Reconstruction Attacks against Mobile-based Continuous Authentication Systems in the Cloud

Mohammad Al-Rubaie, *Student Member, IEEE,* and J. Morris Chang, *Senior Member, IEEE*

***Abstract*—Continuous authentication for mobile devices using behavioral biometrics is being suggested to complement initial authentication for securing mobile devices, and the cloud services accessed through them. This area has been studied over the past few years, and low error rates were achieved; however, it was based on training and testing using SVM and other non-privacy-preserving machine learning algorithms. To stress the importance of carefully designed privacy-preserving systems, we investigate the possibility of reconstructing gestures raw data from users' authentication profiles or samples testing results. We propose two types of reconstruction attacks based on whether actual user samples are available to the adversary (as in SVM profiles) or not. We also propose two algorithms to reconstruct raw data: a numerical-based algorithm that is specific to one compromised system, and a randomization-based algorithm that can work against almost any compromised system. For our experiments, we selected one compromised and four attacked gesture-based continuous authentication systems from the recent literature. The experiments, performed using a public data set, showed that the attacks were feasible, with a median ranging from 80% to 100% against one attacked system using all types of attacks and algorithms, and a median ranging from 73% to 100% against all attacked systems using the randomization-based algorithm and the negative support vector attack. Finally, we analyze the results, and provide recommendations for building active authentication systems that could resist reconstruction attacks.**

***Index Terms*—Mobile devices, continuous authentication, gestures, privacy, reconstruction attacks, machine learning.**

I. Introduction

DIGITAL media access in the United States has been dominated by mobile devices since 2014. The time spent on mobile apps has increased to 52% of the total U.S. digital media time, compared to only 40% for desktop access [1]. Mobile browser usage formed the remaining 8%, adding to the total mobile device usage. Such reliance on mobile devices is coupled with an increase in using cloud storage services for storing personal data to enable easy backup, and easy access across multiple devices. However, such trends were not accompanied by an increase in the security of authentication to these services. This was clearly demonstrated by the iCloud hacking incident in 2014 that exposed victims' private data [2], [3]. It was not done from the victims' own mobile devices, and was a targeted attack on user names, passwords and security questions [2]. While two-factor authentication is normally suggested, it would be tedious for users to use these lengthy steps every time they need to access an online service. Hence, continuous authentication (also called active or implicit authentication, and used interchangeably hereinafter) is a possible solution to this issue as it is transparent to users while providing security based on their behavioral biometrics.

Active authentication (AA) on mobile devices has received significant attention over the past few years with solutions utilizing touchscreen gestures ([4]–[7]), keystrokes ([8]), sensors ([9]) or more than one input type (e.g., [10]). With the exception of papers like [11], almost all active authentication research on mobile devices utilized regular machine learning methods that didn't preserve the privacy of users. Almost all of them used Support Vector Machines (SVM) for classification, either as the only algorithm, or one of few. Systems that used SVM achieved lower error rates than the few ones that used privacy-preserving methods.

Continuing to rely on SVM, or similar algorithms, without studying their effect on users' privacy and security might give a premature expectation that AA systems are ready to be widely deployed. It is important to notice that SVM (and k-NN) stores actual feature vector samples in user authentication profiles. In each user's profile, some of these samples belong to the user herself: positive support vectors (PSVs). The rest of the samples belong to other users: negative support vectors (NSVs). Each feature vector contains information that represents an object's important characteristics. For gestures, this information (features) might include average velocity and trajectory length. Using such features to forge raw data has not been addressed in the literature and, up till now, it was unclear how to perform this reconstruction (as was also reported by [12]). It is worth noting that reconstructing fingerprint images from minutiae templates was proven to be possible although these templates are very compact, and many did not think it was possible at first [13]–[16]. We believe that it is important to understand the possibility of reconstructing raw data from users' authentication profiles before designing secure and privacy-preserving AA systems.

In this paper, we investigate the problem of reconstructing behavioral biometrics on mobile devices. Since gestures received the most attention in the recent literature ([4]–[7], [12], [17]–[26]), it was chosen to be studied in this paper. We foresee that cloud services would start adopting behavioral-based AA systems for mobile devices to prevent attacks like the iCloud hacking incident mentioned earlier [2], [3]. Such services might be possible targets for successful attacks. In fact, even large companies were targeted by successful attacks that revealed both personal and corporate private data [27]. If one of these cloud web services that uses behavioral biometrics

M. Al-Rubaie and J. M. Chang are with the Department of Electrical and Computer Engineering, Iowa State University, Ames, IA, 50011 USA e-mail: mti and morris@iastate.edu.

for authentication was hacked, a user with an account on these systems might have their behavioral biometrics-based profiles compromised. If these profiles were not privacy-preserved, a reconstruction attack could potentially be launched to reconstruct the raw data that could be used afterwards to hack into that user's accounts on other cloud services. Furthermore, we investigate the possibility of reconstructing raw gesture data even in the absence of user samples in their profiles. This could be done by utilizing the output of user samples' testing such as the SVM decision value or the logistic regression probability. Such type of attack can even be launched by an adversary with no access to the AA server.

Unlike passwords or digital certificates, behavioral biometrics cannot be revoked once compromised, and the users' behavior might not change quickly enough overtime for the compromised profiles to be obsolete [28]. Hence, we intend to highlight the importance of using carefully designed privacy-preserving AA systems by studying reconstruction attacks.

To the best of our knowledge, this is the first work that proposes reconstruction attacks against gesture-based AA systems. The contributions of this paper are as follows:

First. We propose two types of reconstructions attacks depending on the amount of private information leaked to the adversary. The first one (full profile attack) closely resembles the current usage of SVM in AA systems as it assumes that both PSVs and NSVs are stored in the clear within the authentication profile. These are the actual feature vectors that belong to the profile owner herself, and other users, and can possibly be utilized to reconstruct raw data that is used to hack into user accounts on other systems.

On the other hand, the decision value attack assumes that none of the user samples were available for the adversary (since he has no access to the AA server or the victim's own device). Alternatively, this adversary could submit a user sample to the AA server for testing and retrieve a decision value. To perform the decision value attack, we developed an algorithm that starts with a generic feature vector, and randomizes it while monitoring the change in the SVM decision value. This allows the algorithm to finally obtain feature vectors that can be used to reconstruct raw data using our reconstruction algorithms. The decision value attack shares the same target of recovering feature vectors as the model inversion attack proposed in [29] (section II). However, their algorithm is based on gradient descent while ours is based on randomization.

Second. We developed two reconstruction algorithms for mobile gestures in order to assess the vulnerabilities of AA systems in both proposed attack scenarios. Both of these algorithms take feature vectors as input and output raw gesture data. The first algorithm is numerical, and we use numerical estimation and procedures on the summary statistics contained in the feature vector to reconstruct the raw data. We further test our reconstructed raw data against the user's profile, and only use the raw gesture data that passes as user's samples.

When designing a numerical algorithm, it has to be tailored to a single compromised system. Hence, we developed a randomization algorithm that can reconstruct raw data from almost any compromised system. This algorithm has the added benefit of reconstructing raw data that can yield the closest feature vector possible to the original feature vector. The randomization algorithm starts with generic raw data, and continuously randomizes it while monitoring how close the extracted feature vector is to the original one. It stops when no further randomization can help reduce the mean square error (MSE) between the reconstructed feature vector and original feature vector.

Third. We tested both algorithms and the two types of attacks using two mobile gesture classifiers: left-to-right and right-to-left swipes. For this purpose, we used a public data set that was provided by Frank et al [7]. We reviewed the literature to select one AA system as a compromised system, and four others to be the attacked systems against which the reconstructed raw data would be used. Our tests revealed the feasibility of reconstruction attacks using the proposed algorithms. The experiments showed high attack success rates with a median ranging from 73% to 100% against all attacked systems using the randomization algorithm and the full-profile attack (with negative support vectors).

Fourth. We analyze the results, and provide recommendations for future design attempts of AA systems that could resist our proposed reconstruction attacks. Finally, we show the effectiveness of these recommendations through experiments.

The remainder of this paper is organized as follows. We review the related work in Section II. In Section III, we introduce the threat model and the two reconstruction attacks. The numerical and randomization reconstruction algorithms are presented in section IV. Section V presents the experimental results and their analysis while section VI concludes this paper.

II. Related Work

Attacks on behavioral biometrics Non-zero effort attacks on behavioral biometrics did not receive as much attention as it should have. On gesture-based AA systems, Phoha et al [30] used a simple "lego" robotic arm to forge gestures based on general population statistics. Their results indicated a noticeable increase in EER when considering their non-zero effort attack instead of the zero-effort attacks normally considered in the literature. However, we have a different threat model, as we target online services protected by a gesture-based AA system, while they targeted information stored on owners' devices. Moreover, they used their own feature vector while we select five from the literature: one to be a target of our reconstruction attack, and four others for testing. More recently, Gong et al [12] proposed a forgery-resistant method that works especially well against robotic arm forgeries, by utilizing the impact of screen settings on users' behavior. However, it does not address our threat model.

Model Inversion Attacks Fredrikson et al [29] proposed model inversion attacks that used the confidence value revealed along with a classification result to recover feature vectors that were used to build the model. Their case studies used the raw data as features. In their face recognition case study, they recovered feature vectors which were the face images themselves that were used for training the models. Whereas in gesture-based active authentication, gesture raw data cannot be used directly as feature vectors. Therefore, we develop

reconstruction algorithms to reconstruct the raw gesture data from feature vectors. This step was not performed in [29] since the feature vectors matched the raw data in their case studies.

Reconstruction Attacks While being in another biometrics-research area, the closest work to ours is that related to reconstructing fingerprints using minutiae-based templates. Minutiae-based templates are very compact, and for a long time, it was assumed that reconstructing the original fingerprint image from them was not possible. That was challenged by [15] and [16], and was later followed by other successful attempts like [13]. The closest of these attempts to our work are likely those similar to [14] where a minutiae-based template is used to reconstruct another type of representation, namely phase image in [14]. This template is later used to reconstruct the fingerprint greyscale image. Its similarity to our work stems from its use of one type of representation to create another type of representation. We might think of different feature vectors as different representations of the raw gesture data. To the best of our knowledge, ours is the first research that addresses the issue of reconstructing the raw gesture data from feature vectors. We add the extra challenge of using the reconstructed raw gesture data to attack additional AA systems that utilize different feature vectors than the compromised one.

Gesture-based active authentication. Gesture-based AA research is presented in this subsection. Our aim is to select a group of systems that can be utilized for testing our hypothesis. For papers to be selected, they had to clearly define a feature vector which is required by SVM (and other machine learning algorithms like logistic regression, k-NN and neural networks) since we are studying the privacy vulnerabilities of such algorithms. Moreover, the features in the selected systems had to be diverse to be able to study the effect of having different features on our reconstruction attack algorithms. After examining the recent literature, we were able to find a group of papers that matched the previous criteria and had other desirable characteristics: (1) they were less susceptible to over-fitting as their feature vectors were not too long; (2) they were flexible as they enabled testing a single sample, as well as aggregate testing of more than one (as opposed to some papers that only allowed aggregation of testing samples), and (3) they did not use special equipment or testing conditions.

We start by presenting the papers that matched our criteria mentioned above. In [7], the authors selected features that represented location, shape, velocity, and acceleration of strokes. Pressure and area only had one feature each. They used SVM and k-NN for classification to obtain approximately 13% EER for one-stroke, and 0-4% median EER when bundling 11 strokes together for testing. Li et al [6] did a study on gestures and taps. After feature selection, they chose 10 gesture features that represented location, shape, duration, and area of strokes. They used SVM for classification. In [4], an evaluation of 10 classification algorithms was done. Logistic regression and SVM were found to be the two best performing classifiers. They used 28 features that described location, shape, velocity, acceleration, area and pressure of strokes. Xu et al [10] performed a study on gestures, taps, pinches, and handwriting on touch screens. As for gestures, they chose 37 features representing location, shape, velocity, area and pressure of strokes, and used SVM for classification. In [31] and [5], the authors chose 15 features that were a subset of the features in [7]; and used k-NN, SVM and Random Forest for classification. The previously mentioned papers were selected for our testing purposes since they all mentioned their feature vectors clearly. They also matched our selection criteria including diverse selection of features.

The papers that were not selected include [17] and [18] which relied on graphical techniques to create a graphic touch gesture feature (GTGF) in [17], and improved the performance on mobile devices in [18]. It was not selected as it did not use a regular feature vector. There was a similar case with [19] that used HMM-based classification. Velten et al [20] had a much longer feature vector than the number of samples per user that we have in the public data set from [7], so we couldn't use it as it would lead to overfitting. In [21], 10 minutes worth of samples were needed for a decision, allowing intruders long time to carry their attacks. The features from [7] were used in both [22] and [23], so we excluded them. The features were used in [22] to test new classification algorithms, and in [23] to test application-centric active authentication. The features in [12] were also the same as in [7] but were used with different screen settings to resist forgery attacks; however, we did not have the appropriate data set to do the testing. A similar issue with the data set happened in [24] and [25]. In [24], a glove was used to complement the features collected from the touch screen, and in [25], the authors proposed analyzing touch interactions with common user interface (UI) elements (e.g., button, checkboxes and sliders). Finally, [26] was not selected because of the absence of a detailed list of features.

III. Preliminaries and Attacks Overview

In this section, we start with a brief description of classification using SVM, since we are targeting systems that use SVM and similar machine learning algorithms. We follow that by describing the system model and attack scenarios. Finally, we present our choice of the compromised and attacked systems.

A. SVM Classification and System Evaluation

To perform classification, *raw* data belonging to multiple classes (e.g., different users) need to have its features extracted to create *feature vectors*. For mobile gestures, such features may include trajectory length and average velocity. Before using these feature vectors to train an AA profile, normalization (or standardization) can be applied, which might include normalizing the feature values to be between -1 and +1, for example. In a binary classification problem, we want to know if the new sample $\mathbf{x}$ (feature vector) belongs to the user k (positive class) or not (negative class). Thus, when training profiles, for each user, all their training data is placed in the positive class, and all other users' training data is placed in the negative class (one-versus-all classification).

Support Vector Machines (SVM) [32] is a maximal margin classifier, meaning that it tries to maximize the margin (distance) between classes, rather than merely choosing any separating hyperplane. After solving the SVM problem in its dual form, the support vectors are identified. Some of these

support vectors belong to the positive class while others belong to the negative class (formed from many other classes). Thus, any user's profile is in reality formed of actual user samples. In fact, even if some user leaves the system, and her profile is deleted, some of her samples might still be left as NSVs in other users' profiles. Using these support vectors, any new sample $\mathbf{x}$ can be tested with the following equation:

$$f(\mathbf{x}) = \sum_{i:\alpha_i>0} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b \quad (1)$$

The sample $\mathbf{x}$ is then classified based on its SVM decision value (returned by eq. 1) either through varying a threshold, or based on the sign of the SVM decision function $f(\mathbf{x})$, which is equivalent to setting the threshold to zero.

AA systems are normally evaluated based on false acceptance rate (FAR) and false rejection rate (FRR). There's a tradeoff between FRR and FAR, and by varying the threshold used with the decision value from eq. 1, one can obtain different values for FAR and FRR. Some systems, or individual users, might want to have a smoother user experience by decreasing FRR at the expense of FAR, while others might want more security than convenience by decreasing FAR at the expense of FRR. In order to compare different systems more easily, equal error rate (EER) is used, and it is the value where FAR equals FRR. The value of EER can be determined by varying the SVM decision value threshold.

B. System Model and Design Goals

1) System Model: We consider a system with two main entities: a mobile device and a cloud service provider. The cloud service provider has an authentication server and an application server. A mobile device user wants to gain access to the application server, while ensuring that no intruders can obtain access to her account. Hence, the cloud service provider is using an AA server to continuously and transparently authenticate the user. An active authentication system consists of the following entities (fig. 1):

AA Server: provides a transparent way to continuously authenticate the user based on touch gestures. Since these systems protect cloud services, and might be accessed using different devices, the user profiles have to be stored on the authentication server itself. The user profiles were built using SVM since most papers report the efficacy of their systems based on SVM.

Client App: The user uses this app to gain access to the cloud services. This app collects raw gesture data R, extract the features from R, and sends the feature vector F to the AA server to be matched against the stored profile P. The decision value $decVal$ is returned to the app that would allow or prevent the app user based on that decision value.

2) Design Requirements: Extending the design goals given in [33], we outline the following four requirements:

Biometric Viability: A biometric modality that is used for continuous authentication has to satisfy several requirements including being universal (i.e. it applies to everyone), unique, *permanent* (i.e. it lasts at least through the lifetime of the system), unobtrusive, difficult to circumvent, cost effectiveness, and collectible [34]. The viability of using gestures as a biometric was shown in many studies including [7].

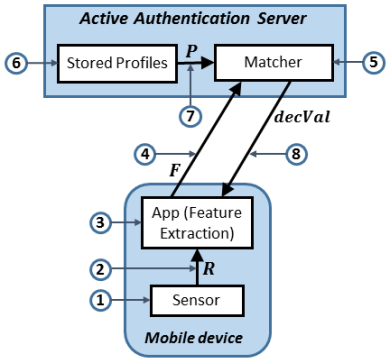


Fig. 1: Attack points in AA systems

Security and Privacy Requirements: The primary requirement of an AA system is to prevent illegitimate users from continuing to use the app. Thus, it protects the original user's private data in the cloud, which makes it a security and privacy requirement at the same time. Another privacy requirement is ensuring that none of the raw gesture data is sent to the AA server. Thus, if the AA server is compromised, hackers would not have access to such data [12]. Instead, only the feature vector is sent to the AA server (fig. 1).

Usability Requirement: Attempting to make the system more resistant to adversaries would require tuning the threshold to decrease FAR. However, decreasing FAR could potentially increase the FRR which would be obtrusive to usability. Hence, it would make sense to allow the user to tune her threshold to allow for more usability or security.

C. Problem Setting and Threats

When password databases are compromised [35], the damage can be limited to the related service. However, the permanence property of biometrics could jeopardize user accounts on other services, and might even prevent that user from using said biometrics in the future.

We foresee the existence of multiple online services that implement gesture-based AA systems. Each one of these services would implement their own AA system with a separate set of features. A user u can have accounts on more than one online service (e.g., Amazon, Google and eBay). She might also have accounts for other online services that also implement AA systems, but might have less secure infrastructure, which are called here the compromised systems $SysC$.

An adversary could use information leaked from $SysC$ to obtain raw gesture data R, which in turn can be used to access the victim's account on another online service provider, e.g. eBay or iCloud. The adversary might want to access the victim's account to perform purchases (eBay or Amazon), or to to gain access to their private data stored online (iCloud, Dropbox, or Gmail). We call these the attacked systems $SysA$.

As stated in Ratha et al [36], there are eight attack points on pattern recognition systems. We show them in fig. 1 as they apply to a mobile-based AA system. It is worth noting

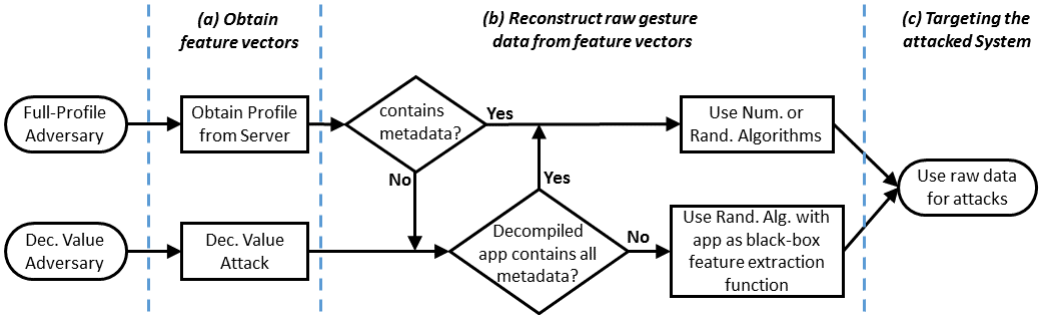


Fig. 2: Overview of attack procedures

that our attack regarding $SysA$ are limited to $type$ 1 (fig. 1), i.e. injecting forged raw data to the sensor, which is possible through replaying events directly within the mobile device (see [37]). We do not require any vulnerabilities in $SysA$ for our attacks to happen. As for $SysC$, we defer the discussion to the next section.

D. Adversary Model

The goal of the adversary is to utilize the biometric ***permanence*** property to gain access to a victim's cloud stored data on $SysA$ by using the victim's leaked biometric information from $SysC$.

We present two types of adversaries associated with two types of attacks: the full-profile attack and the decision value attack. Each one of these attacks has two stages: the first stage is where the adversary obtains his knowledge, uses different methods to do so for each attack. It basically comprises (a) obtaining feature vectors, and (b) reconstructing raw gesture data from these vectors (fig. 2). The second stage, which corresponds to step (c) in fig. 2, is similar for both adversary types. It includes injecting the raw data into the sensor while using the attacked system app. As stated before, this is the $type$ 1 attack point according to [36], and does not require the attacked system to have any vulnerabilities.

Before describing each adversary type, it should be noted that the numerical reconstruction algorithm (section IV-A) requires having the features metadata beforehand. This metadata contains the features' names and normalization parameters. On the other hand, the randomization algorithm can accept a black-box feature extraction function. Such black-box functionality can be achieved by the app itself. The adversary, using its account, can inject randomized raw data into the sensor and monitor the output feature vectors sent to the cloud. Even if the traffic is encrypted, a Man-in-the-middle HTTPS proxy could be utilized, and such software is readily available such as Fiddler, Charles or mitmproxy.

We characterize each of the two adversaries by describing how their knowledge is obtained (fig. 2):

- Full-profile (FP) adversary: This could be an internal entity with respect to the system with a legitimate system role (e.g., system admin, an employee). It can also be an external entity that hacked into the AA server (similar to the Sony hack [27] or the CSDN password database leak [35]). This adversary targets the privacy of the victim by overcoming the privacy safeguard of not uploading the raw gesture data to the AA server. If the features' metadata was available with the profile, or can be obtained through decompiling the app, this would be a passive adversary with respect to $SysC$; otherwise, it would be active adversary.
- Decision value (DV) adversary: This is a weaker adversary, and a relaxation from the previous one. We assume the adversary has no access to the victims mobile device or the AA server. The adversary operates a mobile app, impersonates a victim, and continuously sends feature vector samples to the AA server; thus being an active adversary, and monitors the returned decision value (e.g., by using MITM HTTPS proxy). This adversary targets the victims privacy by utilizing the decision value which is used mainly used for ***usability*** reasons. To obtain the metadata, this adversary can decompile the app (e.g., using a website for Android apps [38]). The adversary can also resort to using the app as a black-box feature extraction function as mentioned earlier.

An adversary is deemed successful in their attack when they can impersonate the victim. It is unrealistic to assume that an adversary can obtain success rates greater than $(1 - EER)$. Hence, if the success rate is equal to, or greater than, $(1 - EER)$, then we call the attack "successful". In order to decrease the EER, many studies aggregated results in the form of taking a majority vote of a group of gesture strokes. Since a majority can be obtained using 3 out of 5 gesture strokes (e.g. as was done in [10]), we call an attack "conditionally successful" if it had a success rate ranging from 0.6 to $(1 - EER)$.

E. Adversary Knowledge

This section corresponds to step (a) in fig. 2. We elaborate more on how each adversary obtains the feature vectors necessary for reconstructing the raw gesture data of the victim.

Full Profile Adversary: As stated earlier, this adversary gained access to the victim's profile which are regular SVM profiles as described in the system model (section III-B1). Such profiles contain both positive (PSVs) and negative support vectors (NSVs), and any other values needed for the SVM decision function in eq. 1.

For each user's profile, ***Positive support vectors*** are feature vector samples that belong to that specific user. If any of them

is used with the SVM decision function (eq. 1), the decision value would most likely equal +1 (or smaller with soft margin classifiers). To perform full profile attacks using PSVs, these samples can be directly extracted from these profiles, and used as input for our reconstruction algorithms. Other user samples that weren't selected to be PSVs might be more representative of user's behavior as they can give decision values that exceed +1. Some of these samples might be used as NSVs for other users' models.

Each user's model also contains ***Negative support vectors***. These are feature vector samples from other users that are used to help set the boundary of the negative class. They would most likely have decision values that equal -1 (or larger with soft margin classifiers). Our method involves: (1) collecting all the negative samples from all available profiles; (2) removing any samples that already exist as PSVs for any profile (To isolate the effect of only using NSVs); and finally (3) testing the remaining NSVs against each user's profile using the SVM decision function (eq. 1). The NSVs that yield decision values greater than +1 for a particular user are selected to be used in our attacks. We anticipate that these samples would give better attack results than the PSVs as they are more representative of user's behavior, especially that their decision values always exceed those of the PSVs for a particular user.

Decision Value Adversary: To synthesize (generate) a victim's feature vectors, this adversary can use algorithm 1. This adversary would basically inject randomized feature vectors into the upstream at attack point 4 (fig. 1), and monitor the returned decision value at attack point 8.

Algorithm 1 starts with any feature vector (all zeros or extracted from a generic gesture by the attacker), randomizes the features, and tests the feature vector against the user profile to see if the current decision value is greater than the previous best decision value. If that is the case, the algorithm updates the feature vector to the new randomized one. This process continues until the algorithm reaches the target decision value V_{th} (set in our algorithm to be greater than +1 to obtain more representative samples of user behavior than PSVs).

The randomization of the feature vector is done by adding Gaussian noise to each feature independently. The additive noise has zero mean and a standard deviation, σ, controlled by the value $sRatio$ (line 7 in algorithm 1). σ is dependent on the features standard deviation, and we set it to 1 in our case, since that is the standard deviation of our standardized feature vectors. $sRatio \in (0, 1]$ is a design parameter that scales the value of σ; thus controlling the added noise which would affect how quickly and effectively the algorithm reaches its target. Setting $sRatio$ will be discussed in section V. Furthermore, to limit the amount of randomization as the algorithm approaches its target value, $sRatio$ is decreased after each successful update since it means that the algorithm is one step closer to reaching its target. Hence, the feature values do not need to be changed as much as the previous steps to reach its final target.

With this method, as many samples as needed can be created. Therefore, additional steps were taken to ensure having reconstructed raw data that can be useful for performing the attack (steps 13-16 in algorithm 1). The synthesized feature vector is passed into any of our two reconstruction algorithms (section IV). The resulting raw data would then have its features extracted, and tested against the user's profile. We only use reconstructed raw data that passes this test.

TABLE I: Similar and different features to $SysC$ [7]

	Similar	Different
$SysA1$ [4]	11	17
$SysA2$ [5]	15	0
$SysA3$ [6]	5	5
$SysA4$ [10]	7	30

Algorithm 1 Generate synthetic feature vectors for user u

1: **function genSyntheticFV**($model_u$, $initF$, V_{th}, $initR$, σ)
2: **do**
3: $sRatio = initR$;
4: $sF_u = initF$;
5: $decVal = testFV(model_u, sF_u)$;
6: **while** $decVal < V_{th}$ **do**
7: $rndF = sF_u + normRand(0, sRatio * \sigma)$;
8: $tmpDecVal = testFV(model_u, rndF)$;
9: **if** $tmpDecVal > decVal$ **then**
10: $sF_u = rndF$;
11: $decVal = tmpDecVal$;
12: $sRatio = 0.999 * sRatio$;
13: $recGesture = reconstructRawData(sF_u)$;
14: $recF = extractFeatures(recGesture)$;
15: $recDecVal = testFV(model_u, recF)$;
16: **while** $recDecVal < V_{th}$
17: return sF_u;

F. Choice of Systems and their features

Out of the five gesture-based AA systems ([4]–[7], [10]) that we selected in section II, we chose the features from Frank et al [7] to represent that of the compromised system (called hereinafter $SysC$). This system has achieved low error rates which might make it a good candidate for deployment. It was also previously chosen by other researchers when testing new idea or machine learning algorithms [12], [22], [23]. An important reason behind our choice was that they only had one feature each for both pressure and area (table II). This would enable us to study the effect of using the reconstructed raw data to attack other systems that have more reliance on pressure and area.

The remaining four: Serwadda et al [4], Antal et al [5], Li et al [6] and Xu et al [10] were selected to be the attacked systems, and will be called $SysA1$, $SysA2$, $SysA3$ and $SysA4$ respectively. In table I, we show the number of features of the attacked systems which are similar or different to the compromised system. Table II shows all five systems and the distribution of their features according to 8 categories. It can be seen from tables I and II that the selected attacked systems ([4]–[6], [10]) have varying levels of differences with our chosen compromised system [7]. These differences would enable us to study the effect of having different features on the proposed attacks. For example, $SysA1$ and $SysC$ are different in more features than they share. Moreover, $SysA4$ only shares 7 features with $SysC$, and differs in 30 features.

TABLE II: Distribution of features for different systems

	$SysC$	$SysA1$	$SysA2$	$SysA3$	$SysA4$
Location	4	4	4	2	6
Shape	9	3	7	4	13
Velocity	5	5	1	-	4
Acceleration	4	5	-	-	-
Pressure	1	5	1	-	5
Area	1	5	1	3	5
Time	2	1	1	1	3
Others	3	-	-	-	1

IV. The Reconstruction Approach

Raw data of each touch gesture stroke is composed of a series of vectors $(S_1, S_2, ..., S_N)$ with N being the number of touch events. Each touch event produces a vector, S_i, of the following values: location of the point (represented by X and Y coordinates), time stamp, pressure, area, phone and finger orientation [7]. The number of touch events (N) varies for different gesture strokes based on multiple factors including the length and duration of strokes and the type of the mobile device. Out of the raw data, feature extraction has to be done to create a feature vector. Normally, feature vectors do not contain any indication of the original number of touch events (N) from which it was extracted. Hence, in our algorithms, we allow passing this number as an argument, and we denote it as $length$ as shown in algorithms 2-4.

The chosen compromised system [7] has 34 elements in the feature vector. Out of these features, there are five that will not be used in the feature vector, and they are PhoneID, UserID, Document ID, direction flag, and direction of end-to-end line. The direction flag is used in our case as a classifier as we distinguish between left-to-right and right-to-left strokes; as opposed to [7] that treated both as a single classifier called horizontal strokes. The remaining 29 features from [7] are listed below (along with their abbreviated names that will be used in the algorithms below):

- Location and Shape: Start X and Start Y ($startP$), End X and End Y ($endP$), Direct End-to-End Distance, Mean Resultant Length, largest deviation from straight line (LD), 20%, 50% and 80% percentiles of deviation from end-to-end line ($prctD$), average direction, length of trajectory, and ratio of end-to-end distance.
- Velocity and Acceleration: 20%, 50% and 80% percentiles of pairwise velocity ($prctV$) and acceleration, median velocity at last 3 points, average velocity ($avgV$), median acceleration at first 5 points.
- Pressure: Mid-stroke pressure ($midP$).
- Area: Mid-stroke area covered ($midA$).
- Time stamps: Stroke Duration, and inter-stroke time.
- Others: phone Orientation (seems to be fixed at 1, and mid-stroke finger orientation and change of finger orientation (both seem to be fixed at 0).

In the listing above, the features with no abbreviations within the parentheses were not used in our numerical approach (section IV-A). Some features were not included for being redundant like the direct end-to-end distance that can be calculated from other features, or the percentiles of pairwise acceleration as we already use those of the velocity.

In the next two subsections, two approaches will be presented. The first would be based on starting with the features, and using numerical estimation and procedures to reconstruct the raw data (section IV-A). Whereas the second would be based on randomizing the raw data until we get a feature vector that is close enough to our target feature vector (section IV-B).

In both approaches, we start with a feature vector F, the required gesture events $length$, and we finally compute the reconstructed raw data matrix $recGesture$. For this to happen, the following four sub-tasks are needed:

- ReconstructXY: reconstruct X and Y coordinates based on location and shape features (sections IV-A2 and IV-B1).
- ReconstructTime: reconstruct time based on time, velocity and acceleration features (sections IV-A3 and IV-B2).
- ReconstructPA: reconstruct pressure and area based on pressure and area features (section IV-A4).
- ReconstructOthers: reconstruct other raw data like finger and phone orientation (section IV-A4).

A. The Numerical Approach

With this approach, we use the summary statistics contained in the feature vector to estimate the original raw data. This estimation is done using procedures which aim to generate raw data that has the same characteristics as the original, so that this reconstructed raw data could produce a feature vector that would resemble the user's behavior. This is particularly true for reconstructing time and location vectors. Since the feature vector does not include sufficient information for pressure and area (with one feature only each), we rely on the general population touch behavior to estimate the gesture's pressure and area data.

In the following subsections, we start by describing how to estimate values from percentiles (section IV-A1) which is needed for both $reconstructXY()$ and $reconstructTime()$ functions (sections IV-A2 and IV-A3). We finally describe reconstructing pressure, area and other values in section IV-A4.

1) Estimating values from percentiles: There are 3 sets of percentiles (largest deviation from straight line, velocity and acceleration) in the compromised system feature vector. These can provide information about the values from which these percentiles were calculated. Hence, a function is needed to estimate all the values from 3 percentiles only. For this purpose, we chose to estimate the points using shape-preserving piece-wise hermite cubic interpolation (PCHIP). It was found to perform better than other options like linear interpolation especially since linear interpolation cannot extrapolate the points outside its range. PCHIP also preserves monotonicity and the shape of the data [39]. Since we are using MATLAB, we use its implementation of this function (denoted as $estimateValues()$ in our algorithms). It takes the three percentiles and the required number of values as inputs, and returns the estimated values.

2) Reconstructing X and Y coordinates: We mainly utilize the deviation from straight line features to reconstruct X and Y. This is the perpendicular distance between the end-to-end straight line and the trajectory. It can have a positive or a

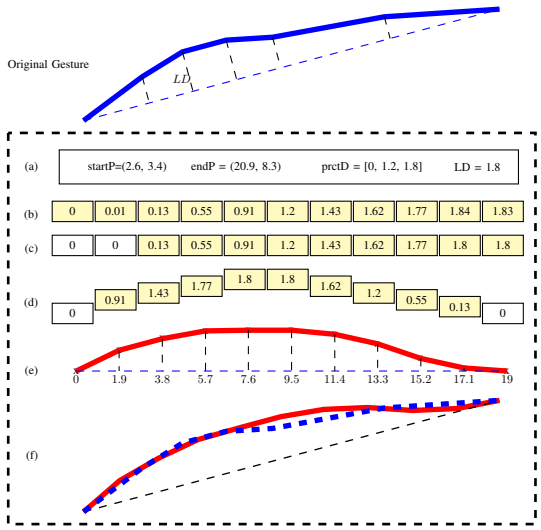


Fig. 3: Reconstruct X & Y

negative sign based on whether it is above or below the straight line. In fig. 3, the largest absolute deviation LD is shown for the gesture denoted "Original Gesture". In the same gesture, it can be seen that all deviation values are positive. In other gestures, they can be all negative, or a mix of both.

Observing the red gestures in e and f of fig. 3, it can be noticed that they have the same shape, and that one of them is the rotated version of the other. The deviation values would be the same for both, and they can represent the Y values for the horizontal version, e. Hence, we start with a horizontally laid gesture that starts at the origin $(0, 0)$. After reconstructing all the X and Y values for it, we rotate and shift it back to match the original gesture.

We demonstrate our algorithm using the graphical example shown in fig. 3: (a) we only start with the features shown in the fig. 3; (b) the function $estimateValues()$ (section IV-A1) is used to estimate the deviation values from the three deviation percentiles; (c) since the start and end points of any gesture have zero deviation, the two smallest absolute deviations are set to zeros. Also, any values exceeding the largest possible deviation are set to LD; (d) the values that were originally in ascending order have to be arranged so that the gesture would approximately have a bell shape to resemble normal gestures; (e) now that the deviations values are arranged appropriately, they need to be matched to X values. These are selected to be equidistant values from 0 to the total gesture straight line distance from $startP$ to $endP$; (f) finally, the horizontal gesture from (e) is rotated and shifted to match the original gesture based on its starting and ending points ($startP$ and $endP$) given in the feature vector.

3) Reconstructing time vector T: Having the pairwise distances between all points, and the pairwise velocities, makes it possible to calculate the pairwise timestamps. Algorithm 2 starts by computing the pairwise distances from the X and Y values previously generated. The pairwise velocities can be estimated using $estimateValues()$ but the returned values are sorted in an ascending order, and need to be organized according to velocity rate of change characteristics before being used to find the pairwise timestamps.

Unfortunately, features like the median acceleration at the first 5 points does not contain helpful information about velocities at the beginning, middle or end of the stroke. It would mean different thing if it were extracted from a 5-points gesture as opposed to a 50-points gesture. Hence, we rely on general population statistics (from public data sets) in the function $arrangeVelocities()$ to arrange the estimated pairwise velocities. Thus, having the first value become the smallest, followed by the rest of the values sorted in descending order.

In order to have the average velocity of the reconstructed raw data match that of the original feature vector, we adjust all the pairwise velocities in line 5 of algorithm 2. After that, calculating the pairwise time is followed by setting all the elements of the time vector T starting from zero with increments that equal the pairwise timestamps.

Algorithm 2 Reconstruct time vector T

1: **function reconstructTime**($prctV, avgV, X, Y, length$)
2: $distances = getPairwiseDistances(X, Y)$;
3: $valuesV = estimateValues(prctV, length - 1)$;
4: $valuesV = arrangeVelocities(valuesV)$;
5: $valuesV = valuesV + (avgV - mean(valuesV))$;
6: $pairwiseT = distances./valuesV$;
7: $T = setTime(pairwiseT)$;
8: $returnT$;

4) Reconstructing other values: Pressure and Area were grouped together in the function $reconstructPA()$ because we use similar methods to reconstruct them (as there is only one feature each in the compromised feature vector). These features are mid-stroke pressure $midP$ and area $midA$. Whereas we discuss the pressure data reconstruction here, the same discussion and operations are applicable to reconstructing the area raw data A from the mid-stroke area $midA$.

Since only the mid-stroke pressure $midP$ is available in the feature vector, and the user pressure behavior is unknown, we take a look at general population statistics to find out how gesture pressure changes. In fig. 4, we show some gesture pressure data. Since gesture strokes do not have a fixed number of points, the pressure data for all strokes was interpolated to have the same number of points, which was 11. The interpolated pressure values were then scaled from 0 to 10. Figure 4 shows the average overall relative pressure as well as the average of one of the users. It also shows two samples of gestures. It can be noticed that "Avg. Overall", "Avg. for User 4" and "sample 2" all have a similar rate of change relative to the pressure in their mid-points. Hence, we conclude that knowing the overall pressure rate of change would help find the other pressure values from $midP$. "Sample 1" on the other hand cannot be predicted using $midP$ but we can never accommodate all exceptions to the rule as this would require more data than a single point pressure value.

Therefore, we rely on the rate of change curve of "Avg. Overall" pressure in fig. 4, and scale it using the maximum and minimum possible values to be between 0 and 10 to obtain the vector ($avgPressV$). Having only one value ($midP$) the pressure vector P is found using the equation:

$$P(i) = \frac{midP}{avgPressV(\lfloor \frac{N}{2} \rfloor)} * avgPressV(i) \qquad (2)$$

where $i \in [1, 2, ..., N]$ and N is the gesture length.

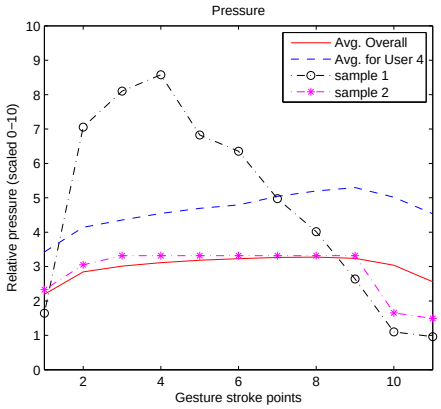


Fig. 4: Touch pressure

As for the other values in the raw data matrix, we noticed that they do not change. These include phone orientation and finger orientation. Therefore, the function $reconstructOthers()$ only returns vectors of the $length$ required that hold the values found in the feature vector F.

B. The Randomization Approach

Numerical algorithms have to be customized according to the features available in the compromised system. Hence, the previous algorithm was specific to one compromised system. In this section, we present an algorithm that can work with almost any compromised system without the need for customizing it by the adversary according to the available features. This stems from utilizing a small subset of features (e.g. starting and ending gesture points, or gesture duration) that we have found to be used in almost all systems due to having high importance when doing feature selection.

This approach aims to produce a feature vector that is as close to the original feature vector as possible. The main idea of the randomization approach is adding Gaussian noise to certain values in the raw data matrix (e.g. time vector T); extracting the feature vector from the randomized raw data; and checking if the new randomized values have made the feature vector closer to the original feature vector. If it did become closer, we would update the raw data with the randomized values and repeat the process until an upper limit of number of iterations is reached. This upper limit is found experimentally in section V-A, and it represents a number after which no significant improvement might happen.

Since changing one raw data value affects most of the extracted features, and might delay converging to the original feature vector, we opted to handle the problem with a divide and conquer approach. Therefore, we define three categories of features: (1) those affected only by X and Y coordinates; (2) those affected either only by time, or time and X and Y coordinates; and (3) the rest that has only one feature per raw data type like pressure and area. For the latter, we use the same set of functions that were described in section IV-A4 because randomization will not help with them. As for the first two categories, we process each one in a different stage. Features related to X and Y coordinates have to be optimized first before moving on to the time-related features. While velocity depends on X, Y, and timestamps, it is still possible to work on X and Y first to optimize the location-related features. Then one can move on to change the timestamps to optimize the velocity-related features, while having the X and Y coordinates fixed. In other compromised systems where feature vectors have rate of change features for pressure and area, the same approach can be used to optimize these features.

To evaluate how close two feature vectors are, we rely on the mean square error (MSE) between them. Of course, only the part of the feature vector related to the randomized values is tested. MSE is used to compare the normalized (or standardized) feature vectors to make sure that all features have similar value ranges and thus have equal importance. For this comparison, the function $testRawData()$ (please refer to algorithm 3) is used. It takes the new randomized values, the feature vector F and the indices of the relevant feature vector subset as inputs, and returns the MSE.

1) Reconstructing X and Y coordinates: Since the deviation from straight line defines the shape of the gesture (section IV-A2), we apply the Gaussian noise to these values (lines 6-9 in algorithm 3). This has to be preceded by shifting and rotating the gesture to be horizontal, and followed by rotating and shifting back. After that, the new randomized gesture is tested to see if it decreases the MSE (lines 10-14 in algorithm 3), and if it does, the Y vector is updated accordingly.

Algorithm 3 Reconstruct X and Y vectors randomly

```
1:  function reconstructRandXY(F, startP, endP, LD, length, iterLimit, indFeatXY, sRatio)
2:      [X, Y, score] = initalizeXY(F, length, indFeatXY);
3:      theta = getAngle(firstP, lastP);
4:      for i = 1 → iterLimit do
5:          [tmpX, tmpY] = rotateShiftXY(X, Y, −theta);
6:          for j = 2 → length − 1 do
7:              σ = |tmpY_j| ;
8:              tmpY_j = tmpY_j + normRand(0, sRatio ∗ σ);
9:          [tmpX, tmpY] = rotateShiftXY(tmpX, tmpY, theta);
10:         tmpScore = testRawData(tmpX, tmpY, F, indFeatXY);
11:         if tmpScore < score then
12:             score = tmpScore;
13:             Y = tmpY;
14:             sRatio = 0.999 ∗ sRatio;
15:     return [X, Y];
```

Algorithm 3 starts by initializing X and Y. X is initialized in the same way as in section IV-A2, and it does not change throughout the execution of this algorithm. whereas Y is initialized to one of eight gesture shapes depending on which one yields lower MSE ($score$). This step could minimize the amount of time needed to optimize the features. The eight gesture shapes are all scaled to the largest deviation (LD) with two only having positive deviations, two only having negative deviations, and four having both positive and negative deviations. This way, we can start with the gesture pattern closest to the original gesture.

The remaining part of the algorithm contains the randomization steps. Similar to algorithm 1, we use additive Gaussian noise with mean zero, and a standard deviation σ scaled by

$sRatio$ which is determined experimentally. However, in this algorithm, each point's additive noise has its own standard deviation, which equals that point's deviation from the straight line. Hence, points on the gesture edges that normally have less deviation would have less noise applied to them; thus maintaining its relatively less deviation compared to other points. This helps maintain an almost realistic gesture shape even after randomization, especially since the algorithm starts by initializing the gesture to one of eight realistic gesture shapes. Preserving such realistic gesture shapes would be desirable in case future AA systems flag unrealistic gesture shapes as being forgeries. Finally, similar to algorithm 1, $sRatio$ is reduced after each successful update (line 14 in algorithm 3).

2) Reconstructing time vector T: The algorithm starts by initializing the time vector T in a similar way to initializing Y in the previous section. Based on the total gesture duration, from the feature vector F, we set five variations of the time vector T that produce five different velocity curves, and then select the one that produces the least MSE. After that, this algorithm is similar to algorithm 3 described in the previous section except for the randomization part. Hence, algorithm 4 only describes the randomization of the time vector T.

In algorithm 4, all points, except for the first and last (that are previously set to zero and the gesture duration respectively), are randomized successively. To ensure that the new randomized values ($rndT$) for point j won't fall before the preceding timestamp ($tmpT_{j-1}$) or after the following timestamp ($tmpT_{j+1}$), and that enough room is left for randomization in subsequent iterations, we do not set the newly generated random timestamp unless its difference from the preceding or the following timestamps exceeds a required time buffer ($bufT = 0.1 * (duration/length)$).

Algorithm 4 Randomize time vector T

1: **function randomizeTime**($tmpT$, $length$, $sRatio$, $bufT$)
2: **for** $j = 2 \rightarrow length - 1$ **do**
3: $\sigma = min(tmpT_{j+1} - tmpT_j, tmpT_j - tmpT_{j-1})$;
4: $rndT = tmpT_j + normRand(0, sRatio * \sigma)$;
5: $tmpT_j = conditionalUpdate(tmpT_j, rndT, bufT)$;
6: return $tmpT$;

To limit the chance of having any new randomized value very close to either the preceding or the following timestamps, we generate the additive noise for every point using a different standard deviation value σ scaled by $sRatio$ (similar to algorithm 3). σ is set to be the smallest difference between the current timestamp and the preceding or following timestamps. By having this value, we limit the possibility of generating additive noise that would result in a value $rndT$ so close to the preceding or following timestamps. Thus, we ensure successful updates of the randomized time vector and possibly achieve better use of each randomization iteration.

V. Experimental Results and Discussion

A. Experimental Setup and Remarks

We used MATLAB and LIBSVM [40] to conduct the experiments. LIBSVM was chosen because it is more efficient than the MATLAB SVM functions. We used SVM with RBF kernel, and we tuned the parameters C and γ using five-fold cross validation. For prediction, we did not use the sign function where the threshold would be 0 to determine different classes. We change the threshold to find the equal error rate (EER) which is the value where both FAR and FRR are equal. This threshold is used throughout the experiments.

TABLE III: EERs of different attacked systems

	RTL	LTR
$SysA1$ [4]	7%	7.3%
$SysA2$ [5]	13%	10.2%
$SysA3$ [6]	14.6%	12.8%
$SysA4$ [10]	11.1%	8.9%

For full profile attacks, PSVs and NSVs are used to attack one of the four attacked systems. For each user's sample used (whether it was PSV or NSV), that sample is excluded from the data set before training user's models for the attacked systems. This is to avoid having biased results if that sample contributes to building the user model on the attacked system.

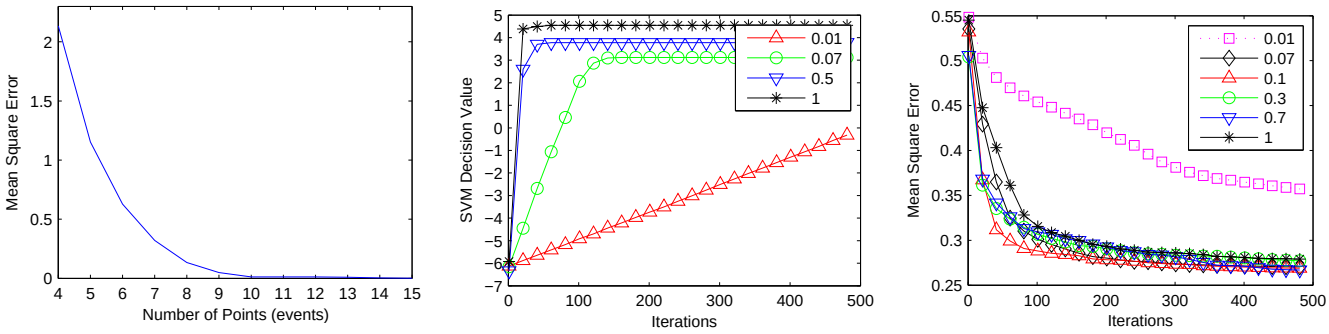
Because the reconstruction attacks given in the related work are in a different biometric research area (namely fingerprints), they are not directly comparable to our attacks and algorithms. Hence, our experiments only compare the attacks and algorithms we proposed, because our work is the first to address reconstruction attacks for mobile gestures.

Classifiers Selection: We used the data set provided by Frank et al [7] in our experiments. To test the attacks and our algorithms, we chose two classifiers: left-to-right (LTR) swipe, and right-to-left (RTL) swipe. We do not consider them to be one horizontal classifier since we believe that they each have distinct characteristics, and thus we classify them separately as was done in [4]. In the public data set we have, and for many users, one of the two vertical classifiers did not have enough samples to train an SVM classifier without overfitting. For this reason, and the fact that our algorithms are not dependent on gestures' directional angles (RTL and LTR are 180°apart, and our algorithms work successfully on both of them), the results are limited to LTR and RTL swipes. Finally, in table III, we report the EER values of the four attacked systems when using RTL or LTR classifiers.

Parameter Selection: We use data from a single user, chosen randomly, to perform parameter selection since the attacker will not have access to the original data of many users.

The parameter $length$ (number of reconstructed points) is used in all our reconstruction algorithms in section IV. We needed to find the smallest number of events that would preserve the characteristics of gestures. When testing $length = k$, (1) we took a gesture from the data set (say of $length = 25$); (2) we interpolated this gesture to only have a $length$ of k; then (3) interpolated the resulting reduced gesture to find the original 25 points; and (4) we finally measure the MSE between the original gesture, and the interpolated one. We did the previous steps for a number of gestures and aggregated the error. As shown in fig. 5a, the error could not be decreased after $k = 11$, thus we chose $length$ to equal that number.

Figure 5b shows the effect of choosing different values



(a) $length$ parameter

(b) $sRatio$ for Dec. Value Attack

(c) $sRatio$ for time randomization

Fig. 5: Parameter Selection

of $sRatio$ on the performance of the decision value attack algorithm (algorithm 1). In this case, we chose the target SVM decision value, V_{th}, to be 3 in order to show that we can obtain samples with much higher SVM decision values than PSVs. It can be noticed that setting $sRatio$ to be 1 would ensure the least number of iterations for this algorithm 1 (about 20).

For the randomized time reconstruction algorithm (algorithm 4), we also show the effect of using different $sRatio$ values on the performance of this algorithm (fig. 5c). We selected $sRatio$ to be 0.1 since it required the least number of iterations to reach the almost plateau stage of MSE. Since the plot is based on average MSE values, it is not possible to determine the plateau stage MSE value for each reconstructed gesture. Hence, we selected an upper limit of iterations as the stopping condition for this algorithm (which was 300 as can be seen from fig. 5c). As for the randomized coordinates reconstruction algorithm (algorithm 3), the plots were very similar, so we also selected $sRatio$ to be 0.1, and $iterLimit$ to be 300.

Negative Support Vectors' Results: It is worth noting that NSVs were not found for all users in the latter attack; however, they were only three users for RTL swipes and two for LTR swipes (out of 41 users). The results in the following section are reported excluding these users since we wanted to test how the algorithms would perform using NSVs regardless of the percentage of users that didn't have corresponding NSVs, which might vary from one compromised system to another.

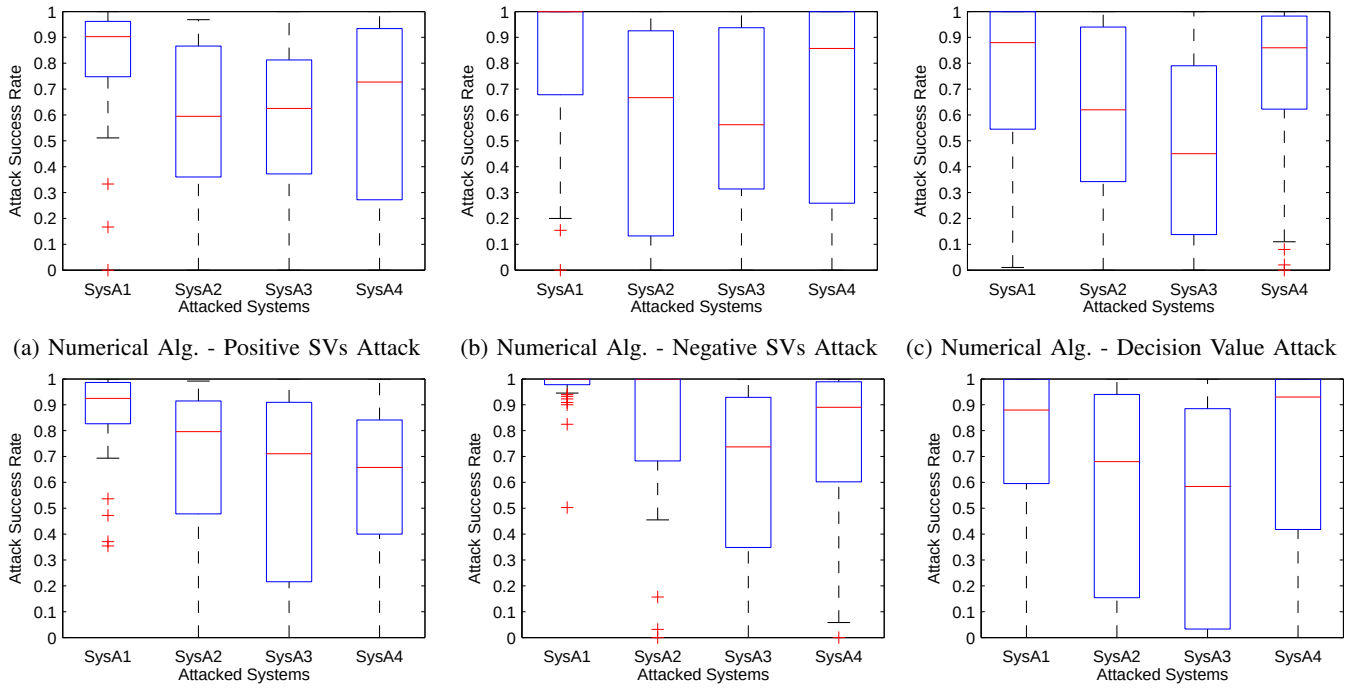
Attack Evaluation Metrics and Results Representation: To evaluate the attacks, we use the successfully reconstructed samples from the compromised system to attack four other systems. The choice of these systems was explained in section III-F. For user u with N_c^u reconstructed samples from system $SysC$, if N_{ai}^u samples were successful at attacking system $SysAi$, then the success rate would be equal to N_{ai}^u/N_c^u. Hence, the success rate value is always between 0 and 1. For the randomization algorithms, we ran each test ten times, and reported the average. To show the results of the different attacks with our algorithms, box plots of these success rates are used (figures 6 and 7). The box plots show the median success rate (center red line) and the 25^{th} and 75^{th} percentiles which are the edges of the box. The whiskers extend to the most extreme data points that are not considered outliers. The default MATLAB whisker spans 1.5 times the inter-percentile distances. Outlying values are represented by red plus signs.

B. Results and Analysis

Gesture stroke profiles (and our synthetic sample generation algorithm) provide a pool of feature vectors that can be used with the reconstruction algorithms. If the attempts to reconstruct any of these samples were successful, then an attack could be launched against the compromised system successfully. In our experiments, the reconstructed raw data is first tested against the compromised system, and it is only used for attacks if it succeeds at impersonating the legitimate user. Because both of our algorithms were successful at reconstructing a subset of gestures' raw data that would always pass the compromised system tests, results for success against the same system are not shown.

Results Overview: Figures 6 and 7 show attack results for LTR and RTL classifiers, respectively. Each of these figures contains six sub-figures that represent the combination of three types of attacks (positive SVs, negative SVs and decision value attacks) and two algorithms (numerical and randomization algorithms). Each sub-figure includes four box plots representing the four attacked systems: $SysA1 - SysA4$.

Impact of EER on Attack Success: In all types of attacks, it was clear that both our algorithms performed better against $SysA1$ compared to the other three. This was not initially expected as we thought that our algorithms would perform best against $SysA2$, since it uses a subset of the compromised system features. From observing the EER of both systems (table III), we noted that $SysA1$'s EER was lower than $SysA2$'s EER by 3% to 6% for both classifiers. Higher EER indicates higher FRR which means that more legitimate user's samples would be falsely rejected by the authentication system. This might lead to more reconstructed raw data being rejected by the system with the higher ERR, and it is possibly the reason behind having a lower attack success rate against $SysA2$ compared to $SysA1$. Moreover, the EER for $SysA1$ was also lower than both $SysA3$ and $SysA4$, and the attack success rate was better against $SysA1$. This coincides with the previous observation regarding $SysA1$ and $SysA2$. That being said, EER is probably not the only reason why our attacks did not perform so well against $SysA3$ and $SysA4$.



(a) Numerical Alg. - Positive SVs Attack (b) Numerical Alg. - Negative SVs Attack (c) Numerical Alg. - Decision Value Attack

(d) Randomized Alg. - Positive SVs Attack (e) Randomized Alg. - Negative SVs Attack (f) Randomized Alg. - Decision Value Attack

Fig. 6: Results for left-to-right swipe classifier (LTR)

Impact of Features' Differences on Attack Success: In the following discussion, we analyze how the attack success rates are affected by the proportion of similar features and the relation of features in compromised and attacked systems.

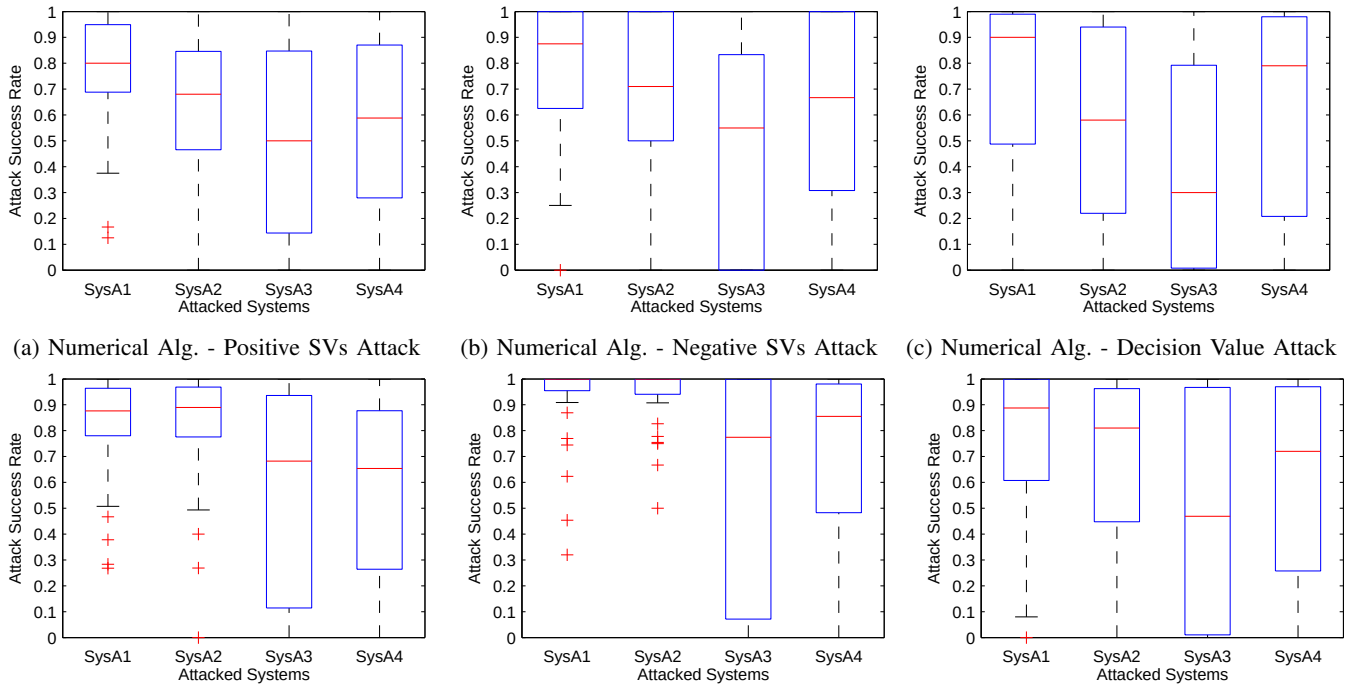
It can be seen from figures 6 and 7 that attacks against $SysA3$ [6] did not reach "conditional success" in 7 out of the 12 attack combinations. In comparison, attacks against other systems were all at least conditionally successful except for one case each for $SysA2$ and $SysA4$ (figs. 7c and 7a respectively). From tables I and II, it can be noticed that $SysC$ (the compromised system) only has one feature each for both pressure and area. Examining the 10 features used by $SysA3$, it can be noted that at least four of them cannot be deduced from $SysC$ features. These are the three area features and the average moving curvature. Such difference in the features, combined with $SysA3$ having the lowest EER, could have caused these low attack success rates. Intuitively, the relation between the features in the compromised and attacked systems would have an effect on the attack success rates.

Examining $SysA1$, which has less than 40% of its features similar to $SysC$ (as opposed to 50% for $SysA3$), it can be noted that there were 3 "successful" attacks against $SysA1$. The remaining 9 attacks were all on the high end of "conditional success". Hence, by comparing $SysA1$ to $SysA3$, we can deduce that it is not just the proportion of similar features that matters, but the possibility of deducing the attacked system features from the compromised system features. In the case of $SysA1$, most of its features could be deduced from $SysC$ using the reconstruction algorithms. The high success rates for $SysA1$ could also be attributed to its low EER.

On the other hand, $SysA4$ [10] had 2 "successful" attacks against it, 9 "conditionally successful" attacks, and one failed attack. The proportion of similar features with $SysC$ was only 19%, and this could have caused the lower attack success rates compared to $SysA1$. While $SysA4$ had features that cannot be deduced from $SysC$ features (e.g. the straight LDP-to-stop length), these features had lower importance in their respective feature vector (see [10] for features ranking). For this reason, and the lower EER of $SysA4$, the attacks against it were more successful than against $SysA3$ mentioned previously. Finally, the results of $SysA2$ were comparable to $SysA4$. However, these results are mostly tied to $SysA2$'s low EER rather than its features, since its features are a subset of those used in $SysC$.

Hence, we conclude that the attack success rates decrease if: (1) there are fewer similar (common) features between the compromised and attacked systems, (2) the similar features have lower importance (ranking) than differing features in their respective feature vector, and (3) fewer of the attacked system features can be deduced from the compromised system features.

Different Attacks Performance: Table IV shows that the NSV attack performed better than both the PSV and the decision value attacks. Furthermore, in comparing the results shown in figures 6 and 7, it is clear that using NSVs yielded noticeably better attack success rates than using PSVs to the point that the median attack success rates against $SysA1$ and $SysA2$ reached 100% for both LTR and RTL swipes using the randomization algorithm. This was expected since the NSVs found for a user (from other users' profiles) would normally have a higher decision value than the PSVs in her own profile.



(a) Numerical Alg. - Positive SVs Attack (b) Numerical Alg. - Negative SVs Attack (c) Numerical Alg. - Decision Value Attack

(d) Randomized Alg. - Positive SVs Attack (e) Randomized Alg. - Negative SVs Attack (f) Randomized Alg. - Decision Value Attack

Fig. 7: Results for right-to-left swipe classifier (RTL)

TABLE IV: Pairwise comparisons of attacks and algorithms using paired t-test. The alternative hypothesis was that the first of the compared pair has greater mean than the second. In all comparisons, the difference in mean was significant as evident by the p-values (much smaller than 0.05).

Compared Pair	Mean Difference	p-value
Randomized vs. Numerical	0.076	$3.1 * 10^{-15}$
Negative SV vs. Positive SV	0.066	$1.6 * 10^{-13}$
Negative SV vs. Decision Value	0.112	$2.6 * 10^{-16}$
Positive SV vs. Decision Value	0.046	$8.5 * 10^{-5}$

Finally, the results of the decision value attacks indicate that even without actual feature vector samples, it is feasible to launch successful attacks against many systems. In fact, the attacks achieved at least "conditional success" against three systems out of four using both reconstruction algorithms. The results against $SysA1$ were also in the lead with medians exceeding 88% for both algorithms and classifiers, while the results against $SysA4$ showed even better median attack success rates than the PSVs attack results against the same system.

Numerical vs. Randomization Algorithms: Considering reconstruction algorithms, it can be seen from table IV that the randomization approach performed better than the numerical. While the randomization algorithm was mainly designed to make it easier to utilize any compromised system without having to make changes to our reconstruction algorithm, it also had the added benefit of reconstructing raw data which can generate a feature vector that is very close to the target feature vector. Therefore, it can reach closer SVM decision values to those achieved using original feature vectors. On the other hand, the numerical algorithm does not use an iterative approach; nor does it refine the raw data. The numerical estimation is done by one-way functions, with no feedback on how well they performed. Hence, it is not always guaranteed that the reconstructed raw data would yield a sample that is the closest possible to the original feature vector.

TABLE V: Average reconstruction attack times per user

Attack Type	Time
Numerical - Positive SV	14.7 sec
Numerical - Negative SV	4.7 sec
Numerical - Decision Value	79.4 sec
Randomized - Positive SV	154.4 sec
Randomized - Negative SV	79.4 sec
Randomized - Decision Value	483.8 sec

Attacks Efficiency: The experiments were performed on a desktop computer with i5-3470 processor and 8GB of RAM. The average times required for the attacker to perform reconstruction attacks against one user are shown in table V (run using single-threaded MATLAB script). The PSVs attacks took longer than the NSVs attacks because the number of PSVs was greater than the number of NSVs for each user on average (38 and 11 respectively). All attacks had reasonable efficiency except for the randomized reconstruction algorithm with the decision value attack which took almost 8 minutes per user to generate ten user samples. This performance could be easily improved since the generation of these user samples can be done in parallel. Moreover, the randomized reconstruction algorithm can be modified to have its optimization target be the SVM decision value, thus eliminating the need for running

two consecutive randomized algorithms. This improvement is not our main target, and can be left for future work.

C. Discussion and Recommendations

From the analysis in the previous section, our observations can be summarized as follows:

First. Full profile attacks can yield high attack success rates, especially NSVs attacks. Hence, it is vital to avoid exposing user samples when building users' authentication profiles either by utilizing a machine learning algorithm that does not use them, or by using a privacy-preserving algorithm.

Second. Even in the absence of user samples, it is feasible to synthesize user samples using decision value attacks. This indicates that machine learning algorithms (like logistic regression) that do not store user samples, but return a probability, can also be susceptible to reconstruction attacks. Hence, it is recommended to only return binary classification results, and to rely on other methods to provide a trade-off between security and usability (other than using a threshold with the SVM decision value or logistic regression probability).

Third. It is not necessary for the attacker to spend a long time tailoring a reconstruction algorithm to a compromised system since a general randomization algorithm can yield high success rates. This shows that the AA system designer must not rely on choosing features that seem harder to reverse using numerical methods. It also further signifies the importance of using privacy-preserving (PP) techniques.

Fourth. Success rates are lower when the attacked systems share fewer features with the compromised system, especially if these features cannot be deduced from the compromised feature vectors. However, this should not deter the AA system designers from using PP techniques just because they think their feature vectors are so different from other systems, since this includes high uncertainty and great risk.

Fifth. Systems with lower EERs can be more susceptible to reconstruction attacks. This indicates that just because a system has a lower EER, it does not mean it is ready for deployment. In fact, it is more dangerous to deploy such a system before taking preventative measures against reconstruction attacks, than with a higher EER system.

1) Recommendations: Based on these observations, we provide our recommendations for building AA systems starting with modifications to the system model that was described in section III-B1 (fig. 1). The improved system architecture is shown in fig. 8, where the modifications are two-fold:

AA Server: To prevent full-profile attacks, it is imperative to use privacy-preserving machine learning techniques. This includes having "transformed" profiles (P_T in fig. 8) and a privacy-preserving matching algorithm (PP Matcher in fig. 8). Such transformation can be achieved using encryption [11] or reduced SVM [41], and the PP Matcher would be tied to the specific transformation performed. Unfortunately, such approaches come at the cost of decreasing the accuracy of the system. For example, the lowest reported EER in [11] was 18% when using the features from [7]. Nevertheless, it is still recommended to use such approaches, both to circumvent full-profile attacks, and to report the realistic readiness of such AA systems for deployment.

Client App and Gesture aggregation: To thwart decision value attacks, the decision value of testing a sample must not be returned. In fact, returning a binary result for each tested sample is not recommended as well. Instead, the client app should collect multiple gestures (represented by their raw data $\{R_1, \cdots, R_S\}$ in fig. 8, where S is the number of gestures), extract their individual feature vectors, and send them to the AA server ($\{F_1, \cdots, F_S\}$ in fig. 8). The *PP Matcher* can classify each sample F_i, and take a majority vote before returning a single binary result $binRes$ (which equals 1 for a legitimate user, and 0 for an intruder).

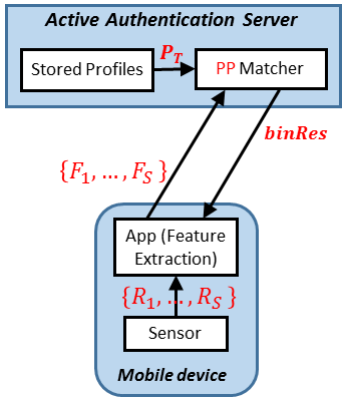


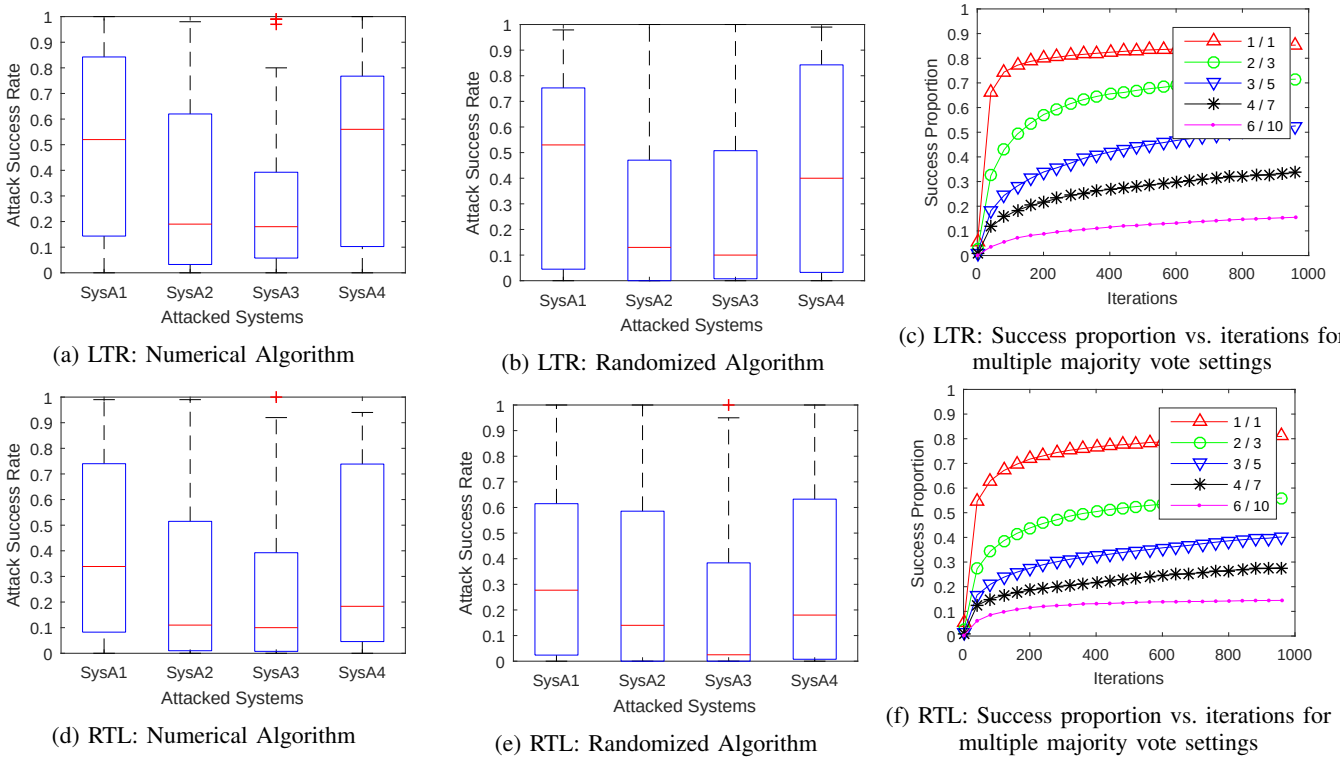
Fig. 8: System Architecture

This way, no individual sample can be tested by the adversary. Moreover, the possibility of finding a useful sample would decrease since the adversary has to come up with more than one successful synthetic sample through randomization (increased entropy). To avoid having the adversary send the same group of samples each time with one different sample to test that one specifically, the AA system can maintain an adequate history of submitted feature vectors to avoid repetition. It is worth noting that previous research (e.g., [7], [10]) has shown that aggregating multiple users' test samples for authentication proved to lower the EER of many systems. Hence, it comes with an additional advantage in this case.

To demonstrate the effectiveness of our recommendations, we conduct further experiments in the following section. Since the security and privacy of privacy-preserving machine learning algorithms were discussed in their respective work [11], we concentrate on the gesture aggregation advantages.

2) Binary and Aggregated Classification Results: We assume a ***binary result adversary (BR)*** that is similar to the decision value adversary described in section III-D. However, for BR adversary, only binary results are returned whether it was for a single sample or an aggregation of samples. To obtain a feature vector, the BR adversary uses algorithm 5 which is similar in concept to algorithm 1. However, since no decision value is returned, it is reduced to random guessing in each iteration.

Figure 9 shows the results for using algorithm 5. By comparing the binary result attack (figures 9a, 9b, 9d and 9e) and the decision value attack (figures 6c, 6f, 7c and 7f), we can see that returning a binary result lowers the attack success rates. In fact, the binary result attack mean was less than the



(a) LTR: Numerical Algorithm

(b) LTR: Randomized Algorithm

(c) LTR: Success proportion vs. iterations for multiple majority vote settings

(d) RTL: Numerical Algorithm

(e) RTL: Randomized Algorithm

(f) RTL: Success proportion vs. iterations for multiple majority vote settings

Fig. 9: Results of the binary result attack for left-to-right (LTR) and right-to-left (RTL) swipes

decision value attack mean by 0.27 (was found significant using paired t-test with $p - value = 2.2 * 10^{-16}$). It is also worth noting that none of the binary result attacks achieved "conditional success" against any system.

We show the benefit of aggregating gestures in figures 9c and 9f. These figures show the number of iterations required to generate synthetic feature vectors (using algorithm 5) that can pass as user samples. The y-axis in figures 9c and 9f represent the proportion of working synthetic samples to the total required samples at a specific iteration (we required 100 synthetic samples for each user to account for randomness). In figures 9c and 9f, we show different majority vote schemes and how they compare to each other (these curves do not include the iterations required for the reconstruction algorithms to represent the worst case scenario if the adversary had a perfect algorithm). From figures 9c and 9f, it can be seen that aggregating more gestures lowers the proportion of successfully synthesized feature vectors. For example, when the majority scheme is 6 out of 10 gestures (6 / 10), an adversary can only obtain 15% of the required samples even after 1000 iterations. In conclusion, more aggregated samples make it harder for the adversary to obtain synthetic samples, and even these generated ones cannot achieve attacks that are at least "conditionally successful".

VI. Conclusions

The aim of this work was to highlight the importance of carefully designed privacy-preserving AA systems. Otherwise, AA systems would fail at fulfilling their premise of providing security and privacy for mobile devices' users, especially while accessing cloud services. To show this, we proposed a numerical-based and a randomization-based reconstruction algorithms. These algorithms were utilized to reconstruct raw gesture data from users' authentication profiles (the full-profile attack), and from the decision value returned by SVM sample testing (the decision value attack). We further used such reconstructed raw data for attacking users' accounts on other AA systems. To test our algorithms, the literature was surveyed to select one compromised system and four attacked systems. The experimental results showed that reconstruction attacks against gesture-based AA systems are feasible, and that the most effective attack was using negative SVs. The most effective reconstruction algorithm was the randomization-based one. The results also showed that even when the adversary had no access to the users' samples, reconstruction attacks were still feasible against most systems. In our future work, we will pursue the problem of designing privacy-preserving systems that are resilient to reconstruction attacks.

Algorithm 5 Generate synthetic feature vectors for user u using binary results

function binGenSyntheticFV($model_u$, $initF$, $initR$)
  $binRes = testFV(model_u, initF)$;
  while $binRes \neq 1$ **do**
    $rndF = initF + normRand(0, initR)$;
    $binRes = testFV(model_u, rndF)$;
  return $rndF$;

Acknowledgment

The authors would like to thank Dr. Neil Gong for insightful discussion. This material is based on research sponsored in

part by the DARPA Active Authentication Program and the Brandeis Program under agreement numbers FA8750-12-2-0200 and N66001-15-C-4068, respectively [1].

REFERENCES

[1] "The u.s. mobile app report," White Paper, comScore, 2014, accessed 17-August-2015. [Online]. Available: http://www.comscore.com/Insights/Presentations-and-Whitepapers/2014/The-US-Mobile-App-Report

[2] "Update to celebrity photo investigation," Apple Media Advisory, Apple, 2014, accessed 17-August-2015. [Online]. Available: http://www.apple.com/pr/library/2014/09/02Apple-Media-Advisory.html

[3] M. Rose, "Think icloud's two-factor authentication protects your privacy? it doesn't," Report, Engadget, 2014, accessed 17-August-2015. [Online]. Available: http://www.engadget.com/2014/09/02/think-iclouds-two-factor-authentication-protects-your-privacy/

[4] A. Serwadda, V. V. Phoha, and Z. Wang, "Which verifiers work?: A benchmark evaluation of touch-based authentication algorithms," in *Biometrics: Theory, Applications and Systems (BTAS), 2013 IEEE Sixth International Conference on*. IEEE, 2013, pp. 1–8.

[5] M. Antal, Z. Bokor, and L. Z. Szabó, "Information revealed from scrolling interactions on mobile devices," *Pattern Recognition Letters*, vol. 56, pp. 7–13, 2015.

[6] L. Li, X. Zhao, and G. Xue, "Unobservable re-authentication for smartphones." in *NDSS*, 2013.

[7] M. Frank, R. Biedert, E.-D. Ma, I. Martinovic, and D. Song, "Touchalytics: On the applicability of touchscreen input as a behavioral biometric for continuous authentication," *Information Forensics and Security, IEEE Transactions on*, vol. 8, no. 1, pp. 136–148, 2013.

[8] T. Feng, X. Zhao, B. Carbunar, and W. Shi, "Continuous mobile authentication using virtual key typing biometrics," in *Trust, Security and Privacy in Computing and Communications (TrustCom), 2013 12th IEEE International Conference on*. IEEE, 2013, pp. 1547–1552.

[9] J. Zhu, P. Wu, X. Wang, and J. Zhang, "Sensec: Mobile security through passive sensing," in *Computing, Networking and Communications (ICNC), 2013 International Conference on*. IEEE, 2013, pp. 1128–1133.

[10] H. Xu, Y. Zhou, and M. R. Lyu, "Towards continuous and passive authentication via touch biometrics: An experimental study on smartphones," in *Symposium On Usable Privacy and Security, SOUPS*, vol. 14, 2014, pp. 187–198.

[11] J. Sedenka, S. Govindarajan, P. Gasti, and K. S. Balagani, "Secure outsourced biometric authentication with performance evaluation on smartphones," *Information Forensics and Security, IEEE Transactions on*, vol. 10, no. 2, pp. 384–396, 2015.

[12] N. Z. Gong, M. Payer, R. Moazzezi, and M. Frank, "Forgery-resistant touch-based authentication on mobile devices," in *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*. ACM, 2016, pp. 499–510.

[13] S. Li and A. C. Kot, "An improved scheme for full fingerprint reconstruction," *Information Forensics and Security, IEEE Transactions on*, vol. 7, no. 6, pp. 1906–1912, 2012.

[14] J. Feng and A. K. Jain, "Fingerprint reconstruction: from minutiae to phase," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 33, no. 2, pp. 209–223, 2011.

[15] A. Ross, J. Shah, and A. K. Jain, "From template to image: Reconstructing fingerprints from minutiae points," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 29, no. 4, pp. 544–560, 2007.

[16] R. Cappelli, D. Maio, A. Lumini, and D. Maltoni, "Fingerprint image reconstruction from standard templates," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 29, no. 9, pp. 1489–1503, 2007.

[17] X. Zhao, T. Feng, and W. Shi, "Continuous mobile authentication using a novel graphic touch gesture feature," in *Biometrics: Theory, Applications and Systems (BTAS), 2013 IEEE Sixth International Conference on*. IEEE, 2013, pp. 1–6.

[18] X. Zhao, T. Feng, W. Shi, I. Kakadiaris *et al.*, "Mobile user authentication using statistical touch dynamics images," *Information Forensics and Security, IEEE Transactions on*, vol. 9, no. 11, pp. 1780–1789, 2014.

[1]The views, opinions, and/or findings contained in this article/presentation are those of the author/presenter and should not be interpreted as representing the official views or policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the Department of Defense.

[19] A. Roy, T. Halevi, and N. Memon, "An hmm-based behavior modeling approach for continuous mobile authentication," in *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*. IEEE, 2014, pp. 3789–3793.

[20] M. Velten, P. Schneider, S. Wessel, and C. Eckert, "User identity verification based on touchscreen interaction analysis in web contexts," in *Information Security Practice and Experience*. Springer, 2015, pp. 268–282.

[21] Y. Meng, D. S. Wong, R. Schlegel *et al.*, "Touch gestures based biometric authentication scheme for touchscreen mobile phones," in *Information Security and Cryptology*. Springer, 2013, pp. 331–350.

[22] H. Zhang, V. M. Patel, M. Fathy, and R. Chellappa, "Touch gesture-based active user authentication using dictionaries," in *Applications of Computer Vision (WACV), 2015 IEEE Winter Conference on*. IEEE, 2015, pp. 207–214.

[23] H. Khan and U. Hengartner, "Towards application-centric implicit authentication on smartphones," in *Proceedings of the 15th Workshop on Mobile Computing Systems and Applications*. ACM, 2014, p. 10.

[24] T. Feng, Z. Liu, K.-A. Kwon, W. Shi, B. Carbunar, Y. Jiang, and N. K. Nguyen, "Continuous mobile authentication using touchscreen gestures," in *Homeland Security (HST), 2012 IEEE Conference on Technologies for*. IEEE, 2012, pp. 451–456.

[25] P. Saravanan, S. Clarke, D. H. P. Chau, and H. Zha, "Latentgesture: active user authentication through background touch analysis," in *Proceedings of the Second International Symposium of Chinese CHI*. ACM, 2014, pp. 110–113.

[26] H. Seo, E. Kim, and H. K. Kim, "A novel biometric identification based on a users input pattern analysis for intelligent mobile devices," *International Journal of Advanced Robotic Systems*, 2012.

[27] K. Zetter, "Sony got hacked hard: What we know and dont know so far," Article, Wired, 2014, accessed 17-August-2015. [Online]. Available: http://www.wired.com/2014/12/sony-hack-what-we-know/

[28] J. Bonneau, E. W. Felten, P. Mittal, and A. Narayanan, "Privacy concerns of implicit secondary factors for web authentication," in *SOUPS Workshop on Who are you*, 2014.

[29] M. Fredrikson, S. Jha, and T. Ristenpart, "Model inversion attacks that exploit confidence information and basic countermeasures," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2015, pp. 1322–1333.

[30] A. Serwadda and V. V. Phoha, "When kids' toys breach mobile phone security," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. ACM, 2013, pp. 599–610.

[31] M. Antal, L. Z. SZABÓ, and Z. Bokor, "Identity information revealed from mobile touch gestures," *Studia Universitatis Babes-Bolyai, Informatica*, vol. 59, 2014.

[32] V. N. Vapnik and V. Vapnik, *Statistical learning theory*. Wiley New York, 1998, vol. 1.

[33] K. Niinuma and A. K. Jain, "Continuous user authentication using temporal information," in *SPIE Defense, Security, and Sensing*. International Society for Optics and Photonics, 2010, pp. 76 670L–76 670L.

[34] K. B. Rasmussen, M. Roeschlin, I. Martinovic, and G. Tsudik, "Authentication using pulse-response biometrics." in *NDSS*, 2014.

[35] M. Kumar, "China software developer network (csdn) 5 million user data leaked," Report, 2011, accessed 30-April-2016. [Online]. Available: http://thehackernews.com/2011/12/china-software-developer-network-csdn-6.html

[36] N. K. Ratha, J. H. Connell, and R. M. Bolle, "An analysis of minutiae matching strength," in *Audio-and Video-Based Biometric Person Authentication*. Springer, 2001, pp. 223–228.

[37] L. Gomez, I. Neamtiu, T. Azim, and T. Millstein, "Reran: Timing-and touch-sensitive record and replay for android," in *Software Engineering (ICSE), 2013 35th International Conference on*. IEEE, 2013, pp. 72–81.

[38] "Decompile android apps," Apple Media Advisory, 2016, accessed 30-April-2016. [Online]. Available: http://www.decompileandroid.com/

[39] C. Moler, *Numerical Computing with MATLAB*. Society for Industrial and Applied Mathematics, 2004. [Online]. Available: https://books.google.com/books?id=-vPtcriflH0C

[40] C.-C. Chang and C.-J. Lin, "LIBSVM: A library for support vector machines," *ACM Transactions on Intelligent Systems and Technology*, vol. 2, pp. 27:1–27:27, 2011, software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

[41] O. L. Mangasarian and E. W. Wild, "Privacy-preserving classification of horizontally partitioned data via random kernels." in *DMIN*, 2008, pp. 473–479.

Mohammad Al-Rubaie is currently pursuing the Ph.D. degree in computer engineering at Iowa State University, Ames, Iowa. He received the B.Sc. and M.Sc. degrees from the University of Baghdad. His industry experience include working in Telecom companies as IN engineer and R&D engineer. He is interested in cyber security, privacy enhancing technologies and data mining, and his current research focus is continuous authentication and privacy-preserving machine learning.

Morris Chang is an associate professor at Iowa State University. He received the Ph.D. degree from the North Carolina State University. His past industrial experiences include positions at Texas Instruments, Microelectronic Center of North Carolina and AT&T Bell Labs. He received the University Excellence in Teaching Award at Illinois Institute of Technology in 1999. His research interests include: cyber security, wireless networks, and energy efficient computer systems. Currently, he is an editor of Journal of Microprocessors and Microsystems and the Associate Editor-in-Chief of IEEE IT Professional. He is a senior member of IEEE.