November 2014

SECURITY SUMMIT 2014

Las Vegas, NV

digicert

SQRL

A simple & straightforward, open, intellectual property unencumbered, easily explained, provably secure, pseudonymous, 2-party, web domain based, ***authenticated identity*** solution… for the Internet.

(**S**ecure **Q**uick **R**eliable **L**ogin.)

Presentation outline:

- Identity authentication yesterday & today.
- What identity authentication <u>needs</u> to be.
- SQRL’s proposal for tomorrow.
- The concept and the algorithm.
- Benefits and consequences.
- Operational details.
- What are the problems?
- Solutions to the problems.
- Known attacks – offline and online.
- Where do we get the crypto?
- Where SQRL stands today.
- How does it get adopted?
- Summary & questions?

The earliest authenticated identity:

- Local terminal login:
    - *Identification* by username
    - *Authentication* by password
- Remote (Internet) user login:
    - Telnet
    - FTP
- And the progress since then?

Progress?... Huh?...

Identity Auth today: (Broken)

- 2-Party Systems
  - Ad hoc, with differing, arbitrary characteristics & requirements.
  - Many very poor implementations.
  - Credentials routinely escaping.
  - In self-defense, ***<u>users bear burden</u>*** of protecting themselves:
    - Never use the same password twice!
    - Give fake responses to “security questions.”
    - Remember everything you’ve ever done!

Identity Auth today: (Broken)

- 3-Party Systems
  - Post-Snowden, no one trusts (nor should) any 3rd party.
    - Paranoia is running high.
  - The model is "we're in the middle."
  - Hosted by "for profit" entities.
  - Utilize closed "you can trust us" proprietary "magic."

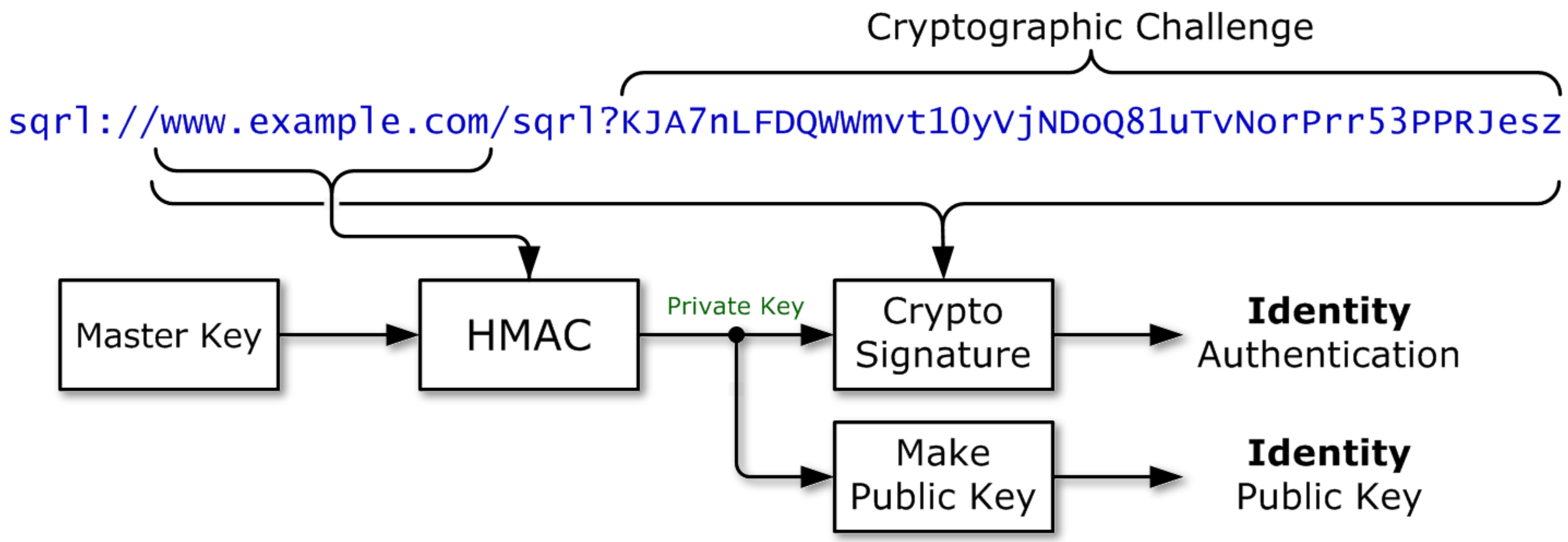
Just like information, identity authentication wants to be free!

- Identity authentication can not be a profit center.
- The winning solution will be 100% license-free and open.
  (like usernames & passwords… only something that works!)
- It will be understandable, clean and simple.
- It will have a straightforward easily understood security model.
- It will be easy and inexpensive to implement.
- It will offer “low friction” incremental adoption.

SQRL in a nutshell

- User generates one “Master Identity” for everything, forever. (A randomly chosen 256-bit token.)
- A web site’s domain name is used to key a hash which produces a private key. The matching public key is created and registered with the website as the user’s identity token for that site.
- To authenticate the user at logon, the website presents a per-logon “nonce.” The user signs the “nonce” using the site-specific private key and returns both the site-specific public key and the “nonce” signed by the matching private key.

SQRL in a nutshell



This means:

- Users are per-site pseudonymous.
  They present a unique but fixed identity to every site.
  Inter-site tracking is eliminated.

- These unique identities are synthesized from the site's domain name, so no per-site data needs to be stored.

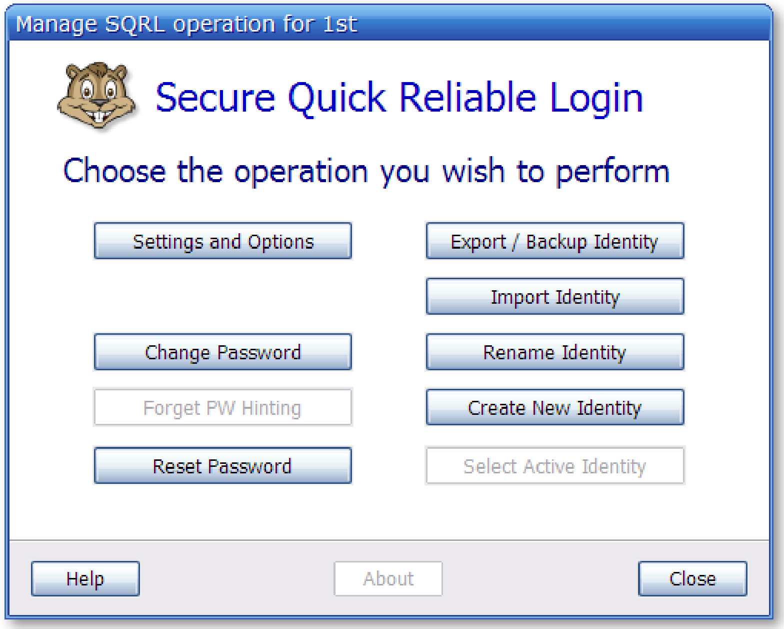
- Users give web servers "no secrets to keep." Web servers receive an identity token that is <u>only</u> meaningful for <u>that</u> site.

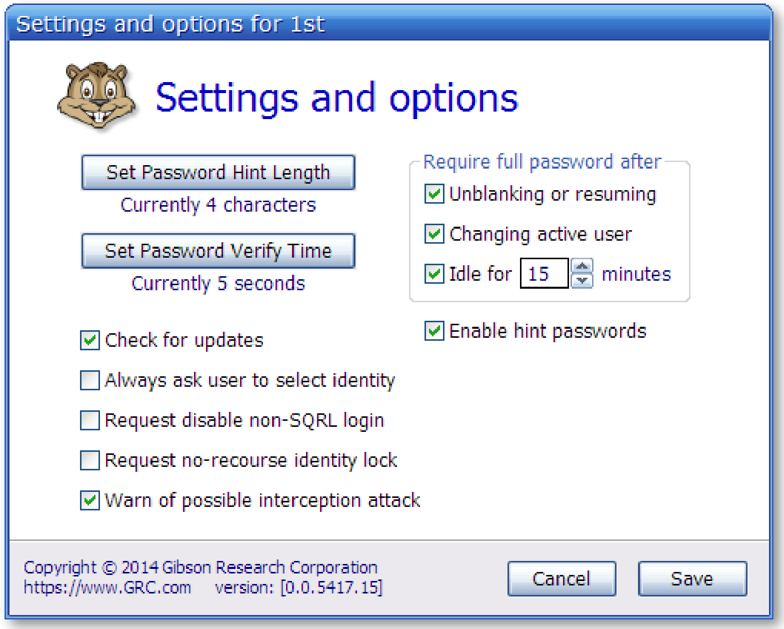
- The use of a nonce using a simple challenge/response mechanism prevents reuse / replay attacks.

Two compatible modes of operation

- Same device login
  - If the device being logged in already has a SQRL client installed, it can respond to the website's SQRL URL.
- Cross device login
  - If the user has a SQRL client installed in their smartphone, the smartphone can use the website's SQRL QR code to perform a "cross-device" login.
- Note: In neither case are ANY user credentials entered into the machine's keyboard. Using SQRL, the user just becomes "magically" logged in.

Identity management & options




Settings and options for 1st

Settings and options

Set Password Hint Length
Currently 4 characters

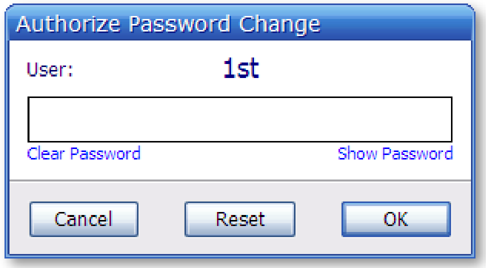
Set Password Verify Time
Currently 5 seconds

Require full password after
- [x] Unblanking or resuming
- [x] Changing active user
- [x] Idle for 15 minutes

- [x] Enable hint passwords
- [x] Check for updates
- [ ] Always ask user to select identity
- [ ] Request disable non-SQRL login
- [ ] Request no-recourse identity lock
- [x] Warn of possible interception attack

Copyright © 2014 Gibson Research Corporation
https://www.GRC.com version: [0.0.5417.15]

Cancel Save


Authorize Password Change

User: 1st

What are the problems?

(1 of 2)

- We still need to authenticate the user to their device.
- (One way to look at this is that we have reeled the authentication problem way in. With secure trans-network authentication, now the problem is mostly local… and that's much easier.)
- In the future (and kinda now) "TouchID" might be sufficient.
- Until then, SQRL uses a traditional password for "something the user knows" authentication.
- Since this single password is used only for "local" authentication to the SQRL client, it is never transmitted and is unlikely to be forgotten… especially as SQRL becomes often used.

What are the problems?

(2 of 2)

- Without a 3rd party, the user has no recourse.
- *“I forgot my password!”*
- *“I just changed my password and forgot the new one!”*
- *“Malware got into my phone and stole my SQRL identity!”*
- The whole point of SQRL is that websites cannot help either. (If they could help, they could be hacked.)
- So we had to come up with some sort of secure emergency recovery system that worked entirely on the client side…

The SQRL “Rescue Code”

- An offline, guaranteed-high-entropy, 24-digit random number.
  Example: **1484-3606-4253-9577-0233-6070**
  (Not user-generated, since people are not random.)

- Generated randomly when a SQRL identity is generated.

- NEVER stored in any device, must be printed or recorded
  and stored offline.

- Has two special powers:

    - Forgives any lost or forgotten password,
      so used for “password recovery.”

    - Unlocks SQRL’s “Identity Lock” for identity management…

SQRL's "Identity Lock" protocol

(1 of 3)

- A unique application of Diffie-Hellman key agreement.
- SQRL clients carry sufficient information to generate new identity associations, but NOT to authenticate associations.
- Only the "Rescue Code" is able to authenticate (and thus "unlock") an association.
- Since the "Rescue Code" is never stored in any SQRL client, it cannot be stolen by malware or bad guys.
- If, somehow, a user were to lose control of their identity, bad guys could not lock them out of their accounts.
  But users can lock the bad guys out by using their rescue code to proactively change and take back their identity.

SQRL's "Identity Lock" protocol

(2 of 3)

Diffie-Hellman Key Agreement:

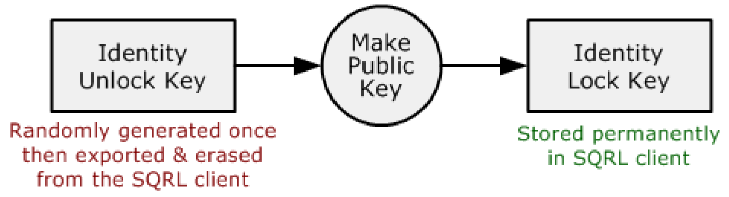
If we have public key one (PubKey1) and secret key one (SecKey1) and public key two (PubKey2) and secret key two (SecKey2), then:

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

IdentityLock := MakePublic(IdentityUnlock)

ServerUnlock := MakePublic(RandomLock)

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

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

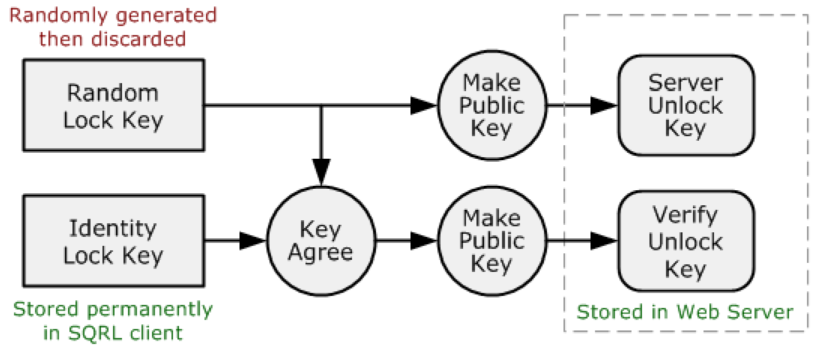
SQRL's "Identity Lock" protocol

(3 of 3)

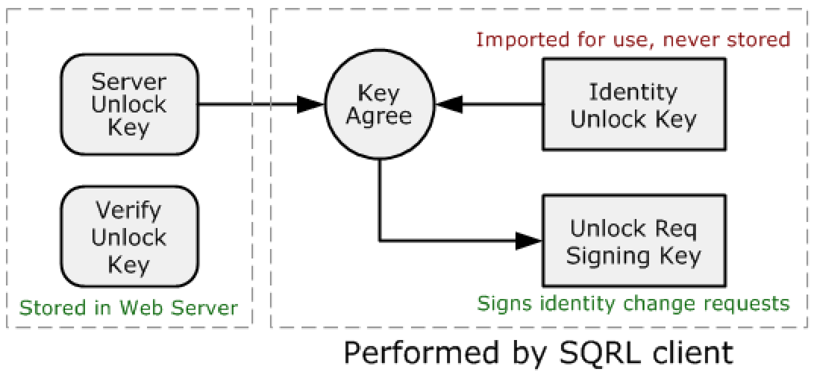
Performed during identity creation


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



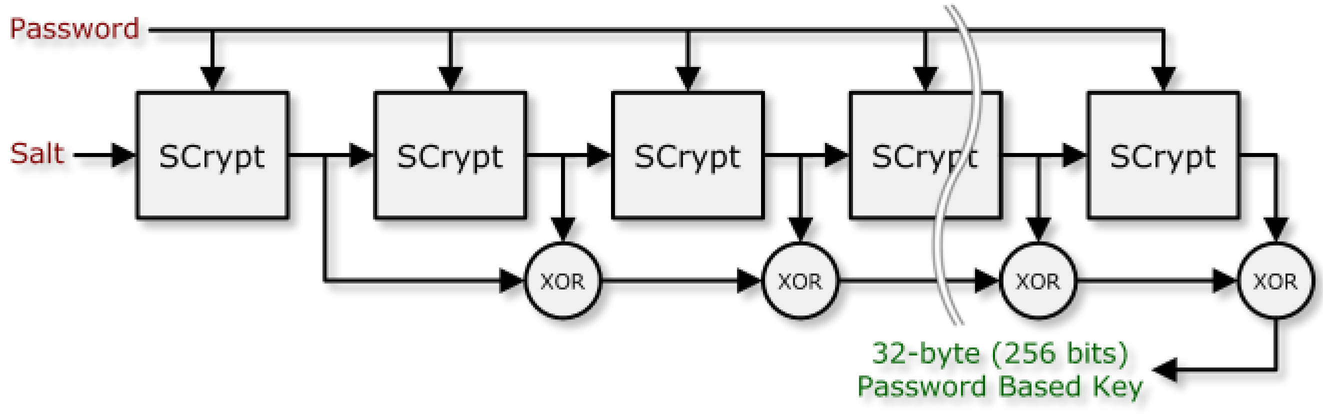
Performed to unlock identity



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

Known (offline) attacks

- Only known weakness is brute-force password cracking.
- SQRL adds some unique technology to the industry:
- <u>Very time consuming</u> memory-hard one-time password process:



- Subsequent "Password Hint" for successive re-auths.

Known (online) attacks

- Spoofed SQRL login for site B on site A:
  - “LooksInnocent.com” presents user with SQRL login for “www.Amazon.com” and obtains logged in session.
- Spoofed “www.Amaz0n.com” site presents user with SQRL login for the real Amazon.com site.
  - Warn user of IP address differing between queries!
- Man-In-The-Middle interception of HTTPS
  - If attacker has this power, “authentication” is the least of the user’s problems!
- More and more of the Internet relies crucially upon the functional strength and integrity enforced by the Certificate Authority (CA) system.

“Off the shelf” implementation crypto

- Colin Percival’s Scrypt for “memory-hard”, acceleration resistant password-based key derivation.
- A deliberately time-consuming PBKDF2 construction, using XOR-sum chaining with Scrypt as its PRF, to render brute-force cracking orders of magnitude more difficult and daunting.
- Dan Bernstein’s “NaCl” 25519 elliptic-curve cryptosystem.
    - EdDSA (elliptic curve digital signature system).
- Elliptic curve Diffie-Hellman (via a scalarmult on EC).
- Cross-platform “Libsodium” library.
- Misc. crypto pieces… SHA256, SHA512, HMAC256, etc.

Where does SQRL stand today?

- Client support
  - GRC's reference (fully usable) SQRL client
  - (Github) A functioning Android client
  - (Github) A SQRL client in Python.
  - (Github) A SQRL client in Java.
  - (Github) A Windows Phone 8 Proof of Concept.
  - (Github) Some work on a iOS library.
- Server support
  - GRC's ISAPI (Windows IIS) reference library.
  - (Github) A Drupal implementation
  - (Github) A PHP server-side implementation
  - (Github) A "Dot-Net" full SQRL implementation.
  - (Github) A Haskell implementation.
- Various sites, guides and articles.

How does SQRL get adopted?

- Everything about SQRL is “low-friction”:
- A user only needs ONE identity created ONCE.
  (Even if it’s not heavily used at first, it’s free and compelling.)
- Some sites need, and will appreciate, ultra-low-friction identification.
- Over time, users will (hopefully) ask for SQRL support everywhere.
- Adding server-side SQRL support is already easy and will become easier.
- Most of SQRL’s “heavy lift” (which already is not very heavy) is on the client, which only needs to be implemented once.

In summary

- Encourages one identity for everything.
- Secure against website breaches (no secrets to keep).
- Pseudonymous, prevents tracking. Users choose to de-anonymize.
- Simple, straightforward, open, non-proprietary, completely free,
- Provably secure, understandable and easily auditable.
- Most complexity in the client to ease server-side development.
- Can co-exist side-by-side with existing identity & authentication.
- Low-friction adoption… so perhaps it has a chance ?

Questions?