T/Key: Second-Factor Authentication From Secure Hash Chains

Dmitry Kogan*
Stanford University
dkogan@cs.stanford.edu

Nathan Manohar*
Stanford University
nmanohar@cs.stanford.edu

Dan Boneh
Stanford University
dabo@cs.stanford.edu

ABSTRACT

Time-based one-time password (TOTP) systems in use today require storing secrets on both the client and the server. As a result, an attack on the server can expose all second factors for all users in the system. We present T/Key, a time-based one-time password system that requires no secrets on the server. Our work modernizes the classic S/Key system and addresses the challenges in making such a system secure and practical. At the heart of our construction is a new lower bound analyzing the hardness of inverting hash chains composed of independent random functions, which formalizes the security of this widely used primitive. Additionally, we develop a near-optimal algorithm for quickly generating the required elements in a hash chain with little memory on the client. We report on our implementation of T/Key as an Android application. T/Key can be used as a replacement for current TOTP systems, and it remains secure in the event of a server-side compromise. The cost, as with S/Key, is that one-time passwords are longer than the standard six characters used in TOTP.

1 INTRODUCTION

Static passwords are notorious for their security weaknesses [11, 46, 64–66], driving commercial adoption of two-factor authentication schemes, such as Duo [55], Google authenticator [24], and many others. Several hardware tokens provide challenge-response authentication using a protocol standardized by the FIDO industry alliance [59].

Nevertheless, for desktop and laptop authentication, there is a strong desire to use the phone as a second factor instead of a dedicated hardware token [41, 55, 56, 67]. Several systems support phone-based challenge-response authentication (e.g., [55]), but they all provide a fall back mode to a one-time password scheme. The reason is that challenge-response requires two-way communication with the phone: uploading the challenge to the phone and sending the response from the phone to the server. However, one cannot rely on the user's phone to always be connected. When the user is traveling, she may not have connectivity under the local cell provider, but may still wish to use her laptop to log in at a hotel or to log in using a workstation at an Internet Cafe. In this case, authentication systems, such as Duo, fall back to a standard timed-based one-time password (TOTP) scheme.

Standard TOTP schemes [48] operate using a shared key k stored on both the phone and the authentication server. The phone displays a six digit code to the user, derived from evaluating HMAC(k, t),

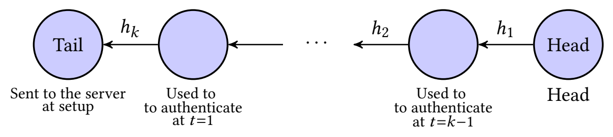

Tail
h_k
h_2
h_1
Head
Sent to the server at setup
Used to to authenticate at t=1
Used to to authenticate at t=k−1
Head

Figure 1: Sketch of T/Key

where t is the current time, rounded to the current 30 second multiple. This way, the code changes every 30 seconds and can only be used once, hence the name *one-time password*. The user enters the code on her laptop, which sends it to the server, and the server verifies the code using the same key k. The server accepts a window of valid codes to account for clock-skew.

The benefit of TOTP schemes is that they only require *one-way* communication from the phone to the laptop, so they can function even if the phone is offline (challenge-response requires two-way communication with the phone and is mostly used when the phone is online). However, a difficulty with current TOTP is that the server must store the user's secret key k *in the clear*. Otherwise, the server cannot validate the 6-digit code from the user. With this design, a break-in at the server can expose the second factor secret for all users in the system. A well-publicized event of this type is the attack on RSA SecurID, which led to subsequent attacks on companies that rely on SecurID [61].

Our work. We introduce a TOTP system called T/Key that requires no secrets on the server. Our starting point is a classic one-time password system called S/Key [27], which is not time-based and suffers from a number of security weaknesses, discussed in the next section. Our work modernizes S/Key, makes it time-based (hence the name T/Key), and addresses the resulting security challenges.

In T/Key, the phone generates a hash chain seed and uses this seed to construct a long hash chain, say of length two million, as depicted in Figure 1. The phone encodes the tail of the chain T in a QR code, which the user scans with her laptop and sends to the authentication server for storage. The phone then starts at the element immediately preceding T in the chain and walks one step backwards along the chain once every 30 seconds. It does so until it reaches the head of the chain, which is the seed. At every step, the phone displays the current element in the chain, and the user logs in by copying the displayed value onto her laptop. At the rate of one step every 30 seconds, a single chain is good for approximately two years, at which point the phone generates a new chain. The details of the scheme are presented in Section 3. As in TOTP, there is only one-way communication from the phone to the laptop, and the phone can be offline. Moreover, a server compromise reveals nothing of value to the attacker.

Such a TOTP scheme presents a number of challenges. First, security is unclear. Imagine an attacker breaks into the server and

*Both authors contributed equally to the paper

CCS '17, October 30–November 3, 2017, Dallas, TX, USA
2017. This is the author's version of the work. It is posted here for your personal use. Not for redistribution. The definitive Version of Record was published in *Proceedings of CCS '17, October 30–November 3, 2017*, https://doi.org/10.1145/3133956.3133989.

[illegible]

Table 1: A comparison of OTP schemes.

	No server secrets	Time-varying passwords	Password length in bits (at 2^{128} security)
S/Key	✓	✗	N/A
TOTP (HMAC)	✗	✓	20
Digital Signatures (ECDSA/EdDSA)	✓	✓	512
T/Key	✓	✓	130

Note: S/Key does not support this level of security.

steals the top of the chain T. The attacker knows the *exact time* when the millionth inverse of T will be used as the second factor. That time is about a year from when the break-in occurs, which means that the attacker can take a year to compute the millionth inverse of T. This raises the following challenge: for a given k, how difficult is it to compute the k-th inverse of T?

If the same function is used throughout the entire hash chain, as in S/Key, the scheme is vulnerable to "birthday attacks" [31] and is easier to break than the original hash function [28]. A standard solution is to use a different hash function at every step in the chain. The question then is the following: if H is the composition of k *random* hash functions, namely

$$H(x) := h_k(h_{k-1}(\cdots(h_2(h_1(x)))\cdots)),$$

how difficult is it to invert H given $H(x)$ for a random x in the domain? We prove a time lower bound for this problem in the random oracle model. Additionally, given the possibility of making time-space tradeoffs in attacks against cryptographic primitives [17, 29, 49], a natural follow up question is whether the scheme is still secure against offline attackers. Building on the recent results of Dodis, Guo and Katz [18], we prove a time-space lower bound for this problem that bounds the time to invert H, given a bounded amount of preprocessing space. As hash chains are a widely used primitive, we believe that our lower bounds, both with and without preprocessing, may be of independent interest.

From this security analysis, we derive concrete parameters for T/Key. For 2^{128} security, every one-time password must be 130 bits. Since entering these one-time passwords manually would be cumbersome, our phone implementation displays a QR code containing the one-time password, which the user scans using her laptop camera. We describe our implementation in Section 6 and explain that T/Key can be used as a drop-in replacement for Google Authenticator. The benefit is that T/Key remains secure in the event of a server-side compromise.

We also note that USB-based one-time password tokens, such as Yubikey [69], can be set up to emulate a USB keyboard. When the user presses the device button, the token "types" the one-time password into a browser field on the laptop. This one-way communication setup is well suited for T/Key: the token computes a T/Key one-time password and enters it into a web page by emulating a keyboard. Again, this TOTP system remains secure in the event of a server-side compromise.

The second challenge we face is performance. Because the hash chain is so long, it is unreasonable for the phone to recompute the entire hash chain on every login attempt, since doing so would take several seconds for every login. Several amortized algorithms have been developed for quickly walking backwards on a hash chain, while using little memory on the phone [13, 33]. The problem is that these schemes are designed to walk backwards a *single* step at a time. In our case, the authenticator app might not be used for a month or, perhaps, even longer. Once the user activates the app, the app must quickly calculate the point in the hash chain corresponding to the current time. It would take too long to walk backwards from the last login point, one step at a time, to reach the required point.

Instead, we develop a new approach for pebbling a hash chain that enables a quick calculation of the required hash chain elements. We model the user's login attempts as a Poisson process with parameter λ and work out a near-optimal method to quickly compute the required points with little memory on the phone.

Other approaches. T/Key is not the only way to provide a TOTP with no secrets on the server. An alternate approach is to use a digital signature. The phone maintains the signing key, while the server maintains the signature verification key. On every authentication attempt, the phone computes the signature on the current time, rounded to a multiple of 30 seconds. This can be scanned into the laptop and sent to the server to be verified using the verification key.

While this signature-based scheme has similar security properties to T/Key, it has a significant limitation. Standard digital signatures such as ECDSA [34] and EdDSA [5, 35] are *512 bits long* for 2^{128} security[1]. These are about four times as long as the tokens used in T/Key. For example, when encoded as QR codes, the longer tokens result in a denser QR code. To preserve the maximal scanning distance, the denser QR code must be displayed in a larger image [51]. (Alternatively, the signatures could be decomposed into several QR codes of the original size, but scanning multiple images introduces additional complexity for the user.) Short authentication tokens might also be desirable in other applications such as Bluetooth Low Energy (which supports a 23-byte long MTU [58]).

Table 1 provides a comparison of the different TOTP mechanisms and their properties. The last column shows the required length of the one-time password.

Beyond authentication. Hash chains come up in a number of other cryptographic settings, such as Winternitz one-time signatures [12] and the Merkle-Damgard construction [44]. Existing

[1]BLS signatures [8] are shorter, but require a pairing operation on the server which makes them less attractive in these settings.

security proofs for Winternitz signatures often only take into account the attacker's online work. Our lower bound on inverting hash chains is well suited for these settings and can be used to derive time-space tradeoff proofs of security for these constructions. This is especially relevant as these schemes are being standardized [32, 36, 43].

2 OFFLINE 2ND FACTOR AUTHENTICATION

We begin by briefly reviewing several approaches to one-time passwords that are most relevant to our scheme.

S/Key. The idea of a one-time password authentication scheme was first considered by Lamport [38]. Loosely speaking, in such a scheme, following an initial setup phase, authentication is performed by the client presenting the server with a password that is hard for an attacker to guess, even given all previous communication between the server and the client. In particular, no password is valid for more than one authentication. In his work, Lamport proposed a concrete instantiation of this idea using *hash chains*, and this idea has been subsequently developed and implemented under the name S/Key [27]. The setup phase of S/Key consists of the client choosing a secret passphrase[2] x and sending the computed value $y_0 = h^{(k)}(x)$ (where h is some cryptographic hash function, k is some integer, and $h^{(k)}$ denotes k successive iterations of h) to the server, which the server then stores. Subsequently, to authenticate for the ith time, the client must present the server with $y_i = h^{(k-i)}(x)$, which the server can verify by computing $h(y_i)$ and comparing it to the stored value y_{i-1}. If the authentication is successful, the server updates its stored value to y_i.

S/Key has a number of undesirable properties. First, one-time passwords remain valid for an indefinite period of time unless used, making them vulnerable to theft and abuse. This vulnerability is magnified if the counter value for each authentication attempt is communicated to the client by the server, as is the case in both the original S/Key [27] and in the newer OPIE [42] (presumably to allow for stateless clients). In this common setting, the scheme is vulnerable to a so-called "small n" attack [45], where an attacker impersonating the server can cause the client to reveal a future one-time password. Second, the fact that S/Key utilizes the same hash function at every iteration in the chain makes it easier to break S/Key than to break a single hash function (see Theorem 4.1). This also implies that any modification to the scheme that requires using much longer hash chains (such as, for example, a naïve introduction of time-based passwords) could lead to insecurity.

HOTP. In an *HMAC-based one-time password scheme* (HOTP) [47], a secret and a counter, both shared between the server and the client, are used in conjunction with a pseudorandom function (HMAC) to generate one-time passwords. The setup phase consists of the server and the client agreeing on a random shared secret k and initializing a counter value c to 0. One-time passwords are then generated as $\mathtt{HMAC}(k, c)$. The counter is incremented by the client every time a password is generated and by the server after every successful authentication.

[2]Usually, the client's secret passphrase is concatenated with a random salt to prevent dictionary attacks and reduce the risk of reusing the same passphrase on multiple servers.

The most significant advantage of this scheme is that the number of authentications is unbounded. Moreover, it allows using short one-time passwords without compromising security. However, HOTP still suffers from many of the weaknesses of S/Key, namely that unused passwords remain valid for an indefinite period of time. A bigger concern is that the secret key k must be stored on the server, as discussed in the previous section.

TOTP. Time-based one-time password schemes (TOTP) [48] were introduced to limit the validity period of one-time passwords. In TOTP, the shared counter value used by HOTP is replaced by a timestamp-based counter. Specifically, the setup phase consists of the server and the client agreeing on the 'initial time' t_0 (usually the UNIX epoch) and a time slot size I (usually 30 seconds), as well as on a secret key k. Subsequently, the client can authenticate by computing $\mathtt{HMAC}(k, (t - t_0)/I)$, where t is the time of authentication. As with HOTP, the TOTP scheme is vulnerable to a server-side attack.

3 OUR CONSTRUCTION

T/Key combines the ideas used in S/Key and TOTP to achieve the best properties of both schemes: T/Key stores no secrets on the server and ensures that passwords are only valid for a short time interval. The scheme works as follows:

Public Parameters. The scheme's parameters are the password length n (in bits), a time slot size I (in seconds), representing the amount of time each password is valid for, and the maximal supported authentication period k (measured as the number of slots of size I). Furthermore, our scheme uses some public cryptographic hash function $H : \{0,1\}^m \to \{0,1\}^m$ for an arbitrary $m \geq n + s + c$, where s is the number of bits used for the salt and c is the number of bits needed to represent the time. Typical values are given in Table 2.

Table 2: Scheme public parameters and their typical values.

Parameter	Value	Description
n	130 bits	One-time password length
s	80 bits	Salt length
c	32 bits	Number of bits used for time
m	256 bits	Hash function block size
k	2×10^6	Chain length
I	30 sec	Time slot length

Setup. The client chooses and stores a uniformly random secret key $sk \in \{0,1\}^n$, as well as a random salt $id \in \{0,1\}^s$, and notes the setup time t_{init} (measured in slots of length I). The public hash function H together with the initialization time t_{init} induce the k independent hash functions $h_1, \ldots, h_k : \{0,1\}^n \to \{0,1\}^n$ as follows: for $1 \leq i \leq k$ define

$$h_i(x) = H\left(\langle t_{\text{init}} + k - i \rangle_c \,\|\, id \,\|\, x\right)\Big|_n ,$$

where for a numerical value t, $\langle t \rangle_c$ denotes the c-bit binary representation of t, and for strings $x, y \in \{0,1\}^*$, we write $x|_n$ and $x\|y$ to denote the n-bit prefix of x and the concatenation of x and y,

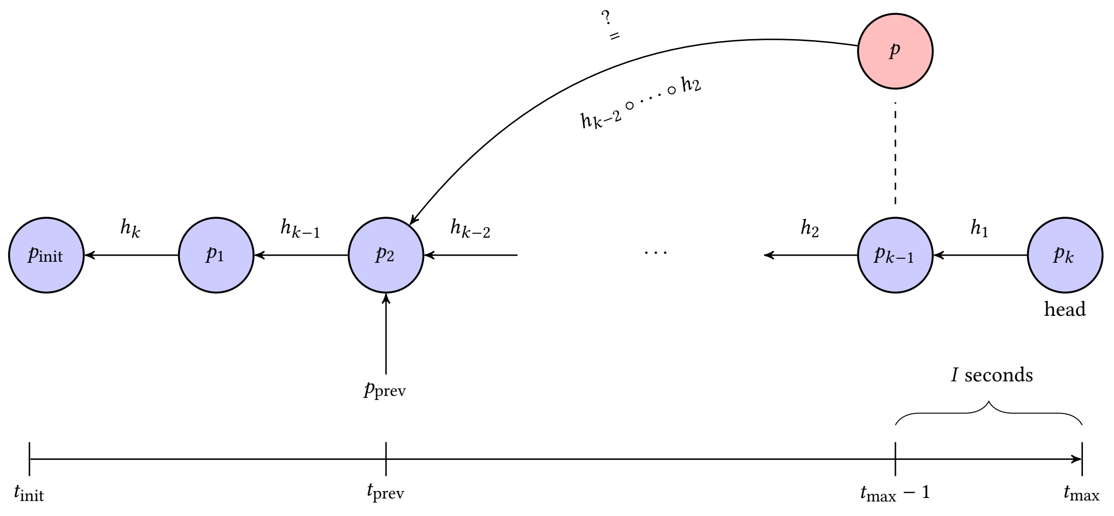


Figure 2: A basic overview of T/Key. A user has submitted the password p at time $t_{\max} - 1$. Since the previous login occurred at time $t_{\text{prev}} = t_{\text{init}} + 2$, the server has stored $p_{\text{prev}} = p_2$ as the previous password. To authenticate the user, the server computes $h_{k-2}(\ldots(h_2(p))\ldots)$ and checks if it is equal to p_{prev}.

respectively. This simple method of obtaining independent hash functions from a single hash function over a larger domain is called *domain separation*, and it is often attributed to Leighton and Micali [39]. Note that since all inputs to the hash function are of equal size, this construction is not susceptible to length extension attacks, and therefore, there is no need to use HMAC.

The client then computes

$$p_{\text{init}} = h_k(h_{k-1}(\ldots(h_1(sk))\ldots))$$

and sends it to the server together with *id*. The server stores p_{init} as p_{prev} as well as the time t_{init} as t_{prev} (we discuss time synchronization issues below).

Authentication. To authenticate at a later time $t \in (t_{\text{init}}, t_{\max}]$ (measured in units of length I where $t_{\max} = t_{\text{init}} + k$), the client and server proceed as follows: the client uses sk and t to generate the one-time password

$$p_t = h_{t_{\max}-t}(h_{t_{\max}-t-1}(\ldots(h_1(sk))\ldots)).$$

Alternatively, when $t = t_{\max}$, we use $p_t = sk$. To check a password p, the server uses the stored values, t_{prev} and p_{prev}, and the current time-based counter value $t > t_{\text{prev}}$. The server computes

$$p'_{\text{prev}} = h_{t_{\max}-t_{\text{prev}}}(h_{t_{\max}-t_{\text{prev}}-1}(\ldots(h_{t_{\max}-t+1}(p))\ldots)).$$

If $p'_{\text{prev}} = p_{\text{prev}}$, then authentication is successful, and the server updates p_{prev} to p and t_{prev} to t. Otherwise, the server rejects the password.

Reinitialization. Just as in authentication, initialization requires communication only from the client device to the server, and the server does not need to send anything to the client. The only difference is that during initialization, the client needs to supply the server with the salt in addition to the initial password. The finite length of the hash chain requires periodic reinitialization, and the length of this period trades off with the time step length I and the time it takes to perform the initialization (which is dominated by the full traversal of the hash chain by the client). For standard use cases, one can set $I = 30$ seconds and $k = 2 \times 10^6$, which results in a hash chain valid for 2 years and takes less than 15 seconds to initialize on a modern phone.

Since key rotation is generally recommended for security purposes (NIST, for example, recommends “cryptoperiods” of 1-2 years for private authentication keys [3]), we don’t view periodic reinitialization as a major limitation of our scheme. While reinitialization is obviously somewhat cumbersome, there are several properties of our scheme that mitigate the inconvenience. First, the fact that our setup is unidirectional makes it very similar to authentication from the user’s point of view. Second, from a security standpoint, the setup is not vulnerable to passive eavesdrop attacks, unlike TOTP schemes that rely on shared secrets.

A scenario where the hash chain expires before the user is able to reinitialize it with the server can be handled out-of-band in a manner similar to password recovery or loss of the second-factor. Alternatively, some implementations could choose to accept the head of the chain even after its validity period, which would incur a loss in security proportional to the time elapsed since expiration.

Clock Synchronization. As with current TOTP schemes, authentication requires a synchronized clock between the server and the client. Time skew, or simply natural delay between the moment of password generation and the moment of verification, might result in authentication failure. To prevent this, the server may allow the provided password to be validated against several previous time steps (relative to the server’s clock), as was the case in the TOTP scheme. When this occurs, the previous authentication timestamp t_{prev} stored on the server should be updated to the timestamp which resulted in successful verification.

Figure 2 illustrates the design of T/Key.

4 SECURITY

Although our scheme bears a resemblance to both S/Key and TOTP, it has several essential differences that eliminate security issues present in those schemes.

First and foremost, T/Key does not require the server to store any secrets, which mitigates the risk of an attack that compromises the server's database, unlike TOTP, which requires the client's secret key to be stored by the server.

Second, T/Key's passwords are time limited, unlike those in S/Key, which makes phishing attacks more difficult because the attacker has a limited time window in which to use the stolen password. However, the fact that T/Key's passwords are time limited makes it necessary for the hash chain used by T/Key to be significantly longer than those in S/Key, since its length must now be proportional to the total time of operation rather than to the supported number of authentications. This modification raises the issue of the dependence of security on the length of the hash chain. Hu, Jakobsson and Perrig [31] discuss the susceptibility of iterating the same hash function to "birthday" attacks and Håstad and Näslund [28] show that if the *same* hash function h is used in every step of the chain, then inverting the k-th iterate is actually k times easier than inverting a single instance of the hash function. We reproduce their proof here for completeness and clarity.

We set $N = 2^n$ and denote by $\mathcal{F}_N$ the uniform distribution over the set of all functions from $[N]$ to $[N]$. For a function $h : [N] \to [N]$, we let $h^{(k)}$ denote h composed with itself k times. For functions $h_1, h_2, \ldots, h_k$ and $1 \le i \le j \le k$, we let $h_{[i,j]}$ denote the composition $h_j \circ h_{j-1} \circ \cdots \circ h_i$. When writing A^h, we mean that algorithm A is given oracle access to all k functions $h_1, \ldots, h_k$.

Theorem 4.1 ([28]). *For every $N \in \mathbb{N}$, $k \le \sqrt{N}$ and $2k \le T \le N/k$, there exists an algorithm A that makes at most T oracle queries to a random function $h : [N] \to [N]$ and*

$$\Pr_{\substack{h \in \mathcal{F}_N \\ x \in [N]}} \left[h\left(A^h(h^{(k)}(x))\right) = h^{(k)}(x) \right] = \Omega\left(\frac{Tk}{N}\right).$$

Moreover, every algorithm that makes at most T oracle queries succeeds with probability at most $O(Tk/n)$.

Proof. We prove the first part of the theorem (the existence of a "good" algorithm) and refer the reader to [28] for the proof of the second part. Consider the following algorithm: On input $h^{(k)}(x) = y \in [N]$, the algorithm sets $x_0 = y$ and then computes $x_j = h(x_{j-1})$ until either $x_j = y$, in which case it outputs x_{j-1}, or until $x_j = x_i$ for some $i < j$. In the latter case, it picks a new random x_j from the set of all points it hasn't seen before and continues. If the algorithm makes T queries to h without finding a preimage, it aborts.

To analyze the success probability of this algorithm, consider the first $(T - k)$ points $\{x_j\}_{j=1}^{T-k}$. If any of these points collides with any of the values along the hash chain $\{h^{(i)}(x)\}_{i=1}^{k}$, the algorithm will output a preimage of y after at most k additional queries. Therefore, the probability of failure is at most the probability of not colliding with the hash chain during the first $T - k$ queries. But as long as a collision does not happen, each query reply is independent of all previous replies and of the values $\{h^{(i)}(x)\}_{i=1}^{k}$. Each query therefore collides with the chain with probability at most k/N, and overall, the algorithm fails with probability at most $(1 - k/N)^{T-k} \le (1 - k/N)^{T/2} = 1 - \Omega(Tk/N)$. □

This loss of a multiplicative factor of k in security is undesirable as it forces us to increase the security parameters for the hash function to resist long-running adversaries. A standard solution is to use a different hash function at every step in the chain. The question then is the following: if H is the composition of k *random* hash functions, namely

$$H(x) := h_k(h_{k-1}(\cdots(h_2(h_1(x)))\cdots)),$$

how difficult is it to invert H given $H(x)$ for a random x in the domain? To the best of our knowledge, this aspect of hash chain security has not been analyzed previously.

In Section 4.1 we prove a time lower bound for inverting a hash chain composed of independent hash functions. We show that as opposed to the case in Theorem 4.1, where the same function is used throughout the chain, resulting in a loss of security by a factor of $O(k)$, using independent hash function results in a loss of only a factor of 2. Thus for most practical applications, a hash chain is as hard to invert as a single hash function. In Section 8, we prove a time-space lower bound for inverters that can preprocess the hash function.

4.1 A lower bound for inverting hash chains

Theorem 4.2 (Security of hash chains against online attacks). *Let functions $h_1, \ldots, h_k \in [N] \to [N]$ be chosen independently and uniformly at random. Let A be an algorithm that gets oracle access to each of the functions $\{h_i\}_{i=1}^{k}$ and makes at most T oracle queries overall. Then,*

$$\Pr_{\substack{h_1,\ldots,h_k \in \mathcal{F}_N \\ x_0 \in [N]}} \left[h_k\left(A(h_{[1,k]}(x_0))\right) = h_{[1,k]}(x_0) \right] \le \frac{2T+3}{N}.$$

Proof. Let $W = (w_0, w_1, \ldots, w_k)$ be the sequence of values of the hash chain, i.e., $w_0 = x_0$ and $w_i = h_i(w_{i-1})$ for $i \in [1, k]$. Let A be an adversary that makes at most T oracle queries. Denote by $q_j = (i_j, x_j, y_j)$ the j-th query made by A, where i_j is the index of the oracle queried, x_j is the input queried, and y_j is the oracle's response. We say that a query q_j *collides* with W if $y_j = w_{i_j}$, namely the reply to the query is a point on the hash chain. At the cost of one additional query, we modify A to query h_k on its output before returning it. Thus, we can assume that if A successfully find a preimage, at least one of its $T + 1$ queries collides with W.

Let $R = \{(i, x, y) : h_i(x) = y\}$ be the set of all random oracle queries and their answers. Using the principle of deferred decision, we can construct the set R incrementally as follows. Initially $R = \emptyset$; subsequently whenever A makes an oracle query of the form (i, x), if $x = w_{i-1}$, we respond with $y = w_i$ and add (i, w_{i-1}, w_i) to R. Else if $(i, x, y) \in R$, we reply with y. Otherwise, we choose y uniformly at random from $[N]$, add (i, x, y) to R, and reply with y.

As mentioned above, to invert the hash chain, at least one query $q_j \in R$ must collide with W. It follows that

$$\Pr_{H,x_0}[A \text{ loses}] = \Pr_{H,x_0}\left[\bigwedge_{j=1}^{T+1} y_j \neq w_{i_j}\right]$$
$$= \prod_{j=1}^{T+1} \Pr_{H,x_0}\left[y_j \neq w_{i_j} \,\middle|\, \bigwedge_{\ell=1}^{j-1} y_\ell \neq w_{i_\ell}\right].$$

To bound each term inside the product, we use the basic fact that

$$\begin{aligned}\mathbf{Pr}(A|C) &= \mathbf{Pr}(A|B,C)\,\mathbf{Pr}(B|C) + \mathbf{Pr}(A|\neg B,C)\,\mathbf{Pr}(\neg B|C)\\ &\leq \mathbf{Pr}(A|B,C) + \mathbf{Pr}(\neg B|C)\end{aligned}$$

to obtain

$$\Pr_{H,x_0}\left[y_j = w_{i_j} \,\middle|\, \bigwedge_{\ell=1}^{j-1} y_\ell \neq w_{i_\ell}\right]$$
$$\leq \Pr_{H,x_0}\left[y_j = w_{i_j} \,\middle|\, x_j \neq w_{i_j - 1} \wedge \bigwedge_{\ell=1}^{j-1} y_\ell \neq w_{i_\ell}\right]$$
$$+ \Pr_{H,x_0}\left[x_j = w_{i_j - 1} \,\middle|\, \bigwedge_{\ell=1}^{j-1} y_\ell \neq w_{i_\ell}\right].$$

Notice that the first of the two events in the last sum can only occur if x_j does not appear in R. Otherwise, $y_j \neq w_{i_j}$ due to the fact that none of the previous queries collided with W. Therefore, the reply y_j is sampled uniformly at random, and this term is at most $\frac{1}{N}$.

To bound the second term, note that each previous reply y_ℓ, provided that it does not collide with W, rules out at most one possible value for $w_{i_j - 1}$: either x_ℓ if $i_\ell = i_j$, or y_ℓ if $i_\ell = i_j - 1$. Therefore, $w_{i_j - 1}$ is distributed uniformly over the remaining values, of which there are at most $N - (j-1)$. Specifically w_{i_j-1} is equal to x_j, which is a function of all the previous replies $y_1, \ldots, y_{j-1}$, with probability at most $\frac{1}{N-j+1}$.

Overall,

$$\Pr_{H,x_0}[A \text{ loses}] \geq \prod_{j=1}^{T+1}\left(1 - \frac{1}{N} - \frac{1}{N-j+1}\right)$$
$$\geq \prod_{j=1}^{T+1}\left(1 - \frac{2}{N-j+1}\right).$$

We note that this is a telescopic product, which simplifies to

$$\frac{(N-T-2)(N-T-1)}{N(N-1)} \geq \frac{N^2 - (2T+3)N}{N^2}$$

and therefore,

$$\Pr_{H,x_0}[A \text{ wins}] \leq \frac{2T+3}{N}.$$

□

Theorem 4.2 establishes the difficulty of finding a preimage of the last iterate of the hash chain. For T/Key, we also need to bound the success probability of attacks that "guess" a preimage of the entire chain.

Corollary 4.3. *Let functions $h_1, \ldots, h_k \in [N] \to [N]$ be chosen independently and uniformly at random. Let A be an algorithm that gets oracle access to each of the functions $\{h_i\}_{i=1}^k$ and makes at most T oracle queries overall. Then,*

$$\Pr_{\substack{h_1,\ldots,h_k \in \mathcal{F}_N\\ x_0 \in [N]}}\left[h_{[1,k]}\left(A(h_{[1,k]}(x_0))\right) = h_{[1,k]}(x_0)\right] \leq \frac{2T+2k+1}{N}.$$

Proof. Let A be an algorithm as in the statement of the corollary. We use it to construct an algorithm A' that finds a preimage of the last iterate of the hash chain (as in the statement of Theorem 4.2). On input y, algorithm A' runs algorithm A to get a point z and then computes and outputs $z' = h_{[1,k-1]}(z)$. If $h_{[1,k]}(z) = h_{[1,k]}(x)$, then $h_k(z') = h_{[1,k]}(x)$. Moreover, algorithm A' makes at most $T' = T + k - 1$ queries to its oracles. Therefore by Theorem 4.2, its success probability is at most $(2T'+3)/N = (2T+2k+1)/N$. □

Optimality. One might ask whether the above lower bound is tight. Perhaps composing k independent hash functions not only avoids some of the problems associated with using a hash chain derived by composing the same hash function, but actually results in a function that is k times more difficult to invert than the basic hash function. Ideally, one might have hoped that the probability of inverting the hash chain in T queries would be at most $O\left(\frac{T}{kN}\right)$.

However, this is not the case, because every iteration of the hash chain introduces additional collisions and shrinks the domain of the function at a rate of $1/k$, where $k = o(N)$ is the length of the chain (see Lemma A.1 for a proof sketch). An attacker can use these collisions to her advantage. Consider an attack that evaluates the chain on T/k random points in its domain (at a total cost of T hash computations). Lemma A.4 shows that a point in the image of the hash chain has k preimages in expectation. Therefore, each of the T/k randomly chosen points collides with the input under the chain with probability k/N, and so the overall success probability of the attack is roughly T/N.

4.2 Security of T/Key

Our threat model assumes the adversary can repeatedly gain access to the server and obtain all information needed to verify the password. The adversary can also obtain multiple valid passwords at times of his choice. Finally, we allow the adversary to choose the time when he makes his impersonation attempt. To mitigate preprocessing attacks, we salt all our hash functions (in Section 8, we discuss preprocessing attacks in more detail, including the extent to which salting helps prevent them).

Non-threats. First, we assume that there is no malware on the phone or on the user's laptop. Otherwise, the user's session can be hijacked by the malware, and strong authentication is of little value. Second, because the channel between the laptop and the authentication server is protected by TLS, we assume there is no man-in-the-middle on this channel. Third, all TOTP schemes are susceptible to an *online* phishing attack where the attacker fools the user into revealing her short-lived one-time password to a phishing site, and the attacker then immediately authenticates as the user, within the allowable short window. This is a consequence of the requirement for *one-way* communication with the authentication token (the phone). Note however that the limited time window makes the exploitation of credentials time-sensitive, which makes the attack more complicated.

We begin by presenting a formal definition of security. Our definitions are based on standard definitions of identification protocols (see, for example, [57]).

Definition 4.4 (Time-based One-Time Password Protocol). A one-time password protocol is a tuple $\mathcal{I} = (\mathsf{pp}, \mathsf{keygen}, P, V)$ where

- Public parameter generator $\mathsf{pp}(1^\lambda, k) \rightarrow n$ is a polynomial time algorithm that takes as input the security parameter in unary along with the maximal supported authentication period k and outputs the password length n.
- Key generator $\mathtt{keygen}(n, k) \rightarrow (sk, vst)$ is a probabilistic polynomial time algorithm that takes as input the parameters, n and k, and outputs the prover's secret key sk and the initial verifier state vst.
- Prover $\mathsf{P}(sk, t) \rightarrow p_t$ is a polynomial time algorithm, which takes as input the prover's secret key sk, and a time $t \in [1, k]$, and outputs a one-time password p.
- Verifier $\mathsf{V}(vst, p, t) \rightarrow (\mathtt{accept/reject}, vst')$ is a polynomial time algorithm, which takes as input the previous state vst, a password p, and time $t \in [1, k]$ and outputs whether the password is accepted and the updated verifier state vst'.

For correctness, we require that when executed on monotonically increasing values of t with the state vst properly maintained as described above, the verifier $\mathsf{V}(vst, \mathsf{P}(sk, t), t)$ always outputs $\mathtt{accept}$.

We now proceed to define the security game, where we use the random oracle model [4].

Attack Game 4.5. *Let $\mathcal{I}$ be a time-based one-time password protocol, and let $\mathcal{O}$ be a random oracle. Given a challenger and an adversary A, the attack game runs as follows:*

- Public Parameter Generation – *The challenger generates* $n \leftarrow \mathsf{pp}(1^\lambda, k)$.
- Key Generation Phase – *The challenger generates* $(vk, sk) \leftarrow \mathtt{keygen}^{\mathcal{O}}(n, k)$*, given access to the random oracle.*
- Query Phase – *The adversary runs the algorithm A, which is given the verifier's initial state vst as well as the ability to issue the following types of (possibly adaptive) queries:*
  - *Password Queries: The adversary sends the challenger a time value t.*
    The challenger generates the password $p \leftarrow \mathsf{P}^{\mathcal{O}}(t, sk)$, feeds it to the verifier to obtain $(\mathtt{accept}, vst') \leftarrow \mathsf{V}^{\mathcal{O}}(t, vst, p)$, updates the stored verifier state to vst', and sends p to the adversary.
  - *Random Oracle Queries: The adversary sends the challenger a point x, and the challenger replies with $\mathcal{O}(x)$.*

  The above queries can be adaptive, and the only restriction is that the values of t for the password queries must be monotonically increasing.
- Impersonation attempt – *The adversary submits an identification attempt $(t_{\text{attack}}, p_{\text{attack}})$, such that t_{attack} is greater than all previously queried password values.*

We say that the adversary A wins the game if $\mathsf{V}^{\mathcal{O}}(vst, p_{\text{attack}}, t_{\text{attack}})$ outputs $\mathtt{accept}$. We let $\boldsymbol{Adv}_A(\lambda)$ denote the probability of the adversary winning the game with security parameter λ, where the probability is taken over the random oracle as well as the randomness in the key generation phase.

We are now ready to prove that T/Key is secure. Specifically, given an adversary that makes at most T queries, we establish an upper bound on the advantage the adversary can have in breaking the scheme. We note that no such result was previously known for the original S/Key scheme, and the key ingredient in our proof is Theorem 4.2.

Theorem 4.6 (Security of T/Key). *Consider the T/Key scheme with password length n and maximum authentication period k. Let A be an adversary attacking the scheme that makes at most T random oracle queries. Then,*

$$\boldsymbol{Adv}_A \leq \frac{2T + 2k + 1}{2^n}.$$

Proof. First, recall that our scheme uses a hash function $H : \{0,1\}^m \rightarrow \{0,1\}^m$ to get k functions $h_1, \ldots, h_k : \{0,1\}^n \rightarrow \{0,1\}^n$, where $h_i(x) = H(t_{\text{init}} + k - i \| id \| x)|_n$. In the random oracle model, we instantiate H using the random oracle, and so the resulting k functions, $h_1, \ldots, h_k$, are random and independent.

Without loss of generality, we assume that $t_{\text{init}} = 0$ and that the latest password requested by the adversary is the top of the chain p_k (since the functions $h_1, \ldots, h_k$ are independent, any random oracle or password queries corresponding to times earlier than the latest requested password do not help the adversary to invert the remaining segment of the chain).

By definition, the verifier accepts $(t_{\text{attack}}, p_{\text{attack}})$ if and only if $h_{[k-t_{\text{attack}}+1,k]}(p_{\text{attack}}) = h_{[1,k]}(sk)$. Therefore, if the adversary wins the game, it must hold that at least one query $q_j \in R$ collides with W. The proof then follows from Corollary 4.3. □

Concrete Security. With this result at hand, we compute the password length required to make T/Key secure. For moderate values of k (say, negligible in 2^n), to make our scheme as secure as a λ-bit random function, it is enough to set $n = \lambda + 2$, since then, assuming $k < T$,

$$\mathbf{Adv}_A \leq \frac{2T + 2k + 1}{2^n} \leq \frac{4T}{2^{\lambda+2}} = \frac{T}{2^\lambda}.$$

For standard 128-bit security, we require passwords of length 130 bits.

5 CHECKPOINTING FOR EFFICIENT HASH CHAIN TRAVERSAL

In our scheme, the client stores the secret sk, which is used as the head of the hash chain. In the password generation phase, as described in Section 3, the client must compute the value of the node corresponding to the authentication time each time it wishes to authenticate. A naive implementation would simply traverse the hash chain from the head of the chain all the way to the appropriate node. Since T/Key uses long hash chains, this approach could lead to undesirable latency for password generation. To decrease the number of hashes necessary to generate passwords, the client can store several values (called "pebbles") corresponding to various points in the chain.

There exist multiple techniques for efficient hash chain traversal using dynamic helper pointers that achieve $O(\log n)$ computation cost per chain link with $O(\log n)$ cells of storage [13, 33]. However, there are two key differences between the goals of those schemes and our requirements.

(1) These techniques all assume sequential evaluation of the hash chain, whereas in our scheme, authentication attempts are likely to result in an access-pattern containing arbitrary gaps.
(2) Previous schemes aim to minimize the overall time needed to take a single step along the hash chain, which consists of two parts: the time needed to fetch the required value in the hash chain, and the time needed to reposition the checkpoints in preparation for fetching the future values. In our setting, however, it makes sense to minimize only the time needed to fetch the required hash value, potentially at the cost of increasing the time needed to reposition the checkpoints. This is reasonable since the gaps between a user's authentication attempts provide ample time to reposition the checkpoints, and it is the time to generate a password that is actually noticeable to the user.

If the user's login behavior is completely unpredictable, we can minimize the worst-case password generation time by placing the checkpoints at equal distances from one another. We call this the *naïve* checkpointing scheme. However, in many real-world scenarios, user logins follow some pattern that can be exploited to improve upon the naïve scheme.

To model a user's login behavior, we consider a probability distribution that represents the probability that the user will next authenticate at time t (measured in units of time slots) given that it last authenticated at time 0. Additionally, we let each node in the hash chain be indexed by its distance from the tail of the chain and let ℓ be the index of the head of the chain (i.e., ℓ is the length of the remaining part of the hash chain). In this model, valid future login times are the integers $\{1, 2, \ldots, \ell\}$, and each node in the hash chain is indexed by the corresponding login time. By this, we mean that the valid password at time t is the value at node t. This notation is illustrated in Fig. 3.

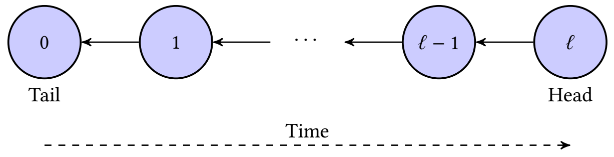


Figure 3: The hash chain with time-labeled nodes.

The problem is then to determine where to place q checkpoints, $0 \leq c_1 \leq c_2 \leq \ldots \leq c_q < \ell$, in order to minimize the expected computation cost of generating a password. We note that if the client authenticates at time t and c_i is the closest checkpoint to t with $c_i \geq t$, then the computational cost of generating the password is $c_i - t$. If no such checkpoint exists, then the cost is $\ell - t$. We do not take into account the number of additional hash computations required to reposition the checkpoints after generating a password.

In order to make the analysis simpler, we relax the model from a "discrete" notion of a hash chain to a "continuous" one. By this, we mean that we make the probability distribution modeling the client's next login time continuous and allow the checkpoints to be stored at any real index in the continuous interval $(0, \ell]$. Additionally, we allow authentications to occur at any real time in $(0, \ell]$. Formally, let $p(t)$ be the probability density function (pdf) of this distribution with support over the positive reals and let $F(t) = \int_0^t p(t)dt$ be its cumulative distribution function (cdf). We can then express the computational cost C in terms of the checkpoints by the formula

$$C = \int_0^{c_1} (c_1 - t)p(t)dt + \int_{c_1}^{c_2} (c_2 - t)p(t)dt + \ldots + \int_{c_q}^{\ell} (\ell - t)p(t)dt$$
$$= c_1 F(c_1) + c_2(F(c_2) - F(c_1)) + \ldots + \ell(F(\ell) - F(c_q)) - \int_0^{\ell} tp(t)dt.$$

In order to determine the values of the c_i's that minimize C, we take the partial derivatives $\frac{\partial C}{\partial c_i}$ for each variable and set them equal to 0. This gives the following system of equations:

$$\frac{F(c_1)}{p(c_1)} = c_2 - c_1 \quad (1)$$
$$\frac{F(c_2) - F(c_1)}{p(c_2)} = c_3 - c_2 \quad (2)$$
$$\vdots \quad (3)$$
$$\frac{F(c_q) - F(c_{q-1})}{p(c_q)} = \ell - c_q \,. \quad (4)$$

Solving these equations yields the values of the c_i's that minimize C, which we then round to the nearest integer, since checkpoints can only be placed at integer coordinates. We refer to this as the *expectation-optimal* solution.

Depending on the specific distribution, this system of equations may or may not be numerically solvable. If necessary, one can simplify the problem by replacing the set of dependent multivariate equations with a set of independent univariate equations. This is done using the following recursive approach. We first place a single checkpoint c optimally in $[0, \ell]$, then place optimal checkpoints in the subintervals $[0, c]$ and $[c, \ell]$, and then place checkpoints in the next set of subintervals, etc. The problem then reduces to the problem of placing a single checkpoint in an interval $[a, b]$, and the optimal location x can be determined by solving the equation

$$\frac{F(x) - F(a)}{p(x)} = b - x \,. \quad (5)$$

In practice, mobile second-factor devices are often not the best environment for running numerical solvers. One solution would be to precompute the expectation-optimal checkpoint positions for some fixed length ℓ (e.g., the initial length of the chain) and distribution F and then hardcode those values into the second-factor application. However, as time progresses, these precomputed positions will no longer be expectation-optimal for the the length of the *remaining* part of the hash chain. Moreover, one might want to adaptively reposition the checkpoints based on the past average time between logins of the user.

Repositioning the checkpoints. Each time a password is generated, we reposition the checkpoints by computing the optimal checkpoint positions for the length of the remaining chain. We then compute the hash values at these positions by traversing the hash chain from the nearest existing checkpoint. This is done in the background after presenting the user with the generated password.

5.1 User logins as a Poisson process

One choice for modeling the distribution $F(t)$ between logins is the exponential distribution

$$p(t) = \lambda e^{-\lambda t} \qquad F(t) = 1 - e^{-\lambda t} \, .$$

The exponential distribution is a distribution of the time between events in a Poisson process, i.e. a process in which events occur continuously and independently at a constant average rate. Previous works state that this is a reasonable model for web login behavior [7, 52]. In our setting, the value of the average time between logins could vary anywhere between hours and months depending on the specific application and whether a second factor is required on every login, once in a period, or once per device.

For the exponential distribution, Equation 5 gives:

$$\frac{-e^{-\lambda x} + e^{-\lambda a}}{\lambda e^{-\lambda x}} = b - x \, .$$

Conveniently, this equation admits the analytic solution

$$x = -\frac{W(e^{\lambda(x-a)+1})}{\lambda} + b + 1/\lambda \, , \tag{6}$$

where $W(\cdot)$ is the Lambert-W function [14]. The recursive solution in this case can then be easily implemented on the second-factor device.

Figure 4 compares the expected performance of the following checkpointing procedures: naïve, expectation-optimal (obtained by numerically solving Equations 1-4) and recursive (obtained using Equation 6). We also compare against the pebbling scheme of Coppersmith and Jakobsson [13], although as we've noted above, their scheme optimizes a different metric than ours, so it is no surprise that it does not perform as well as the recursive or expectation-optimal approaches in our setting.

Balancing worst and expected performance. One disadvantage of both the expectation-optimal and recursive checkpoints is that they perform poorly in the worst-case. Specifically, if a user does not log in for a long period of time, a subsequent login might result in an unacceptably high latency. A simple solution is to place several additional checkpoints in order to minimize the maximal distance between checkpoints, which bounds the worst case number of hash computations.

Figure 5 illustrates the placement of checkpoints given by the different checkpointing schemes discussed in this section plotted along the probability density function of the exponential distribution.

6 IMPLEMENTATION

We implemented our scheme by extending the Google Authenticator Android App and the Google Authenticator Linux Pluggable Authentication Module (PAM) [25].

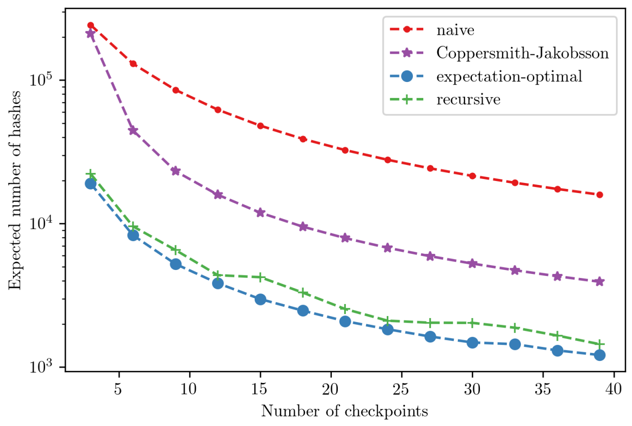


Figure 4: Performance of checkpointing schemes. Chain length is 1.05×10^6 (one year when using 30-second time slots). Login times are assumed to be a Poisson process with mean of 20160 (one week when using 30-second time slots).

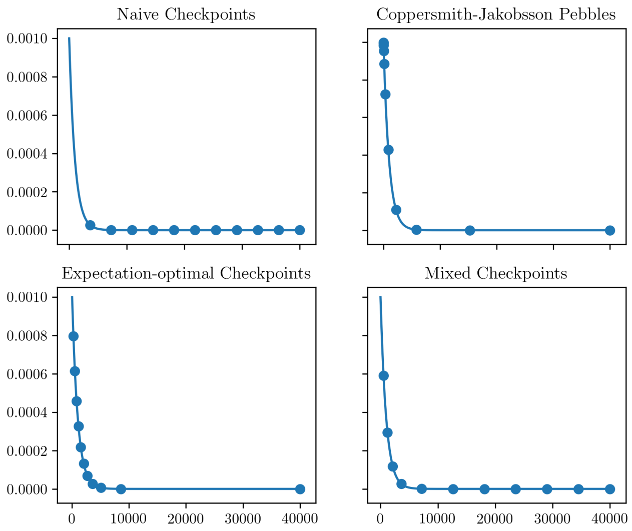
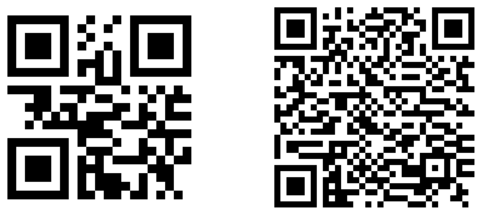


Figure 5: Illustration of different checkpointing schemes with logins modeled by the exponential distribution.

Scheme Details and Parameters. We use passwords of length 130 to obtain the level of security discussed in Section 4. As a concrete instantiation of a family of independent hash functions, for $0 \le i < 2^{32}$, we take $h_i : \{0,1\}^{130} \rightarrow \{0,1\}^{130}$ to be defined as $h_i(x) = \texttt{SHA-256}(\langle i \rangle_{32} \| id \| x)|_{130}$, where $\langle i \rangle_{32}$ is the index of the function represented as a 32-bit binary string, and *id* is a randomly chosen 80-bit salt. Our time-based counter uses time slots of length 30 seconds with 0 being the UNIX epoch. The length of the hash chain has to be chosen to balance the resulting maximal authentication period and the setup time (which is dominated by the time to serially evaluate the entire hash chain). We use 2^{21} as our default hash chain length, resulting in a maximum authentication period



(a) 21x21 QR encoding of a 128-bit OTP **(b) 33x33 QR encoding of a 512-bit signature**

Figure 6: Password encoding using QR codes: T/Key vs. ECDSA signatures

of approximately 2 years and a setup time of less than 15 seconds on a modern mobile phone (see Section 7 for more details).

6.1 Password Encoding

Since the one-time passwords in our scheme are longer than those in the HMAC-based TOTP scheme (130 bits vs. 20 bits), we cannot encode the generated passwords as short numerical codes. Instead, we provide two encodings, which we believe are better suited for passwords of this length.

QR Codes. First, our Android app supports encoding the one-time password as a QR code. Among their many other applications, QR codes have been widely used for second factor authentication to transmit information from the authenticating device to the mobile device. For example, in Google Authenticator, a website presents the user with a QR code containing the shared secret for the TOTP scheme, which the user then scans with her mobile phone, thus providing the authenticator app with the secret. QR codes have also been used for transaction authentication as a communication channel from the insecure device to the secure one [60].

In our scheme, QR codes are used in the authentication process as a communication channel from the secure mobile device to the authenticating device. Such a use case was previously considered by [56] and was shown to be practical [54]. Specifically, our app encodes the 130 bit password as a QR code of size 21×21 modules, which is then displayed to the user. To log in on a different device, the user can then use that device's camera to scan the QR code from the mobile phone's screen. This method is best suited for use on laptops, tablets, and phones, where built-in cameras are ubiquitous, yet it can also be used on desktops with webcams. The QR code password encoding also provides a clear visualization of the relatively short length of our passwords compared to schemes using public key cryptography. For example, the standard ECDSA digital signature scheme [34] with a comparable level of security would result in 512-bit long one-time passwords, which would consequently require larger 33×33 QR codes [62] (a visual comparison appears in Figures 6a and 6b). More recent digital signature constructions [5, 8] could be used to obtain shorter signatures, yet at 384 and 256 bits, respectively, those are still considerably longer than the one-time passwords in our scheme.

Manual Entry. Since our usage of QR codes requires the sign-in device to have a camera, we present an alternative method that can be used for devices without cameras. In these instances, our Android app also encodes one-time passwords using a public word list. Using a word list of 2048 short words (1 to 4 letters), as used in S/Key, results in 12-word passwords, and using a larger 4096 word list (of words up to 6 letters long), results in 11-word passwords. Additionally, more specialized word lists such as those in [9] can be used if word lists that enable autofilling and error correction are desired. These would be particularly useful if the sign-in device was a mobile phone.

Alternatively, it would be possible to generate the one-time passwords as arbitrary strings that the user would then manually enter. Assuming every character in the strings has 6 bits of entropy (which is roughly the case for case-sensitive alphanumeric strings), the resulting one-time passwords would be strings composed of 22 characters. While typing these one-time passwords manually would be cumbersome, they are at least somewhat practical, as opposed to 512 bit/86 character long digital signatures.

Hardware Authentication Devices. USB-based hardware authentication devices, such as Yubikey [69] are often used instead of mobile phone apps for generating TOTP passwords. They offer two main advantages: (i) after the initial setup, the TOTP secret never has to leave the secure hardware, which makes it more secure against client-side malware, and (ii) such authentication devices are capable of emulating a keyboard and can "type" the generated one-time passwords into the relevant password field when the user presses a button on the device. However, hardware tokens do not protect the TOTP secret on the server. Additionally, the registration phase is still susceptible to malware since the TOTP secret needs to be loaded into the hardware token. The newer FIDO U2F protocol [59] addresses these problems, yet it requires specialized support by the browser and two-way communication.

Hardware authentication devices and T/Key could therefore be well-suited for each other: the hardware device would generate the hash chain, store the secret, and provide the server with the initial password. When the user needs to authenticate, the hardware token would traverse the chain and generate the one-time password. T/Key would provide the security against server-side hacks, and the hardware token would provide the security against client-side hacks. Moreover, the ability of the hardware token to automatically "type" the password would address one of T/Key's main disadvantages, namely that the passwords are too long for manual entry.

7 EVALUATION

We evaluated the performance of our scheme to ensure the running times of its different stages are acceptable for a standard authentication scenario. The client Android app was tested on a Samsung Galaxy S7 phone (SM-G930F) with a 2.3 Ghz Quad-Core CPU and 4 GB of RAM. The server side Linux PAM module was tested on a 2.6 Ghz i7-6600 CPU with 4 GB RAM running Ubuntu 16.04.

Our evaluation uses 130-bit passwords and hash chains of length one, two, and four million, corresponding to one-year, two-year, and four-year authentication periods when a new password is generated every 30 seconds. We evaluate the following times:

Table 3: Scheme Performance.
130 bit long passwords, 30 second time slots, 20 mixed checkpoints.

Auth. Period	Mean Time Between Logins	Setup Time (seconds)	Password Generation Time (seconds)		Verification Time (seconds)
			average case	worst case	
1 year	1 week	7.5	0.3	0.6	0.4
2 years	2 weeks	14	0.5	0.9	0.8
4 years	1 month	28	0.8	1.6	1.6

- Client setup time: the time it takes for the mobile phone to first generate the salt and the secret and then traverse the entire hash chain to compute the initial password and create the registration QR code.
- Client password generation time: the time to traverse the chain from the closest checkpoint. We present both the worst-case time, which corresponds to the maximal distance between two checkpoints, as well as the expected time, which we simulate with respect to several typical exponential distributions.
- Server verification time: the time to traverse the entire chain on the server. This captures the longest possible period between logins. In practice, this time will be much shorter if the user logs in regularly.

Results appear in Table 3. In general, we view several seconds as being an acceptable time for the initial setup and a sub-second time as acceptable for both password generation and verification.

We attribute some of the differences between the hash chain traversal time on the server and the traversal time on the phone to the fact that the former was tested using native C code, whereas the latter was run using a Java App on the mobile phone.

8 ATTACKS WITH PREPROCESSING

One limitation of the previously discussed security model is that we do not allow the adversary's algorithm to depend on the choice of the random function h. In practice, however, the function h is *not* a random function, but rather some fixed publicly known function, such as SHA-256. This means that the adversary could perhaps query the function prior to receiving a challenge and store some information about it that could be leveraged later. In this section, we bound the probability of success of such an attack by $(ST/N)^{2/3}$, where $N = 2^n$ is the size of the hash function domain. To mitigate the risk of such attacks, we show that by salting all hash functions with a random salt of length n, we can bound the probability of success by $(T/N)^{2/3}$ (assuming $S \leq N$).

More formally, an inverting attack with preprocessing proceeds as follows:

- First, a pair of algorithms (A_0, A_1) are fixed.
- Second, the function h is sampled from some distribution (e.g., the uniform distribution over all random functions over some set).
- Third, given oracle access to h (which is now fixed), *preprocessing* algorithm A_0 creates an advice string st_h.
- Finally, the *online* algorithm A_1 is given the advice string st_h, oracle access to the same h, and its input $y = h(x)$.

The complexity of an attack in this model is usually measured by the maximal length in bits of the advice string st_h, which is referred to as the "space" of the attack and denoted by S, and the maximal number of oracle queries of the algorithm A_1, which is often referred to as the "time" of the attack and is denoted by T. Note that at least for lower bounds we: (i) allow the preprocessing algorithm an unlimited number of queries to its oracle and (ii) only measure the number of queries made by A_1, ignoring all other computation.

The power of preprocessing was first demonstrated in the seminal work of Hellman [29], who showed that with preprocessing, one-way permutations can be inverted much faster than by brute force. Specifically, Hellman showed that for every one-way permutation $f : [N] \to [N]$ and for every choice of parameters S, T satisfying $T \cdot S \geq N$, there exists an attack with preprocessing which uses space S and time T. Hellman also gave an argument for inverting a random function with time-space tradeoff $T \cdot S^2 \geq N^2$. Subsequently, Fiat and Naor [20] gave an algorithm that works for all functions. The inversion algorithm was further improved when Oechslin [49] introduced rainbow tables and demonstrated how they can be used to break Windows password hashes.

Yao [68] investigated the limits of such attacks and proved that $S \cdot T \geq \Omega(N)$ is in fact necessary to invert a random function on every point in its image. Yao's lower bound was further extended in [17, 23, 63], which showed that attacks that invert a random function with probability ϵ must satisfy $ST \geq \Omega(\epsilon N)$. Recently, Dodis, Guo, and Katz [18] extended these results by proving that the common defense of *salting* is effective in limiting the power of preprocessing in attacks against several common cryptographic primitives. Specifically, for one way functions, they show:

Theorem 8.1 ([18]). *Let $h : [M] \times [N] \to [N]$ be a random function. Let (A_0, A_1) be a pair of algorithms that get oracle access to h such that A_0 outputs an advice string of length S bits, A_1 makes at most T oracle queries, and*

$$\Pr_{h,\, m \in [M], x \in [N]} \left[h\left(m, A_1^h(A_0^h, m, h(m,x))\right) = h(m,x) \right] = \epsilon \,.$$

Then,

$$T\left(1 + \frac{S}{M}\right) \geq \tilde{\Omega}(\epsilon N)\,.$$

The above result can be interpreted as stating that by using a large enough salt space M (e.g., taking $M = N$), one can effectively remove any advantage gained by having an advice string of length

$S \leq N$. Here, we study the potential of using salts to defeat attacks with preprocessing on hash chains.

Let $\mathcal{F}_{M,N}$ denote the uniform distribution over the set of all functions from $[M] \times [N]$ to $[M] \times [N]$ such that for all $f \in \mathcal{F}_{M,N}$ and all $(s, x) \in [M] \times [N]$, $f(s, x) = (s, y)$.

Theorem 8.2. *Let functions $h_1, \ldots, h_k \in \mathcal{F}_{M,N}$ be chosen independently and uniformly at random, where $k = o(\sqrt{N})$. Let (A_0, A_1) be a pair of algorithms that get oracle access to each of the functions $\{h_i\}_{i=1}^k$, such that A_0 outputs an advice string of length S bits, A_1 makes at most T oracle queries, and*

$$\Pr_{\substack{h_1,\ldots,h_k \in \mathcal{F}_{M,N} \\ m \in [M]\, x \in [N]}} \left[h_{[1,k]} \left(m, A^h(A_0^h, h_{[1,k]}(m,x)) \right) = h_{[1,k]}(m,x) \right] = \epsilon\,.$$

Then,

$$T\left(1 + \frac{S}{M}\right) \geq \tilde{\Omega}(\epsilon^{3/2} N).$$

We prove this theorem in Appendix B.

Optimality. We do not know whether the above loss in the dependence on ϵ is optimal. It would be interesting to try to prove a stronger version of the above bound by directly applying the techniques of [23]. Even in the setting of constant ϵ, where one looks for the optimal dependence between S, T and N, we do not know of an attack matching the above bound for arbitrary intermediate values of T and S (apart from the boundary scenarios $T = N$ or $S = \frac{N}{k}$). Rainbow tables [49], which are the best generic attack to invert random functions, give $S\sqrt{2T} = N$. Since a hash chain is not a random function (it has many more collisions in expectation), the expected performance of rainbow tables in our case is far from obvious. For arbitrary (rather than random) functions, the best known attacks [20] have higher complexity $TS^2 = qN^3$, where q is the collision probability of the function. Finding better attacks is an interesting open question.

8.1 Security of T/Key against preprocessing

Within the context of T/Key, Theorem 8.2 leaves a couple of gaps from our goal to make the salted T/Key scheme as secure against attacks with preprocessing as it is secure against attacks without preprocessing. First it has suboptimal dependence on the success probability ϵ. Note that if one only wants to rule out attacks that succeed with constant success probability (say 1/2 or 0.01), then this gap is immaterial in terms of its impact on the security parameters. Second, the theorem currently bounds the probability to invert the entire hash chain, whereas to use it in Attack Game 4.5, one needs to prove a stronger version in which the attacker can invert a chain suffix of his choice. We leave these two gaps as two open problems.

9 RELATED WORK

For a discussion of the many weaknesses of static passwords, see [30]. One-time passwords were introduced by Lamport [38] and later implemented as S/Key [27]. HOTP and TOTP were proposed in [47] and [48], respectively. For a review and comparison of authentication schemes, see [10, 50]. Leveraging trusted handheld devices to improve authentication security was discussed in [2] and [40]. Two-factor authentication schemes were analyzed rigorously in [56], which proposes a suite of efficient protocols with various usability and security tradeoffs.

Online Two-Factor Authentication. A large body of work has been devoted to the online setting, where one allows bidirectional digital communication between the server and the second-factor device [15, 22, 40, 55, 59]. In this setting, secrets on the server can usually be avoided by using public-key cryptography. We especially call the reader's attention to the work of Shirvanian et al. [56], who study multiple QR-based protocols. In one of their schemes, called "LBD-QR-PIN," the mobile device generates a key pair and sends the public key to the server. Subsequently, on each authentication attempt, the server generates a random 128-bit challenge, encrypts it using the client's public key, and sends it to the authenticating device. The authenticating device encodes the challenge as a QR code, which the user then scans using his mobile device. The mobile device decrypts the challenge using its stored private key, computes a short 6 digit hash of the challenge, and presents it to the user. The user then enters this 6 digit code on the authenticating device, which sends it to the server for verification. A big advantage of this scheme lies in the fact that the messages that the client sends are very short and can therefore easily be entered manually by the user.

Hash Chains. For an overview of hash chains and their applications, see [13, 26, 31, 33]. In particular, Hu et al. [31] provide two different constructions of one-way hash chains, the Sandwich-chain and the Comb Skipchain, which enable faster verification. They are less suited for our setting since skipping segments of the chain requires the prover to provide the verifier with additional values (which would result in longer passwords). Goyal [26] proposes a reinitializable hash chain, a hash chain with the property that it can be securely reinitialized when the root is reached. Finally, [13, 33] discuss optimal time-memory tradeoffs for sequential hash chain traversal. On the theoretical side, statistical properties of the composition of random functions were studied as early as [53] and gained prominence in the context of population dynamics in the work of Kingman [37]. The size of the image of a set under the iterated application of a random function was studied by Flajolet and Odlyzko [21] and later in the context of rainbow tables in [1, 49]. The size of the image of a set under compositions of independent random functions was studied by Zubkov and Serov [70, 71], who provide several useful tail bounds, some of which we use in Appendix A.

Attacks with preprocessing. Time-space tradeoffs, which we use as our model in Section 8, were introduced by Hellman [29] and later rigorously studied by Fiat and Naor [20]. The lower bound to invert a function in this model was shown by Yao [68] and, subsequently, extended in [17, 18, 23, 63]. The work of Gennaro and Trevisan [23] was particularly influential due to its introduction of the "compression paradigm" for proving these kinds of lower bounds. More attacks in this model were shown by Bernstein and Lange [6].

10 CONCLUSIONS

We presented a new time-based offline one-time password scheme, T/Key, that has no server secrets. Prior work either was not time-based, as in S/Key, or required secrets to be stored on the server, as in TOTP. We implemented T/Key as a mobile app and showed it performs well, with sub-15 second setup time and sub-second password generation and verification. To speed up the password generation phase, we described a near-optimal algorithm for storing checkpoints on the client, while limiting the amount of required memory. We gave a formal security analysis of T/Key by proving a lower bound on the time needed to break the scheme, which shows it is as secure as the underlying hash function. We showed that by using independent hash functions, as opposed to iterating the same function, we obtain better hardness results and eliminate several security vulnerabilities present in S/Key. Finally, we studied the general question of hash chain security and proved a time-space lower bound on the amount of work needed to invert a hash chain in the random oracle model with preprocessing.

ACKNOWLEDGMENTS

We thank David Mazières for very helpful discussions about this work. This work is supported by NSF, DARPA, a grant from ONR, and the Simons Foundation. Opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of DARPA.

A COLLISIONS IN RANDOM FUNCTIONS

We first need to investigate some statistical properties of compositions of random functions. Starting with the work of Kingman [37], the distribution of the image size $\left|h_{[1,k]}([N])\right|$, and specifically its convergence rate to 1 ([16, 19]), was studied. In our setting, we are more interested in the properties of $h_{[1,k]}$ for moderate values of k, and specifically, we assume $k = o\left(\sqrt{N}\right)$.

Lemma A.1. *Let $k, N \in \mathbb{N}$ such that $k = o\left(\sqrt{N}\right)$. Then,*

$$\mathop{\mathbf{E}}_{h_1,\ldots,h_k}\left[|h_{[1,k]}([N])|\right] = O\left(\frac{N}{k}\right).$$

Proof. A formal proof can be found in [70]. Here, we provide a brief sketch of the argument. Let

$$\alpha_k = \mathop{\mathbf{E}}_{h_1,\ldots,h_k}\left[\left|h_{[1,k]}([N])\right|/N\right].$$

Then,

$$\alpha_{k+1} = \mathop{\mathbf{E}}_{h_{k+1}}\left[|h_{k+1}([\alpha_k N])|/N\right].$$

The last expression can be interpreted as a simple occupancy problem of independently throwing $\alpha_k N$ balls into N bins and reduces to the probability that a bin is not empty:

$$\alpha_{k+1} = 1 - (1 - 1/N)^{\alpha_k N}.$$

For large N, we can make the approximation $(1 - 1/N)^N \approx 1/e$. Substituting this gives

$$\alpha_{k+1} = 1 - e^{-\alpha_k}$$

and Taylor expanding the resulting expression gives the following approximation for the recursive relation:

$$\alpha_{k+1} = 1 - (1 - \alpha_k + \alpha_k^2/2 - O(\alpha_k^3)) = \alpha_k - \alpha_k^2/2 + O(\alpha_k^3).$$

Plugging-in the guess $\alpha_k = 2/k + O(1/k^3)$ into the right hand side gives

$$\begin{aligned}\alpha_{k+1} &= 2/k - 2/k^2 + O(1/k^3) = 2(k-1)/k^2 + O(1/k^3)\\ &= 2/(k+1) - 2/(k^2(k+1)) + O(1/k^3) = 2/(k+1) + O(1/k^3)\end{aligned}$$

and therefore

$$\alpha_k = 2/k + O(1/k^3)$$

satisfies the recursive relation.

□

We also need to estimate the probability that any two points in the domain collide under the hash function. We make use of the following lemma, due to [70], and give a short proof here for completeness.

Lemma A.2 ([70]). *Let $k, N \in \mathbb{N}$ such that $k = o\left(\sqrt{N}\right)$, and let $x, x', x'' \in [N]$ such that $x \neq x' \neq x''$. Then,*

$$\mathop{\mathbf{Pr}}_{h_1,\ldots,h_k}\left[h_{[1,k]}(x) = h_{[1,k]}(x')\right] = \frac{k}{N} - o\left(\tfrac{1}{N}\right)$$

$$\mathop{\mathbf{Pr}}_{h_1,\ldots,h_k}\left[h_{[1,k]}(x) = h_{[1,k]}(x') = h_{[1,k]}(x'')\right] = \frac{k(3k-1+o_N(1))}{2N^2}.$$

Proof. Observe that since $h_1, \ldots, h_k$ are independent, the random variables $h_{[1,i+1]}(x)$ and $h_{[1,i+1]}(x')$ are independent when conditioned on $h_{[1,i]}(x) \neq h_{[1,i]}(x')$. Using this fact gives

$$\begin{aligned}&\mathop{\mathbf{Pr}}_{h_1,\ldots,h_k}\left[h_{[1,k]}(x) \neq h_{[1,k]}(x')\right]\\ &= \prod_{i=1}^{k} \mathop{\mathbf{Pr}}_{h_i}\left[h_{[1,i]}(x) \neq h_{[1,i]}(x') \middle| h_{[1,i-1]}(x) \neq h_{[1,i-1]}(x')\right]\\ &= \prod_{i=1}^{k}\left(1 - \frac{1}{N}\right) = \left(1 - \frac{1}{N}\right)^k\end{aligned}$$

and subsequently,

$$\begin{aligned}\mathop{\mathbf{Pr}}_{h_1,\ldots,h_k}\left[h_{[1,k]}(x) = h_{[1,k]}(x')\right] &= 1 - \left(1 - \frac{1}{N}\right)^k\\ &= 1 - \left(1 - \frac{k}{N} + O\left(\frac{k^2}{N^2}\right)\right) = \frac{k}{N} - o\left(\tfrac{1}{N}\right).\end{aligned}$$

To show the second statement of the lemma, we break down the probability of a 3-collision between x, x', x'' by iterating through the different levels in the hash chain where a collision between x

and x' could occur. We have that

$$\begin{aligned}
&\mathbf{Pr}\left[h_{[1,k]}(x) = h_{[1,k]}(x') = h_{[1,k]}(x'')\right] \\
&= \mathbf{Pr}\left[h_{[1,k]}(x) = h_{[1,k]}(x'') \middle| h_{[1,k]}(x) = h_{[1,k]}(x')\right] \cdot \mathbf{Pr}\left[h_{[1,k]}(x) = h_{[1,k]}(x')\right] \\
&= \sum_{i=0}^{k-1}\Bigg(\mathbf{Pr}\left[h_{[1,k]}(x) = h_{[1,k]}(x'') \middle| \min\left\{i' : h_{[1,i']}(x) = h_{[1,i']}(x')\right\} = i+1\right] \\
&\qquad \cdot \mathbf{Pr}\left[\min\left\{i' : h_{[1,i']}(x) = h_{[1,i']}(x')\right\} = i+1\right]\Bigg) \\
&= \sum_{i=0}^{k-1}\left(1 - \left(1 - \frac{2}{N}\right)^{i} \cdot \left(1 - \frac{1}{N}\right)^{k-i}\right) \cdot \left(1 - \frac{1}{N}\right)^{i} \cdot \frac{1}{N} \\
&= \sum_{i=0}^{k-1}\left(1 - \left(1 - \frac{2i}{N} + o\left(\frac{1}{N}\right)\right) \cdot \left(1 - \frac{k-i}{N} + o\left(\frac{1}{N}\right)\right)\right) \cdot \left(1 - \frac{i}{N} + o\left(\frac{1}{N}\right)\right) \cdot \frac{1}{N} \\
&= \sum_{i=0}^{k-1}\left(\frac{i+k}{N^2} + o\left(\frac{1}{N^2}\right)\right) = \frac{3k^2 - k}{2N^2} + k \cdot o\left(\frac{1}{N^2}\right)
\end{aligned}$$

as desired. □

The next lemma estimates, for any $x \in [N]$, the expected number of preimages of the point $h_{[1,k]}(x)$ and its variance. We use I_A to denote the indicator variable of probability event A.

Lemma A.3. *Let $\{h_i\}_{i=1}^{k} \in \mathcal{F}_N$ be independent random functions, and let $L_j = \sum_{i=1}^{N} I_{h_{[1,k]}(i)=j}$ be a random variable of the number of different preimages under $h_{[1,k]}$ of $j \in [N]$. For every $x \in [N]$,*

$$\underset{h_1,\dots,h_k}{\mathbf{E}}\left[L_{h_{[1,k]}(x)}\right] = k + 1 - o(1)$$
$$\mathbf{Var}_{h_1,\dots,h_k}\left[L_{h_{[1,k]}(x)}\right] = \frac{1}{2}(k+1)^2 .$$

Proof. From the linearity of expectation and the previous lemma, we find that

$$\begin{aligned}
\underset{h_1,\dots,h_k}{\mathbf{E}}\left[L_{h_{[1,k]}(x)}\right] &= \underset{h_1,\dots,h_k}{\mathbf{E}}\left[\sum_{x'=1}^{N} I_{h_{[1,k]}(x)=h_{[1,k]}(x')}\right] \\
&= \sum_{x'=1}^{N} \underset{h_1,\dots,h_k}{\mathbf{E}}\left[I_{h_{[1,k]}(x)=h_{[1,k]}(x')}\right] \\
&= \sum_{x'=1}^{N} \underset{h_1,\dots,h_k}{\mathbf{Pr}}\left[h_{[1,k]}(x) = h_{[1,k]}(x')\right] \\
&= 1 + (N-1) \cdot \left(\frac{k}{N} - o\left(\tfrac{1}{N}\right)\right) = k + 1 - o(1).
\end{aligned}$$

Additionally,

$$\begin{aligned}
&\underset{h_1,\dots,h_k}{\mathbf{E}}\left[L^2_{h_{[1,k]}(x)}\right] \\
&= \underset{h_1,\dots,h_k}{\mathbf{E}}\left[\sum_{x'=1}^{N}\sum_{x''=1}^{N} I_{h_{[1,k]}(x)=h_{[1,k]}(x')} \cdot I_{h_{[1,k]}(x)=h_{[1,k]}(x'')}\right] \\
&= \underset{h_1,\dots,h_k}{\mathbf{E}}\left[\sum_{x',x''=1}^{N} I_{h_{[1,k]}(x)=h_{[1,k]}(x')=h_{[1,k]}(x'')}\right] \\
&= (N-1)(N-2) \cdot \underset{\substack{h_1,\dots,h_k \\ x,x',x'' \text{ different}}}{\mathbf{E}}\left[I_{h_{[1,k]}(x)=h_{[1,k]}(x')=h_{[1,k]}(x'')}\right] \\
&\qquad + 3(N-1) \cdot \underset{\substack{h_1,\dots,h_k \\ x \neq x'}}{\mathbf{E}}\left[I_{h_{[1,k]}(x)=h_{[1,k]}(x')}\right] + 1 \\
&= (N-1)(N-2) \cdot \left(\frac{k(3k - 1 + o_N(1))}{2N^2}\right) + 3(N-1) \cdot \left(\frac{k}{N} - o\left(\tfrac{1}{N}\right)\right) + 1 \\
&= \tfrac{3}{2}k^2 + \left(\tfrac{5}{2} + o_N(1)\right)k + 1
\end{aligned}$$

and thus

$$\begin{aligned}
\mathbf{Var}_{h_1,\dots,h_k}\left[L_{h_{[1,k]}(x)}\right] &= \underset{h_1,\dots,h_k}{\mathbf{E}}\left[L^2_{h_{[1,k]}(x)}\right] - \underset{h_1,\dots,h_k}{\mathbf{E}}\left[L_{h_{[1,k]}(x)}\right]^2 \\
&= \tfrac{1}{2}k^2 + \left(\tfrac{1}{2} + o_N(1)\right)k \leq \tfrac{1}{2}(k+1)^2.
\end{aligned}$$

□

Lemma A.4. *Let $\{h_i\}_{i=1}^{k} \in \mathcal{F}_N$ be independent random functions, and let $L_j = \sum_{i=1}^{N} I_{h_{[1,k]}(i)=j}$ be a random variable of the number of different preimages under $h_{[1,k]}$ of $j \in [N]$. For every $x \in [N]$,*

$$\underset{h_1,\dots,h_k}{\mathbf{Pr}}\left[L_{h_{[1,k]}(x)} \geq \frac{2k}{\sqrt{\epsilon}}\right] \leq \frac{\epsilon}{2}.$$

Proof. Applying Chebyshev's inequality, we obtain

$$\begin{aligned}
&\underset{h_1,\dots,h_k}{\mathbf{Pr}}\left[L_{h_{[1,k]}(x)} \geq \frac{2k}{\sqrt{\epsilon}}\right] \\
&\leq \underset{h_1,\dots,h_k}{\mathbf{Pr}}\left[L_{h_{[1,k]}(x)} \geq (k+1) + \sqrt{\frac{2}{\epsilon}} \cdot \sqrt{\frac{1}{2}}(k+1)\right] \leq \frac{\epsilon}{2}.
\end{aligned}$$ □

Lemma A.5. *Let $\{h_i\}_{i=1}^{k} \in \mathcal{F}_{M,N}$ be independent identically random functions, and let $L_j = \sum_{i=1}^{N} I_{h_{[1,k]}(i)=j}$ be a random variable of the number of different preimages under $h_{[1,k]}$ of $j \in [M] \times [N]$. For every $(s,x) \in [M] \times [N]$,*

$$\underset{h_1,\dots,h_k}{\mathbf{Pr}}\left[L_{h_{[1,k]}(s,x)} \geq \frac{2k}{\sqrt{\epsilon}}\right] \leq \frac{\epsilon}{2}.$$

Proof. Fix $s \in [M]$, and define $h_{i,s}(x)$ to be the last n bits of $h_i(s,x)$. Applying the previous lemma to $\{h_{i,s}\}_{i=1}^{k}$ yields the result. □

B PROOF OF THEOREM 8.2

Our proof reduces the problem of inverting a random function to the problem of inverting a random hash chain.

Let $\mathcal{D}$ be a distribution over functions from $\mathcal{X}$ to $\mathcal{X}$ such that for every $x \in \mathcal{X}$

$$\underset{h_1,\dots,h_k \in \mathcal{D}}{\mathbf{Pr}}\left[L_{h_{[1,k]}(x)} \geq \frac{2k}{\sqrt{\epsilon}}\right] \leq \frac{\epsilon}{2}. \tag{7}$$

Let A be an oracle algorithm, which, given oracle access to $h_1, \ldots, h_k \in \mathcal{D}$ and S bits of advice, makes at most T oracle queries to all of its oracles combined and successfully inverts with probability $\epsilon \in (0,1)$. For any choice of functions $\{h_i\}_{i=1}^k \in \mathcal{D}$, let

$$G_{h_1,\ldots,h_k} = \left\{ x \in \mathcal{X} : \left(A(h_{[1,k]}(x)) = x \right) \wedge \left(L_{h_{[1,k]}(x)} \le \frac{2k}{\sqrt{\epsilon}} \right) \right\}$$

be the set of *good points*, where we say that a point is good if A outputs x when executed on $h_{[1,k]}(x)$, and the point does not have many collisions under $h_{[1,k]}$. Note that the first condition is stronger than the condition that A merely inverts $h_{[1,k]}(x)$. Denote by $h_{[1,k]}(G_{h_1,\ldots,h_k})$ the corresponding set of good images. Observe that the second condition above guarantees that each point in the image $h_{[1,k]}(G_{h_1,\ldots,h_k})$ has at most $\frac{2k}{\sqrt{\epsilon}}$ preimages under $h_{[1,k]}$. Using this observation and a union bound, we conclude that

$$\begin{aligned}
&\Pr_{\substack{h_1,\ldots,h_k \\ x \in \mathcal{X}}} \left[x \in G_{h_1,\ldots,h_k} \right] \ge \frac{\sqrt{\epsilon}}{2k} \cdot \Pr_{\substack{h_1,\ldots,h_k \\ x \in \mathcal{X}}} \left[h_{[1,k]}(x) \in h_{[1,k]}(G_{h_1,\ldots,h_k}) \right] \\
&\ge \frac{\sqrt{\epsilon}}{2k} \cdot \left(\Pr_{\substack{h_1,\ldots,h_k \\ x \in \mathcal{X}}} \left[A(h_{[1,k]}(x)) \in h_{[1,k]}^{-1}(h_{[1,k]}(x)) \right] \right. \\
&\qquad \left. - \Pr_{\substack{h_1,\ldots,h_k \\ x \in \mathcal{X}}} \left[L_{h_{[1,k]}(x)} \ge \frac{2k}{\sqrt{\epsilon}} \right] \right).
\end{aligned}$$

Plugging in the probabilities given by the theorem hypothesis and Equation 7, we obtain

$$\Pr_{\substack{h_1,\ldots,h_k \\ x \in \mathcal{X}}} \left[x \in G_{h_1,\ldots,h_k} \right] \ge \frac{\sqrt{\epsilon}}{2k} \cdot (\epsilon - \tfrac{\epsilon}{2}) \ge \frac{\epsilon^{3/2}}{4k}.$$

For any $i \in [k]$, let $G^i_{h_1,\ldots,h_k}$ be the subset of points in $G_{h_1,\ldots,h_k}$ on which A queries its i-th oracle function at most $\frac{2T}{k}$ times. Note that for every input and every choice of hash functions, the total number of queries is at most T, and so for every input, A queries at least $1/2$ of its oracle functions at most $\frac{2T}{k}$ times. Therefore

$$\begin{aligned}
&\Pr_{\substack{h_1,\ldots,h_k \\ x \in \mathcal{X} \\ i \in [k]}} \left[x \in G^i_{h_1,\ldots,h_k} \right] \\
&= \Pr_{\substack{h_1,\ldots,h_k \\ x \in \mathcal{X} \\ i \in [k]}} \left[x \in G^i_{h_1,\ldots,h_k} \middle| x \in G_{h_1,\ldots,h_k} \right] \cdot \Pr_{\substack{h_1,\ldots,h_k \\ x \in \mathcal{X}}} \left[x \in G_{h_1,\ldots,h_k} \right] \\
&\ge \frac{\epsilon^{3/2}}{8k}.
\end{aligned}$$

Therefore, there exists some fixed index $i^* \in [1,k]$ and some fixed choice of all the other hash functions $h_1, \ldots, h_{i^*-1}, h_{i^*+1}, \ldots, h_k$ that achieves a probability of at least $\frac{\epsilon^{3/2}}{8k}$ over a random h_{i^*} and a random $x \in G_{h_1,\ldots,h_k}$. For every function h, denote by $G^{i^*}_h$ the set $G^{i^*}_{h_1,\ldots,h_k}$ with $h_{i^*} = h$ and the other functions fixed as above. We get

$$\Pr_{\substack{h \in \mathcal{D} \\ x \in \mathcal{X}}} \left[x \in G^{i^*}_h \right] \ge \frac{\epsilon^{3/2}}{8k}. \tag{8}$$

The choice of i^* as well as the explicit description of the functions $h_1, \ldots, h_{i^*-1}, h_{i^*+1}, \ldots, h_k$ can be hard-coded into the algorithm A since Theorem 8.1, and therefore also our reduction, can be arbitrarily non-uniform in the *input size*. Another way of thinking about this is that since our model charges the algorithm only for oracle queries, an algorithm in this model can deterministically determine the best i^* and the remaining functions by simulating A's behavior on all possible inputs (without making any oracle queries).

Consider the following algorithm A' for inverting a random function $h \in \mathcal{D}$. Algorithm A' gets the same S bits of advice as A and is given oracle access to h. On input $z \in \mathcal{X}$, A' computes $y = h_{[i^*+1,k]}(z)$ and then simulates A on y as follows: A' uses its own oracle to answer oracle queries to h_{i^*} and uses the chosen functions $h_1, \ldots, h_{i^*-1}, h_{i^*+1}, \ldots, h_k$ to answer all other oracle queries. Furthermore, A' bounds the number of queries to h by $\frac{2T}{k}$. Thus, if during the simulation A tries to make more than this number of queries to h, algorithm A' aborts. Otherwise, A' obtains $A(y)$ and then computes and outputs $h_{[1,i^*-1]}(A(y))$.

To analyze the success probability of A', the key observation is that if $w \in h_{[1,i^*-1]}(G^{i^*}_h)$, then A' inverts $h(w)$ successfully. To see this, note that if $w = h_{[1,i^*-1]}(x)$ for $x \in G^{i^*}_h$, then A' simulates A on

$$y = h_{[i^*+1,k]}(h(w)) = h_{[1,k]}(x) \in G^{i^*}_h,$$

thus $A(y) = x$, and $A'(h(w)) = h_{[1,i^*-1]}(A(y)) = w$ as desired. Therefore

$$\begin{aligned}
\Pr_{\substack{h \in \mathcal{D} \\ w \in \mathcal{X}}} \left[A'(h(w)) \in h^{-1}(h(w)) \right] &\ge \Pr_{\substack{h \in \mathcal{D} \\ w \in \mathcal{X}}} \left[w \in h_{[1,i^*-1]}(G') \right] \\
&= \Pr_{\substack{h \in \mathcal{D} \\ x \in \mathcal{X}}} \left[x \in G^{i^*}_h \right] \ge \frac{\epsilon^{3/2}}{8k},
\end{aligned}$$

where the penultimate equality holds because $h_{[1,k]}$ and therefore also $h_{[1,i^*-1]}$ have no collisions on $G^{i^*}_h$, and the last inequality follows from Equation 8.

To complete the proof of the theorem, we apply the lower bound given by Theorem 8.1 to algorithm A' and the distribution $S_{M,N}$, which gives

$$\frac{2T}{k} \left(1 + \frac{S}{M} \right) \ge \tilde{\Omega} \left(\frac{\epsilon^{3/2} N}{8k} \right),$$

and therefore

$$T \left(1 + \frac{S}{M} \right) \ge \tilde{\Omega}(\epsilon^{3/2} N)$$

as required. □

REFERENCES

[1] Gildas Avoine, Pascal Junod, and Philippe Oechslin. 2008. Characterization and Improvement of Time-Memory Trade-Off Based on Perfect Tables. *ACM Trans. Inf. Syst. Secur.* 11, 4, Article 17 (July 2008), 22 pages. https://doi.org/10.1145/1380564.1380565

[2] Dirk Balfanz and Edward W. Felten. 1999. Hand-Held Computers Can Be Better Smart Cards. In *Proceedings of the 8th USENIX Security Symposium, Washington, D.C., August 23-26, 1999.* USENIX Association. https://www.usenix.org/conference/8th-usenix-security-symposium/hand-held-computers-can-be-better-smart-cards

[3] Elaine Barker. 2016. *Recommendation for Key Management Part 1: General.* Technical Report. National Institute of Standards and Technology (NIST). https://doi.org/10.6028/nist.sp.800-57pt1r4

[4] Mihir Bellare and Phillip Rogaway. 1993. Random Oracles Are Practical: A Paradigm for Designing Efficient Protocols. In *Proceedings of the 1st ACM Conference on Computer and Communications Security (CCS '93).* ACM, New York, NY, USA, 62–73. https://doi.org/10.1145/168588.168596

[5] Daniel J Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. 2012. High-speed high-security signatures. *Journal of Cryptographic Engineering* 2, 2 (2012), 77–89.

[6] Daniel J. Bernstein and Tanja Lange. 2013. Non-uniform Cracks in the Concrete: The Power of Free Precomputation. In *Advances in Cryptology - ASIACRYPT 2013*. Springer, Berlin, Heidelberg, 321–340. https://doi.org/10.1007/978-3-642-42045-0_17

[7] Jeremiah Blocki, Manuel Blum, and Anupam Datta. 2013. Naturally Rehearsing Passwords. In *Advances in Cryptology - ASIACRYPT 2013*. Springer Berlin Heidelberg, 361–380. https://doi.org/10.1007/978-3-642-42045-0_19

[8] Dan Boneh, Ben Lynn, and Hovav Shacham. 2004. Short Signatures from the Weil Pairing. *Journal of Cryptology* 17, 4 (01 Sep 2004), 297–319. https://doi.org/10.1007/s00145-004-0314-9

[9] Joseph Bonneau. [n. d.]. EFF's New Wordlists for Random Passphrases | Electronic Frontier Foundation. ([n. d.]). Retrieved 08/2017 from https://www.eff.org/deeplinks/2016/07/new-wordlists-random-passphrases

[10] Joseph Bonneau, Cormac Herley, Paul C. van Oorschot, and Frank Stajano. 2012. The Quest to Replace Passwords: A Framework for Comparative Evaluation of Web Authentication Schemes. In *Proceedings of the 2012 IEEE Symposium on Security and Privacy (SP '12)*. IEEE Computer Society, Washington, DC, USA, 553–567. https://doi.org/10.1109/SP.2012.44

[11] Joseph Bonneau and Sören Preibusch. 2010. The Password Thicket: Technical and Market Failures in Human Authentication on the Web. In *The Ninth Workshop on the Economics of Information Security*.

[12] Johannes Buchmann, Erik Dahmen, Sarah Ereth, Andreas Hülsing, and Markus Rückert. 2011. On the Security of the Winternitz One-time Signature Scheme. In *Proceedings of the 4th International Conference on Progress in Cryptology in Africa (AFRICACRYPT'11)*. Springer-Verlag, Berlin, Heidelberg, 363–378. http://dl.acm.org/citation.cfm?id=2026469.2026501

[13] Don Coppersmith and Markus Jakobsson. 2003. Almost Optimal Hash Sequence Traversal. In *Financial Cryptography*. Springer Nature, 102–119. https://doi.org/10.1007/3-540-36504-4_8

[14] R. M. Corless, G. H. Gonnet, D. E. G. Hare, D. J. Jeffrey, and D. E. Knuth. 1996. On the LambertW function. *Advances in Computational Mathematics* 5, 1 (01 Dec 1996), 329–359. https://doi.org/10.1007/BF02124750

[15] Alexei Czeskis, Michael Dietz, Tadayoshi Kohno, Dan Wallach, and Dirk Balfanz. 2012. Strengthening User Authentication Through Opportunistic Cryptographic Identity Assertions. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security (CCS '12)*. ACM, New York, NY, USA, 404–414. https://doi.org/10.1145/2382196.2382240

[16] Avinash Dalal and Eric Schmutz. 2002. Compositions of random functions on a finite set. *The Electronic Journal of Combinatorics* 9, 1 (2002), R26.

[17] Anindya De, Luca Trevisan, and Madhur Tulsiani. 2010. Time Space Tradeoffs for Attacks against One-Way Functions and PRGs. In *Advances in Cryptology – CRYPTO 2010*, Tal Rabin (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 649–665. https://doi.org/10.1007/978-3-642-14623-7_35

[18] Yevgeniy Dodis, Siyao Guo, and Jonathan Katz. 2017. Fixing Cracks in the Concrete: Random Oracles with Auxiliary Input, Revisited. In *Advances in Cryptology – EUROCRYPT 2017*, Jean-Sébastien Coron and Jesper Buus Nielsen (Eds.). Springer International Publishing, Cham, 473–495. https://doi.org/10.1007/978-3-319-56614-6_16

[19] Peter Donnelly. 1991. Weak Convergence to a Markov Chain with an Entrance Boundary: Ancestral Processes in Population Genetics. *The Annals of Probability* 19, 3 (jul 1991), 1102–1117. https://doi.org/10.1214/aop/1176990336

[20] Amos Fiat and Moni Naor. 1991. Rigorous Time/Space Tradeoffs for Inverting Functions. In *Proceedings of the Twenty-third Annual ACM Symposium on Theory of Computing (STOC '91)*. ACM, New York, NY, USA, 534–541. https://doi.org/10.1145/103418.103473

[21] Philippe Flajolet and Andrew M. Odlyzko. 1990. *Random Mapping Statistics*. Springer Berlin Heidelberg, Berlin, Heidelberg, 329–354. https://doi.org/10.1007/3-540-46885-4_34

[22] Scott Garriss, Rámon Cáceres, Stefan Berger, Reiner Sailer, Leendert van Doorn, and Xiaolan Zhang. 2008. Trustworthy and Personalized Computing on Public Kiosks. In *Proceedings of the 6th International Conference on Mobile Systems, Applications, and Services (MobiSys '08)*. ACM, New York, NY, USA, 199–210. https://doi.org/10.1145/1378600.1378623

[23] R. Gennaro and L. Trevisan. 2000. Lower Bounds on the Efficiency of Generic Cryptographic Constructions. In *Proceedings of the 41st Annual Symposium on Foundations of Computer Science (FOCS '00)*. IEEE Computer Society, Washington, DC, USA, 305–. http://dl.acm.org/citation.cfm?id=795666.796554

[24] Google. [n. d.]. 2-Step Verification - Google Account Help. ([n. d.]). Retrieved 08/2017 from https://support.google.com/accounts/topic/7189195

[25] Google. [n. d.]. google-authenticator: Open source version of Google Authenticator (except the Android app). ([n. d.]). Retrieved 08/2017 from https://github.com/google/google-authenticator

[26] Vipul Goyal. 2004. How To Re-initialize a Hash Chain. Cryptology ePrint Archive, Report 2004/097. (2004). http://eprint.iacr.org/2004/097.

[27] N. Haller. 1995. *The S/KEY One-Time Password System*. RFC 1760. Internet Engineering Task Force. 1–12 pages. https://doi.org/10.17487/RFC1760

[28] Johan Håstad and Mats Näslund. 2007. Practical Construction and Analysis of Pseudo-Randomness Primitives. *Journal of Cryptology* 21, 1 (sep 2007), 1–26. https://doi.org/10.1007/s00145-007-9009-3

[29] Martin Hellman. 1980. A cryptanalytic time-memory trade-off. *IEEE transactions on Information Theory* 26, 4 (1980), 401–406.

[30] Cormac Herley and Paul Van Oorschot. 2012. A research agenda acknowledging the persistence of passwords. *IEEE Security & Privacy* 10, 1 (2012), 28–36.

[31] Yih-Chun Hu, Markus Jakobsson, and Adrian Perrig. 2005. Efficient Constructions for One-way Hash Chains. In *Proceedings of the Third International Conference on Applied Cryptography and Network Security (ACNS'05)*. Springer-Verlag, Berlin, Heidelberg, 423–441. https://doi.org/10.1007/11496137_29

[32] Andreas Hülsing, D Butin, S Gazdag, and A Mohaisen. 2015. XMSS: Extended hash-based signatures. Crypto Forum Research Group Internet-Draft. (2015). draft-irtf-cfrg-xmss-hash-based-signatures-01.

[33] M. Jakobsson. 2002. Fractal hash sequence representation and traversal. In *Proceedings IEEE International Symposium on Information Theory,*. Institute of Electrical and Electronics Engineers (IEEE). https://doi.org/10.1109/isit.2002.1023709

[34] Don Johnson, Alfred Menezes, and Scott Vanstone. 2001. The elliptic curve digital signature algorithm (ECDSA). *International Journal of Information Security* 1, 1 (2001), 36–63.

[35] S. Josefsson and I. Liusvaara. 2017. *Edwards-Curve Digital Signature Algorithm (EdDSA)*. RFC 8032. RFC Editor.

[36] Jonathan Katz. 2016. Analysis of a Proposed Hash-Based Signature Standard. In *Security Standardisation Research: Third International Conference, SSR 2016, Gaithersburg, MD, USA, December 5–6, 2016, Proceedings*, Lidong Chen, David McGrew, and Chris Mitchell (Eds.). Springer International Publishing, Cham, 261–273. https://doi.org/10.1007/978-3-319-49100-4_12

[37] John Frank Charles Kingman. 1982. The coalescent. *Stochastic processes and their applications* 13, 3 (1982), 235–248. https://doi.org/10.1016/0304-4149(82)90011-4

[38] Leslie Lamport. 1981. Password Authentication with Insecure Communication. *Commun. ACM* 24, 11 (Nov. 1981), 770–772. https://doi.org/10.1145/358790.358797

[39] Frank T Leighton and Silvio Micali. 1995. Large provably fast and secure digital signature schemes based on secure hash functions. (1995). US Patent 5,432,852.

[40] Mohammad Mannan and P. C. van Oorschot. 2007. Using a Personal Device to Strengthen Password Authentication from an Untrusted Computer. In *Financial Cryptography and Data Security: 11th International Conference, FC 2007, and 1st International Workshop on Usable Security, USEC 2007, Scarborough, Trinidad and Tobago, February 12-16, 2007. Revised Selected Papers*, Sven Dietrich and Rachna Dhamija (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 88–103. https://doi.org/10.1007/978-3-540-77366-5_11

[41] Jonathan M. McCune, Adrian Perrig, and Michael K. Reiter. 2009. Seeing-Is-Believing: using camera phones for human-verifiable authentication. *IJSN* 4, 1/2 (2009), 43–56. https://doi.org/10.1504/IJSN.2009.023425

[42] Daniel L. McDonald, Randall J. Atkinson, and Craig Metz. 1995. One Time Passwords in Everything (OPIE): Experiences with Building and Using Stronger Authentication. In *Proceedings of the 5th Conference on USENIX UNIX Security Symposium - Volume 5 (SSYM'95)*. USENIX Association, Berkeley, CA, USA, 16–16. http://dl.acm.org/citation.cfm?id=1267591.1267607

[43] David McGrew, Michael Curcio, and Scott Fluhrer. 2017. *Hash-Based Signatures*. Internet-Draft draft-mcgrew-hash-sigs-07. IETF Secretariat. http://www.ietf.org/internet-drafts/draft-mcgrew-hash-sigs-07.txt http://www.ietf.org/internet-drafts/draft-mcgrew-hash-sigs-07.txt.

[44] Ralph Charles Merkle. 1979. *Secrecy, Authentication, and Public Key Systems*. Ph.D. Dissertation. Stanford University, Stanford, CA, USA. AAI8001972.

[45] Chris J. Mitchell and Liqun Chen. 1996. Comments on the S/KEY User Authentication Scheme. *SIGOPS Oper. Syst. Rev.* 30, 4 (Oct. 1996), 12–16. https://doi.org/10.1145/240799.240801

[46] Robert Morris and Ken Thompson. 1979. Password security: A case history. *Commun. ACM* 22, 11 (1979), 594–597.

[47] D. M'Raihi, M. Bellare, F. Hoornaert, D. Naccache, and O. Ranen. 2005. *HOTP: An HMAC-Based One-Time Password Algorithm*. RFC 4226. Internet Engineering Task Force. https://doi.org/10.17487/rfc4226

[48] D. M'Raihi, S. Machani, M. Pei, and J. Rydell. 2011. *TOTP: Time-Based One-Time Password Algorithm*. RFC 6238. Internet Engineering Task Force. https://doi.org/10.17487/rfc6238

[49] Philippe Oechslin. 2003. Making a Faster Cryptanalytic Time-Memory Trade-Off. In *Advances in Cryptology - CRYPTO 2003*, Dan Boneh (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 617–630. https://doi.org/10.1007/978-3-540-45146-4_36

[50] Lawrence O'Gorman. 2003. Comparing passwords, tokens, and biometrics for user authentication. *Proc. IEEE* 91, 12 (2003), 2021–2040.

[51] QRStuff. [n. d.]. What Size Should A Printed QR Code Be? ([n. d.]). Retrieved 08/2017 from https://blog.qrstuff.com/2011/01/18/what-size-should-a-qr-code-be

DocB1

[52] Georg Rasch. 1963. The poisson process as a model for a diversity of behavioral phenomena. In *International congress of psychology*, Vol. 2. American Psychological Association (APA), 2. https://doi.org/10.1037/e685262012-108

[53] Herman Rubin and Rosedith Sitgreaves. 1954. *Probability Distributions Related to Random Transformations of a Finite Set.* Technical Report 19A. Applied Mathematics and Statistics Laboratory, Stanford University. https://statistics.stanford.edu/sites/default/files/SOL%20ONR%2019A.pdf

[54] SecurEnvoy. [n. d.]. SecurEnvoy Overview Presentation. ([n. d.]). Retrieved 08/2017 from https://www.securenvoy.com/animations/overview/animations.shtm/#oneswipe

[55] Duo Security. [n. d.]. Using the Duo Prompt - Guide to Two-Factor Authentication. ([n. d.]). Retrieved 08/2017 from https://guide.duo.com/prompt

[56] Maliheh Shirvanian, Stanislaw Jarecki, Nitesh Saxena, and Naveen Nathan. 2014. Two-Factor Authentication Resilient to Server Compromise Using Mix-Bandwidth Devices. In *Proceedings 2014 Network and Distributed System Security Symposium*. Internet Society. https://doi.org/10.14722/ndss.2014.23167

[57] Victor Shoup. 1999. On the Security of a Practical Identification Scheme. *Journal of Cryptology* 12, 4 (sep 1999), 247–260. https://doi.org/10.1007/s001459900056

[58] Bluetooth SIG. 2016. Bluetooth core specification version 5.0. (2016). https://www.bluetooth.org/DocMan/handlers/DownloadDoc.ashx?doc_id=421043

[59] Sampath Srinivas, Dirk Balfanz, Eric Tiffany, and FIDO Alliance. 2015. Universal 2nd factor (U2F) overview. (2015). https://fidoalliance.org/specs/fido-u2f-v1.2-ps-20170411/fido-u2f-overview-v1.2-ps-20170411.html

[60] G. Starnberger, L. Froihofer, and K. M. Goeschka. 2009. QR-TAN: Secure Mobile Transaction Authentication. In *2009 International Conference on Availability, Reliability and Security*. IEEE, 578–583. https://doi.org/10.1109/ARES.2009.96

[61] The New York Times. 2011. Security Firm Offers to Replace Tokens After Attack. (June 2011). Retrieved 08/2017 from http://www.nytimes.com/2011/06/07/technology/07hack.html?_r=1

[62] Denso Wave. [n. d.]. Information capacity and versions of QR Code. ([n. d.]). Retrieved 08/2017 from http://www.qrcode.com/en/about/version.html

[63] Hoeteck Wee. 2005. On Obfuscating Point Functions. In *Proceedings of the Thirty-seventh Annual ACM Symposium on Theory of Computing (STOC '05)*. ACM, New York, NY, USA, 523–532. https://doi.org/10.1145/1060590.1060669

[64] Wikipedia. [n. d.]. Ashley Madison data breach. ([n. d.]). Retrieved 08/2017 from https://en.wikipedia.org/wiki/Ashley_Madison_data_breach

[65] Wikipedia. [n. d.]. iCloud leaks of celebrity photos. ([n. d.]). Retrieved 08/2017 from https://en.wikipedia.org/wiki/ICloud_leaks_of_celebrity_photos

[66] Wikipedia. [n. d.]. Yahoo! data breaches - Wikipedia. ([n. d.]). Retrieved 08/2017 from https://en.wikipedia.org//wiki/Yahoo!_data_breaches

[67] Min Wu, Simson Garfinkel, and Rob Miller. 2004. Secure web authentication with mobile phones. In *DIMACS workshop on usable privacy and security software*, Vol. 2010. https://www.researchgate.net/profile/Simson_Garfinkel/publication/228856911_Secure_web_authentication_with_mobile_phones/links/550de6d20cf27526109c831d.pdf

[68] A. C.-C. Yao. 1990. Coherent Functions and Program Checkers. In *Proceedings of the Twenty-second Annual ACM Symposium on Theory of Computing (STOC '90)*. ACM, New York, NY, USA, 84–94. https://doi.org/10.1145/100216.100226

[69] Yubico. [n. d.]. Trust the Net with YubiKey Strong Two-Factor Authentication. ([n. d.]). Retrieved 08/25/2017 from https://www.yubico.com/

[70] Andrey M Zubkov and Aleksandr A Serov. 2015. Images of subset of finite set under iterations of random mappings. *Discrete Mathematics and Applications* 25, 3 (2015), 179–185. https://doi.org/10.1515/dma-2015-0017

[71] Andrey M Zubkov and Aleksandr A Serov. 2017. Limit theorem for the image size of a subset under compositions of random mappings. *Discrete Mathematics and Applications* 29, 1 (2017), 17–26. https://doi.org/10.4213/dm1403

SuB

Freedom1

9/2/2020

STARGAte1