XEP-0222: Persistent Storage of Public Data via PubSub

Peter Saint-Andre
mailto:stpeter@stpeter.im
xmpp:stpeter@jabber.org
https://stpeter.im/

2008-09-08
Version 1.0

Status	Type	Short Name
Active	Informational	N/A

This specification defines best practices for using the XMPP publish-subscribe extension to persistently store semi-public data objects such as public keys and personal profiles.

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

1.1 Motivation

Personal Eventing Protocol (XEP-0163) [1] introduced the idea of a virtual Publish-Subscribe (XEP-0060) [2] service associated with an IM user's bare JID <localpart@domain.tld>. However, the default node configuration options associated with PEP nodes are not optimized for the persistent storage of semi-public data objects such as public keys or user profiles. Therefore this document defines a set of best practices that enable IM users to persistently store semi-public data objects at their virtual pubsub service; in effect, we "sub-class" PEP by showing how a particular pubsub node can be configured for persisting objects.

1.2 How It Works

Imagine that you are a Shakespearean character named Juliet and that you want to persistently store information such as your public keys (see Public Key Publishing (XEP-0189) [3]) and user profile (see User Profile (XEP-0154) [4]).
We assume that you have three contacts with the following relationship to you:

1. benvolio@montague.lit, who has no subscription to your presence
2. nurse@capulet.lit, who has a bidirectional subscription to your presence and who is in your "Servants" roster group
3. romeo@montague.lit, who has a bidirectional subscription to your presence and who is in your "Friends" roster group

We also assume that your server (capulet.lit) supports PEP along with the "publish-options" feature, and that your client discovered that support when you logged in.
Because you want to keep your communications with Romeo confidential, you decide to start encrypting your messages. Therefore you reconfigure your client, which generates an RSA key that it publishes to the virtual pubsub service hosted at your bare JID <juliet@capulet.lit>.

Listing 1: Publishing a key

```
<iq from='juliet@capulet.lit/balcony' type='set' id='pub1'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <publish node='urn:xmpp:tmp:pubkey'>
      <item id='julietRSAkey1hash'>
        <key xmlns='urn:xmpp:tmp:pubkey'>
          <x509cert>
```

[1] XEP-0163: Personal Eventing Protocol <https://xmpp.org/extensions/xep-0163.html>.
[2] XEP-0060: Publish-Subscribe <https://xmpp.org/extensions/xep-0060.html>.
[3] XEP-0189: Public Key Publishing <https://xmpp.org/extensions/xep-0189.html>.
[4] XEP-0154: User Profile <https://xmpp.org/extensions/xep-0154.html>.

```
            der-encoded-cert
          </x509cert>
        </key>
      </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#persist_items'>
          <value>true</value>
        </field>
        <field var='pubsub#send_last_published_item'>
          <value>never</value>
        </field>
        <field var='pubsub#access_model'>
          <value>roster</value>
        </field>
        <field var='pubsub#roster_groups_allowed'>
          <value>Friends</value>
        </field>
      </x>
    </publish-options>
  </pubsub>
</iq>
```

Your publish request is a standard pubsub request except that:

1. The item is persisted (pubsub#persist_items is true).
2. The last published item is never sent (pubsub#send_last_published_item is set to ”never” so that items are pushed out only when modified).

(In this case, access is limited to people in your Friends roster group.)
If all goes well (see Publishing an Item), your key will be pushed out to all appropriate individuals (in this case only Romeo). In particular, Romeo receives your key because he has auto-subscribed to the virtual pubsub service at your bare JID via a presence subscription and because his Entity Capabilities (XEP-0115) [5] data indicated that he is interested in the ”urn:xmpp:tmp:pubkey” payload type.

Listing 2: Appropriate entities receive event notifications

```
<message from='juliet@capulet.lit'
         to='romeo@montague.lit/orchard'
```

[5] XEP-0115: Entity Capabilities <https://xmpp.org/extensions/xep-0115.html>.

```
        type='headline'
        id='keyfoo1'>
  <event xmlns='http://jabber.org/protocol/pubsub#event'>
    <items node='urn:xmpp:tmp:pubkey'>
      <item id='julietRSAkey1hash'>
        <key xmlns='urn:xmpp:tmp:pubkey'>
          <x509cert>
            der-encoded-cert
          </x509cert>
        </key>
      </item>
    </items>
  </event>
</message>
```

Because PEP services must send notifications to the account owner, you too receive the notification at each of your resources (here "balcony" and "chamber").

Listing 3: Publisher receives event notification

```
<message from='juliet@capulet.lit'
         to='juliet@capulet.lit/balcony'
         type='headline'
         id='keyfoo2'>
  <event xmlns='http://jabber.org/protocol/pubsub#event'>
    <items node='urn:xmpp:tmp:pubkey'>
      <item id='julietRSAkey1hash'>
        <key xmlns='urn:xmpp:tmp:pubkey'>
          <x509cert>
            der-encoded-cert
          </x509cert>
        </key>
      </item>
    </items>
  </event>
</message>

<message from='juliet@capulet.lit'
         to='juliet@capulet.lit/chamber'
         type='headline'
         id='keyfoo3'>
  <event xmlns='http://jabber.org/protocol/pubsub#event'>
    <items node='urn:xmpp:tmp:pubkey'>
      <item id='julietRSAkey1hash'>
        <key xmlns='urn:xmpp:tmp:pubkey'>
          <x509cert>
            der-encoded-cert
          </x509cert>
        </key>
```

```
      </item>
    </items>
  </event>
</message>
```

So that's the general idea.

2 Concepts and Approach

The best practices described herein re-use the concepts already defined in XEP-0060 and XEP-0163. In order to optimize for object persistence instead of transient event notifications, a node MUST be configured as follows:

1. Set pubsub#persist_items to true.
2. Set pubsub#send_last_published_item to "never".

The access model MAY be any model defined in XEP-0060.

3 Publishing an Item

An account owner publishes an item to a node by following the protocol specified in XEP-0060:

Listing 4: Account owner publishes item

```
<iq from='juliet@capulet.lit/balcony' type='set' id='pub1'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <publish node='urn:xmpp:tmp:pubkey'>
      <item id='julietRSAkey1hash'>
        <key xmlns='urn:xmpp:tmp:pubkey'>
          <x509cert>
            der-encoded-cert
          </x509cert>
        </key>
      </item>
    </publish>
  </pubsub>
</iq>
```

If the node does not already exist, the virtual pubsub service MUST create the node. As described in XEP-0163, this "auto-create" feature (defined in XEP-0060) MUST be supported by a PEP service. (Naturally, the account owner's client MAY follow the node creation use case specified in XEP-0060 before attempting to publish an item.)
In order for the client to reliably persist objects, the virtual pubsub service must also support

the "publish-options" feature defined in XEP-0060. Typically, a client will publish with options so that the object is properly persisted.

Listing 5: Publishing a key

```
<iq from='juliet@capulet.lit/balcony' type='set' id='pub1'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <publish node='urn:xmpp:tmp:pubkey'>
      <item id='julietRSAkey1hash'>
        <key xmlns='urn:xmpp:tmp:pubkey'>
          <x509cert>
            der-encoded-cert
          </x509cert>
        </key>
      </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#persist_items'>
          <value>true</value>
        </field>
        <field var='pubsub#send_last_published_item'>
          <value>never</value>
        </field>
        <field var='pubsub#access_model'>
          <value>roster</value>
        </field>
        <field var='pubsub#roster_groups_allowed'>
          <value>Friends</value>
        </field>
      </x>
    </publish-options>
  </pubsub>
</iq>
```

If the publication logic dictates that event notifications shall be sent, the account owner's server generates notifications and sends them to all appropriate entities as described in the Receiving Event Notifications section of XEP-0163.

4 Composition

Each item published to the node is a logically separate instance of the data to be stored. It is the responsibility of the publishing and receiving entities to construct a complete view of

all such items, if desired. For example, each bookmark published to a private data node is a separate piece of data, whereas the history of all items published to the node provides a complete list of the user's bookmarks. This history may include items that are republished with an existing ItemID (thus overwriting the previous version of that item).

5 Determining Support

Before an account owner attempts to complete the use cases defined herein, its client SHOULD verify that the account owner's server supports both PEP and the "publish-options" feature; to do so, it MUST send a Service Discovery (XEP-0030) [6] information request to the server (or cache Entity Capabilities information received from the server).

Listing 6: Account owner queries server regarding protocol support

```
<iq from='juliet@capulet.lit/balcony'
    to='capulet.lit'
    id='disco1'
    type='get'>
  <query xmlns='http://jabber.org/protocol/disco#info'/>
</iq>
```

The server MUST return an identity of "pubsub/pep" and include the "publish-options" feauture in the list of the namespaces and other features it supports:

Listing 7: Server communicates protocol support

```
<iq from='capulet.lit'
    to='juliet@capulet.lit/balcony'
    id='disco1'
    type='result'>
  <query xmlns='http://jabber.org/protocol/disco#info'>
    <identity category='server' type='im'/>
    <identity category='pubsub' type='pep'/>
    ...
    <feature var='http://jabber.org/protocol/pubsub#publish-options'/>
    ...
  </query>
</iq>
```

6 Security Considerations

This document introduces no security considerations above and beyond those specified in XEP-0060 and XEP-0163.

[6] XEP-0030: Service Discovery <https://xmpp.org/extensions/xep-0030.html>.

7 IANA Considerations

This document requires no interaction with the Internet Assigned Numbers Authority (IANA) [7].

8 XMPP Registrar Considerations

This document requires no interaction with the XMPP Registrar [8].

[7] 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/>.

[8] 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/>.