On Constructing a Trusted Path to the User

Mark Allman and Nicholas Weaver

TR-09-009

December 2009

Abstract

One of the fundamental activities within a network is authentication. Current schemes fail for a number of reasons, but crucially they are almost all vulnerable to end-host compromise and an inability to authenticate transactions. In this preliminary report we argue that a generic trusted path to the user is an essential building block for the future Internet architecture. We sketch the design of a "key fob" that will readily fit on peoples' physical key rings and provide such a trusted path from Internet services to users regardless of the state of the components of that path.

1 Introduction

Computer networks connect users with a rich array of valuable services. Key to the operation of these services is some way for the service to authenticate users and *transactions*. While the required strength of the authentication varies across applications, it is clear that some services involve sensitive activity (banking, e-commerce, etc.) and thus strongly established identity is crucial. Unfortunately, establishing strong identity and transactional authentication remains elusive.

Current authentication mechanisms largely utilize username and password pairs. However, passwords are inherently insecure because they can be readily stolen and used by attackers. Proper management of passwords can go a long way towards mitigating the fundamental risk associated with their use. However, as the number of services users access balloons the pain of constructing strong and unique passwords for each service and keeping those passwords safely offline increases dramatically. This leads users to coping strategies that expose the fundamental flaws of simple password schemes, such as (i) choosing simple passwords, (ii) sharing passwords across services and (iii) leveraging password caches on their computers to manage their large sets of credentials. In turn, each of these coping mechanisms can be leveraged by attackers to steal passwords (e.g., by compromising machines, setting up phishing schemes, brute force guessing, etc.). Once stolen there is little protection against fraudulent password use. A number of systems that move beyond username and password pairs have been proposed, including the following[1]:

Cryptography: Ubiquitous use of cryptography-based identities mitigates some of the problems with establishing identity. Users could simply register a public key with a service and retain a corresponding passphrase-protected private key. A key pair could be safely used across services and therefore the user would only need to deal with one passphrase, meaning that caching that passphrase in some database would not be necessary. Even if we extend to a few key pairs per user to decrease cross-service sharing the task is not onerous. A user could also use the same passphrase for numerous key pairs for various services—again, lessening the burden on the user. A final consideration is that cryptographic schemes can work well for relatively sophisticated users (e.g., *ssh*), but these systems have to be implemented in a way that unsophisticated users can readily utilize. There are also already several software keystores [9, 2] which manage these keys for different applications.

OpenID: This project [8] seeks to move beyond users grappling with a large number of credentials by providing a single independent credential that can be used across services[2]. Users will have to then manage only a single (or fewer, anyway) identities. This is useful for users and likely reduces the reliance on password caches.

[1] This is not a comprehensive list, but describes the general classes of solutions that have been proposed.

Pwdhash: This scheme tries to thwart phishing attacks by constructing site-specific passwords on the user's behalf [11]. The passwords are of the form $h(p, n)$, where $h()$ is a hash function (e.g., SHA-1), p is the user-entered password and n is the service's fully-qualified domain name (FQDN). In this way, the user can employ the same p relatively safely across services because authenticating does not reveal p to the service. Further, if a fraudulent service coaxes a user into providing a password the given password will not match the actual password for the legitimate service because the password will actually be $h(p, n')$ where n' is the FQDN of the fraudulent service and therefore the password will not work if replayed to the legitimate service.

Smartcards: These devices, including those built into USB devices (such as the Aladdin eToken [1]), provide a public key cryptographic resource. They can perform key operations when inserted in the computer. When a smartcard is connected to the host, the host can have unlimited *use* of the keys to perform signature and key exchange operations, but does not gain *access* to the keys.

SecureID: These fobs [13] provide an external one-time password. The token presents users with a new password periodically (e.g., every minute). The corresponding service understands the progression of passwords the token presents to the user and therefore accepts only the currently legitimate password. This system is highly secure in that it forces users to *possess* information that is external to the transaction path to authenticate. The downside is that these tokens are not generic and are tied to *a particular service* (which is usually highly sensitive). Therefore, it is not possible for the user to leverage a single SecureID token across the large variety of services they utilize, except perhaps in the context of some transitive-trust framework like OpenID.

All these previous systems are vulnerable to host compromise. These can include extracting keys from a keystore, capturing passwords (for OpenID or *pwdhash*) with a keylogger, a *credential usage* attack where a compromised host uses the keys in an attached smartcard, or *session hijacking* where an established session is hijacked for the attacker's use, even when perfectly secure user authentication is established [12]. Thus, for critical transactions, such as those involved in transferring money (or the equivalent [3]) we need additional authentication schemes that work in the face of end-host com-

[2] OpenID does not actually do *authentication*, but only *transitive authentication*. Once a user authenticates to one OpenID service they can be authenticated to others. How users are authenticated to the first service is service-dependent, although passwords are used in many cases.

promise. Apart from our proposal, there are three other systems with this goal.

Trusted Boot: A few users, such as one of the authors, simply assume that their system may be compromised when performing on-line transactions. Thus instead of using their normal operating system, they will boot from a clean, trusted CD, such as a new Ubuntu CD. Even some financial industry groups recommend that businesses use this technique [4], although it does impose a huge burden on users who wish to conduct an online transaction.

Independent Paths: Out-of-band communication with a user allows for an additional level of authentication. For instance, some banks in Europe have begun authenticating transactions via text messages to a customer's cell phone—which provides a message service that is independent from the Internet path over which the transaction is occurring [18]. Although effective for some applications, there are cost and convenience issues with this technique.

ZTIC: The IBM Zone Trusted Information Channel (ZTIC) [16] is a USB-based trusted path to the user, and is designed to authenticate transactions. The ZTIC has a small display, a push-button to acknowledge consent, and a smart-card connection. Although the theme is very similar to our intent, the implementation is very different. The ZTIC acts as a cryptographic endpoint thereby inserting itself into the data path for transactions and potentially becoming a performance bottleneck. Also the ZTIC requires site-specific information to understand the application being proxied, which means new applications will require modifications to the ZTIC. Finally, the ZTIC lacks a keystore, instead relying on an external smartcard.

Our position in this report is that a crucial missing piece within the Internet architecture is a general-purpose *trusted path to the user* for authentication of sessions and transactions, combined with the cryptographic key management in an easy-to-use design. Such a path must be trustworthy regardless of the state of the components of the path between the service and the fob itself.[3] We use the notions of (i) host-independence (a la the SecureID card), (ii) the genericness of using cryptography-based identification, (iii) the intuitiveness of a physical key (e.g., a house key), and (iv) the ubiquity of USB to propose a fob that users and services can leverage for direct communication between the user and a large number of arbitrary services. Such a fob would not only be enabled for new applications, but could work within existing application frameworks.

[3]Obviously these components can prevent communication, but our goal is that they should not be able to fraudulently authenticate.

2 The Necessity of Trusted Paths

With the growing use of online banking for both business and personal use, attackers have now focused considerable effort on stealing money through technical means. The primary focus has been on targeting end-user systems with increasingly sophisticated malcode, as a profitable criminal enterprise.

For example, some strains of bank attacking malcode will capture SecurID and similar one-time passwords [6], launder connections through the victim's computer [6], and act as a full man in the middle [7]. Attackers have even targeted two-agent control, where two parties in a company must approve a transaction [6]. The losses from these attacks are substantial, the most recent attack cost Duanesburg Central School District in upstate New York $500,000 [5], and individual small bulinesses have lost upwards of hundreds of thousands of dollars in single attacks [6].

This all points to a simple observation: conventional end hosts *can not be trusted*. Therefore going forward sensitive transactions (e.g., financial) must be conducted within a framework that *assumes that a typical user's end system is already compromised*, and therefore must not rely on it operating correctly to prevent fraudulent activity. Thus of the previous techniques, only **trusted boot**, **independent paths**, and **the ZTIC** can work in a secure manner.

3 Approach

Our goal in developing an identification and authorization system is to increase security without overburdening users with complicated new machinery, while being able to assume that the end host is compromised by an attacker. To this end we envision an independent key fob for authentication that meets the following goals.

First, people already have a wealth of experience with handling one particular identification token: their physical keys. From an early age people are taught the rules of handling their keys: anyone who has your key can unlock your door and copy your key, losing a key means not being able to unlock something, re-keying (e.g., doors) is expensive, duplicating a key allows a lock to be shared, etc. In addition, people are well attuned to not losing (rather than "misplacing") their keys. People know how to handle keys and so our goal is to develop a fob that is both literally and figuratively akin to these objects that people already understand.

Second, as noted in § 1, a general purpose device is susceptible to a variety of mischief, from watching a user's keystrokes to replicating information found on a disk to changing transactions as they occur. An independent purpose-built authentication device with limited capabilities and a narrow interface is much less vulnerable to such attacks. Further, because the device is independent and highly portable, identity can be readily established from arbitrary hosts.

Third, while information must pass through the network and a user's end host which could both potentially be compromised it is crucial to establish a trusted path to the *user*. By employing simple input and output on the device itself transactions can be authorized by the user regardless of the state of the information path.

Fourth, by only performing signatures, the device is not on the critical path for bulk traffic. This works because the amount of information that needs to be authenticated is a small fraction of the total traffic.

To achieve our goal of constructing a *trusted path to the user* we propose building a cryptographic store that would fit on peoples' physical key ring. Such a device would consist of a USB port, a speaker, a single push-button and a modest amount of storage space to hold a user's key pairs (and ancillary information). We provide a more in-depth discussion of the feasibility of the hardware in § 5. To use the key fob the user simply inserts it into an open USB port of their host machine. Services wishing to authenticate via the fob will interact with clients on the host machine that will in turn communicate with the fob via the API given in § 4. All messages from the service to the fob will be cryptographically signed by the service such that they cannot be tampered with in transit and so the fob can establish the identity of the ultimate requester.

The fob will be capable of holding keys of three *usage types*. The usage type is specified when the key is generated or imported.

Autonomous Keys merely verify the presence of the key fob. A service can request that a particular piece of data be signed by an autonomous key and the fob will comply. The fob will inform the user of the key's use through the speaker. Except for this latter feature, these keys are akin to those provided by USB smartcards (e.g., the eToken [1]), which resist key-capture attacks.

Prompted Keys verify the presence of both the key fob and a human. When using such a key, the fob first presents a request to the user ("please confirm the use of key X") and the activity is not conducted until the user presses the button. The prompt should be expected based on the context of whatever work the user is conducting. Prompted keys can transparently work within many existing protocols, such as *ssh* and client-side certificates for TLS. Prompted keys, by establishing user presence, also resist key-usage attacks from compromised hosts.

Service Prompted Keys require the service to supply an audio file that will be played via the fob's speaker, which is then signed if the user consents. For instance, the audio file might prompt "Push the button to authorize transfer of $50 from savings account 8372 owned by John Smith to checking account 2954 owned by Alice Jones". If the user agrees with the request they press the button which will then trigger the signing of the audio file with the given key. This key type offers a trusted path between the service and the user to verify a transaction, adding resistance to session capture attacks for transactions.

We also note that each local key pair held in the fob, k_l, can be configured at generation or import time be *service-scoped*—i.e., to track the services that have used k_l in the past. Any time a service makes a request (signed by the service's key k_s) to use a service-scoped k_l the fob consults k_l's service-scope list and if it does not contain k_s the fob will prompt the user to verify that they want to interact with this unknown requester. If the user agrees—e.g., because they are in the midst of a transaction with a new service—the transaction continues and k_s is added to k_l's service-scope list. This is akin to *ssh*'s host key cache and user prompt to verify an unknown host.

Our full vision is for service prompted keys to be the norm as they represent a secure path from the service to the user. The other two usage types are not as strong. In particular, autonomous keys that will sign any data for any requester should be avoided in the general case, but they have some uses (see § 8). The prompted usage type is envisioned mainly for backward compatibility for services that already use keys for various activities and therefore would benefit from the key store on the fob but are not yet savvy enough to send audio files to the fob.

For instance, consider *ssh*. Currently a user can activate an *ssh-agent* to cache their passphrase and provide authentication without interacting with the user. Optionally, this agent can be accessed remotely using existing *ssh* connections (via agent forwarding). This allows the credentials on some host A to be used not only to login to a host B, but also then to login to host C from host B. Of course, if B is compromised, it leaves the user vulnerable to a credential usage attack. If, instead, a prompted key is used in the fob, this improves security by limiting credential usage attacks by either A or B. And ideally, the *ssh* tools could be made savvy enough to use a service prompted key to get specific authorization to use a key (e.g., "push button to authorize login from host B to host C").

In addition, the fob will have an optional password to provide some protection against casual mis-use of a fob someone happens upon. If the user has set a password the user will have to enter their password on the host computer before prompted keys or service prompted keys will work, as these keys will be encrypted with the password. Autonomous keys will work regardless of password-based authentication (see § 8 for an example of the necessity of this approach). We note two problems with using a password in an attempt to protect the fob. First, the password is being entered into a possibly compromised host computer and therefore could be stolen. Second, simple passwords could be brute forced if someone has possession of the fob itself. Therefore, while strong fob passwords would aid security we do not expect that passwords will be of large benefit in the general case and, hence, consider them optional.

We note that our approach's heavy reliance on keeping the user in-the-loop via the speaker has downsides, including presenting difficulty for hearing impaired people, language issues, operating in noisy environments, as well as potential privacy concerns when used in public spaces. However, we believe these issues can be worked around (e.g., by providing a headphone jack for use in public spaces or allowing users configure their language in their profile on a service's web site). We also do not believe that audio is fundamental to the security provided by the fob. A small screen with service-signed textual messages would provide the same level of trustworthiness to the authentication process [16]. There are clear cost and usability tradeoffs between a speaker and a display. While these usability issues will need to be addressed in some fashion we consider them out of scope for this initial sketch of our design.

A final usability note is that practical issues like repeating prompts or dealing with multiple simultaneous prompts will need to be addressed. We do not further explore such problems in this report as our initial goal is to develop a strong trusted path before delving into the nitty-gritty practical details. However, while we defer usability issues initially we do believe that the device we propose passes the smell test (e.g., relative to the criteria developed in [10]) for being a plausibly attractive mechanism for users and services.

4 API

Here we specify the fob's API in general terms. Experience may dictate that this set of routines be changed and we elide small details, but we believe the given API captures the fundamental operation of the fob.

CheckPassword (*p*) This routine is used to validate a password p that has been entered on the attached host, which will in turn allow the fob's full capabilities to be used. The fob will announce each password check as this would serve to warn users of password guessing or brute force cracking attempts. Further, the fob will block all activity after a small number of failed attempts (until the fob is removed and reinserted).

SetPassword (*p_{old}, p_{new}*) This routine changes the fob's password from p_{old} to p_{new}. The fob will use the built-in speaker to ask the user for permission to change the password to protect against illegitimate password changes.

GenerateKey (*t, s*) This routine will generate a new key pair of usage type t (as discussed in the previous section) and return the public key. The s parameter is a boolean that denotes whether the new key should be service-scoped. *GenerateKey()* is the most secure way to populate the fob with a key pair because the private half of the key is never stored outside the fob. The generation process will be confirmed by the user.

ImportKey (*k_s, k_p, t, s*) This routine will import the given key pair (k_s,k_p) and consider the key to be of type t. The s parameter is a boolean that denotes whether the imported key should be service-scoped. This routine is useful for migrating identities that have been created elsewhere to the fob. However, this method of populating the fob's key store is not as secure as generating keys on the fob itself because the private half of the imported key was previously stored at some other (potentially insecure) location. The importing process will be confirmed by the user.

Sign (*d, k_{id}*) This routine returns the signature of the given data d created by using the fob's key k_{id}. For an autonomous key, this proceeds normally with the fob announcing the action. For a prompted key this routine will ask the user to authorize the use of k_{id} before returning the signed data to the caller. For a service prompted key this routine will fail (see next routine). If the key is configured to be service-scoped and the requester has not previously been authorized, the user will be asked to authorize the interaction with the requester before the data is signed and returned.

SignAudio (*a, k_{id}*) This routine is similar to the *Sign()* call, but rather than arbitrary data an audio track a is provided. The audio is played for the user and the user then authorizes the transaction by pressing the button, resulting in a signature of the audio file. This is the only signing routine available for service prompted keys.[4]

SessionKey (*c, k_{id}*) This routine will generate a session key with the specified server certificate c and locally stored key k_{id} that is suitable for for SSL/TLS [15] or other appropriate authenticated key-exchange protocols. As with the *Sign()* and *SignAudio()* routines this operation can be service-scoped depending on the configuration of k_{id}.

GetPublicKeyList () This routine returns a subset of the public keys stored on the fob. In particular the list consists of all non-service-scoped keys and any keys within whose service-scope the requester falls. Further, if the user has set a password and the password has not yet been verified only autonomous keys are included in the response to this call.

GetPublicKey (*k_{id}*) This routine returns the public half of k_{id}. If no key is given then a default key will be returned. This will allow new services to easily bootstrap.

[4]Note this requirement will necessarily be relaxed in the real fob to accommodate displays as suggested above.

It should be noted that the fob's primary objective is *integrity*, not *confidentiality*. However, the *Sign()* routine is data type agnostic and can therefore deal with encrypted data. Further, the *SignAudio()* will accept encrypted, as well as unencrypted, audio tracks. While perhaps not of great import in many cases, this functionality can be used to protect information between the service and the user (e.g., so that details of a transaction such as an account number cannot be found by eavesdroppers along the path).

5 Feasibility

Our vision is that for security to be dramatically increased, wide-scale use of the fob is necessary—both to protect users and to incentivize services to utilize the mechanism. Hence, one obstacle to the feasibility of the fob is economics: a fob must be inexpensive to produce in large quantities or the device will not be adopted. Although we have not yet produced a fob, a preliminary evaluation suggests that the bill of materials could be kept under \$30. Beyond the simple packaging and circuit board, the device requires a small speaker, an op-amp to drive the speaker, approximately 256 MB of flash storage (sufficient for thousands of keys), 5 V to 3.3 V power conversion, a small noise circuit for a physical RNG, and a CPU with sufficient processing power (such as the 66MHz ST Microsystems STR710 [14]) to perform cryptographic operations and implement a Delta-Sigma D-A to drive the speaker [17]. The total cost is \$15 in single-unit quantities. Assuming another \$15 for assembly and a plastic case, it seems reasonable to construct the fob for \$30, enabling a \$60 retail price.

6 Additional Considerations

In this section we briefly touch on a number of additional issues that have come come up during our high-level design.

Tamper Resistance: Our vision prefers easy duplication to tamper resistance. Tamper resistance to any high degree of certainty would require substantial changes to the fob's CPU and would likely drastically increase the cost. On the other hand, allowing for relatively easy duplication matches peoples' expectation with their physical keys, allowing them to store a backup in a safe place: simply open the case and attach an internal connector to copy the contents of one fob to another.[5] This expectation could be reinforced with a notice on the fob itself.

Cell Phone Integration: Rather than constructing a new device for the fob functionality an alternate approach would be to integrate the fob with a cell phone. This is tempting because such a combination would not require people to carry another device.[6] However, modern cell phones are not simple devices, but rather approach the complexity of general computing platforms. Therefore, securely implementing the fob's functionality in cell phone software will be difficult at best and will likely fall into the familiar traps end host software falls into (as discussed in § 1). This could potentially impinge on the fob's ability to securely reach the end user. If the fob and cell phone were kept distinct devices and both simply housed in the cell phone's case this would allay the security concerns and might be attractive to users.

Bluetooth: In addition to using USB the fob could be connected to hosts using bluetooth. This capability might be crucial for devices that do not have USB ports (e.g., smartphones). While we do not consider the detailed implications of using bluetooth in this initial exploration we note that exposure to malicious bluetooth peers is not dramatically different from plugging the fob into a compromised host.

Multi-Stage Authentication: We also note that the fob can be used to provide only partial authentication. That is, a user may still be required to know a password for two-factor authentication. The optional password check in the fob itself *must not* be considered a complete two-factor authentication scheme. In fact, the service will not even know whether the fob has a password set or not.

7 Attacks

We now briefly discuss several avenues for attacking key stores and how our fob design mitigates these concerns.

Capturing Private Keys: Since private keys never leave the fob[7] a host or network compromise should never be able to acquire a private key stored in the fob.

Denial-Of-Service: Elements on the transaction path can deny the user service by not forwarding messages to the fob. Alternatively, the fob is susceptible to DoS attacks that simply overwhelm the fob's computational capabilities by requesting a large number of signatures. While no doubt annoying to users neither of these attacks leads to fraudulent use of credentials. Further, the fob's announcements will prevent a silent DoS whereby the user is left wondering why legitimate operations are not working or are slow.

Credential Usage Attack: The fob is specifically designed to resist credential usage attacks present in other cryptographic keystores. First, *all* accesses are exposed to the user, even autonomous keys can't be silently used. Second, and more importantly, much of the use of the fob *requires* user presence and authorization.

Session Hijacking Attack: If the attacker controls the user's end host, they can hijack a session after it has been

[5] Even without such a feature, if an attacker can open the fob, the flash can be read directly, so an internal "copy port" benefits legitimate users without significantly helping attackers.

[6] Arguably since people generally carry their keys and we are aiming for a form factor that would fit on their key ring, carrying a fob is not overly onerous.

[7] At least for keys that are generated by the fob—which is the recommended usage scenario.

established. The fob cannot protect against this attack for autonomous or prompted keys, but authenticating on the transaction level with service prompted keys limits the damage that can be done by session hijacking.

Sidechannel Attacks from the USB: There are two classes of side-channel attacks which can be attempted. The first class is through an *unmodified* USB port, where a corrupted host can attempt a timing or similar attack. The fob should be designed to resist such attacks. The second class arise from *modified* USB ports, which could perform power or power-glitching attacks. We do not expect to be able to defend against such attacks with off-the-shelf hardware components.

Physical Capture: The greatest aspect of resisting physical attacks is the placement of the fob on peoples' key rings. Users already have a model of physical keys which matches the fob's access model and therefore we expect the fob to be reasonably controlled by users. Additionally, if the user can use a *strong* password, the password can protect prompted and service prompted keys. However, we note that using a weak password affords no protection from brute force attacks on a stolen fob and could perhaps give the user a false sense of security.

Social Engineering Attacks: Many attacks on the Internet (and elsewhere) are based on social engineering. Although the fob can't mitigate all social engineering attacks, the use of service prompted keys may limit some attacks. For example, the attacker would need to convince the user that a transaction spoken by the fob (or perhaps the computer's speaker if lucky) represents a legitimate action the user wishes to commit.

Resistance to attack also requires due diligence on the part of service providers. For example, the audio prompts should not be general ("authorize transfer from savings account to checking account"), but should include information about the particular transaction ("authorize transfer of $50 from Nick Weaver's checking account to Mark Allman's checking account"). Further, the messages should contain a nonce such that the second author cannot replay messages to pad his income.

8 Extensions

In addition to the functionality described above there are several additional jobs we could task the fob to perform. However, we wish to keep functionality tightly constrained, so possible extensions are tightly related to the fob's primary functionality. First, the key fob could be used to navigate physical locks (e.g., for a room), for which autonomous keys are a natural fit. Second, the fob could ease key exchange by simply inserting the fob into a foreign computer (e.g., at a bank branch) and allowing that computer to retrieve one's public key. Finally, we envision that the storage space on the fob could also provide "thumb drive" functionality for small pieces of software that would run on the connected hosts and facilitate the use of the fob (e.g., a fob-savvy *ssh-agent* or Firefox TLS extension). This could allow ready use of the fob even before host-based software is updated to interact with the fob. These additional topics will be considered in more detail in future work.

9 Summary

Being able to construct secure network services ultimately requires a service to be able to soundly authenticate users and transactions. The current method of authenticating transactions relies on an uncompromised path between the service and the user. However, time and again we have seen that attackers compromise crucial elements of the path allowing them to easily steal users' credentials and employ these for fraudulent activities. Our proposed approach to this problem is to construct a cryptographic key store on a small USB device that fits on peoples' key rings. Further, this device will have simple input and output capabilities such that requests can be validated and authorized without relying on any element of the path between the service and the key store—and in fact will even work in the face of compromised elements. Yet at the same time, such a device could integrate into existing cryptographic protocols, including *ssh* and SSL/TLS, granting a more secure foundation even in the presence of compromised end-hosts.

We believe that a small independent device is both feasible and the best path for building a crucial and foundational element of future Internet systems. However, we are also interested in engaging the community in discussions about how to secure information in the face of inevitably compromised elements in such a way that users can both get their work done and have confidence that their information is not being used fraudulently.

Acknowledgments

This report benefits from comments on previous versions by Ethan Blanton and Tom Callahan. We are also grateful for fruitful discussions with Misha Rabinovich. This work was funded in part by NSF grants CNS-0433702 and CNS-0831780.

References

[1] Alladin Inc eToken Strong Authentication and password management, http://www.aladdin.com/etoken/default.aspx.

[2] Keychain Services Programming Guide. `http://developer.apple.com/documentation/Security/Conceptual/keychainServConcepts/01introduction/introduction.html`.

[3] GOODIN, D. Wow players learn value of windows updates. `http://www.theregister.co.uk/2007/04/10/wow\_hijackings/`.

[4] KREBS, B. Avoid Windows Malware: Bank on a Live CD, `http://voices.`

washingtonpost.com/securityfix/2009/10/avoid_windows_malware_bank_on.html.

[5] KREBS, B. Fbi investigating theft of $500,000 from ny schoo district, http://www.krebsonsecurity.com/2010/01/fbi-investigating-theft-of-500000-from-ny-school-district/.

[6] KREBS, B. Story driven resume, my best work, http://www.krebsonsecurity.com/2009/12/story-driven-resume-my-best-work-2005-2009-3/.

[7] MUNCHU, L. Banking in silence, http://www.symantec.com/connect/blogs/banking-silence.

[8] OpenID. http://openid.net/.

[9] Openssh, http://www.openssh.com/.

[10] PIAZZALUNGA, U., SALVANESCHI, P., AND COFFETTI, P. The Usability of Security Devices. In *Security and Usability*. 2005, ch. 12.

[11] ROSS, B., JACKSON, C., MIYAKE, N., BONEH, D., AND MITCHELL, J. Stronger Password Authentication Using Browser Extensions. In *Usenix Security Symposium* (2005).

[12] RSA Alert: New Universal Man-in-the-Middle Phishing Kit Discovered. http://www.rsa.com/press_release.aspx?id=7667.

[13] The RSA SecurID. http://www.rsa.com/node.aspx?id=1156.

[14] ST Microsystems STR7 ARM 32 bit Microcontrollers. http://www.st.com/mcu/inchtml-pages-str7.html.

[15] T. DIERKS, E. R. The Transport Layer Security (TLS) Protocol Version 1.1, Apr. 2006. RFC 4346.

[16] WEIGOLD, T., KRAMP, T., HERMANN, R., HORING, F., BUHLER, P., AND BAENTSCH, M. The zurich trusted information channel - an efficient defense against man-in-the-middle and malicious software attacks. In *TRUST 2008* (2008).

[17] Delta-Sigma Modulation. http://en.wikipedia.org/wiki/Delta-sigma_modulation.

[18] Wikipedia, SMS banking, http://en.wikipedia.org/wiki/SMS_banking.