# Using Trusted Execution Environments to Enable Integrity of Offline Test Taking

Rahul Shivu Mahadev[1], Arvind Seshadri[2], Sriram Rajamani[3], and Viraj Kumar[4]

[1] Department of ISE, PESIT, Bangalore,
rahul.mahadev7@gmail.com,
[2] IBM Research,
asesdri@gmail.com,
[3] Microsoft Research,
sriram@microsoft.com,
[4] Department of CSE, PES University, Bangalore,
viraj.kumar@pes.edu

**Abstract.** Today automated assessment in online courses (MOOCs, SPOCS, etc.) is done in a client-server fashion. The only trusted component is the server as it is run by the test creator. The user accesses the automated assessment module by logging into the server using their mobile device. The server sends questions to the user's device one-by-one. For each question, the user types their answer into their device, which is then transmitted to the server for evaluation and scoring. This process requires constant internet connectivity on the user's device, which may be a difficult requirement in developing nations. For evaluating complex assignments, it also places significant load on the server. This paper is aims to address these issues by providing a secure offline solution for the automated assessment problem. An offline approach performs the evaluation and scoring on the user's mobile device and we must protect the evaluation and scoring process from being tampered by the user. To achieve this, we divide the assessment module into two parts: (1) an untrusted user facing GUI component, and (2) a trusted enclave component that handles the evaluation and scoring, and use a hardware-rooted isolated execution technology (ARM TrustZone) to create an isolated enclave for the execution of the enclave component. The execution of the enclave component remains untampered due to the isolation provided by ARM TrustZone. With this approach we can successfully encourage the adoption of online courses in developing nations with limited internet connectivity and also reduce the load on the server during the assessment.

## 1 Introduction

The extraordinary rise in the number of high quality online technical courses has created a viable medium for students and professionals alike in developing nations to enhance or learn new skills. These courses use an online assessment mechanism to deliver quizzes and tests to users, which are evaluated and scored in an automated fashion by servers maintained by the course provider. The online assessment mechanism usually works in the following way. First the user logs in to the server of the course provider and requests to take a quiz. The user may be using a desktop computer or a mobile device.

In this paper, we focus on mobile devices as the use of mobile devices for online access is prevalent in developing countries where smartphones are the primary online access device for most people. The server then delivers the quiz questions to the mobile device of the user one-by-one. For each question, the user enters their answer into their mobile device which then sends the answer to the server for evaluation and scoring. The server being under administrative control of the course provider, the integrity of the evaluation mechanism and the quiz scores against tampering by the user is ensured.

As can be readily observed, online courses require the user to be connected to the course provider's server throughout the quiz-taking process. There are two main problems associated with this approach:

(1) The user's device is required to be connected to the server via internet throughout the quiz. In areas with poor or slow internet connectivity, this is a big hurdle. Also it demotivates users without an internet data plan from using online courses.

(2) The user's device communicates with the course provider's server after every question is answered and the server performs the evaluation and scoring. The load on the server could be high particularly when several users are taking quizzes simultaneously or if the evaluation algorithms are complex.

An offline solution, which enables the course app installed on the user's mobile device to perform evaluation and scoring of the user's answers, would address the above problems. But this approach would be vulnerable to tampering by the user that wants to cheat on their quiz. For example, a user can read out the answer key that would be maintained by the course app for evaluation of the user's responses to the quiz and answer the quiz using the answer key. Therefore, in order to ensure the integrity of the evaluation and scoring process, we must isolate the execution of the evaluation and scoring modules against tampering by the user.

In this paper, we describe an approach that uses a hardware-rooted isolation technology called ARM TrustZone to ensure the integrity of the evaluation and scoring modules. Our approach partitions the quiz assessment module in the course app into two components: (1) a trusted *enclave component* that performs the evaluation and scoring, and (2) an untrusted user-facing *GUI component* that displays the quiz questions, collects the user's answers, and transfers the answers to the enclave component for evaluation and scoring. The enclave component runs inside a container created using TrustZone that fully isolates its execution from all other software running on the mobile device, thereby preventing the user from tampering with its execution. ARM TrustZone is supported by CPUs present in most smartphones and other mobile devices in use today making it practical to deploy our approach. We implement our prototype on the HiKey board from LeMaker [6] running the Android operating system. We target the Android platform as it is by far the most common mobile platform in developing nations like India.

The rest of this paper is organized as follows: Section 2 states our attacker model, and problem. Section 3 presents an overview of the technologies used by our approach. Section 4 discusses our design and implementation, Section 5 discusses the limitations of our approach, Section 6 discusses related work, and Section 7 concludes.

## 2 Problem Definition

In this section we discuss our threat model and the problem statement.

### 2.1 Threat model

We assume that the user taking the quiz (who is also the owner of the mobile device) is untrusted and wants to cheat the quiz evaluation and scoring process. The user is free to modify all software not executing in TrustZone isolated containers on the mobile phone including the bootloader, the Android OS kernel, middleware and libraries, and apps including the course app. Also, the user is free to modify OS permissions and can thereby grant herself administrative privileges.

The user does not modify the CPU hardware or microcode, and therefore TrustZone functions according to its stated functional specification. We also do not consider the use of side or covert channels to violate confidentiality.

### 2.2 Problem Statement

We address the problem of ensuring the integrity of evaluation and scoring of offline automated assessment under the attacker model described above. In our approach, the evaluation and scoring of quiz answers is performed on the user's mobile device rather than on servers administered by the course provider. In this scenario the user could tamper with the evaluation and scoring process in three ways: (1) modify the execution of the evaluation and scoring code or change the answer key so that an incorrect answer entered by the user is evaluated as being correct, (2) read out the answer key or quiz questions present in the course app before taking the quiz, and (3) directly modify the quiz score maintained by the course app by writing to the memory locations holding the score. Then, in order to ensure the integrity of the evaluation and scoring process in an offline scenario, we must fulfill the following three requirements: (1) ensure untampered execution of the quiz evaluation and scoring logic, (2) ensure confidentiality and integrity of the answer key and the quiz questions, and (3) ensure integrity of the user's quiz score. In this paper, we describe an approach that uses a hardware-rooted isolation technology called ARM TrustZone to fulfill the three requirements listed above.

## 3 Overview of Technologies Used

In this section, we present an overview of ARM TrustZone and OPTEE which we use in our approach.

### 3.1 ARM TrustZone

TrustZone is hardware-based isolation technology developed by ARM [7]. A CPU supporting TrustZone executes in two modes called the *normal world* and the *secure world*. Also, all hardware resources present on a mobile device are divided into two partitions called the normal world and the secure world. When the CPU executes in secure world it

can access hardware resources of both the secure world and the normal world, but when the CPU executes in the normal world it cannot access secure world hardware resources. The secure world is the more privileged CPU execution mode as it permits software to configure how hardware resources are to be placed into the secure and normal world partitions. The standard use case for TrustZone is to have the normal world execute a commodity operating system, like Linux or Android, and the associated middleware stack and applications. The secure world executes small pieces of sensitive application logic all of which will be supported by a small secure OS kernel, and a set of libraries for performing common functions like cryptography and communication with software executing in the normal world.

### 3.2 OPTEE

OPTEE (Open Source Trusted Execution Environment) provides the base software infrastructure for executing programs in the secure world of the ARM TrustZone. It consists of a small OS kernel, called the *OPTEE_OS* and a set of libraries that can be used by secure programs. OPTEE assumes the OS kernel executing in the normal world is Linux and provides a Linux kernel module to establish IPC channels with programs executing in the secure world via the OPTEE_OS. User processes executing in the normal world can communicate with OPTEE Linux kernel module using a device file in the `/dev` directory. Thus, two-way communication is possible between user programs executing in the normal and secure worlds using the OPTEE IPC mechanism. OPTEE is open source and is developed and maintained by Linaro in accordance with standards established by the GlobalPlatform association [2].

## 4 Design and Implementation

In this section we discuss the design of our secure quiz app and present details of its implementation on the Android platform.

### 4.1 Architecture and Security Analysis

The architecture of our secure quiz app partitions the quiz app into an untrusted GUI component and a trusted enclave component. The enclave component performs the evaluation and scoring, while GUI component displays the quiz questions, collects the user's answers, and transfers the answers to the enclave component for evaluation and scoring. The enclave component executes inside an isolated container provided by ARM TrustZone. The idea of running parts of the application in a hardware backed isolation containers has been described in literature; for example the construct of Secure Enclaves described in [1]. We now describe the data flow between the servers of the course provider, and the GUI component and the enclave component of the course app installed on the user's mobile device, as illustrated in Figure 4.1.

1. The user logs in to the course provider's servers using the GUI component of the course app and requests to take a quiz

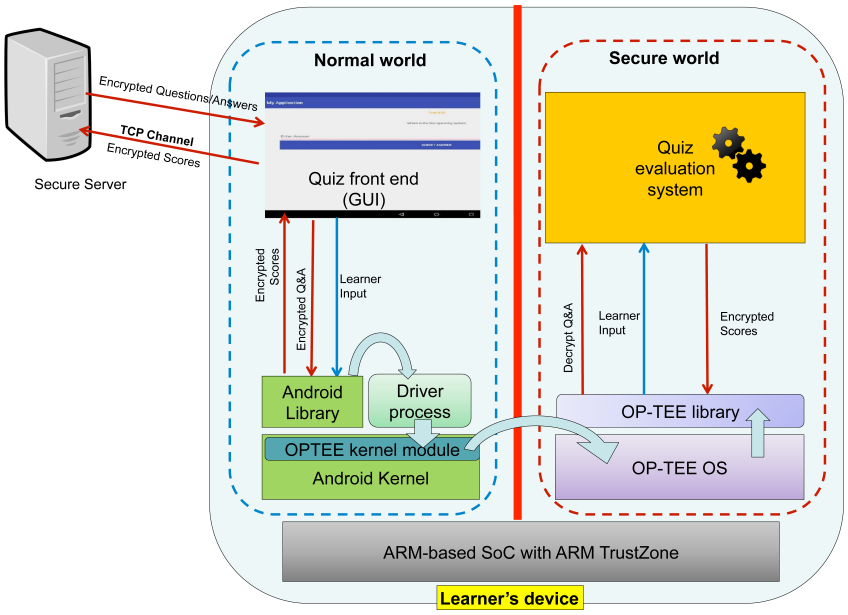


**Fig. 1.** Architecture

2. The course provider transmits the quiz questions and answers to the GUI component. The quiz questions and answers are encrypted and integrity protected by the course provider's servers and transmitted to the GUI component over the network. The keys for decryption and integrity verification are known only to the enclave component. Once this step is finished the user is free to disconnect from the course provider's servers and from the network
3. The GUI component transfers the encrypted and integrity-protected quiz questions and answers to the enclave component using the IPC mechanism of OPTEE
4. The enclave component performs an integrity check on the encrypted quiz questions and answers it receives. If the integrity check passes, it decrypts the quiz questions and answers using its decryption key, else it signals an error to the GUI component
5. The enclave component initializes the quiz score to zero
6. The enclave component transfers the decrypted quiz questions one by one to the GUI component via the IPC mechanism of OPTEE. After transferring the first question, the enclave component starts a timer in the secure world. This timer cannot be affected by modifying any of the timers in the normal world such as the Android system clock
7. The user inputs their answer to each question via the GUI component which transfers the user's answers to the enclave component via the IPC mechanism of OPTEE
8. Upon receiving answer, the enclave component uses the answer key and its evaluation and scoring logic to allocate a score to the answer and adds the score into the running quiz score

9. Once the enclave component receives answers to all the questions, it stops the timer and checks if the total time taken by the user to answer the quiz is less than the time allowed by the course provider. If not, it signals a timeout error to the GUI component
10. The quiz score is encrypted and integrity protected by the enclave component and passed to the GUI component using the IPC mechanism of OPTEE
11. The GUI component transfers the encrypted and integrity protected quiz score to the course provider's servers the next time the user logs in

**Security analysis.** We now discuss how the architecture described above fulfills the requirements described in Section 2. These requirements are: (1) untampered execution of the quiz evaluation and scoring logic, (2) confidentiality and integrity of the answer key and the quiz questions, and (3) integrity of the user's quiz score. Our architecture achieves requirement (1) by executing the evaluation and scoring logic of the quiz app as part of the enclave component whose execution is isolated by ARM TrustZone. To fulfill requirements (2) and (3) we ensure that the plaintext answer key and quiz questions, and the plaintext quiz score are only present in secure world memory that can be read and written only by the enclave component. If we assume that the correct enclave component binary is loaded for execution inside the secure world and the execution integrity of the enclave component, then the isolation guarantees provided by ARM TrustZone in combination with the memory placement of the plaintext answer key and quiz questions, and the quiz score ensures that requirements (2) and (3) are fulfilled by our architecture.

### 4.2 Implementation

We implement our secure quiz app on the HiKey development board manufactured by LeMaker [6]. The HiKey development board has an ARM Cortex-A53 CPU that supports TrustZone. We install the Android OS and OPTEE on the HiKey board. We compile the OPTEE source code using the Android Native Development Kit (NDK) and the `arm-gcc` cross compiler. Using the Android NDK allows us to run OPTEE, which is written in C, on the Android platform. We copy the resulting binaries to the boot image of the HiKey board and modify the boot process so that the OPTEE_OS starts in the secure world before the Android OS boots in the normal world.

OPTEE is designed and implemented to run on a Linux platform rather than on Android. Therefore, the GUI component, which is written as an Android app, cannot directly communicate with the device file exposed by the Linux kernel module of OPTEE. To interface the GUI component to the device file we write a Linux program, called the *driver*, that executes on top of the Android OS kernel. The GUI component communicates with the driver using Android's IPC mechanism. The driver then invokes OPTEE's IPC using the device file to communicate with the enclave component executing in the secure world. Figure 4.1 illustrates this data flow. Figure 4.2 shows screenshots of the execution of the GUI component and the enclave component.

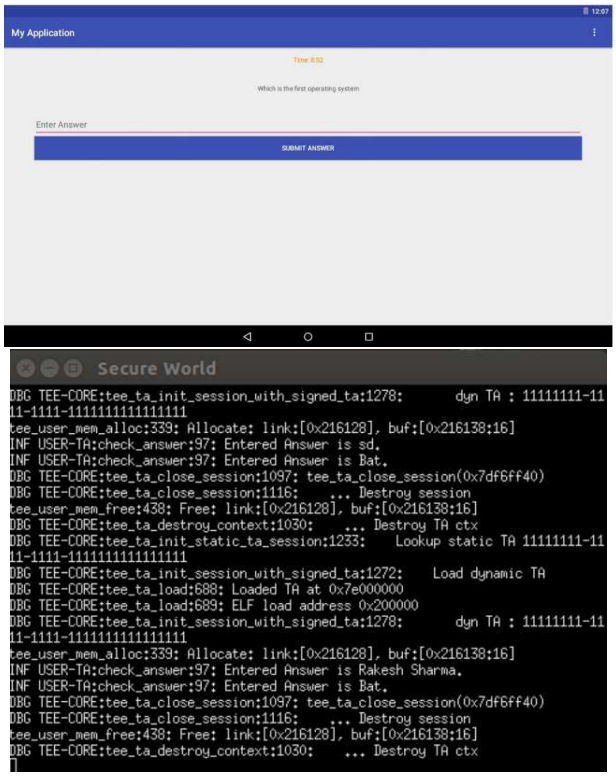


**Fig. 2.** GUI component screenshot (above) and enclave component messages (below)

## 5 Discussion

In this section we discuss some limitations of our present secure quiz app prototype and possible approaches to extend our current prototype to overcome these limitations.

**Authentication of the enclave component to the course provider.** The course provider must be able to verify that the expected enclave component binary was loaded for execution in the secure world on top of a CPU that supports ARM TrustZone. This verification can be performed by sending an attestation containing the hash of the enclave component binary that was loaded into memory and its hardware execution environment to the course provider. Our prototype does not currently have support for such attestation generation and transmission. It is possible to use a hardware-based attestation mechanism similar to that provided on Intel platforms by the Software Guard Extensions (SGX) technology to attest that the correct enclave component has been loaded for execution into secure world memory on a CPU that supports TrustZone [3]. However, ARM TrustZone currently does not support such an attestation mechanism. It may be possible to use an attestation mechanism similar to that proposed by Sailer et al. implemented within the OPTEE_OS kernel to attest that the correct enclave component binary has been loaded into memory [9]. There are two issues with this approach: (1) the mech-

anism requires that the mobile device support a Trusted Platform Module (TPM) chip and many current mobile devices do not, and (2) the mechanism cannot attest whether or not the CPU of the mobile device supports ARM TrustZone and whether or not the enclave component was loaded into secure world memory. It may also be possible to use a software-based attestation mechanism similar to that proposed by Seshadri et al. to generate the required attestation [10]. Such a mechanism can fully attest that the enclave component has been loaded for execution into secure world memory on a CPU that supports TrustZone. However, using such a mechanism with a remote verifier (as is the case here) will require it to be modified to use a TPM chip for timing as proposed by Kovah et al. [5].

**Initial key exchange between the enclave component and the course provider.** Our current prototype embeds the encryption and integrity check keys into the enclave component binary. This is insecure since a malicious user can perform a binary dump of the enclave component binary to recover these keys and thereby invalidate the security guarantees offered by prototype. A better approach is to have the course provider perform a key exchange to set up the encryption and integrity check keys with the enclave component once it has been able to verify that the expected enclave component binary was loaded for execution in the secure world on top of a CPU that supports ARM TrustZone. In this scenario the course provider could embed the course provider's public key into the enclave component binary. A verified attestation will inform the course provider that an enclave component with the expected public key has been loaded into secure world memory on a given mobile device. The initial key exchange can then be performed in an authenticated manner using protocols like the Diffie-Hellman key exchange protocol.

**Secure offline key storage.** Our current prototype stores the encryption and integrity check keys in plaintext on the persistent storage of the mobile device. It is easy to read out these keys from the persistent storage. We could use secure storage mechanisms such as sealed storage to ensure that the keys are not stored in plaintext. However, using sealed storage requires the mobile device to support a TPM chip and many current mobile devices do not.

## 6 Related Work

Assessments in online courses, particularly MOOCs, are usually automated. A 2013 survey conducted by The Chronicle of Higher Education found that 74% of MOOC faculty use automated grading. Of these, 67.1% found the technique "very reliable" and 30.1% found it "somewhat reliable" [4]. Students typically upload their solutions to a server, where they are automatically evaluated without revealing the solutions. For offline evaluation, answers can be revealed after students upload their submissions. However, it is possible for dishonest students to undermine such a system by creating a "harvester" account to submit a guessed solution, and then submitting the correct answers using a separate "master" account. A technique for detecting such a strategy has been proposed [8], but to the best of our knowledge, no method for evaluating answers securely in an offline manner without revealing the answers themselves has been proposed prior to this work.

## 7 Conclusion

We present a technique that permits a user of an online course to take the quiz components of the course while being disconnected from the course provider's servers. We use hardware-based isolation containers to ensure the integrity of the evaluation and scoring of the quiz even when the corresponding logic executes on a mobile device that is under the control of the quiz-taking user that wants to cheat on the quiz. Our technique allows users with limited internet connectivity to take advantage of high-quality online course offerings while simultaneously reducing the load on the course provider's servers during the quiz components. We hope that our technique would benefit students of developing nations where internet connectivity is limited or expensive.

## References


1. Manuel Costa, Sriram Rajamani, Mark Russinovich, Kapil Vaswani, Orion Hodson, and Marcus Peinado. Secure computation interfaces. Technical report, August 2015.
2. GlobalPlatform. *TEE Client API Specification v1.0*, July 2010.
3. Intel Corp. *Intel64 and IA-32 Architectures Software Developers Manual*, July 2016.
4. S Kolowich. The professors who make the moocs. the chronicle of higher education, p. 1. March 2013.
5. Xeno Kovah, Corey Kallenberg, Chris Weathers, Amy Herzog, Matthew Albin, and John Butterworth. New results for timing-based attestation. In *2012 IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2012.
6. LeMaker. Hikey board. Technical report, August 2015.
7. ARM Limited. Arm trustzone. Technical report, December 2008.
8. Curtis G. Northcutt and Isaac L. Chuang Andrew D. Ho. Detecting and preventing "multiple-account" cheating in massive open online courses, arxiv:1508.05699 [cs.cy].
9. Reiner Sailer, Xiaolan Zhang, Trent Jaeger, and Leendert van Doorn. Design and implementation of a tcg-based integrity measurement architecture. In *Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13*. USENIX Association, 2004.
10. Arvind Seshadri. *A Software Primitive for Externally-verifiable Untampered Execution and its Applications to Securing Computing Systems*. PhD thesis, Carnegie Mellon University, 2009.