# Mobile Touchscreen Gestures Biometric with Physical Unclonable Function (PUF) Characteristics

Timothy M. Dee, Ryan A. Scheel, Nicholas Montelibano, and Akhilesh Tyagi

***Abstract*—We entrust our personal, social, financial profiles with mobile devices. This necessitates robust security and authentications mechanisms for mobile devices. Fortunately, mobile devices are relatively sensor-rich compared to traditional desktop computing systems. These sensors can be bootstrapped to construct innovative authentication mechanisms.**

**In this paper, device touchscreen characteristics are exploited to build a user-device (UD-) biometric physical unclonable function (PUF). When a human user interacts with a touchscreen, as in tracing a geometric shape, the system generates a stream of pressure values. These pressure values capture the dynamic capacitive differences induced by the finger's tracing motion. This difference in measured current values is a function of both (1) a human biometric of how a shape is traced and (2) silicon foundry process transistor-level variability embedded in the touchscreen grid. This forms a physical function with input $x$ defining a shape and output $y$ abstracted from the measured pressure value stream. We argue and establish that this physical function has PUF attributes. Moreover, it provides a robust user-device biometric based authentication mechanism.**

**Authentication is based on geometric shapes (challenges) drawn on the touchscreen. Users trace them. The authentication layer creates a response abstract, and validates it against a user profile. Authentication accuracy is affected by the complexity of geometric shapes as well as the validation algorithm. We consider poly-line shapes (simple gestures) and complex closed geometric shapes (complex gestures). Complex gestures offer higher response entropy, but are computationally less efficient with a slightly lower validation accuracy. $100\%$ accuracy is achieved on simple gestures. $99.6\%$ accuracy is achieved on complex gestures.**

**User profiles exhibit physical unclonable function (PUF) properties. Touchscreen gestures are quantized into binary strings. Gesture hamming distance is 60+ bits for 128-bit strings for different user-device profiles; it is 0 bits for the same profile. This demonstrates variability and reproducibility respectively. Montreal TESTU01 tests binary string pseudorandom characteristics; the majority of tests pass showing pseudorandom number generator (PRG) characteristics.**

CONTENTS



Electrical & Computer Engineering, Iowa State University, Ames, IA, USA. Emails: deetimothy33@gmail.com, scheel.ryan@gmail.com, tyagi@iastate.edu

## I. INTRODUCTION

Mobile devices are becoming the primary user interface terminals of the modern world with the computing servers infrastructure being pushed to the cloud. With wearable devices on the horizon, the number of mobile devices per capita is likely to explode.

Securing a mobile device is significantly more challenging. Physical possession allows for side-channel attacks. Authentication is more difficult since device connectivity is not always guaranteed due to mobility. Not being able to rely upon constant connectivity also reduces the options available to implement a mobile root-of-trust.

Biometrics enhance authentication; they provide a basis for trust. Both *what* data and *how* it is generated are measured. A profile captures these elements. Profiles must be reproducible

(invariable) – for the same user – and variable (unique) – for different users. Authentication compares new biometric data against a profile.

User mobile device interactions are rich with biometric data. We propose innovative methods for (1) profile construction and (2) authentication. Profiles are generated using a statistical concentration scheme. Input data maps into an evaluation point set; a set of these sets is a profile. Authentication compares an evaluation point set to a profile. Points within a statistical error band are considered authentic.

Authentication utilizes machine learning. Statistically concentrated profiles are training observations. Statistically concentrated input data is classified as "authentic" or "not authentic".

A Physical Unclonable Function (PUF) characterizes silicon. It is a mapping from challenges to responses. This mapping is unique per chip; it depends on variations in the silicon foundry manufacturing process. PUF is seen as a silicon biometric.

Touchscreen input is a composite user-device biometric. Gestures – a finger-traced path – are a user behavioral biometric. Current sensors at touchscreen edges detect gestures; they monitor current. Sensed current models both user level behavioral variation in touch pressure and in the current flow, delay, and transistor widths in silicon. Current sensors are seen as a PUF – mapping a challenge gesture to sensed current. This composite biometric is termed User Device-PUF (UD-PUF). Since the composition of the user behavior and silicon variability is physical at the sensed current level, a mathematical decomposition into the two components is highly unlikely.

Mobile device sensor-based authentication is available. Existing solutions share data sources – sensors, user input, touchscreen gestures – with this work. Other works also invoke machine learning for authentication. Compared to existing work, we achieve higher accuracy using less data. This ostensibly results from statistical concentration; profile variability and reproducibility are enhanced thereby.

**Contributions:**
**(1)** Profile Creation – An innovative statistical concentration scheme constructs constructs biometric profiles. Gestures provide user biometric information. Touchscreen sensors provide silicon biometric information. We generate a robust profile therefrom. **(2)** Authentication – Touchscreen gestures are compared to profiles. Machine learning and statistical concentration are utilized. **(3)** Profile PUF Properties – Authentication generates binary strings. These strings exhibit variability, reproducibility, and randomness characteristics. **(4)** We achieve higher accuracy using less data compared to extant solutions; fewer machine learning features – sensor data sources – and fewer training observations are necessary.

**Paper Organization:**
Section II presents related work in the area. An outline of a conceptual framework to use the UD-PUF for device authentication and other secure services is given in Section IV. Section V provides implementation details. Section VI presents UD-PUF variability characterization (VI-A,VI-B), reproducibility properties (VI-A), pseudo-random response characteristics (VI-B), and machine learning (VI-C) applications. Finally, Section VII offers insights on results.

## II. Related Work

### A. Biometric

Touchscreen based user identities have been created. Napa et al. [1] characterizes a user using finger movement in multi-finger gestures. The authors observed consistency in user profiles generated on different days. This suggests temporal robustness of UD-PUF identities is likely. The work shows the variability of touchscreen interactions enables user differentiation. However, features influenced by silicon variability are not used. We propose incorporation of silicon variability to enhance profile uniqueness.

A ranking of gesture-based authentication approaches [2] evaluates sample invariance, train-ability, adaptability, computational efficiency storage consciousness, configurability, and attack resistance are evaluated for each approach. These categories evaluate authentication approaches providing features for mobile deployment. Higher rankings imply greater mobile deployment potential. Support Vector Machine (SVM) and Geometric authentication rank highly in most categories. Both approaches are utilized in this paper.

Gesture-based authentication using machine learning [3] employ neural networks classifying 21 touch-generated features achieving $3\%$ average error rate. Continuous gesture-based authentication schemes employ touch features including finger pressure, trajectory, speed, and acceleration. [4] additionally uses a user-worn glove to collecting biometric hand movement data. This work achieves a False Accept Rate (FAR) of $4.66\%$ and False Reject Rate (FRR) of $0.13\%$. Secure mobile device unlocking [5] achieves an average equal error rate of $0.5\%$ using finger velocity, device acceleration, and stroke time as features. These existing gesture-based schemes require a large number of features. Machine learning based computation time typically has a non-linear relationship to the number of features. Thus, existing work is computationally intense and fails to provide perfect accuracy Our work achieves near perfect accuracy using fewer features at similar sample sizes; this likely results from the innovative statistical concentration scheme.

### B. Physical Unclonable Function (PUF)

Physical Unclonable Functions (PUFs) are like a silicon biometric – unique to each individual chip serving as an identity with a random distribution. The delay of a fabricated silicon transistor is a random function despite an identical mask level geometry due to random distribution of dopants and lithographic diffraction. These statistical variations are large enough to differentiate chips from the same wafer. Many different kinds of silicon PUFs have been proposed [6], [7], [8], [9], [10], [11], [12], [13] – ring oscillator, SRAM memory cell, arbiter, latch and flip-flop.

Such physical layer functions need both (1) per chip variability and (2) same chip reproducibility. The variability ensures that distinct devices produce different outputs on the same input. Reproducibility is needed for predictability and determinism in device authentication behavior.

These PUFs have been deployed in many applications [12], [11]. Their security has also been analyzed in many scenarios [14], [15].

A modern mobile device has an increasingly large number of sensors – mostly to introduce new UI paradigms. It is not uncommon to have accelerometer, gyroscope, LCD touchscreen, temperature and humidity sensor, barometer, RGB intensity sensor, proximity sensor, and gesture sensor. All these sensors are rooted in some kind of physical layer based on some analog activity. If these activities arise from some electrical phenomena based on capacitances, inductances and resistances, it is likely that the fabrication process has inherent randomized statistical distribution. Any of these activities or a combination thereof can constitute a viable atomic user-device identity.

Some work has started within the last few years along this line of thought. All of them seem to still be aiming to establish a unique PUF-like fingerprint for the device. A unified atomic user-device identity is not the goal. Dey et al. [16] use an accelerometer as the device fingerprint. The challenge is certain level signal applied to the vibrator, which activates the accelerometer to generate a response signature. Aysu et al. [17] use Micro Electro Mechanical Systems (MEMS) as a fingerprint PUF for low cost embedded solution in place of relatively more expensive ring oscillators or arbiters. More recently, Boneh's crypto group [18] has developed fingerprinting techniques for mobile devices. The approach measures properties of the device sensors by flipping over a mobile device. The measured response or fingerprint is the accelerometer activity. This could have elements of a combined user-device fingerprint - however, it is presented as a device fingerprint with the goal of masking out the user variance. Liu at al. [19] have a purported goal of creating a new UI by acknowledging and computing accelerometer activity driven gestures. Their hypothesis is that such gestures give many more degrees of freedom than the traditional gestures giving rise to richer user interfaces.

Limiting a service to a specific user may be advantageous; securing mobile device payment is one such service. SRAM PUFs are applied in this area to secure user credentials [20], [21]. However, mobile devices typically have one user account. Many apps and services store user credentials upon first entry. These SRAM PUFs do not provide protection in these cases. A mother may lend her device to a child to use an app; this should not empower the child to make purchases. A combined user-device identity enables differentiation of distinct users. The child is then distinct from the mother for purchasing purposes.

A resent survey discusses PUF usefulness in identifying mobile devices [22]; it includes methods, classification algorithms, and features of current applications. Touchscreen-based schemes are absent from this survey; this suggests the novelty of our application. *PUF use* and *multiple component incorporation* are presented as "Future trends and Research Opportunities"; our work accomplishes both.

No existing research efforts have goals identical to ours – that of establishing an atomic user-device biometric.

## III. User Device Entanglement

In today's mobile devices many sensors are explicitly designed to interact with the user. If these sensors exhibit their own semiconductor unique identities, they constitute a good initial candidate set for UD-PUF. Touchscreen is one such sensor set consisting of a wire grid connected to current sensors at the screen edge. These current sensors are subject to fabrication process variability. Therefore, they potentially have PUF variability and reproducibility properties. The challenge is how to extract such properties within the Android framework.

### A. Physical Unclonable Function (PUF)

A physical unclonable function (PUF) [12] provides a device chip-set biometric which can be wrapped into various cryptographic protocols. A typical PUF is based on the inherent variability in the silicon fabrication process. The dopant distribution and lithographic etching are examples of physical phenomena that lead to individualized energy-delay profile for each transistor. The same transistor in two different chips, even when fabricated on the same wafer, exhibits its individuality. Such biometric individuality can be captured as a challenge-response function block. Typical PUF implementations use a cascaded ring of inverters where these delay variations are amplified. Challenges result in a sampling of a specific bit in this ring oscillator. Another schema sends a 0 down one path and 1 down another. At each stage, the two bits carry on in straight paths or switch over their paths based on a challenge bit. At the last stage, an arbiter determines whether 0 or 1 bit won the race, which also becomes one bit of the response.

### B. Touchscreen

Capacitive and resistive touchscreens use an array of electronics capable of measuring touch-induced capacitive or resistive change. These CMOS transistors exhibit the same variability that has been exploited in the traditional silicon PUFs. In addition, there is sensing circuitry that detects the row and column number where the touch-induced capacitance or resistance change occurs. The sensing logic is replicated into a grid of touchscreen regions so that the events in multiple regions can be detected in parallel. The size of this grid keeps increasing to support fine-grained multiple gestures. This sensing logic is another source of variability in the touchscreen-based PUFs. It is plausible that the same silicon fabrication processes that lead to variability in the traditional silicon PUFs will induce similar variability in the measured current and voltages detecting touch in touchscreens.

The challenge can be a simple line segment drawn on the screen. The user is asked to trace it. Android framework generates a sequence of `MotionEvent` objects in response to such tracing. This `MotionEvent` class, among many other methods, includes a method `final float getPressure()` which returns a value between 0 and 1. On a capacitive touchscreen, it stands to reason that the device level variability must exist for the same reasons as for the classical silicon PUFs. The doping levels of capacitive permittivity layer will have the same statistical variation in the fabrication process. Similarly, the capacitance measurement circuits in these screens have the same statistical delay (or current level) variance as in silicon PUFs. The `getPressure()` method should reflect such per device variability in the pressure measurement.

The other desirable characteristic of pressure returned by the `getPressure()` method is that it entangles the user behavior

and the device biometrics in an atomic, inseparable manner. Part of the pressure reading is determined by the transistors and capacitors embedded within the touchscreen. Part of the pressure reading is influenced by the user behavior, how s/he traces a path: how much physical pressure, how closely is the challenge line segment followed in the response. There is no non-trivial deterministic quantitative model to separate the user pressure contribution from the device pressure contribution.

### C. Variability and Reproducibility of Touchscreen UD-PUF

Any PUF is designed to give a unique, variable response to a unique challenge. The variability aspect quantifies how different the responses are over two different challenges – we could even specify a metric capturing some kind of distance within the challenge space. For a UD-PUF $f$, one aspect of its variability for instance is captured by the following equation $\sum_{i,j,k} HD(f(C_i, U_j, D_k),\ \ f(C'_i, U'_j, D'_k))$. $f(C_i, U_j, D_k)$ captures the UD-PUF $f$ response on a challenge line segment $C_i$, traced by a user $U_j$, on device $D_k$. The function $HD()$ measures usual Hamming distance of two binary response strings. We expect *(different challenge, same user, same device)* scenarios $HD(f(C_i, U_j, D_k),$ $f(C'_i, U_j, D_k))$ to generate variability with nonzero response Hamming distances. We will similarly expect variability in *(same challenge, different user, same device)* $HD(f(C_i, U_j, D_k),$ $f(C_i, U'_j, D_k))$ and *(same challenge, same user, different device)* $HD(f(C_i, U_j, D_k),\ f(C_i, U_j, D'_k))$ scenarios. Question is how much variability is desirable? Should the average Hamming distance in $HD(f(C_i, U_j, D_k),\ \ f(C'_i, U_j, D_k))$ equation be half the response length $N$? We characterize this variability in this paper for touchscreen based UD-PUF. An interesting, somewhat unexpected, lesson learned is the effect of quantization hysteresis on the variability.

For the *(same challenge, same user, same device)* $HD(f(C_i, U_j, D_k),\ f(C_i, U_j, D_k))$ scenarios, we expect reproducibility. In other words, the response Hamming distance should be 0. This is more difficult than it seems. In the classical silicon PUF implementations, the reproducibility violations occur when chip's operating conditions such as temperature are different than its characterization operating conditions. With UD-PUF this is further complicated by the fact that the human user behavior – path tracing, is rarely identically reproduced. We characterize the reproducibility error rate for the proposed UD-PUF. We then introduce a statistical concentrator ECC (error correcting concentrator like an error correcting code in coding theory) to generate close to perfect reproducibility.

## IV. Model, Schema, and Mobile Secure Services

### A. Model

The UD-PUF consists of a characterization/profiling phase followed by the deployment phase.

During characterization, for a given user-device pair, a broad set of challenge-response pairs is collected – $CRSET(U_i, D_j) = \{(C_{i,j,k_0}, R_{i,j,k_0}), (C_{i,j,k_1}, R_{i,j,k_1}), \ldots, (C_{i,j,k_l}, R_{i,j,k_l})\}$.

Each challenge $C_{i,j,k_m}$ can be a line segment, a series of points specifying several connected line segments, a seed point $(x_i, y_i)$ and a curve generation schema such as the Poincare function or fractal curves. The main expectation is that a challenge be a path long enough to generate a response of desired length (128-256 bits for AES keys, 1024-2048 bits for RSA keys). It may also be feasible to allow the user to trace their own repeatable, personalized path such as a signature.

The proposed UD-PUF layer converts a given challenge $C_{i,j,k_l}$ into the corresponding response through a reproducible profiling phase - $R_{i,j,k_l} = UD - PUF_f(C_{i,j,k_l}, U_i, D_j)$.

### B. Schema

Who should be holding the pre-characterized challenge-response set $CRSET(U_i, D_j)$? This entity necessarily serves as a root-of-trust for all the transactions based on this set. Based on the applications that need to be supported, a third party certification authority could be such an entity. Google Wallet like applications can use such a scenario to verify the atomic user and device identity before providing services. The local root of trust within the device could be this entity to support UD-PUF based user-device unique signatures/passwords. Note that how to wrap the challenge-response set $CRSET(U_i, D_j)$ securely within such a root of trust, and the corresponding secure protocols for the communication between the cloud services layer, the mobile device, and a third party certification authority (CA) are not the focus of this paper.

### C. Mobile Secure Services

The scenarios plausible for UD-PUF based authentication include the following:

1) Cloud based secure services maintain a user-device domain oriented access control layer. These domains may or may not have partial orders defining access control. A third party CA may be needed as a root of trust to hold the challenge-response set $CRSET(U_i, D_j)$. Note that it may be feasible to wrap this third party CA into the secure services provider.
2) Replace the classical password based authentication with a UD-PUF challenge response mechanism to access the device or device-based apps.
3) UD-PUF could also be composed with the software-based cryptographic primitives such as pseudo-random number generator, encryption, and hash to make these primitives more robust.

## V. Implementation Framework

The user traces a challenge line; a set of points are produced. Two forms of challenge lines are provided (V-A). Produced points are `(x,y,pressure)` along the traced path. Point alignment (V-B) and quantization (V-C produce a response; this is then used in authentication (V-D).

Evaluation is conducted on 10 Nexus 7 devices with 3 distinct users. An Android app presents challenges and collects traces.

PUF applications usually have very large data sets. Data sets of such magnitude are made impractical due to user interaction;

this is a distinct type of PUF. The statistical significance of the results suggest the data size is sufficient.

### A. Challenge

Two challenge forms are used. Simple challenges are a sequence of line segments. Complex challenges are closed geometric shapes. Challenges are drawn to the screen. The response is the user's tracing thereof. Line segment and shape definitions are formalized below.

*1) Simple:* Simple challenges are a sequence of line segments; these line segments are created using Java libraries to generate 4 $(x, y)$ pairs connected into a convex path. The algorithm ensures all $(x, y)$ pairs are a minimum distance from each other. Seeding the random pair generator with an incrementing count ensure path reproducibility. The LHS figure in Figures 1 & 2 shows the generated path in dashed lines and the user traced path in solid lines for two different users and devices.

*2) Complex:* Complex challenges are closed geometric shapes. The user is presented with a regular polygon or circle. A starting point and direction is specified.

The shapes cover a large portion of the screen. A large number of pressure detectors – detecting screen-edge current – are involved in measurement; this ensures sufficient device contribution to the user-device profile.

### B. Point Alignment

When (same challenge, same user, same device) combination is expected to produce perfectly reproducible responses; otherwise authentication will reject a response close to the user profile. In silicon PUFs, false negatives – close enough response rejection – arises due to ambient condition variance. Ambient conditions may change between challenge generation and authentication. Variable temperature affects transistor delays which in turn affects PUF's responses.

In UD-PUF, a user's interaction, tracing a path with some pressure, is inherently more variable than reproducible. How do we generate consistent responses despite the presence of a human user?

We rely on statistical error correction for this. Just like in error correcting codes (ECC) in coding theory, a large sphere of codes adjacent to a codeword are reserved. Analogously we leave unused space – a collection of paths around each valid response. ECC force maps invalid codes to the closest valid code word. Corrupted responses are mapped to their center-of-gravity – the closest valid response. The downside of this approach is that the valid challenge/response space is reduced.

*1) MotionEvent Alignment:* Traced paths from (same challenge, same user, same device) should result in the same response; pressure detector measurement frequency inhibits this. Android events layer receives `MotionEvent` objects containing pressure, a proxy for screen-edge current. A response is a set of `MotionEvent` objects. Comparing responses is comparing sets of `MotionEvent` objects.

Ideally, `MotionEvent` sampling would occur at the position in both responses. The OS sampling is not ideal. How are comparison points between two traced paths derived?

Figure 3 shows a challenge path and two traced paths by the same user. The blue-point path sampled MotionEvent objects are misaligned with the red/orange-point path. Even the number of points sampled is different between the two paths. Comparing the two traced paths at the sampled points amplifies the pressure value differences since the sampled points are not aligned at the same $(x, y)$ coordinates.

Point alignment corrects for misaligned $(x, y)$ coordinates. A set of evaluation points – points along the challenge for comparing responses – are fixed. In Figure 3, the vertical dashed lines form these evaluation points.

Evaluation point $(x, y, pressure)$ values are derived from sampled points. Sampled points are: (1) Projected onto an axis. (2) Used to interpolate evaluation point values. Evaluation points are projected onto an axis for interpolation.

(1) Points are projected onto the vertical axis for challenge line segment slopes in the range $(-45°, 45°)$. The horizontal axis is used otherwise; this maximizes the projection resolution. (2) Interpolation uses and distance along the axis and rate of pressure value change between sampled points. In Figure 3, both the blue and red/orange paths will be evaluated at the 4 dashed vertical lines through interpolation. This will be a canonical representation for each traced path which is more likely to be reproducible.

*2) Statistical Concentration or Error Correction:* Profiling a (user, device) entails developing a challenge-response model. As shown in Figure 4, there is a challenge path, whose proxy is the dark blue line capturing the mean of the sampled data. We define an acceptable statistical band around it - green and red lines in Figure 4.

For these experiments, we limited ourselves to single line segments. The number of evaluation points defined on these line segments is 32. This results in 32-bit responses. This is easily extended to multi-line segment challenge paths yielding a greater number of response bits.

The user traces each challenge path 50 times. Hence, at each evaluation point $i$, we get 50 samples $p_{i,0}$, $p_{i,1}$, $\ldots$, $p_{i,49}$. Each evaluation point $i$ is modeled as a statistical acceptor or hypothesis $H_i$. These points are assumed to have normal distribution $\mathcal{N}(\mu, \sigma)$. We estimate this distribution with classical estimation theory as $H_i = \mathcal{N}(\mu_i, \sigma_i)$.

Once this model has been built through profiling, let us consider a user tracing a response to a challenge. At the $i$th evaluation point, let the interpolated pressure be $p_i$. By 3-sigma rule, the probability that the point $\mu - 3\sigma \leq p_i \leq \mu + 3\sigma \approx 0.997$ and $\mu - 2\sigma \leq p_i \leq \mu + 2\sigma \approx 0.95$.

Our statistical acceptor can set a range $[\mu - k\sigma, \mu + k\sigma]$. If $k$ is high, the band of influence in Figure 4 denoted by green and red lines is bigger. If it is too high, another user's $(U_j)$ response might be acceptable within the profile of user $U_i$. We have found $k = 2$ or the ECC band $[\mu - 2\sigma, \mu + 2\sigma]$ to be a good choice.

Figure 4 shows user $U_i$'s response in cyan (light) blue within the profile generated for user $U_i$. Note the response is fully contained within the acceptable band for each evaluation point. If $p_i$ at $i$th evaluation point is outside $[\mu - 2\sigma, \mu + 2\sigma]$ band, that point is rejected. If a point is accepted, the mean value $\mu_i$ at that evaluation point becomes the response.

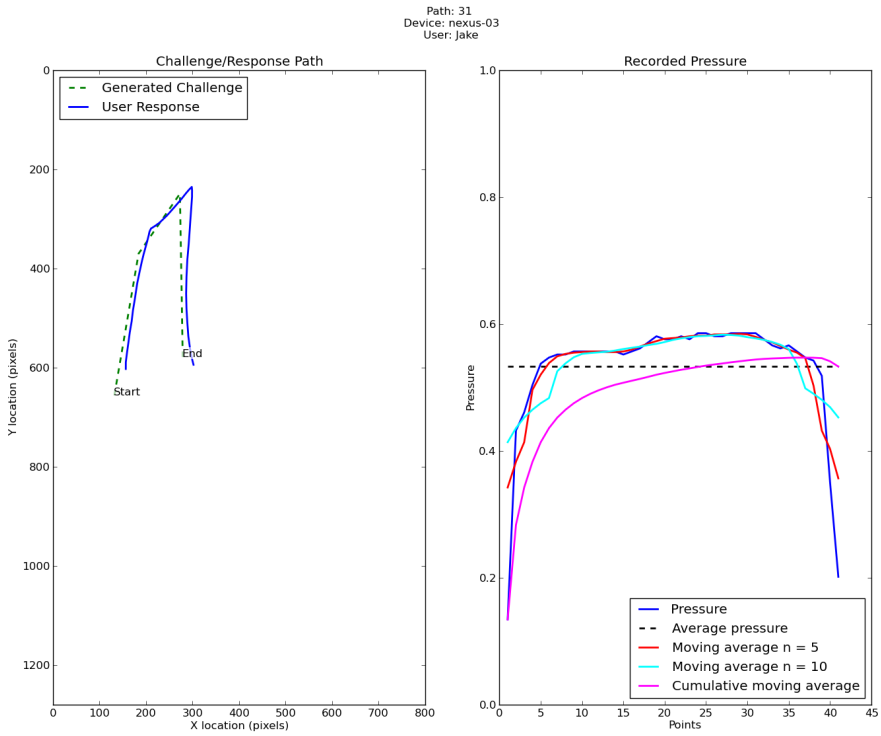


Fig. 1. Path Challenges and Pressure Responses

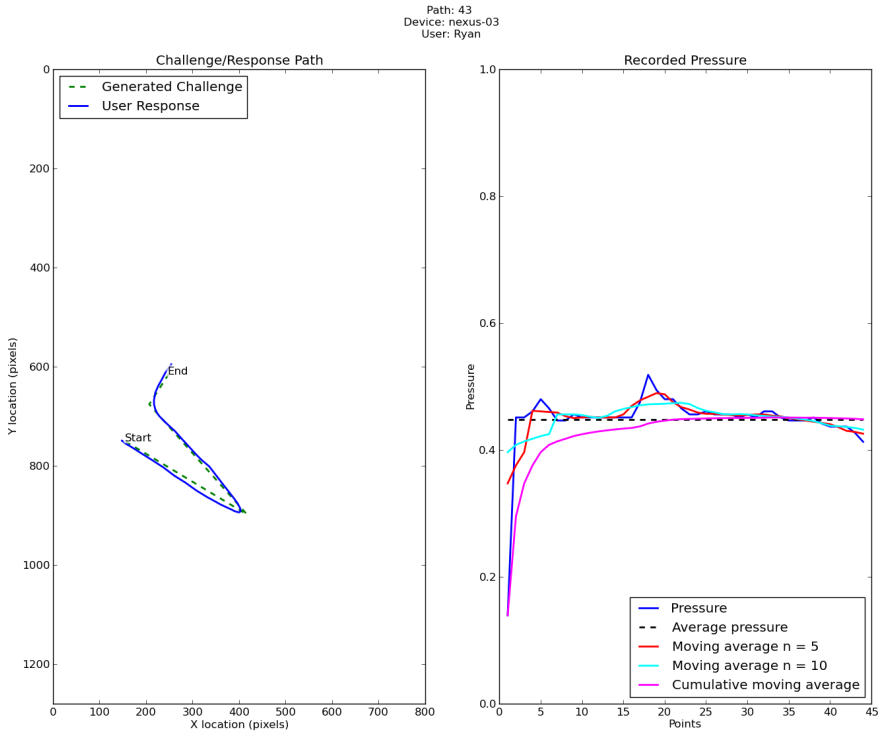


Fig. 2. Another Path Challenge and Pressure Responses

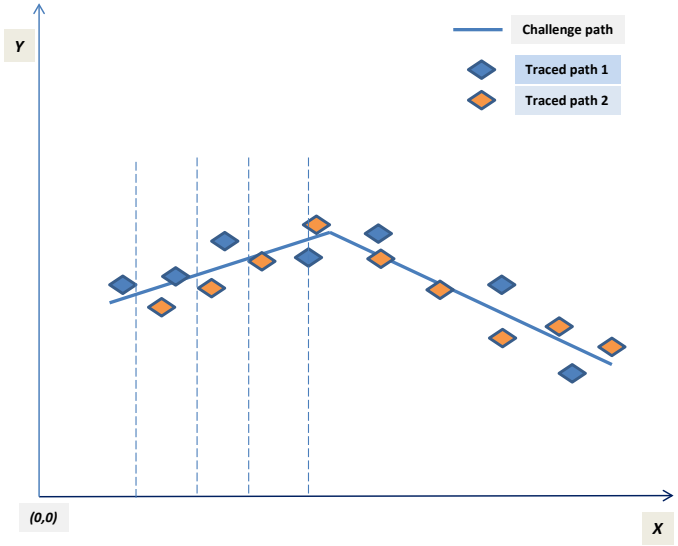


Fig. 3. Point Alignment for Two Separate Traced Paths

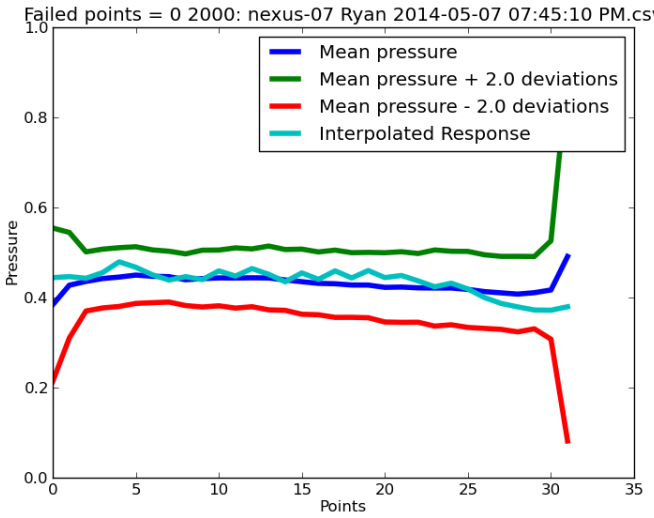


Fig. 4. Statistical Concentration/Correction for Profiled User

In summary, an interpolated, traced sequence of pressure values $p_0, p_1, \ldots, p_k$ is transformed into a canonical $\mu_0, \mu_1, \ldots, \mu_k$ pressure sequence if no points are rejected. This pressure sequence $\mu_0, \mu_1, \ldots, \mu_k$ will then be quantized using moving average $n = 5$ arbiter to generate the binary response.

*3) Neighborhood Decay:* Neighborhood decay is an alternative point alignment method. The first method is *MotionEvent Alignment* followed by *Statistical Concentration*. Sampled points are projected onto an axis; Evaluation point pressure values are interpolated therefrom. This first method performs poorly on complex shapes.

*Neighborhood Decay* is a point alignment mechanism. Evaluation points are equispaced along the challenge path. Evaluation point pressure is computed from $|S|$ neighborhood sampling point pressures; it is a weighted average. Exponentially less weight is given as euclidean distance increases between evaluation and sampled points. This method performs well on complex shapes.

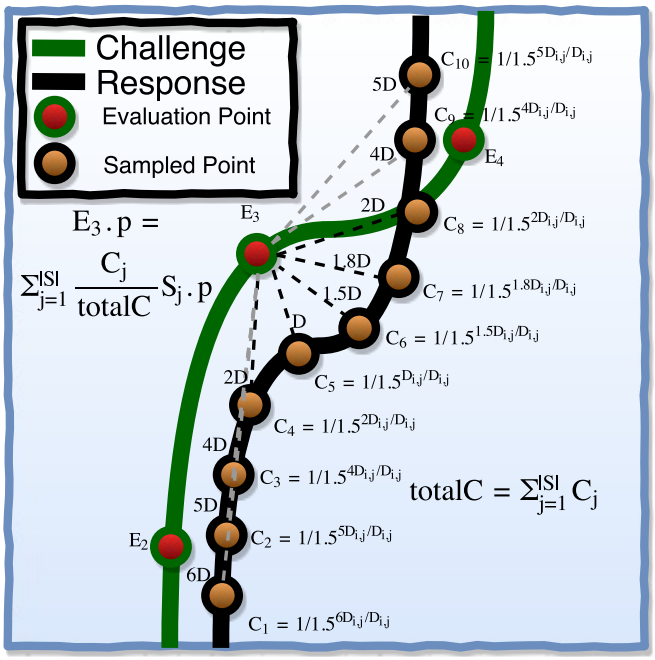


Fig. 5. Computation of evaluation point pressure estimate, $E_i.p$, under $|S|$-*neighborhood-decay* scheme. Evaluation point pressure is computed from $|S|$ neighborhood sampling point pressures. Exponentially less weight is given as euclidean distance increases.

Variations of *Neighborhood Decay* are described. Distance is defined as euclidean distance.

*a) 1-neighborhood-decay:* Evaluation point pressure value is set to the closest sampled point pressure value. If physically adjacent sensors share similar silicon behavior then the closest point provides the best estimate.

*b) 2-neighborhood-decay:* Extending the 1-neighborhood-decay scheme to consider the two closest sampled points to an evaluation point is 2-neighborhood-decay. This raises a new question. How to compute the evaluation point pressure value from two sampled points?

In this scheme, the two closest sampling points are given contribution weight, $C_j$, equal to $\frac{D_{i,j}}{D_i}$ where $D_i$ is the euclidean distance of the closest sampling point to evaluation point $i$ and $D_{i,j}$ is the euclidean distance from sampled point $j$ to the $i^{th}$ evaluation point. The pressure value of the $j_{th}$ sampled point is $S_j.p$. The pressure value of evaluation point, $E_i.p$, is assigned a weighted average pressure value of the two closest sampled points. The weight given to each sampled point is $\frac{C_j}{totalC}$. Where $totalC$ is the sum of contributions from all points.

$$E_i.p = \Sigma_{j=1}^{2} \frac{C_j}{totalC} * S_j.p \quad (1)$$

*c) $|S|$-neighborhood-decay:* $|S|$-neighborhood-decay extends the scheme to consider all sampled points. Evaluation point pressure is a weighted average of all sampled point pressures; these carry exponentially less weight as distance from the evaluation point increases.

Sampled points contribute to evaluation point pressure as a function of distance from the evaluation point. $C_j$ is

redefined as $\frac{1}{1.5^{(D_{i,j}/D_i)}}$. Sampled point pressure contribution at small distances have approaches $D_{i,j}/D_i$; contribution at large distance approaches 0. Evaluation point pressure ($E_i.p$) is computed as follows:

$$E_i.p = \Sigma_{j=1}^{|S|} \frac{C_j}{totalC} * S_j.p \quad (2)$$

Figure 5 provides an example of computing $E_i.p$.

1) Compute $C_j$ for all $S$
2) $totalC$ is computed as the sum of all contributions
3) $E_i.p$ is then estimated by computing a weighted average of $S_j.p$.

Distance of $S_j$ to $E_i$ are described in multiples of $D_i$ in the exponent; the pressure contribution from more points becomes larger as the closest point ($D_i$) becomes further away. If one sampled point is very close to an evaluation point then it receives the majority weight. If no sampling points are close to an evaluation point then contributions from sampling points in a larger neighborhood are significant; their distance is a smaller multiple of $D_i$; they are given increased weight.

### C. Quantization

The response generated by point alignment is a `MotionEvent` list; this corresponds to the evaluation points. Point pressure values are modified using statistical error correction. The result is a "center-of-gravity" response.

`MotionEvent` provides raw analog pressure values as a float between 0 and 1. Quantization converts them into a binary sequence. Each sampled path results in $150 - 400$ `MotionEvent` objects. The Android events layer chooses a sampling frequency on the basis of its own loading.

In Figures 1 & 2 RHS, the solid blue line shows the raw pressure data which is a jagged graph. We need to convert this pressure value sequence into a binary response sequence.

One approach is to generate a reference pressure line. Response pressure values are compared against this line; Pressure values above it are interpreted as a 1, values below as a 0. The choice of an arbiter reference line is what we call a quantizing mechanism. There are many possible approaches to quantizing this data.

The quantization process is summarized. (1) Generate a reference line. (2) Compare response pressure values to this line. (3) Output a binary sequence; each bit corresponding to one evaluation point.

*1) Moving Average:*

average based arbiter:

A simple approach is to take the average of all the pressure points. If a given path point's pressure exceeds this average, it is quantized as a binary 1, otherwise it is quantized as a binary 0. This has the advantage of resulting in responses that have roughly equal 0's and 1's - one of the required properties of a random string.

This quantization however has many runs of 0's and 1's since as seen in Figures 1 & 2 RHS, entire path segments are above or below the flat dashed blue pressure average line. Such 0 and 1 runs are bad if these responses are to be pseudorandom. As we discuss later, these strings fail 11 out of 26 PRG (pseudo-random generator) tests.

moving average $n = 5$ based arbiter:

An alternate is to create a low-pass filter by keeping an $n$-size moving window. A running average in this $n$-window is maintained. The quantization of the Point $p_m$ is done based on the $n = 5$ moving average $avg5_m = (p_{m-1} + p_{m-2} + p_{m-3} + p_{m-4} + p_{m-5})/5$ where $p_i$ is the pressure at Point $i$. If the pressure $p_m \geq avg5_m$ then it is quantized to 1, else it is quantized to 0. This arbiter passes almost all tests for PRG. In other words, a PRG based on this UD-PUF behaves like a cryptographic pseudo-random number generator. Note that this is a significantly stronger property than any of the silicon passive PUFs [6], [7], [8], [9], [10], [11], [12], [13] have been able to show. Most of the physical PUFs can best show a non-zero Hamming distance between two supposedly different responses. They have to use a cryptographic hash function such as SHA [23] on the PUF response to get any kind of pseudo-randomness. The red solid line in Figures 1 & 2 RHS shows the $n = 5$ quantization. Note that this line closely hugs the original raw data curve.

moving average $n = 10$ based arbiter:

This quantization is similar to the preceding $n = 5$ quantization except that it uses the moving average of the preceding 10 points. This $n = 10$ moving average is shown as green/blue solid line in Figures 1 & 2 RHS. It does worse than $n = 5$ quantization on PRG tests.

cumulative moving average based arbiter:

This is the extreme of the moving average with maximum dampening - the average of $p_0, p_1, \ldots, p_{m-1}$ is used to quantize $p_m$. This is shown as purple solid line in Figures 1 & 2 RHS. It has longer 0- and 1-runs than the $n = 5$ and $n = 10$ moving averages and hence does worst among the moving average quantization approaches on PRG tests. However, it still performs better than the flat average based quantization.

### D. Authentication

*1) Average and Standard Deviation:*

*a) Security Analysis:* Note that for normal distribution, the probability that a given point is within $2\sigma$ band is bounded by $Pr[\mu - 2\sigma \leq p \leq \mu + 2\sigma] = 0.95$ by 3-sigma rule. Similarly, $Pr[\mu - 3\sigma \leq p \leq \mu + 3\sigma] = 0.997$.

With $2\sigma$ method, the *false negative* probability that a valid response is rejected depends on the threshold $k$ selected in the authentication algorithm. When $0 \leq k \leq N$ points out of $N$ points from the response place outside the $2\sigma$ band of $\mu$, the response is rejected.

Let us consider an $N$ bit response. Each bit corresponds to a normal distribution. Assume that each bit is independent of the others. Let us start with $k = 1$, that is if even one point fails to place within the $2\sigma$ band, the user is not authenticated.

The probability that at least one pressure point out of $N$ points falls outside the acceptable $2\sigma$ band is *1 - Pr(all $N$ points place within $2\sigma$ band)*. This is given by $1-.95^N$. For $N = 32$, this false negative probability is 0.80629, which is quite high. If we set $k = 2$, the false negative probability that at least two points will fall outside $2\sigma$ band for a genuine user is *1- Pr(all $N$ points place within $2\sigma$ band) - Pr(Exactly one point places outside $2\sigma$ band)*. This is given by $1-.95^N - N*.05*.95^{N-1}$. For $N = 32$ with $k = 2$, this probability is .48004. Continuing this further, for $k = 3$, this probability is *1- Pr(all $N$ points place within $2\sigma$ band) - Pr(Exactly one point places outside $2\sigma$ band) - Pr(Exactly two points place outside $2\sigma$ band)*. This is $1 - .95^N - N * .05 * .95^{N-1} - \binom{N}{2} * .05^2 * .95^{N-2}$. For $N = 32$, this probability reduces to .21389.

Continuing in the same vein, if we go to $k = 4$, this probability rapidly reduces to $1 - .95^N - N * .05 * .95^{N-1} - \binom{N}{2} * .05^2 * .95^{N-2} - \binom{N}{3} * .05^3 * .95^{N-3} - \binom{N}{4} * .05^4 * .95^{N-4}$. For $N = 32$, this value is 0.020354, which is an acceptable value. However, if we go up to $k = N/4$, this expression reduces significantly to $1 - (\sum_{i=0}^{k} \binom{N}{i} * .05^i * .95^{N-i})$. For $N = 32$, $k = N/4 = 8$, this evaluates to $1.9112e^{-005}$. Overall, the choice of $k = N/4$ will give negligible probabilities for false-negative.

With $k = N/4$, in the false negative probability expression among the $k$ terms subtracted from 1 in $\sum_{i=0}^{k} \binom{N}{i} * .05^i * .95^{N-i}$, the $i = k = N/4$ term dominates. This term has the form $\binom{N}{N/4} * .05^{N/4} * .95^{3N/4}$. By Stirling's approximation for $N!$ given by $\sqrt{2\pi N}\left(\frac{N}{e}\right)^N$, $\binom{N}{N/4}$ can be simplified to approximately $\frac{(4/3)^{3N/4} * 4^{N/4}}{\sqrt{(3/8)*\pi*N}}$. When multiplied by the rest of the multiplicands, this leads to $\frac{.2^{N/4} * 1.2667^{3N/4}}{\sqrt{(3/8)*\pi*N}}$. This term asymptotically grows rapidly to subtract significant amount from 1 to lead to asymptotically 0 false negative probability.

An alternate mechanism to reduce the false negative probability would be to perform authentication a few times, say 3 times. If majority of authentication runs succeed, the response is authenticated. This is similar to triple modular redundancy (TMR) system. With .02 probability of false negative for $N = 32$ with $k = 4$, the TMR false negative probability is further reduced to $1 - .98^3 - 3 * .02 * .98^2$ which equals .00118, one order reduction.

*b) Authentication of User $U_j$ in User $U_i$ Profile:* Figure 6 shows the same data when user $U_i$ profile is used to evaluate a path generated by user $U_j$. Note that on average, in this scenario, 22 out of 32 points are rejected with acceptable band $[\mu - 2\sigma, \mu + 2\sigma]$. If we were to increase the acceptable band to $[\mu - 3\sigma, \mu + 3\sigma]$, the number of rejected points will decrease reducing the robustness of the reproducibility phase.

*2) Machine Learning:* Machine learning classifies responses into a profile – $(C_i, U_j, D_k)$. Point alignment is performed generating an evaluation point list. Each evaluation point is a classification feature. Each trace is a training data point with a corresponding profile classification.

A one-vs-one approach to classification is used. One classifier is trained for each pair of classifications. These classifiers distinguish between only that pair of classes. If there are $k$ classifications then there will be $k - 1$ classifiers trained for each classification. The total number of classifiers is $k(k-1)/2$, the number of unique pairs. Each classifier is applied to the evaluation points generated in a new response. The overall classification is the mode classifier classification.

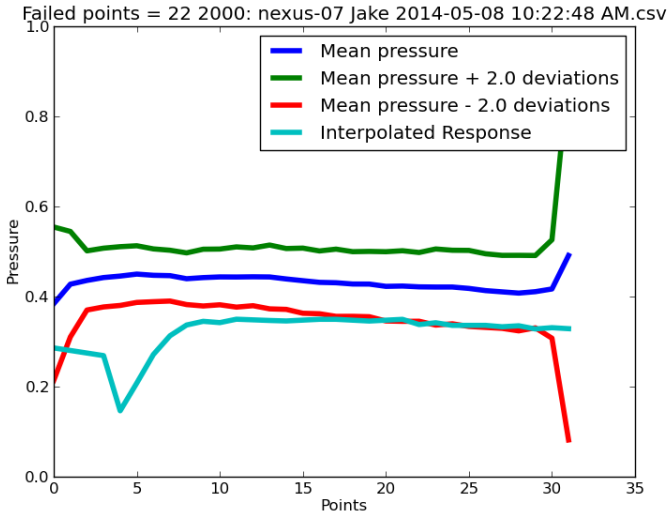


Fig. 6. Statistical Concentration/Correction for a User other than Profiled User

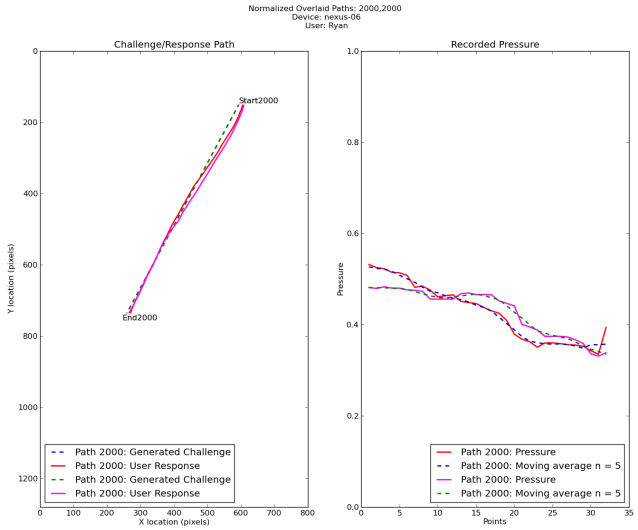


Fig. 7. Challenge Path and Two Traced Paths result in Hamming Distance 14 on 40-bit Responses

## VI. Results

UD-PUF variability, reproducibility, and application properties are demonstrated.

Variability is demonstrated by response hamming distances (VI-A) and pseudo-randomness (VI-B). hamming distance between same user-device and different user-device responses. Shown is large hamming distance between same user-device responses to different challenges and different user-device responses. UD-PUF response modeling would compromise security; pseudo-randomness implies this is not possible.

Reproducibility is demonstrated by response hamming distances (VI-A). Shown is small hamming distance between same user-device responses to same challenges.

Authentication is one application of UD-PUF; machine learning (VI-C) achieves this.

*a)* **Variability in UD-PUF Responses::** Recall that the (same user, same device, different path), (different user, same device, same path), and (same user, different device, same path) should result in different responses. How much variability exists in these scenarios?

There are two ways to evaluate this variability. In one case, we can just look at Hamming distance between the responses. In another, we can try to assess if the response outputs behave like a pseudorandom sequence. The second case endows more robustness and more desirable cryptographic properties on these PUFs.

*b) Initial Reproducibility Schema::* We evaluated reproducibility after incorporating point alignment canonical representation. As shown in Figure 7, for a challenge path, two traced responses are drawn. We would like the Hamming distance of these traced paths' pressure values after quantization to be low. Hamming distance varies with the quantization scheme. In this case, Hamming distance between the two path responses was 14 bits out of a 40-bit response with moving average $n = 5$ quantization. This Hamming distance reduces to 3 with cumulative $n = 40$ quantization. The point alignment by itself does not seem to be sufficient for reproducibility. Ideally, we would have liked to bring down the Hamming distance to be within 2-6 range, and then apply an error correcting code (ECC).

### A. Hamming Distance

*a)* **Hamming Distance between UD-PUF Responses::** We measured average Hamming distance between responses within the scenarios where variability is expected such as $\sum_{i,j,k} HD(f(C_i, U_j, D_k),\ f(C'_i, U'_j, D'_k))$. Table I shows our preliminary results when same path is used with different device/user scenarios. Due to symmetry of data, we have only shown an upper-triangular matrix. These are Hamming distance results when 128-bits responses are generated. Note that the average Hamming distance is close to 60 bits, almost half the string length. As a comparison, AEGIS arbiter PUFs [9] reported a Hamming distance of about 40 over 128-bit strings.

Table II reports the same data when quantization is performed with $n = 5$ moving average. Note that there are no appreciable differences between the flat average quantization of Table I and $n = 5$ moving average quantization of Table II. In fact, $n = 10$ moving average and cumulative moving average yield similar data and hence we do not report them.

Similar data capturing average Hamming distance over multiple challenge paths for many user/device combinations ((user, device, *) space) is presented in Tables III and IV for flat average quantization and $n = 5$ moving average quantization respectively. Note that the flat average quantization yields a lower variability for the different path, same device/user scenario than the $n = 5$ moving average quantization.

We show two of these different paths for the same user on different devices along with the corresponding pressure data in Figure 8. Note that the pressure differences exist along a significant fraction of the paths.

### B. Pseudo-Random Number Generator

*1) Test-U01:* An Ideal PUF will appear to generate its responses to successive challenges as if they came from a pseudo-random generator. If the PUF behavior is close to that of a PRG, an adversary is not likely to succeed in modeling it. This prevents a large class of attacks against a PUF. For classical silicon PUFs, such characterizations do not exist. Most silicon PUFs target sufficient Hamming distance separation between responses. If pseudo-randomness in responses is needed, the PUF responses are put through a hash function such as SHA-1.

Such a characterization is also ideal for small data sets. Note that out of $2^{128}$ possible responses, our experiments generated a very small subset. More precisely, if our challenge is 80 bits (10-bits for each $x$ or $y$ coordinate, and 4 such $x, y$ pairs). The overall challenge, response space is 208 bits with a 128-bit response. We only generated a few thousand challenge-response pairs. This is a very sparse data set. Estimating average Hamming distance of the underlying domain is a very difficult, and poorly understood problem. Predicting whether such a thousand long sequence came from a PRG has been studied more thoroughly. Simard et al. [24] have developed a TESTU01 suite for pseudo-randomness that puts the response sequences through a variety of linear-congruential PRG equivalence. It also checks for weaknesses such as long 0- and 1- runs.

We put our responses from all four quantization techniques (1) flat average quantization, (2) $n = 5$ moving average quantization, (3) $n = 10$ moving average quantization, and (4) cumulative moving average quantization through TESTU01 suite of tests. The following results indicate PRG effectiveness of these responses on a battery of 26 statistical tests enumerated in the following.

1. smultin_MultinomialBitsOver.
2. snpair_ClosePairsBitMatch in $t = 2$ dimensions.
3. snpair_ClosePairsBitMatch in $t = 4$ dimensions.
4. svaria_AppearanceSpacings.
5. scomp_LinearComp.
6. scomp_LempelZiv.
7. sspectral_Fourier1.
8. sspectral_Fourier3.
9. sstring_LongestHeadRun.
10. sstring_PeriodsInStrings.
11. sstring_HammingWeight with blocks of $L = 32$ bits.
12. sstring_HammingCorr with blocks of $L = 32$ bits.
13. sstring_HammingCorr with blocks of $L = 64$ bits.
14. sstring_HammingCorr with blocks of $L = 128$ bits.
15. sstring_HammingIndep with blocks of $L = 16$ bits.
16. sstring_HammingIndep with blocks of $L = 32$ bits.
17. sstring_HammingIndep with blocks of $L = 64$ bits.
18. sstring_AutoCor with a lag $d = 1$.
19. sstring_AutoCor with a lag $d = 2$.
20. sstring_Run. 21. smarsa_MatrixRank with 32x32 matrices.
22. smarsa_MatrixRank with 320x320 matrices.
23. smarsa_MatrixRank with 1024x1024 matrices.
24. swalk_RandomWalk1 with walks of length $L = 128$.

| | D/U 0 | D/U 1 | D/U 2 | D/U 3 | D/U 4 | D/U 5 | D/U 6 | D/U 7 | D/U 8 |
|---|---|---|---|---|---|---|---|---|---|
| D/U 0 | 0 | 61 | 61 | 67 | 64 | 67 | 63 | 66 | 67 |
| D/U 1 | | 0 | 65 | 56 | 63 | 60 | 63 | 64 | 62 |
| D/U 2 | | | 0 | 58 | 65 | 58 | 60 | 59 | 61 |
| D/U 3 | | | | 0 | 62 | 59 | 64 | 61 | 60 |
| D/U 4 | | | | | 0 | 63 | 63 | 64 | 65 |
| D/U 5 | | | | | | 0 | 65 | 60 | 56 |
| D/U 6 | | | | | | | 0 | 64 | 68 |
| D/U 7 | | | | | | | | 0 | 62 |
| D/U 8 | | | | | | | | | 0 |

TABLE I

HAMMING DISTANCE, SAME PATH, DIFFERENT DEVICE/USER, FLAT AVERAGE QUANTIZATION. D/U DENOTES A DEVICE/USER COMBINATION

| | D/U 0 | Dev/User 1 | D/U 2 | D/U 3 | D/U 4 | D/U 5 | D/U 6 | D/U 7 | D/U 8 |
|---|---|---|---|---|---|---|---|---|---|
| D/U 0 | 0 | 63 | 63 | 62 | 63 | 67 | 65 | 63 | 64 |
| D/U 1 | | 0 | 63 | 63 | 63 | 64 | 63 | 63 | 63 |
| D/U 2 | | | 0 | 63 | 62 | 64 | 63 | 62 | 64 |
| D/U 3 | | | | 0 | 63 | 65 | 64 | 61 | 64 |
| D/U 4 | | | | | 0 | 63 | 65 | 64 | 63 |
| D/U 5 | | | | | | 0 | 65 | 63 | 62 |
| D/U 6 | | | | | | | 0 | 65 | 63 |
| D/U 7 | | | | | | | | 0 | 64 |
| D/U 8 | | | | | | | | | 0 |

TABLE II

HAMMING DISTANCE, SAME PATH, DIFFERENT DEVICE/USER, $n = 5$ MOVING AVERAGE QUANTIZATION. D/U DENOTES A DEVICE/USER COMBINATION

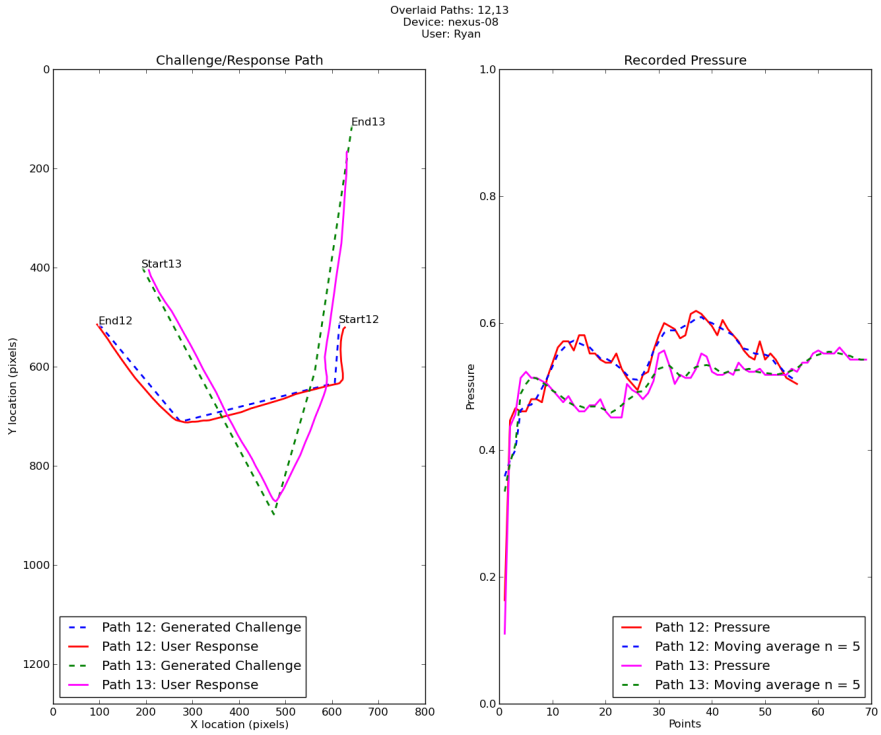


Fig. 8. Two Distinct Paths/Same User/Same Device and Pressure Responses

| D/U 0 | D/U 1 | D/U 2 | D/U 3 | D/U 4 |
|---|---|---|---|---|
| 63 | 59 | 56 | 41 | 61 |
| D/U 5 | D/U 6 | D/U 7 | D/U 8 | |
| 62 | 40 | 63 | 25 | |

TABLE III
AVERAGE HAMMING DISTANCE OVER DIFFERENT PATHS FOR SEVERAL DEVICE/USER COMBINATIONS, FLAT AVERAGE QUANTIZATION. D/U DENOTES A DEVICE/USER COMBINATION

| D/U 0 | D/U 1 | D/U 2 | D/U 3 | D/U 4 |
|---|---|---|---|---|
| 63 | 64 | 64 | 64 | 62 |
| D/U 5 | D/U 6 | D/U 7 | D/U 8 | |
| 63 | 64 | 63 | 63 | |

TABLE IV
AVERAGE HAMMING DISTANCE OVER DIFFERENT PATHS FOR SEVERAL DEVICE/USER COMBINATIONS, $n = 5$ MOVING AVERAGE QUANTIZATION. D/U DENOTES A DEVICE/USER COMBINATION

25. swalk_RandomWalk1 with walks of length $L = 1024$.
26. swalk_RandomWalk1 with walks of length $L = 10016$.

**(1) Flat average quantization:**

```
Version:  TestU01 1.2.3
Number of bits:   480
Number of statistics:  16
Total CPU time:   00:00:00.01
The following tests gave p-values outside
    [0.001, 0.9990]:
(eps  means a value < 1.0e-300):
(eps1 means a value < 1.0e-15):

     Test                              p-value
----------------------------------------------
 1  MultinomialBitsOver             4.2e-104
 2  ClosePairsBitMatch, t = 2        1.3e-6
 4  AppearanceSpacings              1 - 4.1e-10
 6  LempelZiv                       1 - eps1
12  HammingCorr, L = 32               eps
13  HammingCorr, L = 64               eps
14  HammingCorr, L = 128              eps
18  AutoCor                         1 - eps1
19  AutoCor                         1 - eps1
20  Run of bits                       eps
20  Run of bits                       eps
----------------------------------------------
All other tests were passed
```

**(2) $n = 5$ moving average quantization:**

```
Version: TestU01 1.2.3
Number of statistics:  16
Total CPU time:   00:00:00.01
The following tests gave p-values outside
       [0.001, 0.9990]:
(eps  means a value < 1.0e-300):
(eps1 means a value < 1.0e-15):

      Test                             p-value
----------------------------------------------
 1  MultinomialBitsOver              2.8e-12
 6  LempelZiv                         0.9996
19  AutoCor                           3.2e-6
20  Run of bits                       3.9e-9
----------------------------------------------
All other tests were passed
```

**(3) $n = 10$ moving average quantization:**

```
Version: TestU01 1.2.3
Number of bits:   480
Number of statistics:  16
Total CPU time:   00:00:00.01
The following tests gave p-values outside
     [0.001, 0.9990]:
(eps  means a value < 1.0e-300):
(eps1 means a value < 1.0e-15):

      Test                             p-value
----------------------------------------------
 1  MultinomialBitsOver              3.4e-38
 6  LempelZiv                        1 -  2.9e-6
18  AutoCor                          1 - eps1
20  Run of bits                        eps
20  Run of bits                        eps
----------------------------------------------
All other tests were passed
```

**(4) cumulative moving average quantization:**

```
Version: TestU01 1.2.3
 Number of bits:   480
 Number of statistics:  16
 Total CPU time:   00:00:00.01
 The following tests gave p-values outside
     [0.001, 0.9990]:
 (eps  means a value < 1.0e-300):
 (eps1 means a value < 1.0e-15):

       Test                            p-value
-----------------------------------------------
 1  MultinomialBitsOver             3.2e-121
 4  AppearanceSpacings               1 -  3.1e-9
 6  LempelZiv                        1 - eps1
12  HammingCorr, L = 32                eps
13  HammingCorr, L = 64                eps
14  HammingCorr, L = 128               eps
18  AutoCor                          1 - eps1
19  AutoCor                          1 - eps1
20  Run of bits                        eps
20  Run of bits                        eps
-----------------------------------------------
All other tests were passed
```

Note that out of twenty-six PRG tests applied in the TestU01 battery, $n = 5$ moving average quantization fails only 4 categories, $n = 10$ moving average quantization fails 5, cumulative moving average quantization fails 10, and flat average quantization fails 11. We plan to use $n = 5$ moving average as our quantization technique on this basis. The $n = 5$ moving average quantization consistently fails *AutoCorr*, *MultinomialBitsOver* , and *Run Of Bits*. It fails *LempelZiv* occasionally. This is fairly good performance as PRG.

### C. Machine Learning

An evaluation point list is classified into a profile – $(C_i, U_j, D_k)$. Each evaluation point list is an observation. Each evaluation point is a feature.

A point alignment scheme in Section V-B generates evaluation points. The results presented in this section fix 32 such points. 32 is chosen for two reasons. First, the best empirical accuracies are achieved with 32 or more points. Second, the evaluation points are given as features to the machine learning algorithm. Each observation must possess the same number of features. Increasing the number of features increases the computation time; minimizing them is advantageous.

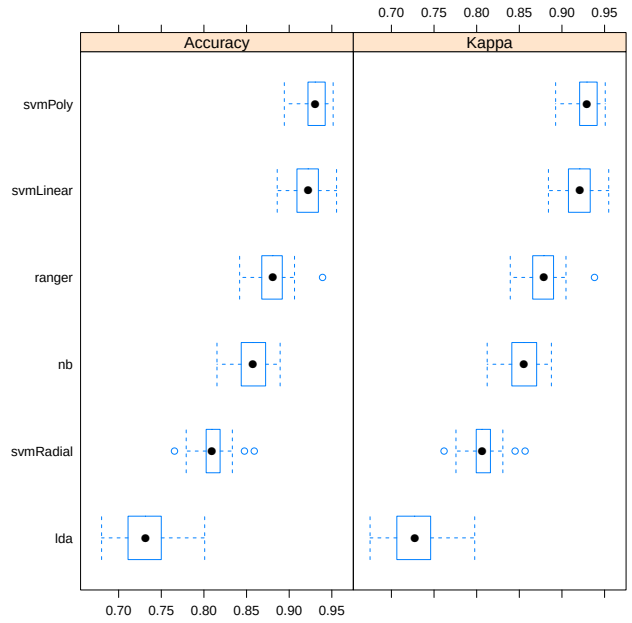


Fig. 9. Accuracy achieved using machine learning algorithms applied to `complex` challenges. Accuracy describes the rate of correct classifications from preforming repeated $k$-fold cross validation on the data with $k = 10$ repeated 3 times. The kappa statistic is measure of how much better each classifier performs compared to how a random classifier would be expected to preform. The black circle indicates the mean. The box shows the boundaries of the 25th and 75th percentile. The dashed lines indicate the inter quartile range (1.5 * the height of the box). The inter quartile range is where data is expected to fall. Data outside of this range are considered outliers and marked with a circle.

```
Accuracy
          svmLinear svmRadial svmPoly   ranger    nb
svmLinear            0.112683 -0.006023  0.041331  0.066179
svmRadial < 2.2e-16           -0.118706 -0.071351 -0.046504
svmPoly   1         < 2.2e-16            0.047354  0.072202
ranger    1.046e-09 5.104e-13 5.397e-11             0.024847
nb        2.309e-13 1.117e-08 1.075e-13 8.081e-05

Kappa
          svmLinear svmRadial svmPoly   ranger    nb
svmLinear            0.114477 -0.006118  0.041997  0.067237
svmRadial < 2.2e-16           -0.120595 -0.072481 -0.047241
svmPoly   1         < 2.2e-16            0.048115  0.073355
ranger    1.043e-09 5.060e-13 5.366e-11             0.025240
nb        2.316e-13 1.117e-08 1.071e-13 8.071e-05
```

TABLE V
STATISTICAL SIGNIFICANCE OF CLASSIFICATION ACCURACY DIFFERENCES APPLIED TO `complex` CHALLENGES. THE VALUES ON THE UPPER DIAGONAL ARE ESTIMATES OF THE DIFFERENCE BETWEEN THE MODELS. THE VALUES ON THE LOWER DIAGONAL ARE P-VALUES FOR A TEST WITH NULL HYPOTHESIS $difference = 0$. THE DIFFERENCE BETWEEN SVMLINEAR AND SVMPOLY IS NOT STATISTICALLY SIGNIFICANT AT SIGNIFICANCE LEVEL 0.05.

*1) Complex Challenges:* Machine learning algorithms including support vector machine (svm) with linear, polynomial, and radial kernels, random forest, naive Bayes, and linear discriminant analysis are applied. Figure 9 provides accuracy results. The labels correspond to the previous list of algorithms with svmLinear, svmPoly, svmRadial, ranger, nb, and lda respectively. Accuracy describes the rate of correct classifications from preforming repeated $k$-fold cross validation on the data with $k = 10$ repeated 3 times.

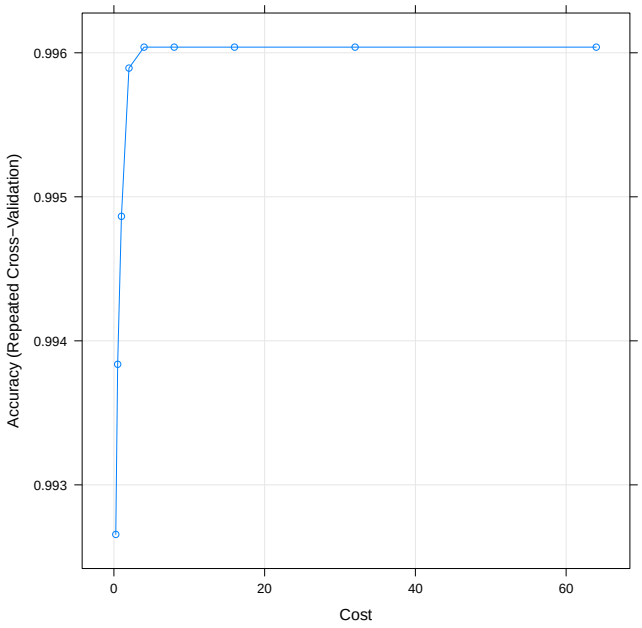


Fig. 10. SVMLinear tuning parameters are explored. Prediction accuracy and classification time depend on $cost$. Accuracy reaches maximum at $cost = 16$. In our experiments, increasing cost from 1 to 16 results in a $0.5ms$ increase in prediction time from $22.8ms$ to $23.3ms$.

$k$-fold cross validation estimates the performance of an algorithm on observations which it has not seen. For $N$ observations, $k$-fold cross validation leaves $N/k$ observations out when training the model. It can then be seen if the model is able to predict the classification of these $N/k$ observations correctly. For $k = 10$ this process will be repeated for 10 non-overlapping subsets of observations.

The kappa statistic compares a classifier's accuracy to a random classifier's accuracy; it is the improvement over random class selection.

$$kappa = \frac{observed - expected}{1 - expected} \tag{3}$$

$observed$ is classifier prediction accuracy. $expected$ is expected random classifier accuracy. Kappa describes how well the classifier performs on a scale from the accuracy achieved by random chance to $100\%$ accuracy. A kappa of $90\%$ means the classifier achieved $\frac{9}{10}$ the distance between expected accuracy and $100\%$ accuracy.

SVM with linear kernel provides the best result. In Figure 9 svm with polynomial kernel attains the greatest accuracy. However, linear kernel has advantages over polynomial kernel: (1) Polynomial kernel computation time exceeds linear kernel computation time. (2) Accuracy differences are not statistically significant.

Statistical significance of results in Figure 9 are given in Table V. The test null hypothesis is $difference = 0$ – there is no accuracy difference. Difference p-values are 1 for accuracy and kappa. We fail to reject the null hypothesis – no accuracy difference – at a significance level of 0.05. Accuracy difference

| Classifier | Prediction Time(ms) |
|---|---|
| svmLinear | 23.3 |
| svmRadial | 38.6 |
| svmPoly | 29.05 |
| ranger | 0.75 |
| nb | 232.25 |
| lda | 0.15 |

TABLE VI
CLASSIFICATION TIME PER SAMPLE AVERAGED OVER 20 SAMPLES.

between linear and polynomial kernels is not statistically significant.

SVM with linear kernel provides the highest accuracy and low classification time.

Classification time is estimated in R. Table VI-C1 gives the classification time per sample averaged over 20 samples. Svm with linear kernel predicts the fastest among algorithms with high accuracy.

*a) SVM Insight:* An insightful formulation of SVM with linear kernel can be found in [25]. SVM with linear kernel can be written as:

$$\begin{aligned} f(x) &= \beta_0 + \Sigma_{i=1}^{n} \alpha_i K(x, x_i) \\ K(x, x_i) &=< x, x_i >= \Sigma_{j=1}^{p} x_j x_{ij} \end{aligned} \tag{4}$$

Where $\beta_0$ is the intercept, $n$ is the number of observations, $x_i$ is the $i^{th}$ observation of $x$, $p$ is the number of features of $x$, and $K(x, x_i)$ is the kernel. Equation 4 gives the formulation of the linear kernel on line 2. The estimated parameters are the $\alpha_i$'s and $\beta_0$. According to [25] it can be shown that only the support vectors have nonzero $\alpha_i$. Therefore Equation 4 can be reduced to:

$$f(x) = \beta_0 + \Sigma_{i=1}^{S} \alpha_i < x, x_i > \tag{5}$$

Where $S$ is the set of support vectors. The support vectors are only those points modifying the separating hyperplane position. The size of set $S$ is likely less than the total number of observations, $n$. Due to the simplicity of the kernel and the above simplification in estimating the $\alpha_i$s, SVM with linear kernel is much simpler to compute compared to alternative machine learning algorithms; it achieves similar prediction performance at lower cost.

*b) Parameter Tuning:* *cost* is a linear kernel SVM parameter; it corresponds to the number of support vectors. Achieving the accuracy in Figure 9 requires tuning *cost*. Figure VI-C displays tuning results.

Maximum accuracy is achieved at large *cost*. Large *cost* increases support vectors [25]; this increases computation time (Equation 5). Lower values of cost are preferred. The first maximum, $cost = 16$, in Figure VI-C is selected.

This approach achieves $99.6\%$ accuracy classifying $(C_i, U_j, D_k)$ given a sampled point list. False accept rate (FAR) and false reject rate (FRR) are $.01\%$ or lower for $87.5\%$ of classifications; they never exceed $4\%$ for any classification. Further, such accuracies are achievable with relatively few observations – 25 in this case.

*2) Simple Challenges:* Classification is performed on simple challenges. SVM with linear kernel is superior for the same reasons delineated for complex challenges: (1) Classifier accuracy differences do not reach statistical significance. (2) Computation time is minimized.

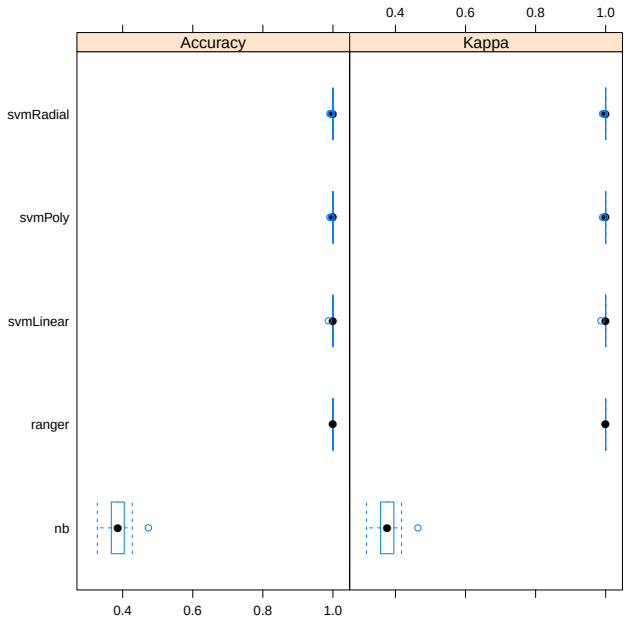


Fig. 11. The plot above presents the best accuracy achieved with several different machine learning algorithms applied to `simple` challenges. Many algorithms preform with near perfect accuracy. The difference in prediction among these algorithms is not statistically significant at a significance level of .05.

Figure 11 displays the results of applying svmLinear, svmPoly, svmRadial, ranger, and nb to sets of evaluation points generated by simple challenges. Similar accuracy is achieved for svmLinear, svmPoly, svmRadial, and ranger. The accuracy differences between these top-performing algorithms are not statistically significant at a significance level of .05. For the computation performance advantages discussed in Subsubsection VI-C1, SVM with linear kernel is chosen as best for application to simple challenge shapes.

Near perfect, $100\%$, accuracy is achievable with this approach; such are achieved with relatively few observations – 25 in this case.

*a) Parameter Tuning:* Tuning of the *cost* model parameter is preformed again for SVM with linear kernel applied to simple challenge shapes. The results of this tuning are presented in Figure 12. The chart shows near perfect accuracies for every cost value. The ideal *cost* value according to these results is then 1.

## VII. CONCLUSIONS

Delivering cloud-based services in a user-device differentiated domain is a challenging problem. Luckily, a mobile device is more tightly coupled to its user than the traditional computing model. Biometric data rich touchscreen interactions provide the foundation of a user-device identity. We profile the impact of touchscreen gestures on current sensors at the screen edge. These sensors exhibit PUF properties of variability and reproducibility. A novel authentication scheme

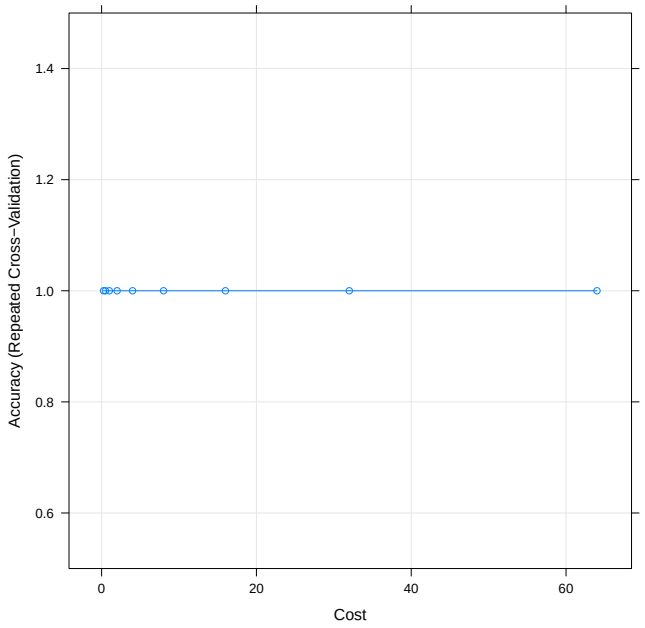


Fig. 12. Tuning parameter values are explored when applied to `simple` challenges. Near perfect accuracies are achieved at every cost value. $cost = 1$ minimizes computation time.

validates simple and complex gestures against these profiles. Two validation methods including a statistical concentration scheme and machine learning achieve near perfect accuracy. Further, profile PUF properties are demonstrated.

UD-PUF captures a biometric of both the silicon on the device and the native user behavior. For UD-PUF to be effective, it should exhibit high variability in its response over any of user, device, or challenge axes. Challenge is a path drawn on the touchscreen. We have established high degree of variability in responses. For a change in challenge, device, or user, 60+ bits change in a 128-bit response. $n$-moving average based arbiter is needed to provide dampening to generate high variability.

A profile – same challenge, same user, same device – response must be reproducible. Deploying a statistical concentrator/acceptor/ECC scheme achieves this. Perfect differentiation between users is shown when a traced path (response) is evaluated within a given user's profile.

Complex shapes require more sophisticated point alignment. A scheme were sampled points are weighed based on their distance from the challenge is given.

Machine learning classification differentiates between (user, device, challenge) tuples. Simple challenge response traces are differentiated with accuracy $100\%$. Complex challenge response traces are differentiated with accuracy greater than $99.6\%$.

UD-PUF viability as a mobile device biometric authentication mechanism is established.

## REFERENCES


[1] Napa Sae-Bae, N. Memon, K. Isbister, and K. Ahmed. Multitouch gesture-based authentication. *Information Forensics and Security, IEEE Transactions on*, 9(4):568–582, April 2014.

[2] Gradeigh D Clark and Janne Lindqvist. Engineering gesture-based authentication systems. *IEEE Pervasive Computing*, 14(1):18–25, 2015.

[3] Yuxin Meng, Duncan S Wong, Roman Schlegel, et al. Touch gestures based biometric authentication scheme for touchscreen mobile phones. In *International Conference on Information Security and Cryptology*, pages 331–350. Springer, 2012.

[4] Tao Feng, Ziyi Liu, Kyeong-An Kwon, Weidong Shi, Bogdan Carbunar, Yifei Jiang, and Ngac Ky Nguyen. Continuous mobile authentication using touchscreen gestures. In *Homeland Security (HST), 2012 IEEE Conference on Technologies for*, pages 451–456. IEEE, 2012.

[5] Muhammad Shahzad, Alex X Liu, and Arjmand Samuel. Secure unlocking of mobile touch screen devices by simple gestures: you can see it but you can not do it. In *Proceedings of the 19th annual international conference on Mobile computing & networking*, pages 39–50. ACM, 2013.

[6] G. Edward Suh and Srinivas Devadas. Physical unclonable functions for device authentication and secret key generation. In *Proceedings of the 44th Annual Design Automation Conference*, DAC '07, pages 9–14, New York, NY, USA, 2007. ACM.

[7] Blaise Gassend, Dwaine Clarke, Marten van Dijk, and Srinivas Devadas. Controlled physical random functions. In *Proceedings of the 18th Annual Computer Security Applications Conference*, ACSAC '02, pages 149–, Washington, DC, USA, 2002. IEEE Computer Society.

[8] Blaise Gassend, Dwaine Clarke, Marten van Dijk, and Srinivas Devadas. Silicon physical random functions. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, CCS '02, pages 148–160, New York, NY, USA, 2002. ACM.

[9] G. Edward Suh, Charles W. O'Donnell, Ishan Sachdev, and Srinivas Devadas. Design and implementation of the aegis single-chip secure processor using physical random functions. In *Proceedings of the 32Nd Annual International Symposium on Computer Architecture*, ISCA '05, pages 25–36, Washington, DC, USA, 2005. IEEE Computer Society.

[10] Jorge Guajardo, Sandeep S. Kumar, Geert-Jan Schrijen, and Pim Tuyls. Fpga intrinsic pufs and their use for ip protection. In *Proceedings of the 9th International Workshop on Cryptographic Hardware and Embedded Systems*, CHES '07, pages 63–80, Berlin, Heidelberg, 2007. Springer-Verlag.

[11] Srini Devadas. Physical unclonable functions and secure processors. In *Proceedings of the 11th International Workshop on Cryptographic Hardware and Embedded Systems*, CHES '09, pages 65–65, Berlin, Heidelberg, 2009. Springer-Verlag.

[12] Dominik Merli, Georg Sigl, and Claudia Eckert. Identities for embedded systems enabled by physical unclonable functions. In Marc Fischlin and Stefan Katzenbeisser, editors, *Number Theory and Cryptography*, volume 8260 of *Lecture Notes in Computer Science*, pages 125–138. Springer Berlin Heidelberg, 2013.

[13] G. Edward Suh, Charles W. O'Donnell, and Srinivas Devadas. Aegis: A single-chip secure processor. *IEEE Design & Test of Computers*, 24(6):570–580, 2007.

[14] Stefan Katzenbeisser, Ünal Kocabaş, Vladimir Rožić, Ahmad-Reza Sadeghi, Ingrid Verbauwhede, and Christian Wachsmann. Pufs: Myth, fact or busted? a security evaluation of physically unclonable functions (pufs) cast in silicon. In *Proceedings of the 14th International Conference on Cryptographic Hardware and Embedded Systems*, CHES'12, pages 283–301, Berlin, Heidelberg, 2012. Springer-Verlag.

[15] Ulrich Rührmair, Frank Sehnke, Jan Sölter, Gideon Dror, Srinivas Devadas, and Jürgen Schmidhuber. Modeling attacks on physical unclonable functions. In *Proceedings of the 17th ACM Conference on Computer and Communications Security*, CCS '10, pages 237–249, New York, NY, USA, 2010. ACM.

[16] Sanorita Dey, Nirupam Roy, Wenyuan Xu, and Srihari Nelakuditi. Acm hotmobile 2013 poster: Leveraging imperfections of sensors for fingerprinting smartphones. *SIGMOBILE Mob. Comput. Commun. Rev.*, 17(3):21–22, November 2013.

[17] Aydin Aysu, Nahid Farhady Ghalaty, Zane Franklin, Moein Pahlavan Yali, and Patrick Schaumont. Digital fingerprints for low-cost platforms using mems sensors. In *Proceedings of the Workshop on Embedded Systems Security*, WESS '13, pages 2:1–2:6, New York, NY, USA, 2013. ACM.

[18] Hristo Bojinov, Yan Michalevsky, Gabi Nakibly, and Dan Boneh. Mobile device identification via sensor fingerprinting. *arXiv preprint arXiv:1408.1416*, 2014.

[19] Jiayang Liu, Lin Zhong, Jehan Wickramasuriya, and Venu Vasudevan. uwave: Accelerometer-based personalized gesture recognition and its applications. *Pervasive Mob. Comput.*, 5(6):657–675, December 2009.

[20] Bo Yang, Kang Yang, Zhenfeng Zhang, Yu Qin, and Dengguo Feng. Aep-m: Practical anonymous e-payment for mobile devices using arm

trustzone and divisible e-cash. In *International Conference on Information Security*, pages 130–146. Springer, 2016.

[21] Yingjun Zhang, Shijun Zhao, Yu Qin, Bo Yang, and Dengguo Feng. Trusttokenf: A generic security framework for mobile two-factor authentication using trustzone. In *Trustcom/BigDataSE/ISPA, 2015 IEEE*, volume 1, pages 41–48. IEEE, 2015.

[22] Gianmarco Baldini and Gary Steri. A survey of techniques for the identification of mobile phones using the physical fingerprints of the built-in components. *IEEE Communications Surveys & Tutorials*, 19(3):1761–1789, 2017.

[23] National Institute of Standards and Technology (NIST). *FIPS-180-2: Secure Hash Standard*, Aug 2002. available online at http://www.itl.nist.gov/fipspubs/.

[24] Pierre L'Ecuyur and Richard Simard. Testu01: A c library for empirical testing of random number generators. *ACM Transactions on Mathematical Software*, 33(4):22–40, 2007.

[25] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An introduction to statistical learning*, volume 6. Springer, 2013.