Secure Privacy-Preserving Protocols for Outsourcing Continuous Authentication of Smartphone Users with Touch Data

Sathya Govindarajan Paolo Gasti Kiran S. Balagani
New York Institute of Technology
{sgovin03, pgasti, kbalagan}@nyit.edu

Abstract

We introduce new secure privacy-preserving protocols for outsourcing continuous authentication of smartphone users. Our protocols allow a smartphone to privately perform continuous and unobtrusive authentication using touch behaviors. Through our protocols, the smartphone does not need to disclose touch information to the authentication server. Further, neither the server nor the smartphone have access to the content of the user's template.

We present formal proofs to substantiate security claims on our protocols. We then perform experiments on publicly available touch data, collected from forty-one users. Our experiments on a commodity Android smartphone show that our protocols incur an overhead between 263ms and 2.1s.

1. Introduction

Continuously authenticating users of commodity smartphones has become a significant priority because: (1) there has been a tremendous surge in the amount of personal data and sensitive information stored or accessed on today's smartphones, and (2) due to their small size and high mobility, smartphones are constantly exposed to unauthorized access via theft, loss, and coercion. Login-time *pins*, and *textual* and *graphical* passwords are by far the most popular mechanisms for authenticating smartphone users. However, these mechanisms suffer from at least two drawbacks:[1] (1) they are *static*, i.e., they authenticate the user once – at the beginning of a session – and do not offer any protection against unauthorized access post login; and (2) passwords and pins require user's attention for their entry and therefore are not suitable for continuous authentication.

Implicit or "active" authentication is an emerging area in smartphone domain, which aims to continuously authenticate users without interrupting them. Authentication is performed by harnessing the users' natural interactions with the smartphone. Lately, behavioral modalities that have been used for active authentication include on-screen touch and gestures [13, 23], hand or smartphone movement and orientation [12], geolocation patterns and combinations of these [30].

Among the aforementioned modalities for active authentication of smartphone users, touch has been gaining popularity because: (1) there is growing evidence [12, 13, 23] that touch patterns can be used to create reliable signatures for user authentication, and (2) most interactions with smartphone applications happen as taps, touch strokes and gestures, so these patterns are easily available for enrollment and subsequent authentication.

As new and reliable active authentication modalities become available, more businesses have started to integrate these behavior-based technologies into their multi-factor online-user authentication solutions [2]. Because deploying, maintaining, and updating these technologies requires specialized expertise and infrastructure, they are often *outsourced* to companies that provide Authentication as a Service and Identity Assurance as a Service (e.g., Admit One Security [1] and BehavioSec [3]). These companies focus on building the software, maintaining the infrastructure, and updating user templates for their customers.

Although outsourcing continuous authentication may improve security, it also raises privacy concerns because behavioral information is disclosed to a third party. In the case of outsourced touch-based authentication, this could lead an authentication server to learn what keys have been pressed on a virtual keyboard using tap locations, or to infer what type of application (e.g., a map or a browser) the user is currently interacting with via gesture information. To the best of our knowledge, no prior work has addressed the problem of securely outsourcing touch-based continuous authentication. In this paper, we address this problem.

Our Contributions. We design, implement and evaluate a framework for securely and privately outsourcing continuous authentication based on touch data to a (possibly untrusted) server. Our framework consists of two efficient

[1]There is also growing evidence that users tend to choose simple text and graphical passwords, making them relatively easy to guess [4].

"custom" privacy-preserving protocols which allow a server to *privately* authenticate a smartphone user via scaled Euclidean and scaled Manhattan verifiers. After executing our protocols, the server learns the distance between the template and the user's sample, and no additional information. Furthermore, our protocols do not disclose the content of the user's template to either the server or the smartphone.

We provide formal proofs of security in the semi-honest model. Next, via experiments on a prototype implementation running on an Android smartphone, we show that our protocols are practical and that their overhead is small.

Of independent interest, to the best of our knowledge this work is the first to introduce efficient privacy-preserving computation of *exact* Manhattan distance.

Organization. We review related literature in Section 2. In Section 3, we present the dataset, verification experiments, and results. We introduce our protocols in Section 4 and evaluate them in Section 5. We conclude in Section 6.

2. Related Work

Continuous Authentication with Touch Patterns. Because touch-based continuous authentication of smartphone users is an emerging field, there are a limited number of studies on this topic. Below, we review recent work.

Li et al. [23] selected 8 gesture and tap features out of 52 gesture and 3 tap features to continuously authenticate users. A total of 75 users participated in their experiments. Out of them, 25 were target and 47 were non-target users. During verification, a support vector machine training model was created for each target user using positive examples (that belong to the same target user) and negative examples (that belong to the remaining target and non-target users). Four-hundred positive and negative examples were used for training each model and another 400 for testing. The authors ensured that users used for generating negative training examples were not used in testing. The authors reported close to 4% false accept and 4% impostor pass rates with sliding gestures.

Frank et al. [13] used 28 touch based behavioral features, with a total of 41 users participated in the experiments. The authors used support vector machine and kNN training models for each user, using positive and negative examples. They reported between 0 and 4 percent equal error rates under inter-week (enrollment and testing sessions separated by a week), inter-session (enrollment and testing data belong to different sessions), and intra-session (enrollment and and testing belong to the same session) scenarios. No training data was used during testing. The authors achieved between 11 and 43 seconds verification delay, depending on the scenario.

Feng et al. [12] used 53 touch and gesture features. In addition, they used a digital glove to capture 36 triaxial angular rate features when users performed touch activity. A total of 40 subjects participated in the study, out of which digital glove data was collected for 11 subjects. For each user, a training model was developed using random forest, J48 decision tree, and the Bayes network classifier. From [12], it is unclear how many positive/negative examples were used or what percentage of the data were allocated for training and testing. With Bayesian classifier, the authors reported 2.15% FAR and 1.63% FRR when the data from the digital glove was used with touch data and 11.96% FAR of and 8.53% FRR without the digital glove data.

Privacy-Preserving Protocols. Starting from the seminal work on garbled circuit evaluation [32, 15], it has been shown that any function can be securely evaluated by representing it as a boolean circuit. Similar results exist for secure evaluation of any function using secret sharing techniques, e.g., [28], or homomorphic encryption, e.g., [7].

In recent years, a number of tools have been developed for automatically creating a secure protocol from its function description written in a high-level language. Examples include Fairplay [25], VIFF [10] and TASTY [16]. However, "custom" optimized protocols for specific applications are often more efficient than such general techniques.

A number of recent publications address the problem of privacy-preserving biometric authentication and identification. Secure face recognition was first treated by Erkin et al. [11]. In this paper the authors designed a privacy-preserving face recognition protocol based on Eigenfaces. Sadeghi et al. [29] subsequently improved the performance of the protocol. More recently, Osadchy et al. [26] designed a new face recognition algorithm together with its privacy-preserving realization called SCiFI. The design targeted to simultaneously address robustness and efficiency when used for secure computation.

Recent work of Blanton et al. [6] focused on privacy-preserving iris and fingerprint matching. The authors rely on a hybrid approach based on garbled circuits and homomorphic encryption for optimal performance. Barni et al. [5] presented a privacy-preserving protocol for fingerprint identification using FingerCodes [17], which is not as discriminative as techniques based on location of minutiae points, but is particularly suited for efficient realization in the privacy-preserving framework.

Techniques based on fuzzy commitments (e.g., [18, 31, 19]) are commonly used to provide template protection and to implement access control on encrypted documents. However, such techniques require biometric comparisons to be performed in a feature space different from that of the original biometrics, possibly increasing ERR [22]. In contrast, our protocols do not affect EER of the underlying biometric modality, since the comparison between the user sample and the template is functionally identical to the same comparison in the unencrypted domain.

3. Continuous Authentication with Touch Data

We used a publicly available touch dataset,[2] which was originally introduced in [13]. Here, we briefly discuss the dataset for completeness. (A detailed description can be found in [13].) The dataset contains 30 behavioral touch features from 41 participants, who were asked to perform two tasks: reading text and comparing pictures, on Android based commodity smartphones. A recording tool captured touch data when the participants were performing the tasks. The dataset was collected in two phases: in the first phase, the participants were asked to read different documents on their smartphones. In the second phase, the participants were presented with two similar images and were asked to spot the differences. We used the dataset that was available in ARFF format. The dataset contains a total of 21,158 feature vectors. Each vector has 34 features that include 30 touch features, device ID, user ID, document ID, and device orientation. On an average, each user has 516.04 (227.18 standard deviation) feature vectors.

We split the feature vectors belonging to each user into two sets: (1) training, and (2) testing. During training, we used first 90 percent of a user's vectors to build the template. We used the remaining to generate genuine test attempts. For each user, we used the feature vectors belonging to the remaining 40 users to generate zero-effort impostor attempts. In our experiments, we performed biometric *verification* by matching a user's template (created from the user's own samples) with test attempts. We did not use impostor samples to create a user's training model (as done in recent touch-based authentication studies [13, 23]).

3.1. Feature Selection

Traditionally, feature selection has been used to define a subset of most informative features to improve the performance of a prediction task. However, in the dataset we used, some features had abnormally high variance (plausibly because the features were extracted during different task contexts such as reading different documents and switching between pictures). We used feature selection to identify *stable* features, i.e., the features with least amount of deviation from their median values. We used unsupervised feature selection and ranked each feature using *median absolute deviation from median* (MAD), given by:

$$\mathsf{MAD}(\mathsf{F}) = \mathsf{Median}\{|x_1 - \hat{x}|, |x_2 - \hat{x}|, \cdots, |x_m - \hat{x}|\},$$

where x_i represents an instance of a feature F, m represents the number of instances of F, and $\hat{x}$ represents the median of F. We performed feature selection on the training data. Our feature selection process ranks individual features based on the increasing order of their MAD values (i.e., lower ranks correspond to lower MAD values). The lower the MAD values, the higher the stability of that feature. In the same fashion, for comparison purposes, we also used the supervised feature ranking method based on normalized mutual information that was given in [13].

[2]http://www.mariofrank.net/touchalytics/index.html

Remark. There are two types of feature selection methods: (1) supervised, which use class information, and (2) unsupervised, which do not use class information. Though supervised methods are known to perform better than unsupervised (see [24]), using them in a verification setting requires running the feature selection procedure each time a user is added into or deleted from the database. This clearly affects scalability. For this reason, we opted for unsupervised feature selection.

3.2. Data Cleaning and Preprocessing

Like any behavioral data, touch data is not exempt from extreme values (outliers). Outliers may arise due to several factors including noisy sensors, users' movement and physical activity, users' state of mind (e.g., cognitive stress), and users' adaptation to situational impairments (e.g., user carrying a smartphone with one hand and a heavy object with the other). Outliers can potentially distort users' templates, especially if the templates maintain summary statistics sensitive to outliers (e.g., mean). For each feature of a user, we use values between 3 and 97 percentiles for creating the template and ignore the rest. We chose the percentile thresholds to achieve a balance between eliminating outliers and at the same time retaining enough values to create a template representative of the user's touch behavior. Additionally, we performed Zero Component Analysis (ZCA) whitening on training and testing data of each user to decorrelate features.

3.3. Discretization

Because our privacy-preserving protocols are designed to work on positive integer values, we mapped each real-valued feature to $[0, 2^e - 1]$ with equal-width bins. We used the following formula for discretization:

$$\mathsf{discretize}_{e,\mathsf{F}}(x_i) = \left\lfloor \frac{(2^e - 1) \cdot |x_i - \mathsf{min}_\mathsf{F}|}{|\mathsf{min}_\mathsf{F} - \mathsf{max}_\mathsf{F}|} \right\rfloor,$$

where F is the feature being discretized, x_i is an instance in F, min_F is the minimum value of F, and max_F is the maximum value of F. The e parameter controls the number of cells a feature is discretized into. Therefore, higher the e parameter value, the lower the potential loss of information due to discretization. We experimented with e values 8, 10, and 16. We opted for an unsupervised discretization method for the same reason that we opted for unsupervised feature selection in Section 3.1.

3.4. Verification Experiments and Results

Verifiers. We used two verifiers: (1) scaled Euclidean and (2) scaled Manhattan, because these verifiers have been previously shown to perform well for behavioral authentication (see [21]). Below, we briefly describe the verifiers.

Let $Y = \{\bar{y}_1, \ldots, \bar{y}_n\}$ be the *mean* vector computed from the feature vectors in the training set of a user. Let $X = \{x_1, \ldots, x_n\}$ be a test vector. The *scaled Euclidean* verifier is defined as:

$$\mathsf{D}_E(X,Y) = \frac{1}{n}\sqrt{\sum_{i=1}^{n}\frac{(x_i-\bar{y}_i)^2}{\sigma_i}}, \text{ and}$$

the *scaled Manhattan* verifier is defined as:

$$\mathsf{D}_M(X,Y) = \frac{1}{n}\sum_{i=1}^{n}\frac{|x_i-\bar{y}_i|}{\sigma_i},$$

where n is the number of features, $\bar{y}_i$ is the mean of the i-th feature in the template, x_i is the corresponding feature value in the test vector, and σ_i is the standard deviation of the i-th feature.

Results. Figures 1 and 2 depict the percentage equal error rates (%EERs) of scaled Euclidean and Manhattan verifiers. Solid plots show EERs of original (real-valued) features and dotted plots show the %EERs of corresponding discretized features when e is set to 8, 10, and 16. The x-axes in figures 1 and 2 show the highest rank of the feature subset selected with normalized mutual information (figures 1a and 1b) or MAD (figures 2a and 2b) criteria. That is, 1 on the x-axis represents the subset containing the 1st ranked feature (the feature with highest normalized mutual information or lowest MAD value), 2 represents the subset containing 1st and 2nd ranked features (features with the highest and the second highest normalized mutual information or lowest and the second lowest MAD), and so on.

Performance of Verifiers. With normalized mutual information based feature subset selection performed on original features (that were *not* discretized), the scaled Manhattan verifier yielded lowest EER of 22.50% (with rank 6 subset) and the highest EER of 38.28%(with rank 1 subset). In comparison, the scaled Euclidean verifier yielded between 21.27% (with rank 6 subset) and 38.28% (with rank 1 subset) EERs. With MAD feature selection performed on original features (which were *not* discretized), the scaled Manhattan verifier yielded between 27.04% (with rank 26 subset) and 45.77% (with rank 1 subset) EERs. In comparison, the scaled Euclidean verifier yielded between 26.17% (with rank 4 subset) and 45.77% (with rank 1 subset) EERs. Irrespective of the feature subset selection method we used, the scaled Euclidean performed slightly better than the scaled Manhattan for most of the m-ranked feature subsets.

Impact of Discretization. From figures 1 and 2, we observe that discretized features have higher %EERs compared to the original real-valued features between 4 and 12 feature subset sizes. Outside this size range, the discretized features have comparable %EERs to that of the original features. Figure 2 shows that the MAD feature subsets outside of sizes 4 through 9 have their %EERs of scaled versions relatively close to their original unscaled counterparts, with deviations less than 1% in most cases. A similar trend can be observed for normalized feature subsets (see figures 1), where the %EERs of feature subsets sizes beyond 10 have higher values compared to their unscaled counterparts, with deviations between 1% and 2%. Also, it is interesting to note that there are larger deviations of %EERs for 4, 5 and 6 feature subsets for both mutual information and MAD based features and the scaled feature subsets of MAD better approximate their unscaled counterparts.

4. Privacy-Preserving Protocols

In this section we review the security model and the cryptographic tools used in our constructions. We then introduce our protocols.

4.1. Cryptographic Preliminaries

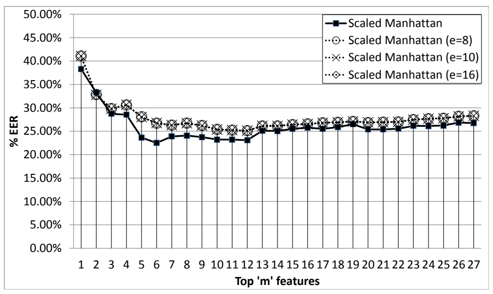
Security model. We use the term *adversary* to refer to insiders, i.e., protocol participants. Outside adversaries are not considered, since their actions can be mitigated via standard network security techniques.

Our protocols are secure in the presence of semi-honest (also known as honest-but-curious or passive) participants. In this model, while participants follow prescribed protocol behavior, they might try to learn additional information beyond that obtained during normal protocol execution. Formally [14]:

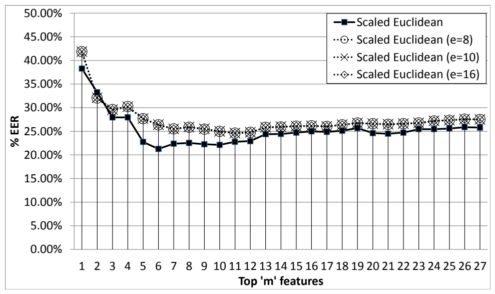
Definition 1 *Let P_1 and P_2 participate in protocol π that computes function $f(\mathsf{in}_1, \mathsf{in}_2) = (\mathsf{out}_1, \mathsf{out}_2)$, where in_i and out_i denote P_i's input and output, respectively. Let $\mathrm{VIEW}_\pi(P_i)$ denote the view of participant P_i during the execution of protocol π. More precisely, P_i's view is formed by its input, internal random coin tosses r_i, and messages $m_1, \ldots, m_t$ passed between the parties during protocol execution: $\mathrm{VIEW}_\pi(P_i) = (\mathsf{in}_i, r_i, m_1, \ldots, m_t)$.*

We say that protocol π is secure against semi-honest adversaries if for each party P_i there exists a probabilistic polynomial time simulator S_i such that $\{S_i(\mathsf{in}_i, f_i(\mathsf{in}_1, \mathsf{in}_2))\} \equiv \{\mathrm{VIEW}_\pi(P_i), \mathsf{out}_i\}$

Homomorphic Encryption. Our constructions use a semantically secure additively homomorphic encryption scheme. In an additively homomorphic encryption scheme, $\mathsf{Enc}(m_1) \cdot \mathsf{Enc}(m_2) = \mathsf{Enc}(m_1 + m_2)$ which also implies that $\mathsf{Enc}(m)^a = \mathsf{Enc}(a \cdot m)$. While any encryption

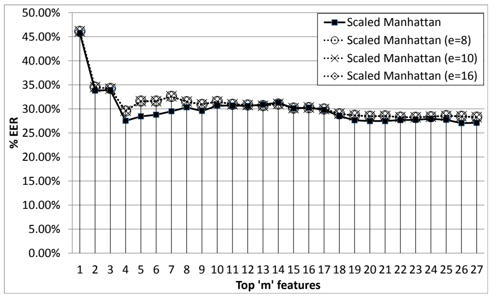


(a) Scaled Manhattan Verifier

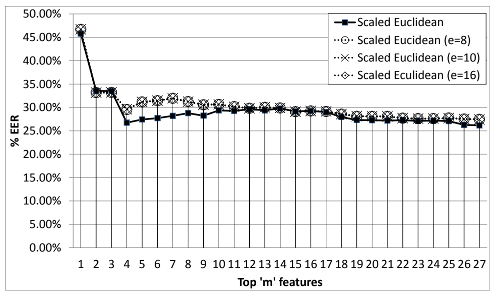


(b) Scaled Euclidean Verifier

Figure 1: %EERs of scaled Manhattan (a) and scaled Euclidean (b) verifiers for m-ranked feature subsets obtained with normalized mutual information criterion. The solid plot shows %EERs with real-valued features and dotted plots show %EERs with discretized features with e values 8, 10, and 16.



(a) Scaled Manhattan Verifier



(b) Scaled Euclidean Verifier

Figure 2: %EERs of scaled Manhattan (a) and scaled Euclidean (b) verifiers for m-ranked feature subsets obtained with MAD criterion. The solid plot shows %EERs with real-valued features and dotted plots show %EERs with discretized features with e values 8, 10, and 16.

scheme with the above properties (such as the well known Paillier encryption scheme [27]) suffices for the purposes of this work, the construction due to Damgård et al. [9, 8] (DGK) is of particular interest here because it is fast and it produces small ciphertexts. In DGK a public key consists of (1) a (small, possibly prime) integer u that defines the plaintext space; (2) k-bit RSA modulus $N = pq$ such that p and q are $k/2$-bit primes, v_p and v_q are t-bit primes, and $uv_p|(p-1)$ and $uv_q|(q-1)$; and (3) elements $g, h \in \mathbb{Z}_N^*$ such that g has order uv_pv_q and h has order v_pv_q. Given a message $m \in \mathbb{Z}_u$, encryption is performed as $\mathsf{Enc}(m) = g^m h^r \bmod N$, where $r \leftarrow \{0,1\}^{2.5t}$. We refer the reader to [9, 8] for any additional information.

In the rest of the paper, we use $\mathsf{HE}_s(m)$ to refer to the DGK encryption of message m under the server's public key. (The server is assumed to have access to the corresponding decryption key.)

Homomorphic Comparison. Our privacy-preserving protocol for computing scaled Manhattan distance requires privacy-preserving comparison of two encrypted values. For this task, we rely on the comparison protocol of Erkin et al. [11]. This protocol is based on the observation that $x < y$ is true iff the l-th bit of $a = 2^l + x - y$ is 1. Given $\mathsf{Enc}(x)$, encryption of a is computed by the client as $\mathsf{Enc}(a) = \mathsf{Enc}(2^l) \cdot \mathsf{Enc}(x) \cdot \mathsf{Enc}(y)^{-1}$. Encryption of the l-th bit of a is then computed as $\mathsf{Enc}(a_l) = \mathsf{Enc}(2^{-l} \cdot (a - (a \bmod 2^l))$. Value a is available to the client only in encrypted form, and computing $a \bmod 2^l$ in the encrypted domain requires interaction between the client and the server: the client "masks" $\mathsf{Enc}(a)$ by selecting a random value r and computing $\mathsf{Enc}(a') = \mathsf{Enc}(a) \cdot \mathsf{Enc}(r)$. Then, the client sends $\mathsf{Enc}(a')$ to the server, who decrypts it and returns the encryption of $c = a' \bmod 2^l$ to the client. Next, the client "unmasks" $\mathsf{Enc}(c)$ by computing $\mathsf{Enc}(c) \cdot \mathsf{Enc}(r)^{-1} = \mathsf{Enc}(a \bmod 2^l)$. We refer the reader to [11] for any additional information.

Symmetric Encryption. Our protocols use a semantically secure symmetric encryption scheme. For this purpose, we rely on AES in counter (CTR) mode. This mode is known to be semantically secure under the assumption that AES is a pseudorandom permutation [20]. We use $\mathsf{SE}_c(m)$ to indicate symmetric encryption performed under a key known by the client.

4.2. Protocols Description

Enrollment Phase. During enrollment, user biometrics are collected, encrypted and stored on the authentication server. The server does not have access to an unencrypted copy of the biometrics.

The client generates a template $Y = \{y_1, \ldots, y_n\}$ and the corresponding values $\sigma_1, \ldots, \sigma_n$. Let $\alpha_i = 1/\sigma_i$. Then, for scaled Manhattan, $\llbracket Y \rrbracket$ is computed as:

$$\llbracket Y \rrbracket = \mathsf{SE}_c(\mathsf{HE}_s(\alpha_1 y_1), \ldots, \mathsf{HE}_s(\alpha_n y_n))$$

while for scaled Euclidean we have:

$$\llbracket Y \rrbracket = \mathsf{SE}_c(\mathsf{HE}_s(\alpha_1 y_1^2), \mathsf{HE}_s(\alpha_i 2y_n), \ldots, \\ \mathsf{HE}_s(\alpha_1 y_n^2), \mathsf{HE}_s(\alpha_n 2y_n))$$

Verification Phase. We designed privacy-preserving protocols for computing scaled Manhattan distance (Figure 3), and scaled Euclidean distance (Figure 4). In both protocols, client's input is a vector X (which represents a biometric sample from the user), server's public key and the symmetric key used to protect the template. Server's input is the decryption key for $\mathsf{HE}_s(\cdot)$ and $\llbracket Y \rrbracket$, which corresponds to the encrypted template Y collected during enrollment. (We denote the party that does not receive any output from the protocol as the client, and the party that learns the outcome of the protocol as the server.)

4.3. Security Analysis

Security of our protocols relies on the security of the underlying building blocks. In particular, we need to assume that $\mathsf{HE}_s(\cdot)$ is a semantically secure homomorphic encryption scheme, and that $\mathsf{SE}_c(\cdot)$ is a semantically secure symmetric encryption scheme.

We instantiated $\mathsf{HE}_s(\cdot)$ using the DGK encryption scheme, which has been shown to be semantically secure under a hardness assumption that uses subgroups of an RSA modulus [9, 8]. We used AES-CTR for $\mathsf{SE}_c(\cdot)$, which is a construction secure under the assumption that AES is pseudorandom permutation [20].

The privacy-preserving comparison protocol of Erkin et al. was shown to be secure in [11], and therefore we do not include it in our analysis.

To show the security of the protocols, we sketch how to simulate the view of each party using its inputs and outputs alone. If such simulation is indistinguishable from the real execution of the protocol, for semi-honest parties this implies that the protocols do not reveal any unintended information to the participants (i.e., they learn only the output and what can be deduced from their respective inputs and outputs).

Privacy-Preserving Scaled Manhattan Distance. Since $\mathsf{SE}_c(\cdot)$ is semantically secure, the server cannot extract any information from $\llbracket Y \rrbracket$.

The server's view of the protocols consists of the decryption key for $\mathsf{HE}_s(\cdot)$, encrypted vector $\llbracket Y \rrbracket$, and ciphertext $\mathsf{HE}_s(d)$ from the client. (The server's view also contains the message exchanged during the comparison and multiplication protocols. We ignore these messages since the two protocols have been proven secure in [11].) The server's output is d/n. Simulator S_s provides the server with $\llbracket Y \rrbracket$ and with the decryption key for $\mathsf{HE}_s(\cdot)$ as input. It then uses the protocol output d/n to build $\mathsf{HE}_s(d)$, which is sent to the server. Since $\mathsf{HE}_s(d)$ is properly distributed, the server cannot distinguish between the simulation and a real execution of the protocol. Therefore, the protocol is secure against a curious server.

The client's view of the protocol consists in the server's public key, the symmetric key for $\mathsf{SE}_c(\cdot)$, X and $\llbracket Y \rrbracket$. The client has no output. Simulator S_c selects a random set of values $y'_1, \ldots, y'_n$, constructs $\llbracket Y' \rrbracket = \mathsf{SE}_c(\mathsf{HE}_s(\alpha_1 y'_1), \ldots, \mathsf{HE}_s(\alpha_n y'_n))$ and sends it to the client. The semantic security of $\mathsf{HE}_s(\cdot)$ prevents the client from determining that $\llbracket Y' \rrbracket$ corresponds to the encryption of random values. Therefore, $\llbracket Y' \rrbracket$ is properly distributed. For this reason, the client cannot distinguish between interaction with the S_c and with a honest server. Hence the protocol is secure against a curious client.

Privacy-Preserving Scaled Euclidean Distance. The semantic security of $\mathsf{SE}_c(\cdot)$ prevents the server from extracting any information from $\llbracket Y \rrbracket$.

The server's view of the protocols consists of the decryption key for $\mathsf{HE}_s(\cdot)$, encrypted vector $\llbracket Y \rrbracket$, and ciphertext $\mathsf{HE}_s(d)$ from the client. The server's output is $(\sqrt{d})/n$. Simulator S_s provides the server with $\llbracket Y \rrbracket$ and the decryption key for $\mathsf{HE}_s(\cdot)$ as input. It then uses $(\sqrt{d})/n$ to construct $\mathsf{HE}_s(d)$, and sends it to the server. Since $\mathsf{HE}_s(d)$ is properly distributed, the server cannot distinguish between the simulation and a real execution of the protocol. Therefore, the protocol is secure against a curious server.

The client's view of the protocol consists in the server's public key, the symmetric key for $\mathsf{SE}_c(\cdot)$, X and $\llbracket Y \rrbracket$. The client has no output. Simulator S_c selects a random set of values $y'_1, \ldots, y'_n$, constructs $\llbracket Y' \rrbracket = \mathsf{SE}_c(\mathsf{HE}_s(\alpha_1 (y'_1)^2), \mathsf{HE}_s(2\alpha_1 y'_1), \ldots, \mathsf{HE}_s(\alpha_n (y'_n)^2), \mathsf{HE}_s(2\alpha_n y'_n))$ and sends $\llbracket Y' \rrbracket$ to the client. The semantic security of $\mathsf{HE}_s(\cdot)$ prevents the client from determining that $\llbracket Y' \rrbracket$ corresponds to the encryption of random values. Therefore, $\llbracket Y' \rrbracket$ is properly distributed. For this reason, the client cannot distinguish between interaction with the S_c and with a honest server. Hence the protocol is secure against a curious client.

5. Performance Analysis

The computational complexity of our scaled Manhattan protocol is $\mathcal{O}(n)$ where $n = |X|$, i.e., linear in the number of features for both client and server. The complexity of our

Input: Client: sample $X = (x_1, \ldots, x_n)$, server's public key and decryption key for $\mathsf{SE}_c(\cdot)$; Server: encrypted template $[\![Y]\!] = \mathsf{SE}_c(\mathsf{HE}_s(\alpha_1 y_1), \ldots, \mathsf{HE}_s(\alpha_n y_n))$ (where $\alpha_i = 1/\sigma_i$) and decryption key for $\mathsf{HE}_s(\cdot)$.
Output: The server learns $\mathsf{D}_M(X, Y)$.
Protocol steps:

1. The server sends $[\![Y]\!]$ to the client, which decrypts it, obtaining $\mathsf{HE}_s(\alpha_1 y_1), \ldots, \mathsf{HE}_s(\alpha_n y_n)$.
2. For $i = 1, \ldots, n$, the client and the server interact in a privacy-preserving comparison protocol. At the end of the protocol the client learns the encryption of bit $b_i = (\alpha_i x_i < \alpha_i y_i)$
3. For $i = 1, \ldots, n$, the client computes: $\mathsf{HE}_s(d_i) = \mathsf{HE}_s(|\alpha_i x_i - \alpha_i y_i|) = \mathsf{HE}_s(MAX(\alpha_i x_i, \alpha_i y_i) - MIN(\alpha_i x_i, \alpha_i y_i)) = \mathsf{HE}_s((b_i \cdot (\alpha_i y_i - \alpha_i x_i) + \alpha_i x_i) - (b_i \cdot (\alpha_i x_i - \alpha_i y_i) + \alpha_i y_i))$ as:

$$\mathsf{HE}_s(d_i) = \mathsf{HE}_s(b_i \cdot \alpha_i y_i)^2 \cdot \mathsf{HE}_s(b_i \cdot \alpha_i x_i)^{-2} \cdot \mathsf{HE}_s(\alpha_i x_i) \cdot \mathsf{HE}_s(\alpha_i y_i)^{-1}$$

   The computation of $\mathsf{HE}_s(d_i)$ requires client and server to perform a short interactive protocol for computing $\mathsf{HE}_s(b_i \cdot \alpha_i x_i)$ and $\mathsf{HE}_s(b_i \cdot \alpha_i x_i)$ [11].
4. Then, the client computes:

$$\mathsf{HE}_s(d) = \mathsf{HE}_s\left(\sum_{i=1}^{n} d_i\right) = \prod_{i=1}^{n} \mathsf{HE}_s(d_i)$$

5. The client sends $\mathsf{HE}_s(d)$ to the server, which decrypts it and outputs $\mathsf{D}_M(X, Y)$ as d/n.

Figure 3: Computation of Privacy-Preserving Scaled Manhattan Distance

Input: Client: sample $X = (x_1, \ldots, x_n)$, decryption key for $\mathsf{SE}_c(\cdot)$ and server's public key; Server: encrypted template $[\![Y]\!] = \mathsf{SE}_c(\mathsf{HE}_s(\alpha_1 y_1^2), \mathsf{HE}_s(\alpha_1 2 y_1), \ldots, \mathsf{HE}_s(\alpha_n y_n^2), \mathsf{HE}_s(\alpha_n 2 y_n))$ (where $\alpha_i = 1/\sigma_i$) and decryption key for $\mathsf{HE}_s(\cdot)$.
Output: The server learns $\mathsf{D}_E(X, Y)$.
Protocol steps:

1. The server sends $[\![Y]\!]$ to the client, which decrypts it as $\mathsf{HE}_s(\alpha_1 y_1^2), \mathsf{HE}_s(\alpha_1 2 y_1), \ldots, \mathsf{HE}_s(\alpha_n y_n^2), \mathsf{HE}_s(\alpha_n 2 y_n)$.
2. For $i = 1, \ldots, n$, the client computes:

$$\mathsf{HE}_s(d_i) = \mathsf{HE}_s\left(\alpha_i \cdot (x_i - y_i)^2\right) = \mathsf{HE}_s\left(\alpha_i \cdot (x_i^2 + y_i^2 - 2x_i y_i)\right) = \mathsf{HE}_s\left(\alpha_i x_i^2\right) \cdot \mathsf{HE}_s\left(\alpha_i y_i^2\right) \cdot \mathsf{HE}_s\left(\alpha_i 2 y_i\right)^{-x_i}$$

3. Then, the client computes

$$\mathsf{HE}_s(d) = \mathsf{HE}_s\left(\sum_{i=1}^{n} d_i\right) = \prod_{i=1}^{n} \mathsf{HE}_s(d_i)$$

4. The client sends $\mathsf{HE}_s(d)$ to the authentication server, which computes d and outputs $\mathsf{D}_E(X, Y)$ as $(\sqrt{d})/n$.

Figure 4: Computation of Privacy-Preserving Scaled Euclidean Distance

scaled euclidean protocol is also $\mathcal{O}(n)$ for the client, and $\mathcal{O}(1)$ for the server, i.e., constant in the number of features. In fact, the server only needs to decrypt one ciphertext regardless of the number of features involved in the protocol.

In terms of communication, both protocols require the server to send $[\![Y]\!]$ to the client. Additionally, the protocol that computes scaled Manhattan distance also requires the parties to run n instances of the comparison and multiplication protocols, both of which exchange constant number of messages. As an optimization, if the client caches a copy of $[\![Y]\!]$, then the communication cost of computing the scaled Euclidean distance is reduced to $\mathcal{O}(1)$.

We performed experiments on a Linux server with two Intel Xeon E5420 CPUs at 2.5 GHz. The server software is written in C and relies on the GNU GMP library. For the client we used a Samsung Galaxy Nexus smartphone, with dual core ARM Cortex A9 CPU running at 1.2 GHz and Android 4.2. The client software is written in Java and uses the BigInteger library.

From verification results in Section 3.4, we determined that between 4 and 20 touch features yield lowest %EERs. In Table 1, we summarize the performance results of our protocol implementation in this range. Results in Table 1 are obtained using $e = 10$ (i.e., each feature is represented using 10 bits) which offers the best tradeoff between %EER and protocol overhead (i.e., time to compute the distance between the template and the user sample).

Although all devices used for the experiments have multiple CPU cores, for the sake of generality all experiments have been run on a single core. Both protocols can take advantage of multiple cores by performing multiple instances of Step 3 of the scaled Manhattan protocol and Step 2 of the scaled Euclidean protocol concurrently on the client. On the server, multiple cores can be used to run concurrent

Table 1: Performance of our prototype implementation.

	Scaled Manhattan		Scaled Euclidean	
# of Feat.	**Server**	**Client**	**Server**	**Client**
4	23 ms	520 ms	≈ 1 ms	263 ms
8	47 ms	833 ms	≈ 1 ms	520 ms
12	70 ms	1245 ms	≈ 1 ms	781 ms
16	95 ms	1676 ms	≈ 1 ms	1039 ms
20	120 ms	2102 ms	≈ 1 ms	1302 ms

instances of the comparison protocol for scaled Manhattan, although this may not reduce overall protocol execution time when the server is interacting with a smartphone.

6. Conclusions

In this paper, we introduced the first efficient privacy-preserving protocols for securely outsourcing continuous authentication with touch data. Our protocols allow a smartphone and an authentication server to privately compute *exact* scaled Euclidean and scaled Manhattan distances between their respective inputs. Furthermore, our protocols do not disclose any additional information about the parties' input. The security of our protocol is based on standard assumptions and supported by formal security proofs in the semi-honest model, presented in the paper.

We performed experiments to demonstrate the accuracy and practicality of our protocols. Our experiments confirm that the EER of the protocols is low. Furthermore, the overhead introduced by the privacy-preserving computation of Euclidean and Manhattan distance is between 263ms and 2.1s, depending on the specific parameters, using a commodity Android smartphone.

References

[1] AdminOne Security – Identity Assurance as a Service. `http://www.admitonesecurity.com`.

[2] BBC News - End of line for online passwords, says PayPal. `http://www.bbc.co.uk/news/business-21577594`.

[3] Behaviosec. `http://www.behaviosec.com`.

[4] More top worst passwords. `http://xato.net/passwords/more-top-worst-passwords#more-269`.

[5] M. Barni, T. Bianchi, D. Catalano, M. Di Raimondo, R. Labati, P. Failla, D. Fiore, R. Lazzeretti, V. Piuri, F. Scotti, and A. Piva. Privacy-preserving fingercode authentication. In *MM&Sec*, 2010.

[6] M. Blanton and P. Gasti. Secure and efficient protocols for iris and fingerprint identification. In *ESORICS*, 2011.

[7] R. Cramer, I. Damgård, and J. Nielsen. Multiparty computation from threshold homomorphic encryption. In *EUROCRYPT*, 2001.

[8] I. Damgård, M. Geisler, and M. Krøigård. A correction to efficient and secure comparison for on-line auctions. Cryptology ePrint Archive, Report 2008/321, 2008.

[9] I. Damgård, M. Geisler, and M. Krøigård. Homomorphic encryption and secure comparison. *Journal of Applied Cryptology*, 1(1), 2008.

[10] I. Damgård, M. Geisler, and M. Krøigård. Asynchronous multiparty computation: Theory and implementation. In *PKC*, 2009.

[11] Z. Erkin, M. Franz, J. Guajardo, S. Katzenbeisser, I. Lagendijk, and T. Toft. Privacy-preserving face recognition. In *PETS*, 2009.

[12] Tao Feng, Ziyi Liu, Kyeong-An Kwon, Weidong Shi, B. Carbunar, Yifei Jiang, and N. Nguyen. Continuous mobile authentication using touchscreen gestures. In *HST*, 2012.

[13] M. Frank, R. Biedert, E. Ma, I. Martinovic, and D.Song. Touchalytics: On the applicability of touchscreen input as a behavioral biometric for continuous authentication. *IEEE Transactions on Information Forensics and Security*, 8(1):136–148, 2013.

[14] O. Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge University Press, 2004.

[15] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *STOC*, 1987.

[16] W. Henecka, S. Kogl, A.-R. Sadeghi, T. Schneider, and I. Wehrenberg. TASTY: Tool for Automating Secure Two-partY computations. In *CCS*, 2010.

[17] A. Jain, S. Prabhakar, L. Hong, and S. Pankanti. Filterbank-based fingerprint matching. *IEEE Transactions on Image Processing*, 9(5), 2000.

[18] A. Juels and M. Sudan. A fuzzy vault scheme. *Des. Codes Cryptography*, 38(2), 2006.

[19] A. Juels and M. Wattenberg. A fuzzy commitment scheme. In *CCS*, 1999.

[20] J. Katz and Y. Lindell. *Introduction to Modern Cryptography*. Chapman & Hall/CRC, 2008.

[21] K. Killourhy and R. Maxion. Comparing anomaly-detection algorithms for keystroke dynamics. In *DSN-09*, 2009.

[22] G. Kumar, S. Tulyakov, and V. Govindaraju. Combination of symmetric hash functions for secure fingerprint matching. In *ICPR*, 2010.

[23] Lingjun Li, Xinxin Zhao, and Guoliang Xue. Unobservable reauthentication for smartphones. In *NDSS*, 2013.

[24] Tao Liu, Shengping Liu, and Zheng Chen. An evaluation on feature selection for text clustering. In *ICML*, 2003.

[25] D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella. Fairplay – a secure two-party computation system. In *USENIX Security Symposium*, 2004.

[26] M. Osadchy, B. Pinkas, A. Jarrous, and B. Moskovich. SCiFI – A system for secure face identification. In *IEEE Symposium on Security and Privacy*, 2010.

[27] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT*, 1999.

[28] T. Rabin and M. Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority. In *STOC*, 1989.

[29] A.-R. Sadeghi, T. Schneider, and I. Wehrenberg. Efficient privacy-preserving face recognition. In *ICISC*, 2009.

[30] E. Shi, Y. Niu, M. Jakobsson, and R. Chow. Implicit authentication through learning user behavior. In *ISC*, 2010.

[31] U. Uludag, S. Pankanti, and A. Jain. Fuzzy vault for fingerprints. In *AVBPA*, 2005.

[32] A. Yao. How to generate and exchange secrets. In *FOCS*, 1986.