A Zero-Knowledge Protocol for Keystroke Authentication

Noam Zilberstein and Jonathan Chen[*]
University of Pennsylvania

December 16, 2014

Abstract

We present a zero-knowledge protocol to authenticate users without the use of a password. Passwords have many inherent problems; several well known companies such as Dropbox and Apple have recently been targets of large scale security breaches in which passwords were compromised. Because of this risk, users are urged to use different passwords for each account. However, it is very inconvenient to create and remember enough secure passwords considering how many online accounts most people have today.

Many common websites such as Facebook and Dropbox are already encouraging the use of two-factor authentication to subvert these issues. These forms of authentication usually involve confirming the possession of a piece of hardware. Instead, we suggest an alternate authentication factor that does not require possession of additional hardware. Our authentication scheme is based on keystroke dynamics; users are authenticated based on the unique timing of their keystrokes when typing a particular phrase (such as their name). When typing, individuals have their own typing pattern consisting of lag and dwell time (how long a key is pressed) and rhythms (typing certain clusters of letters faster). This typing pattern translates into a typing signature which can be measured and used as a form of biometric data. Instead of sending this naked data through vulnerable communication channels, we design a zero-knowledge protocol where the server verifies the identity of a user without ever having to exchange the biometric data itself. In this way, if there was ever a breach of security, such as the case with Target or J.P. Morgan Chase, no useful information would be leaked.

1 Introduction

Using passwords for authentication is an ancient concept that continues to be the commonly accepted standard. For the sake of convenience, many people choose low-strength passwords and reuse them for multiple websites [5]. This is extremely insecure and begs the question: is there a better way to authenticate users?

Several fixes to this problem are already in place and work by augmenting the password with a secondary authentication factor. Websites that support two factor authentication include Amazon, Facebook, Google, Dropbox, and many others [1]. In each case, the first factor is a password and the second factor is some kind of trusted hardware such as a personal computer or smartphone. When users try to log in from an untrusted machine, they must confirm their identity by typing a code that is sent to a trusted source such as an SMS, email, or phone call. While this process greatly improves the security of a poorly chosen password, it is extremely cumbersome. It also fails if the user does not have access to a trusted piece of hardware. The solution presented in this paper uses trusted hardware that users have at all times: their hands.

[*] Advised by Nadia Henninger

We propose an authentication protocol based on keystroke dynamics, the unique timings associated with the way that a user types. Because individuals prefer their own hand layouts on a keyboard, their finger movements will be distinct. For example, if Alice keeps her left pinky on the *shift* key in rest position, then the *a* key will be covered by her ring finger and her index finger will be split between *d* and *f*. This will affect her typing speed on certain words, as opposed to Bob who uses his left pinky for the *a* and thus has both his index and middle fingers available for *d* and *f*. In this paper, we exploit this fact and use it as the sole authentication factor. Of course, this protocol could easily be extended to incorporate more factors. As stated before, keystroke dynamics act as a hardware factor without requiring users to carry any piece of physical hardware with them other than their hands. Login on an untrusted machine is therefore equally as secure as login from a trusted one, but it also requires no extra steps on behalf of the user.

A problem arises with this protocol in the case that the database containing users' keystroke data is compromised. In a password model, if the password is leaked, then the user can simply create a new one. However, in an authentication scheme based on keystroke dynamics, if an adversary gains knowledge of keystroke data, then the affected users can never confidently use keystroke dynamics to authenticate themselves again. In order to fix this, as little data as possible should be stored in the database. In fact, the protocol we present is zero-knowledge. This means that users will never at any point in the enrollment or authentication process jeopardize the secrecy of their keystroke timings.

In Section 2, an overview of prior and similar work is provided. Next, a summary of the methodology and techniques that are used is explained in Section 3. The protocol itself is also defined in Section 3; first, a standard (but insecure) protocol is defined as a point of comparison and then several improvements are made to arrive at the zero-knowledge version. The accuracy and viability of this model as well as the tradeoffs and challenges of zero-knowledge keystroke authentication are examined in Section 4. Finally, future work is discussed in Section 5.

2 Related Work

The final protocol that we present brings together many concepts from the fields of cryptography, complexity theory, and information theory. This section gives overviews of these ideas based on prior work.

2.1 Keystroke Dynamics

Long before computers were invented, written signatures were used to verify people's identities. Signatures are commonly used to confirm the acceptance of contracts and to verify credit card and bank transactions. Typing one's name into a computer is the digital age's moral equivalent to writing it down on a piece of paper. It is not considered a breach of security to see someone's signature written on a piece of paper; the security assumption is that another person cannot copy it to a sufficient level of precision. Similarly, people type their names with high enough frequency that they develop a muscle memory for this process [9].

Of course, the fact that people can consistently type their names with the same keystroke timings does not alone make keystroke dynamics a good authentication technique; it is necessary, but not sufficient. If many people type in very similar ways, then they will all be authenticated as each other. There must be sufficient entropy in the keystroke timings. In other words, a vector of keystroke timings must contain enough information to differentiate between users. In 1999, Monrose, Reiter, and Wetzel showed that there is a relatively small amount of entropy in a keystroke vector generated by typing a fixed 8-character password [8]. However, this does not rule out the possibility of keystroke based authentication. In the same way that long passwords are considered more secure than short ones, longer keystroke vectors contain more entropy than short ones. Many more bits of entropy are generated by typing "Arnold Schwarzenegger " than by typing "Jane Doe". Most people have names that are longer than eight characters, so the entropy will be boosted over the results of Monrose, Reiter, and Wetzel [8]. In addition, people can generally type their own name with more consistency than an arbitrary word (this conjecture was validated in our trials). This means that the protocol can have less slack than the one used to measure entropy in the experiments of Monrose, Reiter, and Wetzel [8].

In fact, despite potential entropy concerns, Araújo, Sucupira, Lizarraga, Ling, and Yabu-Uti showed that people can be authenticated by measuring keystroke timings with roughly 99% accuracy [3]. This means that the false acceptance rate (FAR) and false rejection rate (FRR) are both about 1%. The tolerance of the model can be tuned in order to obtain slightly different rates. The FAR and The FRR are inversely related; by increasing the false rejection rate, the false acceptance rate will decrease thereby making the protocol more secure (but harder to be authenticated). This idea will be examined more closely in Section 4.

So far, the threat model being used has only assumed that the adversary is an actual human being. That is, no person can accurately reproduce other people's keystrokes by watching them type (or any other means). In reality, the adversary may not be a human, but rather an automated script. If some user's keystroke timings were compromised, this information would not help another human impersonate the user. However, an adversary could write a script that produces keystrokes with the precise timings that are expected. In 2010, Stefan and Yao showed that given a pool of keystroke data for users other than the target, an automated bot cannot impersonate the target [11]. However, there is no defense against a bot that does know the timings of its target. This is why it is of utmost importance that the authentication protocol leaks no information about the user's keystroke timings.

2.2 Zero-Knowledge Proofs for Authentication

In the classical definition of a zero-knowledge proof, a prover P must convince a less powerful verifier V of some fact without giving away any information that would allow V to prove it himself [7]. An intuitive example of this is the following:

> Peggy would like to convince Victor that two cards are different colors, however Victor is color blind. First, Victor holds up the two cards and asks Peggy to identify the color of each one. Victor then labels the back of the two cards with Peggy's responses, shuffles them behind his back and asks Peggy to identify the color of each card again. If the cards indeed have different colors, then Peggy can easily identify them. However, if the cards are the same color then she can only give an answer that is consistent with the labels with probability $\frac{1}{2}$. By repeating this process several times, Victor can be sure that Peggy is not lying. However, Victor cannot recreate the proof himself because he is color blind.

Relating this to an authentication protocol, the user can be viewed as a prover who knows some secret and the server is a verifier that must confirm that the user knows the secret, but has no access to the secret itself. In the case of keystroke dynamics, the secret is the user's keystroke timings. In a sense users do not "know" their timings, but as stated in Section 2.1, they can consistently produce them. If users know the secret then they should always be able to prove their identities, but with high probability an imposter should not be able to prove knowledge of the secret. These conditions relate directly to the example above. If Peggy were cheating, then she would only have a small probability of guessing the labels correctly many times. In an authentication scheme, this

Prover (knows x)		**Verifier** (knows $h = g^x \mod p$)
Randomly choose $r \in \mathbb{Z}_g$, let $a = g^r$	$\xrightarrow{a}$	
	$\xleftarrow{b}$	Randomly choose $b \in \mathbb{Z}_g$
Let $c = r + xb$	$\xrightarrow{c}$	$ah^b \overset{?}{=} g^c$

Figure 1: Zero-Knowledge Proof for the Discrete Log Problem

failure probability should be negligibly small. In other words, the probability that anyone can prove that they are a certain user without knowing that user's secret is negligible.

A zero-knowledge protocol for proving knowledge of a secret already exists and is based on the discrete log problem. In the discrete problem, the inputs are integers g, y, and p where p is prime and the output is an integer x such that $g^x \equiv y \mod p$. It is commonly believed that the discrete log problem is hard, meaning that it generally cannot be solved in polynomial time. This means that if x is some secret, then a user can broadcast the value $g^x \mod p$ with confidence that no one will be able to determine x. A protocol for proving knowledge of x without revealing x is provided in Figure 1 [4]. Note that all operations are done mod p. If the prover knows the secret, then the verifier is guaranteed to accept:

$$ah^b = (g^r)(g^x)^b = g^{r+xb} = g^c$$

However, if the prover does not know the secret, then it would have to produce an alternate value for c such that $g^c \equiv g^{r+bx} \mod p$. The probability of being able to do this is negligible. The protocol is zero-knowledge because, by the discrete log assumption, an adversary can gain no information about r from the value g^r, and without r it is impossible to determine x from the value $r + xb$.

This protocol can certainly be used to authenticate users when the secret x is some fixed value. However, in the case of keystroke dynamics, the secret may have variations. That is, the timings will be slightly different on each trial [9]. In a regular protocol this is not a problem; with access to the timing data itself, it is easy to calculate the distance between the actual timings and expected ones. With zero-knowledge, this is more difficult. Instead of calculating a distance, the random noise in the sample must be removed in order to obtain a unique value.

2.3 Fuzzy Extractors

Fuzzy extractors are used to extract a uniformly random string S from a noisy input $\mathfrak{B}$, also known as a biometric template. A similar input $\tilde{\mathfrak{B}}$ will produce the exact same string S [2]. In the case of keystroke based authentication, S is a user's unique signature which can be extracted from a noisy input. It is then possible to prove knowledge of the secret value S using a zero-knowledge proof. The following construction for a fuzzy extractor is due to Álvarez, Encinas, Hernández, and Sanchez-Avila.

The enrollment phase for the fuzzy extractor is shown in Figure 2 [2]. Given a biometric template $\mathfrak{B}$ and a secret S, the enrollment function produces additional data H and Δ that are used to recover S. The values of H and Δ can safely be made public without compromising the security of the secret S. First, the secret is represented as an integer in base b, ie $S = s_0 s_1 \ldots s_d$ where $d = \log_b S$. A polynomial $p(x)$ is now derived whose coefficients are the base b digits of S:

$$p(x) = s_0 + s_1 x + s_2 x^2 + \cdots + s_d x^d$$

Next, n random integers $x_1, x_2, \ldots, x_n$ are generated and $p(x)$ is evaluated at each of these points to obtain $y_i = p(x_i)$. The value n controls how fuzzy an accepting template can be. For technical reasons, n must be much greater than d. The n coordinate pairs are then concatenated together as $x_i||y_i$ and encoded using a Reed-Solomon code to form the set $C = \{c_1, c_2, \ldots, c_n\}$. A Reed-Solomon code is an encoding of a message using extra bits such that if the message is corrupted, it can still be recovered accurately [10]. A hash function $\mathfrak{h}$ is applied to all of the Reed-Solomon encoded values to obtain $H = \{\mathfrak{h}(c_1), \mathfrak{h}(c_2), \ldots, \mathfrak{h}(c_n)\}$. The input template $\mathfrak{B}$ is divided into n parts $b_1||b_2||\ldots||b_n$ and each entry in $\Delta = \{\delta_1, \delta_2, \ldots, \delta_n\}$ is obtained as $\delta_i = c_i - b_i$. The values H, Δ, b, d, and $\mathfrak{h}$ are then stored to be used for extraction [2].

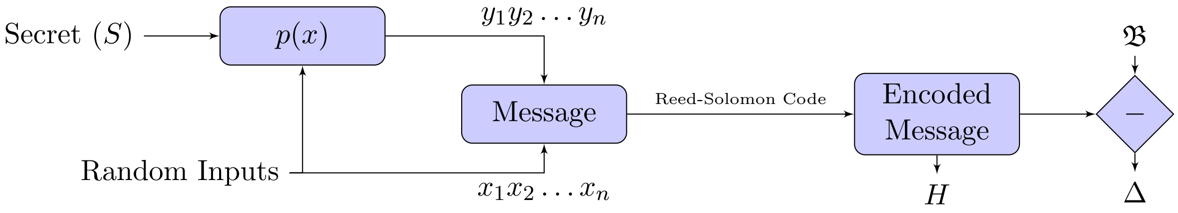

Figure 2: Enrollment phase for a Fuzzy Extractor

The key feature of a fuzzy extractor is the ability to extract the secret S given a noisy, but genuine template $\tilde{\mathfrak{B}}$. The extraction process is detailed in Figure 3. First, the query template $\tilde{\mathfrak{B}}$ is partitioned into n parts $\tilde{b}_1||\tilde{b}_2||\ldots||\tilde{b}_n$ and each part is added with the corresponding element of $\Delta = \{\delta_1, \delta_2, \ldots, \delta_n\}$ to get $\tilde{C} = \{\tilde{c}_1, \tilde{c}_2, \ldots, \tilde{c}_n\}$. The values of $\tilde{C}$ should be similar to those of C, however they will be slightly different due to variations in the noisy template $\mathfrak{B}$. Note that the values of C are not known, but that hashed values of C are, so the entries of $\tilde{C}$ are also hashed to obatin $\tilde{H} = \mathfrak{h}(\tilde{C})$ and the entries of $\tilde{H}$ are compared to those of $H = \mathfrak{h}(C)$. If at least $d+1$ of the entries in H are equal to the corresponding entries in $\tilde{H}$ then $\tilde{\mathfrak{B}}$ is precise enough to be decoded, otherwise the system immediately rejects.

The next step is to obtain the (x, y) pairs using the Reed-Solomon code. Since there were at least $d+1$ matching hashes, at least $d+1$ coordinate pairs will be decoded. The next step is to use Lagrange Interpolation to recover the polynomial $p(x)$. Interpolation of a polynomial of degree d requires at least $d+1$ points, thus it is crucial that at least $d+1$ points are decoded. The parameter n defines the amount of fuzziness that is allowed. Since there are a fixed number of points *needed* to interpolate the polynomial, increasing n decreases the fraction of the partitions of $\mathfrak{B}$ that must be accurate. Once $p(x)$ is known, it is easy to determine the secret S. Each base b digit of S is simply a coefficient of $p(x)$ [2].

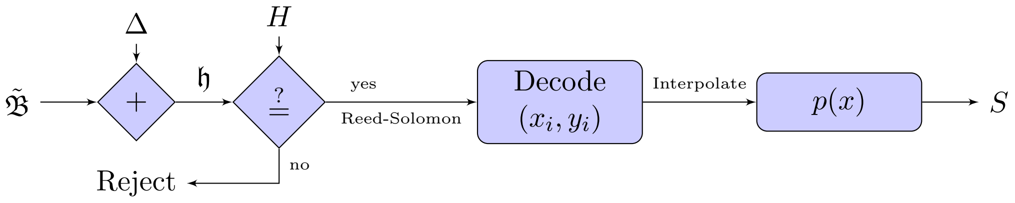

Figure 3: Extraction phase for a Fuzzy Extractor

Álvarez, Encinas, Hernández, and Sanchez-Avila showed that the above method of fuzzy extraction has a genuine acceptance rate of 89% and a false acceptance rate of around 1% on iris scan authentication [2]. While a different biometric such as keystroke dynamics would undoubtably produce different error rates, prior work shows that this fuzzy extractor has potential to accurately distinguish between people. The genuine acceptant rate of 89% is fairly good; it implies that the expected number of trials people will need to perform in order to authenticate themselves is $\frac{100}{89} \approx 1.12$. In the context of the keystroke dynamics protocol, this is convenient enough to satisfy even a lazy user. In most instances, the user will only have to type his name once, but occasionally a second trial may be required.

Prior work shows that authentication via keystroke dynamics is feasible [9]. However, storing information about keystroke timings in a database is dangerous, therefore a zero-knowledge keystroke authentication protocol is very attractive. A protocol already exists to prove knowledge of a secret without giving away any information [4]. This protocol relies on the secret being an exact value, not a noisy sample such as a keystroke timing vector. This problem is solvable using a fuzzy extractor to retrieve an exact secret from a noisy input [2]. Composing these ideas, a zero-knowledge keystroke dynamics authentication protocol can be created.

3 Methodology and Techniques

We wish to examine the feasibility of keystroke based authentication from both a theoretical and empirical perspective. Two protocols are presented. The first protocol is extremely insecure, but acts as a proof of concept and comparison baseline in terms of accuracy and security. This protocol has been implemented as a simple web app. The second protocol is a zero-knowledge version of the first one that relies on an extractor to obtain a secret from a noisy input. The secret is never stored or communicated across a network; instead knowledge of the secret is proven using a zero-knowledge protocol. This protocol has been implemented using a rounding extractor and the feasibility of using the fuzzy extractor that is described in Section 2.3 is also examined.

When discussing these protocols, the words *prover* and *client* are used almost interchangeably as are the words *verifier* and *server*. The client is an implementation of a prover and the server is an implementation of a verifier. The client is naturally more 'powerful' than the server as the client knows the secret keystroke timings (or can at least extract them from a noisy input). Both the regular and the zero-knowledge versions of the web app have two capabilities: account creation and login. The account creation phase requires users to type their names in a text box. The keystroke timings are recorded and communicated to the server. The server then stores some data about the keystrokes in a database. When users try to log in, they also simply types their names in a text box. The names that are typed are looked up in the database and then their keystroke timings are compared against the ones that are stored. Note that the word *compare* is used loosely here; in the zero-knowledge protocol, no direct comparison occurs per se. Instead, the difference between the two signatures is measured by some specialized technique. The user experience is identical (aside from login accuracy) in the regular and zero-knowledge versions, although the implementation is quite different.

3.1 The Basic Protocol

The basic protocol authenticates users by directly comparing the given keystroke vector to the expected keystroke vector. This protocol is not designed with security in mind; any adversary that can see the information exchange between the prover and the verifier will be able to directly discern the keystroke vectors. More concretely, anyone eavesdropping on the communications over the network can see these values communicated in the clear. This protocol is not meant to be used in a real implementation of the system, but instead it is meant to be a point of comparison for the zero-knowledge version.

Prover		**Verifier**
User types name (A) in web client. Keystroke vector (V) is recorded	$\xrightarrow{(A,V)}$	(A, V) is stored in the database

Figure 4: Account creation in the basic protocol

Let $V = v_1 v_2 \dots v_n$ be the real keystroke vector for user A and $\tilde{V} = \tilde{v}_1 \tilde{v}_2 \dots \tilde{v}_n$ be the keystroke vector associated with a login attempt by someone claiming to be A. Account creation in the basic protocol is very simple. First, the prover sends a pair (A, V) containing the user's name and keystroke timings to the verifier. The verifier then stores this information in its knowledge bank. This protocol is outlined in Figure 4. Authentication (as seen in Figure 5) is then performed by applying a comparison function c to the expected keystroke vector for user A and the login vector. The comparison function that is used in the actual implementation is defined as follows:

$$c(V, \tilde{V}) = \begin{cases} 1, & d(V, \tilde{V}) \leq \epsilon \\ 0, & \text{otherwise} \end{cases} \qquad d(V, \tilde{V}) = \sqrt{\tfrac{1}{n} \textstyle\sum_{i=1}^{n} (v_i - \tilde{v}_i)^2}$$

Above, ϵ is a constant error tolerance which is carefully chosen considering the tradeoffs between security and ease of login. This tradeoff will be examined in Section 4.

Prover		**Verifier**
User types name (A) in web client. Keystroke vector ($\tilde{V}$) is recorded	$\xrightarrow{(A,\tilde{V})}$	Lookup key A in the database to get V
Report result	$\xleftarrow{c(V,\tilde{V})}$	Send decision

Figure 5: Authentication in the basic protocol

Disregarding the fact that the keystroke vector V is immediately discernable from eavesdropping on the account creation process, this protocol will be the baseline for evaluating the performance of the zero-knowledge version.

3.2 The Zero-Knowledge Protocol

The zero-knowledge protocol builds on top of the basic protocol with the added constraint that no meaningful information about the keystroke vector is ever transferred across the network. This also means that the representation of the keystroke vector that is stored in the database does not leak any information about the keystroke vector itself. If the database were completely compromised, there would thus be no breach in security; the information in the database can be used to verify a user's identity and nothing more.

The login process is detailed in Figure 6. In this figure, f is the extractor being used. The protocol itself is abstract in that f could represent any extractor; for example, the fuzzy extractor described in Section 2.3. The protocol is based on the zero-knowledge proof presented in Section 2.2. The client first chooses a generator g and a prime p, and then sends the value $g^{f(V)} \mod p$ to the server as the user's comparison point. Recall from Section 2.2 that it is safe to transmit this value because of the assumption that taking the discrete log is hard.

The authentication protocol is shown in Figure 7. Since the client in this implementation is a web page that may be accessed from any computer, information about a particular user cannot be

Prover		**Verifier**
Choose some generator g	$\xrightarrow{g}$	Remember g
User types name (A) in web client. Keystroke vector (V) is recorded. Let $h = g^{f(V)}$, where $f(V)$ is the fuzzy extraction of V.	$\xrightarrow{h}$	(A, g, h) is stored in the database

Figure 6: Account creation in the zero-knowledge protocol

stored and thus the value of g must be provided by the server after discovering the user's name. Aside from this caveat, the protocol is identical to the zero-knowledge proof presented in Section 2.2.

Prover		**Verifier**
User types name (A) in web client. Keystroke vector ($\tilde{V}$) is recorded	$\xrightarrow{A}$	
	$\xleftarrow{g}$	Lookup key A in the database to get g and h
Choose $r \in_R \mathbb{Z}_g$, let $a = g^r$	$\xrightarrow{a}$	
	$\xleftarrow{b}$	Choose $b \in_R \mathbb{Z}_g$
Let $c = r + f(\tilde{V}) \cdot b$	$\xrightarrow{c}$	
Report decision	$\xleftarrow{ah^b \overset{?}{=} g^c}$	

Figure 7: Authentication in the zero-knowledge protocol

In the implementation, each keystroke is hashed in a locally sensitive way. First, the keystroke timing is rounded within some small error tolerance and then a hash function is applied to the rounded value. The purpose of the hash function is to scatter the data points in the case that a poor generator is used. If g^x is less than p, then it is easy to recover x by taking the base g log of the exponentiated value. A rounding parameter $0 < \delta \leq 1$ must be chosen such that it is relatively easy for a person to produce each keystroke while preserving a δ fraction of the information stored in each keystroke timing, but it is hard for anyone else to achieve this. The extractor function f is defined as follows:

$$f(V) = \{g(v_1), g(v_2), \ldots, g(v_n)\} \quad \text{where} \quad g(v_i) = h(\lceil v_i \cdot \delta \rceil)$$

The function h is the hash function that is applied to the rounded result. After applying $f(V)$, each secret in the resulting set is proven to the server using the zero-knowledge protocol. It is difficult to produce every single keystroke within the δ tolerance, therefore only a γ fraction of the secrets must be successfully proven to the server.

4 Results

The performance of the system is evaluated on two main metrics: precision (how easily people can repeat their own typing pattern), and protection (how easily an imposter can forge the typing pattern). The performance of the application is tested with four participants, all with varying typing speeds, preferred hand layouts, and dominant hands. The weighted difference in average typing speed at which the target user was typing at is recorded as ϵ' and compared to the threshold level of $\epsilon = 40$ in the zero-knowledge case and $\epsilon = 50$ for the basic protocol. The consistency that the target user could replicate that result is measured as a percentage. The three other participants then attempt to forge the typing signature, having not seen the target user's test trials. Identical words were selected as the control for both protocols in order to measure the variance between the two success rates.

Table 1 contains the results from this experiment en the Basic Protocol implementation that is described in Section 3.1. Trials were conducted between 20 and 40 times with varying word lengths. The values of the timings were compared directly, allowing a total error of $\epsilon = 50$. The value ϵ' in Table 1 indicates the average error value that the target user scored. Some trends arise from these results. As word length increases, typing signatures become more unique and therefore adversary forgery success rate drops significantly. However, this also affects target users' ability to recreate their typing signature consistently. There appears to be a tradeoff between security and convenience which is due to choosing a constant ϵ. Longer keystroke vectors amortize the allowed error over a larger number of keystrokes, therefore each keystroke must be more exact.

Length of Name	ϵ'	Success by User (%)	Success by Imposter (%)
3	14	100	100
5	16	87	65
8	47	85	25
11	43	87	12
13	47	65	0

Table 1: Basic Protocol: Keystroke variation by length of name.

Table 2 documents repeated test trials with $\delta = 0.05$. Trials were conducted between 20 and 40 times with varying word lengths. The trend of increasing successful authentications with increasing word length is apparent. This is most likely attributed to the fact that typing patterns are more distinct with longer pieces of text. Conversely, imposters are also more likely to successfully impersonate the target user with a longer text, since the scheme does not measure the accumulated difference in time. In other words, an imposter's speed could be very far from the expected timings on many of the strokes, but close on others. In this case, the imposter may still be authenticated because the protocol only cares *how many* matches occurred, not *how much* the totals differ. With shorter words, the implementation is very unforgiving. Since there are fewer keystrokes to average out the mistakes, timing is extremely sensitive. In practice, this is less of a concern since virtually no one has a combined first name and last name of less than three characters. There exists a non-trivial difference between the success rates of target users and their impersonators. This affirms the fact that keystroke dynamics are variable enough and have sufficient entropy to be used as authentication tools. Barring anomalies with short phrases, the difference in patterns becomes apparent. In particular, the increase between a phrase of 11 characters and one of 13 shows that longer strings are not much harder for a user to type, but decrease the success rate of an adversary.

Besides the sampling bias inherent in a small sample size, the frequency in which the trials were conducted would also be different in practice. Trials were conducted consecutively, with participants intent on replicating their own keystrokes or mimicking others. Therefore, the results may be partially due to muscle memory, instead of the unique typing signature under any conditions. The strong performance of adversaries indicates that the parameters should be adjusted to to be more exclusive, especially with longer user names.

In fact, an ideal protocol would combine the information-theoretic security of the zero-knowledge protocol with the protection against forgers of the basic protocol. The false login rate of the current

Length of Name	$\epsilon^{'}$	Success by User (%)	Success by Imposter (%)
3	112	4	0
5	59	10	5
8	47	85	23
11	34	87	43
13	32	87	35

Table 2: Zero-Knowledge Protocol: Keystroke variation by length of name.

implementation of the zero-knowledge protocol is much too high. One way to achieve better results in a zero-knowledge setting may be to use the fuzzy extractor construction presented in Section 2.3. The results of this extractor (89% genuine accept rate and 1% false accept rate [2]) show promise in preserving more of the information about the keystroke timings than the rounding extractor that is implemented. The direct comparison results in Table 1 show that achieving a relatively high genuine acceptance rate is possible while keeping the false accept rate low in a keystroke dynamics authentication scheme. This result is reaffirmed by the experiments of Araújo, Sucupira, Lizarraga, Ling, and Yabu-Uti [3]. Since the purpose of a fuzzy extractor is to separate data from noise, the actual information stored in a keystroke vector should be able to be extracted almost in its entirety [2]. There is no reason to believe that the direct comparator can differentiate data from noise better than a well-designed fuzzy extractor. In fact, in 2013, Fuller, Meng, and Reyzin created a fuzzy extractor construction that has no entropy loss [6]. That is, the amount of entropy in the secret is equal to the amount of entropy in the input template. This result may not be desirable in all cases as some of the noise needs to be filtered out, but it shows that information loss is not inherent in the zero-knowledge protocol.

5 Future Work

It is evident that the implementation of zero-knowledge keystroke authentication, though viable as a proof of concept, is nowhere near robust enough to be used in practice. Currently, for longer names, the prototype has bounded probability of success; it correctly verifies the correct user a significant portion of time (barring human induced errors such as typing anomalies), but also verifies an imposter over 40% of the time. This can be fixed by adjusting δ, the fault tolerance per keystroke. The idea is to find the optimal value of δ and ϵ such that the real users can be easily validated but an imposter cannot be validated. The next logical step would be to adjust the implementation to use a different fuzzy extractor such as the one described in Section 2.3 [3]. If sufficient performance is not achieved, then a fuzzy extractor with no entropy loss [6] could be used.

Aside from accuracy concerns, the protocol presented is very keyboard dependent. This means that this method of verification would be impossible on mobile devices, since people generally use their thumbs to type on smaller keyboards and the keystroke timing on these keyboards is likely to be very different. All of the trials used a QWERTY keyboard, the layout of which is significantly different than that of the French AZERTY keyboard or Dvorak. Although not yet tested, the protocol presented would work just as well on different keyboard layouts assuming that users can type with equal consistency on these keyboards. A problem arises when the keyboard layout is switched. For example, if American users travels to France, they may be unable to access their account due to the different keyboard layout.

The protocol should be tested on many more words of varying lexicographical distances. The demonstration used only five words which is subject to extreme bias, since characters in the words chosen may have been clustered in a way that changed the difficulty of typing that one particular word. The currently observed trends may differ when a larger sample size of words and participants is used.

Though the implementation is not yet accurate enough, it provides a solid proof of concept for the potential of keystroke authentication. The protocol could be improved by augmenting the keystroke vector with additional characteristics of a person's typing pattern such as the actuation force and stroke duration. Because of the nature of our zero-knowledge scheme, the users also do not have to worry about any security breaches in the communication channels or in the database. Just like a written signature, the typing signature will be unique and secure.

References

[1] Two factor auth (2fa), December 2014.

[2] Fernando Hernandez Álvarez, Luis Hernández Encinas, and Carmen Sanchez-Avila. Biometric fuzzy extractor scheme for iris templates. In Hamid R. Arabnia and Kevin Daimi, editors, *Security and Management*, pages 563–569. CSREA Press, 2009.

[3] L.C.F. Araújo, L.H.R. Sucupira, Jr., M.G. Lizarraga, L.L. Ling, and J.B.T. Yabu-Uti. User authentication through typing biometrics features. *Trans. Sig. Proc.*, 53(2):851–855, February 2005.

[4] David Chaum, Jan-Hendrik Evertse, Jeroen van de Graaf, and Ren Peralta. Demonstrating possession of a discrete logarithm without revealing it. In *Advances in Cryptology - CRYPTO '86, Santa Barbara, California, USA, 1986, Proceedings*, volume 263 of *Lecture Notes in Computer Science*, pages 200–212. Springer, 1986.

[5] Dinei Florencio and Cormac Herley. A large-scale study of web password habits. In *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, pages 657–666, New York, NY, USA, 2007. ACM.

[6] Benjamin Fuller, Xianrui Meng, and Leonid Reyzin. Computational fuzzy extractors. Cryptology ePrint Archive, Report 2013/416, 2013.

[7] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity, and a methodology of cryptographic protocol design. In *Proceedings of the 27th FOCS*, pages 174–187, 1986.

[8] Fabian Monrose, Michael K. Reiter, and Susanne Wetzel. Password hardening based on keystroke dynamics. In *Proceedings of the 6th ACM Conference on Computer and Communications Security*, CCS '99, pages 73–82, New York, NY, USA, 1999. ACM.

[9] Fabian Monrose and Aviel D. Rubin. Authentication via keystroke dynamics. In *ACM Conference on Computer and Communications Security'97*, pages 48–56, 1997.

[10] Irving Reed and Golomb Solomon. Polynomial codes over certain finite fields. *Journal of the Society of Industrial and Applied Mathematics*, 8(2):300–304, 06/1960 1960.

[11] Deian Stefan and Danfeng Yao. Keystroke-dynamics authentication against synthetic forgeries. In *CollaborateCom*, pages 1–8. IEEE, 2010.