FASSKEY: A Secure and Convenient Authentication System

John Milburn
FASSKEY Technology, Inc.
Email: jem@fasskey.com

Heejo Lee
Dept. of Computer Science and Engineering
Korea University
Email: heejo@korea.ac.kr

***Abstract*—We present an identity authentication system which is cryptographically strong, pseudonymous, efficient enough to run well on current smart phone devices, and easily extensible for payment and banking functionality. We also describe high security algorithm and programming methods for the implementation, including server, network transmission, and application development. The system is intellectual property unencumbered and provably secure. The end user app implementation uses three factor security, a combination of unique device, user password, and fingerprint. We use well-known and proven cryptographic primitives.**

I. INTRODUCTION

FASSKEY is a replacement for the traditional username/password system still in use on most Internet systems. It replaces the old system with a simple method based on strong public key cryptography, in a way which is transparent and easy to use. User secrets are generated by the user device and only ever known by the user. The architecture and implementation are focused on creating a simple and intuitive user experience while maintaining very strong security – goals which are often thought to be mutually exclusive.

Also, by virtue of its strong authentication capability, FASSKEY securely enables other processes which require authentication, including payments, P2P transfers, banking, POS purchases, and ATM use. While there are many apps which provide some of these features, they often lack the security and authentication architecture required. Attacks on banking applications in Korea, Australia, New Zealand and Turkey have become commonplace[1][2].

FASSKEY is based on the SQRL system described by Steve Gibson[3]. SQRL is designed as a username/password replacement for websites. It uses a user generated secret master key as the basis for per-site key pairs. SQRL has some features unique to cryptographic authentication methods, particularly a per site lock and unlock capability, and the ability for a user to change his credentials in a cryptographically secure way.

We have extended and enhanced the SQRL protocol to add provable site verification, transmission encryption, identity management, transparent and continuous backup, multi-device synchronization, batch mode processing for lock/unlock/re-key operations, an eWallet system, shopping payment methods, P2P payments, casual POS sales, banking, and ATM access.

978-1-4673-9486-4/16/$31.00

A. Logon with *qlink*

For a normal FASSKEY log on process, the website presents a *qlink*, a URL which includes cryptographic information. The *qlink* is delivered to the user's client application by clicking on a link, scanning a QR code[4], or via NFC transfer. A typical FASSKEY login *qlink* is shown in Figure 1. The *qlink* provides a 32-byte "nut", a nonce which includes a timestamp, server identifier, system identifier, random data and a truncated MAC. The *qlink* also provides the FASSKEY protocol type ("SQT-AUTH" in this case), and a 32-byte ephemeral public key (epk). The user client application generates the secret and

```
qrl://demo.fasskey.com/squal_auth_login.php?
nut=zBPmr7UN1rO-kMCWFAfAqEtS5ibAqOYq59n7Cxtlse0
&protocol=SQT-AUTH
&epk=ka46kc_XlCevG34NAYwizRqq9JRJ58QGphQ-5o24Hds
```

Fig. 1. A FASSKEY *qlink*

public keys for the site, and responds with the public key and the full *qlink*, signed by the secret key. This allows the site to confirm that the public key, which corresponds to the user's account at that site, is correct, in that it is signed by the holder of the matching secret key.

B. Cryptography Used

Our implementation uses secure techniques throughout. The established cryptography used includes AES-GCM[5] for encryption and authentication of messages, SHA256 and SHA512 for hashing, twisted Edward's curve digital signature algorithm (EdDSA)[6] for all signatures, and elliptic curve Diffie-Hellman key agreement (DHKA)[7][8] for secure key derivation. The Elliptic Curve used is x25519, as used in many Internet systems[9]. This curve and its 256 bit keys gives a security level of about 2^{126}[10], which is roughly equivalent to the security of a 3 kbyte RSA key. We do not use RSA style cryptography[11] anywhere in the system.

II. RELATED WORK

Existing non username/password login methods, such as OAUTH, rely on credentials from an authentication service provider. This means that the user must put all trust and confidence in the service provider, as the service provider

has access to all user connections, user data, and user secrets. There are also fundamental security and authentication issues with OAUTH, as pointed out by John Bradley in [12]. Most sites using some type of Single Sign-On (SSO) method have the inherent problem of giving the user credentials to the website, removing control from the user over exactly how those credentials are used[13].

All SSO methods have the additional problem of allowing cross-site user tracking, by authenticating service and by connected sites, a type of user privacy invasion.

Any public key system which uses the same key pair for multiple sites has both the cross-site tracking problem, and the additional problem of key overuse.

Two-factor authentication methods are cumbersome for the user and have shown frequent flaws[14][15].

III. SQRL Protocol

SQRL is a protocol that allows for easy and secure multiple site access with no cross-site correlation or tracking ability. This protocol uses strong cryptography methods along with strict user control of identity keys. Deterministic per-site key pairs are generated on the fly and only the public key is disclosed to any website. SQRL also gives the user unique ways to control his identity by having the capability to replace (or re-key) the credentials should they be compromised and lock/unlock sites as deemed necessary.

A. SQRL *in a nutshell*

- User generates one "Master Identity" for everything, forever. (A randomly chosen 256-bit value.)
- A website'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.
- 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 websites 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 only meaningful for that site.
- The use of a nonce applying a simple challenge/response mechanism prevents reuse/replay attacks.

B. Site Specific Key Pairs

A key feature of the SQRL system is the ability to deterministically generate a new and uncorrelated key pair for each site. This is achieved by hashing the user's master key with the website's domain name, via a SHA256 based HMAC function[16], to create a site specific private key. This high entropy result is directly usable as a secret key, and from it a public key is generated, as shown in Figure 2. This is possible with elliptic curve public key cryptography, and is fundamentally not possible with the older, well known "products of primes" RSA cryptography.

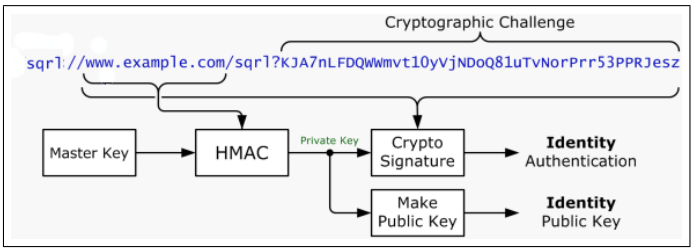


Fig. 2. Site specific key generation and signing

C. Lock and Unlock

When a user first registers to a site, he transmits not only his public key, but also two others keys, the Server Unlock Key (SUK) and Verify Unlock Key (VUK). Should the user ever believe that his master key has been compromised, the user may lock his account on the server. Once locked, the user can unlock it only by using his Identity Unlock Key (IUK), which is never stored in the application or on his device.

1) Lock Key Generation: The IUK is generated at the same time as the user master key. Using the IUK as a secret key, its public key is generated as ILK = $MakePublic$(IUK) and called the Identity Lock Key (ILK) (Figure 3). The ILK is stored in the client along with the Master Key. The IUK is printed (as a QR code) or stored, then deleted from the device.

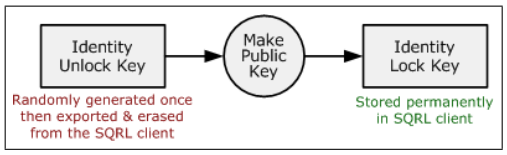


Fig. 3. IUK and ILK Key Generation

2) Locking Methodology: Account locking relies on basic properties of Elliptic Curve Diffie-Hellman Key Agreement (DHKA)[8]. We generate a random secret key and call it the Random Lock Key (RLK). From this we make its public key, and call it the Server Unlock Key (SUK) SUK = $MakePublic$(RLK). Using DHKA, we know that DHKA(RLK,ILK)=DHKA(SUK,IUK). As the ILK is available in the client, we can generate the agreed key. We further create the verification key (VUK) by treating the agreed key as a secret key, and generating from it a public key, i.e. VUK = $MakePublic$(DHKA(RLK,ILK), as shown in Figure 4. The SUK and VUK are sent to the server and stored by the server along with the user's identity, and the RLK is discarded.

To unlock the account, the user requests the SUK from the server. The user must now combine the SUK with his (offline and very secret) Identity Unlock Key to generate the agreed

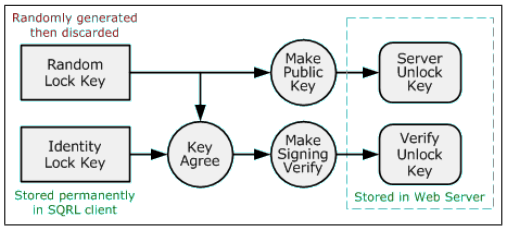


Fig. 4. SUK and VUK Key Generation

key via DHKA(SUK,IUK). This resulting key is used to sign the unlock request (Figure 5). As the server already knows VUK, which is the corresponding public key, the server does a signature check using VUK to confirm that the user possesses the correct IUK. If confirmed, the account is unlocked. A simple illustrated guide to these processes is shown at [17].

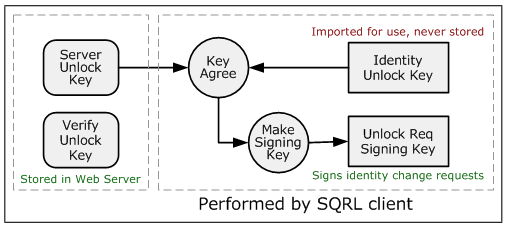


Fig. 5. Unlock Process

3) Re-keying: The same mechanism used for Lock/Unlock is used if a user needs to replace his master key. Detail information on the key replacement mechanism can be found at the SQRL website[3].

IV. FassKey - SQRL Extensions

We have extended and enhanced the SQRL protocol to add provable site verification, transmission encryption, identity management, transparent and continuous backup, muti-device synchronization, an eWallet system, shopping payment methods, P2P payments, casual POS sales, banking, and ATM access. The overall FassKey system architecture is shown in Figure 6.

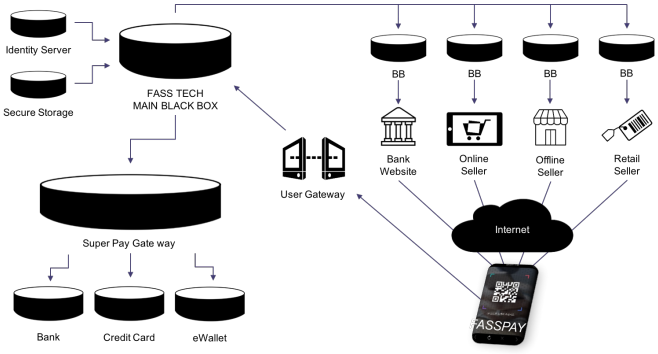


Fig. 6. FassKey System Architecture

A. The FassKey Application

FassKey clients are currently available for iOS and Android. The FassKey app is shown in Figure 7. The app

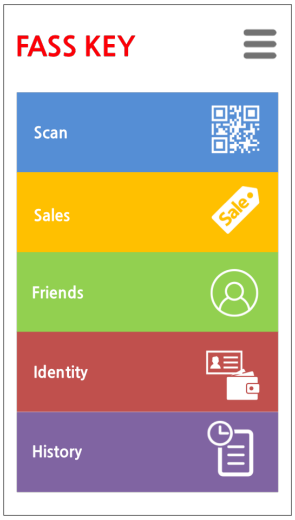


Fig. 7. Front Page of the English FassKey App

implements all of the FassKey features described. Multiple languages are supported.

B. Attestation - Site Verification

1) Initial Connection: At first run, the FassKey app connects to a predefined attestation server. This initial connection is made from the client, using a client generated nonce. The app has two pieces of embedded data, the domain name of the attestation server, and the public key of the attestation server (ADPK). The client uses the domain name to generate a key pair (ASSK, ASPK) for the attestation server, as in Figure 2. The client generates a first session key K via DHKA for the initial connection, as K = DHKA(ASSK, ADPK). The client uses K to encrypt the message to the attestation server using AES-GCM[5], along with the ASPK in the clear, all signed by the ASSK.

The server is able to verify the signature of the message, and then uses its secret key (ADSK) to generate the decryption key as K = DHKA(ASPK, ADSK). The attestation server next generates a user specific key pair (USSK, USPK) via an HMAC operation, USSK = HMAC(ASPK, ADSK) and USPK = $MakePublic$(USSK). The attestation server responds to the client with a new session key, and delivers the USPK to the client. The client stores this key, and uses it as the basis for all future communication with the attestation server, rather than the ADPK, to prevent overuse of the primary key pair.

2) Attestation: Websites which participate in the FassKey system register with the attestation server, By this registration, the attestation server stores the website public key (SDPK).

When a client connects to a website. it receives a nonce from the website. Encoded into the nonce is a 4 byte website identifier, and a 4 byte attestation server identifier. The client presents this nonce to the attestation server, encrypting the

message with the key $K = \text{DHKA}(\text{USPK}, \text{ASSK})$, and signing with ASSK. The attestation server is able to decode the message via the equivalent key $K = \text{DHKA}(\text{USSK}, \text{ASPK})$, after checking the signature. The attestation server decodes the website from the identifier in the nonce, and responds to the client with the website public key SDPK.

C. Transmission Link Encryption (TLE)

FASSKEY sites, when generating the initial nonce, also generate an ephemeral secret key (SESK). From that key, a public key ($\text{SEPK} = MakeKey(\text{SESK})$) is generated, and delivered along with the nonce in the $qlink$, as shown in Figure 1. The client generates its key pair for the site (SSSK, SSPK), gets the site's general public key (SDPK) from the attestation server, and generates its own ephemeral secret key (CESK) and corresponding public key ($\text{CEPK} = MakePublic(\text{CESK})$). The client does a series of three DHKA operations:

$$\left.\begin{aligned} k1 &= \text{DHKA}(\text{SEPK}, \text{SSSK}) \\ k2 &= \text{DHKA}(\text{SDPK}, \text{CESK}) \\ K &= \text{DHKA}(k1, k2) \end{aligned}\right\} \quad \text{Client Session Key} \quad (1)$$

K is used to key the AES-GCM encoded response to the server. With its response, the client sends, in the clear (but signed), both the site specific public key SSPK and its ephemeral public key CEPK.

The server receives sufficient information to check and decrypt the message, using the three-step key generation method to build the same set of keys:

$$\left.\begin{aligned} k1 &= \text{DHKA}(\text{SESK}, \text{SSPK}) \\ k2 &= \text{DHKA}(\text{SDSK}, \text{CEPK}) \\ K &= \text{DHKA}(k1, k2) \end{aligned}\right\} \quad \text{Server Session Key} \quad (2)$$

This server generated K is the same as the client generated K, and thus can be used to decrypt the AES-GCM encoded payload.

By incorporating ephemeral keys at both the server and the client side, full perfect forward secrecy is achieved. Also, since the server public key SDPK is delivered by the attestation server, no Man-in-the-Middle (MITM) has sufficient information to intercept and mimic another server, or to re-create the session key.

D. Identity Management

FASSKEY allows the user to store personal information securely in his device. This information may be used to simplify interaction with service providers of all sorts.

From the perspective of protecting information, the user may wish to always be completely anonymous. This is not achievable in the current Internet environment. Many nations have regulations which require some type of proof of identity for registration to web services. In other cases the requirement comes from the website or service itself. As such policies vary significantly between countries, FASSKEY implements such policies on a per country basis. For example, in Korea certain connections or transactions require "Mobile Device Authentication", a method to correlate user identity to his registered, post-paid mobile phone. Confirmation is done by a code sent via SMS message from a government approved service.

In FASSKEY, all identity data is stored, encrypted, on the user's device. When a site requests data from the user, the app prompts the user with the listed information and displays the stored data. The user then has the option to send the information, edit the reply, or cancel and send nothing.

Payment mechanisms, such as credit card, debit cards, etc., can be entered into the identity data, and then used automatically to make payments. Delivery addresses may also be entered, and later shared easily during a purchase. A sample of our credit card data entry screen is shown in Figure 8.

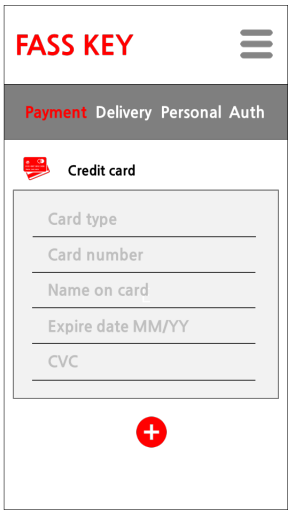


Fig. 8. Credit Card Info Entry Page

E. Transparent Backups

All end user data is backed up to a FASSKEY service backup server. All such data is encrypted by the client using AES-GMC with a per-record derived key, then sent to the backup server using the TLE protocol. The data at rest on the backup server is unreadable by the backup server operator.

F. Multi-device Synchronization

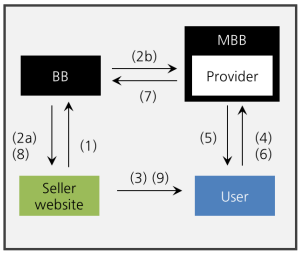
All user database entries have per record time stamps for both creation and modification time. The client program periodically connects to the backup server and updates the local on-device database should updates be found. This is done for all user database types, including site-visited DB, identity DB, and alt-id DB.

G. eWallet

Every user within the FASSKEY system has an eWallet, as does every participating website. Users may add value to their eWallet via receipt from other users, cash deposit via retail merchants, cash deposit via ATM machine, or charging from credit card, cash card, or bank account.

H. Shopping Payment

FASSKEY partner websites can enable shopping payments. Users need not register an account at the site to enable shopping, as payments are handled by the FASSKEY payment server (MBB). For simplicity, FASSKEY provides a dedicated server to the website (BB), which provides all cryptographic functionality. The website connects to the BB server via a RESTful protocol running inside a dedicated VPN link. The protocol flow for shopping is shown in Figure 9.



V (1). Seller website gets qlink from BB.
V (2a). BB sends qlink to Seller website.
V (2b). BB registers qlink with MBB.
A (3). User gets qlink from Seller website.
T (4). User sends qlink to Provider(MBB), proving Seller website.
T (5). MBB confirms qlink and sends payment query to User.
T (6). User confirms purchase and payment method.
V (7). MBB sends purchase confirm to BB.
V (8). BB confirms purchase to Seller website.
A (9). Seller website updates to show paid.

V - VPN connection
T - TLE protocol
A - Visual QR Code or NFC

Fig. 9. Shopping Protocol Flow

Note that no user payment information is sent to the selling website, as all payments are processed by the FASSKEY payment server. The seller's eWallet is credited with the payment, and required delivery information is provided from the FASSKEY app. Again note that to complete a purchase with a partner merchant, buyer need not register.

I. P2P Payment

Any FASSKEY user can add any other FASSKEY user to his friend list. This is done securely – face to face via QR code or NFC[18] exchange with cryptographic confirmation. The user may now transfer eWallet funds from his own account to the friend's account. We specifically disallow remote fund transfers to non-friends, to minimize phishing and other attacks or fraudulent behavior.

J. Casual POS Sales

Using the Sale button available to all users, face to face payments are enabled. The seller enters the amount of the sale, after which a QR code is displayed on the seller's screen. Then the buyer scans the code, selects a payment method, and approves the transaction. After processing at the FASSKEY payment server, the payment amount is credited to the seller's eWallet.

K. Banking

FASSKEY's banking protocol allows strong, secure access to individual bank account management via web or app access. The app requires fingerprint authentication, One-Time Password OTP[19] device, or PIN code for certain operations, as specified by the bank.

L. ATM

FASSKEY's ATM protocol is designed to minimize changes to existing ATM systems. FASSKEY participates in the existing ATM system as a bank member, and treats user eWallets as accounts. Use is shown in Figure 10.

a. User initiates ATM session, requests FASSKEY link.
b. ATM machine displays QR code and/or NFC connection.
c. User chooses account to use. App sends account auth info to bank's backend system.
d. User chooses transaction type via ATM terminal.
e. User receives or deposits cash.

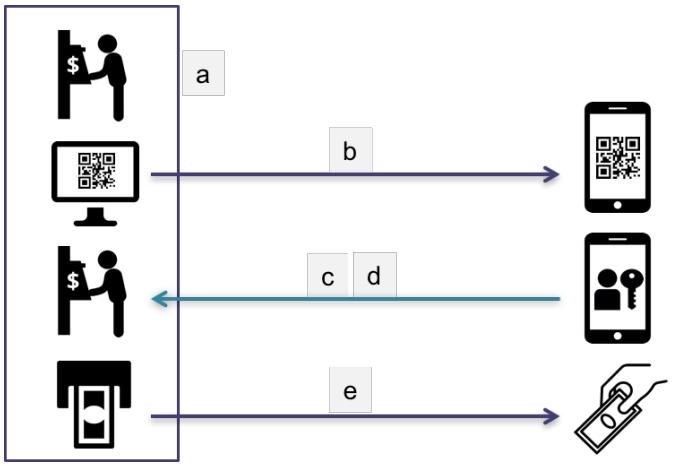


Fig. 10. FASSKEY ATM method

V. VALIDATION - ATTACK MITIGATION

Typical attacks are against the app, the network communication, or the server. Typical attack types are shown in Table I. Current smart device apps are exceptionally vulnerable to many of these attacks. Many network protocols have also been proven to be unsafe.

FASSKEY is resistant to all these attacks. Some of the methods we have designed and implemented include:

1) Attestation: When connecting to any site, the encoded nonce from that site includes site identifying information. We connect, via a TLE encrypted connection, to the central server and confirm the identity of the site. The data received in the confirmation message, plus some random component, is used to create the session encryption key for the response message. No MITM can recreate or spoof the necessary key components. This gives absolute assurance that the site is valid, and assures the user that no one except the real site can decrypt his response messages.

2) Scrambler: All memory used by the app to store any data is done via a scrambled memory system. In addition to the system-level and compiler-level scrambling, our scrambler uses high quality entropy from our internal entropy harvester to do runtime scrambling. Decrypted keys or other data are stored and scattered amidst random data, making memory dump analysis impractical.

TABLE I
TYPICAL ATTACK TYPES

	Description
MITM	Man in the Middle, attacker intercepts and substitutes malicious data in the client-server transaction.
Replay	Attacker records and then re-uses real client data to mimic the client.
Data Dump	Attacker causes the app to dump its memory to a file. Memory is inspected to find keys and user private information.
Key Logger	Malware in the OS records keyboard events to reconstruct passwords.
Side Channel	Malicious app, or connected machine, analyzes target system to get key information.
Fake App	Attacker creates or modifies app to obtain a user's password.
Password Cracking	Brute force attack to find passwords.
Reverse Engineering	Attacker runs the app in debug mode, traces execution, in order to change data.
Buffer overflow	Attacker causes the app to execute malicious code.
App infection	Attacker modifies or changes operation of app.

3) TLE*:* All network connections are done using AES-GCM encryption of payload, using ephemeral derived keys for session based key generation, to provide both a public key agreement method, and perfect forward secrecy. This completely stops network level eavesdropper activity and attacks.

4) Random payload: All network communications include random payload components, to defeat any type of known text or replay attack.

5) Secure Wiping: All data stored in memory is securely erased immediately when it is no longer needed. The secure wipe is done by first overwriting the real data with random values, then deallocating the memory.

6) Constant time functions: All cryptographic functions are designed and implemented using constant time and constant compute operation. This means that the processing path is not in any way dependent on the key values or data values. This eliminates most types of side channel attacks.

7) Single use: Every nonce and $qlink$ in our system is single use. No nonce may be reused in any situation. This eliminates replay attacks.

8) Time constraint: All nonces are also time constrained. There is a relatively short time window during which the nonce may be used. If there is any attempt to use the nonce outside of the time window, processing is denied.

9) Device binding: Decryption of the stored, encrypted master key data file is locked to the device. After copying this stored data to another device, the attacker is left with only brute force attacks, even if he knows the users password.

10) Anti-Reverse Engineering: Although there is no perfect solution to prevent reverse engineering, our goal is to make it as difficult as possible by including: code function name scrambling, string encryption, time window trapping between threads and function calls, critical data randomization and data flow abstraction.

11) Buffer overflow protection: Strict data boundary control is implemented for all incoming and run time data. All data is passed between threads or over the network using serialized buffers, with strict boundary checking in both serialization and de-serialization processing.

12) App infection: Any attempt to tamper with our app will result in alarms sent to the FASSKEY monitoring system. These alarms are sent directly, or are queued and sent when the device is online.

13) Key Logging: The one remaining vulnerability is key logging. This is an especially large problem on Windows machines, somewhat less so on Android, a modest issue on Mac OSX, and highly unlikely on iOS devices. This is an Operating System and hardware device and driver issue. We always implement industry best practices to mitigate this attack.

VI. CONCLUDING REMARKS

FASSKEY, based on the SQRL protocol, provides for per site deterministic key pair derivation, removing any need to store such keys, and removing any cross-site tracking capability. The cryptographic signature method is strong against all known attack methods. We use established, trusted cryptographic primitives with no known exploitable weaknesses. No user secret data is stored on servers, so such servers are less attractive targets, and cannot release important user authentication data if compromised. FASSKEY extends SQRL with multiple new features, including provable site verification, transmission encryption, identity management, transparent and continuous backup, muti-device synchronization, an eWallet system, shopping payment methods, P2P payments, casual POS sales, banking, and ATM access. FASSKEY combines simple usability with high security. It is suitable for all ages. Incorporation of payment mechanisms is efficient and secure, allowing for low transaction fees and ease of use.

ACKNOWLEDGMENTS

The authors would like to thank Steve Gibson for his introduction, description and development of the SQRL protocol.

Initial support in November, 2013, was provided by KT Corporation for a proof of concept implementation of a server and an iPhone application.

REFERENCES

[1] Simon Huang, *The South Korean Fake Banking App Scam*, Trend Micro, https://www.trendmicro.com/cloud-content/us/pdfs/security-intelligence/white-papers/wp-the-south-korean-fake-banking-app-scam.pdf, 2015.

[2] ESET, *Android Trojan Targets Customers of 20 Major Banks*, ESET https://t.co/rGpEmfwJjm, March 2016.

[3] Steve Gibson, SQRL – *Secure Quick Reliable Login*, https://www.grc.com/sqrl/sqrl.htm, October 2013.

[4] ISO/IEC 18004:2000, *Information technology - Automtic identification and data capture techniques - Bar code symbology - QR Code*, June 2000.

[5] Wikipedia, *Galois/Counter Mode*, https://en.wikipedia.org/wiki/Galois/Counter_Mode

[6] Bernstein, Daniel J.; Duif, Niels; Lange, Tanja; Schwabe, Peter; Bo-Yin Yang (2012). *High-speed high-security signatures*, https://ed25519.cr.yp.to/ed25519-20110926.pdf. Journal of Cryptographic Engineering 2 (2): 7789. doi:10.1007/s13389-012-0027-1 https://dx.doi.org/10.1007%2Fs13389-012-0027-1.

[7] Daniel J. Bernstein, *Curve25519: new Diffie-Hellman speed records.* Pages 207228 in: Public key cryptographyPKC 2006, 9th international conference on theory and practice in public-key cryptography, New York, NY, USA, April 2426, 2006, proceedings, edited by Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin. Lecture Notes in Computer Science 3958, Springer, 2006. ISBN 3-540-33851-9. http://cr.yp.to/papers.html#curve25519

[8] Wikipedia, *Elliptic Curve Diffie-Hellman*, https://en.wikipedia.org/wiki/Elliptic_curve_Diffie-Hellman

[9] *Things that use Ed25519*, https://ianix.com/pub/ed25519-deployment.html.

[10] Bernstein, Daniel J.; Lange, Tanja, *SafeCurves: Introduction*, https://safecurves.cr.yp.to.

[11] Wikipedia, *RSA (cryptosystem)*, https://en.wikipedia.org/wiki/RSA_(cryptosystem)

[12] John Bradley, *The problem with OAuth for Authentication.*, http://www.thread-safe.com/2012/01/problem-with-oauth-for-authentication.html, January 2012.

[13] Tim Wilson, *Security Flaw Found In OAuth 2.0 And OpenID*, DARKReading, http://www.darkreading.com/security-flaw-found-in-oauth-20-and-openid-third-party-authentication\-at-risk/d/d-id/1235062, May, 2014.

[14] Antone Gonsalves, *Google flaw exposes weakness in two-factor authentication*, http://www.csoonline.com/article/2133038/access-control/google-flaw-exposes-weakness-in-two-factor-authentication.html, February 2013.

[15] Brian Krebs, *Attackers Hit Weak Spots in 2-Factor Authentication*, http://krebsonsecurity.com/2012/06/attackers-target-weak-spots-in\-2-factor-authentication/, June 2012.

[16] H. Krawczyk et al., *HMAC: Keyed-Hashing for Message Authentication*, IETF RFC 2014, Feb., 1997.

[17] Ben Cooper, SQRL - *An Illustrated Guide*, http://sqrl.pl/guide/.

[18] *Near Field Communication (*NFC*)*, http://nfc-forum.org/.

[19] Wikipedia, *One-time password*, https://en.wikipedia.org/wiki/One-time_password.