her correct hardened password and thus, e g , might be unable to log in The most common circumstance in which this could happen is if the user attempts to log in using a different style keyboard than her regular one, which can cause a dramatic change in the user's typing patterns. In light of this, applications for which our scheme is ideally suited are access to virtual private networks from laptop computers, and file or disk encryption on laptop computers Laptops provide a single, persistently available keyboard at which the user can type her password, which is the ideal situation for repeated generation of her hardened password Moreover, with the rising rate of laptop thefts (e g , see [22]), these applications demand security better than that provided by traditional passwords

2 Related work

The motivation for using keystroke features to harden passwords comes from years of research validating the hypothesis that user keystroke features both are highly repeatable and different between users (e g , [6, 28, 14, 15, 1, 9, 20, 24]). Prior work has anticipated utilizing keystroke information in the user login process (e g , [9]), and indeed products implementing this are being marketed today (e g , see `http://www.biopassword.com/`) All such prior schemes work by storing a model of user keystroke behavior in the system, and then comparing user keystroke behavior during password entry to this model Thus, while they are useful to defend against an online attacker who attempts to log into the system directly, they provide no additional protection against an offline attacker who captures system information related to user authentication and then conducts an offline dictionary attack to find the password (e.g , to then decrypt files encrypted under the password). On the contrary, the captured model of the legitimate user's keystroke behavior can leak information about the password to such an attacker, as discussed in Section 1 Thus, our work improves on these schemes in two ways. First, our method is the first to offer stronger security against *both* online and offline attackers. Second, our scheme is the first to generate a repeatable secret based on the password and keystroke dynamics that is stronger than the password itself and that can be used in applications other than login, such as file encryption

The only work of which we are aware that previously proposed generating a repeatable key based on biometric information is [3] In this scheme, a user carries a portable storage device containing (i) error correcting parameters to decode readings of the biometric (e.g , an iris scan) with a limited number of errors to a "canonical" reading for that user, and (ii) a one-way hash of that canonical reading for verification purposes Moreover, they further proposed a scheme in which the canonical biometric reading for that user is hashed together with a password Their techniques, however, are inappropriate for our goals because the stored error correcting parameters, if captured, reveal information about the canonical form of the biometric for the user. For this reason, their approach requires a biometric with substantial entropy. e g , they considered iris scans offering an estimated 173 bits of entropy, so that the remaining entropy after exposure of the error correcting parameters (they estimated 147 bits of remaining entropy) was still sufficiently large for their application. In our case, the measurable keystroke features for an 8-character password are relatively few (at most 15 on standard keyboards), and indeed in our scheme, the password's entropy will generally dominate the entropy available from keystroke features. Thus, exposing error-correcting parameters in our setting would substantially diminish the available entropy from keystroke features, almost to the point of negating their utility Moreover, exposing information about the keystroke features can, in turn, expose information about the password itself (as discussed in Section 1) This makes the careful utilization of keystroke features critical in our setting, whereas in their setting, the biometrics they considered were presumed independent of the password chosen.

Our method to harden user passwords has conceptual similarities to password "salting" for user login Salting is a method in which the user's password is prepended with a random number (the "salt") of s bits in length before hashing the password and comparing the result to a previously stored value [21, 16] As a result, the search space of an attacker is increased by a factor of 2^s if the attacker does not have access to the salts. However, the correct salt either must be stored in the system or found by exhaustive search at login time Intuitively, the scheme that we propose in this paper can be used to improve this approach, by determining some or all of the salt bits using the user's typing features. In addition, an advantage of our approach over salting is that our scheme can be effective against an online attacker who learns the legitimate user's password (e.g , by observing the user type it) and who then attempts to log in as that user.

Finally, we note that several other research efforts on password security have focused on detecting the unauthorized modification of system information related to password authentication (e g , the attacker adds a new account with a password it knows, or changes the password of an existing account) [13, 12, 8] Here we do not focus on this threat model, though our hardened passwords can be directly combined with these techniques to provide security against this attacker, as well

3 Preliminaries

The hardened passwords generated in our scheme have many potential uses, including user login, file encryption, and authentication to virtual private networks However, for concreteness of exposition, in the rest of this paper we focus on the generation and use of hardened passwords for the purposes of user login Extending our discussion to these other applications is straightforward.

We assume a computer system with a set A of user accounts Access to each user account is regulated by a login program that challenges the user for an account name and password. Using the user's input and some stored information for the account a that the user is trying to access, the login program either accepts or rejects the attempt to log into a. Like in computer systems today, the characters that the user types into the password field are a factor in the determination to accept or reject the login. For the rest of this paper, we denote by pwd_a the correct string of characters for the password field when logging into account a. That is, pwd_a denotes the correct text password as typically used in computer systems today.

In our architecture, typing pwd_a is necessary but not sufficient to access a. Rather, the login program combines the characters typed in the password field with keystroke features to form a hardened password that is tested to determine whether login is successful. The correct hardened password for account a is denoted hpwd_a. The login program will fail to generate hpwd_a if either something other than pwd_a is entered in the password field or if the user's

typing patterns significantly differ from the typing patterns displayed in previous successful logins to the account Here we present our scheme in a way that maintains hpwd_a constant across logins, even despite gradual shifts in the user's typing patterns, so that hpwd_a can also be used for longer-term purposes (e g , file encryption) However, our scheme can be easily tuned to change hpwd_a after each successful login, if desired

3.1 Features

In order to generate hpwd_a from pwd_a and the (legitimate) user's typing patterns, the login program measures a set of features whenever a user types a password Empirically we will examine the use of keystroke duration and latency between keystrokes as features of interest, but other features (e g , force of keystrokes) could be used if they can be measured by the login program. Abstractly, we represent a feature by a function $\phi \quad A \times \mathbb{N} \to \mathbb{R}^+$ where $\phi(a, \ell)$ is the measurement of that feature during the ℓ-th (successful or unsuccessful) login attempt to account a For example, if the feature ϕ denotes the latency between the first and second keystrokes, then $\phi(a, 6)$ is that latency on the sixth attempt to log into a Let m denote the number of features that are measured during logins, and let $\phi_1, \ldots, \phi_m$ denote their respective functions.

Central to our scheme is the notion of a *distinguishing feature*. For each feature ϕ_i, let $t_i \in \mathbb{R}^+$ be a fixed parameter of the system Also, let μ_{a_i} and σ_{a_i} be the mean and standard deviation of the measurements $\phi_i(a, j_1), \quad , \phi_i(a, j_h)$ where $j_1, \quad ., j_h$ are the last h successful logins to the account a and $h \in \mathbb{N}$ is a fixed parameter of the system We say that ϕ_i is a distinguishing feature for the account (after these last h successful logins) if $|\mu_{a_i} - t_i| > k\sigma_{a_i}$ where $k \in \mathbb{R}^+$ is a parameter of the system. If ϕ_i is a distinguishing feature for the account a, then either $t_i > \mu_{a_i} + k\sigma_{a_i}$, i e., the user consistently measures below t_i on this feature, or $t_i < \mu_{a_i} - k\sigma_{a_i}$, i.e , the user consistently measures above t_i on this feature

3.2 Security goals

In our login architecture, the system stores information per account that is accessed by the login program to verify attempts to log in. This information is necessarily based on pwd_a and hpwd_a, but will not include either of these values themselves This is similar to Unix systems, for example, where the `/etc/passwd` file contains the salt for that password and the result of encrypting a fixed string with a key generated from the password and salt In our login architecture, the information stored per account will be more extensive but will still be relatively small

The primary attacker with which we are concerned is an "offline" attacker who captures this information stored in the system, and then uses this information in an offline effort to find hpwd_a (and pwd_a) A first and basic requirement is that any such attack be at least as difficult as exhaustively searching for pwd_a in a traditional Unix setting where the attacker has `/etc/passwd`. In particular, if the user chooses pwd_a to be difficult for an attacker to find using a dictionary attack, then hpwd_a will be at least as secure in our scheme

A more ambitious goal of our scheme is to increase the work that the attacker must undertake by a considerable amount even if pwd_a is chosen poorly, i.e , in a way that is susceptible to a dictionary attack. The amount of additional work that the attacker must undertake in our scheme generally grows with the number of distinguishing features for the account (when the attacker captured the system information) On one extreme, if there are no distinguishing features for the account, then the attacker can find pwd_a and hpwd_a in roughly the same amount of time as the attacker would take to find pwd_a in a traditional Unix setting. On the other extreme, if all m features are distinguishing for the account, then the attacker's task can be slowed by a multiplicative factor up to 2^m. In Section 7, we describe an empirical analysis that sheds light on what this slowdown factor is likely to be in practice. In addition, we show how our scheme can be combined with salting techniques, and so the slowdown factor that our scheme achieves is over and above any benefits that salting offers.

A second attacker that we defend against with our scheme is an "online" attacker who learns pwd_a (e g., by observing it being typed in) and then attempts to log in using it Our scheme makes this no easier and typically harder for this attacker to succeed in logging in.

4 Overview

In this section we give an overview of our technique for generating hpwd_a from pwd_a and user keystroke features When the account a is initialized, the initialization program chooses the value of hpwd_a at random from $\mathbb{Z}_q$, where q is a fixed, sufficiently large prime number, e.g , a q of length 160 bits should suffice The initialization program then creates $2m$ shares $\{s_i^0, s_i^1\}_{1 \le i \le m}$ of hpwd_a using a secret sharing scheme such that for any $b \in \{0,1\}^m$, the shares $\{s_i^{b(i)}\}_{1 \le i \le m}$ can be used to reconstruct hpwd_a (Here, $b(i)$ is the i-th bit of b.) These shares are arranged in an "instruction table".

	$< t_i$	$\geq t_i$
1	s_1^0	s_1^1
2	s_2^0	s_2^1
$\vdots$	$\vdots$	$\vdots$
m	s_m^0	s_m^1

The initialization program encrypts each element of both columns (i.e , the "$< t_i$" and "$\geq t_i$" columns) with pwd_a This (encrypted) table is stored in the system. In the ℓ-th login attempt to a, the login program uses the entered password text pwd' to decrypt the elements of the table, which will result in the previously stored values only if $\mathsf{pwd}_a = \mathsf{pwd}'$. For each feature ϕ_i, the value of $\phi_i(a, \ell)$ indicates which of the two values in the i-th row should be used in the reconstruction to find hpwd_a: if $\phi_i(a, \ell) < t_i$, then the value in the first column is used, and otherwise the value in the second column is used. In the first logins after initialization, the value in either the first or second column works equally well However, as distinguishing features ϕ_i for this account develop over time, the login program perturbs the value in the second column of row i if $\mu_{a_i} < t_i$ and perturbs the value in the first column of row i otherwise. So, the reconstruction to find hpwd_a in the future will succeed only when future measurements of features are consistent with the user's previous distinguished features.

In this way, our scheme helps defend against an online attacker who learns (or tries to guess) pwd_a and then attempts to log into a using it. Unless this attacker can mimic the legitimate user's keystroke behavior for the account's distinguishing features, the attacker will fail in logging into the account Moreover, numerous prior studies have shown that

keystroke dynamics tend to differ significantly from user to user (see Section 2), and so typically the online attacker will fail in his attempts to log into a. Thus, the security analysis in the rest of this paper will focus on the offline attacker.

Not any secret sharing scheme satisfying the properties described above will suffice for our technique, since to defend against an offline attacker, the shares must be of a form that does not easily reveal if a guessed password pwd′ successfully decrypts the table. In the following sections, we present instances of our technique using two different sharing schemes.

Our scheme can be easily combined with salting to further improve security. A natural place to include a salt is in the validation of hpwd_a just after reconstructing it. For example, when hpwd_a is generated during a login, it could be prepended with a salt before hashing it and testing against a previously stored hash value. The salt can be stored as is typically done today, or may not be stored so that the system must exhaustively search for it [16]. In this case, the extra salt results in an additional work factor that the offline attacker must overcome.

5 An instance using polynomials

In this section, we describe an instance of the technique of Section 4 using Shamir's secret sharing scheme [25]. In this scheme, hpwd_a is shared by choosing a random polynomial $f_a \in \mathbb{Z}_q[x]$ of degree $m-1$ such that $f_a(0) = \mathsf{hpwd}_a$. The shares are points on this polynomial. We present the method in two steps, by first describing a simpler variation and then extending it in Section 5.4 to be more secure against an offline attack.

5.1 Stored data structures and initialization

Let G be a pseudorandom function family [23] such that for any key K and any input x, $G_K(x)$ is a pseudorandom element of $\mathbb{Z}_q^*$.[1] In practice, a likely implementation of G would be $G_K(x) = F(K,x)$ where F is a one-way function, e.g., SHA-1 [26]. There are two data structures stored in the system per account.

- An *instruction table* that contains "instructions" regarding how feature measurements are to be used to generate hpwd_a. More specifically, this instruction table contains an entry of the form $<i, \alpha_{ai}, \beta_{ai}>$ for each feature ϕ_i. Here,

$$\begin{aligned} \alpha_{ai} &= y_{ai}^0 \cdot G_{\mathsf{pwd}_a}(2i) \bmod q \\ \beta_{ai} &= y_{ai}^1 \cdot G_{\mathsf{pwd}_a}(2i+1) \bmod q \end{aligned}$$

  and y_{ai}^0, y_{ai}^1 are elements of $\mathbb{Z}_q^*$. Initially (i.e., when the user first chooses pwd_a), all $2m$ values $\{y_{ai}^0, y_{ai}^1\}_{1 \le i \le m}$ are chosen such that all the points $\{(2i, y_{ai}^0), (2i+1, y_{ai}^1)\}_{1 \le i \le m}$ lie on a single, random polynomial $f_a \in \mathbb{Z}_q[x]$ of degree $m-1$ such that $f_a(0) = \mathsf{hpwd}_a$.

- An encrypted, constant-size *history file* that contains the measurements for all features over the last h successful logins to a for some fixed parameter h. More specifically, if since the last time pwd_a was changed, the login attempts $j_1, \ldots, j_\ell$ to a were successful, then this file contains $\phi_i(a,j)$ for each $1 \le i \le m$ and $j \in \{j_{\ell-h+1}, \ldots, j_\ell\}$. In addition, enough redundancy is added to this file so that when it is decrypted with the key under which it was previously encrypted, the fact that the file decrypted successfully can be recognized.

  This file is initialized with all values set to 0, and then is encrypted with hpwd_a using a symmetric cipher. The size of this file should remain constant over time (e.g., must be padded out when necessary), so that its size yields no information about how many successful logins there have been.

[1] That is, a polynomially-bounded adversary not knowing K cannot distinguish between $G_K(x)$ and a randomly chosen element of $\mathbb{Z}_q^*$, even if he is first allowed to examine $G_K(\hat{x})$ for many $\hat{x}$'s of his choice and is allowed to even pick x (as long as it is different from every $\hat{x}$ he previously asked about).

5.2 Logging in

The login program takes the following steps whenever the user attempts to log into a. Suppose that this is the ℓ-th attempt to log into a, and let pwd′ denote the sequence of characters that the user typed. The login program takes the following steps.

1. For each ϕ_i, the login program uses pwd′ to "decrypt" α_{ai} if $\phi_i(a,\ell) < t_i$, and uses pwd′ to "decrypt" β_{ai} otherwise. Specifically, it assigns

$$(x_i, y_i) = \begin{cases} (2i,\ \alpha_{ai}\ G_{\mathsf{pwd}'}(2i)^{-1} \bmod q) & \text{if } \phi_i(a,\ell) < t_i \\ (2i+1,\ \beta_{ai}\ G_{\mathsf{pwd}'}(2i+1)^{-1} \bmod q) & \text{if } \phi_i(a,\ell) \ge t_i \end{cases}$$

   The login program now holds m points $\{(x_i, y_i)\}_{1 \le i \le m}$.

2. The login program sets

$$\mathsf{hpwd}' = \sum_{i=1}^{m} y_i \cdot \lambda_i \bmod q$$

   where

$$\lambda_i = \prod_{1 \le j \le m, j \ne i} \frac{x_j}{x_j - x_i}$$

   is the standard Lagrange coefficient for interpolation (e.g., see [19, p. 526]). It then decrypts the history file using hpwd′. If this decryption yields a properly-formed plaintext history file, then the login is deemed successful. (If the login were deemed unsuccessful, then the login procedure would halt here.)

3. The login program updates the data in the history file, computes the standard deviation σ_{ai} and mean μ_{ai} for each feature ϕ_i over the last h successful logins to a, encrypts the new history file with hpwd′ (i.e., hpwd_a), and overwrites the old history file with this new encrypted history file.[2]

4. The login program generates a new random polynomial $f_a \in \mathbb{Z}_q[x]$ of degree $m-1$ such that $f_a(0) = \mathsf{hpwd}'$.

5. For each distinguishing feature ϕ_i, i.e., $|\mu_{ai} - t_i| > k\sigma_{ai}$, the login program chooses new random values $y_{ai}^0, y_{ai}^1 \in \mathbb{Z}_q^*$ subject to the following constraints:

$$\begin{aligned} \mu_{ai} < t_i &\Rightarrow f_a(2i) = y_{ai}^0 \wedge f_a(2i+1) \ne y_{ai}^1 \\ \mu_{ai} \ge t_i &\Rightarrow f_a(2i) \ne y_{ai}^0 \wedge f_a(2i+1) = y_{ai}^1 \end{aligned}$$

[2] For maximum security, this and the previous step should be performed without writing the plaintext history file to disk. Rather, the login program should hold the plaintext history in volatile storage only.

For all other features ϕ_i—i e , those for which $|\mu_{a_i} - t_i| \leq k\sigma_{a_i}$, or all features if there have been fewer than h successful logins to this account since initialization (see Section 3 1)—the login program sets $y^0_{a_i} = f_a(2i)$ and $y^1_{a_i} = f_a(2i+1)$

6 The login program replaces the instruction table with a new table with an entry of the form $<i, \alpha'_{a_i}, \beta'_{a_i}>$ for each feature ϕ_i. Here,

$$\alpha'_{a_i} = y^0_{a_i} \; G_{\mathsf{pwd}'}(2i) \bmod q$$
$$\beta'_{a_i} = y^1_{a_i} \; G_{\mathsf{pwd}'}(2i+1) \bmod q$$

where $y^0_{a_i}, y^1_{a_i}$ are the new values generated in the previous step

Step 4 above is particularly noteworthy for two reasons First, due to this step, the polynomial f_a is changed to a new random polynomial during each successful login This ensures that an attacker viewing the instruction table at two different times will gain no information about which features switched from distinguishing to non-distinguishing and vice-versa during the interim logins. That is, each time the attacker views an instruction table for an account, either all values will be the same since the last time (if there were no successful logins since the attacker last saw the table) or all values will be different. Second, though generated randomly, f_a is chosen so that $f_a(0) = \mathsf{hpwd}_a$ This ensures that hpwd_a remains constant across multiple logins

Step 5 is also noteworthy, since it shows that whether each feature is distinguishing is recomputed in each successful login So, a feature that was previously distinguishing can become undistinguishing and vice-versa This is the mechanism that enables our scheme to naturally adapt to gradual changes in the user's typing patterns over time

5.3 Security

Consider the "offline" attacker who obtains account a's history file and instruction table, and attempts to find the value of hpwd_a Presuming that the encryption of the history file using hpwd_a is secure, since the values $y^0_{a_i}, y^1_{a_i}$ are effectively encrypted under pwd_a, and since pwd_a is presumably chosen from a much smaller space than hpwd_a, the easiest way to find hpwd_a is to first find pwd_a Thus, to argue the benefits of this scheme, we have to show two things First, we have to show that finding pwd_a is not made easier in our scheme than it is in a typical environment where access is determined by testing the hash of the password against a previously stored hash value. Second, we have to show that the cost to the attacker of finding hpwd_a is generally greater by a significant multiplicative factor

That searching for pwd_a is not made easier in our scheme is clear The attacker has available only the instruction table and the encrypted history file. Since there is a row in the instruction table for each feature (not just those that are distinguishing for a), and since the contents of each row are pseudorandom values, the rows reveal no information about pwd_a And, all other data available to the attacker is encrypted with hpwd_a

The more interesting security consideration in this scheme is how much security it achieves over a traditional password scheme. Suppose that the attacker captured the history file and instruction table after $\ell \geq h$ successful logins to a, and let d be the number of distinguishing features for this account in the ℓ-th login When guessing a password pwd', the attacker can decrypt each field α_{a_i} and β_{a_i} using pwd' to yield points $(2i, \hat{y}^0_{a_i})$ and $(2i+1, \hat{y}^1_{a_i})$, respectively, for $1 \leq i \leq m$ Note that $\hat{y}^0_{a_i} = y^0_{a_i}$ and $\hat{y}^1_{a_i} = y^1_{a_i}$, where $y^0_{a_i}, y^1_{a_i}$ are as generated in Step 5, if and (with overwhelming probability) only if $\mathsf{pwd}' = \mathsf{pwd}_a$. Therefore, there exists a bit string $b \in \{0,1\}^m$ such that $\{(2i + b(i), \hat{y}^{b(i)}_{a_i})\}_{1 \leq i \leq m}$ interpolates to a polynomial $\hat{f}$ with $\hat{f}(0) = \mathsf{hpwd}_a$, if and only if $\mathsf{pwd}' = \mathsf{pwd}_a$. Consequently, one approach that the attacker can take is to enumerate through all $b \in \{0,1\}^m$ and, for each $\hat{f}$ thus computed, see if $\hat{f}(0) = \mathsf{hpwd}_a$ (i e , if $\hat{f}(0)$ will decrypt the history file). This approach slows down the attacker's search for hpwd_a (and pwd_a) by a multiplicative factor of 2^m In practice, the slowdown that the attacker suffers may be substantially less because user typing patterns are not random. In Section 7, we use empirical data to quantify the degree of security achieved against this form of attack, and show that it is nevertheless substantial

However, the attacker has potentially more powerful attacks against this scheme using the $2m$ points $\{(2i, \hat{y}^0_{a_i}), (2i+1, \hat{y}^1_{a_i})\}_{1 \leq i \leq m}$, due to the following contrast On the one hand, if $\mathsf{pwd}' \neq \mathsf{pwd}_a$, then with overwhelming probability, no $m+1$ points will lie on a single degree $m-1$ polynomial, i e , each subset of m points interpolates to a different polynomial with a different y-intercept (not equal to hpwd_a). On the other hand, if $\mathsf{pwd}' = \mathsf{pwd}_a$, then there are $2m - d \geq m$ points that all lie on a polynomial f of degree $m-1$ (and $f(0) = \mathsf{hpwd}_a$), in particular if $d < m$, then there are at least $m+1$ points that all lie on some such f. Asymptotically (i.e , as m grows arbitrarily large), it is known that the second case can be distinguished from the first in $O(m^2)$ time if $d \leq (2-\sqrt{2})m \approx .585m$ using error-correcting techniques [7]. These techniques do not directly break our scheme, since our analysis in Section 7 suggests that for many reasonable values of k, d will typically be too large relative to m for these techniques to succeed (unless the attacker captures the account information before the account is used). Moreover, typically m will be too small in our scenario for these techniques to offer benefit over the exhaustive approach above. However, because these techniques might be improved with application-specific knowledge—e g , that in the second case, at least one of $(2i, \hat{y}^0_{a_i})$ and $(2i+1, \hat{y}^1_{a_i})$ lies on f—it is prudent to look for schemes that confound the use of error-correcting techniques. This is the goal of Section 5 4

5.4 A variation using exponentiation

In this section we present a minor variation of the scheme presented in Sections 5 1–5.2, to which we refer as the "original" scheme below. The scheme of this section is more secure in several ways that will be described below.

Let p be a large prime such that computing discrete logarithms modulo p is computationally intractable (e g., choose p of length 1024 bits) and such that q divides $p-1$. Also, let g be an element of order q in $\mathbb{Z}^*_p$ The main conceptual differences in this variation are that hpwd_a is defined to be $g^{f_a(0)} \bmod p$, and rather than storing α_{a_i} and β_{a_i} in the instruction table, the values

$$\gamma_{a_i} = g^{\alpha_{a_i}} \bmod p$$
$$\delta_{a_i} = g^{\beta_{a_i}} \bmod p$$

are stored instead. Intuitively, since the attacker cannot compute discrete logarithms modulo p, this hides $y^0_{a_i}, y^1_{a_i}$ from him even if he guesses pwd_a.

There are a number of reasons to prefer this variation to the original in practice. First, this modified instruc-

tion table can yield no more information about $f_a(0)$ to the attacker than that of the original, since the attacker can easily transform any instruction table in the original scheme to an instruction table for this variation by computing $g^{\alpha_{a,i}} \bmod p$ and $g^{\beta_{a,i}} \bmod p$ for each $\alpha_{a,i}$ and $\beta_{a,i}$. Second, error-correcting algorithms such as [7] that offer faster-than-brute-force attacks when m grows large and d is small do not directly apply to this variation, and we are unaware of any technique that the attacker can use to search for hpwd_a faster than brute force. Third, as a practical matter, this variation seems to require the attacker to perform modular exponentiations per guessed password when conducting a dictionary attack. Since these are computationally intensive operations, this should slow the attacker's efforts even further

This modification imposes other changes to the scheme In particular, the job of determining hpwd_a from pwd_a and the feature measurements changes somewhat Moreover, re-randomizing the polynomial f_a after each successful login must be done a bit differently, since $f_a(0)$ is hidden even from the login program The resulting login process for the ℓ-th login attempt to a is as follows Let pwd' denote the sequence of characters that the user typed

1 For each ϕ_i, the login program assigns

$$(x_i, z_i) = \begin{cases} (2i,\ (\gamma_{a,i})^{G_{\mathsf{pwd}'}(2i)^{-1} \bmod q} \bmod p) & \text{if } \phi_i(a,\ell) < t_i \\ (2i+1,\ (\delta_{a,i})^{G_{\mathsf{pwd}'}(2i+1)^{-1} \bmod q} \bmod p) & \text{if } \phi_i(a,\ell) \geq t_i \end{cases}$$

The login program now holds m pairs $\{(x_i, z_i)\}_{1 \leq i \leq m}$

2. The login program sets

$$\mathsf{hpwd}' = \prod_{i=1}^{m} (z_i)^{\lambda_i} \bmod p$$

where λ_i is the standard Lagrange coefficient. It then decrypts the history file using hpwd' If this decryption yields a properly-formed plaintext history file, then the login is deemed successful. (If the login were deemed unsuccessful, then the login procedure would halt here.)

3. The login program updates the data in the history file, computes the standard deviation $\sigma_{a,i}$ and mean $\mu_{a,i}$ for each feature ϕ_i over the last h successful logins to a, encrypts the new history file with hpwd' (i.e , hpwd_a), and overwrites the old history file with this new encrypted history file.

4 The login program generates a new random polynomial $f \in \mathbb{Z}_q[x]$ of degree $m-1$ such that $f(0) = 0$.

5 For each distinguishing feature ϕ_i, i e , $|\mu_{a,i} - t_i| > k\sigma_{a,i}$, the login program chooses new random values $y^0_{a,i}, y^1_{a,i} \in \mathbb{Z}_q^*$ subject to the following constraints:

$$\begin{aligned} \mu_{a,i} < t_i &\Rightarrow f(2i) = y^0_{a,i} \wedge f(2i+1) \neq y^1_{a,i} \\ \mu_{a,i} \geq t_i &\Rightarrow f(2i) \neq y^0_{a,i} \wedge f(2i+1) = y^1_{a,i} \end{aligned}$$

For all other features ϕ_i—i.e , those for which $|\mu_{a,i} - t_i| \leq k\sigma_{a,i}$, or all features if there have been fewer than h successful logins to this account since initialization (see Section 3.1)—the login program sets $y^0_{a,i} = f(2i)$ and $y^1_{a,i} = f(2i+1)$

6. The login program replaces the instruction table with a new table with an entry of the form $<i, \gamma'_{a,i}, \delta'_{a,i}>$ for each feature ϕ_i Here,

$$\begin{aligned} \gamma'_{a,i} &= (\mathsf{hpwd}' \cdot g^{y^0_{a,i}})^{G_{\mathsf{pwd}'}(2i)} \bmod p \\ \delta'_{a,i} &= (\mathsf{hpwd}' \cdot g^{y^1_{a,i}})^{G_{\mathsf{pwd}'}(2i+1)} \bmod p \end{aligned}$$

where $y^0_{a,i}, y^1_{a,i}$ are the new values generated in the previous step.

Step 4 is again noteworthy In this case, f_a is determined by choosing a random polynomial f of degree $m-1$ such that $f(0) = 0$ The polynomial f_a is then implicitly determined as $f_a(x) = f(x) + \log_g(\mathsf{hpwd}_a)$, where the logarithm is taken mod p, due to the construction of $\gamma'_{a,i}$ and $\delta'_{a,i}$ in Step 6 This roundabout method of re-randomizing f_a in order to maintain the same $\mathsf{hpwd}_a = g^{f_a(0)} \bmod p$ is needed because the login program cannot compute $\log_g(\mathsf{hpwd}_a)$.

6 An instance based on vector spaces

In this section we briefly describe a second candidate instance of the technique outlined in Section 4 This solution addresses a potential weakness of the scheme of Section 5, namely that any m of the $2m$ values in the instruction table could conceivably be used to reconstruct hpwd_a. That is, the attacker need not limit his attempts at reconstructing hpwd_a to those involving one value from each row of the table since, e.g., the topmost m values in the instruction table could be used to reconstruct hpwd_a if none of the first $m/2$ features are distinguishing It would seem that our technique could be strengthened if the secret sharing scheme used to populate the table would allow reconstruction only with one value from each row Here we present such a sharing scheme and corresponding instance of our method

In this method, hpwd_a is expressed as the determinant of a matrix over $\mathbb{Z}_q$, where q is chosen as in Section 5. Specifically, when an account is initialized, m (column) vectors $\underline{v}_{a1}, \cdots, \underline{v}_{am} \in \mathbb{Z}_q^m$ are chosen at random from $\mathbb{Z}_q^m$ The hardened password is $\mathsf{hpwd}_a = \det(\underline{v}_{a1}, \cdots, \underline{v}_{am}) \bmod q$. The instruction table initially contains an entry of the form $<i, \underline{\alpha}_{a,i}, \underline{\beta}_{a,i}>$ for each feature ϕ_i, where

$$\begin{aligned} \underline{\alpha}_{a,i} &= \underline{v}_{a,i}\ G_{\mathsf{pwd}_a}(2i) \bmod q \\ \underline{\beta}_{a,i} &= \underline{v}_{a,i}\ G_{\mathsf{pwd}_a}(2i+1) \bmod q \end{aligned}$$

Note that at initialization, and more generally when there are no distinguishing features, the "shares" in $\underline{\alpha}_{a,i}$ and $\underline{\beta}_{a,i}$ are the same (albeit encrypted under different outputs from G_{pwd_a}) This is reasonable since when there are no distinguishing features, our approach offers no additional security over that offered by pwd_a anyway.

The login process for the ℓ-th login attempt to a is as follows. Let pwd' denote the sequence of characters that the user typed.

1. For each ϕ_i, the login program assigns

$$\underline{v}_i = \begin{cases} \underline{\alpha}_{a,i} \cdot G_{\mathsf{pwd}'}(2i)^{-1} \bmod q & \text{if } \phi_i(a,\ell) < t_i \\ \underline{\beta}_{a,i} \cdot G_{\mathsf{pwd}'}(2i+1)^{-1} \bmod q & \text{if } \phi_i(a,\ell) \geq t_i \end{cases}$$

The login program now holds m vectors $\{\underline{v}_i\}_{1 \leq i \leq m}$

2 The login program sets $\mathsf{hpwd}' = \det(\underline{v}_1, \cdots, \underline{v}_m) \bmod q$. It then decrypts the history file using hpwd' If this decryption yields a properly-formed plaintext history file,

then the login is deemed successful (If the login were deemed unsuccessful, then the login procedure would halt here)

3. The login program updates the data in the history file, computes the standard deviation σ_{a_i} and mean μ_{a_i} for each feature ϕ_i over the last h successful logins to a, encrypts the new history file with hpwd' (i e., hpwd_a), and overwrites the old history file with this new encrypted history file.

4 The login program generates new random vectors $\underline{w}_1, \ldots, \underline{w}_m \in \mathbb{Z}_q^m$ such that $\det(\underline{w}_1, \ldots, \underline{w}_m) \bmod q = \mathsf{hpwd}'$

5. The login program takes one of the following two steps, depending on whether there are distinguishing features

   a. If there are no distinguishing features, then the login program sets $\underline{v}_{a_i}^0 = \underline{v}_{a_i}^1 = \underline{w}_i$ for each $1 \le i \le m$

   b Otherwise, the login program generates new random vectors $\underline{u}_1, \ldots, \underline{u}_m \in \mathbb{Z}_q^m$ such that[3]

$$\forall b \in \{0,1\}^m : \ \det(\underline{u}_1^{b(1)}, \ldots, \underline{u}_m^{b(m)}) \bmod q = 1 \quad (1)$$

   where

$$\underline{u}_i^{b(i)} = \begin{cases} \underline{e}_i & \text{if } b(i) = 0 \\ \underline{u}_i & \text{if } b(i) = 1 \end{cases}$$

   and $\underline{e}_i$ is the unit vector with a 1 in position i and a 0 in all other positions. (How to compute $\underline{u}_1, \ldots, \underline{u}_m$ efficiently is described below) Then, for each distinguishing feature ϕ_i, the login program chooses new random vectors $\underline{v}_{a_i}^0, \underline{v}_{a_i}^1 \in \mathbb{Z}_q^m$ subject to the following constraints, where $W = (\underline{w}_1, \ldots, \underline{w}_m)$.

$$\mu_{a_i} < t_i \ \Rightarrow \ \underline{v}_{a_i}^0 = \underline{w}_i \wedge \underline{v}_{a_i}^1 \neq W \cdot \underline{u}_i$$
$$\mu_{a_i} \ge t_i \ \Rightarrow \ \underline{v}_{a_i}^0 \neq \underline{w}_i \wedge \underline{v}_{a_i}^1 = W \ \underline{u}_i$$

   For all other features ϕ_i—i.e., those for which $|\mu_{a_i} - t_i| \le k\sigma_{a_i}$—the login program sets $\underline{v}_{a_i}^0 = \underline{w}_i$ and $\underline{v}_{a_i}^1 = W \ \underline{u}_i$

6. The login program replaces the instruction table with a new table with an entry of the form $<i, \underline{\alpha}'_{a_i}, \underline{\beta}'_{a_i}>$ for each feature ϕ_i. Here,

$$\underline{\alpha}'_{a_i} = \underline{v}_{a_i}^0 \cdot G_{\mathsf{pwd}'}(2i) \bmod q$$
$$\underline{\beta}'_{a_i} = \underline{v}_{a_i}^1 \ G_{\mathsf{pwd}'}(2i+1) \bmod q$$

   where $\underline{v}_{a_i}^0, \underline{v}_{a_i}^1$ are the new vectors generated in the previous step

To perform Step 4 efficiently, the login program can select any factorization $\mathsf{hpwd}_a = \prod_{i=1}^{2m} \eta_i \bmod q$ of hpwd_a. Then, the login program can set $(\underline{w}_1, \ldots, \underline{w}_m) = T_{\mathsf{up}} \ T_{\mathsf{lo}} \bmod q$ where $T_{\mathsf{lo}}, T_{\mathsf{up}}$ satisfy $T_{\mathsf{lo}}[i,j] = T_{\mathsf{up}}[j,i] = 0$ for $1 \le i < j \le m$, $T_{\mathsf{lo}}[i,j]$ and $T_{\mathsf{up}}[j,i]$ are random elements of $\mathbb{Z}_q$ for $1 \le j < i \le m$, and $\{T_{\mathsf{lo}}[i,i], T_{\mathsf{up}}[i,i]\}_{1 \le i \le m} = \{\eta_i\}_{1 \le i \le 2m}$.

An efficient algorithm to generate $\underline{u}_1, \ldots, \underline{u}_m$ in Step 5b so that they contain significant randomness and satisfy condition (1) is as follows The login program first chooses an upper-triangular matrix $U' = (\underline{u}'_1, \ldots, \underline{u}'_m)$ that has 1 for each diagonal element and random elements of $\mathbb{Z}_q$ above the diagonal Then, the login program sets $(\underline{u}_1, \ldots, \underline{u}_m) = \Pi \ U' \ \Pi^{-1}$ where $\Pi = (\underline{\pi}_1, \ldots, \underline{\pi}_m)$ is a random permutation matrix (i e , the identity matrix with columns permuted randomly) subject to the constraint that if $\phi_{i_1}, \ldots, \phi_{i_d}$ are the distinguishing features for this account, then $\{\underline{\pi}_j\}_{1 \le j \le d} = \{\underline{e}_{i_j}\}_{1 \le j \le d}$. In particular, this stipulation ensures (with high probability) that $\underline{v}_{a_i}^0 \neq \underline{v}_{a_i}^1$ for each $1 \le i \le m$ when created in Step 5b

A property of this scheme is that when an offline attacker decrypts the instruction table with a candidate password pwd' to yield vectors $\{\hat{\underline{v}}_{a_i}^0, \hat{\underline{v}}_{a_i}^1\}_{1 \le i \le m}$, the only combinations of these vectors that could conceivably yield hpwd_a are of the form $\det(\hat{\underline{v}}_{a1}^{b(1)}, \ldots, \hat{\underline{v}}_{am}^{b(m)}) \bmod q$ for some $b \in \{0,1\}^m$. That is, not any combination of the m vectors holds the possibility of generating hpwd_a

As in Section 5, the security of this scheme against an offline attacker depends most directly on how quickly the attacker can distinguish the cases $\mathsf{pwd}' = \mathsf{pwd}_a$ and $\mathsf{pwd}' \neq \mathsf{pwd}_a$ When an attacker decrypts the instruction table with a password $\mathsf{pwd}' \neq \mathsf{pwd}_a$, the result will be $2m$ random vectors. If $\mathsf{pwd}' = \mathsf{pwd}_a$, however, the table may have more structure For example, if $\mathsf{pwd}' = \mathsf{pwd}_a$ and there is only one distinguishing feature ϕ_i, then either $\hat{\underline{v}}_{a_i}^0$ or $\hat{\underline{v}}_{a_i}^1$ will be expressible as a linear combination of $\hat{\underline{v}}_{a_j}^0$ and $\hat{\underline{v}}_{a_j}^1$ for some $j \neq i$ (due to our construction of $\underline{u}_1, \ldots, \underline{u}_m$ above) In general, whether there is enough additional structure for the attacker to efficiently exploit depends on the number and distribution of distinguishing features

7 Empirical analysis

In order to evaluate the viability of our approach, we developed and deployed an experiment to collect password typing measurements from users. Specifically, we replaced the `basic-auth` function of a Netscape Enterprise Server 3 0 in active use with an implementation that uses a Java applet to record each user's keystroke features (keystroke durations and latencies between keystrokes) when typing her password On this web server, all privileged users use the *same* password to access the password-protected pages This provided an interesting case study, since it enabled a direct comparison of user typing behavior on the same password. The password used in this experiment has 8 characters (i e., $m = 15$), but because it is still in active use, we cannot disclose it here. At the time of this writing, login measurements have been recorded for approximately 11 weeks. For the discussion in this section, we use data gathered from the 13 users for which we have at least 4 logins recorded on her usual keyboard Our analysis employs only each user's logins from her usual keyboard, as reported by the user. In total, this analysis is based on 188 recorded logins.

The goal of our experiment is to empirically evaluate the number of distinguishing features for the average user, the entropy of users' distinguishing features, and the reliability of successful password entry. The number of distinguishing features for the average user is important because the strength of our proposal is enhanced if the number d of distinguishing features for a user is large relative to the number m of features overall However, this alone is not enough to ensure that our scheme offers a significant increase in security. To see why, suppose for an extreme case that all users could be partitioned into "slow typists" and "fast typists": slow typists have the property that for any of their

[3] Condition (1) is stronger than necessary Rather, using terminology introduced in Section 7, it suffices that $\det(\underline{u}_1^{b(1)}, \ldots, \underline{u}_m^{b(m)}) \bmod q = 1$ only for any $b \in \{0,1\}^m$ that extends the feature descriptor of this account However, we know a fast algorithm for computing $\{\underline{u}_i\}_{1 \le i \le m}$ satisfying only the more restrictive condition (1)

distinguishing features ϕ_i, $\mu_{ai} > t_i$ (where a is the user's account), and analogously fast typists have the property that $\mu_{ai} < t_i$ for all of their distinguishing features ϕ_i. Then, even if *all* of an account's features are distinguishing, the offline attacker needs to examine only two possibilities upon guessing a password pwd′: the values in the first column of the (decrypted) instruction table, and the values in the second column. Consequently, the *entropy* of users' distinguishing features (defined below) is as important to our scheme as the *number* of distinguishing features. Finally, obviously the ability of a user to reliably generate her hardened password is important to the usability of our scheme.

We evaluated each of these facets for varying values of k, where a feature ϕ_i is distinguishing if $|\mu_{ai} - t_i| > k\sigma_{ai}$ (see Section 3.1). In general, a lower value of k increases the number of distinguishing features per user and thus increases the sensitivity of login to user typing patterns. On the other hand, a higher value of k makes it easier for the user to log in, but tends to decrease the number of distinguishing features per user. In addition, for simplicity of presentation, in our evaluation we ignored the parameter h, i.e., all of an account a's logins were used to compute μ_{ai} and σ_{ai}.

7.1 Entropy due to keystrokes

Fundamental to our empirical evaluation is the measure of keystroke entropy we chose, which we now describe. As described above, all users employ the same password in our experiments. Intuitively, our measure of entropy should capture the amount of remaining uncertainty there is in hpwd_a for a randomly chosen account a.

We define a *feature descriptor* to be a partial function $b : \{1, \ldots, m\} \to \{0, 1\}$, and let B be the set of all feature descriptors. For a fixed k, let the *feature descriptor* b_a *for account* a be defined by

$$b_a(i) = \begin{cases} 0 & \text{if } \mu_{ai} + k\sigma_{ai} < t_i \\ 1 & \text{if } \mu_{ai} - k\sigma_{ai} > t_i \\ \perp & \text{otherwise} \end{cases}$$

That is, $b_a(i) = 1$ for every distinguishing feature ϕ_i on which the user is "slow" and $b_a(i) = 0$ for every distinguishing feature ϕ_i on which the user is "fast". For other features ϕ_i, $b_a(i)$ is undefined ($\perp$).

We would like to compute the entropy of a randomly chosen account's feature descriptor. However, this is complicated by the fact that a feature descriptor may (and typically will) have undefined values. For example, suppose that $|A| = m$, that each account has only a single distinguishing feature, and that no feature is distinguishing for two accounts. Then, the Shannon entropy of a randomly chosen account a's feature descriptor would seem to be at least $\log m$, due to the uncertainty in the position i of the account's distinguishing feature (i.e., $b_a(i) \neq \perp$). Nevertheless, an attacker knowing pwd_a need only attempt to reconstruct hpwd_a using at most two different (total) feature descriptors, e.g., b such that $b(i) = 0$ for each $1 \leq i \leq m$, and b such that $b(i) = 1$ for each $1 \leq i \leq m$.

As a tool to better capture the entropy available due to keystrokes, we define a *cover* to be a function $C : A \to B$ such that $C(a)$ is total for each $a \in A$, and $b_a(i) \neq \perp \Rightarrow b_a(i) = C(a)(i)$. That is, a cover maps each account a to a (total) feature descriptor that is identical to b_a wherever b_a is defined. Given a cover, we can evaluate the entropy of $C(a)$ under random choice of a, in a way that will be defined below. We then choose a cover that minimizes this entropy, and take this cover's entropy as "the entropy due to keystrokes". This provides a more conservative evaluation of the entropy due to keystrokes, because multiple accounts can map to the same total feature descriptor under C. So, in the example of the previous paragraph, all accounts can map to at most two such descriptors.

Guessing entropy [17] is a natural way to define the entropy of a cover. Let $\mathrm{Img}(C) = \{b \in B \mid \exists a \in A : C(a) = b\}$, and $w_C(b) = |\{a \in A \mid C(a) = b\}|/|A|$. If we denote $\mathrm{Img}(C) = \{b_1, \ldots, b_\ell\}$ such that $w_C(b_1) \geq w_C(b_2) \geq \cdots \geq w_C(b_\ell)$, then the guessing entropy of the cover C is

$$E_C = \sum_{i=1}^{|\mathrm{Img}(C)|} (i \cdot w_C(b_i))$$

Intuitively, the guessing entropy is the expected number of feature descriptors in $\mathrm{Img}(C)$ an attacker would need to examine (and perform the corresponding reconstruction) to find hpwd_a for a randomly chosen account a. Moreover, this expected value supposes that the attacker knows the "weight" $w_C(b)$ of each element in $\mathrm{Img}(C)$ and thus examines elements of $\mathrm{Img}(C)$ in an optimal order to minimize this expected value. As described above, in the worst case an attacker will know $\mathrm{Img}(C)$ and w_C for a cover C that *minimizes* E_C, and so it is this cover we use in our computations of Section 7.2.

7.2 Results

Our analysis methodology consisted of the following steps for each value of k. We first found values t_{dur} and t_{lat} that maximized the guessing entropy, when $t_i = t_{\mathsf{dur}}$ for each duration feature ϕ_i and when $t_i = t_{\mathsf{lat}}$ for each latency feature ϕ_i. More specifically, for each pair of candidate integer values t_{dur}, t_{lat} in the ranges 80 ms $\leq t_{\mathsf{dur}} \leq$ 125 ms and 70 ms $\leq t_{\mathsf{lat}} \leq$ 140 ms, we computed the feature descriptor for each account and a cover C for these feature descriptors with minimum guessing entropy. We then chose a pair t_{dur}, t_{lat} that resulted in the highest guessing entropy from this calculation. In this way, we captured the guessing entropy faced by the attacker in the case that the system was configured with optimal values of t_{dur}, t_{lat}. The reliability of password login was computed by calculating the percentage of each account's logins that would have succeeded for these values of t_{dur}, t_{lat}, and then averaging these percentages over all accounts. If there were multiple pairs that yielded the same maximum guessing entropy as computed above, then t_{dur}, t_{lat} were chosen from among them as the pair yielding the highest reliability. The average number of distinguishing features d per user given k, t_{dur}, and t_{lat} was then computed.

The results of this analysis are shown in Figure 1. The smallest value of k studied was $k = 0.4$. This choice yields a guessing entropy of roughly 6.1, which is strong given the small number of users (13) in our study. (For this number of users, the maximum possible guessing entropy would be 7.) Moreover, this choice yields roughly 12.3 distinguishing features for the average account and an approximately 51.6% success rate for legitimate logins. That is, the expected number of attempts before a user succeeds in logging into her account is less than 2. If this reliability is insufficient, however, then increasing k to 1.0, for example, increases login reliability to 77.1% while retaining a respectable guessing entropy (2.8) and number of distinguishing features (7.7). Due to the computational expense of analyzing our data for values of k greater than 1.0, we cannot report results for these cases here.

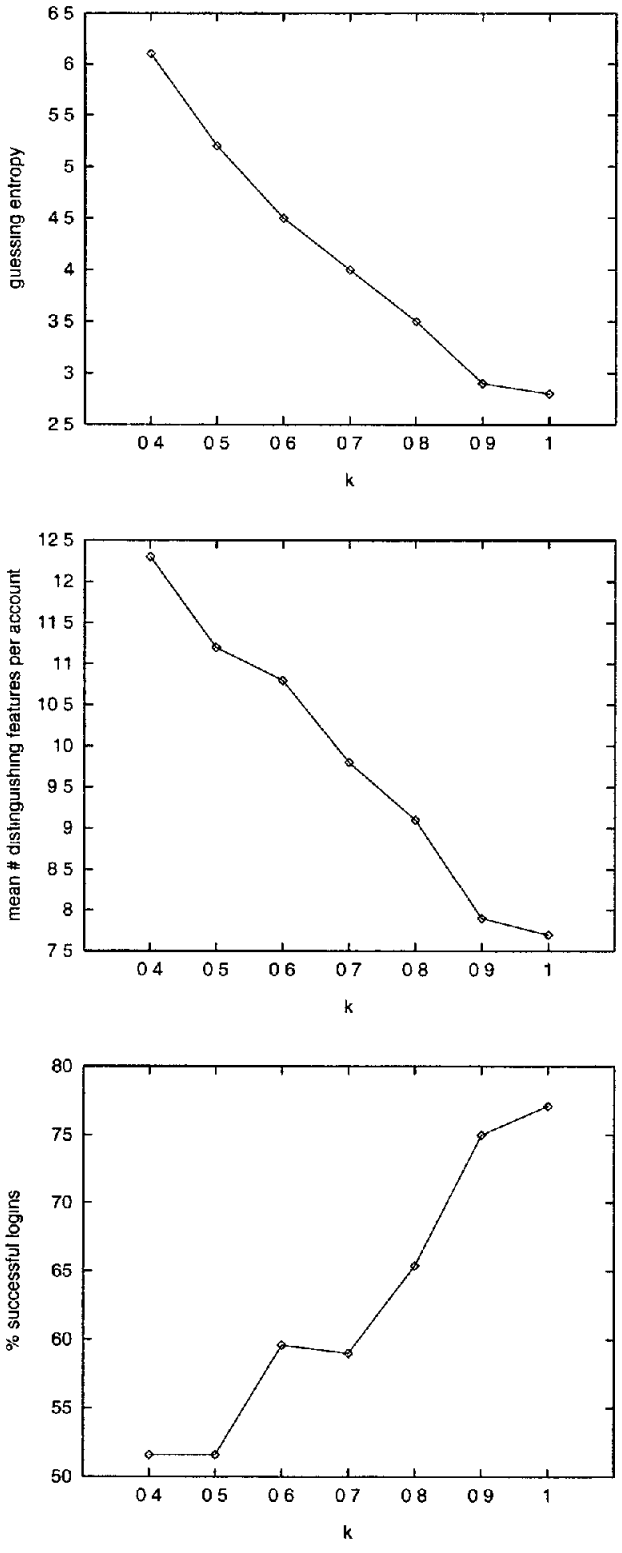

Figure 1 Empirical results

8 Implementation

We have implemented the method of Section 5.4 to experiment with our techniques further. Our reference implementation is built in C/C++ for Microsoft Windows platforms, and utilizes the Microsoft Foundation Classes (MFC) for constructing its user interface. In particular, the MFC provides the low-level key press and key release events necessary to time the duration and latency of keystrokes. Our implementation utilizes the CryptoLib library [11] version 1.2 for its basic cryptographic operations, extended with the use of addition chains to optimize modular exponentiations [2].

Our implementation provides three types of functions: initialization, login, and recovery. We have already described the first two of these functions in detail. The third, recovery, is intended for use in circumstances where the user finds herself unable to generate her correct hardened password after repeated attempts, due to a sharp change in her typing patterns. We have shown in Section 7 that this should be a rare occurrence for reasonable values of k, but it is nevertheless one that must be anticipated. The recovery program that we have implemented is easily derived from the login program described in Section 5.4. the recovery program decrypts all instruction table entries using the password pwd_a (provided by the user) and then exhaustively searches to find hpwd_a (within time proportional to 2^m). However, this recovery program should not simply be used as an alternative login program, since it would enable an attacker who learns pwd_a to generate hpwd_a without having to recreate the legitimate user's keystroke dynamics. Rather, the use of this recovery program should be under tighter controls, e.g., an administrator's. Other recovery techniques are possible, such as additionally storing the hardened password encrypted under a much stronger secret that can be accessed only with administrator assistance or with an additional hardware token.

We have performed a battery of tests to evaluate the performance of the method in Section 5.4. These tests were run on a Dell Inspiron 3200 computer with a 266 MHz Pentium II processor running Windows NT Workstation 4.0. In these tests, q and p were 160 bits and 1024 bits, respectively. Triple-DES in CBC mode was used to encrypt the history file. The pseudorandom function family G was implemented as $G_K(x) = F(K, x)$ where F was SHA-1. The history length was $h = 8$. The number of measured features was $m = 15$.

Of the three functions, the times required for initialization and recovery are highly variable. The time for initialization is overwhelmingly dominated by the time needed to generate p and q, which can be substantial but in our tests always completed in under one minute. Since p and q can be generated once and then used for all accounts, this should not be a bottleneck in practice. Recovery is the other function with highly variable delays. Our implementation exhaustively searches through the 2^{15} possible (total) feature descriptors, using each to attempt to generate hpwd_a. The enumeration and testing of all 2^{15} possibilities completes in roughly 11 hours in the worst case.

In contrast to the times for initialization and recovery, delays for successful and failed logins are virtually constant. Beginning when the user finishes typing her password, successful logins require roughly 4.5 seconds to complete, and failed logins complete in approximately 1.2 seconds. The delay for a failed login is substantially shorter than for a successful one because a login failure causes most of the login steps to be bypassed.

9 Conclusion

We have presented a novel approach for hardening passwords by exploiting the keystroke dynamics of users. Our approach enables the generation of a long-term secret (the hardened password) that can be tested for login purposes or used for encryption of files, entry to a virtual private network, etc. Our technique increases the time for an offline attacker to exhaustively search for this hardened password (or the text password used to generate it), and can be used in conjunction with salting to slow the attacker further In addition, our approach improves security against an online attacker who learns the text password (e g , by observing it being typed) and attempts to login to an account protected by the hardened password

As our prototype implementation suggests, our technique is viable for use in practice It adapts to gradual changes in a user's keystroke dynamics over time, while still generating the same hardened password. And, using actual keystroke data, we have given evidence that our scheme both improves upon the security of conventional passwords and is easy to use by the average user. There remains a small risk in our scheme that due to a sudden shift in typing behavior, a user will be unable to log into her account. This risk can be minimized if the use of our scheme is restricted to local logins on the same keyboard (e.g., on laptops). In addition, our scheme can be coupled with recovery mechanisms, as we have described

For future work, we intend to validate our methods on a larger user population. We are also investigating the performance of our techniques when applied to other biometrics, particularly other non-static biometrics such as voice, where features such as pitch and amplitude can be used in place of latencies and durations.

Acknowledgements

We are grateful to Markus Jakobsson and Amin Shokrollahi for insightful discussions Phil MacKenzie and the anonymous referees provided helpful comments that improved the presentation of this paper. Thanks also to Daniel Bleichenbacher for providing an implementation of [2]

References

[1] S Bleha, C Slivinksy, and B Hussein Computer-access security systems using keystroke dynamics *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-12(12) 1217–1222, December 1990

[2] D Bleichenbacher Addition chains for large sets Manuscript, 1999

[3] G I Davida, Y Frankel, and B J Matt On enabling secure applications through off-line biometric identification. In *Proceedings of the 1998 IEEE Symposium on Security and Privacy*, pages 148–157, May 1998

[4] D Feldmeier and P Karn UNIX password security—Ten years later In *Advances in Cryptology—CRYPTO '89 Proceedings* (Lecture Notes in Computer Science 435), 1990

[5] M R Garey and D S Johnson *Computers and Intractability A Guide to the Theory of* NP-*Completeness* W H Freeman and Company, New York, 1979.

[6] R Gaines, W Lisowski, S. Press, and N Shapiro *Authentication by keystroke timing. Some preliminary results* Rand report R-256-NSF Rand Corporation, 1980

[7] V Guruswami and M Sudan Improved decoding of Reed-Solomon and algebraic-geometric codes In *Proceedings of the* 39th *IEEE Symposium on Foundations of Computer Science*, pages 28–37, 1998

[8] G Horng Password authentication without using a password table *Information Processing Letters* 55 247–250, 1995

[9] R Joyce and G Gupta Identity authorization based on keystroke latencies *Communications of the ACM* 33(2) 168–176, February 1990

[10] D Klein Foiling the cracker A survey of, and improvements to, password security In *Proceedings of the* 2nd *USENIX Security Workshop*, August 1990

[11] J. B Lacy, D P Mitchell, and W M Schell. CryptoLib Cryptography in software In *Proceedings of the* 4th *USENIX Security Workshop*, pages 1–17, October 1993

[12] C H. Lin, C C Chang, T C Wu, and R C T Lee Password authentication using Newton's interpolating polynomials *Information Systems* 16(1) 97–102, 1991

[13] R E Lennon, S M Matyas, and C H Meyer Cryptographic authentication of time-invariant quantities *IEEE Transactions on Communications* COM-29(6) 773–777, June 1981

[14] G Leggett and J Williams Verifying identity via keystroke characteristics *International Journal of Man-Machine Studies* 28(1) 67–76, 1988

[15] G Leggett, J Williams, and D Umphress Verification of user identity via keystroke characteristics *Human Factors in Management Information Systems*, 1989

[16] U Manber A simple scheme to make passwords based on one-way functions much harder to crack *Computers & Security* 15(2) 171–176, 1996

[17] J L Massey Guessing and entropy In *Proceedings of the 1994 IEEE International Symposium on Information Theory*, 1994

[18] D Mahar, R Napier, M Wagner, W Laverty, R Henderson and M Hiron Optimizing digraph-latency based biometric typist verification systems inter and intra typists differences in digraph latency distributions *International Journal of Human-Computer Studies* 43 579–592, 1995

[19] A J Menezes, P C van Oorschot, and S A Vanstone *Handbook of Applied Cryptography*, CRC Press, 1997

[20] F Monrose and A Rubin Authentication via keystroke dynamics In *Proceedings of the* 4th *ACM Conference on Computer and Communications Security*, pages 48–56, April 1997

[21] R Morris and K Thompson Password security A case history *Communications of the ACM*, 22(11) 594–597, November 1979

[22] K S Nash Rising laptop theft tacks on $150 a box *ComputerWorld*, August 3, 1998 Available at http //www computerworld com/home/print nsf/all/9808035ED6

[23] R L Rivest Cryptography In *Handbook of Theoretical Computer Science*, Chapter 13, pages 717–755, Elsevier Science Publishers, B V , 1990

[24] J A Robinson, V M Liang, J A Chambers and C L MacKenzie Computer user verification using login string keystroke dynamics *IEEE Transactions on System, Man, and Cybernetics*, 28(2), 1998

[25] A Shamir How to share a secret *Communications of the ACM* 22(11) 612–613, November 1979

[26] FIPS 180-1, Secure hash standard Federal Information Processing Standards Publication 180-1, U S Department of Commerce/N I S T , National Technical Information Service, April 17, 1995

[27] E Spafford Observations on reusable password choices In *Proceedings of the* 3rd *USENIX Security Symposium*, September 1992

[28] D Umphress and G Williams Identity verification through keyboard characteristics *International Journal of Man-Machine Studies* 23(3) 263–273, 1985

[29] T Wu A real-world analysis of Kerberos password security In *Proceedings of the 1999 Network and Distributed System Security Symposium*, February 1999