



XMPP

XEP-0348: Signing Forms

Peter Waher

<mailto:peterwaher@hotmail.com>

<xmpp:peter.waher@jabber.org>

<http://www.linkedin.com/in/peterwaher>

2017-09-11

Version 0.3

Status	Type	Short Name
Deferred	Standards Track	signing-forms

This specification describes a method whereby a client can sign a form using credentials not related to the current connection.

Legal

Copyright

This XMPP Extension Protocol is copyright © 1999 – 2018 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	Signing a form	1
2.1	Choose Hash function	3
2.2	Sort parameters	4
2.3	Calculating Parameter String (PStr)	4
2.4	Calculating Signature Base String (BStr)	4
2.5	Generating Signature	4
2.6	Sending Form	5
2.7	Validating Form	5
3	Use Cases	5
3.1	Securing In-Band Registration of new accounts	5
4	Determining Support	8
5	Implementation Notes	9
5.1	Signature algorithms and namespaces	9
6	Security Considerations	9
6.1	PLAINTEXT	9
6.2	Session parameters	9
7	IANA Considerations	10
8	XMPP Registrar Considerations	10
8.1	Field Standardization	10
9	Acknowledgements	11

1 Introduction

Sometimes there might be a need for an application to sign a data form ([Data Forms \(XEP-0004\)](#)¹) using other credentials than provided by the XMPP account or connection. An example can be automatic account creation using [In-Band Registration \(XEP-0077\)](#)². Many server operators disable the in-band registration feature since it makes it possible for malicious users or robots to freely create accounts on the server. One way to combat robots, has been through the use of [CAPTCHA Forms \(XEP-0158\)](#)³. But in some cases, like in the Internet of Things, it is not robots that are the problem, but malicious users.

This document describes a method whereby forms can be signed using other credentials. This can be used in an In-band registration form to sign the form with the credentials of a special account on the server with permissions to create new XMPP accounts, with perhaps a limit on number of accounts that can be created. This method can be used by manufacturers of devices for Internet of Things, so that devices can create accounts automatically on XMPP servers in an orderly fashion, and manufacturers are allowed to administer and control their automatically created accounts separately. It also provides a mechanism whereby server operators can monitor who is responsible for account creation and to what extent.

A fixed algorithm ([OAuth 1.0 Protocol](#)⁴) has been chosen in favor of a method where the user can select an authentication method from a list of available methods, modelled in the likeness of SASL. The main reason is to avoid multiple callbacks during form signature. The idea is to make form signature possible without having to do any intermediate server callbacks, or having to change the original request returning the form. The method is still extensible, allowing possible future extensions. The form signing algorithm to use is defined by the `FORM_TYPE` parameter in the form being signed.

2 Signing a form

A form that needs to be signed by the client using external credentials, shows this by including a hidden `FORM_TYPE` field valued `urn:xmpp:xdata:signature:oauth1`. The sub-namespace `:oauth1` identifies the algorithm to be used, in this case OAUTH v1.0.

Listing 1: Form requiring signature

```
<x xmlns='jabber:x:data' type='form'>
  <title>Create Account</title>
  <field type='hidden' var='FORM_TYPE'>
    <value>urn:xmpp:xdata:signature:oauth1</value>
  </field>
  <field type='hidden' var='oauth_version'>
```

¹XEP-0004: Data Forms <<https://xmpp.org/extensions/xep-0004.html>>.

²XEP-0077: In-Band Registration <<https://xmpp.org/extensions/xep-0077.html>>.

³XEP-0158: CAPTCHA Forms <<https://xmpp.org/extensions/xep-0158.html>>.

⁴RFC-5849: The OAuth 1.0 Protocol <<http://tools.ietf.org/html/rfc5849>>.

```

    <value>1.0</value>
  </field>
  <field type='hidden' var='oauth_signature_method'>
    <value>HMAC-SHA1</value>
  </field>
  <field type='hidden' var='oauth_token'>
    <value>...<value>
  </field>
  <field type='hidden' var='oauth_token_secret'>
    <value>...<value>
  </field>
  <field type='hidden' var='oauth_nonce'>
    <value/>
  </field>
  <field type='hidden' var='oauth_timestamp'>
    <value/>
  </field>
  <field type='hidden' var='oauth_consumer_key'>
    <value/>
  </field>
  <field type='hidden' var='oauth_signature'>
    <value/>
  </field>
  <field type='text-single' label='User_Name' var='UserName'>
    <required/>
  </field>
  <field type='text-single' label='Password' var='Password'>
    <required/>
  </field>
</x>

```

The form contains a set of invisible parameters that the underlying software signing the form and supporting this specification must understand. These are:

Name	Description
oauth_version	Must be 1.0. Is not changed by the client performing the signing.
oauth_signature_method	Specifies the signature method, or hash function, to use when signing the form. This can be changed by the client. Possible values are: HMAC-SHA1, RSA-SHA1 and PLAINTEXT.
oauth_token	This is a token provided by the server to the client. This parameter might not be available if the server has provided the client with this token earlier during the session.
oauth_token_secret	This is a temporary secret shared between the server and client, and is related to the token. This parameter might not be available if the server has provided the client with this token earlier during the session.

Name	Description
oauth_nonce	A nonce value that the client has to set. Can be a random alphanumeric string.
oauth_timestamp	Number of seconds since 1st of January 1970, 00:00:00 GMT. The client has to set this at the time of signature.
oauth_consumer_key	A key identifying the account doing the signing of the form. The client has to set this to identify who performs the signature.
oauth_signature	The signature, signing the form. The client has to set this with the signature of the form, as calculated and described below.

Before calculating the signature, the following functions need to be defined:

Base64(x) Converts the sequence of octets x into a Base-64 encoded string.

Escape(s) The string s are escaped using the RFC 3986 RFC 3986: Uniform Resource Identifiers (URI): Generic Syntax <<http://tools.ietf.org/html/rfc3986>>. percent-encoding (%xx) mechanism. Characters not in the unreserved character set (§ 2.3) MUST be encoded. Characters in the unreserved character set MUST NOT be encoded. Hexadecimal characters in encodings MUST be upper case. Text names and values MUST first be normalized using Normalization Form C (NFC) as defined in Unicode Standard Annex #15, Unicode Normalization Forms

Unicode Standard Annex #15, Unicode Normalization Forms <http://unicode.org/reports/tr15/#Norm_Forms>. and then encoded as UTF-8 octets before percent-encoding them per RFC 3629 RFC 3629: UTF-8, a transformation format of ISO 10646 <<http://tools.ietf.org/html/rfc3629>>..

Unreserved characters are alphanumeric characters (a-z, A-Z, 0-9), hyphen ('-'), period ('.'), underscore ('_') and tilde ('~').

H(x) Corresponds to the Hash-function to use, according to the parameter `oauth_signature_method`. It computes the hash value of the octets x, and returns a sequence of octets.

To calculate the signature of the form, follow the steps in the following subsections, in the same order.

2.1 Choose Hash function

You choose Hash function by setting the parameter `oauth_signature_method` to either **HMAC-SHA1**, **RSA-SHA1** or **PLAINTEXT**.

2.2 Sort parameters

First, all form parameters to be signed, including hidden parameters and the OAUTH parameters except `oauth_token_secret` and `oauth_signature`, are sorted by variable name (**var**).

2.3 Calculating Parameter String (PStr)

Each (name, value) pair in the list of sorted parameters are first transformed into pairs of **Escape(name)=Escape(value)** segments, and then concatenated into one string, where each segment is delimited using an ampersand ('&') character.

Listing 2: Calculating Parameter String

```
PStr := Escape(name_1) + '=' + Escape(value_1) + '&' + ... + Escape(name_N) + '=' +
      Escape(value_N)
```

2.4 Calculating Signature Base String (BStr)

The Signature Base String (BStr) is then formed concatenating **Escape(type)** (the form type used when submitting the form), **Escape(to)** (the full destination address, including resource, if any) and **Escape(PStr)**, using ampersands ('&') as delimiter.

Listing 3: Calculating Signature Base String

```
BStr := Escape(type) + '&' + Escape(to) + '&' + Escape(PStr)
```

2.5 Generating Signature

Depending on which signature method is chosen, different Hash functions are used. If **HMAC-SHA1** is chosen, then the Hash function **HMAC(text, key)** according to [RFC 2104](#)⁵ is used as follows:

Listing 4: HMAC-SHA1

```
H(x) := HMAC-SHA1(x, Escape(oauth_consumer_secret) + '&' + Escape(
  oauth_token_secret))
  oauth_signature := Escape(Base64(H(BStr)))
```

Here, the client signing the form must have the `oauth_consumer_secret` available to be able to sign the form correctly.

If **RSA-SHA1** is chosen, then the signature method **RSASSA-PKCS1-v1_5(K, M)** according to

⁵RFC 2104: HMAC: Keyed-Hashing for Message Authentication <<http://tools.ietf.org/html/rfc2104>>.

RFC 3447⁶ is used as follows:

Listing 5: RSA-SHA1

```
H(x) := RSASSA-PKCS1-v1_5(oauth_consumer_secret, x)
oauth_signature := Escape(Base64(H(BStr)))
```

In this example, the **oauth_consumer_secret** must be an RSA private key. If **PLAINTEXT** is chosen, no Hash function is used. Instead, the signature is calculated as follows:

Listing 6: RSA-SHA1

```
oauth_signature := Escape(oauth_consumer_secret) + Escape(
    oauth_token_secret)
```

2.6 Sending Form

Once the signature has been calculated, the corresponding parameter **oauth_signature** has to be set in the form before sending it to the destination address.

2.7 Validating Form

When the recipient receives the signed form it performs the same calculations as above, based on the parameters received, and knowledge of the shared secret which it can look up by using the parameter **oauth_consumer_key**.

3 Use Cases

3.1 Securing In-Band Registration of new accounts

As mentioned above, a major use case for signing forms, is in-band registration for creating new accounts on an XMPP Server, as defined in XEP-0077.

Listing 7: Entity Requests Registration Fields from Host

```
<iq type='get'
    from='juliet@capulet.com/balcony'
    to='contests.shakespeare.lit'
    id='reg3'>
  <query xmlns='jabber:iq:register' />
</iq>
```

⁶RFC 3447: Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1 <<http://tools.ietf.org/html/rfc3447>>.

Listing 8: Host Returns Registration Form to Entity requiring signature

```

<iq type='result'
  from='contests.shakespeare.lit'
  to='juliet@capulet.com/balcony'
  id='reg3'>
  <query xmlns='jabber:iq:register'>
    <instructions>
      Use the enclosed form to register. If your Jabber client
      does not
      support Data Forms, visit http://www.shakespeare.lit/contests.php
    </instructions>
    <x xmlns='jabber:x:data' type='form'>
      <title>Contest Registration</title>
      <instructions>
        Please provide the following information
        to sign up for our special contests!
      </instructions>
      <field type='hidden' var='FORM_TYPE'>
        <value>urn:xmpp:xdata:signature:oauth1</value>
      </field>
      <field type='text-single' label='Given_Name' var='first'>
        <required/>
      </field>
      <field type='text-single' label='Family_Name' var='last'>
        <required/>
      </field>
      <field type='text-single' label='Email_Address' var='email'>
        <required/>
      </field>
      <field type='list-single' label='Gender' var='x-gender'>
        <option label='Male'><value>M</value></option>
        <option label='Female'><value>F</value></option>
      </field>
      <field type='hidden' var='oauth_version'>
        <value>1.0</value>
      </field>
      <field type='hidden' var='oauth_signature_method'>
        <value>HMAC-SHA1</value>
      </field>
      <field type='hidden' var='oauth_token'>
        <value>...<value>
      </field>
      <field type='hidden' var='oauth_token_secret'>
        <value>...<value>
      </field>
      <field type='hidden' var='oauth_nonce'>
        <value/>
      </field>
    </x>
  </query>
</iq>

```

```

    <field type='hidden' var='oauth_timestamp'>
      <value/>
    </field>
    <field type='hidden' var='oauth_consumer_key'>
      <value/>
    </field>
    <field type='hidden' var='oauth_signature'>
      <value/>
    </field>
  </x>
</query>
</iq>

```

Listing 9: User Submits Signed Registration Form

```

<iq type='set'
  from='juliet@capulet.com/balcony'
  to='contests.shakespeare.lit'
  id='reg4'>
  <query xmlns='jabber:iq:register'>
    <x xmlns='jabber:x:data' type='submit'>
      <field type='hidden' var='FORM_TYPE'>
        <value>urn:xmpp:xdata:signature:oauth1</value>
      </field>
      <field type='text-single' label='Given_Name' var='first'>
        <value>Juliet</value>
      </field>
      <field type='text-single' label='Family_Name' var='last'>
        <value>Capulet</value>
      </field>
      <field type='text-single' label='Email_Address' var='email'>
        <value>juliet@capulet.com</value>
      </field>
      <field type='list-single' label='Gender' var='x-gender'>
        <value>F</value>
      </field>
      <field type='hidden' var='oauth_version'>
        <value>1.0</value>
      </field>
      <field type='hidden' var='oauth_signature_method'>
        <value>HMAC-SHA1</value>
      </field>
      <field type='hidden' var='oauth_token'>
        <value>...<value>
      </field>
      <field type='hidden' var='oauth_token_secret'>
        <value>...<value>
      </field>
      <field type='hidden' var='oauth_nonce'>
        <value>...<value>

```

```

    </field>
    <field type='hidden' var='oauth_timestamp'>
      <value>...<value>
    </field>
    <field type='hidden' var='oauth_consumer_key'>
      <value>...<value>
    </field>
    <field type='hidden' var='oauth_signature'>
      <value>...<value>
    </field>
  </x>
</query>
</iq>

```

In case a form signature is not value, the server MUST respond with a **bad-request** error message, as follows:

Listing 10: Error message when form signature is invalid

```

<iq type='error'
  from='contests.shakespeare.lit'
  to='juliet@capulet.com/balcony'
  id='reg4'>
  <error code='400' type='modify'>
    <bad-request xmlns='urn:iETF:params:xml:ns:xmpp-stanzas' />
  </error>
</iq>

```

4 Determining Support

If an entity supports signing forms as specified herein, it MUST advertise that fact by returning a feature of "urn:xmpp:xdata:signature:oauth1" in response to [Service Discovery \(XEP-0030\)](#)⁷ information requests.

Listing 11: Service discovery information request

```

<iq type='get'
  from='example.org'
  to='device@example.org'
  id='disco1'>
  <query xmlns='http://jabber.org/protocol/disco#info' />
</iq>

```

Listing 12: Service discovery information response

⁷XEP-0030: Service Discovery <<https://xmpp.org/extensions/xep-0030.html>>.

```
<iq type='result'
  from='device@example.org'
  to='example.org'
  id='disco1'>
  <query xmlns='http://jabber.org/protocol/disco#info'>
    ...
    <feature var='urn:xmpp:xdata:signature:oauth1' />
    ...
  </query>
</iq>
```

In order for an application to determine whether an entity supports this protocol, where possible it SHOULD use the dynamic, presence-based profile of service discovery defined in [Entity Capabilities \(XEP-0115\)](#)⁸. However, if an application has not received entity capabilities information from an entity, it SHOULD use explicit service discovery instead.

5 Implementation Notes

5.1 Signature algorithms and namespaces

This document only specifies signatures using OAUTH v1.0. Each entity that supports form signatures using this method, reports this by including the namespace **urn:xmpp:xdata:signature:oauth1** in its features. The specification is extensible, in that it can allow for future signature methods to be used, by defining new namespaces replacing **oauth1** by another string corresponding to the desired method, but leaving the base part of the namespace **urn:xmpp:xdata:signature:** intact.

6 Security Considerations

6.1 PLAINTEXT

The PLAINTEXT signature method should only be used if SSL/TLS is used by both the entity signing the form as well as the creator of the form. If the creator of the form is a server, this later part can be ignored. If unsure, PLAINTEXT should only be used in development & debugging cycles of an application, and not in production environments.

6.2 Session parameters

If the server provides information to be used in signing a form, it must also verify that the client only changes values it is allowed to change. An alternative is to not use the values provided by the client for the corresponding server-side parameters when calculating the

⁸XEP-0115: Entity Capabilities <<https://xmpp.org/extensions/xep-0115.html>>.

signature on the server side. This makes sure a client cannot willfully alter server-side parameters when it returns the signed form.

7 IANA Considerations

This document requires no interaction with the [Internet Assigned Numbers Authority \(IANA\)](#)⁹.

8 XMPP Registrar Considerations

The [XMPP Registrar](#)¹⁰ includes the following information in its registries.

8.1 Field Standardization

[Field Standardization for Data Forms \(XEP-0068\)](#)¹¹ defines a process for standardizing the fields used within Data Forms qualified by a particular namespace, and XEP-0128 describes how to use field standardization in the context of service discovery. This section registers fields for server information scoped by the "urn:xmpp:xdata:signature:oauth1" FORM_TYPE.

Listing 13: Registry submission

```
<form_type>
  <name>urn:xmpp:xdata:signature:oauth1</name>
  <doc>XEP-xxxx</doc>
  <desc>
    Forms that require signatures using OAuth v1.0 signature algorithm
  </desc>
  <field
    var='oauth_version'
    type='hidden'
    label='OAuth_version_Must_be_1.0.'/>
  <field
    var='oauth_signature_method'
    type='hidden'
```

⁹The 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/>.

¹⁰The 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/>.

¹¹XEP-0068: Field Data Standardization for Data Forms <https://xmpp.org/extensions/xep-0068.html>.

```

        label='OAuth_signature_method._Can_be_"HMAC-SHA1",_"RSA-SHA1"_or
            _"PLAINTEXT".' />
    <field
        var='oauth_token'
        type='hidden'
        label='OAuth_token._Provided_by_the_creator_of_the_form.' />
    <field
        var='oauth_token_secret'
        type='hidden'
        label='OAuth_token_secret._Provided_by_the_creator_of_the_form.'
        />
    <field
        var='oauth_nonce'
        type='hidden'
        label='Nonce_value._Random_string,_created_by_the_entity_signing
            _the_form.' />
    <field
        var='oauth_timestamp'
        type='hidden'
        label='Seconds_since_1st_of_January_1970,_00:00:00_GMT._Value_
            provided_by_the_entity_signing_the_form.' />
    <field
        var='oauth_consumer_key'
        type='hidden'
        label='Consumer_Key,_identifying_the_entity_signing_the_form_
            using_an_external_set_of_credentials._Value_provided_by_the_
            entity_signing_the_form.' />
    <field
        var='oauth_signature'
        type='hidden'
        label='Form_signature._Value_provided_by_the_entity_signing_the_
            form.' />
</form_type>

```

9 Acknowledgements

Thanks to Kevin Smith, Lance Stout, Matthew Wild, Philipp Hancke and Tobias Markmann for all valuable feedback.