# On the Wire SQRL

## Secure Quick Reliable Login

This fourth and final SQRL system document assumes that its reader has read and understood the concepts described in the previous three "SQRL Explained", "SQRL Operation Details" and "SQRL Cryptography" documents. This document describes SQRL's communications "on the wire" encoding for the exchange of all messaging.

Documentation Download Links:

| | |
|---|---|
| SQRL Explained | https://www.grc.com/sqrl/sqrl_explained.pdf |
| SQRL Operating Details | https://www.grc.com/sqrl/sqrl_operating_details.pdf |
| SQRL Cryptography | https://www.grc.com/sqrl/sqrl_cryptography.pdf |
| SQRL On The Wire | https://www.grc.com/sqrl/sqrl_on_the_wire.pdf |

## Contents

## List of Figures

## SQRL On the Wire

SQRL clients issue a series of one or more (usually more) standard HTTP POST queries over TLS-secured TCP connections to a remote SQRL server. The SQRL server might be integrated into the remote website's web server or it might be a separate free-standing SQRL server presenting the SSP (SQRL Service Provider) or other API to the website's web server.

Since the client relies upon the communication mechanisms provided by its underlying host operating system, a valid TLS certificate and standard HTTPS connection must be available from the SQRL server. SQRL clients perform authentication and maintenance tasks by issuing one or more commands to the remote server, where each command is carried within an individual HTTPS POST query.

The starting point for any SQRL authentication is a SQRL URL containing an embedded nonce "nut". As previously discussed, SQRL servers issue these "nuts" in an unpredictable sequence and track each nut issued so that they can only be used once. The nuts serve the dual roles of providing replay protection and forcing the client to sign unique, nut-containing envelopes.

Each query command issued by a SQRL client receives a reply from the server containing:

- A version specifier indicating the version of the SQRL protocol supported by the server.
- A TIF (Transaction Information Flags) result indicating the success or failure of the client's command and additional informational status flags.
- A freshly issued "nut" to be returned within the client's next query (if the client queries again).
- A "qry" query URL to be used for the next client query (if the client queries again).
- One or more additional "named values" as required by the context of the command.

With each query, the SQRL client returns the SQRL server's previously supplied data without modification. For the first transaction originated by a URL, the exact URL is returned. For subsequent queries, the server's previous reply is returned.

Upon receipt of every query, the server must examine and verify that the returned server data has not been changed. Since every such return includes a "nut" from the server, and since each return envelope is signed by the client, the server's verification of its own unchanged data, of the nut's validity and of the client's signature, creates an interlocked cryptographically strong chain of client commands and server replies.

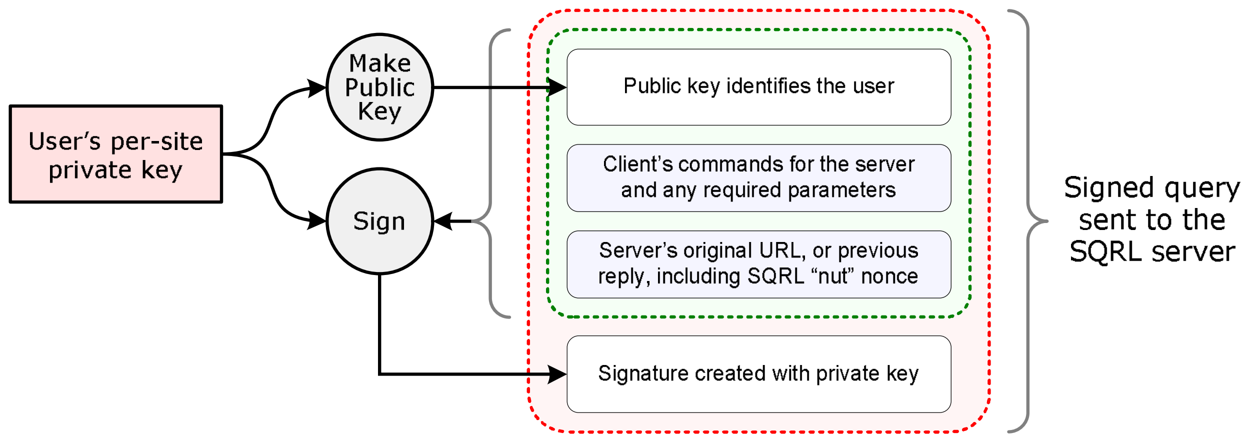

User's per-site private key
Make Public Key
Sign
Public key identifies the user
Client's commands for the server and any required parameters
Server's original URL, or previous reply, including SQRL "nut" nonce
Signature created with private key
Signed query sent to the SQRL server

*Figure 1: Client's POST data assembly*

Upon receiving a SQRL client query, SQRL servers begin by:

- Verify that the returned "server" data exactly matches the original URL or the last reply sent to the SQRL client.
- Verify that the nut returned is valid and unused, then discard the nut so it cannot be reused.
- Verify that every signature returned by the client is valid.

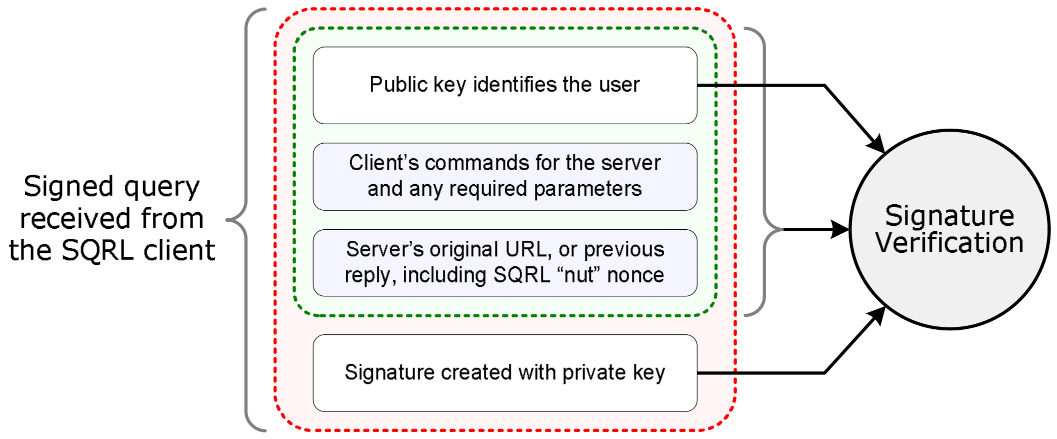

Signed query received from the SQRL client
Public key identifies the user
Client's commands for the server and any required parameters
Server's original URL, or previous reply, including SQRL "nut" nonce
Signature created with private key
Signature Verification

*Figure 2: Server's POST data verification*

## GRC's Demo Diagnostics

GRC maintains a SQRL "diagnostic" service which allows developers to capture and display live SQRL transactions: https://www.grc.com/sqrl/diag.htm. The page may be visited and used with any SQRL client. After a series of individual transactions with the SQRL client, the diagnostics page shows the textual content of the entire authentication process. Developers seeking additional clarification about SQRL's protocol should remember that this facility is available and avail themselves of the visibility it provides.

## SQRL Data Exchange Format

SQRL's "on the wire" protocol uses the traditional "name=value" HTTP POST data format which is known as "application/x-www-form-urlencoded." Although XML and JSON are currently in vogue, their interpreters are often large and complex due to the expressive power of their formats. They have been subject to many parsing errors resulting in serious security vulnerabilities. In keeping with SQRL's minimalism, and since SQRL has no need for such complexity, SQRL adopts the much simpler, flat, "name=value" data format.

An equals (=) sign joins each name to its value, and multiple "name=value" pairs are concatenated with a joining ampersand (&). Since the data contained within the values may themselves contain the '=' and '&' characters, or in many instances binary data such as keys or signatures, all values are encoded using base64url encoding. Since some base64url encoders pad their output with unnecessary '=' characters, those are removed after encoding and are not required when decoding. This encoding is detailed in IETF RFC #4648: https://tools.ietf.org/html/rfc4648#section-5

Sample POST data from a SQRL client looks like this:

```
client=dmVyPTENCmNtZD1xdWVyeQ0KaWRrPWlnZ2N1X2UtdFdxM3NvZ2FhMmFBRENzeFJaRUQ5b245S
DcxNlRBeVBSMHcNCnBpZGs9RTZRczJnWDdXLVB3aTlZM0tBbWJrdVlqTFNXWEN0S3lCY3ltV2xvSEF1bw0
Kb3B0PWNwc35zdWsNCg&server=c3FybDovL3Nxcmwuc3RldmUuY29tL2NsaS5zcXJsP3g9MSZudXQ9ZTd
3ZTZ3Q3RvU3hsJmNhbj1hSFIwY0hNNkx5OXNiMk5oYkdodmMzUXZaR1Z0Ynk1MFpYTjA&ids=hcXWTPx3
EgP9R_AjtoCIrie_YgZxVD72nd5_pjMOnhUEYmhdjLUYs3jjcJT_GQuzNKXyAwY1ns1R6QJn1YKzCA
```

*Figure 3: Sample SQRL client POST data*

If we add some color and line breaks, which do not occur in the data, it becomes less opaque:

```
client=dmVyPTENCmNtZD1xdWVyeQ0KaWRrPWlnZ2N1X2UtdFdxM3NvZ2FhMmFBRENzeFJaRUQ5b245S
DcxNlRBeVBSMHcNCnBpZGs9RTZRczJnWDdXLVB3aTlZM0tBbWJrdVlqTFNXWEN0S3lCY3ltV2xvSEF1bw0
Kb3B0PWNwc35zdWsNCg
&
server=c3FybDovL3Nxcmwuc3RldmUuY29tL2NsaS5zcXJsP3g9MSZudXQ9ZTd3ZTZ3Q3RvU3hsJmNhbj1h
SFIwY0hNNkx5OXNiMk5oYkdodmMzUXZaR1Z0Ynk1MFpYTjA
&
ids=hcXWTPx3EgP9R_AjtoCIrie_YgZxVD72nd5_pjMOnhUEYmhdjLUYs3jjcJT_GQuzNKXyAwY1ns1R6QJn1
YKzCA
```

*Figure 4: Sample SQRL client POST data breakdown*

As can be easily seen now, the names "client", "server" and "ids" are each followed by an equals (=) sign and joined with an ampersand (&). "client" is the client's data. "server" is the server's previous reply. "ids" is the signature formed from the concatenation of the previous two values signed with the current identity's private key.

## Client-to-Server Query

Every client query must contain, at minimum, the following three name=value pairs:

1. "client": A base64url-encoded list of client parameters. The details of all client parameters is detailed below.
2. "server": The base64url-encoded data returned by the server in response to the previous client query, if any, or the sqrl:// URL if this is the client's initial query.
3. "ids": The base64url-encoded signature of the concatenated client and server values above, in that order. In this instance, the signature's key is the per-site private key corresponding to the per-site public key provided by the client within its parameter list contained in the "client" data.

In the example shown above, the "client" value is:

```
dmVyPTENCmNtZD1xdWVyeQ0KaWRrPWlnZ2N1X2UtdFdxM3NvZ2FhMmFBRENzeFJaRUQ5b245SDcxNlRBeVBS
MHcNCnBpZGs9RTZRczJnWDdXLVB3aTlZM0tBbWJrdVlqTFNXWEN0S3lCY3ltV2xvSEF1bw0Kb3B0PWNwc35zdWs
NCg
```

The "server" value is:

```
c3FybDovL3Nxcmwuc3RldmUuY29tL2NsaS5zcXJsP3g9MSZudXQ9ZTd3ZTZ3Q3RvU3hsJmNhbj1hSFIwY0hNNkx5
OXNiMk5oYkdodmMzUXZaR1Z0Ynk1MFpYTjA
```

Therefore, the concatenated data over which the signature is taken, is:

```
dmVyPTENCmNtZD1xdWVyeQ0KaWRrPWlnZ2N1X2UtdFdxM3NvZ2FhMmFBRENzeFJaRUQ5b245SDcxNlRBeVBS
MHcNCnBpZGs9RTZRczJnWDdXLVB3aTlZM0tBbWJrdVlqTFNXWEN0S3lCY3ltV2xvSEF1bw0Kb3B0PWNwc35zdWs
NCgc3FybDovL3Nxcmwuc3RldmUuY29tL2NsaS5zcXJsP3g9MSZudXQ9ZTd3ZTZ3Q3RvU3hsJmNhbj1hSFIwY0hN
Nkx5OXNiMk5oYkdodmMzUXZaR1Z0Ynk1MFpYTjA
```

Clients may need to append multiple signatures to a single query. All signatures are taken from the same concatenated {client value}{server value} using different keys.

Beyond these three always present and required name=value parameters, several optional name=value datums may be present, as needed for the command or context.

## Client Parameters

The query's “client” value parameter consists of a base64url encoded list of name=value pairs, one per line, with each line (including the last line) terminated by a CRLF character pair (0x0D and 0x0A). Before it has been base64url encoded a sample client parameter list might look like this:

```
ver=1
cmd=query
idk=QQ5kaUUDw6Hd7jDJkPCFFEizlQ92SpVU_YMeheJrY_c
pidk=I42_ousudqpLwpfOd7pCT3WJPNTBSVlOdP9w-6qF8kg
opt=cps~suk
```

A more extensive client list might look like this:

```
ver=1
cmd=ident
idk=QQ5kaUUDw6Hd7jDJkPCFFEizlQ92SpVU_YMeheJrY_c
pidk=iCoWeOOqtWZENJGlfQ9IImMC0_1kU-AtlzOngAWANmI
ins=vbmaT9oXjNNDp6ti-TxCE6xIlO36q4EyvjZjhrOmlQY
pins=a45mOg3N19D8p_ZVu_j8lACQlHEbTJFy2dQjUd6xvM0
suk=j-NEB3cbsYiW5YlLqi_Lva_N0QVVPMrIU2zny_ickCw
vuk=q7udgqKKuY0ggdlxC0sdOiv5ySOtWh1ubtCZSXafOpQ
opt=cps~suk
```

The client assembles a list of the following name=value pairs to send data to the web server, to specify the command action it is requesting from the server, and to provide any cryptographic keying material required to authorize the requested actions and/or authenticate its user's identity:

- ver = 1

  As with the server's supported version set declaration, the client MUST declare the set of protocol versions it understands, supports, and is willing to use. Moreover, the version specification MUST be the first name=value pair appearing in the client's argument list. The protocol version specification consists of an unordered comma-separated list of one or more whole integer version numbers (1, 2, 5-9 ...) and/or version ranges. A range is specified by a hyphen (-) joining two integers and specifying all ranges from the first integer to the second integer inclusive. The server and client both declare their supported version sets and use the highest version available from the overlap of their sets.

- cmd = { <u>one</u> of the tokens below }

  The value of the “cmd” parameter specifies the action the client is requesting from the web server. It consists of one, and only one, of the following tokens:

**query** All SQRL transactions *must* begin with one or more **query** commands.

The **query** command allows the SQRL client to determine whether and how its user is known to the SQRL server, whether SQRL's use is currently enabled for the site, and whether the user's query IP matches the IP which was associated with the original SQRL URL. (See the TIF parameter flag bits below for more detail.)

SQRL queries can provide, at most, identity assertions for the user's current and one previous identity. If more than one previous identity key is known to the client (GRC's reference client retains up to four and this should be a minimum) successive **query** commands are used successively, presenting the most recent previous identity first followed by successively older previous identities. This allows the client to determine which, if any, of the identity keys carried by the client are known to the server.

NO modification of any of the user’s server-side account data shall occur as a result of a **query** command.

**ident** An **ident** command will usually follow one or more **query** commands.

Whereas the **query** command allows the client to obtain information from the server, the **ident** command requests the web server to accept the user's identity assertion as it is provided by this signed query. When a SQRL client is used to authenticate its users’ identity to a website there is usually no user-interface action from the client after a successful **ident** command. The client's dialogs close and the website will update to show that the user has been authenticated.

**disable** A **disable** command instructs the web server to immediately disable the SQRL system's authentication privilege for this domain. This might be requested if the user had reason to believe that their current SQRL identity key had been compromised. It is primarily intended to be used as a short-term emergency stop-gap measure to protect important accounts until a new identity key can be created and set into the server. The disable and enable operations are not symmetrical: An identified user may request to have their authentication disabled, but subsequently re-enabling authentication requires the identity's privileged unlock (RescueCode) credentials, which no attacker could obtain from the client because identity unlock credentials are never stored in the client.

**enable** An **enable** command reverses the **disable** command by re-enabling SQRL system identity authentication for the user's account. Unlike **disable**, however, **enable** requires the additional authorization provided by the account's current unlock request signature (URS). The server will always return the user's stored server unlock key (SUK) whenever it is replying to a disabled account (TIF bit 0x08) so that the client will be able to generate the unlock request signature (URS) to authorize this privileged operation.

**remove** This **remove** command instructs the web server to immediately remove all trace of this SQRL identity from the server. For example, this process would allow an account to be disassociated from one SQRL identity and subsequently re-associated with another. This differs from the user's identity being rekeyed, since **remove** allows a change to an unassociated identity. As with the *enable* command above, it must be authorized by the inclusion of the user's SQRL identity's unlock request signature (URS). The client can add the “suk” option flag to its query to request the server to return its stored SUK key so that the client may generate the URS signature.

Since the associated account might not have an alternative (non-SQRL) method of subsequently authenticating a user (e.g. no other login means) *the web server should leave the current account logged-in* so that a replacement SQRL identity can be associated, and it should also advise the user of the need to immediately add a new SQRL identity if they wish to retain SQRL-based access to their account.

- opt = option1[~option2[~option3]...]

The value of the "opt" option parameter provides an unordered list of transaction options which the client asserts to the web server with each query. Individual options, as detailed below, may have an advisory or an explicit behavior-modifying role, which may also depend upon the command being issued by the client.

The value of the "opt" parameter consists of one or more tilde-separated assertions:

**noiptest** By default, SQRL servers fail any incoming SQRL query whose originating source IP address does not match the IP which requested the initial SQRL URL. This detects and prevents one of the easiest to exploit attacks on SQRL. However, some cross-device uses of SQRL, where the SQRL client is expected to have a different source IP than the browser that initially requested the SQRL URL, such as with a mobile client using cellular bandwidth, will need to override and disable this default server same-IP verification. The presence of this **noiptest** option instructs the server to ignore any IP mismatch and to proceed to process the client's query even if the IPs do not match.

If the **noiptest** option is not present and the IPs do not match, the server will reply with a TIF value of exactly 0x40 to indicate that the command failed. The 0x04 "IPs matched" bit will also be off, as is appropriate, but no other TIF bits have meaning in this case.

If the **noiptest** option is present, the server will proceed with the client's query, and the 0x04 "IPs matched" TIF bit will reflect whether the IPs did or did not match.

**sqrlonly** When present, this option requests the web server to set a flag on this user's account to disable any alternative non-SQRL authentication capability, such as traditional username and password authentication.

Server support for this feature is optional but strongly encouraged. Servers that support this option MUST retain and honor the most recently received state of this option as presented during the most recent successful non-query command.

Users who have become confident of their use of SQRL may ask their client to include this optional request. The web server should only assume this intention if the option is present in a valid non-query command. Its absence from any valid non-query command should immediately reset the flag and the prohibition in the web server. The web server may, at its option, notice when any change has occurred and explicitly ask the user to affirm their changed intention.

SQRL servers should ignore any residual effects of those options after a user's account has had SQRL removed.

**hardlock** When present, this option requests the web server to set a flag on this user's account to disable any alternative out of band account recovery measures for this user's web account such as "I forgot my password" eMail or "what was the name of your first pet?" non-SQRL identity recovery.

Server support for this feature is optional but strongly encouraged. Servers that support this option MUST retain and honor the most recently received state of this option as presented during the most recent successful non-query command.

Users who have become confident of their use of SQRL may ask their client to include this optional request. The web server should only assume this intention if the option is present in a valid non-query command. Its absence from any valid non-query command should immediately reset the flag and the prohibition in the web server. The web server may, at its option, notice when any change has occurred and explicitly ask the user to affirm their changed intention.

SQRL servers should ignore any residual effects of those options after a user's account has had SQRL removed.

**cps**

**cps** is the abbreviation for Client Provided Session. The presence of this flag informs the server that the client has established a secure and private means of returning a server-supplied logged-in session URL to the web browser after authentication has succeeded.

When the **cps** option is provided by the client to a successful **ident** query, the server will return a 'url=' parameter which the SQRL client then forwards to the web browser. When the server returns the 'url=' parameter to the client it must abandon the pending login of the browser session associated with the SQRL query. The use of this **cps** process robustly thwarts website spoofing, man-in-the-middle attacks, and abuse of SQRL authentication by returning the authenticated URL directly to the user's browser via the SQRL client.

In the non-cps case where this flag is NOT present the web server will arrange to refresh the web browser's page or redirect the web browser to an authenticated logged-on page.

**suk**

**suk** is the abbreviation for Server Unlock Key. The presence of this flag instructs the SQRL server to return the stored server unlock key (SUK) associated with whichever identity matches the identity supplied by the SQRL client. The SQRL specification requires the SQRL server to automatically return the account's matching SUK whenever it anticipates that the client is likely to require it, such as when the server identifies the client with a previous identity key, or when the account is disabled. However, there are instances where the client may know that it is going to need the stored SUK from the server, such as when it wishes to remove a non-disabled account. The client could first disable the account to induce the server to return the SUK, but it's simpler for the client to request the SUK from the server whenever it wants it. It's also conceivable that future extensions of this specification will incorporate other instances where the server's stored SUK is required for RescueCode based authentication.

- btn = ‘0’, ‘1’ or ‘2’ - submit trigger from an “ask” prompt

The client must include a **btn** parameter in any server query generated by a reply to the server's **ask** prompting (**ask** is described below). The text value of the **btn** parameter will be the character ‘0’, ‘1’ or ‘2’ if the dialog's “OK” or the first or second optional buttons, respectively, were used to submit the *ask* query.

## Client Keys

- idk = IDentity Key

  This is the user's SQRL ID which uniquely identifies them to the site. It is the elliptic curve public key derived from the user's Identity Master Key (IMK) by the HMAC hash of the site's effective domain name. The binary key is base64url encoded with trailing equals sign padding removed.

- pidk = Previous IDentity Key

  When a user has rekeyed their identity to change their master key, all websites still holding the previous identity key need to be updated to the current identity key. So, SQRL S4 identity storage retains up to four Previous Identity Unlock Keys (PIUK) so that the client may synthesize both this PIDK along with the Unlock Request Signature (URS) to allow websites to replace their obsolete SQRL identity key data with the new data. During one or more **query** commands, the client will present the server with the user's current IDK, a previous identity key, and its matching previous identity signature (PIDS). In this manner the SQRL client is able to search for any previous identity the server may have and to update the server to the current identity during a subsequent **ident** command.

- suk = Server Unlock Key

  This SUK key is included in every client ident query when the immediately previous server reply did not have the 0x01 bit of its TIF flags set. The lack of TIF bit 0x01 indicates that the server does not recognize the client by its current identity key and signature. The server might not recognize the client's identity at all (neither TIF 0x01 or 0x02 bits are set) or it might recognize the client by its previous identity (TIF bit 0x02 set). In either case, the client must assert its *current* identity by providing the server with both this SUK and a corresponding VUK (see below). As required by the identity lock protocol, the server unlock key is a DHKA (Diffie-Hellman Key Agreement) public key generated by the client and then stored by the server and returned to the client with every reply.

- vuk = Verify Unlock Key

  This VUK key is generated with and always accompanies the SUK key described above. As required by the identity lock protocol, the verify unlock key is a DHKA (Diffie-Hellman Key Agreement) public key generated by the client, stored by the server, and used by the web server to verify the unlock request signature (URS) provided by the client whenever it wishes to authorize any operation requiring the identity's RescueCode. The RescueCode, in turn, allows the client to transiently decrypt the Identity Unlock Key (IUK) which is required for the DHKA function with the client's provided SUK to synthesize the unlock request signature.

## Client Secrets

- ins = INdex Secret

  When the immediately previous server reply contained a Secret Index (sin=value) name & value parameter, the output of the user's identity key forming HMAC256 is passed through the EnHash function and used to key to a secondary HMAC256 which is used to hash the server's provided *sin* literal value without modification. The secondary HMAC's 256-bit output is base64url encoded and returned to the server as the value for this INS parameter. This allows servers to obtain user-tied secrets, typically for decryption, which they do not need to store.

- pins = Previous INdex Secret

  Whenever the INS parameter, above, is being returned to the server, and if the user's identity also contains at least one previous identity key (PIDK), that previous identity key is used, as above, to also produce and include the matching PINS parameter and value. When more than one previous identity key exists, and while the server continues indicating that it does not identity the user by either of the two presented identity keys, the client will successively iterate through each of the previous identity keys, producing and returning synchronized PIDK and PINS name=value parameters.

## Client Signatures

All client queries sent to the web server are signed using private keys matching one or more of the public keys included in the *"client="* name=value list. The data signed are the two base64url encoded values of the *"client="* and *"server="* parameters with the *"server="* value concatenated to the end of the *"client="* value as described on page 4 above. Each of the signatures below, generated as a binary token, must be base64url encoded before being appended to their respective "name=" parameter.

- ids = IDentity Signature

  This is the signature used to authenticate the contents of the query block sent to the web server. It serves to prove, to the server, that the client "owns" the identity expressed as their IDK by signing the client/server values block with its matching private key. The SQRL client synthesizes the site-specific private key, uses that to sign the concatenated values of the previously mentioned client and server values, base64url encodes the resulting signature, and sends the signature to the web server as the value of this IDS parameter. The web server verifies the signature using the accompanying IDK which must also match the value stored in the user's SQRL account association.

- pids = Previous IDentity Signature

  As with the PIDK (previous identity key), one or more of these are sent (in successive queries if more than one until recognized) to the web server when a SQRL client has rekeyed its identity and thus changed its Identity Unlock Key (IUK). The client retains up to four previous identity unlock keys (PIUKs) to enable it to generate the PIDK and PIDS values. These will both be necessary for the web server to identify and authenticate a user by one of their *previous*, not-yet-updated, identities.

- urs = Unlock Request Signature

  The unlock request signature provides proof to the web server that the SQRL client is in possession of the identity unlock key (IUK) for which the server provided its stored server unlock key (SUK).

SQRL clients must provide this URS signature in three instances: When the client's identity has been rekeyed so that the server's stored identity can be updated and when the client is requesting to either enable a currently disabled account or remove the SQRL authentication information entirely.

If the SQRL client does not currently have the decrypted Identity Unlock Key available it will not, and cannot, supply the matching URS value and it may prompt the user to provide the identity's secret RescueCode if needed. As noted it SQRL previous documents, SQRL's most security-critical (and uncommon) operations are protected by this additional signing requirement. The web server supplies its stored server unlock key (SUK) with any replies that might require the SQRL client to provide a URS signature. The web server uses its stored verify unlock key (VUK) to verify the client's signature. The use of the **enable** or **remove** commands to re-enable or remove a disabled identity, or the replacement of a previous identity key with the current identity key, must be accompanied by a valid unlock request signature.

## Server-to-Client Reply

A remote web server provides data to a SQRL client through two different channels:

1. A unique SQRL URL containing the required “nut=”, and an optional “x=” domain extension specifier.
2. The HTTP response body to SQRL client queries. Any of the parameters shown below may be used, as required, in the server's replies to client queries.

As we have previously seen, parameters are named with a simple name=value syntax. When appearing in a URL query tail, the name=value pairs must be URL-safe and ampersand-separated following standard HTTP GET query syntax. When returned by a web server in response to a client's query, the name=value pairs occupy the body of the reply, appearing one per line with each line (including the last line) terminated by a CRLF character pair.

Note: To increase the clarity of the descriptions below, the equals signs have been set off with spaces, but spaces are never used in the SQRL protocol:

### Server Parameters

- ver = 1[,n],[n-m]

  The server MUST indicate the set of protocol versions it understands, supports, and is willing to use. Moreover, the version specification MUST be the first name=value pair appearing in the server's argument list. The protocol version specification consists of an unordered comma-separated list of one or more integer version numbers and/or version ranges. The server and client both declare their supported version sets and use the highest version available from the overlap of their sets.

- nut = base64url encoded opaque token

  The server's “nut” value has been discussed extensively previously. It is a never-repeating opaque cryptographically strong single-use nonce which may, at the server's discretion, contain reversibly encrypted data used to associate and maintain state. A unique nut MUST be included with every response to guarantee uniqueness and prevent reuse/replay and it MUST be removed from a valid nuts list after it has been returned and validated by the next client query, if any. As with all of SQRL's use of base64url encoding, any trailing equals signs used for padding must be removed.

- tif = hexadecimal integer

The **Transaction Information Flags (TIF)** is a single *hexadecimal* encoded integer which MUST be included in every server response. It is generated by the web server to convey a variety of user identity, command and connection status information for the client query it is in response to. The individual bits are shown using the familiar "0x" hex prefix, but the TIF's value does not need or use the "0x" prefix. The individual bits have the following values and meanings:

| | |
|---|---|
| 0x01 | **(Current) ID match:** When set, this bit indicates that the web server has found an identity association for the user based upon the default (current) identity credentials supplied by the client: the IDentity Key (IDK) and verified by the IDentity Signature (ids). |
| 0x02 | **Previous ID match:** When set, this bit indicates that the web server has found an identity association for the user based upon the *previous* identity credentials supplied by the client in the previous IDentity Key (PIDK) and the previous IDentity Signature (PIDS).<br><br>If neither the 0x01 (IDK) or 0x02 (PIDK) status bits are set, and if the client has earlier previous identity credentials, it should successively try each one, in turn, hoping to find a match. |
| 0x04 | **IPs matched:** When set, this bit indicates that the IP address of the entity which requested the initial logon web page containing the SQRL link URL (and probably encoded into the SQRL link URL's "nut") is the same IP address from which the SQRL client's query was received for this query & reply. Note that the server MUST retain the IP associated with the original SQRL URL, not any subsequent query, and compare each query IP address against that original IP address. |
| 0x08 | **SQRL disabled:** When set, this bit indicates that SQRL authentication for this identity has previously been disabled. While this bit is set, the **ident** command and any attempt at authentication will fail. This bit can only be reset, and the identity re-enabled for authentication use, by the client issuing an **enable** command signed by the unlock request signature (URS) for the identity known to the server. Since the URS signature requires the presence of the identity's RescueCode, only SQRL's strongest identity authentication is permitted to re-enable a disabled identity. |
| 0x10 | **Function(s) not supported:** This bit indicates that the client requested one or more SQRL functions (through command verbs) that the server does not currently support. The client will likely need to advise its user that whatever they were trying to do is not possible at the target website. The SQRL server will fail this query, thus also setting the 0x40 "Command Failed" bit. |
| 0x20 | **Transient error:** The server replies with this bit set to indicate that the client's signature(s) are correct, but that something about the client's query prevented the command from completing. This is the server's way of instructing the client to retry and reissue the immediately previous command using the fresh "nut=" crypto material and "qry=" url the server will have also just returned in its reply.<br><br>The typical cause of this error is almost certainly a static, expired, or previously used "nut" which the server cannot authenticate. Therefore, reissuing the previous command under the newly received server parameters may succeed. *See the "Transient Error Details" section of the TIF Bits and Client Server Interaction Details discussion on page 17 for additional details.* The 0x40 "Command failed" bit (shown next) will also be set since the client's command will not have been processed. |

0x40 **Command failed:** When set, this bit indicates that the web server has encountered a problem processing the client's query. In any such case, no change will be made to the user's account status. All SQRL server-side actions are atomic. This means that either everything succeeds or nothing is changed. This is important since some commands might produce multiple changes, such as identity updates and account status changes.

If this bit is set without the 80h bit set (see below) the trouble was not with the client's provided data, protocol, etc. but with some other aspect of completing the client's request – such as none of the client's provided identity keys being recognized when a command is issued (disable, enable, remove) that requires a known identity. With the exception of the following "Client failure" status bit, the SQRL semantics do not enumerate all possible reasons for the command's failure. If necessary, the web server could use its "ask" feature to explain the problem to the client's user.

If this bit is set with both of the ID match bits (0x01 & 0x02) reset, the trouble may be that the server's "nut" is invalid or has become stale. SQRL clients should retry their query using the nut provided in the failed command reply.

0x80 **Client failure:** This bit is set by the server when some aspect of the client's submitted query - other than expired but otherwise valid transaction state information - was incorrect and prevented the server from understanding and/or completing the requested action. This could be the result of a communications error, a mistake in the client's SQRL protocol, a signature that doesn't verify, or required signatures for the requested actions which are not present. And more specifically, this is NOT an error that the server knows would likely be fixed by having the client silently reissue its previous command using the fresh "nut" it will have also received from the server, though that should still be the first recourse for the client. Since any such client failure will also result in a failure of the command, the 40h bit will also be set.

0x100 **Bad ID Association:** This bit is set by the server when a SQRL identity which may be associated with the query nut does not match the SQRL ID used to submit the query. If the server is maintaining session state, such as a logged on session, it may generate SQRL query nuts associated with that logged-on session's SQRL identity. If it then receives a SQRL query using that nut, but issued with a different SQRL identity, it should fail the command (setting both the 0x40 and 0x80 bits) and also return this 0x100 error bit so that the client may inform its user that the wrong SQRL identity was used with a nut that was already associated with a different identity.

0x200 **Identity superseded:** This bit is set by the server when the client has presented a current identity (IDK) known to the server as having been superseded. If this occurs during a **query** command the command succeeds without error. SQRL clients should terminate their authentication and explain the failure to their user. If this occurs during any non-query command, the command fails, returning the 0x40 command failed flag in addition to the 0x200 flag. SQRL servers maintain a durable list of all valid previous SQRL identities (PIDKs) they have encountered in order to catch and prevent the inadvertent use of a previous identity. This can occur when a user rekeys their identity on one client and fails to share that updated identity among their other clients. (For additional information, see "Handling Superseded Identities" on page 17, below.)

Note that the number of characters in the *"tif="* value may change depending upon the number of characters required to represent the most significant "set" bit flag within the value. Also, additional bit flags defined in future versions of the protocol could be expected to expand the value beyond the ten bits defined by the version 1 release. Therefore, neither the client nor the server should make any restrictive assumptions about the length of the *"tif's"* value.

Also note that all SQRL clients MUST immediately terminate any connection and abort any authentication operation with any SQRL server which includes TIF bits not defined by the specified and agreed upon SQRL protocol version. Protocol extensions within SQRL v1.0 are fully supported with additional name=value pairs in the query and response payload. But TIF bits may not be used for such purpose.

- qry = /query-path

  The SQRL client initially makes contact with the remote web server by issuing the query contained in the SQRL link URL. But subsequent interactions may be made to different web server objects at the same domain and port as specified by the initial SQRL URL. The "qry" parameter is *required* in every reply. It instructs the client what server object to query in its next query, if any. To mitigate the potential for tampering, this qry parameter *only* supplies the full path from the root ( / ) and the object, *not* the scheme, domain name, or port. The scheme, domain and optional port override may *only* be specified once, in the initial URL, and they cannot subsequently be changed and will always be taken from the initially submitted SQRL URL.

- url = successful authentication browser redirection URL

  This parameter **must** be provided by the server in its response to any command, other than **query**, when the SQRL client includes the "opt=cps" (client provided session) option. The presence of "cps" from the client informs the web server that it MUST NOT authenticate the browser session upon successful authentication. Instead, the SQRL client will use the URL provided by the server to redirect the user's waiting browser, with an HTTP 302 Found redirect, to an authenticated session.

- suk = server unlock key

  As defined by the identity lock protocol, the SUK value is originally generated and provided to the server by the SQRL client when the client is creating a new identity association, or modifying an existing association with an update identity. The server retains this value as part of the client's identity and returns it to the client whenever it might be required by the client.

  This would include when the server identifies the user's previous identity, so that the client may update the server to the user's current identity, and also whenever the user's account is disabled, so that the client may have the option of either re-enabling the user's account or completely removing the user's identity. In these cases, the server's provision of the stored SUK value allows the client to then provide the identity unlock signature to either update the server's stored identity, re-enable a previously disabled account, or entirely remove the account from the server.

  The SUK key must also be returned to the client whenever the client has explicitly requested the SUK by including "suk" in its query "opt=" list.

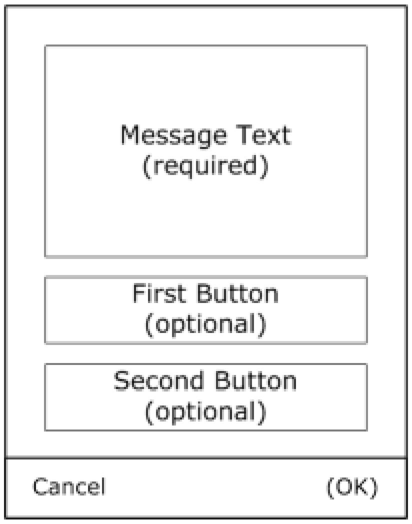
- can = optional cancellation browser redirection URL

  This optional value **may** be provided by the server in its response to any query to provide or replace any previously provided authentication cancel URL. If the user aborts the authentication in a "same-device" client provided session (CPS) authentication, the SQRL client will redirect the user's waiting browser to the URL most recently specified by this URL parameter. This allows the website's server to "change its mind" about where it would like to send the user in the event of a cancellation.

- sin = secret index

The server may include this optional parameter to request a named ("indexed") identity-based, high-entropy, 256-bit "indexed secret" from the client. When this SIN name & value are sent to the client, the immediately following client query, regardless of type (query, ident, etc.) will include the indexed secret (INS) named & value and possibly also a previous-identity indexed secret (PINS) This SIN parameter is called the secret index because this value is hashed as a literal through the client's identity-keyed secondary HMAC256. In this way, servers may obtain any number of "named" (indexed) unique high-entropy secret values from the client, each of which is uniquely based upon the user's current (and possible previous) SQRL identity.

- ask = message text[~button1[;url][~button2[;url]]]



The *ask* parameter may be included in the reply to a client's **query** command when, and only when, TIF bit 0x01 is also set, meaning that the server recognizes the user under their current identity key. It provides a means of implementing a simple but flexible means for a remote server to prompt the user with a free-form question or action confirmation message. This flexible *ask* facility allows the server to obtain client-signed confirmations of the user's intent through the SQRL client-server channel in situations where the web browser-to-server channel cannot offer sufficient security.

In typical use, a website onto which the SQRL user is already logged-on would confirm a high-value action by saying "Please click the SQRL link or scan the SQRL QR code to confirm this transaction." The user would click the provided SQRL button or scan the SQRL QR code to initiate another authentication operation. However, in its first response to the client's initial **query** command, the server would include this 'ask=' parameter to cause the client to display a custom, server-provided confirmation dialog to which the user would respond.

Since the *ask* is contained within the server's response to a previous query, the user and client are not compelled to reply and may choose to cancel the dialog and end their interaction with the server. This would presumably prevent the server from proceeding with whatever action it was seeking confirmation.

Note that the presence of the 'ask=' parameter is **only** valid in response to a client's **query** command and should be ignored by SQRL clients if any SQRL server attempts to add an 'ask=' to any other non-query command.

If the user wishes to submit a reply they may use the dialog's standard "OK" acknowledgement which will be present *only if* no *ask* reply buttons are explicitly specified. Otherwise they may select one of up to two optionally present buttons (or cancel).

To support international characters, UTF-8 encoding is used for all textual content. The message text and optional button texts are individually base64url encoded to that they may contain characters that have semantic meaning to the transport protocol (CR, LF, and '~'). The semicolon (;) is reserved within button text to delimit optional URLs and may not appear on buttons.

The optional *ask* parameters are specified by a sequential list of tilde-separated (~) arguments as shown by the semantics above. If no parameters are supplied, the message text will be displayed and the dialog's standard Cancel/Ok buttons will be used to cancel the transaction or to acknowledge the receipt of the message, respectively. The message text should be worded accordingly.

The message text may be followed by either zero, one, or two button text and optional URL specifications. In clients where display space and device orientation permits, the buttons should be placed side-by-side with the "first" button on the left.

The client must *always* include a "btn=" parameter in its reply so the server will recognize the query as a reply to its *ask*. If the dialog is acknowledged by pressing either of the buttons, the "btn=" parameter's value will be '1' or '2' depending upon which button was chosen, the first or second, respectively. If the dialog is acknowledged by clicking the dialog's "OK" button "btn=0" will be returned.

A button's label text specification may be terminated by a separating tilde (~), a semicolon (;) or the end of the string. If the button text is terminated by a semicolon, the following character begins the specification of a root-anchored URL path which is bound to the button's selection. If the provided path does not begin with a '/' it will be provided by the SQRL client. If the user selects the associated button, in addition to immediately generating a query containing the appropriate "btn=" parameter, the SQRL client will submit the specified URL, using the same scheme, domain and port as the SQRL link, to its host operating system for handling. An example button text string would be: "Click to Launch;/sqrl/extra-confirmation.html".

Since this is test supplied by a potentially untrusted server, all SQRL implementations *must* protect against exploitation of unnecessarily powerful display surfaces. A simple text window should be used rather than a full HTML parser. If a limited display surface cannot be provided, implementations must filter/escape any dangerous characters so that they are displayed only and not treated with any exploitable capability.

## TIF Bits and Client Server Interaction Details

The TIF Bits returned by the server represent the state of the query and the user's account **AFTER** the server has processed the client's query. This is a subtle but important point: Take the case that a client's identity has been rekeyed <u>*twice*</u> since it's last authentication to a website...

The client's initial **query** command will result in the SQRL server returning a TIF with bit 0x01 (current identity) of '0' and 0x02 (previous identity) of '0'. This occurs because the identity that the server has will be the 2nd most recent, not the first most recent.

When the client receives the server's TIF recognizing neither of its proposed identities it's not worried because it knows that it has still-older previous identities. So, the client sends another **query** command to the server, this time presenting its current identity and its next most recent identity as the "previous".

This time the SQRL server recognizes the client by its "previous" identity. So, in response to the client's **query** command it returns a TIF with bit 0x01 set to '0' and 0x02 set to '1'. Also, since the server has recognized the client by its previous identity, the client will need to supply the additional URS (unlock request signature) verification to allow the server to update its account information. So, the SQRL server proactively returns the account's current SUK key so that the client may synthesize the URS signature to unlock the account for updating.

Upon receiving this successful recognition of its 2nd most recent previous identity, the SQRL client has established the identity by which it is known to the remote website. So, it issues an **ident** command containing the same pair of identities – the current and the 2nd most recent previous – which it knows the SQRL server will accept.

Upon receiving an **ident** command, where the client's previous identity is the one it recognizes, and assuming that everything else is in order, all signatures verify, and the client provided a valid URS (unlock request signature), the SQRL server will immediately replace the previous identity with the new current identity.

*And, since the server NOW recognizes the client by its new CURRENT identity, it will reply with a TIF with the 0x01 bit set to '1' and the 0x02 bit reset to '0', thus confirming to the SQRL client that everything has gone as expected and the client's new identity has been accepted and updated.*

If the client had no previous identities, or if it runs out of previous identities without the SQRL server ever acknowledging any of the client's available identities, the client may finally issue an **ident** command to assert its current, not-yet-known, identity to the SQRL server. The server's response to such an **ident** command may – or may not – immediately reflect the association with the newly presented identity. In other words, both 0x01 and 0x02 bits might be reset, since the client would still be unknown to the server at that time. Depending upon the circumstances, the server would probably then prompt the user, through their web browser session, whether they wish to associate their SQRL identity with a new or existing account on the website, and the server would take the appropriate action given the current identity information provided by the client.

To be clear, the 0x01 bit, which indicates that the current identity is known to the server might be delayed until that becomes true, based upon subsequent interaction with the user.

### Transient Error Details

SQRL servers vary in their handling of expired nuts due to implementation differences between servers. Some SQRL servers may encode sufficient information into an expired nut that a valid browser session binding can be decoded and reused. Other implementations may use the nut as a pointer to transient data that is removed with the nut's expiration. In that case any previous browser session binding would be unrecoverable. These differences determine the server's response to its receipt of an expired nut.

In the first case, the server would return a TIF with the "Transient Error" bit 0x20 set and with a newly minted fresh nut that's associated with the browser session for the previous expired nut. The receipt of the 0x20 TIF bit would instruct the client to reissue their previous query using the new nut, to which the client should expect success.

In the second case, the server would also return a TIF with the "Transient Error" bit 0x20 set, and that reply would also offer a newly issued nut. However, since this second-case server cannot reconnect any expired nuts to any previous browser sessions, it will respond to this second query with a new nut and **exactly** the same error having the 0x20 TIF bit set.

SQRL clients are always checking for a pair of identical error messages received in succession. If this should ever occur, the user must be informed that the web page they are logging into has expired and that they should refresh the page and retry their login. This page refresh will reestablish a new browser session and with its associated data structures and will allow the second-case server to succeed any subsequent SQRL client queries.

## Handling Superseded Identities

Due to the independence of individual SQRL clients and SQRL's deliberate eschewing of centralized coordination, an unlikely but possible situation can occur where a user's previously superseded SQRL identity is used for authentication to a website where the user's updated & current SQRL identity has been used and is known.

As we know, websites autonomously update their SQRL identity information during any non-query client command whenever they are presented with an unknown current identity and a previous identity they recognize; the previous identity is replaced by the user's new current identity.

This presents a problem if a SQRL user rekeys their identity on one client but fails to share their new identity among their other devices. If such a user were to attempt to authenticate with a superseded identity to a website that was already aware of their new current identity, it would not recognize their previous (forgotten and discarded) identity and would offer to have them either create a new account or associate their SQRL identity with an existing account. Neither of these outcomes are ideal.

The solution is for websites to maintain a durable list of every valid but superseded previous identity key (PIDK) they encounter, and to always check any current identity key (IDK) against this list before accepting that current identity and returning a TIF with the 0x01 (current ID match) bit set.

In the event that any SQRL client attempts to authenticate using a current identity key (IDK) which is a member of the server's superseded identity keys list, the server must immediately fail that client query by returning a TIF of 0x240. This has the 0x200 "Superseded Identity" bit and the 0x40 "Command Failed" bits set.

Upon receiving this failure code, all SQRL clients must present their users with a dialog explaining that they are using an old and obsolete SQRL identity which has been replaced by a newer rekeyed identity on another of their devices. They should immediately update this and another other SQRL client devices with the most recent identity from that other device.

## Client & Server Parameters on One Page

### Client-to-server query parameters

EVERY client query MUST include the "server", "client" and "ids" parameters. Additional parameters MAY or MUST be included as described by the specification above.

| | |
|---|---|
| **client**= | the base64url encoding of a name=value list of required and optional parameters: |

Required:

| | |
|---|---|
| ver= | the SQRL protocol versions supported by the client – must be the first parameter. |
| cmd= | one command token specifying this query's command. |
| idk= | the ECC public key corresponding to the user's current identity. |

Optional:

| | |
|---|---|
| opt= | a tilde-separated list of command-modifier options. |
| btn= | a single decimal digit indicating the user's reply to a server's "ask" parameter. |
| pidk= | the ECC public key corresponding to (one of) the user's previous identity(s). |
| ins= | the indexed secret produced upon request of the server. |
| pins= | the previous indexed secret produced upon request of the server. |
| suk= | the server unlock key corresponding to the user's current identity. |
| vuk= | the verify unlock key corresponding to the user's current identity. |

| | |
|---|---|
| **server**= | the base64url encoding of the server's initial SQRL URL or its subsequent response. |
| **ids**= | the current identity signature of all preceding non-signature data. |
| **pids**= | the previous identity signature of all preceding non-signature data. |
| **urs**= | the signature used to validate a RescueCode authentication. |

### Server-to-client response parameters

EVERY server response to a SQRL client's query MUST include all of the following name-value parameters:

Required:

| | |
|---|---|
| ver= | the SQRL protocol versions supported by the server – must be the first parameter. |
| nut= | a nonce to guarantee the uniqueness of every server reply. |
| tif= | the status condition code results of the client's query. |
| qry= | the URL /path to be used for the client's next query, if any. |

Optional:

| | |
|---|---|
| url= | the URL /path to be used only upon successful (CPS) authentication. |
| can= | the URL /path used to redirect the browser upon user cancellation. |
| sin= | sent by the server to request the client's matching indexed secrets. |
| suk= | returned when requested or when the client may need to generate a URS signature. |
| ask= | returned in **query** command replies when the server wishes to prompt the user. |
| ???= | anything else: The web server may include any additional name=value pair data it wishes to, beyond those values enumerated above. (For example, a ‘mac=’ name and value might provide a message authentication code to detect any changes in the returned parameters.) If present, any additional parameters will have no protocol meaning whatsoever to the SQRL client and they will be silently ignored. They will, however, be returned by the client in its subsequent query, and signed. A web server may use additional parameters to validate or help it maintain state between its replies and the client’s next query. |