchange encrypted and signed data.

Listing 1: The <openpgp/> extension within a message.

```
<message to='juliet@example.org'>
  <openpgp xmlns='urn:xmpp:openpgp:0'>
    BASE64_OPENPGP_MESSAGE
  </openpgp>
</message>
```

The text content of <openpgp/> ("BASE64_OPENPGP_MESSAGE") is a Base64 encoded (RFC 4648 [4] § 4) OpenPGP message as specified in RFC 4880 [5] which contains an encrypted and/or signed UTF-8 (RFC 3629 [6]) encoded string. This string MUST correspond to exactly one OpenPGP content element, that is, it represents either a <signcrypt/>, a <sign/> or a <crypt/> extension element qualified by the 'urn:xmpp:openpgp:0' namespace. Note that OpenPGP's ASCII Armor is not used, instead the XMPP client MUST encode the raw bytes of the OpenPGP message using Base64.
In case of a <signcrypt/> element, the OpenPGP message embedded in the <openpgp/> element MUST be encrypted and signed, and SHOULD also be encrypted to self. In case of a <sign/> element, the OpenPGP message MUST be signed and MUST NOT be encrypted. In case of <crypt/> the OpenPGP message MUST NOT be signed, but MUST be encrypted.

Listing 2: The <signcrypt/> extension element.

```
<signcrypt xmlns='urn:xmpp:openpgp:0'>
  <to jid='juliet@example.org'/>
  <time stamp='2014-07-10T17:06:00+02:00'/>
  <rpad>
    f0rm1l4n4-mT8y33j!Y%fRSrcd^ZE4Q7VDt1L%WEgR!kv
  </rpad>
  <payload>
    <body xmlns='jabber:client'>
      This is a secret message.
    </body>
  </payload>
</signcrypt>
```

OpenPGP content elements MUST possess exactly one 'time' element as direct child elements. The <signcrypt/> and <sign/> content elements MUST contain at least one 'to' element(s),

[4] RFC 4648: The Base16, Base32, and Base64 Data Encodings <http://tools.ietf.org/html/rfc4648>.
[5] RFC 4880: OpenPGP Message Format <http://tools.ietf.org/html/rfc4880>.
[6] RFC 3629: UTF-8, a transformation format of ISO 10646 <http://tools.ietf.org/html/rfc3629>.

which MUST have a 'jid' attribute containing the intended recipient's XMPP address of the signed and/or encrypted data to prevent Surreptitious Forward Attacks[7]. The XMPP address found in the 'to' element's 'jid' attribute SHOULD be without Resourcepart (i.e., a bare JID). A <crypt/> content element may not carry a 'to' attribute. The 'time' element MUST have a 'stamp' attribute which contains the timestamp when the OpenPGP content element was signed and/or encrypted in the DateTime format as specified in XMPP Date and Time Profiles (XEP-0082) [8] § 3.2. The <signcrypt/> and <crypt/> elements SHOULD furthermore contain a 'rpad' element which text content is a random-length random-content padding.

Content Element	'to' Element	'time' Element	<rpad/> Element	<payload/> Element
<signcrypt/>	MUST have at least one	MUST have exactly one	SHOULD have exactly one	MUST have exactly one
<sign/>	MUST have at least one	MUST have exactly one	OPTIONAL	MUST have exactly one
<crypt/>	OPTIONAL	MUST have exactly one	SHOULD have exactly one	MUST have exactly one

OpenPGP content elements MUST possess exactly one <payload/> element. The child elements of <payload/> can be seen as OpenPGP secured Stanza extension elements which are encrypted and/or signed. After the <openpgp/> element and the including <signcrypt/>, <sign/> or <crypt/> element was verified, they are processed according to the specification of the relevant OpenPGP for XMPP profile (see for example OpenPGP for XMPP Instant Messaging (XEP-0374) [9]).

3.2 Verification of <openpgp/> Content

Recipients MUST verify that the signature is valid, that the signature's key corresponds to the sender's key, and that the sender's key has a User ID containing the sender's XMPP address in the form "xmpp:juliet@example.org" (for details see "OpenPGP User IDs"). Thus, the recipient may need to retrieve the key from the Personal Eventing Protocol node as described above. At least one of the XMPP addresses found in the 'to' elements contained in OpenPGP content element MUST correspond to the outer 'to' of the XMPP <message/>. Furthermore, recipients are RECOMMENDED to verify the 'time' element for plausibility or to display it to a user for

[7] Jee Hea An, Yevgeniy Dodis, and Tal Rabin. 2002. On the Security of Joint Signature and Encryption. In Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques: Advances in Cryptology (EUROCRYPT '02), Lars R. Knudsen (Ed.). Springer-Verlag, London, UK, UK, 83-107. <https://www.iacr.org/archive/eurocrypt2002/23320080/adr.pdf>

[8] XEP-0082: XMPP Date and Time Profiles <https://xmpp.org/extensions/xep-0082.html>.

[9] XEP-0374: OpenPGP for XMPP Instant Messaging <https://xmpp.org/extensions/xep-0374.html>.

verification.

4 Announcing and Discovering Public Keys via PEP

Parties interested in exchanging encrypted data between each other via OpenPGP need to know the public key(s) of the recipients. The following section specifies a mechanism to announce and discover public keys.
Two PEP node types are invovled: A ”medatata node” is used to store meta information about OpenPGP keys used by an entity while the actual public keys are stored in ”data nodes”.

4.1 The OpenPGP Public-Key Data Node

The public key data, as specified in RFC 4880, is stored in a PEP data node. Note that OpenPGP's ASCII Armor is not used, instead the XMPP client MUST encode the public key using Base64.
The id of the node MUST be ”urn:xmpp:openpgp:0:public-keys:” followed by the fingerprint string of the OpenPGP public-key contained in the data node.
In absence of a use-case specific access model, it is RECOMMENDED to use the 'open' access model for the public key data node in order to give entities without presence subscription read access to the public key.
The access model can be changed efficiently by using publish-options.
The *OpenPGP v4 fingerprint string* is obtained as follows: First the raw bytes of the fingerprint are computed as specified in RFC 4880 § 12.2.. Then the bytes are encoded as a hexadecimal string using upper case characters[10].
The publishing entity SHOULD set the PubSub item ID to the time the item is published encoded as DateTime format specified in XEP-0082.
The data node MUST contain an <pubkey/> element qualified by the 'urn:xmpp:openpgp:0' namespace. The element MUST include a <data/> element which contains the data of the key Base64 encoded.

Listing 3: Saving the public key in the data node.

```
<iq type='set' from='juliet@example.org/balcony' id='publish1'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <publish node='urn:xmpp:openpgp:0:public-
        keys:1357B01865B2503C18453D208CAC2A9678548E35'>
      <item id='2020-01-21T10:46:21Z'>
        <pubkey xmlns='urn:xmpp:openpgp:0'>
          <data>
            BASE64_OPENPGP_PUBLIC_KEY
          </data>
        </pubkey>
```

[10] This matches the representation used by GnuPG minus the SPACE separation.

```
      </item>
    </publish>
    <publish-options>
      <x xmlns='jabber:x:data' type='submit'>
        <field var='FORM_TYPE' type='hidden'>
          <value>http://jabber.org/protocol/pubsub#publish-options</
              value>
        </field>
        <field var='pubsub#access_model'>
          <value>open</value>
        </field>
      </x>
    </publish-options>
  </pubsub>
</iq>
```

4.2 The OpenPGP Public Key Metadata Node

To update the public keys used by an entity, the metadata node is updated. Before adding a OpenPGP key fingerprint to the metadata node, the publisher MUST ensure that the public key is available at the corresponding data node.
Just like with the public key data node, in absence of a use-case specific access model, it is RECOMMENDED to set the access model of the metadata node to 'open', such that entities without mutual presence subscription are still able to access the node items.
The ID of the metadata node is 'urn:xmpp:openpgp:0:public-keys'. It contains a <public-keys-list/> element qualified by the 'urn:xmpp:openpgp:0' namespace containing one or more <pubkey-metadata/> elements. Every pubkey-metadata element MUST have a 'v4-fingerprint' attribute, containing the OpenPGP v4 fingerprint string, and a 'date' attribute, containing the time the key was published or updated in DateTime format of XEP-0082. An OpenPGP V4 fingerprint MUST NOT occur in the list more than once.

Listing 4: Publishing a public key to the metadata node.

```
<iq type='set' from='juliet@example.org/balcony' id='publish1'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <publish node='urn:xmpp:openpgp:0:public-keys'>
      <item>
        <public-keys-list xmlns='urn:xmpp:openpgp:0'>
          <pubkey-metadata
            v4-fingerprint='1357B01865B2503C18453D208CAC2A9678548E35'
            date='2018-03-01T15:26:12Z'
            />
          <pubkey-metadata
            v4-fingerprint='67819B343B2AB70DED9320872C6464AF2A8E4C02'
            date='1953-05-16T12:00:00Z'
            />
        </public-keys-list>
```

```
      </item>
    </publish>
    <publish-options>
      <x xmlns='jabber:x:data' type='submit'>
        <field var='FORM_TYPE' type='hidden'>
          <value>http://jabber.org/protocol/pubsub#publish-options</
              value>
        </field>
        <field var='pubsub#access_model'>
          <value>open</value>
        </field>
      </x>
    </publish-options>
  </pubsub>
</iq>
```

4.3 Discovering Public Keys of a User

In order to discover the OpenPGP public keys of a remote entity, the interested entity first queries the remote entity's metadata note to learn about the currently annouced OpenPGP keys.

Listing 5: Requesting the metadata node of a user.

```
<iq from='romeo@example.org/orchard'
    to='juliet@example.org'
    type='get'
    id='getmeta'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <items node='urn:xmpp:openpgp:0:public-keys'/>
  </pubsub>
</iq>
```

Listing 6: Personal Eventing Protocol result containing the metadata node of the user.

```
<iq from='juliet@example.org'
    to='romeo@example.org/orchard'
    type='result'
    id='getmeta'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <items node='urn:xmpp:openpgp:0:public-keys'>
      <item>
        <public-keys-list xmlns='urn:xmpp:openpgp:0'>
          <pubkey-metadata
            v4-fingerprint='1357B01865B2503C18453D208CAC2A9678548E35'
            date='2018-03-01T15:26:12Z'
            />
          <pubkey-metadata
```

```
            v4-fingerprint='67819B343B2AB70DED9320872C6464AF2A8E4C02'
            date='1953-05-16T12:00:00Z'
            />
        </public-keys-list>
      </item>
    </items>
  </pubsub>
</iq>
```

4.4 Requesting Public Keys

OpenPGP key(s) can be retrieved by querying the data node for a specific fingerprint.

Listing 7: Requesting an OpenPGP public key from an XMPP entity.

```
<iq from='romeo@example.org/orchard'
    to='juliet@example.org'
    type='get'
    id='getpub'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <items node='urn:xmpp:openpgp:0:public-
        keys:1357B01865B2503C18453D208CAC2A9678548E35'
           max_items='1'/>
  </pubsub>
</iq>
```

Listing 8: Personal Eventing Protocol result containing the requested public key.

```
<iq from='juliet@example.org'
    to='romeo@example.org/orchard'
    type='result'
    id='getpub'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <items node='urn:xmpp:openpgp:0:public-
        keys:1357B01865B2503C18453D208CAC2A9678548E35'>
      <item id='2020-01-21T10:46:21Z'>
        <pubkey xmlns='urn:xmpp:openpgp:0'>
          <data>
            BASE64_OPENPGP_PUBLIC_KEY
          </data>
        </pubkey>
      </item>
    </items>
  </pubsub>
</iq>
```

Note that the result may contain multiple pubkey elements. Only the public keys found in the most recent item MUST be used. Requesters may want to limit the results to the most

recent item using the 'max_items' attribute set to '1'. Clients could alternatively use Result Set Management (XEP-0059) [11] as an alternative to 'max_items' but accoding to XEP-0060 RSM is not (yet) mandatory for PubSub services.
Some XMPP services may not provide the Personal Eventing Protocol feature required to provide the mechanism described here. If so, they will return an <iq/> error of type service-unavailable.

4.5 Receiving notifications about key changes

Entities creating PEP nodes defined herein SHOULD configure the nodes as notification-only nodes by setting 'pubsub#deliver_payloads" configuration field to 'false'.
Entities which are subscribed to the metadata node or advertise the "urn:xmpp:openpgp:0:public-keys+notify" feature via Service Discovery (XEP-0030) [12] (see XEP-0060 § 9.2) receive a notification upon a node update. Entities subscribed to PEP nodes defined herein MUST be prepared that PubSub notifications may be without the payload and only contain the published item's ID.

5 Synchronizing the Secret Key with a Private PEP Node

A private PEP node is used to allow XMPP clients to synchronize the user's secret OpenPGP key. Where private PEP node is defined: A PEP node in whitelist mode where only the bare JID of the key owner is whitelisted as described in Best Practices for Persistent Storage of Private Data via Publish-Subscribe (XEP-0223) [13]. The secret key is additionally encrypted.

5.1 Required PEP features

The used PEP server MUST support PEP and the whitelist access model. It SHOULD also support persistent items.

5.1.1 Discovering support

Listing 9: Account owner queries server regarding protocol support

```
<iq from='juliet@capulet.lit/balcony'
    to='juliet@capulet.lit'
    id='disco1'
    type='get'>
```

[11] XEP-0059: Result Set Management <https://xmpp.org/extensions/xep-0059.html>.
[12] XEP-0030: Service Discovery <https://xmpp.org/extensions/xep-0030.html>.
[13] XEP-0223: Best Practices for Persistent Storage of Private Data via Publish-Subscribe <https://xmpp.org/extensions/xep-0223.html>.

```
  <query xmlns='http://jabber.org/protocol/disco#info'/>
</iq>
```

The service discovery result must contain a PEP identity '<identity category='pubsub' type='pep'/>, and the 'http://jabber.org/protocol/pubsub#access-whitelist' feature. Ideally it also contains the 'http://jabber.org/protocol/pubsub#persistent-items' feature

Listing 10: Server communicates protocol support

```
<iq from='juliet@capulet.lit'
    to='juliet@capulet.lit/balcony'
    id='disco1'
    type='result'>
  <query xmlns='http://jabber.org/protocol/disco#info'>
    <identity category='account' type='registered'/>
    <identity category='pubsub' type='pep'/>
    <feature var='http://jabber.org/protocol/pubsub#access-presence'/>
    <feature var='http://jabber.org/protocol/pubsub#auto-create'/>
    <feature var='http://jabber.org/protocol/pubsub#auto-subscribe'/>
    <feature var='http://jabber.org/protocol/pubsub#config-node'/>
    <feature var='http://jabber.org/protocol/pubsub#create-and-
       configure'/>
    <feature var='http://jabber.org/protocol/pubsub#create-nodes'/>
    <feature var='http://jabber.org/protocol/pubsub#filtered-
       notifications'/>
    <feature var='http://jabber.org/protocol/pubsub#persistent-items'/
       >
    <feature var='http://jabber.org/protocol/pubsub#publish'/>
    <feature var='http://jabber.org/protocol/pubsub#retrieve-items'/>
    <feature var='http://jabber.org/protocol/pubsub#subscribe'/>
    ...
  </query>
</iq>
```

5.2 Requesting Information About the Secret Key PEP Node

In order to synchronize the secret key over a private PEP node, clients first need to discover and verify the node for the correct settings.

5.2.1 Client Sends Request

Listing 11: Requesting the user's secret key.

```
<iq from='romeo@example.org/orchard'
    to='juliet@example.org'
    type='get'
```

```
    id='getsecret'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <items node='urn:xmpp:openpgp:0:secret-key'
           max_items='1'/>
  </pubsub>
</iq>
```

5.2.2 PEP Service Success Response

Listing 12: Personal Eventing Protocol result containing the requested secret key.

```
<iq from='juliet@example.org'
    to='romeo@example.org/orchard'
    type='result'
    id='getsecret'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <items node='urn:xmpp:openpgp:0:secret-key'>
      <item>
        <secretkey xmlns='urn:xmpp:openpgp:0'>
            BASE64_OPENPGP_ENCRYPTED_SECRET_KEY
        </secretkey>
      </item>
    </items>
  </pubsub>
</iq>
```

5.2.3 PEP Node Does Not Exist Response

If the node does not exist the service will return an <iq/> error indicating the item-not-found error condition. The client MUST then create it with an whitelist access model.

Listing 13: Node does not exist

```
<iq from='juliet@example.org'
    to='romeo@example.org/orchard'
    type='error'
    id='getsecret'>
  <error type='cancel'>
    <item-not-found xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

5.2.4 PEP Not Supported

The service will return a service-unavailable error <iq/> if it does not support PEP.

Listing 14: Node does not exist

```
<iq from='juliet@example.org'
    to='romeo@example.org/orchard'
    type='error'
    id='getsecret'>
  <error type='cancel'>
    <service-unavailable xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

5.3 Creating the Secret Key PEP Node

Listing 15: Client creates secret key PEP node

```
<iq type='set'
    from='juliet@example.org/balcony'
    id='create-node'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <create node='urn:xmpp:openpgp:0:secret-key'/>
    <configure>
      <x xmlns='jabber:x:data' type='submit'>
        <field var='FORM_TYPE' type='hidden'>
          <value>http://jabber.org/protocol/pubsub#node_config</value>
        </field>
        <field var='pubsub#access_model'>
          <value>whitelist</value>
        </field>
        <field var='pubsub#send_last_published_item'>
          <value>on_sub</value>
        </field>
      </x>
    </configure>
  </pubsub>
</iq>
```

Listing 16: Service informs requesting entity of success

```
<iq type='result'
    to='juliet@example.org/balcony'
    id='create-node'/>
```

The node is now created and the only affiliated entity is the bare JID of the user, who created the node, with an affiliation as 'owner'.

5.4 Encrypting the Secret Key Backup

In order to set a new secret key, clients store the encrypted secret key as Base64 encoded raw OpenPGP message within an <secretkey/> element qualified by the 'urn:xmpp:openpgp:0'

namespace. These secret key backups are created as follows:

1. All secret keys that should be included in the backup MUST be concatenated in their transferable key format (RFC 4880 § 11.1). The octet indicating string-to-key usage conventions MUST be set to zero in the corresponding Secret-Key Packet(s) (RFC 4880 § 5.5.3). The secret key material will be encrypted in step 4 using a Symmetric-Key Encrypted Session Key Packet.

2. A backup code is generated from secure random: The backup code consists of 24 upper case characters from the Latin alphabet and numbers without 'O' ("LATIN CAPITAL LETTER O") and '0' ("DIGIT ZERO") (alphabet: 123456789ABCDEFGHIJKLMNPQRSTUVWXYZ) grouped into 4-character chunks, e.g., TWNK-KD5Y-MT3T-E1GS-DRDB-KVTW. The characters MUST be generated from cryptographically secure random. For example getrandom(2), SecureRandom or /dev/urandom. More information about the randomness requirements for security can be found in RFC 4086 [14]

3. The whole backup code including the dashes is directly used as a string to encrypt the concatenated transferable keys as an OpenPGP message. More precisely: It is used as the symmetric-key for a Symmetric-Key Encrypted Session Key Packet according to RFC 4880 § 5.3; the symmetric-key is thus 29 characters long including the dashes. The encryption algorithm MUST be one of the standardized OpenPGP symmetric algorithms, e.g, AES-128.

6 Business Rules

6.1 OpenPGP Packet Format Version Restriction

Implementations of this XEP MUST generate and accept only version 4 (or higher) OpenPGP packets. Lower version OpenPGP packets are insecure in many aspects (see for example RFC 4880 § 5.5.2.).

6.2 PubSub Node Configuration

The Public-Key *metadata* node and the Secret-Key node SHOULD be configured to either never send the latest item, or to send the latest item only when a new entity subscribed. Thus the nodes 'send_last_published_item' configuration option SHOULD be set to either 'never' or 'on_sub' (see XEP-0060 § 16.4.4).

[14] RFC 4086: Randomness Requirements for Security <http://tools.ietf.org/html/rfc4086>.

6.3 Key Enforcement

Whenever an entity becomes aware that the metadata node has changed (e.g., by receiving a PEP update from their own account), it SHOULD check that the list contains the key they use. If the key has been removed, the entity SHOULD reannounce it.

7 Implementors Advice

7.1 Design Principles and Techniques

OpenPGP implementations have a sad history of being not very user-friendly which results in users either not using OpenPGP or in users wrongly using OpenPGP. Implementors of this XEP, and additional future XEPs based on this XEP, therefore should read STEED[15] and "Why Johnny can't encrypt"[16]. Implementors of this XEP are encouraged to provide the concepts described in STEED:

- Automatic key generation
- Automatic key distribution
- Opportunistic encryption
- Trust upon first contact

Furthermore implementors should design the user interface for effective security by following the design principles and techniques for security mentioned in "Why Johnny Can't Encrypt".

7.2 Stanza Size

Implementors should be aware that the size OpenPGP public and secret keys is somewhere in the range of tens of kilobytes. Applying Base64 encoding on keys, as it is described herein, further increases the size. The formula to determine the Base64 encoded size is: ceil(bytes / 3) * 4. Thus the lower bound for the maximum stanza size of 10000 bytes, as specified in RFC 6120 § 13.12. 4., is usually exceeded. However all XMPP server implementations, the authors are aware of, follow the recommendation of the RFC and do not blindly set the maximum stanza size to such a low value, but use a much higher threshold. Therefore, this should hardly be an issue for implementations. Nevertheless, it is advised to keep the size of OpenPGP keys small by removing all signatures except the most recent self-signature on each User ID before

[15] Koch, Werner, and Marcus Brinkman "STEED — Usable End-to-End Encryption", White Paper, g10 GmbH, 2011-10-17. <http://g10code.com/steed.html>

[16] Whitten, Alma, and J. Doug Tygar. "Why Johnny Can't Encrypt: A Usability Evaluation of PGP 5.0." Usenix Security. Vol. 1999. 1999. <https://www.cs.berkeley.edu/~{}tygar/papers/Why_Johnny_Cant_Encrypt/OReilly.pdf>

exporting the key (cf. GnuPG's --export-options export-minimal). In addition, implementors are advised to handle <policy-violation/> error responses when trying to transmit Base64 encoded keys.

7.3 XMPP Address Normalization

The format of XMPP addresses, sometimes called JIDs, is well defined. Thus they need to be normalized, as defined in RFC 7622 [17]. When implementations are required to compare XMPP addresses for equality, as it is the case in "Verification of <openpgp/> Content", then they also have to compare the normalized versions of the addresses.

8 Rationale

8.1 Key Handling

This specification intentionally does not specify if the used OpenPGP key should be a primary key or a subkey. It is even possible to announce multiple public keys in the Personal Eventing Protocol node. Implementations MUST be prepared to find multiple public keys. The authors however believe that for ease of use only one OpenPGP key specially crafted for the XMPP use case should be created, announced and used.

8.2 OpenPGP Element and Content Element Design

The <openpgp/> and OpenPGP content elements are container elements for arbitrary signed and encrypted data and can thus act as building blocks for encrypted data included in Message, IQ and Presence stanzas. For example, future specifications may use them to implement encrypted versions of In-Band Bytestreams (XEP-0047) [18] or Jingle In-Band Bytestreams Transport Method (XEP-0261) [19].
Note that signed OpenPGP messages already contain a timestamp as per the OpenPGP specification. OpenPGP content elements nevertheless require the 'time' element because not every OpenPGP API may provide access to the embedded OpenPGP timestamp.
The 'rpad' element of the OpenPGP content elements exists to prevent length-based side channel attacks.

[17] RFC 7622: Extensible Messaging and Presence Protocol (XMPP): Address Format <http://tools.ietf.org/html/rfc7622>.
[18] XEP-0047: In-Band Bytestreams <https://xmpp.org/extensions/xep-0047.html>.
[19] XEP-0261: Jingle In-Band Bytestreams Transport Method <https://xmpp.org/extensions/xep-0261.html>.

8.3 Addressing the Issues and Problems of XEP-0027

This specification addresses all relevant issues of Current Jabber OpenPGP Usage (XEP-0027) [20] (§ 4, § 5). It mitigates replay attacks by including the recipient's address and a timestamp in the OpenPGP content element[21]. It allows for both, signing and encrypting of the element. The scope of the specification was deliberately limited to OpenPGP.
Features like signed presences, which is provided by XEP-0027, may be added later on as add-on XEP to this.

8.4 Not using OpenPGP ASCII Armor

We decided against OpenPGP ASCII Armor (which contains an additional checksum) and in favor for Base64, because encoding should be part of the network application rather than the crypto layer. Also XMPP, needs no additional error correction of payload. In "MIME Security with OpenPGP" (RFC 3156 [22]), ASCII Armor has only been chosen to be backwards compatible with legacy applications supporting non-MIME OpenPGP emails only.

8.5 OpenPGP User IDs

OpenPGP User IDs normally consist of a name - email address pair, e.g., "Juliet <juliet@example.org>" (RFC 4880 § 5.11). For this XEP, we require User IDs of the format "xmpp:juliet@example.org". First, it is required to have at least one User ID indicating the use of this OpenPGP key. When doing certification of keys (key signing), the partner must know what User ID she actually certifies. Second, this format uses the standardized URI from XEP-0147 to indicate that this User ID corresponds to a key that is used for XMPP. Third, having the Real Name inside provides no additional security or guideline if this key should be certified. The XMPP address is the only trust anchor here.

9 Security Considerations

The scope of this XEP is intentionally limited, so that the specification just defines way for XMPP entities to discover, announce and synchronize OpenPGP keys, and how to exchange signed and encrypted data between two or more parties. Everything else is outside its scope. For example, how 'secure' the key material is protected on the endpoints is up to the implementation.
And while this XEP specifies a mechanism how to discover and retrieve a public key, it does not define how the trust relation to this key should be established. Even if key discovery and retrieval over XMPP provides a stronger coupling between the possessing entity (the

[20] XEP-0027: Current Jabber OpenPGP Usage <https://xmpp.org/extensions/xep-0027.html>.
[21] Full Replay attack prevention would require a counter based approach.
[22] RFC 3156: MIME Security with OpenPGP <http://tools.ietf.org/html/rfc3156>.

XMPP address) and the key, as compared to the OpenPGP keyservers, how a XMPP server authenticates a remote server is a server policy, which does vary from server to server. Implementation MUST provide a way for the user to establish and assign trust to a public key. For example by using a QR code shown on the recipient's device screen.
Besides the protocol defined herein, OpenPGP implementations are another big attack surface. Needless to say that the security of encrypted data exchanged using this protocol depends on the security of the used OpenPGP implementation. It is strongly RECOMMENED to use existing implementations instead of writing your own. OpenPGP implementations have suffered from various vulnerabilities in the past which opened up DoS attack vectors. For example CVE-2013-4402 and CVE-2014-4717.

10 IANA Considerations

This document requires no interaction with the Internet Assigned Numbers Authority (IANA) [23].

11 XMPP Registrar Considerations

11.1 Protocol Namespaces

The XMPP Registrar [24] includes 'urn:xmpp:openpgp:0' in its registry of protocol namespaces (see <https://xmpp.org/registrar/namespaces.html>).

12 XML Schema

TODO: Add after the XEP leaves the 'experimental' state.

13 Acknowledgements

Thanks to Emmanuel Gil Peyrot, Sergei Golovan, Marc Laporte, Georg Lukas, Adithya Abraham Philip, Brian Cully, fiaxh, Paul Schaub, Philipp Hörist and Stefan Kropp for their feedback.
The first draft of this specification was worked out and written on the wall of the 'Kymera'

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

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

room in one of Google's buildings by the authors, consisting of members of the XMPP Standards Foundation and the OpenKeychain project, at the GSOC Mentors Summit 2015. The authors would like to thank Google for making it possible by bringing the right people together.