Security Keys: Practical Cryptographic Second Factors for the Modern Web

Juan Lang, Alexei Czeskis, Dirk Balfanz, Marius Schilder, and Sampath Srinivas

Google, Inc., Mountain View, CA, USA

Abstract. "Security Keys" are second-factor devices that protect users against phishing and man-in-the-middle attacks. Users carry a single device and can self-register it with any online service that supports the protocol. The devices are simple to implement and deploy, simple to use, privacy preserving, and secure against strong attackers. We have shipped support for Security Keys in the Chrome web browser and in Google's online services. We show that Security Keys lead to both an increased level of security and user satisfaction by analyzing a two year deployment which began within Google and has extended to our consumer-facing web applications. The Security Key design has been standardized by the FIDO Alliance, an organization with more than 250 member companies spanning the industry. Currently, Security Keys have been deployed by Google, Dropbox, and GitHub. An updated and extended tech report is available at `https://github.com/google/u2f-ref-code/docs/SecurityKeys_TechReport.pdf`.

1 Introduction

Recent account takeovers [1–3] have once again highlighted the challenge of securing user data online: accounts are often protected by no more than a weak password [4] and whatever implicit signals (if any) that the online service provider has collected to distinguish legitimate users from account hijackers.

Academic research has produced numerous proposals to move away from passwords, but in practice such efforts have largely been unsuccessful [5, 6]. Instead, many service providers augment password-based authentication with a second factor in the form of a one-time passcode (OTP), *e.g.*, [7, 8]. Unfortunately, OTPs as a second factor are still vulnerable to relatively common attacks such as phishing [9]. In addition, OTPs have a number of usability drawbacks (see Section 2). These factors limit the success and deployment of OTPs as a reliable and secure second factor.

Meanwhile, secure authentication factors, which use challenge/response-based cryptographic protocols, have their own barriers to deployment. National ID cards [10, 11] and smart cards require custom reader hardware and/or driver software to be installed prior to use. Depending on the implementation, these systems also make it challenging for users to protect their privacy (see Section 2).

In this work, we present Security Keys: second factor devices that improve the state of the art for *practical authentication for real consumers* in terms of privacy, security, and usability. This is evidenced by the fact that Security Keys have been publicly deployed by Google [12], Dropbox [13], and GitHub [14]. Security Keys were designed from the ground up to be practical: simple to implement and deploy, straightforward to use, privacy preserving, and secure against strong attackers. We have shipped support for Security Keys in the Chrome browser, have deployed it within Google's internal sign-in system, and have enabled Security Keys as an available second factor in Google's web services. *In this work, we demonstrate that Security Keys lead to both an increased level of security and user satisfaction as well as cheaper support cost.* The Security Key design has been standardized within an industry alliance, the FIDO Alliance [15] as the Universal Second Factor (U2F) protocol.

2 Related Work

We now give an overview of the most relevant related work. For detailed background of the field, please consult a variety of excellent survey works [5,6,16,17].

One-Time Passcodes. OTPs are short (typically 6 to 8 digit) codes that are one time use and are sent to the user via SMS or are generated by a separate physical dongle. Though they provide more security than passwords, OTPs have a number of downsides. First, they are vulnerable to phishing and man-in-the-middle attacks [9]. Second, OTPs that are delivered by phones are subject to data and phone availability, while those that are generated by dongles cause the user to have one dongle per web site. Finally, OTPs provide a sub-optimal user experience as they often require the user to manually copy codes from one device to another. *Security Keys are resistant to phishing and man-in-the-middle by design; our preliminary study also shows that they provide a better user experience.*

Smartphone as Second Factor. A number of efforts have attempted to leverage the user's phone as a cryptographic second factor, both within academia (*e.g.*, [18]) and in industry (*e.g.*, [19]). While promising, they face a number of challenges: for example, protecting application logic from malware is difficult on a general purpose computing platform. Moreover, a user's phone may not always be reachable: the phone may not have a data connection or the battery may have run out. *Security keys require no batteries and usually have a dedicated tamper-proof secure element.*

Smart Cards. Security Keys fit into the "what you have" category of authentication schemes and have a close relationship to smart cards. While Security Keys can be (and have been) implemented on top of a smart card platform such as JavaCard [20], *Security Keys express a particular protocol for which smart cards are just one possible implementation platform.*

TLS Client Certificates. TLS Client Certificates [21] traditionally bear the user's identity and can be used for on-line authentication. When users navi-

gate to a web service that requires TLS client authentication, their browser will prompt the user for their client certificate. Unfortunately, current implementations of TLS client certificates have a poor user experience. Typically, when web servers request that browsers generate a TLS client certificate, browsers display a dialog where the user must choose the certificate cipher and key length—a cryptographic detail that is unfamiliar and confusing to most users.

When web servers request that the browser provide a certificate, the user is prompted to select the client certificate to use; accidentally choosing the wrong certificate will cause the user's identity to leak across sites. In addition, because client certificates are transmitted in the clear, the user's identity is revealed during TLS client certificate transfer to any network adversary. TLS client certificates also suffer from a lack of portability: they are tough to move from one client platform to another. *Security Keys have none of these issues: as we will describe, they are designed to be fool-proof: with users' privacy in mind, to be simple to use, and to be portable.*

Electronic National Identification Cards. Some countries have deployed national electronic identification cards. In Estonia, for example, electronic ID cards "are used in health care, electronic banking and shopping, to sign contracts and encrypt e-mail, as tram tickets, and much more besides—even to vote" [11]. Despite their rich capabilities, national identity cards have not become a popular global online authentication mechanism [10]. One possible reason is that current national identity cards require special hardware (a card reader) and thus are hard to deploy. Another possible reason is that national identity cards are by definition controlled by one government, which may not be acceptable to businesses in another country. *Security Keys have no such downsides: they work with pre-installed drivers over commonly available physical media (USB, NFC, Bluetooth) and are not controlled or distributed by any single entity.*

Finally, some approaches can combine multiple elements, *e.g.*, some electronic ID cards combine smart card, TLS client certificate, and government identification.

3 Threat Model

We briefly outline major attackers and attacks that we consider in our design.

3.1 Attackers

Web Attackers. Entities who control malicious websites (*e.g.*, `bad.com`[1]) are called *web attackers*. We believe that virtually all users might accidentally visit such a malicious site. The attacker may design `bad.com` to visually mimic a user's bank or e-mail website in an attempt to get (*i.e.*, *phish*) the user's login credentials. As the attackers are quite skilled, and may occupy clever URLs (such as `bamk.com`[2]), we assume that most users will fall for this trick. That is,

[1] This is just an example, the real `bad.com` may not be malicious.
[2] This is just an example, the real `bamk.com` may not be malicious.

they will enter credentials such as their password into the attacker-controlled `bad.com`.

Related-Site Attackers. Some attackers will compromise sites having weak security practices in order to steal the site's user credentials. As users often reuse credentials across sites [22], *related-site attackers* will reuse the stolen credentials on more secure sites in hopes of accessing the user's accounts [23].

Network Attackers. Adversaries may be able to observe and modify network traffic between the user and the legitimate site. We call such adversaries *network attackers*. For example, a network attacker might sniff wireless traffic in a coffee shop [24, 25]) or a nation-state may interpose on all traffic that traverses their physical borders [26, 27]. A man-in-the-middle attacker can defeat the security properties offered by TLS [21], *e.g.*, by forging rogue server TLS certificates [28]. Other network attackers may record traffic and subsequently exploit the TLS cryptographic layer [29, 30] to extract authentication data.

Malware Attackers. In some cases, attackers may be able to silently install and run arbitrary software on users' computers; such attackers are *malware attackers*. We assume that such software can run with arbitrary privileges and can freely help itself to cookies, passwords, and any other authentication material.

3.2 Attack Consequences

We highlight two of the most concerning attack consequences below.

Session Duplication. In some cases, an attacker may be able to steal credentials that allow him/her to access the user's account from any computer and at any time. For example, if an attacker is able to steal cookies or passwords, then he/she may be able to log into the victim's account at virtually any subsequent date (assuming the password isn't changed, and the cookie doesn't expire).

Session Riding. If an attacker can only access or modify the user's account when the user is actively using his/her computer, we call this attack *session riding*. For example, if the website always requires the user to begin a new session by providing proof of a hardware device that the attacker does not control, the attacker is only able to *ride* the user's active sessions.

Security Keys make session duplication and riding much more difficult.

4 System Design

We now give a system overview; for details please consult the official specification on the FIDO Alliance website [15]. In juggling the various requirements, we settled on the following design goals:

- *Easy for Users*: Using Security Keys should be fast, easy, and "brainless". It must be difficult to use Security Keys incorrectly or insecurely.

– *Easy for Developers*: Security Keys must be easy for developers to integrate into their website through simple APIs.
– *Privacy*: Security Keys should not allow tracking of any kind. In addition, if a Security Key is lost, it should be difficult for an attacker to get any useful information from a Security Key.
– *Security*: Security Keys should protect users against password reuse, phishing, and man-in-the-middle attacks.

4.1 System Overview

Security Keys are intended to be used in the context of a web application in which the server wishes to verify the user's identity. At a high level, Security Keys support the following commands which are provided to web pages as browser APIs (see Section 5).

– *Register*: Given this command, the Security Key generates a fresh asymmetric key pair and returns the public key. The server associates this public key with a user account.
– *Authenticate*: Given this command, the Security Key tests for user presence and exercises its private key to provide a response. The server can verify that the response is valid, and thus authenticate the user.

Figure 1 shows two different Security Keys manufactured by Yubico—one of the several vendors who produce Security Keys. Each devices communicates over a USB interface and has a capacitive touch sensor which must be touched by the user in order to authorize any operation (register or authenticate). Both devices contain a tamper-proof secure element.

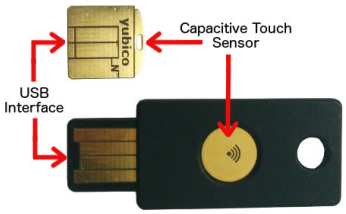

Fig. 1: Two Security Keys. Both have a USB interface and a capacitive touch sensor. One also has an NFC interface.

4.2 Detailed Design

We now focus on the details of both registration and authentication. Full specifications can be found on the FIDO Alliance website [15].

Registration. During registration (see Figure 2), the relying party—the server—produces a random challenge. The user's browser binds the server's challenge into a Client Data structure, to be covered shortly. The browser sends the server's web origin and a hash of the Client Data to the Security Key. In response, the

Security Key generates a new key pair along with a key handle, which will also be covered later. The Security Key associates the key pair with the relying party's web origin and then returns the generated public key, key handle, an attestation certificate, and a signature over: 1. the web origin, 2. hash of the client data, 3. public key, and 4. key handle. The web browser then forwards this data, along with the client data, back to the website. The website verifies the signature and associates the public key and key handle with the user's account.

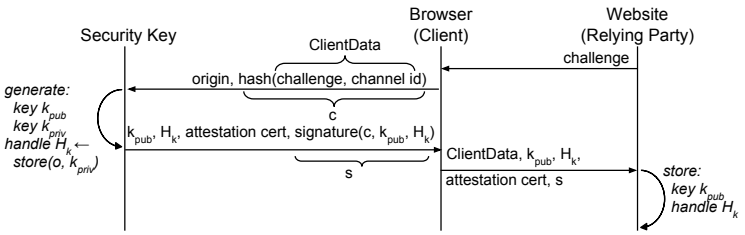

Fig. 2: Security Key registration.

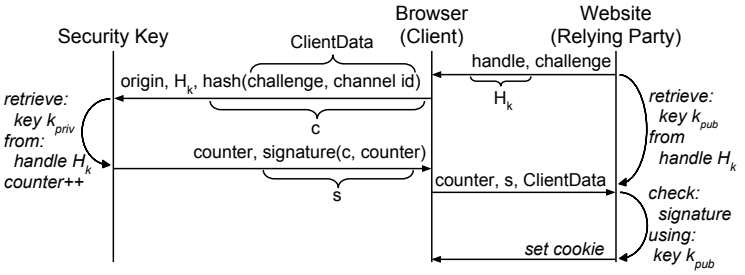

Fig. 3: Security Key authentication.

Authentication. During authentication (see Figure 3), the relying party requests that the Security Key exercise a particular key which has previously been registered for a user account. Specifically, the relying party sends the desired key's handle and a challenge to the web browser. The browser generates the client data (see above) and sends the hash of the client data along with the key handle and the web origin to the Security Key. If the Security Key does not recognize the key handle, or doesn't agree that it is associated with the web origin that requested the signature, it rejects the request. Otherwise, it produces a signature of the client data. The Security Key signs two additional attributes: whether a Test of User Presence (TUP) succeeded, and a counter value. The Test of User Presence is described in more detail below. The counter value is a 32-bit counter that is incremented with every signature the Security Key performs; its presence allows the server to detect potential cloning of a Security Key, *e.g.*, when the counter value appears to decrease from one signature to the next. The counter is described in more detail in Section 7.2.

The browser passes the signature, along with the TUP and the counter value, to the server. The server then checks the signature against the public key it has registered and authenticates the user if the signature matches.

Device Attestation. Each Security Key must provide an attestation certificate during registration. This allows servers to gate the use of a particular security key (for example, if servers trust only certain Security Key suppliers). A related desire is revocability: if a device or model is known to have flaws or have been compromised, a server might wish to not accept it.

Individually identifying devices would reveal a unique identifier for a device across unrelated origins, violating the user's privacy. To achieve both security and privacy, we recommended that devices implement batch attestation: a batch of devices shares a single attestation key (and certificate), such that all devices with a known flaw can be revoked together, and users' privacy is still respected: at worst, a device can be identified as a member of a batch. Alternatives for device attestation are explored further in Section 7.1.

Client Data. The client data binds the server-provided challenge to the browser's view of its connection to the server. Specifically, the client data includes the type of the request (register or authenticate), the challenge, and, when possible, the TLS channel ID [31, 32] of the connection. Binding the TLS channel ID allows the server to detect the presence of a TLS Man in the Middle. When a server receives a signed TLS channel ID, it can compare it with the TLS channel ID it observes in the TLS layer. If they differ, the server will be aware of the presence of a TLS Man in the Middle, and can abort the connection.

Test of User Presence. The Test of User Presence (TUP) allows the caller to test whether a human is present during command execution. This serves two purposes: first, it provides a mechanism for human confirmation of commands. Second, it allows web applications to implement a policy based on that check, e.g. "Transactions for a dollar amount greater than $1,000 require confirmation," or "Credentials must be re-presented by a human being after 90 days."

TUP implementation is left up to the device manufacturer. One vendor uses a capacitive touch sensor, others employ a mechanical button, while another makes a device that stays powered up only a short time after insertion into a USB port, requiring the user to reinsert the device for every operation.

Cryptographic Primitives. For all signing operations, we chose ECDSA over the NIST P-256 curve. For all hashing operations, we chose SHA-256. The choice of the curve and hash algorithm was made because of their wide availability on embedded platforms. At this time we believe these primitives, which offer 128 bit security, to be sufficiently secure.

5 Implementation

We have implemented end-to-end support for Security Keys. This involved building a large number of components; we describe some of them below (others were

omitted because of space). Note that all of the components have been open-sourced, are actively maintained, and can be found at `https://github.com/google/u2f-ref-code`.

5.1 Browser Support

We have implemented and shipped support for Security Keys as part of the Chrome web browser (available since version 41.) The browser support consists of JavaScript APIs that can be called by any web application. In total, the support consists of roughly 8,000 lines of code.

Register Method. A web server requests a new Security Key registration by making use of a new browser API:

```
u2f.register()
```

This API accepts a challenge and a list of already-registered key handles. The list of already-registered key handles allows the browser to avoid double registration of the same Security Key. If the browser finds an eligible Security Key, the client sends a *register* command to the Security Key, described in more detail in the next section. Upon successful completion of the register command, the browser sends the Security Key's Registration Message output, along with the client data described in Section 4.2, to the server. The server verifies the registration signature and that the client data matches its own view of the request. Finally, the server can check the attestation certificate to verify that it meets the server's requirements. Assuming all parameters match and are found acceptable, the server stores the key handle H_k and the public key k_{pub} for the user's account.

Sign Method. A web server requests a signature from a Security Key by making use of a new browser API:

```
u2f.sign()
```

The parameters to the sign API are a challenge, and any registered key handles for the user. The browser then searches for available Security Keys. For each device found, the browser sends a *sign* command, described in more detail in the next section. Upon successful completion of a sign command, the browser provides the signature to the server, along with the client data (Section 4.2) and the key handle H_k that produced the signature. The server checks that the signature verifies with the public key k_{pub} it has stored for H_k, and that the counter value has increased. The server also verifies the client data against its own view of the request. If all the checks succeed, the user is authenticated.

5.2 Security Key Token Implementation

We developed a JavaCard-based implementation of a Security Key. The underlying applet consists of approximately 1,500 lines of Java code. In addition, as

we describe shortly, we use traditional cryptographic techniques to support an arbitrary number of keys and origins given the limited storage capabilities of embedded platforms.

Security Keys support two basic operations: *register*, to create a new key pair, and *sign*, to produce a cryptographic signature.

Register Operation. The *register* operation takes two parameters, an application parameter—the web origin provided by the browser—and a challenge. The Security Key generates a new key pair (k_{pub}, k_{priv}) for the application parameter. It then performs a *store* operation to store both the application parameter and the private key k_{priv}. The *store* operation yields a key handle H_k and is discussed more fully shortly. The public key k_{pub} and the key handle H_k, along with the challenge parameter and application parameter, are then signed with the device's attestation private key, $Priv_{attest}$. The Security Key provides as output a Registration Message (see Figure 4), which includes the public key k_{pub} and the key handle H_k, as well as the device attestation certificate.

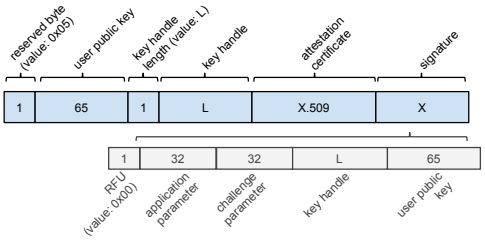

Fig. 4: Security Key Registration Message.

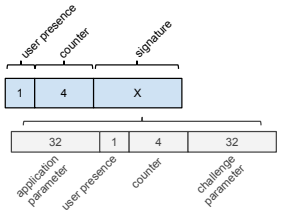

Fig. 5: Security Key Authentication Message.

Sign Operation. The *sign* operation takes three parameters, an application parameter, a key handle, and a challenge. During authentication, the Security Key first performs a *retrieve* operation to retrieve the stored application parameter and the private key for the key handle. If the Security Key does not recognize the key handle, it rejects the operation. Similarly, if it recognizes the key handle but the retrieved application parameter does not match the application parameter to the sign operation, it rejects the operation with the same error value as if it did not recognize the key handle. In this way, a web site that tries to make use of a key handle that was registered on a different origin cannot learn that the key handle is valid for the other origin. The *retrieve* operation is discussed further shortly.

Once the Security Key has retrieved a valid private key k_{priv} for the key handle, and verified that it was generated for the supplied application parameter, it increments the counter value and signs a concatenation of the Test of User Presence indication, the new counter value, and the challenge parameter. It provides the signature s to the browser, along with the Test of User Presence indication and the new counter value. The result of the sign operation is the Authentication Message shown in Figure 5.

5.3 Store and Retrieve Operations

Store and *retrieve* can be thought of database operations: storing a private key yields an index into a table, and this index is returned as the key handle. Retrieving a key handle looks up the value in the table at the given index.

However, a database-like implementation reduces privacy and usability: a predictable index for key handles reveals the number of accounts a Security Key is being used with. Additionally, the users need to be aware of the storage capacity of Security Keys to ensure she doesn't run out of space in the key database.

In our reference implementation, *store* is instead implemented as a key wrapping operation: the private key k_{priv} and the application parameter are encrypted using a secret key K_{wrap} known only to a single Security Key. By implementing *store* and *retrieve* as key wrap/unwrap operations, the Security Key reference implementations can store an unlimited number of key handles: the storage is implemented by the server.

In order to avoid known plaintext attacks, our reference implementations obscure the application parameter with a second secret key, K_{app}, also known only to a single Security Key. The obscured application parameter and the private key are then interleaved together, prior to encrypting with the wrapping key. In the reference implementation, two-key 3DES was chosen as the cipher, due to the quality of the implementation on the hardware we used. Algorithmically, our key wrapping implementation is:

function STORE(k_{priv}, app)
  $app' \leftarrow Encrypt(app)_{K_{app}}$
  plaintext $\leftarrow Interleave(k_{priv}, app')$
  $H_K \leftarrow Encrypt(plaintext)_{K_{wrap}}$
  return H_K
end function

function RETRIEVE(H_K, app)
  $app' \leftarrow Encrypt(app)_{K_{app}}$
  plaintext $\leftarrow Decrypt(H_K)_{K_{wrap}}$
  $(k_{priv}, app'') \leftarrow Deinterleave$(plaintext)
  constant-time check($app' == app''$)
  return k_{priv}
end function

It should be noted that key wrapping is an optional optimization vendors may employ, and that our implementation is one approach. The FIDO U2F specifications allow device manufacturers to choose any approach for key wrapping. For example, one vendor's approach is described in [33].

5.4 Server Implementation

We implemented, open-sourced, and actively maintain a reference Security Key server. It runs on the Google App Engine platform and consists of approximately 2,000 lines of Java code.

6 Evaluation

We evaluate Security Keys using a number of metrics. We begin by comparing the usability, deployability, and security of Security Keys to existing and prior

technologies. Next, we discuss the performance of various Security Key hardware devices. Finally, we give an in-depth analysis of our deployment experience including effect on users, support cost, and other relevant variables.

6.1 Comparative

Table 1: Comparative evaluation of Security Keys to similar schemes

		Usability							Deployability						Security										
Category	Scheme	*Memorywise-Effortless*	*Scalable-for-Users*	*Nothing-to-Carry*	*Physically-Effortless*	*Easy-to-Learn*	*Efficient-to-Use*	*Infrequent-Errors*	*Easy-Recovery-from-Loss*	*Accessible*	*Negligible-Cost-per-User*	*Server-Compatible*	*Browser-Compatible*	*Mature*	*Non-Proprietary*	*Resilient-to-Physical-Observation*	*Resilient-to-Targeted-Impersonation*	*Resilient-to-Throttled-Guessing*	*Resilient-to-Unthrottled-Guessing*	*Resilient-to-Internal-Observation*	*Resilient-to-Leaks-from-Other-Verifiers*	*Resilient-to-Phishing*	*Resilient-to-Theft*	*No-Trusted-Third-Party*	*Requiring-Explicit-Consent*	*Unlinkable*
(incumbent)	Web passwords			●		●	●	○	●	●	●	●	●	●	●		○						●	●	●	●
Hardware tokens	Security Keys	○	○			●	●	●		●	○		○	●	●	●	●	●	●	●	●	●	●	●		●
	RSA SecurID					●	○	○					●	●		●	●	●	●	●	●		●		●	●
	YubiKey					●	○	○		●			●	●		●	●	●	●	●	●		●		●	●
Phone-based	OTP over SMS	●	●	○		●		○	○	○			●	●	●	●	●	●	●	●	●		○		●	●
	Google 2-Step			○		●	○	○	○	○			●	●		○	○	●	●		●		●	●	●	●

● = offers the benefit; ○ = almost offers the benefit; *no circle* = does not offer the benefit.

We use the rating criteria defined in Bonneau *et al.* [5] to compare Security Keys to passwords alone, as well as to other second factor authentication methods in common use in online accounts today. A summary of the comparison can be seen in Table 1. We use the ratings Bonneau *et al.* assign, with the exception of the phishing protection existing second factor schemes provide. We discuss this more shortly. In short, Security Keys offer similar usability to just passwords while being much more secure. In addition, Security Keys can be deployed for supported browsers with a relatively small server-side change.

Usability. Security Keys partially offer the *memorywise-effortless* and *scalable-for-users* benefits because they dramatically reduce the risk of password reuse. They are not *physically-effortless* because they still require a password entry, but the additional burden—a button push—is low. They are both *easy-to-learn* and *efficient-to-use*, and they perform with *infrequent-errors*. These assertions are further supported in Section 6.3.

Deployability. Security Keys are *accessible*—the physical burden of tapping a button is minimal; visually impaired users within Google successfully use Se-

curity Keys. Security Keys nearly offer a *negligible-cost-per-user*: the user only needs one for any number of websites, and they are available from multiple vendors at varying prices. Bonneau *et al.* do not offer guidance to what price is considered "negligible," so we give Security Keys partial credit. Security Keys are partially *browser-compatible*: one major browser has built-in support for them, and we are working toward standardizing support for them.

Security Keys are *mature*: they have been implemented and deployed on several large web properties, discussed further in Section 6.3. Finally, they are *non-proprietary*: There are open standards and source for them. They are available from multiple vendors. Support for them is deployed in one major browser [12], with suport from another major browser announced [34]. Multiple servers have implemented support for them.

Security. Security Keys generate assertions that protect users against phishing and website attackers. While Bonneau *et al.* claim existing second factor schemes provide protection against phishing, they do so under the assumption that relay-based or realtime phishing attacks are hard to mount. We disagree given recent evidence of successful phishing campaigns against accounts protected by OTP credentials [9], hence we downgrade the protection for OTP-based second factors. Security Keys generate unique key pairs per account and restrict key use to a single origin, protecting users against tracking across websites/linkability. Unlike other hardware 2nd factor solutions such as RSA SecurID, there is no trusted third party involved. Security Keys, when used with TLS Channel ID, also provide resistance against man-in-the-middle attacks by letting servers recognize the presence of two different TLS connections. Finally, Security Keys limit the user's exposure to session riding by requiring that a TUP is performed. Note that Security Keys do not allow transaction confirmation via a trusted display, therefore clever attacker may still alter transaction details (*e.g.*, transfer amounts)—though not without a TUP.

6.2 Hardware Performance

The performance of an operation involving a Security Key involves many variables. Nevertheless, our aim was to create a protocol that seems "fast enough" during ordinary use. Our informal guideline was that a sign operation should complete in well under a second, while registration should complete in around a second. Measured times for both operations for commercially available hardware are shown in table 2. These times show the "raw" performance time—the time it takes for hardware devices to execute each operation.

With these speeds, we believe we achieved an experience whereby users do not find the additional step (beyond a password) onerous. We will expand upon our experience further in the next section.

6.3 Deployment Experience

Because of the usability and security benefits Security Keys provide over OTPs, we have deployed them to more than 50,000 employees, and made them an

Table 2: Security Key performance. Raw time needed to complete an operation.

	Operation time (ms)	
Device	Enroll	Sign
Happlink FIDO U2F	1210	660
Yubico FIDO U2F	394	192
Yubico YubiKey NEO	398	192

available option for consumer accounts. Our users have been very happy with the switch: we received many instances of unsolicited positive feedback.

The benefits of Security Keys are difficult to quantify. The first benefit is increased security: users are now protected against phishing, including from well-known campaigns. Unfortunately, the impact of this benefit can only be measured in terms of what did not happen, hence it is hard to quantify. Other impacts include increased productivity due to decreased time spent authenticating, and decreased support cost; we quantify these below.

Time Spent Authenticating. We compared the time it takes to authenticate using Security Keys versus other two-factor methods, using two user populations: Google employees and consumers using our web products.

Figure 6 shows the average time spent authenticating, per user, for Google's employees during an arbitrary two-day period. All authentication steps are measured, *i.e.*, a user is entering a password and providing a second factor. In the measurements, Security Keys are compared with OTPs, where the OTP may be provided by one of several sources, the most prevalent being a USB device that acts as a keyboard. The data originated from authentication events in a two-day period and thus could be biased toward those who reauthenticate frequently, and therefore are best trained in the use of their second factor. Because Security Keys were considered equivalent to OTPs during the period of study, Security Key users were not prompted more or less frequently than OTP users, and we do not expect a bias between the two populations as a result.

Total authentication time decreased markedly when using Security Keys; this may account for the overwhelmingly positive reaction. Previously, the dominant OTP mechanism in the company was a USB-based device very similar to a Security Key, but Security Keys are still faster. One possible reason is that most employees received a tiny Security Key that is meant to never be removed from the USB port. Thus, when the user is prompted for a second factor, she need only touch the Security Key's button, rather than having first to find and insert the device. A second reason is that the USB OTP devices act like a keyboard, and require the user to 1) navigate to a form field before touching the device to release the OTP, then 2) press Enter or click a button to a submit a web form. With Security Keys, on the other hand, the JavaScript API returns the result of the Security Key request directly to the page, and the form submission can happen automatically as a result without additional user action.

Figure 6 also shows the average time required to collect a second factor from consumers over several days in 2016. Again, Security Keys were faster for consumers to use than OTPs, whether they were delivered by SMS or via a

smartphone app. Earlier in this work, we gave several reasons why this could be: SMS delivery can suffer delays, while OTPs in a smartphone app must be manually typed by users.

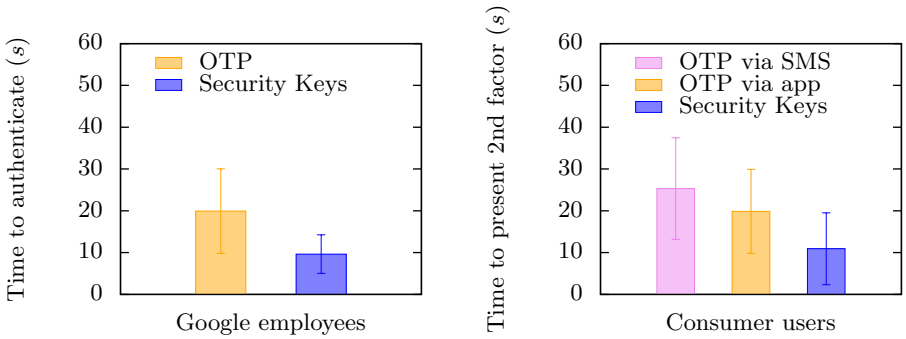

Fig. 6: Time spent authenticating

Authentication Failure Rate. Authentication failures result in increased authentication time and user frustration. In our examination of the time period studied, 3% of OTP-based authentications resulted in failure, while Security Keys did not present any authentication failures.

Support Cost. Figure 7 shows the number of support incidents we received for two kinds of authentication second factors for the period in which we transitioned from using OTPs to Security Keys, normalized by the total number of employees, on a linear scale. The number of authentication events per user did not depend on whether the users had Security Keys or OTPs for a second factor, so the number of support events is believed to be representative. There is a gap in data collection, when support data were transitioned for one system to another, and Security Key support incidents were not collected. The approximate percentage of the company actively using Security Keys is also shown.

The number of support incidents per user rose slightly as the rollout of Security Keys expanded, before decreasing again. It's worth noting that the support load was higher for OTP than for Security Keys for all time periods. By end of the period studied, the vast majority of the company had switched from OTP to Security Keys, and new employees were no longer given OTP devices. Our support organization estimates that we save thousands of hours per year in support cost by switching from OTP to Security Key.

Hardware Cost. For the corporate-wide deployment we studied, the devices purchased had similar per-unit cost to the USB-based OTP devices they replaced. For this deployment, one Security Key was allotted per computer per employee. Roughly, this equated to giving each employee 2 Security Keys, on average. This implies that more was spent on hardware Security Key tokens than for hardware OTP tokens. For the deployment, we found the increased

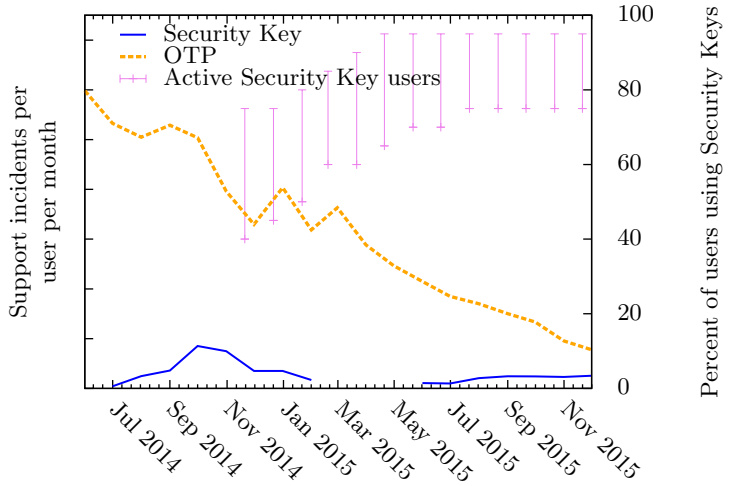

Fig. 7: 2nd factor support incidents per employee per month (gaps where data are unavailable)

user productivity, and decreased support cost, were worth the increased hardware cost.

For consumers, multiple vendors provide Security Keys at different price points, some as low as $6 USD[3]. Since users only need one device, rather than one device per account or site, the resulting cost in our opinion approaches the "negligible cost per user" suggested by Bonneau *et al.* [5].

7 Discussion

7.1 Attestation

We chose batch attestation in order to allow servers to assess the trustworthiness of a device, while still affording privacy for the user. There are two other alternatives that offer attestation while protecting users' privacy: employing a trusted third party, and using advanced cryptographic techniques.

In a trusted third party scheme, a device makes an individually identifiable attestation statement to a privacy CA, first proposed by the Trusted Computing Group in their Trusted Platform Module (TPM) 1.1 specifications. A privacy CA provides its own anonymous attestation of the device's trustworthiness to the interested party. However, one challenge is that the privacy CA must always be available and reachable. Another is the distributed trust it requires: Who will run the privacy CA? Can different entities pick a privacy CA that suits them? What happens if each party doesn't trust the same privacy CA? Finally, a privacy CA would learn the identity of many individuals, presenting a single point of failure for compromising user privacy.

[3] `http://smile.amazon.com/s/ref=sr_kk_1?rh=k:u2f`

Advanced cryptographic techniques such as Direct Anonymous Attestation [35] and Enhanced Privacy ID [36] address many of the shortcomings of a privacy CA scheme, by allowing revocation without involving a trusted third party. We did not choose these approaches because they have not yet been demonstrated to be fast enough on low-cost hardware. For example, Bichsel *et al.* [37] evaluated an implementation of an RSA-based scheme running on JavaCard devices, which runs in roughly 16.5 seconds. This is well beyond the "around a second" guideline we had for registration operations.

7.2 Signature Counter

Our design includes a 32-bit signature counter, but leaves some decisions up to implementers: (1) wrapping behavior and (2) increment amount. Since a TUP is performed with every signature, a 32-bit counter provides the ability to perform a signature per second for more than 100 years before wrapping, and it seems reasonable that this is outside the lifetime of a Security Key. For the increment amount, an implementer could have a per-key handle counter, or could use a single global counter on a particular device. The latter choice is cheaper and easier to implement, although it presents a minor privacy leak if the counter update amount is predictable. Implementers could choose to increment a global counter by a randomly varying amount, though they would have to take care to avoid early counter wrapping as a result.

8 Conclusion

We have presented Security Key, a special-purpose device for improved second factor authentication on the web. Security Keys protect users against password reuse, phishing, and man-in-the-middle attacks by binding cryptographic assertions to website origin and properties of the TLS connection.

Security Keys also score favorably in the usability framework established by Bonneau *et al.* [5]. This is further substantiated by our preliminary data analysis which quantifies the benefits of Security Keys in a two-year deployment study by measuring reduction in sign-in times experienced by users and reduction in burden on a support organization.

The Security Key protocol has been standardized within the the FIDO Alliance organization as the Universal Second Factor (U2F) open standard. We have open-sourced a reference implementation of the standard. Security Keys are supported by the Chrome browser and by the login system of major web service providers such as Google, GitHub, and DropBox. We hope this paper serves as an academic foundation to study and improve Security Keys going forward. Note, an updated and extended tech report is available at `https://github.com/google/u2f-ref-code/docs/SecurityKeys_TechReport.pdf`.

Acknowledgements Listing all of the people who have contributed to the design, implementation, and evaluation of Security Keys is virtually impossible.

We would like to thank the anonymous reviewers, along with the following individuals: Arnar Birgisson, Frank Cusack, Jakob Ehrensvärd, Kenny Franks, Iulia Ion, Benjamin Kalman, Kyle Levy, Brett McDowell, Dan Montgomery, Ratan Nalumasu, Rodrigo Paiva, Nishit Shah, Matt Spear, Jayini Trivedi, Mike Tsao, Mayank Upadhyay, and many Google teams (UX, QA, Legal).

References

1. Fallows, J.: Hacked! The Atlantic (November 2011)
2. Honan, M.: How Apple and Amazon Security Flaws Led to My Epic Hacking. http://www.wired.com/2012/08/apple-amazon-mat-honan-hacking/all/ (August 2012) [Online; accessed 31-December-2014].
3. Wikipedia: 2014 celebrity photo hack — wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=2014_celebrity_photo_hack&oldid=640287871 (2014) [Online; accessed 31-December-2014].
4. Bonneau, J.: The science of guessing: analyzing an anonymized corpus of 70 million passwords. In: 2012 IEEE Symposium on Security and Privacy. (May 2012)
5. Bonneau, J., Herley, C., van Oorschot, P.C., Stajano, F.: The Quest to Replace Passwords: A Framework for Comparative Evaluation of Web Authentication Schemes. In: 2012 IEEE Symposium on Security and Privacy. (May 2012)
6. Herley, C., van Oorschot, P.C., Patrick, A.S.: Passwords: If were so smart, why are we still using them? In: Financial Cryptography and Data Security. Volume 5628. Springer (2009) 230–237
7. Google Inc: Google 2-Step Verification (2015) https://support.google.com/accounts/answer/180744.
8. Bank of America: SafePass Online Banking Security Enhancements (2015) https://www.bankofamerica.com/privacy/online-mobile-banking-privacy/safepass.go.
9. Railton, J.S., Kleemola, K.: London Calling: Two-Factor Authentication Phishing From Iran (2015) https://citizenlab.org/2015/08/iran_two_factor_phishing/.
10. Harbach, M., Fahl, S., Rieger, M., Smith, M.: On the Acceptance of Privacy-Preserving Authentication Technology: The Curious Case of National Identity Cards. In: Privacy Enhancing Technologies, Springer (2013) 245–264
11. Unknown: Estonia takes the plunge: A national identity scheme goes global (2014) http://www.economist.com/news/international/21605923-national-identity-scheme-goes-global-estonia-takes-plunge.
12. Shah, N.: Strengthening 2-Step Verification with Security Key (2014) https://googleonlinesecurity.blogspot.com/2014/10/strengthening-2-step-verification-with.html.
13. Heim, P., Patel, J.: Introducing U2F support for secure authentication (2015) https://blogs.dropbox.com/dropbox/2015/08/u2f-security-keys/.
14. Toews, B.: GitHub supports Universal 2nd Factor authentication (2015) https://github.com/blog/2071-github-supports-universal-2nd-factor-authentication.
15. Fast IDentity Online (FIDO): (2015) https://fidoalliance.org/.
16. Biddle, R., Chiasson, S., Van Oorschot, P.: Graphical Passwords: Learning from the first twelve years. ACM Comput. Surv. **44**(4) (September 2012) 19:1–19:41

17. Jain, A.K., Flynn, P., Ross, A.A.: Handbook of Biometrics. 1st edn. Springer Publishing Company, Incorporated (2010)
18. Parno, B., Kuo, C., Perrig, A.: Phoolproof phishing prevention. In Crescenzo, G.D., Rubin, A.D., eds.: Financial Cryptography and Data Security, February 27-March 2, 2006. Volume 4107 of Lecture Notes in Computer Science., Springer (2006) 1–19
19. Toopher Inc.: Toopher - 2 Factor Authentication (2012) http://toopher.com.
20. Oracle: Java Card Technology (2014) http://www.oracle.com/technetwork/java/embedded/javacard/overview/index.html.
21. Dierks, T., Rescorla, E.: The Transport Layer Security (TLS) Protocol, Version 1.2. http://tools.ietf.org/html/rfc5246 (Aug 2008)
22. Gaw, S., Felten, E.W.: Password Management Strategies for Online Accounts. In: Proc. SOUPS 2006, ACM Press, ACM Press (2006) 44–55
23. Fontana, J.: Stolen passwords re-used to attack Best Buy accounts (2012) http://www.zdnet.com/stolen-passwords-re-used-to-attack-best-buy-accounts-7000000741/.
24. Aircrack: Aircrack-ng Homepage (2015) http://www.aircrack-ng.org/doku.php.
25. Butler, E.: Firesheep (2010) http://codebutler.com/firesheep.
26. Ewen, M.: The NSA Files (2015) http://www.theguardian.com/us-news/the-nsa-files.
27. The Register: Microsoft Outlook PENETRATED by Chinese 'man-in-the-middle' (2015) http://www.theregister.co.uk/2015/01/19/microsoft_outlook_hit_by_mitm_attack_says_china_great_fire_org/.
28. Adkins, H.: An update on attempted man-in-the-middle attacks. http://googleonlinesecurity.blogspot.com/2011/08/update-on-attempted-man-in-middle.html (Aug 2011)
29. Rizzo, J., Duong, T.: BEAST. http://vnhacker.blogspot.com/2011/09/beast.html (Sept 2011)
30. AlFardan, N.J., Paterson, K.G.: Lucky Thirteen: Breaking the TLS and DTLS Record Protocols. http://www.isg.rhul.ac.uk/tls/TLStiming.pdf (2013)
31. Dietz, M., Czeskis, A., Balfanz, D., Wallach, D.S.: Origin-bound Certificates: A Fresh Approach to Strong Client Authentication for the Web. In: Proceedings of the 21st USENIX Conference on Security Symposium. Security'12, Berkeley, CA, USA, USENIX Association (2012) 16–16
32. Popov, A., Balfanz, D., Nystroem, M., Langley, A.: The Token Binding Protocol Version 1.0 (2015) https://tools.ietf.org/html/draft-ietf-tokbind-protocol.
33. Nilsson, D.: Yubico's Take On U2F Key Wrapping. https://www.yubico.com/2014/11/yubicos-u2f-key-wrapping/ (November 2014) [Online; accessed 6-January-2016].
34. Barnes, R.: Intent to implement and ship: FIDO U2F API (2015) https://groups.google.com/forum/#!msg/mozilla.dev.platform/IVGEJnQW3Uo/Eu5tvyLmCgAJ.
35. Brickell, E., Camenisch, J., Chen, L.: Direct Anonymous Attestation. In: Proceedings of the 11th ACM Conference on Computer and Communications Security. CCS '04, New York, NY, USA, ACM (2004) 132–145
36. Brickell, E., Li, J.: Enhanced Privacy ID: A Direct Anonymous Attestation Scheme with Enhanced Revocation Capabilities. In: Proceedings of the 2007 ACM Workshop on Privacy in Electronic Society. WPES '07, New York, NY, USA, ACM (2007) 21–30
37. Bichsel, P., Camenisch, J., Groß, T., Shoup, V.: Anonymous credentials on a standard Java Card. In: Proceedings of the 16th ACM conference on Computer and communications security, ACM (2009) 600–610