

SlashID Protocol v0.1

SlashID Corp.

Contents

1	Conventions	2
1.1	Actors	2
1.2	Messages	3
1.3	Transport	3
1.4	GWT Serialization	4
2	Cryptographic computations	4
2.1	Encoding	4
2.2	Cryptographic algorithms	4
2.3	User Handle	5
2.4	Master secret	5
2.5	Shared Secrets	5
2.6	Session Storage	6
3	Protocol Overview	6
3.1	User Create	6
3.2	User Register	6
3.3	Login	6
3.4	Profile Update	6
3.5	Subscription Cancellation	6
4	Protocol Messages	7
4.1	pwdencrypt	7
4.1.1	Request parameters	7
4.1.2	Response	7
4.2	usrcreate	8
4.2.1	Request parameters	8
4.2.2	Response	9
4.3	getregisterurl	10
4.3.1	Request parameters	11
4.3.2	Response	11
4.4	usrreg	11
4.4.1	Request parameters	11
4.4.2	Response	12
4.5	usrwspreg	12

4.5.1	Request parameters	12
4.5.2	Response	12
4.6	wspreg	12
4.6.1	Request parameters	13
4.6.2	Response	13
4.7	getusrsecret	13
4.7.1	Request parameters	13
4.7.2	Response	13
4.8	storesession	14
4.8.1	Request Parameters	14
4.8.2	Response	14
4.9	usrlogin	14
4.9.1	Request parameters	14
4.9.2	Response	14
4.10	wsplogin	14
4.10.1	Request parameters	15
4.10.2	Response	15
4.11	wspuserupdate	15
4.11.1	Request parameters	15
4.11.2	Response	16
4.12	usrcancel	16
4.12.1	Request parameters	16
4.12.2	Response	16
4.13	wspusercancel	16
4.13.1	Request parameters	17

1 Conventions

1.1 Actors

The protocol described in this spec is performed between three parties: The User, the IdP and the WSP.

The User is the regular Internet user with a browser.

The IdP is the Identity Provider - a party whoes goal is to provide Identity Management service.

The WSP is the Web Service Provider. This is the consumer of the Identity Management service, and the provider of some other service, such as e-commerce or online banking.

1.2 Messages

The protocol consists of a sequence of messages exchanged by any two parties (of the three defined above). Each message exchange takes a form of request-reply conversation. Both request and reply contain a number of pre-defined values; In addition to these values, reply message may contain additional status information.

Each message is defined by its name and the values it contains.

1.3 Transport

The messages are exchanged as HTTP POST requests, except where indicated. Any request values are passed as form parameters in the HTTP POST request. The name of the message is the last URL component of the HTTP request. Return messages are serialized in XML format, and the values are encoded as defined in this spec.

By convention, messages that begin with `usr` are sent by the User to the IdP; messages that begin with `wsp` are sent by the WSP to the IdP. When IdP or User send messages to the WSP, they are defined separately. Since the User uses Web Browser, nobody can initiate message exchange with the User - it is only possible to respond to User's messages.

All requests to the IdP MUST be made over a private and secure channel, via HTTPS. The IdP MUST NOT respond to any requests over cleartext channels such as HTTP.

WSP will have a choice whether to work over HTTP or HTTPS. If the WSP chooses HTTPS, all requests to it MUST be made over HTTPS. A WSP that has HTTPS enabled MAY choose not to respond to any cleartext (HTTP) requests.

If the WSP has chosen to use HTTP, the security will be equivalent to passwords sent over a cleartext channel - i.e. any party listening on the channel will be able to acquire the shared keys. However, this will not affect other WSPs as the shared secrets are independent.

Example of request invocation in JavaScript:

```
function loadURLAsync(url, names, values, handler)
{
    var req = new XMLHttpRequest();

    req.onreadystatechange = function()
        { processReqChange(handler, req) };
    req.open("POST", url, true);
    content = "";
    var i;

    for (i=0; i<names.length; i++)
    {
```

```

        if (i>0)
            content += "&";
        content += encodeURIComponent(names[i]) + "=" +
            encodeURIComponent(values[i]);
    }
    req.setRequestHeader('Content-type',
        'application/x-www-form-urlencoded');
    req.send(content);
}

```

1.4 GWT Serialization

As of the current version, some messages are exchanged using GWT serialization. These messages are documented separately, and marked as GWT in this spec.

In next version, all GWT messages will be ported to plain HTTP POST to promote alternative client-side implementations.

2 Cryptographic computations

2.1 Encoding

Unless otherwise noted, all hashed and encrypted values, as well as cleartext binary data are transmitted in Base64 encoding. No newlines are allowed in the Base64 data for Handles and Secrets.

2.2 Cryptographic algorithms

The following cryptographic algorithms are used in this protocol:

HASH SHA256

ENCRYPT AES256 symmetric encryption. Written as

$ciphertext = ENCRYPT(key, plaintext)$

CBC mode is used to encrypt readable strings (such as fields in the profile). When encrypting secrets (Shared Secret, Master Secret), ECB mode is used on the binary data (not Base64 encoding).

PBE Password Based Encryption, SHA256 based KDF and AES256. The KDF in this version is proprietary due to complexity of implementing full PKCS5/12 ASN.1 blocks in JavaScript. The KDF is performed as follows (pseudocode):

```

key := HASH(salt || password);
for i:=1 to it_count
    key := HASH(key);

```

```
end
return key;
```

The resulting key is used as the Symmetric Key for the AES encryption.

SIG Digital Signature. RSA 1024 bit signatures are used in the current version.

2.3 User Handle

Each User is identified to each WSP as a Handle. The handle is computed as:

$$H = \text{HASH}(\text{username}||\text{url})$$

Where *username* is the name of the user which he types when logging in, and *url* is the canonical form of the url he is trying to log into. The || signifies concatenation.

2.4 Master secret

The Master Secret (*MS*) is generated on the User's side when the User is registering. The Master Secret is encrypted in three layers as:

$$\text{tempKey} = \text{HASH}(\text{"PasswordEncryption"}||\text{slashIDKey}||\text{usrHandle})$$

$$\text{encMS} = \text{PBE}(\text{ENCRYPT}(\text{tempKey}, \text{PBE}(\text{MS}, \text{pwd}), \text{pwd}))$$

where *pwd* is User's password, *usrHandle* is SlashID User Handle, and *slashIDKey* is a secret key of the SlashID server.

To improve JavaScript efficiency, the inner and outer PBE use the same salt, and as a result, have the same derived key. The middle encryption is done on the SlashID side, while the inner and the outer ones are done in the browser.

2.5 Shared Secrets

Shared secrets are generated for each pair of (User, WSP) that wish to establish a relationship. The User's copy of shared secret is encrypted as

$$\text{encSecret} = \text{ENCRYPT}(\text{MS}, \text{secret})$$

The WSP's copy of the shared secret is encrypted as

$$\text{encSecret} = \text{ENCRYPT}(\text{wspKey}, \text{secret})$$

Where *wspKey* is the WSP's secret key, and is not part of the protocol. The *encSecret* value of the WSP's copy is also digitally signed using WSP's private key:

$$\text{wspSecretSig} = \text{SIGN}(\text{"Secret"}||\text{H}||\text{""}||\text{encSecret})$$

H is defined in Section 2.3, and is used in its encoded form (as per Section 2.1)

2.6 Session Storage

In order to allow for Single Sign On, the Master Secret must be securely stored in the browser. Since Cookies cannot be used (otherwise Master Secret will be sent back to SlashID), sensitive cryptographic data is stored using a client-side storage, which is a browser specific mechanism. For browsers that do not support client-side storage (such as Firefox 1.5), JavaScript variable is used.

To protect against leakage of sensitive information through client-side database (for example in the event of power outage while Master Secret is stored), a secret-splitting scheme between the browser and SlashID is implemented. Master Secret is encrypted with an ephemeral key (using AES 256). The key is sent to SlashID, while the ciphertext is stored in the browser's client-side storage.

3 Protocol Overview

The main SlashID protocol consists of five basic operations: Create, Register, Login, Update and Cancel. This section outlines the message flows; the following section contains the details of the messages.

3.1 User Create

This flow is executed when a new user is created with the IdP. WSP is not part of the flow. The flow is shown on figure 1.

3.2 User Register

This flow is executed when the User wishes to establish a relationship with a WSP. The flow is shown on figure 2.

3.3 Login

This flow is invoked when the user logs into a WSP. The flow is shown on figure 3.

3.4 Profile Update

This flow is executed when the User updates a profile within the IdP. The flow is shown on Figure 4.

3.5 Subscription Cancellation

This flow is executed when the User wants to cancel a subscription with a WSP. The flow is shown on Figure 5.

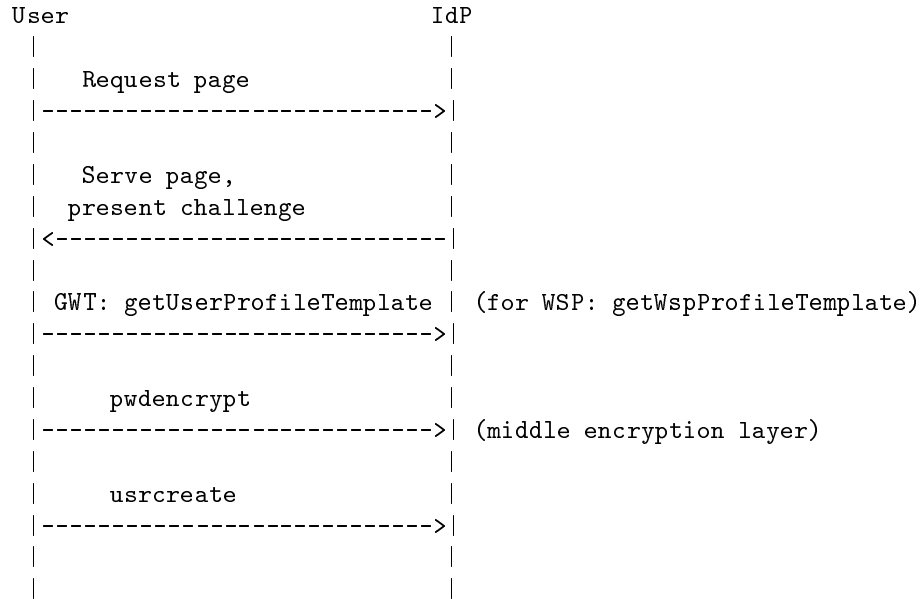


Figure 1: User Create Flow

4 Protocol Messages

This section describes detailed contents of each protocol message.

4.1 pwdencrypt

This call is used in the three-layer encryption of the Master Secret. It executes the second, or middle layer, which requires the knowledge of the SlashID key.

4.1.1 Request parameters

handle (REQUIRED) Handle of the user for whom the encryption or decryption is done.

operation (REQUIRED) Either "encrypt" or "decrypt".

input (REQUIRED) Base64 encoded encrypted value, that needs to be either further encrypted or decrypted.

4.1.2 Response

The response will contain the result of encryption or decryption in the following form:

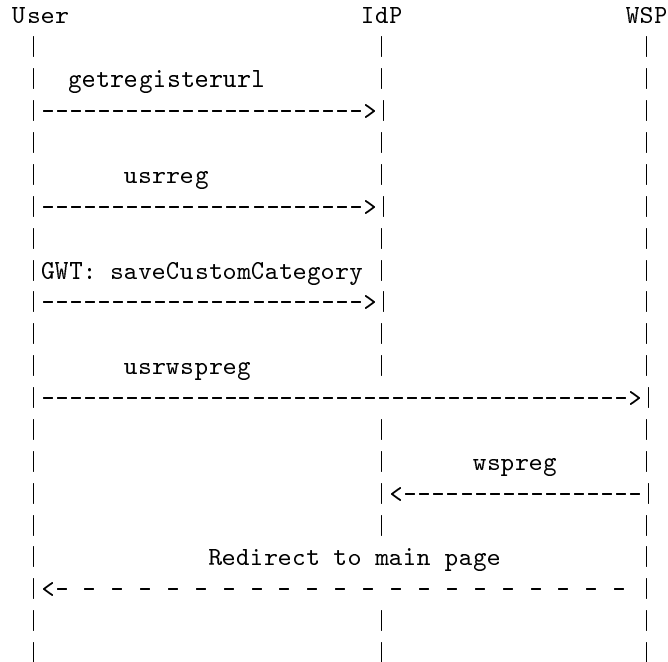


Figure 2: User Registration Flow

```

<password-response>
  <result>VALUE</result>
</password-response>
  
```

Where VALUE is the result of the operation.

4.2 usrcreate

Creates a new User with the IdP. This message is sent by the User to the IdP in order to establish an account.

4.2.1 Request parameters

secret (REQUIRED) A cleartext shared secret to be used when logging into the IdP. The secret MUST be a 32 byte (256 bits) number, encoded as per Section 2.1.

encSecret (REQUIRED) An encrypted value of **secret** (above). The encrypted secret is computed as: $encSecret = ENCRYPT(MS, secret)$

captcha (REQUIRED) The text of the CAPTCHA picture shown to the user.

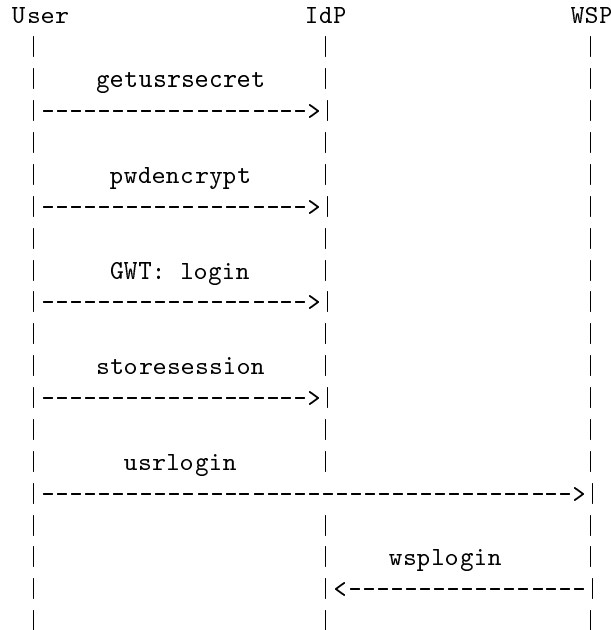


Figure 3: Login Flow

userXml (REQUIRED) User's data in XML format (schema will be published separately)

termsAccepted (REQUIRED) "true" if the user has accepted Terms and Conditions

wspType (REQUIRED) If website was built with one of supported plugins (such as Wordpress or MediaWiki), this is the type of plugin used will be used as prefix to obtain

4.2.2 Response

This message produces status response of the form:

```
<status>OK</status>
```

In case of errors, the value inside the **status** tag MUST be the error description. In case of a successful registration, this value MUST be "OK" (no quotes).

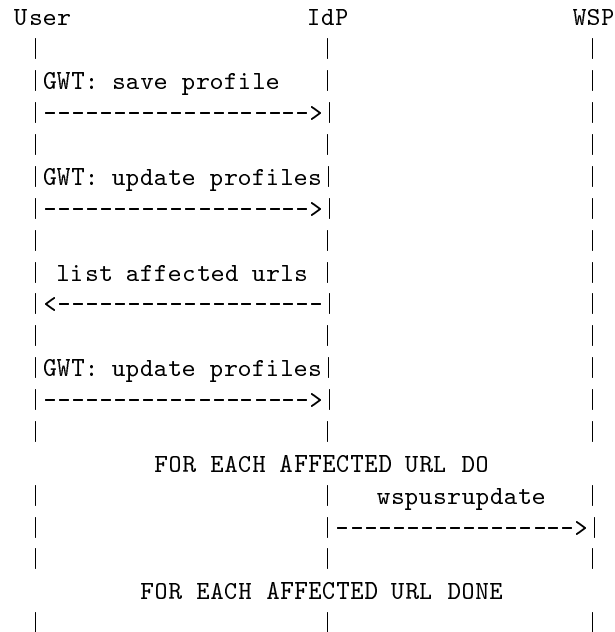


Figure 4: Update Flow

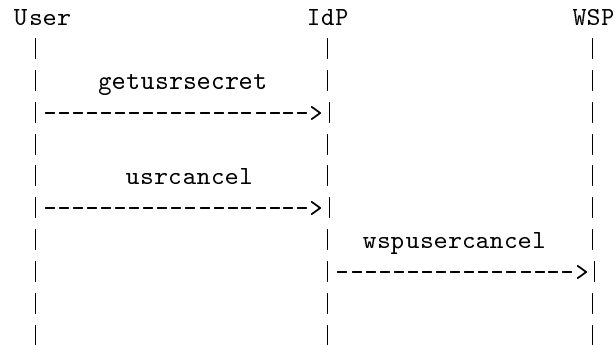


Figure 5: Cancel flow

4.3 getregisterurl

This message is sent from the User to the IdP. In return, the IdP sends any information required for a successful registration with a website.

4.3.1 Request parameters

url (REQUIRED) The URL that the User is trying to register with.

4.3.2 Response

The results are returned as an XML document of the following form:

```
<register-url>
  <protocol>value</protocol>
  <url>value</url/>
  <userIdMessage>value</userIdMessage/>
  <profileTemplate>value</profileTemplate>
</register-url>
```

The values are:

protocol The protocol used by this WSP, either http or https.

url The registration URL, relative to the main url of the website.

userIdMessage Message to display in order to prompt the user for his Access Code.

profileTemplate The template of the profile that contains all the fields required by this website. This is an XML document, escaped and included as CDATA.

4.4 usrreg

Part of the process to establish a relationship between a User and a WSP. This message is sent by the User to the IdP.

4.4.1 Request parameters

secret (REQUIRED) Encrypted shared secret, to be used as a password between the User and the WSP he is registering with. The secret is computed as the **encSecret** value in Section 4.2.1.

url (REQUIRED) The URL of the WSP the User wishes to register with.

encProfile (REQUIRED) The encrypted (field-by-field) profile to be used with this registration.

profileId (REQUIRED) Integer id of the profile used.

4.4.2 Response

In case of an error, the response MUST be as in Section 4.2.2. In case of success, the response MUST be an XML document in the following format:

```
<register-response>
  <ticket>value</ticket>
</register-response>
```

The values are:

ticket Contains a registration ticket generated by the IdP.

4.5 usrwspreg

This message is sent from the User directly to the WSP during User registration flow. The message takes form of HTTP or HTTPS form submit (method is POST). Parameters are encoded as elements in the submitted form.

4.5.1 Request parameters

usrHandle (REQUIRED) Handle of the User. See section 2.3.

secret (REQUIRED) Cleartext Shared Secret, generated by the User. This will be used as a password between the WSP and the User.

profile (REQUIRED) Cleartext profile that the User agrees to expose to the WSP

ticket (REQUIRED) Registration ticket, as provided by the IdP.

userId (OPTIONAL) The UserID value, provided by the WSP to the User out-of-band. This may be a banking card number.

accessCode (OPTIONAL) A secret value provided by the WSP to the User out-of-band, used to securely bind the **usrHandle** to the actual customer record existing on the WSP side.

4.5.2 Response

Once the form has been submitted, the WSP takes over the browser, and returns any HTML page. The returned page SHOULD indicate whether or not the login was successful, and in case of a successful login, redirect the user to the members area.

4.6 wspreg

Sent by the WSP to the IdP whenever the User is trying to register with the WSP.

4.6.1 Request parameters

usrHandle (REQUIRED) User's Handle for this WSP. See Section 2.3.

secret (REQUIRED) Shared Secret for the WSP, generated by the User, encrypted with WSP's key. See Section 2.5.

url (TEMPORARY) The URL of this WSP.

ticket (REQUIRED) Registration Ticket generated by the IdP.

signature (REQUIRED) WSP's signature of the shared secret (See Section 2.5).

profileSig (REQUIRED) WSP's signature of the User's Profile.

profile (OPTIONAL) WSP's version of the profile, encrypted with WSP key.

4.6.2 Response

Same as for `usrcreate` (See Section 4.2.2).

4.7 getusrsecret

This message is sent by the User to the IdP when a User's shared secret is needed for a transaction.

4.7.1 Request parameters

usrHandle (REQUIRED) The Handle of the User for which the Shared Secret is needed. See section 2.3.

4.7.2 Response

The response is of the following form:

```
<user-shared-secret-response>
  <x>value</x>
  <e>value</e>
</user-shared-secret-response>
```

where

- x The Encrypted Master Secret for this User (see Section 2.4). If the Handle cannot be found (User doesn't exist), the return value MUST be generated as:

$$x = \text{HMAC}("DUMMY - MASTER" || \textit{handle}, \textit{key})$$

Where *handle* is the value of `usrHandle` parameter, and *key* is IdP's secret key. This key does not need to be the same as the real encryption key, but needs to be consistent across multiple requests.

- e The Encrypted Shared Secret (User's copy) for this URL. If the Handle cannot be found (User doesn't exist), or the User is not registered with this URL (they have no relationship), the value MUST be generated as
$$e = \text{HMAC}("DUMMY - SHARED" || \text{handle} || \text{url}, \text{key})$$

4.8 storesession

This message is sent by the User to the IdP in order to store the server part of the split Master Secret.

4.8.1 Request Parameters

sessionKey (REQUIRED) The key to store in the session.

4.8.2 Response

Same as for `usrcreate` (See Section 4.2.2).

The key will be stored in the HTTP session, and returned in a JavaScript variable whenever the window is refreshed in the browser.

4.9 usrlogin

This message is sent by the User directly to the WSP to complete the Login process. The message takes form of HTTP or HTTPS form submit (method is POST). Parameters are encoded as elements in the submitted form.

4.9.1 Request parameters

usrHandle (REQUIRED) The Handle of the user (for this WSP) that is trying to login.

secret (REQUIRED) Cleartext version of the shared secret for this WSP.

4.9.2 Response

Once the form has been submitted, the WSP takes over the browser, and returns any HTML page. The returned page SHOULD indicate whether or not the login was successful, and in case of a successful login, redirect the user to the members area.

4.10 wsplogin

This message is sent by the WSP to the IdP whenever a User attempts to log in. This message is only required for the WSPs that use SlashID to store the shared secret. If the WSP can look up the shared secret in his own database, this message does not need to be sent.

4.10.1 Request parameters

usrHandle (REQUIRED) The Handle of the user which is trying to log in (see Section 2.3).

4.10.2 Response

In case of an error, the response MUST be as in Section 4.2.2. In case of success, the response MUST be an XML document in the following format:

```
<shared-secret-response>
  <encrypted-secret>value</encrypted-secret>
  <signature>value</signature>
  <profile>value</profile>
  <profile-sig>value</profile-sig>
</shared-secret-response>
```

The values are:

encrypted-secret WSP's copy of User's Shared Secret (encrypted). See Section 2.5

signature WSP's signature of the shared secret. See Section 2.5.

profile User's profile (as seen by WSP), encrypted with WSP's Secret Key.

profile-sig WSP's signature of the profile.

4.11 wspusrupdate

This message is sent by the IdP to the WSP whenever the user updates his profile.

4.11.1 Request parameters

usrHandle (REQUIRED) Handle of the user for whom the update is performed.

profile (REQUIRED) The new version of the profile. The whole XML document (profile) is encrypted using Shared Secret as a key, and the ciphertext is Base64-encoded.

secret (REQUIRED) Shared secret, encrypted with WSP's secret key.

signature (REQUIRED) Shared secret signature, as signed by the WSP.

4.11.2 Response

The WSP MUST verify the shared secret signature, and return an error if the verification was not successful.

In case of a success, the following document is returned by the WSP:

```
<update-result>
  <profile>value</profile>
  <profileSig>value</profileSig>
</update-result>
```

The values are as follows:

profile Same profile as in request, but this time encrypted with WSP's secret key

profileSig The value of the **profile** field above, signed with WSP's private key.

4.12 usrcancel

Sent by the User to the IdP whenever the user wishes to cancel the subscription with a WSP.

4.12.1 Request parameters

usrHandle User's handle for the URL to be cancelled

url URL to be cancelled

cancelCode Cancel code, computed as follows:

$$cancelCode = HASH("Cancel" || handle || " || secret)$$

Where *handle* is the handle of the user for this URL, and *secret* is the shared secret. Note: if the user re-subscribes to the WSP after one cancellation, the IdP is able to re-play this message.

4.12.2 Response

Same as for **usrcreate** (See Section 4.2.2).

4.13 wspusercancel

This message is sent by the IdP to the WSP whenever the User wishes to cancel the subscription.

4.13.1 Request parameters

usrHandle (REQUIRED) Handle of the user that wishes to cancel the subscription

secret (REQUIRED) Encrypted shared secret (WSP's version)

signature (REQUIRED) WSP's signature on the shared secret

profile (REQUIRED) User's encrypted profile (WSP's version)

profileSig (REQUIRED) WSP's signature on the encrypted profile

cancelCode (REQUIRED) Cancellation code (see Section 4.12.1)