Towards Trusted Execution of Multi-Modal Continuous Authentication Schemes

Carlton Shepherd
Royal Holloway,
University of London,
Egham, UK
carlton.shepherd.2014
@rhul.ac.uk

Raja Naeem Akram
Royal Holloway,
University of London,
Egham, UK
r.n.akram@rhul.ac.uk

Konstantinos
Markantonakis
Royal Holloway,
University of London,
Egham, UK
k.markantonakis@rhul.ac.uk

ABSTRACT

The emergence of powerful, sensor-rich devices has spawned the development of *continuous authentication* (CA) schemes on commodity hardware, where user behaviour is compared to past experience to produce an authentication decision, with the aim of addressing challenges with traditional authentication schemes. Current CA proposals, however, have largely neglected adversaries present in real-world deployments, namely the ubiquity of malware and arbitrary software attacks. This has particular importance when a device cannot be trusted by a third-party, e.g. a corporation, that controls access to assets based on CA decisions. A software compromise, either on the platform or scheme implementation, may enable the modification of authentication scores, gain insights into user behavioural patterns, or gain unauthorised access to restricted assets. For the first time, we examine two standardised constructs that offer isolated and trusted execution – Secure Elements (SEs) and Trusted Execution Environments (TEEs) – even when an adversary has root-level privileges for protecting CA schemes while retaining deployability. Based on these, we implement the first system for evaluating TEE-based CA on a consumer mobile device using Intel SGX – providing confidentiality, integrity and trust assurances over untrusted world implementations. We present an evaluation of TEE- and non-TEE performance using methods proposed in related work. The results indicate that trusted CA can be performed in an efficient fashion while removing the main platform from the TCB.

CCS Concepts

•**Security and privacy** → **Authentication; Trusted computing;** *Mobile platform security;*

Keywords

Continuous Authentication; Trusted Execution Environments; Trusted Computing; Mobile Security

SAC 2017, April 03–07, 2017, Marrakech, Morocco

ISBN 978-1-4503-4486-9/17/04…$15.00

DOI: http://dx.doi.org/10.1145/3019612.3019652

1. INTRODUCTION

Modern mobile devices are often used to store sensitive user data, such as contact, financial and social media information. The physical nature of mobile devices, however, introduces a range of attack vectors typically avoided with desktop workstations. Small form-factor devices can be misplaced, dropped or pickpocketed relatively easily, such as in restaurants and on public transport. Despite this, past studies [8, 19] suggest that as many as 40% of smartphone users do not use a secure, secret-based locking mechanisms. Additionally, Hayashi et al. [10] show that the 'all-or-nothing' nature of mobile authentication, where access to all device functionality is granted/denied upon the completion/failure of a secret, is a 'remarkably poor fit' to user preferences. Rather, users have shown preference towards explicitly unlocking for certain applications – those holding sensitive data, such as banking – while avoiding this for relatively benign ones, like navigation. Similarly, users favour setting different authentication challenges in locations of varying perceived safety [9]. These factors have inspired the exploration of *continuous authentication*[1] (CA) schemes by researchers, where the device state is influenced by contextual information gathered primarily from mobile sensors. However, while numerous schemes have been proposed in the literature, little attention has been paid towards secure and trusted on-device execution of CA schemes in practice. If any element is modified – the sensor data, underlying user model or the decision itself – access could be granted to sensitive assets and services maliciously.

For the first time, we explore candidates for providing trusted execution of CA schemes using a Secure Element (SE) and Trusted Execution Environment (TEE) to offer stronger confidentiality, integrity and trust assurances. After introducing CA and proposed schemes (Section 2), we examine these candidates for trusted CA (Section 3). Next, we use the Intel SGX TEE to implement a test-bed for performing trusted CA using three learning algorithms (Section 4), and evaluate their performance using a publicly-available dataset from real-world users (Section 5). Finally, we discuss related work, the conclusions of our results and identify avenues for future research (Sections 6 and 7).

This paper provides the following contributions:

- The design and implementation of a test-bed for performing trusted execution of multi-modal CA using the

[1]Also known as 'implicit', 'on-going' and 'transparent' authentication in related literature.

Intel SGX architecture, and a discussion of any potential issues at each stage. To our knowledge, this is the first foray in assessing TEE-based CA.

- An assessment of various architectures for executing continuous authentication schemes, along with their associated benefits and drawbacks relating to security, performance and deployability.
- Performance statistics and source code are released for further research by the community[2]. We hope that our SGX-compatible machine learning primitives will allow CA developers to implement trusted schemes.

2. CONTINUOUS AUTHENTICATION

In this section, CA and its motivation is described, along with the methods used in proposed schemes. We formalise the problem statement and expand the existing threat model.

2.1 Background and Motivation

Traditional user authentication has relied on what one *knows*, e.g. passcodes and PINs; *has*, e.g. tokens and smart cards; or *is*, e.g. behavioural and physical biometrics. Password and pattern schemes, however, suffer from well-studied issues relating to rememborability and shoulder-surfing attacks [28]. *Has* methods, such as tokens, are generally seen to be more secure, but are considered burdensome when managing multiple items [3]. Biometrics comprise physiological and behavioural measures, and recent work [10] shows that users are receptive to these due to their convenience. Physiological systems such as fingerprint recognition, however, suffer from inherent reliability issues due to injury or the environment, and replacement – the problem of 'revoking' or 'replacing' a fingerprint after divulgence. With behavioural biometrics, achieving acceptable error rates remains an open problem [3]. CA aims to address the shortcomings of traditional schemes by transparently monitoring the user's environment and context via device sensors. Various schemes have been proposed that rely upon keystroke- [22], motion- [4], environmental- [14] and touch-based [5] features. Recent research – discussed in the following subsection – has tended to focus upon a *multi-modal* approach, where multiple data sources are used to authenticate users with improved error rates. Generally, CA aims to address the following areas:

- Reducing the number of unnecessary login attempts through modulating the lock-screen strength [9, 23].
- A primary or second line of defence for detecting unauthorised users who have bypassed the initial lock-screen, e.g. via shoulder-surfing [13, 16, 27].
- Adjusting access control policies based on context [21].

A typical approach involves training a supervised learning classifier using labelled sensor data, and classifying subsequent data according to these labels. Classes may correspond to locations, e.g. adjusting access control policies in safe/unsafe locations as per [21], or inferring the accept/reject status directly [16, 23, 27]. Certain work does not focus on classifying data, but produces a real-valued probability that is thresholded for purposes decided by the operator [13, 15]. CA schemes typically follow two phases (Figure 1):

[2]Available at: https://bitbucket.org/carltonshepherd/trustedca

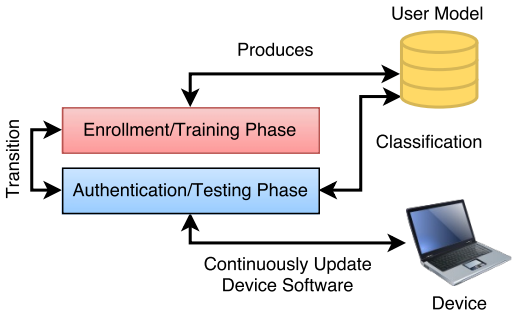

Figure 1: Generalised State Flow of CA Schemes.

Table 1: Summary of Multi-modal CA Schemes.

Paper	Method	Sensors
Hayashi et al. [9]	**NB**	**GPS**†
Micallef et al. [19]	DT	Acc., WiFi, Light, Sound, Mag.
Shi et al. [27]	**NB**	**GPS, Cell ID, Acc., Sound, Touch**
Mittenen et al. [21]	RF, kNN, NB	GPS, WiFi
Riva et al. [23]	**SVM**	**Light, Temp., Hum., BT, Touch, Logins**
Li et al. [16]	SVM	Touch, Acc., Ori., Mag.

† Only GPS evaluated, but approach is generalisable to N inputs.
* Acc.: Accelerometer; BT: Bluetooth; DT: Decision Tree; Hum.: Humidity; Mag.: Magnetometer; NB: Naive Bayes; Ori.: Orientation; RF: Random Forest; Temp.: Temperature.

1. **Enrollment** – sensor data is collected over a period of N days, usually with user prompts to ascertain labelled data, and training a classifier to model user behaviour.
2. **Authentication** – occurs after the model is generated. Sensor data is collected over a time window – hours, minutes or per sample – and used as input to the classifier to produce the corresponding label.

In both stages, data is collected at predetermined intervals of M samples at a rate of T milliseconds. Data is classified directly or after a feature extraction stage. A model, produced from a classifier, may be retrained/reproducing after certain intervals, i.e. transitioning periodically between the authentication to enrollment stage, in order to address gradual shifts in user behaviour over time, e.g. every 1-2 weeks [16] or after each sample in real-time [15]. We summarise the key attributes of related schemes in Table 1.

2.2 Problem Statement

The question of executing a CA scheme in a *secure* and *trustworthy* fashion has not been addressed directly in related work. While attention has duly been given to scheme accuracy, it has yet to be directed to the extended threats faced 'in the wild', where operating systems, user applications and hardware vary significantly between users. A CA scheme operator, such as a corporation altering access to remote resources based on context, would be unwise to place unreserved trust in users' devices given the potential value of assets at stake. The presence of mobile malware, along with the prevalence of rooted/jailbroken devices, potential application, OS and kernel bugs/exploits, provide a plethora of attack vectors on a CA scheme in reality. The information

stored and used by such a scheme during execution may provide privacy-intrusive insights into users' behavioural patterns. As such, a secure architecture for CA schemes is necessary that offers protection against these adversaries while minimising deployment costs. Moreover, the high volume of sensor data and the maintenance of a behavioural model provide unique challenges over traditional biometric schemes (discussed further in Section 3).

2.3 Threat Model

Current CA schemes aim generally to protect against the following: **a)** a primary or second-line of defence against a device thief who bypasses the initial lock-screen through shoulder-surfing, successful guessing, or where no mechanism is present, with the aim of accessing sensitive data; **b)** a curious adversary, e.g. a co-worker, who accesses the device to browse private content. In our work, this threat model is expanded to incorporate adversaries present in a practical deployment, and we specifically consider general software threats that aim to influence or subvert the operation of the scheme on the device. This can be the result of malicious software (malware) installed accidentally or intentionally on the device, or a software exploit at the application, platform or kernel layers where a CA scheme might reside.

Explicitly, we aim to address a strong adversary with the capability to perform one or more of the following: **1)**, the CA code under execution may be modified arbitrarily, as may the user model and authentication decisions, and **2)**, model data may be eavesdropped during execution or observed in storage. It follows that this model considers all software components of a monolithic OS, such as Windows or Android – the kernel, platform and application layers – to be potentially compromised, i.e. the Rich Execution Environment (REE) or Rich OS. We do, however, trust in a TEE or SE (discussed next) that relies upon trusted hardware [7].

3. ARCHITECTURAL DESIGN

In accordance with the extended threat model, we explore candidate architectures that offer the following properties:

- **Integrity**: **a)**, Prohibits unauthorised modification of authentication decisions and behavioural model, and **b)**, provides protection to unauthorised tampering of CA scheme code, during and prior to execution.
- **Confidentiality**: **a)**, Provides cryptographic protection against plaintext disclosure of the user model, e.g. through secure storage; and **b)**, prevents eavesdropping of sensor data during processing.
- **Trust**: **a)**, A third-party should be able to attest the authenticity of authentication decisions, thus providing confidence that they were produced in an integral environment correctly. **b)** Cryptographic operations, e.g. signing and verification, and key storage ought to be performed by tamper-resistant hardware.

Moreover, candidates should also require little or, ideally, no hardware changes to the device, while minimising software modifications in order to maximise deployability.

3.1 Candidate Architectures

Secure execution is the ability to maintain confidentiality and integrity of run-time states during code execution. Conventionally, this leverages software isolation, e.g. through hypervisors and virtualisation, and/or tamper-resistant hardware, such as smart cards. Trusted execution includes extensions for attesting to the state of executed code. A trusted hardware entity, a Root of Trust (RoT), is typically used to measure the state of the executing environment to determine whether it is executing to expectations, as well as responding to third-party requests for such data (remote attestation). Some technologies, such as Intel's Trusted eXecution Technology (TXT), rely on a Trusted Platform Module (TPM) as the RoT, while ARM's TrustZone uses secure ROM on the SoC. We focus on candidates that offer trusted execution in order to protect the sensitive assets of a CA scheme – authentication decisions, user model and sensor data – and the code under execution. Two primitives are explored that have received widespread attention and seen deployment on consumer devices: Secure Elements (SEs) and Trusted Execution Environments (TEEs). The reliance on standardised constructs minimises potential deployment overhead commercially.

3.1.1 Secure Elements (SEs)

GlobalPlatform defines an SE as a *"tamper-resistant component which is used to provide the security, confidentiality and multiple application environment required to support various business models"* [6]. SEs exist in various forms – SIMs, smartSDs, UICCs and discrete embedded SEs – and offer hardware tamper-resistance for security-sensitive applications. A typical SE contains a cryptographic co-processor and on-board storage (EEPROM) capable of hosting applications and data, such as payment credentials and fingerprint matching algorithms (used previously by Apple iOS [1]). The storage and processing capabilities of SEs, however, are significantly restricted relative to TEEs and rich OSs: modern commercial mobile SEs normally offer approximately 300MHz clock speeds and 1.5MB persistent storage[3].

3.1.2 Trusted Execution Environments (TEEs)

According to GlobalPlatform, a TEE is an *"execution environment that runs alongside but isolated from an REE...it protects assets from general software attacks, defines rigid safeguards as to data and functions that a program can access, and resists a set of defined threats"* [6]. Its precise capabilities have been the subject of much debate [24], but is generally considered to offer isolated execution enforced by trusted hardware, and maintains strong resistance to REE software attacks through the use of restricted APIs and memory accesses. Example TEEs include ARM TrustZone and Intel TXT and SGX. Unlike SEs, TEEs share execution hardware, i.e. CPU and RAM, with the REE, and its performance is considered to significantly exceed SEs [7]. The GlobalPlatform TEE specification, which governs many TrustZone TEEs, stipulates a trusted OS and an arbitrary number of Trusted Applications (TAs) that contain sensitive data and logic, e.g. for payments and DRM, and communicates with the REE via a restricted monitor. The 'right' TEE architecture, however, is yet to reach consensus: SGX spawns independent execution environments ('enclaves') on-demand per application, while GP/TrustZone maintains a single trusted world for all TAs. Regardless, all TEEs aim to protect against general software attacks, but do not necessarily offer strong protection against hardware attacks.

[3]See the Infineon SL97 and ST ST33J2M0.

Figure 2: Comparison of Rich OS, TEE and SE attributes using GlobalPlatform definitions [7].

3.2 Discussion

The principle behind separating a CA scheme from the Rich OS is to minimise the Trusted Computing Base (TCB) – the set of hardware, software and firmware components critical to maintaining the security of a system. Moreover, monolithic OSs, such as Windows and Android, are generally too large in size to formally verify their security properties; TEEs and SEs, meanwhile, are significantly smaller and more suitable for such analysis [24]. By relocating CA schemes to a TEE or SE, the TCB is reduced largely to that of the TEE/SE and the CA scheme's application logic, thus reducing the attack surface significantly. A TEE's precise TCB depends primarily on the chosen platform, but for widespread TEEs, such as TrustZone and SGX, the hardware TCB generally comprises a trusted CPU chipset or SoC. The software TCB of a TEE comprises the trusted world containing a trusted OS, firmware and applications (for GlobalPlatform and TrustZone), or a run-time module and enclave logic for SGX [2]. Figure 3.2 compares a Rich OS, TEE and SE in terms of cost, performance and security.

The performance benefits of TEEs make it attractive in pursuit of our design goals, namely for computationally-intensive applications like those requiring machine learning. Our initial investigations using the GCU dataset [15] found that, on average, sensor data surpassed 0.5MB per day per user – far exceeding the memory capacity of commercial SEs at the time of writing when over 3 days of data is used, as per related work [9, 20]. TEEs are supported directly by underlying device CPU/SoC, RAM and storage, offering greater speed and capacity over SEs. Many TEEs, such as Intel SGX, also offer native remote attestation, where third-parties may attest to the integrity of the enclave application. This property allows CA scheme operators to verify the state of the application with a greater degree of trust.

A further consideration of SEs is that other applications/data, such as payment credentials and fingerprint images, would reside simultaneously on one unit. While SEs are significantly less powerful than TEEs, their primary benefit is strong hardware tamper-resistance, thus making it an attractive solution for key storage and auxiliary data small in size. Note that TEEs and SEs are not mutually exclusive; it is possible for a TEE to rely upon an SE, or indeed a TPM, for certain operations such as cryptographic operations and key storage, as noted in GlobalPlatform guidelines [7]. We summarise the key properties of each architecture in Table 2, and compare this to a regular application running in user-mode in the Rich OS (implicit in existing proposals).

4. IMPLEMENTATION

In light of the above discussion, we describe our test-bed implementation for evaluating TEE-based CA. Our system uses the Intel SGX TEE along with implementations of three learning algorithms for performing trusted CA on consumer Intel CPUs, and we present the details and challenges of implementing this on a commercial device.

Table 2: Summary of Candidate Features.

Feature	REE	SE	TEE
Isolated Execution	✗	✓	✓
Performance	Best	Poor	Good
HW Tamper-Resistance	✗	✓	◇
Remote Attestation	✗	✗	✓
Hardware TCB	All	SE MCU†	CPU/SoC*
Software TCB	Large	Smallest	Small

◇ Limited and TEE-dependent.
† Microcontroller unit.
* Applicable for TrustZone and SGX.

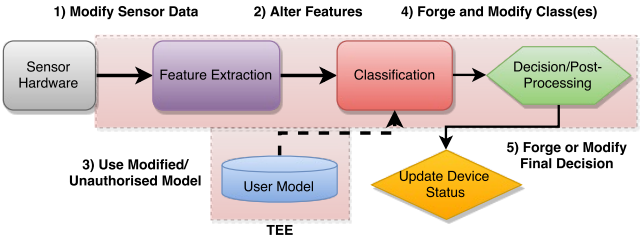

Figure 3: Key CA attack surfaces; we propose using a TEE to isolate and protect those areas in red.

4.1 Intel SGX

Intel SGX [12] is an extension to the x86-64 instruction set that allows the establishment of isolated and protected execution environments known as 'enclaves', while reducing the hardware TCB to the CPU. Simply, SGX aims to defend sensitive assets against general software attacks from a compromised platform. The CPU strictly manages memory accesses to protected enclave code and data, stored in the Enclave Page Cache (EPC) and managed by the Enclave Page Cache Monitor (EPCM); this data may only be accessed by code in enclave memory space. Enclaves may read/write to memory in the Rich OS, but accessing enclave data from the Rich OS or other enclaves is forbidden irrespective of CPU mode, i.e. kernel (ring 0), hypervisor (-1), SMM (-2) or user mode (3). The CPU protects the confidentiality, integrity and freshness of CPU-DRAM traffic through the use of a proprietary hardware Memory Encryption Engine (MEE) using a CPU-bound key generated at boot-time. Enclaves are inaccessible via regular stack calls and jump instructions; switching between the environments is conducted via designated CPU-level instructions: `ECALL`s are responsible for invoking enclave functions, while `OCALL`s are used for calling untrusted world functions from enclaves. During linking and compilation, an enclave and its contents are measured and signed using a key known to the developer(s), which is used to verify the structure and content of the enclave. If altered maliciously, the resulting mismatch is detected remote attestation using the Enhanced Privacy ID (EPID) protocol – a Direct Anonymous Attestation (DAA) scheme that uses a group signature scheme in

conjunction with a CPU-bound attestation key. SGX, however, like the GlobalPlatform TEE, does not defend against logical flaws; design errors may expose an enclave-based application to manipulation by an attacker. Unlike TrustZone and the GlobalPlatform TEE (see Section 3.1.2), trusted enclaves are spawned on-demand by applications wishing to protect sensitive information; broadly, multiple TEEs are maintained by the SGX run-time when required by host applications.

We implement three SGX-compatible learning algorithms: Naive Bayes (NB), k-Nearest Neighbour (kNN) and Logistic Regression (LR). This selection stems from related work and the challenges described in Section 4.3, which necessitated the development of *bespoke* algorithms, rather than employing incompatible third-party libraries. Subsequently, simpler (but not necessarily less powerful) learning algorithms were favoured to assure correctness. The nature of our implementation guarantees that only two context switches between SGX and the Rich OS will occur: **1)**, to transfer raw sensor data from the Rich OS to SGX; and **2)**, transferring an acknowledgement message (after training) or an authentication score (testing) from SGX to the Rich OS.

4.2 Test-bed Construction

The test-bed implements SGX-compatible versions of the above algorithms and a simple API is exposed through which application developers can train and test data in the TEE from the Rich OS. All functions use purely `ECALL`s; no CA-specific data is divulged to the Rich OS other than: 1), a value indicating successful training, and 2), the authentication decision. The test-bed spawns a single enclave to test each algorithm at a time and was implemented in C++ using Microsoft Visual Studio and the Intel SGX SDK – we provide the source code freely (see Section 1). The Intel SGX SDK allows the development of applications that instantiate enclaves natively; we implement the training and testing phases inside the enclave environment, which are managed through a restricted API from the host application.

4.3 Challenges

Initially, we intended to use a smartphone which most CA schemes have been evaluated upon, using ARM TrustZone as our chosen TEE. Unfortunately, TrustZone trusted applications (TAs) require manufacturer certification – prompting researchers to use development kits [17]. It is our opinion that development boards do not truly reflect consumer devices and system environments and, as such, we opt for Intel SGX, available on most new Intel Skylake CPUs, which allows us to evaluate on a commercially-available laptop: a Lenovo Thinkpad T460s. Another motivation for SGX was its close integration with Microsoft's Visual Studio IDE and its publicly-available documentation for developing enclaves.

Intel SGX, however, supports only a subset of C/C++ libraries. We found, for example, that includes such as `<regex>`, `<sstream>` and `<iostream>` were either partially or wholly unavailable. Intra-enclave thread creation is also unsupported, along with `rand` and `srand` PRNGs, forcing calls to the CPU's hardware RNG via `sgx_read_rand`. Fundamental functions like `printf` also had to be redefined: printing from enclaves involved a context switch to the Rich OS through an `OCALL`. We planned originally to use a popular C++ machine learning library, such as Shark[4], to provide algorithm implementations. Such tools, however, rely heavily upon unavailable libraries, meaning bespoke, SGX-compatible algorithms had to be developed for use within enclaves. For complex learning algorithms, such as SVMs, this is difficult while retaining correctness without expert oversight. Popular SVM libraries typically have non-trivial code bases (>5K lines of code for LibSVM). This issue was less problematic for kNN, LR and NB – all relatively simpler in nature – and we freely offer these in our test-bed for use by the community.

[4]Shark Machine Learning: http//image.diku.dk/shark/

Table 3: Training Phase: Average Time Per User (in milliseconds; S.D. given in brackets).

Method	1-Day	5-Days	14-Days	21-Days
NB	80.45 (16.07)	1040 (278.0)	3650 (606.6)	3534 (1362)
kNN, $k = 3$	64.50 (16.50)	655.2 (95.84)	2218 (227.2)	1978 (429.3)
kNN, $k = 5$	66.57 (15.09)	660.4 (76.80)	2278 (304.9)	1868 (315.9)
LR	2027 (26.11)	8843 (101.3)	12114 (421.8)	13018 (843.5)
SGX NB	3.652 (0.557)	20.29 (2.479)	55.49 (7.027)	57.19 (13.95)
SGX kNN, $k = 3$	2.005 (0.189)	11.46 (1.619)	30.74 (3.129)	25.97 (9.17)
SGX kNN, $k = 5$	1.938 (0.347)	11.74 (1.339)	31.13 (4.053)	30.21 (1.828)
SGX LR	15.68 (3.73)	90.30 (5.77)	139.65 (15.90)	120.12 (14.11)

5. EVALUATION

Our initial concern with TEE-based CA was the additional overhead incurred via the use and maintenance of a TEE application, such as the additional memory checks, any context switches (between enclave and non-enclave environments) and memory encryption performed by SGX. In this section, we describe and present the results of TEE versus non-TEE CA by performing an empirical comparison of key functionality implemented in Section 4.

We employ the GCU dataset by Kayacik et al. [15], which provides mobile sensor data collected from 7 users (staff and students at a UK university) over a period of 2–14 weeks. This data is sampled at a rate of 5 minutes, comprising nearby WiFi access points, cell tower IDs and current applications running on the device. Optimal sampling rates and window sizes depend entirely upon scheme and the performance/security requirements of the scheme operator (this trade-off is analysed in [20]). To avoid ambiguity, we use the GCU dataset without re-sampling. Training set size also depends on the requirements of the operator: more data/large training sets will invariably imposes greater computational demand on hardware, but has been shown to improve scheme accuracy [16]. As such, we evaluate a set of training periods discussed in related work, ranging from 1 day to 3 weeks [9, 20]. The following components are evaluated:

- **Training**: the performance of training each algorithm using sets lasting 1, 5, 14 and 21 days, reflecting related work and beyond. KNN is unique in having no dedicated training phase, so we measure the time to load necessary sensor data into internal data structures in preparation for testing (implicit for NB and LR).

- **Testing**: the overhead of producing the authentication decision from a vector of sensor data. We measure the time taken to classify one hour of sensor data using all three of our implemented algorithms.

The round-trip time is measured between issuing the request from the untrusted world and receiving the corresponding result: an acknowledgement and authentication decision for

Table 4: Testing Phase: Average Time Per User (in microseconds).

Method	1-Day	5-Days	14-Days	21-Days
NB	125.7 (5.936)	142.7 (8.712)	158.3 (40.33)	153.3 (35.64)
kNN, $k = 3$	458.4 (64.23)	2539 (319.0)	6554 (972.0)	5976 (1623)
kNN, $k = 5$	464.1 (55.69)	2481 (258.5)	6634 (998.9)	5949 (1803)
LR	95.2 (8.838)	102.8 (30.39)	112.9 (36.90)	113.5 (40.01)
SGX NB	14.28 (2.498)	14.17 (0.951)	12.86 (1.864)	14.67 (9.333)
SGX kNN, $k = 3$	100.4 (20.02)	584.7 (63.30)	1598 (231.6)	1481 (468.5)
SGX kNN, $k = 5$	100.8 (16.79)	590.6 (63.90)	1593 (197.8)	1452 (431.0)
SGX LR	8.876 (1.002)	9.405 (2.803)	9.384 (0.776)	9.523 (1.249)

training and testing respectively. We measure this using the `<chrono>` library from C++11, which provides a high-resolution clock with nanosecond precision. The anonymised GCU data was converted to discrete integers and separated into the training and test sets for each user using a Python script. The training set represents the first 1, 5, 14 and 21 days of sensor data from each user, thus simulating an enrollment period after purchasing the device. We measured this for each user where possible: in total, all 7 users had training data available for up to 14 days, while only 3 users had over 21 days of data.

For training, we follow Li et al. [16] and randomly pair each user with another to construct negative/positive valued training sets. Here, the test user's readings are considered as legitimate (positive) values, while the paired user's values are considered illegitimate (negative). Ideally, a classifier should positively classify all data belonging to a legitimate user, and negatively classify that from an illegitimate user. For the testing/authentication phase, we measure the average time taken to classify a single feature vector. The training sets comprise of all the readings after the authentication date until the end of the file for each user, and the average time is computed to classify each line/individual feature vector across all users. The evaluation was performed using a consumer laptop – a Lenovo Thinkpad T460s with an SGX-enabled Intel i5-6200U CPU (2.8GHz clock speed, 3MB cache) and 8GB DDR4 RAM (2133MHz) – using Microsoft Windows 10 as the deployment platform.

5.1 Results Discussion

We present our findings in Tables 3 and 4. As expected, training times for Logistic Regression, Naive Bayes far exceed kNN, which requires no formal training procedure; NB involves the computation of all prior and conditional probabilities for the occurrence of feature vectors given their class label, while LR minimises the least-squares cost function using gradient descent. We also expected kNN testing to far exceed that of NB and LR, as it iterates all training set members to find the closest k neighbours to the test vector. This is compared with NB where, once priors and conditionals are computed, computing the class is a series of multiplications, which performs in near-constant time; LR simply computes the logit function (Equation 1) for a given feature vector, $\mathbf{x}$, and learned parameters, θ. These trends are consistent across all days worth of data.

The results of training and testing drop slightly between 14 and 21 days, which we found was due to the usage variance between users: while the dataset is sampled at 5-minute intervals, this does not account for periods when the device is disabled. Thusly, Figure 4 depicts the mean number of samples captured per day for each participant. Only three

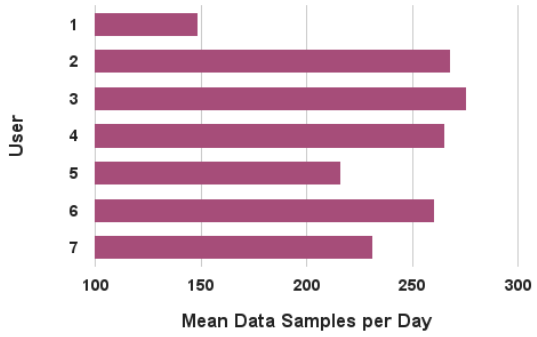

Figure 4: Mean samples per day per participant.

users had data lasting 21-days – users 1, 2 and 5 – constituting 198 samples per day per user on average, while, for 14 days, this rises to 245 samples per day. The differences in sample 'density', due to the usage variance between users, is arguably the cause for similar timings between 14-day and 21-day training and testing, and thus biased heavily by the low size of User 1. Evidently, device usage is a significant influence in determining CA training and testing times.

$$LR_\theta(\mathbf{x}) = \frac{1}{1 + e^{-\theta^T \mathbf{x}}} \tag{1}$$

The surprising factor is the significant difference between SGX and non-SGX performance. At first, we believed SGX would incur some overhead, with the additional memory enforcement and context switching; yet, it exceeds the REE by 3–85 times depending on method. After investigation, while SGX and non-SGX implementations were identical, enclave applications are linked with Intel's own C (`sgx_tstdc.lib`) and C++ standard libraries (`sgx_tstdcxx.lib`), which, for C, must be compulsorily linked to any SGX enclave application. These libraries utilise optimised functions from Intel Integrated Performance Primitives (IPP) [11]. This suite offers parallelised, multi-threaded optimisations of standard routines based on the most recent AVX and SSE instruction sets, which *"significantly increases performance"*. Comparatively, the non-SGX portion uses the Microsoft Visual C++ standard library, with no such enhancements; we illustrate this comparison in Figure 5.

Interestingly, training times benefit most, due likely to the computationally intensive methods involved, e.g. prior and conditional probabilities for NB. Further investigations revealed that IPP *does* favour such arithmetical[5] and string-based[6] operations strongly. The observed performance of SGX, however, raises security concerns in itself: there is a clear incentive for developers to move non-critical components to enclaves if some perceived security benefits can be gained with little performance overhead. This would dangerously increase the TCB of the trusted application, which may expose security-critical components to logical flaws or programming bugs from non-security related code sections. The essence of a secure application in any TEE is to minimise its TCB to that which is functionally necessary, and the incentive to increase the TCB for performance gains may undermine the application's overall security in reality.

[5]https://software.intel.com/en-us/node/502094
[6]https://software.intel.com/en-us/node/501987

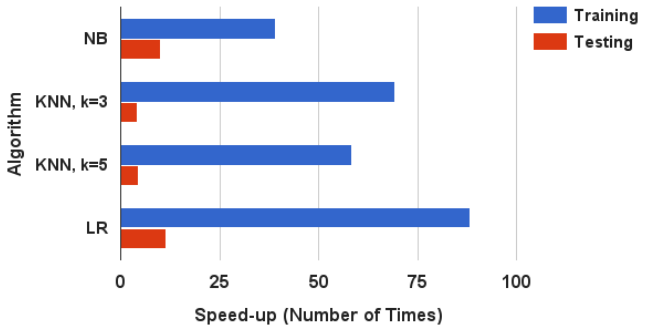

Figure 5: Non-SGX versus SGX speed-up averaged across all training/testing times.

5.2 Limitations

Currently, our test-bed does not support CA approaches using similarity or unsupervised learning, as per [15, 13]. Rather, we support multi-class supervised learning with dedicated enrollment and testing phases, as in [9, 21, 23], due to the paradigm's popularity. It is yet to be seen which CA approach will emerge as the most usable and reliable, but a logical progression would be to investigate similarity-based approaches. Further, only three algorithms were implemented as part of our test-bed – Naive Bayes, LR and kNN – due to the difficulty of porting more intricate methods correctly, such as SVMs and Random Forests, without expert oversight. Given the widespread deployment of SGX, it is hoped that correct SGX implementations of complex learning algorithms will emerge over time. We also focus on only three modalities available to us in the GCU dataset: acquiring data for CA is a non-trivial task, often requiring users to carry devices for many weeks (or months) for a realistic evaluation. Additional modalities would allow a comparison for how trusted CA scales with input sources. Lastly, our work is evaluated only on Intel SGX for x86-64; testing other TEE architectures, such as ARM TrustZone, would provide greater context for our work.

6. RELATED WORK

While trusted CA has not been addressed in the literature to our knowledge, work has been conducted in protecting sensor values and computing authentication decisions remotely. Liu et al. [18] propose an architecture offering confidential, integral and trusted sensor values using a TPM-backed hypervisor for x86 platforms, and TrustZone for ARM systems. In this work, trusted I/O is provided between sensors and the execution environments, and a trusted GPS sensor is implemented that provides attestation of values and sealing (that is, protecting a secret by binding it to a policy based on sensor readings).

Safa et al. [25] propose a scheme for computing authentication decisions remotely from encrypted feature vectors using homomorphic encryption. Here, an encrypted behavioural model is stored on a remote server and a decision is computed from incoming feature vectors (also encrypted) without revealing either to a potentially intrusive server. A security proof is provided; the scheme is not implemented or evaluated, however. This method of cloud-based authentication also requires constant online connectivity between the device and remote server, and does not consider device-centric security issues, which we aim to tackle in this work.

Sedenka et al. [26] propose protocols for secure outsourcing of biometrics using garbled circuits and homomorphic encryption, which enables remote computation of decisions under the honest-but-curious server and malicious client models. A security analysis is provided and the schemes are evaluated using a consumer laptop and smartphone. The results suggest that outsourcing is possible, but not without a communication and time penalty, ranging from 4–174MB and 0.85s–45.9s under varying conditions. As with the preceding work, we consider security issues relating to on-device execution, rather than outsourcing this to a remote party.

6.1 Future Research

As part of our ongoing work, we aim to investigate the following avenues of research:

- Expanding our test-bed to other TEEs, such as AMD PSP and ARM TrustZone, and providing a comparative performance analysis of various environments.
- Evaluating trusted CA on varying device types, such as wearables, and the associated performance penalty, alongside any potential security limitations.
- Investigating the performance of further learning algorithms, such as neural networks and Random Forests.
- Lastly, we plan to collect datasets from new users and explore how trusted CA performance scales with more input sources using additional sensor modalities.

7. CONCLUSION

In this work, we introduced the need for trusted execution CA and broadened the existing threat model to incorporate adversaries encountered in a real-world deployment. To this end, we examined two standardised constructs for offering greater trust and security provisions without compromising deployability – Secure Elements and Trusted Execution Environments – and their suitability was analysed, along with their features, advantages and disadvantages. Following this, we proposed using a TEE to encapsulate sensitive aspects of a CA scheme in order to address the extended threat model. We implemented and evaluated our proposal in a test-bed environment using a commercially-available TEE (Intel SGX) on an off-the-shelf consumer device. Three SGX-compatible learning algorithms – Naive Bayes, Logistic Regression and k-Nearest Neighbour – were implemented, and the performance of each was evaluated using a publicly-available dataset comprising sensor data from real-world users. Our results indicate that CA *can* be performed in a commercially-deployed TEE (SGX) without performance penalty, thus reducing the TCB to that of the TEE, rather than trusting the Rich OS as implied previously. Lastly, we ended with a results discussion, the limitations of our research and future work directions.

8. ACKNOWLEDGMENTS

Carlton Shepherd is supported by the EPSRC and the UK government as part of the Centre for Doctoral Training in Cyber Security at Royal Holloway, University of London (KP/K035584/1). The authors also thank the reviewers for their valuable comments in improving the paper.

9. REFERENCES

[1] Apple, Inc. iOS 9.3 Security Guide, 2016. https://www.apple.com/business/docs/iOS_Security_Guide.pdf.

[2] V. Costan and S. Devadas. Intel SGX Explained. Technical report, Cryptology ePrint Archive, Report 2016/086, IACR, 2016.

[3] A. De Luca and J. Lindqvist. Is secure and usable smartphone authentication asking too much? *Computer*, 48(5):64–68, 2015.

[4] M. O. Derawi, C. Nickel, P. Bours, and C. Busch. Unobtrusive user-authentication on mobile phones using biometric gait recognition. In *Proceedings of the 6th International Conference on Intelligent Information Hiding and Multimedia Signal Processing*, pages 306–311. IEEE, 2010.

[5] 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. *IEEE Transactions on Information Forensics and Security*, 8(1):136–148, 2013.

[6] GlobalPlatform. TEE Secure Element API v1.1, 2015.

[7] GlobalPlatform. Trusted Execution Environment (TEE) Guide, 2016.

[8] M. Harbach, E. von Zezschwitz, A. Fichtner, A. D. Luca, and M. Smith. It's a Hard Lock Life: A Field Study of Smartphone (Un)Locking Behavior and Risk Perception. In *Proceedings of the 10th Symposium On Usable Privacy and Security*, pages 213–230, Menlo Park, CA, July 2014. USENIX Association.

[9] E. Hayashi, S. Das, S. Amini, J. Hong, and I. Oakley. CASA: Context-aware Scalable Authentication. In *Proceedings of the 9th Symposium on Usable Privacy and Security*, pages 3:1–3:10, NY, USA, 2013. ACM.

[10] E. Hayashi, O. Riva, K. Strauss, A. Brush, and S. Schechter. Goldilocks and the two mobile devices: going beyond all-or-nothing access to a device's applications. In *Proceedings of the 8th Symposium on Usable Privacy and Security*, page 2. ACM, 2012.

[11] Intel Inc. Integrated Performance Primitives, 2016. https://software.intel.com/en-us/intel-ipp.

[12] Intel, Inc. Software Guard Extensions Evaluation SDK for Windows OS, 2016. https://software.intel.com/en-us/sgx-sdk/documentation.

[13] M. Jakobsson, E. Shi, P. Golle, and R. Chow. Implicit authentication for mobile devices. In *Proceedings of the 4th USENIX conference on Hot Topics in Security*, pages 9–9. USENIX Association, 2009.

[14] N. Karapanos, C. Marforio, C. Soriente, and S. Capkun. Sound-proof: usable two-factor authentication based on ambient sound. In *Proceedings of the 24th USENIX Security Symposium*, pages 483–498, 2015.

[15] H. G. Kayacik, M. Just, L. Baillie, D. Aspinall, and N. Micallef. Data driven authentication: On the effectiveness of user behaviour modelling with mobile device sensors. *arXiv preprint arXiv:1410.7743*, 2014.

[16] L. Li, X. Zhao, and G. Xue. Unobservable re-authentication for smartphones. In *Proceedings of the 20th Network and Distributed System Security Symposium*, 2013.

[17] W. Li, H. Li, H. Chen, and Y. Xia. AdAttester: Secure Online Mobile Advertisement Attestation Using TrustZone. In *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*, pages 75–88, USA, 2015. ACM.

[18] H. Liu, S. Saroiu, A. Wolman, and H. Raj. Software abstractions for trusted sensors. In *Proceedings of the 10th International Conference on Mobile Systems, Applications and Services*, pages 365–378, NY, USA, 2012. ACM.

[19] N. Micallef, M. Just, L. Baillie, M. Halvey, and H. G. Kayacik. Why Aren't Users Using Protection? Investigating the Usability of Smartphone Locking. In *Proceedings of the 17th International Conference on Human-Computer Interaction with Mobile Devices and Services*, pages 284–294, NY, USA, 2015. ACM.

[20] N. Micallef, H. G. Kayacik, M. Just, L. Baillie, and D. Aspinall. Sensor use and usefulness: Trade-offs for data-driven authentication on mobile devices. In *IEEE International Conference on Pervasive Computing and Communications*, pages 189–197. IEEE, 2015.

[21] M. Miettinen, S. Heuser, W. Kronz, A. Sadeghi, and N. Asokan. ConXsense: Automated Context Classification for Context-aware Access Control. In *Proceedings of the 9th Symposium on Information, Computer and Communications Security*, pages 293–304. ACM, 2014.

[22] F. Monrose and A. D. Rubin. Keystroke dynamics as a biometric for authentication. *Future Generation Computer Systems*, 16(4):351–359, 2000.

[23] O. Riva, C. Qin, K. Strauss, and D. Lymberopoulos. Progressive authentication: deciding when to authenticate on mobile phones. In *Proceedings of the 21st USENIX Security Symposium*, pages 301–316, 2012.

[24] M. Sabt, M. Achemlal, and A. Bouabdallah. Trusted execution environment: What it is, and what it is not. In *Proceedings of the 14th IEEE International Conference on Trust, Security and Privacy in Computing and Communications*, volume 1, pages 57–64. IEEE, 2015.

[25] N. A. Safa, R. Safavi-Naini, and S. F. Shahandashti. Privacy-preserving implicit authentication. In *ICT Systems Security and Privacy Protection*, pages 471–484. Springer Berlin Heidelberg, 2014.

[26] J. Sedenka, S. Govindarajan, P. Gasti, and K. S. Balagani. Secure outsourced biometric authentication with performance evaluation on smartphones. *IEEE Transactions on Information Forensics and Security*, 10(2):384–396, 2015.

[27] W. Shi, F. Yang, Y. Jiang, F. Yang, and Y. Xiong. Senguard: Passive user identification on smartphones using multiple sensors. In *Proceedings of the 7th International Conference on Wireless and Mobile Computing, Networking and Communications*, pages 141–148. IEEE, 2011.

[28] F. Stajano. Pico: No more passwords! In *Security Protocols XIX*, pages 49–81. Springer, 2011.