These slides: https://www.grc.com/sqrl-presentation.pdf

Robust Pseudonymous Identity for the Internet

Full Feature Walkthrough

(A Practical Username & Password Replacement)

S Q R L

Secure Quick Reliable Login

A simple, straightforward, open, intellectual property unencumbered, easily explained, provably secure, pseudonymous, 2-party, web domain based, ***authenticated identity*** solution, including complete identity lifecycle management…

For the Internet.

Why SQRL will succeed

Truly Friction-Free Identity

Those who know SQRL believe it will succeed because, unlike any and all other authentication solutions, using it is virtually friction-free (which matters quite a lot) and it provides much greater operational security than any other system. It is free for everyone to use and relies upon no 3rd-party provider.

~~FIDO~~ ~~User & Pass~~ ~~TOTP~~

~~OAuth~~ ~~SMS~~

Why SQRL will succeed

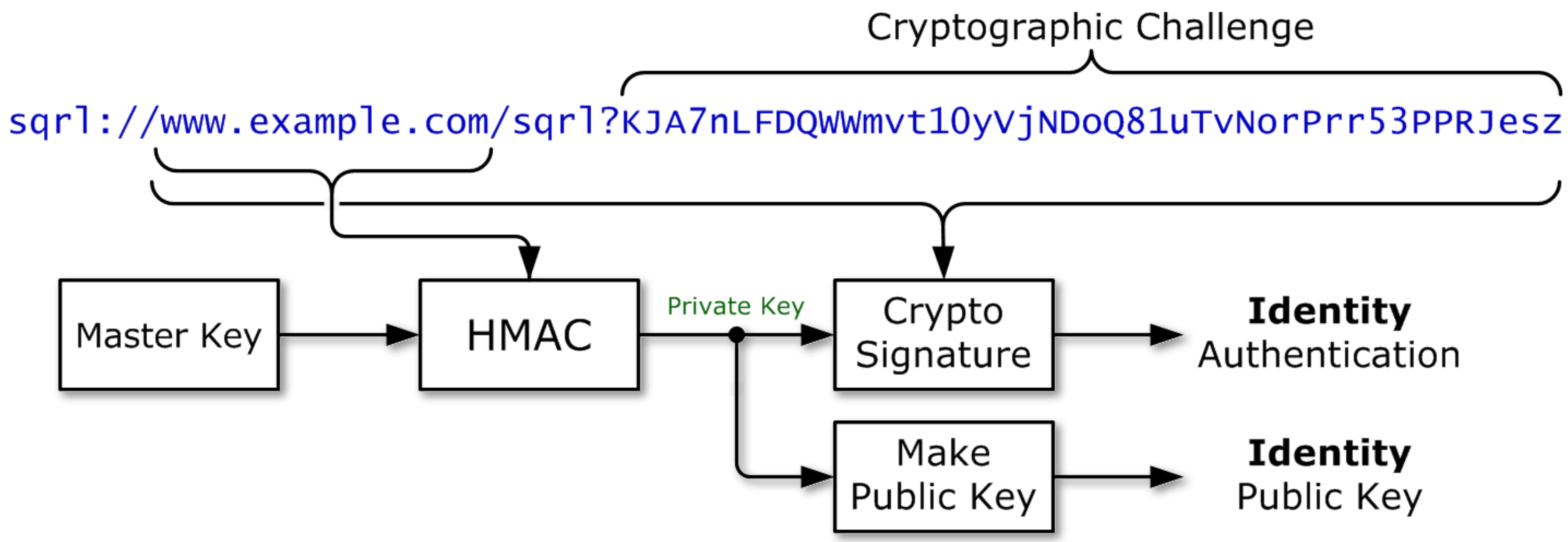
- Supports a single master identity, for everything.
- Incorporates practical identity lifecycle management.
- Secure against website breaches (no secrets to keep).
- Minimal 2-party solution. No central third party to trust.
- Pseudonymous, unlinkable, prevents tracking.
- Simple, straightforward, open, non-proprietary & free.
- Provably secure, understandable & easily auditable.
- Most complexity resides in the client to ease server-side development. Client and server code available and free.
- Plays well with others. Easily coexists with any other identity/authentication solutions for low-friction adoption.

Pseudonymous Identity

- You are just a (very large) number.
- SQRL doesn't know who you are, and neither does any website you visit (unless you choose to tell it).
- For every website you visit, your single lifelong, SQRL identity creates a static, unique, unlinkable, 256-bit "token" which represents your identity at that site.
- After you have logged in traditionally, you "**associate**" your unique SQRL token with your existing account. This tells the site "this is who I am with SQRL, to you."
- From then on, you may use SQRL to identify yourself to login and approve other identity-dependent actions.

Here's how the whole system works…

SQRL in a nutshell



After decryption with the SQRL password, the user’s identity provides the key for an HMAC hash of the website domain to produce an elliptic curve private key which signs every client query, including a unique per-transaction nonce. The ECC private key is transformed into a matching public key, which provides the user’s per-website SQRL identity.

SQRL in a nutshell

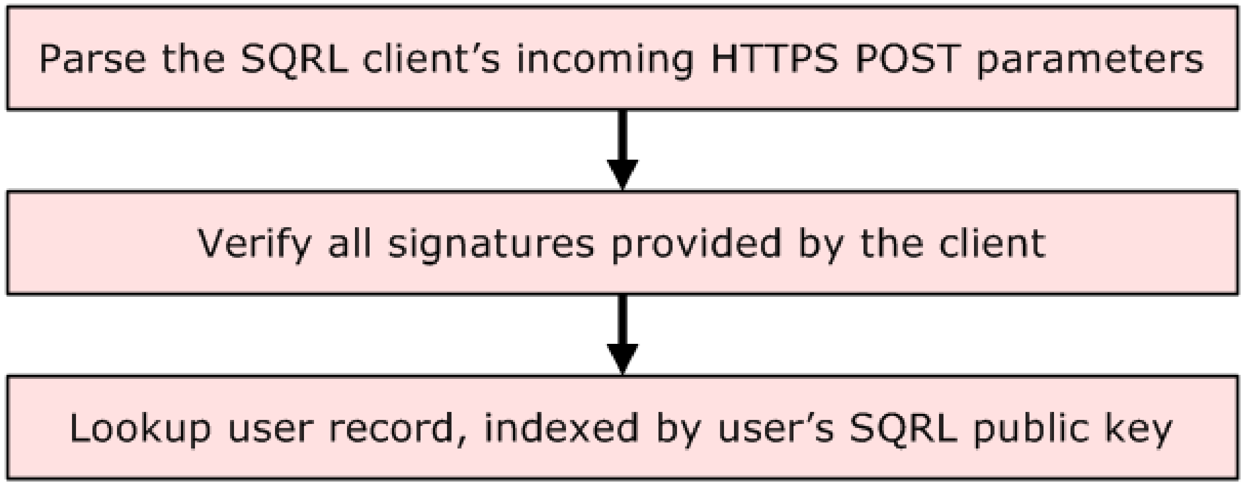
- A user’s master SQRL identity is a randomly derived, static, long-term, high entropy 256-bit token.
- This master identity keys an HMAC-256, which maps a website’s domain name into a per-website elliptic curve private/public key pair.
- The resulting public key identifies the user to the website.
- Users associate this public key with their existing identity at a website and are subsequently identified by this public key. We call this “**creating a SQRL identity association**.”
- Returning users must reassert their identity by signing a server-supplied nonce with their matching private key.

What does this mean?

- User identities are per-site, pseudonymous & unlinkable.
- No per-site data needs to be stored by the SQRL client.
- The use of a unique nonce prevents any replay or reuse.
- No third party intermediary is required for authentication.
- Websites receive a public key that is only useful to them.
- The website's public key can only be used to identify and confirm a visitor's identity. It cannot be used to assert an identity.
- SQRL does not give websites any secrets to keep.

The server's role

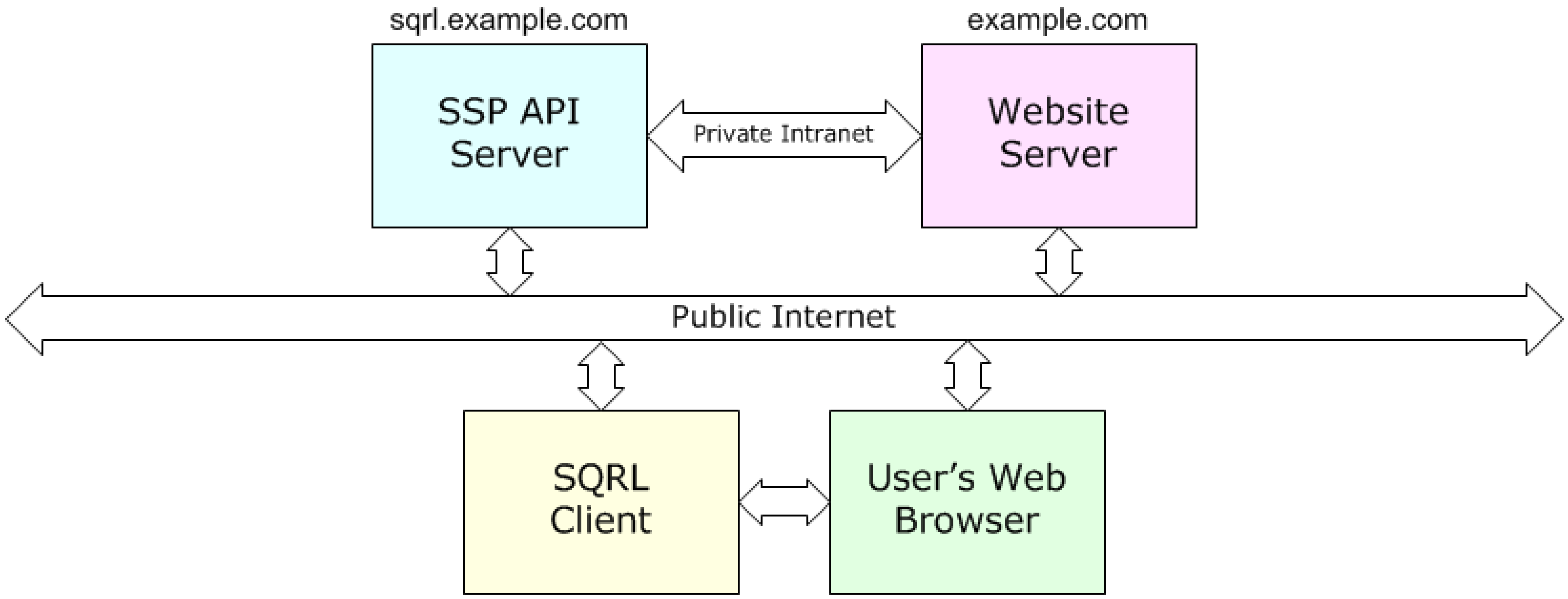
- SQRL's server-side requirements could not be simpler:


Parse the SQRL client's incoming HTTPS POST parameters
Verify all signatures provided by the client
Lookup user record, indexed by user's SQRL public key

- There is, of course, the need for SQRL protocol support and many implementation details. But, **the only server cryptography required is signature verification!**

SQRL’s Service Provider API

- SQRL’s Service Provider API (available in open source “C”) encapsulates **ALL** SQRL protocol details. It provides a simple HTML query/reply protocol to dramatically simplify SQRL’s addition to any existing web server platform:



SQRL's Service Provider API

With the SSPAPI, SQRL authentication is as simple as 1 2 3:

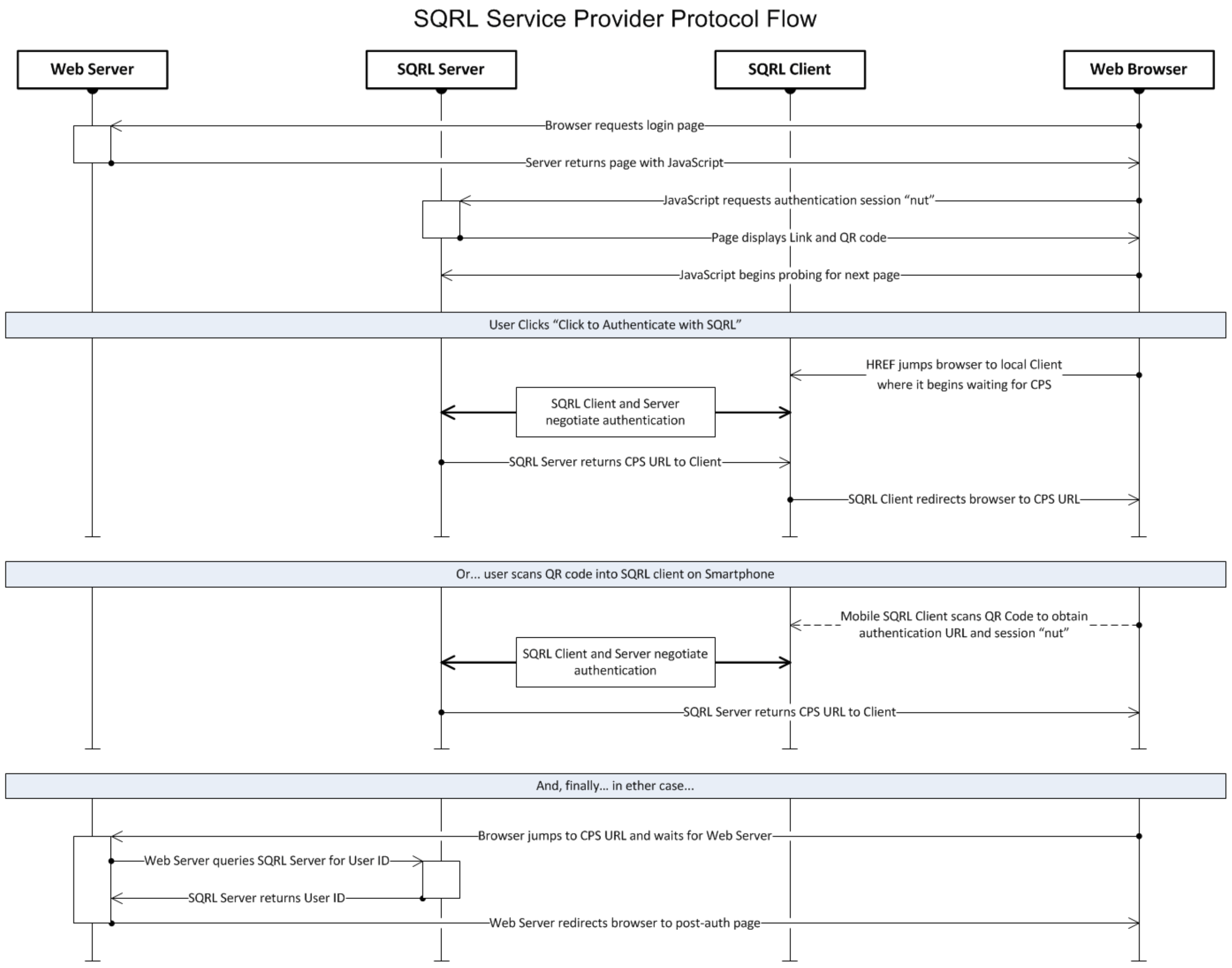
1) Website's SQRL-enabled logon page embeds URLs for a SQRL button and QR code, which the SSP API provides.

2) Upon successful authentication, user's browser jumps to website URL with a token provided by the authentication.

3) Website queries SSP API with token to obtain the user's authenticated SQRL ID and, optionally, website account.

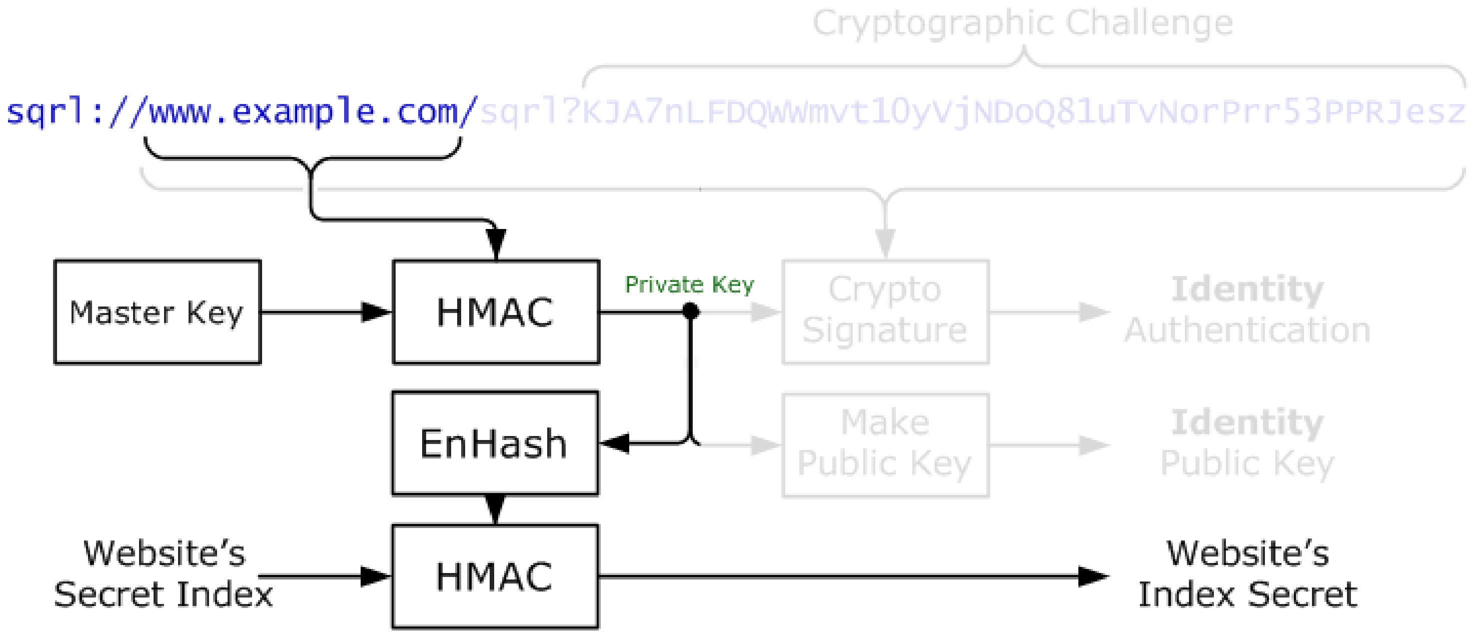
The SSPAPI provides an "association database" to link SQRL users to website accounts. So not even the site's database needs to be modified to accommodate SQRL.

SQRL's Service Provider API


SQRL Service Provider Protocol Flow
Web Server
SQRL Server
SQRL Client
Web Browser
Browser requests login page
Server returns page with JavaScript
JavaScript requests authentication session "nut"
Page displays Link and QR code
JavaScript begins probing for next page
User Clicks "Click to Authenticate with SQRL"
HREF jumps browser to local Client where it begins waiting for CPS
SQRL Client and Server negotiate authentication
SQRL Server returns CPS URL to Client
SQRL Client redirects browser to CPS URL
Or... user scans QR code into SQRL client on Smartphone
Mobile SQRL Client scans QR Code to obtain authentication URL and session "nut"
SQRL Client and Server negotiate authentication
SQRL Server returns CPS URL to Client
And, finally... in ether case...
Browser jumps to CPS URL and waits for Web Server
Web Server queries SQRL Server for User ID
SQRL Server returns User ID
Web Server redirects browser to post-auth page

Secure encryption of user data

- Security conscientious websites would like to securely store user data without risk of post breach decryption.
- SQRL can (optionally) key an HMAC of a server-provided token to return a unique, user and site-specific, static key which can be used for server-side storage encryption.



Desktop & mobile modes

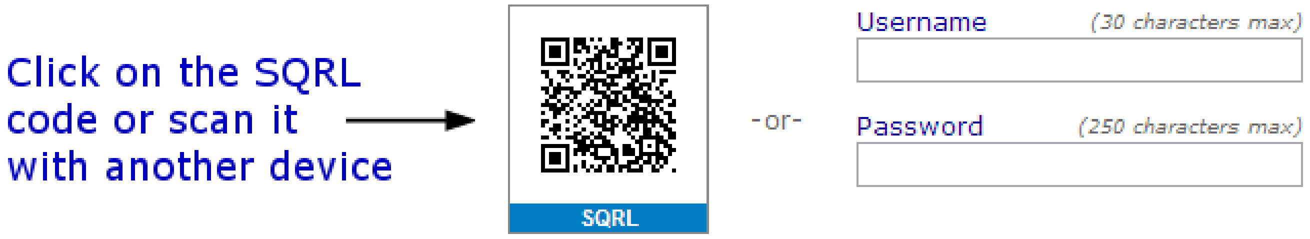
"Same Device" login: (desktop, laptop, or mobile)

- SQRL clients register the "sqrl" URL scheme and respond when a user clicks the S**QR**L code in any local browser.
- The SQRL client receives the sqrl:// URL containing a unique nonce. It converts the URL to https:// and queries the website's SQRL authentication service at that URL.
- Every SQRL client query includes the user's public key identity and is signed by the user's private key for that site.
- The site verifies the signature of all SQRL client queries, performs any requested actions, and returns status to the client.

Desktop & mobile modes

"Cross Device" login: (mobile –to–> desktop, laptop)

- SQRL can be used with camera-equipped devices to securely authenticate its user to websites displaying a S**QR**L code:



Click on the SQRL code or scan it with another device

SQRL

-or-

Username	(30 characters max)
Password	(250 characters max)

- The standard optical QR code encodes the same sqrl:// URL as its clickable link. The SQRL client contacts the website's SQRL authentication service and securely asserts its user's identity. The browser session is transparently logged-in with **no** keyboard interaction.

The devil is in the details

We still need to authenticate the user to their device.

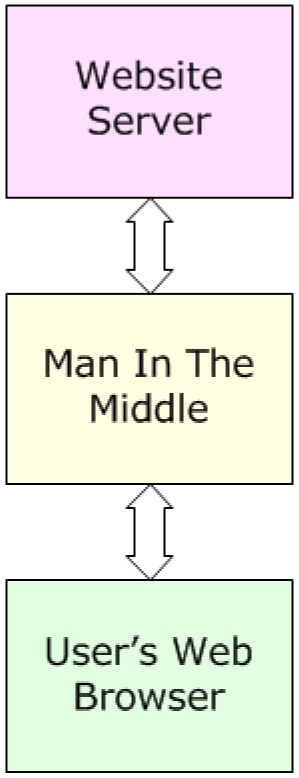
- SQRL becomes a proxy for the user’s identity, able to assert it on their behalf. But this means we must somehow protect against **its** abuse.

- We need simple, per-use, user re-authentication.

- Smartphone biometrics is convenient where available.

- GRC’s SQRL client allows the use of an abbreviated password (“ShortPass”) for quick re-authentication during the same sitting. This eliminates the annoyance of frequent redundant re-entry of a long and secure password during a single session.

The man-in-the-middle threat



- Online authentication systems are inherently vulnerable to various forms of MITM and spoofing vulnerabilities.
- The user believes they are logging into an authentic website, but they are logging into a spoofed website.
- They provide their identity credentials (username and password – even TOTP token in a real time attack – to an imposter, who then logs in as them.

Defeating man-in-the-middle

- SQRL provides protection from all known forms of man-in-the-middle attacks for same-device logon: when the SQRL client resides in the same system as the browser.

- This is the most common authentication mode with a web browser add-on or with a local client.

- SQRL's cross-device (smartphone) authentication offers some protection, but SQRL's extremely robust "Client Provided Session" (CPS) protection leverages the SQRL client and web browser being on the same machine.

Let's examine each of the threats:

Defeating man-in-the-middle

Several MITM attacks are possible.
We will address each:

- An unwitting user visits a malicious website which invites them to login with SQRL. But the SQRL URL they are given is for "Amazon.com", which the malicious site first received from Amazon. If the user authenticates to *that* SQRL URL, the malicious site's session would be logged in as the user.

- Since SQRL URLs are not user-readable SQRL prominently displays the authentication domain to which the user is authenticating. This protects the user from that simple spoofing.

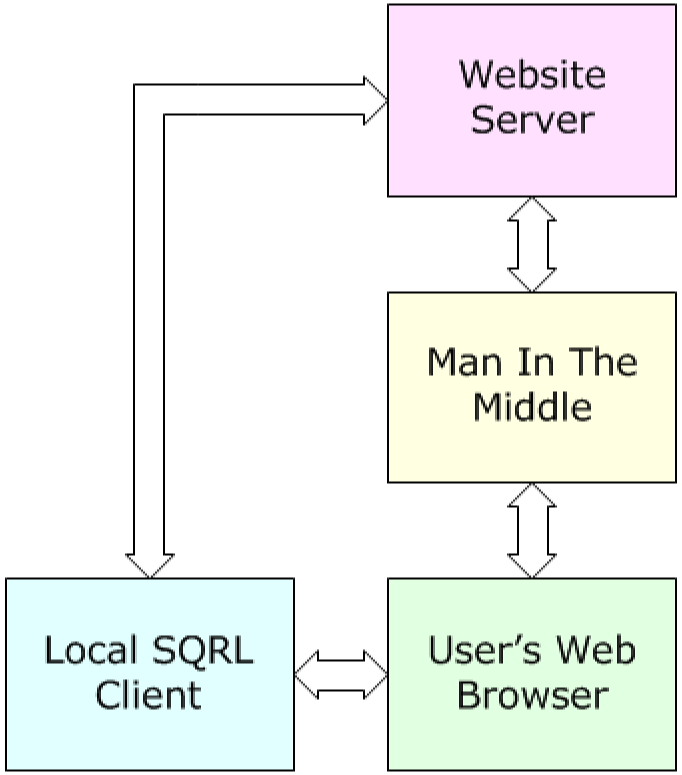
Defeating man-in-the-middle

- SQRL is IP-aware to prevent an attacker at a different IP address from authenticating the user.

- The IP which requested a SQRL code is incorporated into the SQRL session. The SQRL service provider compares this IP with the subsequent SQRL client query IP and fails the authentication (notifying the client) when the two do not match. This robustly prevents an unwitting user from authenticating a MITM located at a different IP.

- But… IP-based MITM mitigation will not detect the case where an attacker can arrange to have the same public IP as the user’s SQRL client. (Behind the same NAT router.) So while this is very useful, it is, at best, a “mitigation.”

CPS: Client Provided Session



- With "Client Provided Session" the user's browser initiates an authentication with the SQRL client by jumping to localhost.
- The client authenticates to the server and receives a valid URL and logged-on session token.
- The client returns the token to the browser by redirecting it to the authentic website, thus bypassing any MITM.

We're getting there… But we still have a few problems to solve:

Creating a practical solution

- **Without any 3rd party, the user has no recourse.**

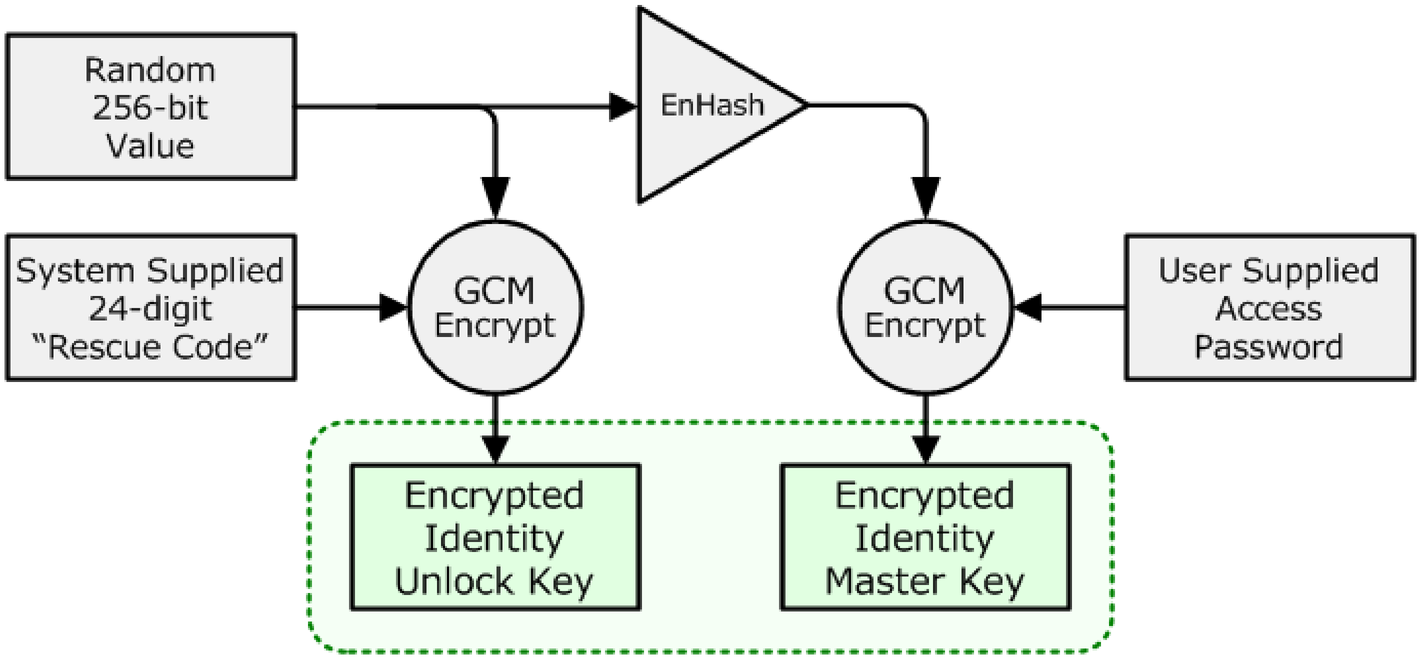
- There is no one they can go to for password recovery. While this presents problems, it also makes SQRL **vastly** more secure, since impersonation, social engineering, and other attacks on password recovery are commonplace. SQRL eliminates them all.

- "I forgot my password!"
- "I just changed my password, but I forgot the new one!"
- The whole point of SQRL is that websites can no longer help. If they could… they could be hacked too.
- "Malware got into my machine and stole my SQRL ID!"

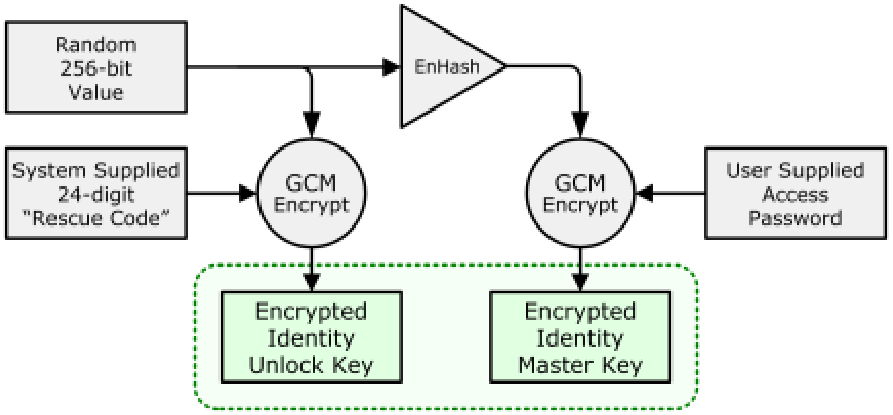
Introducing the "Rescue Code"

- **SQRL needs a secure emergency recovery system.** A "get out of jail free card" that functions on the client side without any 3rd party (because there is no 3rd party).

Identity Synthesis Process (performed once)


Random 256-bit Value
EnHash
System Supplied 24-digit "Rescue Code"
GCM Encrypt
GCM Encrypt
User Supplied Access Password
Encrypted Identity Unlock Key
Encrypted Identity Master Key

Introducing the "Rescue Code"


Random 256-bit Value
EnHash
System Supplied 24-digit "Rescue Code"
GCM Encrypt
GCM Encrypt
User Supplied Access Password
Encrypted Identity Unlock Key
Encrypted Identity Master Key

- SQRL identities carry the "root" identity encrypted by a system supplied, maximum-entropy 24-digit "rescue code" and, also, the derived master identity, encrypted under the user-chosen (lower entropy) access password.

- This allows for the recovery of a forgotten password by decrypting the root identity with the 24-digit rescue code.

- The rescue code is NEVER stored in any client. It always remains offline and inaccessible to client compromise.

The Rescue Code

1484-3606-4253-9577-0233-6070

(sample)

- The rescue code is a maximum-entropy 24-digit key.
- Yes, it is annoying. But it is very rarely (if ever) needed.
- Most users will use SQRL throughout their lives without ever needing their SQRL rescue code… even once.
- It is NEVER stored by any SQRL client. It must be written down or printed, just once, when a SQRL identity is created. It cannot later be recreated.
- In addition to enabling password recovery, the rescue code enables some additional valuable capabilities. . .

SQRL’s Identity Lock

“Malware got into my phone and maybe stole my SQRL ID!”

“Big Brother had my phone and maybe got my SQRL ID!”

“I had to give someone access, now I need a new ID.”

- SQRL’s greatest ***strength*** is that everything flows from a single master ID.
- SQRL’s greatest ***weakness*** is that everything flows from a single master ID.
- SQRL’s “Complete identity lifecycle management” allows secure identity rekeying if, for any reason, its user believes their current identity may, for any reason, no longer be secure or trusted.

SQRL’s Identity Lock

- A unique application of Diffie-Hellman key agreement:
- Given a public key one (PubKey1) & secret key one (SecKey1), and public key two (PubKey2) & secret key two (SecKey2), then:

DHKA(PubKey1, SecKey2) = DHKA(PubKey2, SecKey1)

SQRL’s Identity Lock Construction

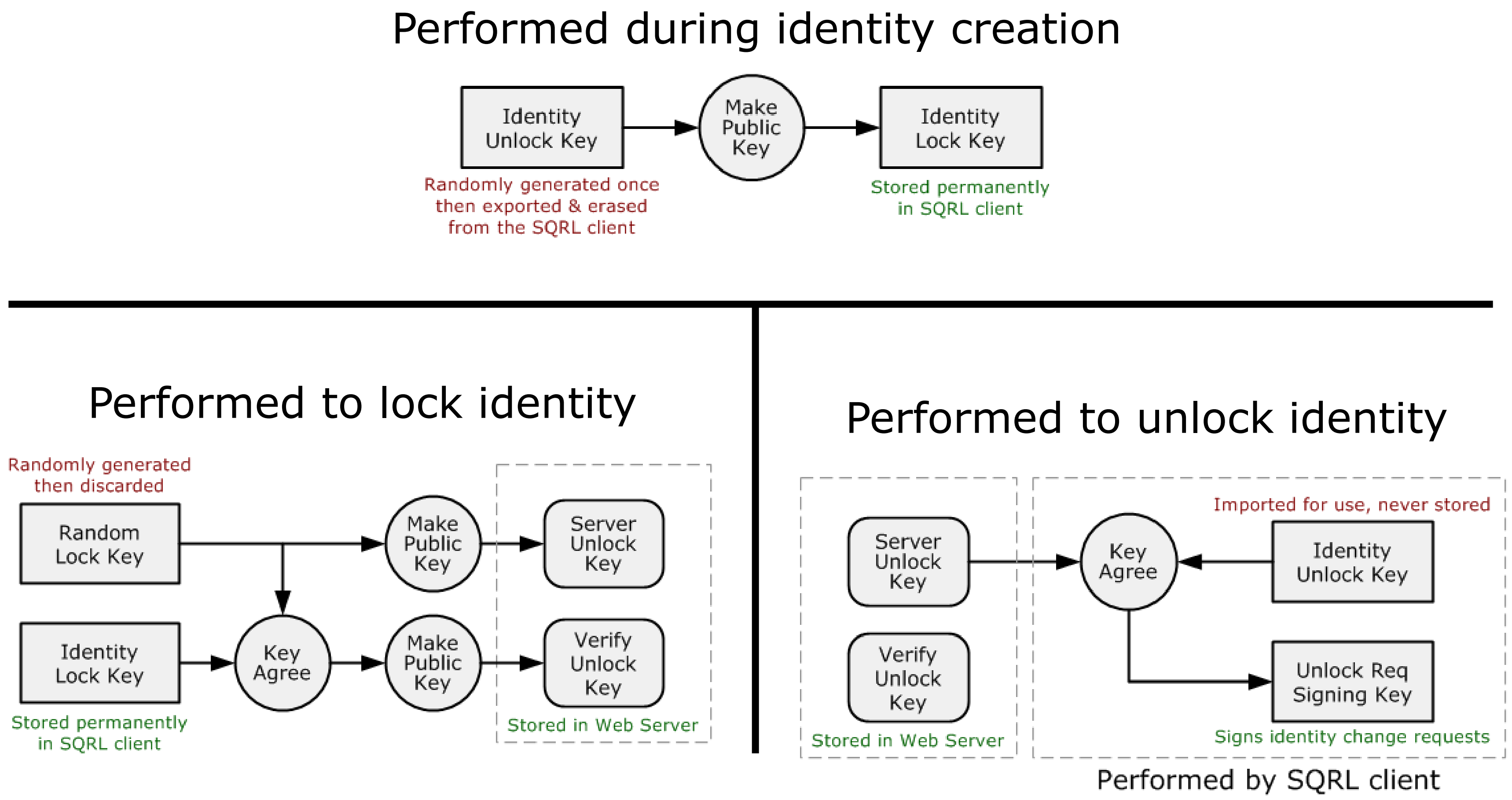
```
IdentityLock := MakePublic(IdentityUnlock)

ServerUnlock := MakePublic(RandomLock)

DHKA(IdentityLock,RandomLock) = DHKA(ServerUnlock,IdentityUnlock)

VerifyUnlock := MakePublic( DHKA(IdentityLock,RandomLock) )
```

SQRL's Identity Lock


Performed during identity creation
Identity Unlock Key
Make Public Key
Identity Lock Key
Randomly generated once then exported & erased from the SQRL client
Stored permanently in SQRL client
Performed to lock identity
Randomly generated then discarded
Random Lock Key
Make Public Key
Server Unlock Key
Identity Lock Key
Key Agree
Make Public Key
Verify Unlock Key
Stored permanently in SQRL client
Stored in Web Server
Performed to unlock identity
Server Unlock Key
Verify Unlock Key
Stored in Web Server
Imported for use, never stored
Key Agree
Identity Unlock Key
Unlock Req Signing Key
Signs identity change requests
Performed by SQRL client

See: https://www.grc.com/sqrl/idlock.htm

SQRL's Identity Lock

- The essential property of SQRL's identity lock is:

  Without the rescue code, SQRL clients only contain sufficient information **to create**, but not **to prove**, the additional cryptographic identity lock property.

- The SQRL identity's rescue code (which no SQRL client ever contains) is required to respond to a cryptographic challenge presented by a website.

- Since the rescue code is never stored in any client, it cannot be stolen and is never at risk from attackers.

SQRL's Identity Lock

- After a SQRL identity has been associated with a website, that association *cannot be changed* without the use of the rescue code. This prevents an attacker who obtains the user's SQRL identity from changing an account's previously associated SQRL identity to lockout the user.

- Any user who believes their SQRL identity may have been compromised can immediately *lockout all SQRL login* at a site to prevent the use of their SQRL identity at that site.

- The rescue code is then required to either unlock and re-enable SQRL login, or to re-key an existing SQRL identity association for a website account.

- This allows SQRL users to "take back" a lost identity. . .

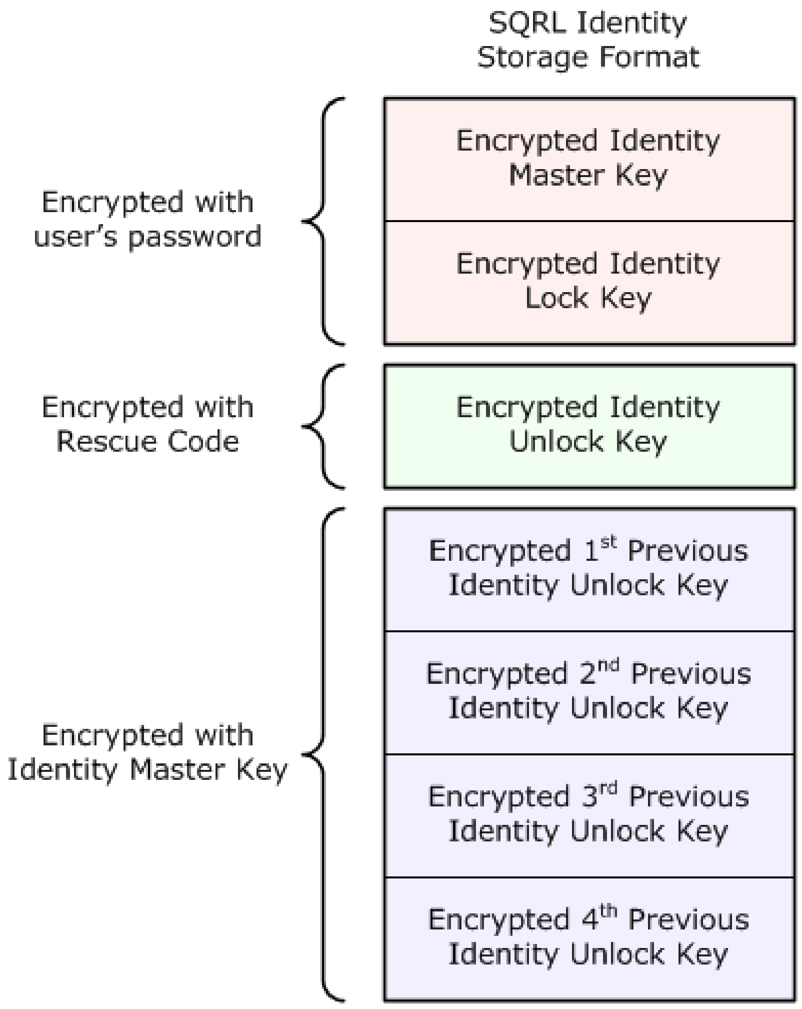
SQRL’s Identity Lock

- A user who somehow lost control of their identity could quickly lock their most sensitive accounts, preventing anyone - including themselves and any attacker – from using SQRL authentication to login.

- At that point, only a SQRL client that has been temporarily provided with the identity’s rescue code can choose to unlock SQRL authentication at that site.

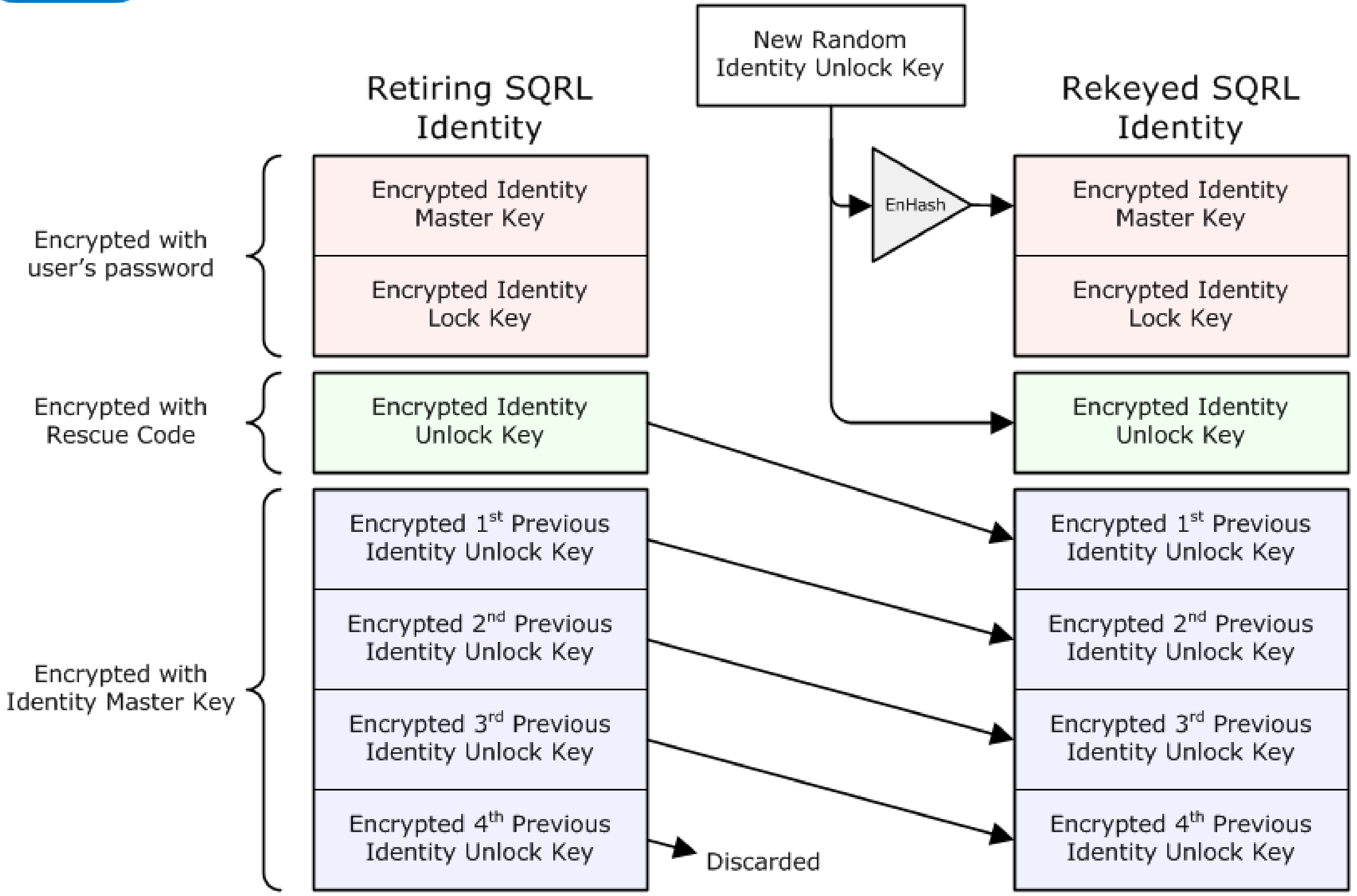
- However, most users would probably choose to rekey and replace their compromised identity. . .

Rekeying SQRL identities


SQRL Identity Storage Format
Encrypted with user's password
Encrypted Identity Master Key
Encrypted Identity Lock Key
Encrypted with Rescue Code
Encrypted Identity Unlock Key
Encrypted with Identity Master Key
Encrypted 1st Previous Identity Unlock Key
Encrypted 2nd Previous Identity Unlock Key
Encrypted 3rd Previous Identity Unlock Key
Encrypted 4th Previous Identity Unlock Key

- SQRL's identity rekeying relies upon properties of the SQRL identity storage format.
- An identity can carry from zero to four previous identity keys.
- Since the master identity key is derived from the identity unlock key, the rescue code can recover the master ID key.
- Either the user's password, or the rescue code, can decrypt the block of previous unlock keys.

Rekeying SQRL identities



New Random Identity Unlock Key
Retiring SQRL Identity
Rekeyed SQRL Identity
EnHash
Encrypted with user's password
Encrypted with Rescue Code
Encrypted with Identity Master Key
Encrypted Identity Master Key
Encrypted Identity Lock Key
Encrypted Identity Unlock Key
Encrypted 1st Previous Identity Unlock Key
Encrypted 2nd Previous Identity Unlock Key
Encrypted 3rd Previous Identity Unlock Key
Encrypted 4th Previous Identity Unlock Key
Discarded

Rekeying SQRL identities

- As shown in the previous diagram, a rekeyed identity retains the previous identity’s Unlock Key in a four-deep encrypted push-down list, accessible by either the user’s password or the identity’s rescue code.

- This allows a single composite “identity” to retain up to four previously retired identity keys.

- Since rekeying an identity should be a rare emergency measure, retaining up to four previous identity keys should protect against even catastrophic atypical misuse.

- The choice of 4 previous identities was arbitrary. Other SQRL clients might choose to support a higher number.

Server-side rekeying

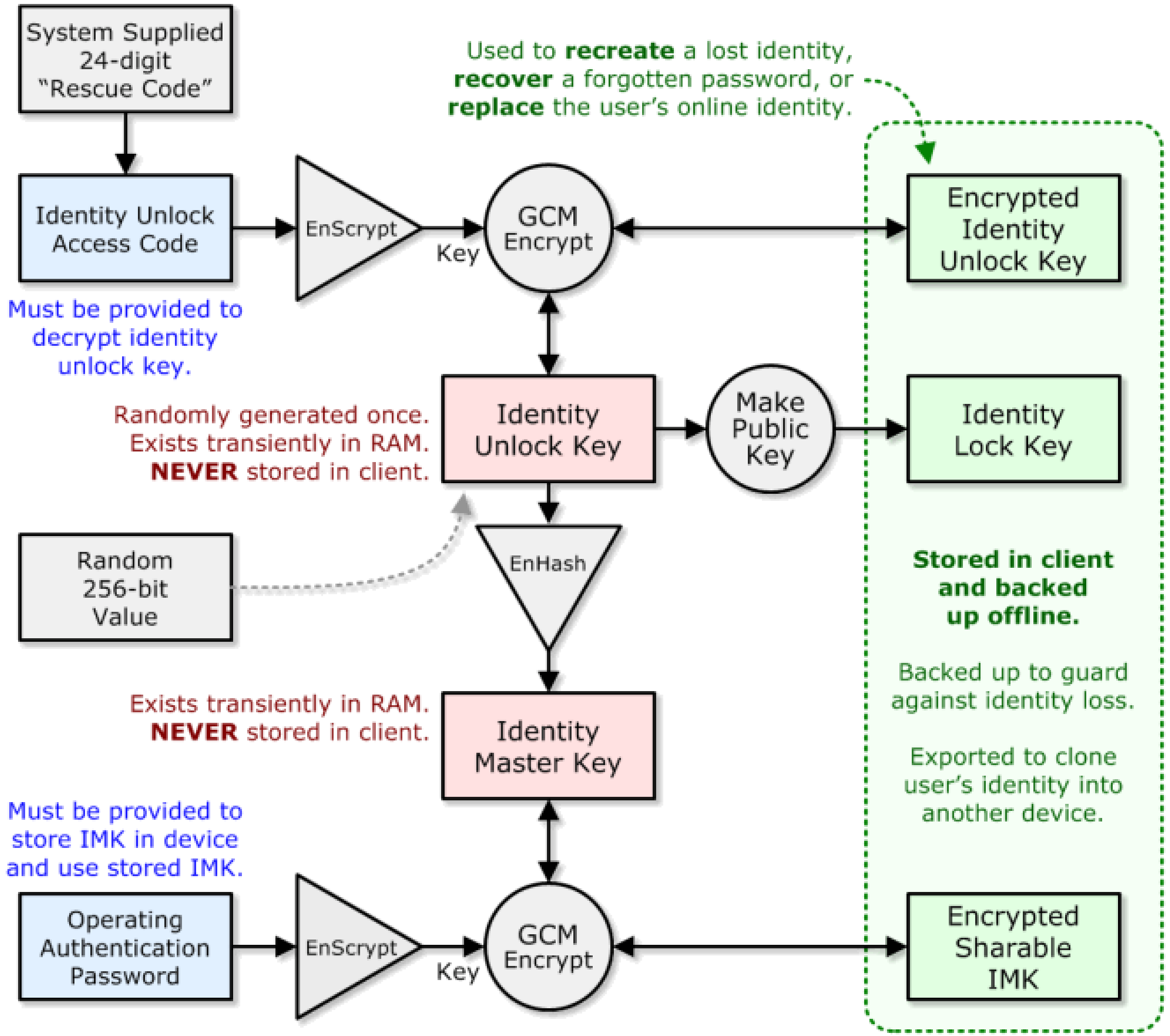
- When a SQRL user visits a website, their current identity public key is sent, and its ownership is proved by signing all query data with the matching (secret) private key.

- If the user's SQRL identity contains one or more previous identity keys, the most recent previous key is also sent, and its matching private key also signs every query.

- The status returned from the server indicates its view of the client's identities, so the SQRL client will query with successively older previous keys until it finds the one the server recognizes.

- When a server recognizes a client by a previous key, the server immediately updates to using the current key.

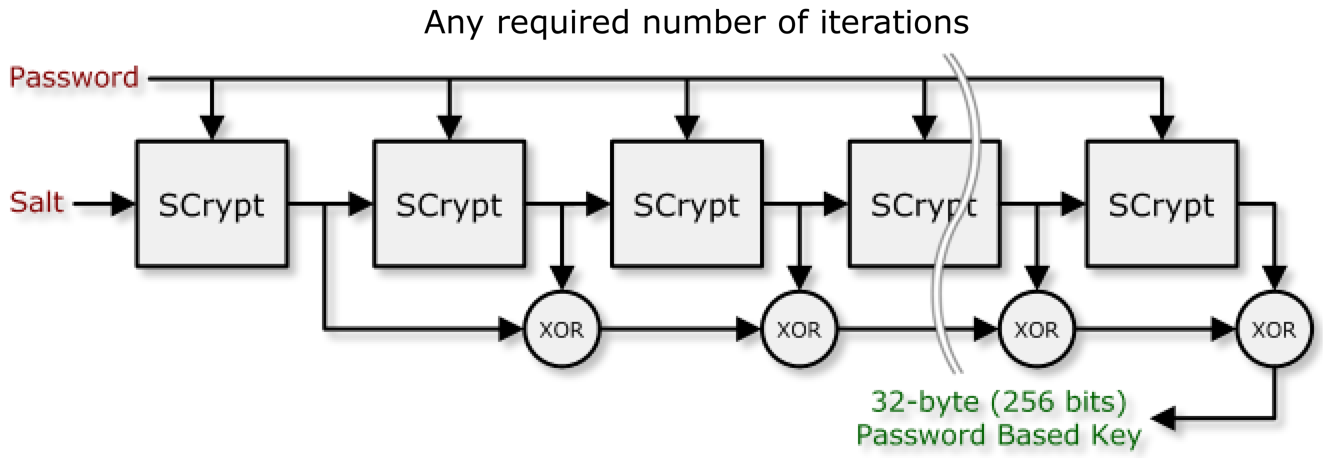
SQRL client-side key mgmt.


System Supplied 24-digit "Rescue Code"
Used to **recreate** a lost identity, **recover** a forgotten password, or **replace** the user's online identity.
Identity Unlock Access Code
EnScrypt
Key
GCM Encrypt
Encrypted Identity Unlock Key
Must be provided to decrypt identity unlock key.
Randomly generated once. Exists transiently in RAM. **NEVER** stored in client.
Identity Unlock Key
Make Public Key
Identity Lock Key
Random 256-bit Value
EnHash
Stored in client and backed up offline.
Backed up to guard against identity loss.
Exported to clone user's identity into another device.
Exists transiently in RAM. **NEVER** stored in client.
Identity Master Key
Must be provided to store IMK in device and use stored IMK.
Operating Authentication Password
EnScrypt
Key
GCM Encrypt
Encrypted Sharable IMK

Thwarting offline attack

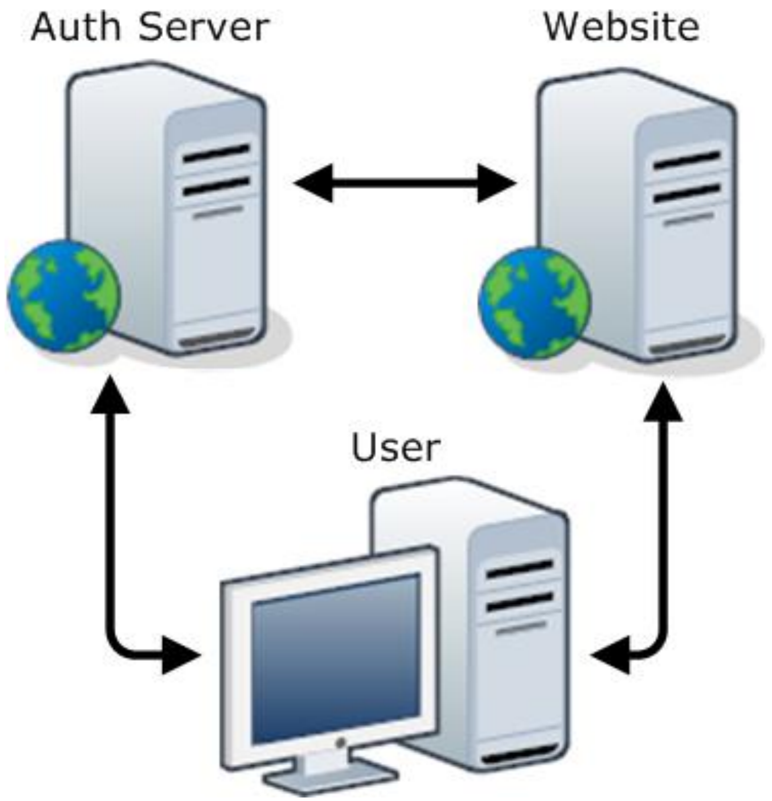
- The obvious (and only known) vulnerability remaining is brute-force password cracking against a user’s typically low-entropy access password.

- SQRL employs “EnScrypt”, a time-controllable, memory-hard function which allows password decryption difficulty to smoothly scale with processor performance:



Federated (3rd-party) identity

Although SQRL is designed for secure 2-party identity authentication, it can easily be extended to 3-party:



- SQRL's Service Provider API can be used, without modification, to create a centralized identity provider:

identity.us.gov

- In this example, any number of us.gov websites can use SQRL to obtain a single universal common identity for each SQRL user.

Improving real world security

These options request sites to disable traditional low-security authentication recovery features:

- **“Request disable non-SQRL login”** *(support optional)*
  An option in SQRL’s UI requests websites to disable non-SQRL login. The site’s traditional username & password login will not acknowledge this user’s (or any attacker’s) login, even with the proper credentials.

- **“Request no-recourse identity lock”** *(support optional)*
  An option in SQRL’s UI requests websites to disable any “out of band” changes to this user’s SQRL identity.

The state of these options are sent with every SQRL query. Websites should always capture and retain the last option settings received.

Convenience additions

- **“ShortPass”**
  GRC’s client requires the user’s full password only once per session. When the full password is provided, the first “n” characters are used to re-encrypt the identity key in RAM. During this sitting, the user may re-authenticate using only those first “n” characters to decrypt the key. If *any* mistake is made (a guess) in the short password, RAM is wiped and the full password must be provided.

- **Multiple identities per site**
  A user might wish to have additional identities for a single site. SQRL allows a user-provided string to be appended to the hashed domain name to synthesize the per-site key. This creates any number of memorable secondary identities per website.

Solving remaining problems

- **Website's domain name changes**

  Since SQRL identities are tied to the domain name, what happens when an online property changes its domain name?

  SQRL identities can be migrated to a new domain while the old domain remains available.

  The new domain is tried first. It will increasingly succeed as users are migrated to the new domain.

  If the new domain fails, a SQRL URL for the retiring domain is tried. If that succeeds, the user's identity for the new domain is updated from the old domain.

Solving remaining problems

- **Shared access to a single online account**

  For example: A husband and wife might share access to their banking site. How can SQRL provide this?

- This is currently provided for by the inherently weak identity binding provided by password secrets.

- Although SQRL clients **do** allow multiple user identities for multiple users, SQRL servers support a “many-to-one” relationship (managed shared access) between SQRL identities and web server accounts. This allows both “mom and dad” to associate their individual SQRL identities with their common online accounts.

- Thus, SQRL elegantly solves another common problem.

Managed Shared Access Demo

(Show everyone the MSA demo site. :)

Well tested off-the-shelf crypto

- All cryptographic primitives are available in the popular, widely tested, multi-platform Libsodium library.
- Dan Bernstein’s Ed25519 elliptic-curve cryptosystem.
- Elliptic curve Diffie-Hellman (via a scalarmult on EC).
- Colin Percival’s Scrypt for “memory-hard” acceleration resistant password-based key derivation.
- “EnScrypt”: SQRL’s deliberately time-consuming PBKDF2 construction, using XOR-sum chaining, with Scrypt as its pseudo-random function, scales execution time smoothly as processor performance increases.
- Miscellaneous standard crypto: SHA256, HMAC256, etc.

How does SQRL get adopted?

In so many ways it **really** is significantly superior to every other existing solution:

~~FIDO~~ ~~SMS~~ ~~TOTP~~

~~User & Pass~~ ~~OAuth~~

SQRL requires a client and some initial one-time setup, but after that **nothing else** is easier to use… or more secure.

How does SQRL get adopted?

- SQRL's adoption is inherently very "low-friction."
- A user only needs one SQRL client, and one identity, created once, which can then be used everywhere it is supported. Even if SQRL is not widely supported initially, it is extremely secure, free, lightweight and compelling.
- Some sites need, and will appreciate, ultra-low-friction account creation and robust identity authentication.
- Users will start asking non-SQRL sites to support SQRL.
- Adding server-side SQRL support is simple with the open source SSP API requiring just a few REST HTTP queries.
- Most of the "heavy lift" work in SQRL is on the client, which only needs to be implemented once.

How does SQRL get adopted?

- For the user, **all** of the work to setup SQRL is up front and one time only… while they are excited by the idea of ***never*** needing to worry about logging in again.

- After SQRL is setup once, it is the easiest-to-use identity authentication system possible.

- **How** could it be easier? There is **zero** per-site work. None. The user links their SQRL identity to a site, and they are known to that site from then on.

- Websites can relax, since they are no longer being given **any** secrets to keep. The public keys they have are only meaningful at their domain, and can **only** be used to identify and confirm a user's identity at their domain.

In summary

- Supports a single master identity, for everything.
- Incorporates practical identity lifecycle management.
- Secure against website breaches (no secrets to keep).
- Minimal 2-party solution. No central 3rd party to trust.
- Pseudonymous, unlinkable, prevents tracking.
- Simple, straightforward, open, non-proprietary & free.
- Provably secure, understandable & easily auditable.
- Most complexity resides in the client to ease server-side development. Client and server code available and free.
- Plays well with others. Easily coexists with any other identity/authentication solutions for low-friction adoption.

Discussion…

These slides: https://www.grc.com/sqrl-presentation.pdf