IJIS (2001) 1: 1–15 / Digital Object Identifier (DOI) 10.1007/s102070100006

Password hardening based on keystroke dynamics

Fabian Monrose, Michael K. Reiter*, Susanne Wetzel

Bell Labs, Lucent Technologies, Murray Hill, N.J., USA

Published online: 2001 – © Springer-Verlag 2001

Abstract. We present a novel approach to improving the security of passwords. In our approach, the legitimate user's typing patterns (e.g., durations of keystrokes and latencies between keystrokes) are combined with the user's password to generate a *hardened password* that is convincingly more secure than conventional passwords alone. In addition, our scheme automatically adapts to gradual changes in a user's typing patterns while maintaining the same hardened password across multiple logins, for use in file encryption or other applications requiring a long-term secret key. Using empirical data and a prototype implementation of our scheme, we give evidence that our approach is viable in practice, in terms of ease of use, improved security, and performance.

Keywords: Security – Biometrics – Cryptographic – Key generation

1 Introduction

Textual passwords have been the primary means of authenticating users to computers since the introduction of access controls in computer systems. Passwords remain the dominant user authentication technology today, despite the fact that they have been shown to be a fairly weak mechanism for authenticating users. Studies have shown that users tend to choose passwords that can be broken by an exhaustive search of a relatively small subset of all possible passwords. In one case study of 14 000 Unix™passwords, almost 25% of the passwords were found by searching for words from a carefully formed "dictionary" of only 3×10^6 words [15] (see also [9, 26, 33, 34]). This high success rate is not unusual despite the fact that there are roughly 2×10^{14} eight-character passwords consisting of digits and upper- and lower-case letters alone.

In this paper, we propose a technique for improving the security of password-based applications by incorporating biometric information into the password. Specifically, our technique generates a *hardened password* based on both the password characters and the user's typing patterns when typing the password. This hardened password can be tested for login purposes or used as a cryptographic key for file encryption, virtual private network access, etc. The primary attacker we consider is one who obtains all stored system information for password verification (the analog of the `/etc/passwd` file in a typical Unix™ environment). We show that this attacker faces a convincingly more difficult task to exhaustively search for the hardened password than in a traditional password scheme.

There are several challenges to realizing this goal. The first is to identify features of a user's typing patterns (e.g., latencies between keystrokes and duration of keystrokes) that the user reliably repeats (approximately) when typing her password. The second is to use these features when the user types her password to generate the correct hardened password. At the same time, however, the attacker who captures system information used to generate or verify hardened passwords should be unable to determine which features are relevant to generating a user's hardened password, since revealing this information could reveal information about the characters related to that password feature. For example, suppose the attacker learns that the latency between the first and second keystrokes is a feature that is reliably repeated by the user and thus is used to generate her hardened password. Then this may reveal information

[*] Corresponding author.

about the first and second characters of the text password, since due to keyboard dynamics, some digraphs are more amenable to reliable latency repetitions than others.

Our approach effectively hides information about which of a user's features are relevant to generating her hardened password, even from an attacker that captures all system information. At the same time, it employs novel techniques to impose an additional (multiplicative) work factor on the attacker who attempts to exhaustively search the password space. Using empirical data, we evaluate both this work factor and the reliability with which legitimate users can generate their hardened passwords. Our empirical studies demonstrate various choices of parameters that yield both increased security and sufficient ease of use.

Our scheme is very attractive for use in practice. Unlike other biometric authentication procedures (e.g., fingerprint recognition, and retina or iris scans), our approach is unintrusive and works with off-the-shelf keyboards. Initially our scheme is as secure as a "normal" password scheme and then adapts to the user's typing patterns over time, gradually hardening the password with biometric information. Moreover, while fully able to adapt to gradual changes in user typing patterns, our scheme can be used to generate the same hardened password indefinitely, despite changes in the user's typing patterns. Therefore, the hardened password can be used to encrypt files, for example, without needing to decrypt and re-encrypt files with a new hardened password on each login.

The main limitation of our scheme is that a user whose typing patterns change substantially between consecutive instances of typing her password may be unable to generate her correct hardened password and thus, for example, 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 of 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 alarming rate of laptop thefts (e.g., see [28]), these applications demand security better than that provided by traditional passwords.

Although we study only the generation of hardened passwords using keystroke patterns in this paper, the techniques we introduce are of more general interest. In particular, any repeatable and unpredictable physical phenomenon for which features can be measured can, in theory, be employed with our techniques to generate cryptographic secrets. Our continuing work, for example, has demonstrated this for voice patterns [24, 25].

2 Related work

The motivation for using keystroke features to harden passwords comes from years of research validating the hypothesis that user keystroke features are both highly repeatable and different between users [1, 10, 13, 17, 18, 23, 30]. Prior work has anticipated utilizing keystroke information in the user login process (e.g., [13]), 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 this to user keystroke behavior during password entry. Thus while they are useful for defending against an attacker who attempts to log into the system directly, they provide no additional protection against an attacker who captures system information related to user authentication and then conducts an off-line 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 Sect. 1. Our work therefore improves on these schemes in two ways: (1) our method is the first to offer stronger security against this stronger attacker, and (2) 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 first work (that we are aware of) that previously proposed generating a repeatable key based on biometric information is due to Soutar and Tomko [32]. This work outlines a technique for using optical computing to generate a cryptographic key from a fingerprint pressed against a glass prism; products based on this technique are marketed by Mytec Technologies (`http://www.mytec.com/`). Due to the dependence of this technique on optical computing, and specifically on the presentation of a two-dimensional surface (such as a finger surface) for generating the cryptographic key, this approach is not directly amenable to generating cryptographic keys from timing information.

A different approach to generating a repeatable key based on biometric data is due to Davida, Frankel, and Matt [6]. In this scheme, a user carries a portable storage device containing: (1) error-correcting parameters to decode readings of biometric data (e.g., an iris scan) with a limited number of errors to a "canonical" reading for that user, and (2) 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 data for the user. For this reason, their approach requires a biometric with substantial entropy:

for example, 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 eight-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. Exposing error-correcting parameters in our setting would therefore substantially diminish the available entropy from keystroke features. Moreover, exposing information about the keystroke features can, in turn, expose information about the password itself (as discussed in Sect. 1). This makes the careful utilization of keystroke features critical in our setting, whereas in [6] the biometric data considered were presumed independent of the password chosen. Juels and Wattenberg [14] generalized and improved the Davida et al. scheme through a novel modification in the use of error-correcting codes, thereby shrinking the code size and achieving higher resilience. However, since the performance of this scheme on actual biometric data was not explored, it is unknown whether this technique is more applicable in our setting.

Ellison et al. independently developed a method for generating a cryptographic key based on answers to questions posed to a user [8]. The work is premised on the assumption that questions can be posed that the legitimate user will answer one way but others attempting to impersonate the user will answer another way. Their construction resembles one instance of our technique, namely that of Sects. 5.1 and 5.2, and in this way their scheme achieves a degree of resilience to forgotten answers. However, Bleichenbacher and Nguyen have shown that the Ellison et al. scheme is insecure, whereas our constructions appear to be much stronger [2]. In work subsequent to ours, a construction similar to that in Sects. 5.1 and 5.2 was used in the design of a forensic database, where a person's medical record can be decrypted only once a DNA sample of the person is obtained (e.g., at a crime scene) [3].

Our method of hardening user passwords has conceptual similarities to password "salting" for user logins. 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 [20, 26]. 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 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) [12, 16, 19]. Here we do not focus on this threat model, although our hardened passwords can be directly combined with these techniques to also provide security against this type of attacker.

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 clarity, in the remainder 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. As in existing computer systems, the characters that the user types into the password field are a factor in determining whether to accept or reject the login. For the remainder of the 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 differ significantly from the typing patterns displayed in previous successful logins to the account. Here we present our scheme in a way that keeps hpwd_a constant across logins, 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. We will empirically 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. In addition, derived features could be used, such as the total duration of a subset of keystrokes, the ratio of one keystroke's duration to another, or the latency scaled according to the particular keys between which the latency is measured.

Abstractly, we represent a feature by a function $\phi : A \times \mathbb{N} \to \mathbb{R}$ where $\phi(a, \ell)$ is the measurement of that feature during the ℓth (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_{ai} and σ_{ai} be the mean and standard deviation of the measurements $\phi_i(a, j_1), \ldots, \phi_i(a, j_h)$ where $j_1, \ldots, j_h$ 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 $|\mu_{ai} - t_i| > k\sigma_{ai}$ where $k \in \mathbb{R}^+$ is a parameter of the system. If ϕ_i is a distinguishing feature for the account a, then either $t_i > \mu_{ai} + k\sigma_{ai}$, that is, the user consistently measures below t_i on this feature, or $t_i < \mu_{ai} - k\sigma_{ai}$, that is, the user consistently measures above t_i on this feature.

We define a *feature descriptor* to be a partial function $b : \{1, \ldots, m\} \to \{0, 1\}$, and the *feature descriptor* b_a *for account* a to be

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

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 ($\perp$). A *total* feature descriptor is one defined on all domain elements; let $\mathcal{T}$ be the set of all total feature descriptors. For any feature descriptor b, the total descriptors that *extend* it is the set

$$\mathcal{T}(b) = \{b' \in \mathcal{T} \mid \forall i \in \{1, \ldots, m\} : b(i) \neq \perp \Rightarrow b'(i) = b(i)\}$$

3.2 Security goals

In our login architecture, the system stores information for each 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 attacker who captures this information stored in the system, and then uses this information in an off-line effort to find hpwd_a (and pwd_a). A first and basic requirement is that any such attack should be at least as difficult as exhaustively searching for pwd_a in a traditional Unix™ setting where the attacker has the `/etc/passwd` file. 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, that is, 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 Sect. 8, 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 that the slowdown factor that our scheme achieves is over and above any benefits that salting offers.

We emphasize that our goals are quite different from the goal of biometrics as usually applied to user authentication. In particular, we do not insist that it is necessarily difficult to find a person who can impersonate the typing of another user on a certain password; i.e., from an authentication perspective, the *false positive* rate may be substantial among users who know the right password. Nevertheless, by incorporating typing variability within the search space of the attacker described above, we can slow the attacker considerably at virtually no additional cost to the legitimate user.

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 from a large space. In the instances of our technique in Sects. 5 and 6, this large space is $\mathbb{Z}_q$ or $\mathbb{Z}_q^*$ where q is a fixed, large prime number (e.g., of 160 bits in length). 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 total feature descriptor b, the shares $\{s_i^{b(i)}\}_{1\le i\le m}$ can be used to reconstruct hpwd_a. These shares are arranged in an "instruction table":

	$< t_i$	$\ge t_i$
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 "$\ge t_i$" columns) with pwd_a. This (encrypted) table is stored in the system. In the lth 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 $\phi_i(a,l)$ indicates which of the two values in the ith row should be used in the reconstruction to find hpwd_a: if $\phi_i(a,l) < t_i$, 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 $\mu_{ai} < t_i$, 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 distinguishing features.

Not all secret sharing schemes satisfying the properties described above will suffice for our technique, since to defend against an attacker who captures the table, the shares must be of a form that does not easily reveal if a guessed password pwd' successfully decrypts the table. Abstractly, suppose the attacker is given either a completely random table (as would be obtained by decrypting the table with an incorrect password pwd') or a plaintext instruction table that is formed according to our scheme as outlined above (as would be obtained if $\mathsf{pwd}' = \mathsf{pwd}_a$), each with equal likelihood. The security of our scheme against an attacker requires that it be costly for the attacker to determine which type of table it is, i.e., random or not. In other words, if the probability that the attacker correctly guesses the type of table is significantly better than $1/2$ after performing up to Δ computational steps, then Δ should be large. Here, Δ corresponds directly to the computational slowdown that the attacker will suffer. The value of Δ will generally depend on a number of factors, not least of which is the distribution on which features are distinguishing and, for each distinguishing feature ϕ_i, the probability that $\mu_{ai} < t_i$ (versus $\mu_{ai} \ge t_i$). Since these distributions are not known, we are not able to mathematically prove that the instances of our schemes we propose in Sects. 5 and 6 will yield large values for Δ, even under reasonable cryptographic assumptions. However, in Sect. 8 we give empirical evidence that these distributions are sufficiently nonconcentrated to suggest that Δ can be substantial in practice.

There are numerous extensions and variations on this scheme:

1. One extension is to combine the scheme 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 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 [20]. In the latter case, the extra salt results in an additional work factor that the attacker must overcome.
2. A second variation is to correct some number of "errors" in the user's typing. For example, if the reconstruction dictated by the feature measurements fails to produce hpwd_a, then the login program can attempt m additional reconstructions, each identical to the first but with the value taken from one row i replaced with the other value in row i. In this way, if the login measurements are similar to the user's previous measurements in all but one feature, the login will still succeed. This "error correction" can naturally be extended to accommodate a larger number of errors.
3. A third extension is to divide the range of each feature ϕ_i into more than two regions and introduce additional, corresponding columns for these regions in the instruction table. For example, rather than having the two regions "$< t_i$" and "$\ge t_i$" as above, there could be thresholds t_{i1} and t_{i2} with columns in the instruction table for "$< t_{i_1}$", "$\ge t_{i1}$ and $\le t_{i2}$", and "$> t_{i2}$". The main challenges when using these variations are to ensure that the instruction table is costly to distinguish from random, and that all regions are roughly equally likely to contain μ_{ai} for a randomly chosen account a. Otherwise, an off-line attacker could discard a guessed password pwd' after examining only those components of the decrypted instruction table that correspond to the high probability regions for each feature.

We omit further discussion of the first and third of these extensions from this paper. In particular, we focus on the original two-column version described above, and leave the study of fruitful extensions to multiple columns to future work. We will omit discussion of the second extension above (error correction) from Sects. 5 and 6, but will return to this in Sect. 7.

5 An instance using polynomials

In this section we describe an instance of the technique of Sect. 4 using Shamir's secret sharing scheme [31].

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 $f_a(0) = \mathsf{hpwd}_a$, where $q > 2m+1$ is a large prime (e.g., of size 160 bits). 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 Sect. 5.4 to be more secure.

5.1 Stored data structures and initialization

Let G be a pseudorandom function family and P be a pseudorandom permutation family [29] such that for any key K, $G_K : \mathbb{Z}_q \to \mathbb{Z}_q$ and $P_K : \mathbb{Z}_q^* \to \mathbb{Z}_q^*$.[1] There are three data structures stored in the system per account:

1. A randomly chosen element $r \in \{0,1\}^\kappa$ where κ is a security parameter of, say, $\kappa = 160$.
2. 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 $\langle i, \alpha_{ai}, \beta_{ai} \rangle$ for each feature ϕ_i. Here,

$$\alpha_{ai} = y_{ai}^0 + G_{r,\mathsf{pwd}_a}(2i) \bmod q$$
$$\beta_{ai} = y_{ai}^1 + G_{r,\mathsf{pwd}_a}(2i+1) \bmod q$$

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 $\{(P_r(2i), y_{ai}^0), (P_r(2i+1), y_{ai}^1)\}_{1 \le i \le m}$ lie on a single, random polynomial $f_a \in \mathbb{Z}_q[x]$ of degree $m-1$ such that $f_a(0) = \mathsf{hpwd}_a$.
3. 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 that pwd_a was changed, login attempts $j_1, \ldots, j_l$ to a were successful, then this file contains $\phi_i(a, j)$ for each $1 \le i \le m$ and $j \in \{j_{l-h+1}, \ldots, j_l\}$. 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 encrypted with hpwd_a using a symmetric cipher. The size of this file should remain constant over time (e.g., it must be padded when necessary), so that its size yields no information about how many successful logins there have been.

5.2 Logging in

The login program takes the following steps whenever the user attempts to log into a. Suppose that this is the lth 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 r (stored in nonvolatile storage) and pwd' to "decrypt" α_{ai} if $\phi_i(a,l) < t_i$, and uses r and pwd' to "decrypt" β_{ai} otherwise. Specifically, it assigns

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

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 [24, 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 $r' \in \{0,1\}^\kappa$ and replaces r with r' in nonvolatile storage. It then 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:

$$\mu_{ai} < t_i \Rightarrow f_a(P_{r'}(2i)) = y_{ai}^0 \wedge f_a(P_{r'}(2i+1)) \ne y_{ai}^1$$
$$\mu_{ai} \ge t_i \Rightarrow f_a(P_{r'}(2i)) \ne y_{ai}^0 \wedge f_a(P_{r'}(2i+1)) = y_{ai}^1$$

For all other features ϕ_i (i.e., those for which $|\mu_{ai} - t_i| \le k\sigma_{ai}$, or all features if there have been fewer than h successful logins to this account since initialization;

[1] That is, a polynomially bounded adversary not knowing K cannot distinguish between $G_K(x)$ ($P_K(x)$) and a randomly chosen element of $\mathbb{Z}_q$ (respectively, $\mathbb{Z}_q^*$), even if he is first allowed to examine $G_K(\hat{x})$ (respectively, $P_K(\hat{x})$) 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. Though the key K for P will be available to the primary attacker considered in this paper, we assume that the pseudorandom property of P destroys any useful structure among the outputs of P.

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

see Sect. 3.1) the login program sets $y^0_{ai} = f_a(P_{r'}(2i))$ and $y^1_{ai} = f_a(P_{r'}(2\mathrm{i}{+}1))$.

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

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

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

Step 4 above is particularly noteworthy for two reasons. Firstly, due to this step, the value r and polynomial f_a are changed 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 undistinguishing 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. Secondly, 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 or not each feature is distinguishing is recomputed in each successful login. So, a feature that was previously distinguishing can become undistinguishing and vice versa. Recomputing the set of distinguishing features allows our scheme to adapt to gradual changes in the user's typing pattern over time.

5.3 Security

Consider the attacker who obtains the history file and instruction table of account a, 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_{ai}, y^1_{ai} 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. Therefore, to argue the benefits of this scheme, we have to show two things: (1) 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, and (2) that the cost to the attacker of finding hpwd_a is generally greater by a significant multiplicative factor.

It is clear that searching for pwd_a is not made easier in our scheme. The attacker has available only r, 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 useful information about pwd_a. Furthermore, all other data available to the attacker is encrypted with hpwd_a.

The more interesting consideration in our scheme is the degree of security increase over a traditional password scheme. Suppose that the attacker captured the history file and instruction table after $l \geq h$ successful logins to a, and let d be the number of distinguishing features for this account in the lth login. When guessing a password pwd', the attacker can decrypt each field α_{ai} and β_{ai} using pwd' to yield points $(P_r(2i), \hat{y}^0_{ai})$ and $(P_r(2\mathrm{i}{+}1), \hat{y}^1_{ai})$, respectively, for $1 \leq i \leq m$. Note that $\hat{y}^0_{ai} = y^0_{ai}$ and $\hat{y}^1_{ai} = y^1_{ai}$, where y^0_{ai}, y^1_{ai} are as generated in Step 5, if and (with overwhelming probability) only if $\mathsf{pwd}' = \mathsf{pwd}_a$. Therefore, there exists a total feature descriptor b such that $\{(P_r(2i + b(i)), \hat{y}^{b(i)}_{ai})\}_{1 \leq i \leq m}$ 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 total feature descriptors 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 $\Delta = 2^m$. In practice, the slowdown that the attacker suffers may be less because user typing patterns are not random. In Sect. 8 we use empirical data to quantify the degree of security achieved against this form of attack, and show that it is substantial.

However, the attacker has potentially more powerful attacks against this scheme using the $2m$ points $\{(P_r(2i), \hat{y}^0_{ai}), (P_r(2i+1), \hat{y}^1_{ai})\}_{1 \leq i \leq m}$, 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 polynomial of degree $m-1$; 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 $d \leq (2 - \sqrt{2})m \approx 0.585m$ using error-correcting techniques [11]. These techniques do not attack our scheme directly, since our analysis in Sect. 8 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, m may be too small in our scenario for these techniques to offer substantial benefit over the exhaustive approach above. However, because these techniques might be improved with application-specific knowledge – for example, that in the second case at least one of $(P_r(2i), \hat{y}^0_{ai})$ and $(P_r(2i+1), \hat{y}^1_{ai})$ lies on f – it is prudent to look for schemes that confound the use of error-correcting techniques. This is the goal of Sect. 5.4.

5.4 A variation using exponentiation

In this section we present a minor variation of the scheme presented in Sects. 5.1 and 5.2, to which we refer as the "original" scheme here. The scheme described in this section is more secure than the original scheme in several ways, as 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_{ai} and β_{ai} in the instruction table, the values

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

are stored instead. Intuitively, since the attacker cannot compute discrete logarithms modulo p, this hides y_{ai}^0, y_{ai}^1 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 instruction table can yield no more information about $f_a(0)$ to the attacker than can the original, since the attacker can easily transform any instruction table in the original scheme to an instruction table for this variation by computing $g^{\alpha_{ai}} \bmod p$ and $g^{\beta_{ai}} \bmod p$ for each α_{ai} and β_{ai}. Second, Bleichenbacher and Nguyen, in their analysis of the noisy polynomial interpolation problem [2], provide good evidence that this variation is indeed significantly stronger than the original. In particular, error-correcting algorithms such as [11] 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 somewhat differently, since $f_a(0)$ is hidden even from the login program. The resulting login process for the lth login attempt to a is as follows. Let pwd' denote the sequence of characters that the user has typed:

1. For each ϕ_i, the login program assigns

$$(x_i, z_i) = \begin{cases} (P_r(2i),\ (\gamma_{ai})g^{-G_{r,\mathsf{pwd}'}(2i)} \bmod p) \\ \hfill \text{if } \phi_i(a,l) < t_i \\ (P_r(2i+1),\ (\delta_{ai})g^{-G_{r,\mathsf{pwd}'}(2i+1)} \bmod p) \\ \hfill \text{if } \phi_i(a,l) \geq t_i \end{cases}$$

   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 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.
4. The login program generates a new random $r' \in \{0,1\}^\kappa$ and replaces r with r' in nonvolatile storage. It then 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., $|\mu_{ai} - t_i| > k\sigma_{ai}$, the login program chooses new random values $y_{ai}^0, y_{ai}^1 \in \mathbb{Z}_q^*$ subject to the following constraints:

$$\mu_{ai} < t_i \Rightarrow f(P_{r'}(2i)) = y_{ai}^0 \wedge f(P_{r'}(2i+1)) \neq y_{ai}^1$$
$$\mu_{ai} \geq t_i \Rightarrow f(P_{r'}(2i)) \neq y_{ai}^0 \wedge f(P_{r'}(2i+1)) = y_{ai}^1$$

   For all other features ϕ_i – i.e., those for which $|\mu_{ai} - t_i| \leq k\sigma_{ai}$, or all features if there have been fewer than h successful logins to this account since initialization (see Sect. 3.1) – the login program sets $y_{ai}^0 = f(P_{r'}(2i))$ and $y_{ai}^1 = f(P_{r'}(2i+1))$.
6. The login program replaces the instruction table with a new table with an entry of the form $\langle i, \gamma'_{ai}, \delta'_{ai} \rangle$ for each feature ϕ_i. Here,

$$\gamma'_{ai} = \mathsf{hpwd}' \cdot g^{y_{ai}^0 + G_{r',\mathsf{pwd}'}(2i)} \bmod p$$
$$\delta'_{ai} = \mathsf{hpwd}' \cdot g^{y_{ai}^1 + G_{r',\mathsf{pwd}'}(2i+1)} \bmod p$$

   where y_{ai}^0, y_{ai}^1 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 modulo p due to the construction of γ'_{ai} and δ'_{ai} 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 describe a second candidate instance of the technique outlined in Sect. 4. This solution addresses a potential weakness of the scheme of Sect. 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, for example, 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 secret sharing scheme and the corresponding instance of our method.

6.1 A secret sharing scheme based on vector spaces

We first briefly present the secret sharing scheme based on vector spaces that we use in our construction. Vector-space secret sharing schemes have been studied extensively (e.g., [4]). The scheme presented here is based on similar ideas, though it is tuned for use in our application. For example, this sharing scheme requires no information other than the shares to reconstruct the secret; i.e., no additional "public information" is needed. Moreover, the m pairs of shares $\{s_i^0, s_i^1\}_{1\le i\le m}$ are constructed from the secret $s \in \mathbb{Z}_q^*$ so that s can be reconstructed from a set of shares if and only if that set contains one share from each pair. That is, in secret sharing parlance, the *access structure* is $\Gamma = \{\{s_i^{b(i)}\}_{1\le i\le m} \mid b \in \mathcal{T}\}$, which corresponds precisely to our needs in this application. The secret sharing scheme works as follows:

1. The dealer[3] chooses random, linearly independent (column) vectors $\underline{w}_1^0, \ldots, \underline{w}_m^0 \in \mathbb{Z}_q^m$ such that $s = \det(\underline{w}_1^0, \ldots, \underline{w}_m^0) \bmod q$.
2. The dealer computes m random vectors $\underline{u}_1, \ldots, \underline{u}_m$ so that
$$\forall b \in \mathcal{T} : \det(\underline{u}_1^{b(1)}, \ldots, \underline{u}_m^{b(m)}) \equiv 1 \bmod q \qquad (1)$$
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 0 in all other positions.
3. The dealer computes vectors $\underline{w}_1^1, \ldots, \underline{w}_m^1 \in \mathbb{Z}_q^m$ as $\underline{w}_i^1 = (\underline{w}_1^0, \ldots, \underline{w}_m^0) \cdot \underline{u}_i$, with all computations performed modulo q.
4. Each share s_i^j is defined as $s_i^j = \underline{w}_i^j$.

An efficient algorithm to generate $\underline{u}_1, \ldots, \underline{u}_m$ so that they contain significant randomness and satisfy (1) is as follows. The dealer 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 dealer sets $(\underline{u}_1, \ldots, \underline{u}_m) = \Pi \cdot U' \cdot \Pi^{-1}$ where $\Pi = (\underline{\pi}_1, \ldots, \underline{\pi}_m)$ is any permutation matrix (i.e., the identity matrix with columns permuted). For the purposes of secret sharing alone, setting Π to be the identity matrix suffices. In Sect. 6.2, however, we will tune Π for our application.

Theorem 1. *The scheme in Steps 1–4 is a secret sharing scheme for access structure* Γ.

Proof. First we show that for any $b \in \mathcal{T}$, the secret s can be reconstructed from $\{s_i^{b(i)}\}_{1\le i\le m}$ by arranging the vectors $\underline{w}_i^{b(i)}$ in ascending order of i in a matrix and computing the corresponding determinant:

$$\begin{aligned}\det(\underline{w}_1^{b(1)}, \ldots, \underline{w}_m^{b(m)}) \bmod q \\ &= \det(\underline{w}_1^0, \ldots, \underline{w}_m^0) \cdot \det(\underline{u}_1^{b(1)}, \ldots, \underline{u}_m^{b(m)}) \bmod q \\ &\overset{(1)}{=} \det(\underline{w}_1^0, \ldots, \underline{w}_m^0) \bmod q \\ &= s \bmod q\end{aligned}$$

We now prove that given any set of shares that contains neither $\underline{w}_l^0$ nor $\underline{w}_l^1$, for some l, any element of $\mathbb{Z}_q^*$ remains possible for the secret. Let $u_{i,j}$ denote the ith element of $\underline{u}_j$. By the construction above,

$$u_{l,j} \cdot \underline{w}_l^0 = \underline{w}_j^1 - \sum_{\substack{1\le i\le m \\ i\ne l}} u_{i,j} \cdot \underline{w}_i^0 \bmod q \qquad (2)$$

for any $j \ne l$ (and $1 \le j \le m$). Let $\hat{u}_{l,j}$, $j \ne l$, and $\hat{\underline{w}}_l^0$ be values that satisfy (2) for $j \ne l$ when $u_{l,j} = \hat{u}_{l,j}$ and $\underline{w}_l^0 = \hat{\underline{w}}_l^0$. Such values must exist due to the construction of the shares. Then for any $x \in \mathbb{Z}_q^*$, setting $u_{l,j} = \hat{u}_{l,j} \cdot x^{-1} \bmod q$, $j \ne l$, and $\underline{w}_l^0 = \hat{\underline{w}}_l^0 \cdot x \bmod q$ also satisfies (2) for $j \ne l$. Since $s = \det(\underline{w}_1^0, \ldots, \underline{w}_m^0) \bmod q$ and since

$$\begin{aligned}\det(\underline{w}_1^0, \ldots, \hat{\underline{w}}_l^0 \cdot x, \ldots, \underline{w}_m^0) \bmod q \\ &= x \cdot \det(\underline{w}_1^0, \ldots, \hat{\underline{w}}_l^0, \ldots, \underline{w}_m^0) \bmod q\end{aligned}$$

it follows that any element of $\mathbb{Z}_q^*$ remains possible for the secret. □

6.2 Hardened password generation based on vector spaces

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

$$\underline{\alpha}_{ai} = \underline{v}_{ai} + G_{r,\mathsf{pwd}_a}(2i) \bmod q$$
$$\underline{\beta}_{ai} = \underline{v}_{ai} + G_{r,\mathsf{pwd}_a}(2i+1) \bmod q$$

[3] Secret sharing schemes are often presented in terms of a *dealer* who creates and distributes shares of the secret. We adopt this terminology for this section.

where r is chosen randomly from $\{0,1\}^\kappa$ and stored in nonvolatile storage, and G is a pseudorandom function family such that for any K, $G_K : \mathbb{Z}_q \to \mathbb{Z}_q^m$. Note that at initialization, and more generally when there are no distinguishing features, the "shares" in $\underline{\alpha}_{ai}$ and $\underline{\beta}_{ai}$ are the same (albeit encrypted under different outputs from G_{r,pwd_a}). This is reasonable since when there are no distinguishing features, our approach offers no additional security over that offered by pwd_a.

The login process for the lth login attempt to a is as follows. Let pwd' denote the sequence of characters that the user has typed:

1. For each ϕ_i, the login program assigns

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

   The login program now holds m vectors $\{\underline{v}_i\}_{1\leq i\leq m}$.
2. The login program decrypts the history file, once with $\det(\underline{v}_1, \ldots, \underline{v}_m) \bmod q$ and once with $-\det(\underline{v}_1, \ldots, \underline{v}_m) \bmod q$. If either of these decryptions yields a properly formed plaintext history file, then the login program sets hpwd' to be whichever of $\pm\det(\underline{v}_1, \ldots, \underline{v}_m) \bmod q$ successfully decrypted the history file. If neither decryption yielded a properly formed history file, then login is deemed unsuccessful and the login procedure halts 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, and the feature descriptor b_a for this account. The login program 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 $r' \in \{0,1\}^\kappa$ and replaces r with r' on stable storage. It then generates new random, linearly independent 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 $\underline{v}^0_{ai} = \underline{v}^1_{ai} = \underline{w}_i$ for each $1 \leq i \leq m$.
   (b) Otherwise, the login program generates new random vectors $\underline{u}_1, \ldots, \underline{u}_m \in \mathbb{Z}_q^m$ such that

$$\forall b \in \mathcal{T}(b_a) : \det(\underline{u}_1^{b(1)}, \ldots, \underline{u}_m^{b(m)}) \equiv \pm 1 \bmod q \quad (3)$$

   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. Then for each distinguishing feature ϕ_i, the login program chooses new random vectors $\underline{v}^0_{ai}, \underline{v}^1_{ai} \in \mathbb{Z}_q^m$ subject to the following constraints, where $W = (\underline{w}_1, \ldots, \underline{w}_m)$:

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

   For all other features ϕ_i – that is, those for which $|\mu_{ai} - t_i| \leq k\sigma_{ai}$ – the login program sets $\underline{v}^0_{ai} = \underline{w}_i$ and $\underline{v}^1_{ai} = W \cdot \underline{u}_i$.
6. The login program replaces the instruction table with a new table with an entry of the form $\langle i, \underline{\alpha}'_{ai}, \underline{\beta}'_{ai}\rangle$ for each feature ϕ_i. Here

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

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

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

The vectors $\underline{u}_1, \ldots, \underline{u}_m$ can be computed efficiently as described in Sect. 6.1. The construction is tuned to the application by allowing ±1 for the diagonal entries of U' and choosing the permutation matrix Π subject to the constraint that if $\phi_{i_1}, \ldots, \phi_{i_d}$ are the distinguishing features for this account, then $\{\underline{\pi}_j\}_{1\leq j\leq d} = \{\underline{e}_{i_j}\}_{1\leq j\leq d}$. In particular, this stipulation ensures (with high probability) that $\underline{v}^0_{ai} \neq \underline{v}^1_{ai}$ for each $1 \leq i \leq m$ when created in Step 5b.

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

As in Sect. 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 $\hat{\underline{v}}^1_{ai}$ will be expressible as a linear combination of $\hat{\underline{v}}^0_{ai}$ and either $\hat{\underline{v}}^0_{aj}$ or $\hat{\underline{v}}^1_{aj}$ 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 Implementation

We have implemented the method of Sect. 5.4 to experiment with our techniques. Our implementation provides three types of functions: initialization, login, and recovery. Initialization is implemented as previously described. Login differs from the description in Sect. 5.4 by correcting a limited number of errors in the user's typing (see Sect. 4). Specifically, the algorithm BuildHpwd that attempts to generate hpwd_a is shown in Fig. 1. In the lth login attempt, the parameters to this algorithm are the entered password pwd', a (total) feature descriptor b defined by

$$b(i) = \begin{cases} 0 & \text{if } \phi_i(a,l) < t_i \\ 1 & \text{if } \phi_i(a,l) \geq t_i \end{cases}$$

and a maximum number maxErrors of errors to correct. The algorithm then performs a reconstruction corresponding to each total feature descriptor b' that differs from b on at most maxErrors values, i.e., such that the Hamming distance between b and b' is at most maxErrors. (In Fig. 1, the Hamming distance between b and b' is denoted by $\mathsf{distance}(b,b')$, and the result of reconstructing according to b' is denoted $\mathsf{reconstruct}(\{s_i^{b'(i)}\}_{1\leq i\leq m})$.) Note that these feature descriptors b' are examined in order of increasing Hamming distance from b. In this way, the algorithm quickly completes when b extends or is similar to the account's feature descriptor as computed during the previous successful login.

This design of BuildHpwd also renders it useful as a recovery routine. 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 show in Sect. 8 that this should be a rare occurrence for an appropriately tuned system, but it is nevertheless one that must be anticipated. The BuildHpwd algorithm can be used as a recovery algorithm by setting $\mathsf{maxErrors} = m$; that is, 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. Again, because BuildHpwd examines feature descriptors b' in order of increasing Hamming distance from b, recovery will succeed quickly when a user types similarly to how she previously typed the password. 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.

```
function BuildHpwd(pwd′, b, maxErrors) {
    decrypt instruction table with pwd′ to yield shares $\{s_i^0, s_i^1\}_{1\leq i\leq m}$
    for errors = 0 ... maxErrors {
        for each b′ such that distance(b, b′) = errors {
            hpwd ← reconstruct($\{s_i^{b'(i)}\}_{1\leq i\leq m}$)
            if (hpwd decrypts history file successfully)
                return hpwd
        }
    }
    return 0 /* failure */
}
```

Fig. 1. Algorithm for building hpwd

In experimenting with this implementation, we have found that our techniques should be accompanied by user education regarding the fact that the user's typing patterns are being taken into account in the login process. In particular, a user should be instructed that when she fails to log in the first time, she should avoid the temptation to type the password particularly slowly or methodically on the second try, as people often do for normal password logins. Rather, the user should simply attempt to type the password as she normally would.

8 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 used 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 had eight characters (i.e., $m = 15$), but because it may still be in active use, we cannot disclose it here. Login measurements were recorded for approximately 6 months. For the discussion in this section, we use data gathered from the 20 users for which we have at least 5 logins recorded in which she typed the correct text password on her usual keyboard, as reported by the user. In total, this analysis is based on 481 recorded logins, in all of which the user typed the correct password.

The goal of our experiment was to empirically evaluate the number of distinguishing features for the average user, the entropy of users' distinguishing features, and the false negative rate associated with our technique. 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 distinguishing features ϕ_i, $\mu_{ai} > t_i$ (where a is the user's account), and 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 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 at least as important to our scheme as the *number* of distinguishing features.

The false negative rate measured in our experiments is the percentage of login attempts by the legitimate user that would have failed to generate the correct hardened password, due to variations in the measured features of her typing. Clearly the false negative rate is essential to evaluating the usability of our technique. For completeness, we also evaluate the false positive rate, calculated as the percentage of impersonations per login (i.e., the percentage of accounts other than the user's own to which each login would have succeeded), averaged over all logins by all users. The false positive rate is a common measure of biometric techniques, but again is not of primary concern to us here; see Sect. 3.2.

We evaluated these measurements for varying values of k, where a feature ϕ_i is distinguishing if $|\mu_{ai} - t_i| > k\sigma_{ai}$ (see Sect. 3.1), and for varying numbers of error corrections (the $\mathsf{maxErrors}$ parameter of Fig. 1). In general, a lower value of k increases the number of distinguishing features per user and thus increases the sensitivity of the 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.

To simplify our analysis, all of the recorded logins for an account a were used to compute μ_{ai} and σ_{ai}. In particular, we ignored the parameter h. A side effect of this simplification is that unlike varying k, varying the number of errors corrected does not impact our computed entropy measurement or the average number of distinguishing features for given values for $t_1, \ldots, t_m$. Varying the number of errors corrected does, however, factor into our calculation of false positives and false negatives.

8.1 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 present in hpwd_a for a randomly chosen account a.

More specifically, we would like to compute the entropy of the feature descriptor of a randomly chosen account. 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 the feature descriptor of a randomly chosen account a 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, for example, b such that $b(i) = 0$ for each $1 \leq i \leq m$, and b such that $b(i) = 1$ for each $1 \leq i \leq m$.[4]

As a tool to better capture the entropy available due to keystrokes, we define a *cover* to be a function $\mathcal{C}$ from accounts to total feature descriptors such that $\mathcal{C}(a) \in \mathcal{T}(b_a)$. That is, a cover maps each account a with feature descriptor b_a to a (total) feature descriptor that extends b_a. Given a cover, we can evaluate the entropy of $\mathcal{C}(a)$ under a 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 $\mathcal{C}$. So, in the example of the previous paragraph, all accounts can map to at most two such descriptors.

Guessing entropy [5, 21] is a natural way to define the entropy of a cover. Let $\mathrm{Img}(\mathcal{C}) = \{b \mid \exists a \in A : \mathcal{C}(a) = b\}$, and $w_{\mathcal{C}}(b) = |\{a \in A \mid \mathcal{C}(a) = b\}|/|A|$. If we denote $\mathrm{Img}(\mathcal{C}) = \{b_1, \ldots, b_l\}$ such that $w_{\mathcal{C}}(b_1) \geq w_{\mathcal{C}}(b_2) \geq \ldots \geq w_{\mathcal{C}}(b_l)$, then the guessing entropy of the cover $\mathcal{C}$ is

$$E_{\mathcal{C}} = \sum_{i=1}^{|\mathrm{Img}(\mathcal{C})|} (i \cdot w_{\mathcal{C}}(b_i))$$

Intuitively, the guessing entropy is the expected number of feature descriptors in $\mathrm{Img}(\mathcal{C})$ that 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" $w_{\mathcal{C}}(b)$ of each element in $\mathrm{Img}(\mathcal{C})$ (plus the password itself) and thus examines elements of $\mathrm{Img}(\mathcal{C})$ in an optimal order to minimize this expected value. As described above, in the worst case an attacker will know $\mathrm{Img}(\mathcal{C})$ and $w_{\mathcal{C}}$ for a cover $\mathcal{C}$ that minimizes $E_{\mathcal{C}}$, and so it is this cover we use in our computations of Sect. 8.2. In practice, however, it is unlikely that an attacker could know such a cover, and so we believe our analysis to be very conservative.

8.2 Performance evaluation

Our analysis methodology consisted of the following steps for each value of k. We first found values D and L that

[4] This example also illustrates that the use of pattern classifiers that aim to separate a population given some similarity measure (e.g., linear discriminant analysis; see [7]) would not necessarily yield good results for our purposes. That is, in the example above the distinguishing feature for each account could be used by a classifier to perfectly separate the population. Nevertheless, our scheme would not leverage this separation to make the attacker's task more difficult, since again the attacker need only attempt two different feature descriptors.

maximized the guessing entropy, when $t_i = D$ for each duration feature ϕ_i and when $t_i = L$ for each latency feature ϕ_i. More specifically, for each pair of candidate integer values D and L in the ranges $50\,\mathrm{ms} \leq D \leq 150\,\mathrm{ms}$ and $50\,\mathrm{ms} \leq L \leq 150\,\mathrm{ms}$, we computed the feature descriptor for each account and a cover $\mathcal{C}$ for these feature descriptors with minimum guessing entropy. We then chose a pair D, L that resulted in the highest guessing entropy from this calculation. We thereby captured the guessing entropy faced by the attacker in the case that the system was configured with optimal values of D and L.

False negatives were computed by calculating the percentage of each account's logins that would have failed in logging into that user's account, given these values of D, L and a number of error corrections. False positives were computed by calculating the average percentage of impersonations per login given these values of D and L and a number of error corrections. If there were multiple D, L pairs that yielded the same maximum guessing entropy as computed above, then D and L were chosen from among them as the pair yielding the smallest sum of false positives and false negatives. The average number of distinguishing features d per user given k, D, and L was then computed. Note that even though the number of error corrections is not factored into the calculation of the number of distinguishing features for a given set of k, D, and L values, it does factor into the calculation of false negatives and positives, and consequently into the choice of the best D and L. This results in a different curve per number of error corrections for the average number of distinguishing features for the best D, L pair. These curves are are qualitatively the same, however, and so Fig. 3 shows the average of these curves over the cases of 0–3 corrections.

The results of this analysis are shown in subsequent figures. In Fig. 2, the guessing entropy is shown for values of k ranging from 0.1 to 0.65. Note that the choice of $k = 0.1$ yields a guessing entropy of 10.5, which is the maximum possible guessing entropy given the number of users (20) in our study. Moreover, this choice yields roughly 14 distinguishing features for the average account, as shown in Fig. 3. Naturally, the entropy and number of distinguishing features decreases as k increases.

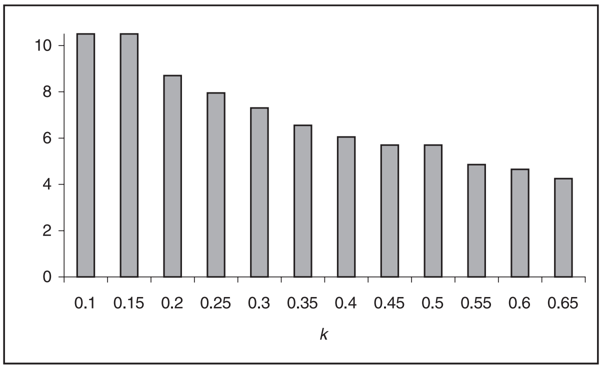

Fig. 2. Guessing entropy as a function of k

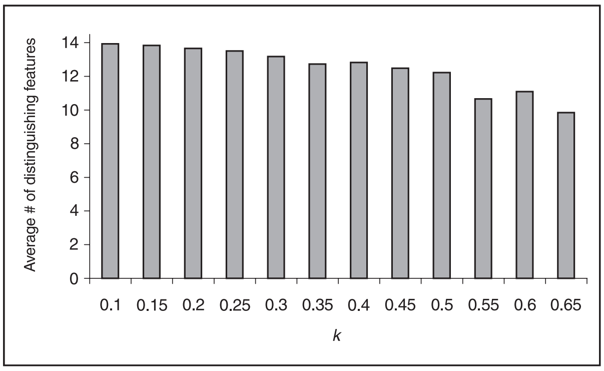

Fig. 3. Average number of distinguishing features

The implications of these numbers become clear only in light of the false negative curves for varying values of k and varying numbers of error corrections. These curves are shown in Fig. 4. Each of these four graphs shows the false negative percentages for a fixed number of error corrections. Again, for completeness, false positive curves are also included.

Figure 4 demonstrates that the value of k yielding a reasonable false negative rate when there are no error corrections is very large, and indeed beyond the range of analysis that we were able to complete. For such a large value of k, there will tend to be few distinguishing features per account and low entropy associated with keystroke patterns; see Figs. 2 and 3. Consequently, we conclude that correcting no errors (and similarly, one error) is a suboptimal approach. In examining the graphs for three error corrections, we see that the false negative rate is under 10% in the vicinity of $k = 0.2$. This is a reasonable false negative rate that is coupled with high guessing entropy; see Fig. 2. Therefore, choosing a small value for k and correcting for three errors would seem to be a good configuration for our system.

We reiterate that the false positive curves in Fig. 4 are of little interest to us here (see Sect. 3.2). They do, however, show why we do not rely on keystroke patterns alone to authenticate users. Indeed, in the configurations we suggest above – for example, $k = 0.2$ and correcting three errors – the false positive rate is greater than 20%. Thus we do not employ keystrokes alone, but rather simply use them to augment the strength of passwords.

In the analysis above, we chose a D, L pair based on data that had been collected previously, and then computed various statistics using this pair. Since in practice D and L need to be set in advance of use, a natural question is how difficult it is to choose close-to-optimal values for D and L. To address this issue we illustrate in Fig. 5 how guessing entropy varies with D and L. There are two relevant points illustrated. First, there are significant ranges for both D and L that yield substantial guessing entropy. That is, guessing entropy does not appear highly sensitive to small variations of D and L. As a result, in practice it should be sim-

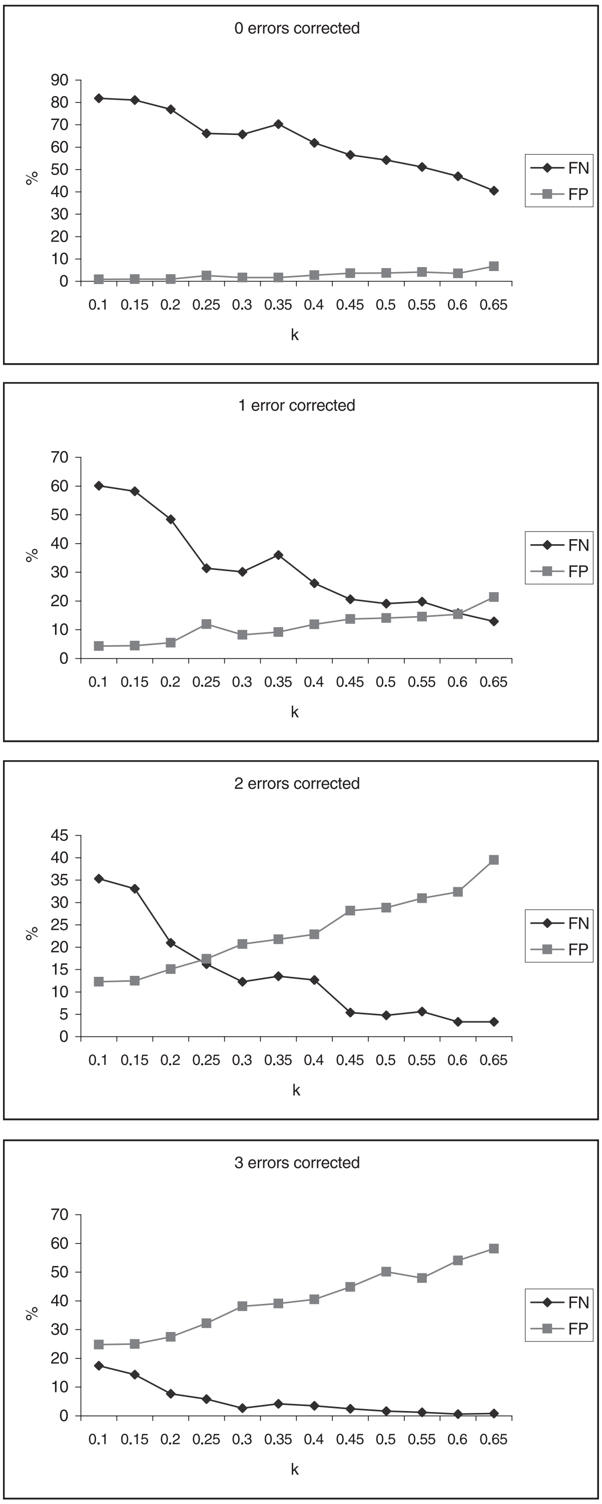

Fig. 4. False positive (FP) and false negative (FN) percentages

ple to choose D and L that yield significant entropy: our data suggests that $90\,\text{ms} \leq D, L \leq 110\,\text{ms}$ should suffice. Second, the guessing entropy is more sensitive to variations in D than in L. This is consistent with previous works that have found keystroke durations to be more variable among users than keystroke latencies (e.g., [27]).

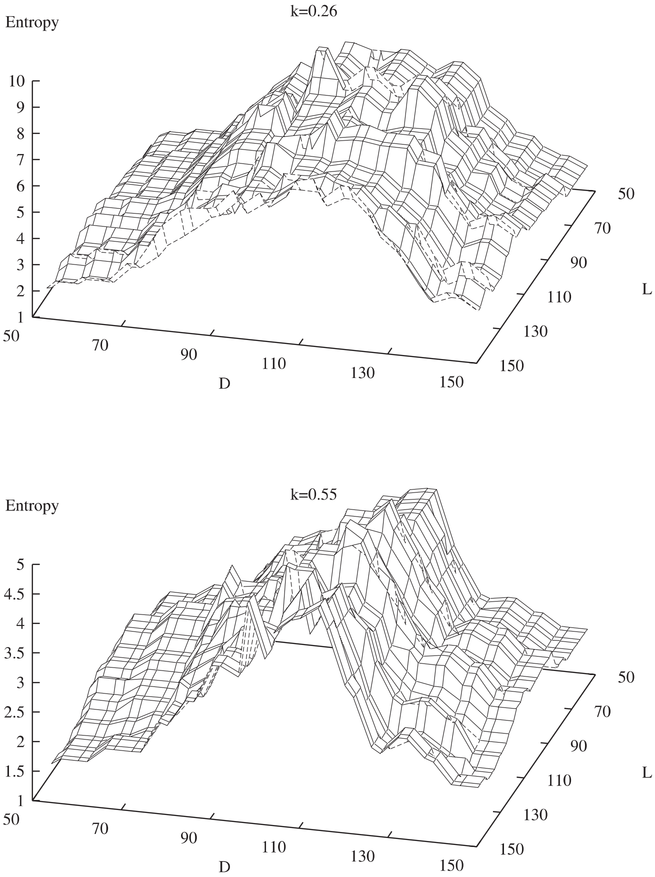

Fig. 5. Guessing entropy as a function of D and L

We remind the reader that all of the foregoing analysis is based on a relatively small trial of our techniques: 20 users, 481 logins, and 1 password. It is thus difficult to draw firm conclusions as to the effectiveness of our techniques, and further trials are thus appropriate. That said, all evidence in our trial suggests that our techniques offer significantly improved security and adequate usability.

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 that it would take an 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.

Our analysis suggests that 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. Furthermore, using recorded keystroke data we have provided 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.

Though we have empirically validated our techniques only for keystroke behavior, it should be clear that they naturally adapt to other biometrics. That is, for any repeatable, unpredictable physical phenomenon for which one can measure features, these features can be used in place of the keystroke durations and inter-keystroke latencies examined in this paper. Our ongoing work focuses on the use of other biometric measurements in this way (e.g., [24, 25]).

Acknowledgements. We are particularly thankful to Daniel Bleichenbacher, who suggested improvements to the original scheme of Sects. 5.1 and 5.2 to address potential weaknesses. We are grateful to Markus Jakobsson and Amin Shokrollahi for insightful discussions. Phil MacKenzie and the anonymous referees for the Sixth ACM Conference on Computer and Communications Security provided helpful comments that improved the presentation of an earlier version of this paper.

References

1. Bleha S, Slivinksy C, Hussein B (1990) Computer-access security systems using keystroke dynamics. IEEE Trans Pattern Anal Mach Intell 12:1217–1222
2. Bleichenbacher D, Nguyen P (2000) Noisy polynomial interpolation and noisy Chinese remaindering. In: Preneel B (ed) Advances in cryptology – EUROCRYPT 2000. (Lecture notes in computer science 1807) Springer, Berlin Heidelberg New York, pp 53–69
3. Bohannon P, Jakobsson M, Srikwan S (2000) Cryptographic approaches to privacy in DNA databases. In: Proceedings of the 2000 International Workshop on Practice and Theory in Public Key Cryptography, Melbourne, Australia, 18–20 January
4. Brickell EF (1989) Some ideal secret sharing schemes. J Combin Math Combin Comput 9:105–112
5. Cachin C (1997) Entropy measures and unconditional security in cryptography. (ETH series in information security and cryptography, vol 1) Hartung-Gorre, Konstanz, Germany
6. Davida GI, Frankel Y, Matt BJ (1998) On enabling secure applications through off-line biometric identification. In: Proceedings of the 1998 IEEE Symposium on Security and Privacy, Oakland, Calif., 3–6 May, pp 148–157
7. Duda R (1973) Pattern Classification and Scene Analysis. Wiley, New York
8. Ellison C, Hall C, Milbert R, Schneier B (2000) Protecting secret keys with personal entropy. Future Gen Comput Syst 16:311–318
9. Feldmeier D, Karn P (1990) UNIX password security – Ten years later. In: Advances in cryptology – CRYPTO 1989. (Lecture notes in computer science 435) Springer, Berlin Heidelberg New York
10. Gaines R, Lisowski W, Press S, Shapiro N (1980) Authentication by keystroke timing: some preliminary results. (Rand report R-256-NSF) Rand Corporation, Santa Monica, Calif.
11. Guruswami V, Sudan M (1998) Improved decoding of Reed-Solomon and algebraic-geometric codes. In: Proceedings of the 39th IEEE Symposium on Foundations of Computer Science, Palo Alto, Calif., 8–11 November, pp 28–37
12. Horng G (1995) Password authentication without using a password table. Inform Process Lett 55:247–250
13. Joyce R, Gupta G (1990) Identity authorization based on keystroke latencies. Comm ACM33:168–176
14. Juels A, Wattenberg M (1999) A fuzzy commitment scheme. In: Proceedings of the Sixth ACM Conference on Computer and Communication Security, Singapore, 1–4 November, pp 28–36
15. Klein D (1990) Foiling the cracker: a survey of, and improvements to, password security. In: Proceedings of the Second USENIX Security Workshop, August, 1990
16. Lin CH, Chang CC, Wu TC, Lee RCT (1991) Password authentication using Newton's interpolating polynomials. Inform Syst 16:97–102
17. Leggett G, Williams J (1988) Verifying identity via keystroke characteristics. Int J Man Mach Stud 28:67–76
18. Leggett G, Williams J, Umphress D (1989) Verification of user identity via keystroke characteristics. In: Carey JM (ed) Human factors in management information systems. Norwood, New York
19. Lennon RE, Matyas SM, Meyer CH (1981) Cryptographic authentication of time-invariant quantities. IEEE Trans Comm 29:773–777
20. Manber U (1996) A simple scheme to make passwords based on one-way functions much harder to crack. Comput Secur 15:171–176
21. Massey JL (1994) Guessing and entropy. In: Proceedings of the 1994 IEEE International Symposium on Information Theory, Trondheim, Norway, 27 June – 1 July, p 204
22. Menezes AJ, van Oorschot PC, Vanstone SA (1997) Handbook of applied cryptography. CRC, Boca Raton, Fla.
23. Monrose F, Rubin A (1997) Authentication via keystroke dynamics. In: Proceedings of the Fourth ACM Conference on Computer and Communications Security, Zurich, Switzerland, 2–4 April, pp 48–56
24. Monrose F, Reiter MK, Li Q, Wetzel S (2001) Cryptographic key generation from voice (extended abstract). In: Proceedings of the 2001 IEEE Symposium on Security and Privacy, Oakland, Calif., 13–16 May, pp 202–213
25. Monrose F, Reiter MK, Li Q, Wetzel S (2001) Using voice to generate cryptographic keys. In: Proceedings of the 2001 Speaker Recognition Workshop, Crete, Greece, 18–22 June, pp 237–242
26. Morris R, Thompson K (1979) Password security: A case history. Comm ACM 22:594–597
27. Omote K, Okamoto E (1999) User identification system based on biometrics for keystroke. In: Proceedings of the Second International Conference on Information and Communication Security, Sydney, Australia, 9–11 November
28. Power R (2001) 2001 CSI/FBI computer crime and security survey. Comput Secur Issues Trends 7
29. Rivest RL (1990) Cryptography. In: van Leeuwen J (ed) Handbook of theoretical computer science. Elsevier, Amsterdam, pp 717–755
30. Robinson JA, Liang VM, Chambers JA, MacKenzie CL (1998) Computer user verification using login string keystroke dynamics. IEEE Trans Syst Man Cybern 28:236
31. Shamir A (1979) How to share a secret. Comm ACM 22:612–613
32. Soutar C, Tomko GJ (1996) Secure private key generation using a fingerprint. In: Proceedings of the Cardtech/Securetech Conference, pp 245–252
33. Spafford E (1992) Observations on reusable password choices. In: Proceedings of the Third USENIX Security Symposium
34. Wu T (1999) A real-world analysis of Kerberos password security. In: Proceedings of the 1999 Network and Distributed System Security Symposium, San Diego, Calif., 3–5 February