
Yingjun Zhang, Shijun Zhao, Yu Qin, Bo Yang and Dengguo Feng
Trusted Computing and Information Assurance Laboratory
Institute of Software, Chinese Academy of Sciences
Beijing, China
Email: zhangyingjun@tca.iscas.ac.cn

Abstract—We give a detail analysis of the security issues when using mobile devices as a substitution of dedicated hardware tokens in two-factor authentication (2FA) schemes and propose TrustTokenF, a generic security framework for mobile 2FA schemes, which provides comparable security assurance to dedicated hardware tokens, and is more flexible for token management. We first illustrate how to leverage the Trusted Execution Environment (TEE) based on ARM TrustZone to provide essential security features for mobile 2FA applications, i.e., runtime isolated execution and trusted user interaction, which resist software attackers who even compromise the entire mobile OS. We also use the SRAM Physical Unclonable Functions (PUFs) to provide persistent secure storage for the authentication secrets, which achieves both high-level security and low cost. Based on these security features, we design a series of secure protocols for token deployment, migration and device key updating. We also introduce TPM2.0 policy-based authorization mechanism to enhance the security of the interface from outside world into the trusted tokens. Finally, we implement the prototype system on real TrustZone-enabled hardware. The experiment results show that TrustTokenF is secure, flexible, economical and efficient for mobile 2FA applications.

Keywords—two-factor authentication; Trusted Execution Environment; TrustZone; SRAM PUF; trusted user interaction; TPM2.0 policy-based authorization.

I. INTRODUCTION

Two-factor authentication (2FA) is increasingly used to strengthen the security of password-based authentication schemes. Dedicated hardware tokens like smart cards, as a secondary authentication factor, can provide high-level security due to their strict physical separation from external untrusted environment, both for runtime execution and persistent storage. However, the high cost of secure hardware and the burden of carrying multiple tokens, additional interface equipments (USB cables or NFC readers) makes them less popular to both service providers and users.

On the other hand, mobile 2FA schemes integrate different authentication services into one single mobile device and require no additional secure hardware, thus achieving both flexibility and low cost, hence are commonly seen as an ideal substitution of dedicated hardware tokens.

The most popular mobile 2FAs are SMS-based schemes. The authentication server generates an One Time Password (OTP), and sends it to the mobile device via Short Messaging Service (SMS). These schemes have been widely deployed by global Internet service providers like Google, Facebook, Twitter and large banks. The OTPs could also be generated on client side, like the Google Authenticator app [1], which are currently used by many third-party service providers.

However, modern mobile OSes, as the Trusted Computing Base (TCB) that mobile 2FA applications rely on, are so complex that it is difficult to ensure the absence of vulnerabilities. Considering the current trend in privilege escalation exploits for mobile platforms [2], compromise of mobile OSes has become commonplace. Root attackers could easily intercept the communication channel like SMS, or corrupt 2FA applications like Google Authenticator for malicious authentications. Various attacks to mobile 2FA applications have been reported recently [3], indicating the increasingly serious security challenges for mobile 2FA schemes.

To secure mobile services despite mobile OS compromise, researchers have proposed using Trusted Execution Environment (TEE) to provide isolated execution for applications. Several mobile payment services have leveraged dedicated TEEs based on SIM card secure element (SE) to store their credentials and perform sensitive operations, like Google Wallet [4] and PayPass. Unfortunately, SEs still face some insurmountable security issues. On the one hand, SEs delegate the access control function to mobile OSes due to the limited memory space, which can be bypassed by root attackers, leading to illegal access. On the other hand, lacking control ability over I/O peripherals, SEs cannot build a trusted user interaction path, which may lead to PIN-stealing, spoofing and relay attacks. Such attacks to Google Wallet have been reported in [6].

TrustZone [7], another kind of TEE based on security extensions to ARM System-On-Chip (SoC), with full control ability over all system resources including processor, memory and peripherals, has the potential to realize some advanced security features for mobile 2FA, like thorough authorization management for authentication secrets and trusted user interaction. However, unlike dedicated hardware tokens, TrustZone-based TEE doesn’t provide secure storage features as it shares the external non-volatile memory with mobile OS. Current trusted systems and secure services based on TrustZone usually assume the availability of a unique device key only accessible in TEE for secure storage [22], [23], [5]. Unfortunately, such secure device keys are not always available on mobile devices [8], which is necessary for mobile 2FA schemes to prevent cloning attackers from reconstructing the authentication secrets on other devices to impersonate the victims.
In this paper, we present TrustToken Framework (TrustTokenF), a generic security framework for mobile 2FA schemes, which is flexible, low-cost and efficiency without the expense of security. It leverages TrustZone and SRAM PUF, which are both commonly available on mobile devices to provide the essential security properties for mobile 2FA applications, i.e., runtime isolated execution, trusted user interaction and persistent secure storage.

In summary, we make the following contributions in the paper.

- TrustTokenF leverages the TEE provided by TrustZone running on powerful ARM processors to perform sensitive computations, leading to a more efficient implementation of authentication algorithms and obviating the need for any additional secure hardware like smart cards. Also, we are the first to introduce TPM2.0 policy authorization in mobile 2FA schemes.
- We build a lightweight trusted user I/O path in TrustZone separated from mobile OS. The trusted path is dynamically configurable according to specific program contexts. To the best of our knowledge, we are the first to give the implementation details on how to build a configurable trusted user I/O path in TrustZone.
- We build a root of trust for persistent secure storage by using SRAM PUF to extract a device-specific storage key. This approach requires no additional secure non-volatile memory like eFuse, which is expensive and difficult to perform key updating.
- We implement the prototype system on real TrustZone-enabled hardware. We provide a standard PKCS#11 interface by porting OpenCryptoki into TrustZone to keep compatible with existing authentication services.

II. BACKGROUND

A. ARM TrustZone

TrustZone[7] is a system-wide security technology which provides hardware-level isolation on ARM System-on-Chips (SoCs). It creates two isolated execution domain: the normal world, where untrusted code runs, and the secure world, where security-critical code runs. The secure world has a higher privilege with permissions to access all the resources of the normal world including memory, registers and peripherals, but not vice versa. A special processor bit, the NS bit, indicates which world the processor is running in. This bit is sent over the memory bus and some I/O buses for peripherals. For worlds switching, a special monitor mode exists in the secure world, as a gatekeeper between the two worlds. A special processor instruction, Secure Monitor Call (smc) need to be called to enter the monitor mode, which performs a secure context switch into the secure world. By configuring the TrustZone Aware Interrupt Controller (TZIC) and some interrupt related registers, hardware interrupts can be directly handled in the monitor mode, thus enabling flexible routing of hardware interrupts to either world. By default, TZIC uses Fast Interrupt (FIQ) as secure interrupt and uses Regular Interrupt (IRQ) as non-secure interrupt. Another important TrustZone component is TrustZone Protection Controller (TZPC), who is responsible for configuring peripherals as secure or normal and preventing non-secure access to secure peripherals. In order to provide a friendly interface to access the secure services in the secure world, the GlobalPlatform consortium formulates the TEE client API specification [9].

B. Physical Unclonable Function

Physical unclonable functions are a promising technology which enable unique device authentication binding software to hardware devices and secure storage of cryptographic secrets. A PUF is a noisy function defined via a physical object. When received a challenge, PUF generates a noisy response depending on both the challenge and the intrinsic manufacturing variations, which cannot be controlled even by the manufactures. PUFs should achieve both high robustness and uniqueness. The robustness means that when received the same challenge for many times, it should produce responses with a limited amount of noise. The uniqueness means that responses of different PUFs to the same challenge should be independent. The two properties together enable every PUF to generate a stable and unique key in case a fuzzy extractor [10] is used to eliminate the noise. There are two steps in a fuzzy extractor algorithm, generate and reproduce. The generate step uses PUF response and a selected secret key to produce a public helper data. The reproduce step uses the helper data and a noisy response to reconstruct the key. Note that the secret key is selected by manufactures so a new key is easy to be deployed by replacing the public helper data. This feature provides a flexible key updating approach with no need to replace the PUF component.

The PUF-generated keys achieve high-level hardware security since they are dynamically extracted from a physical system rather than statically stored in non-volatile memory. They are also cost-effective since they are the results of a preexisting manufacturing process of regular physical components, requiring no additional high-cost non-volatile memory for key storage. In this paper, we use SRAM PUFs [11], which take SRAM’s physical address as the challenge and return its powered up values as the response. SRAM PUFs reduce hardware attack interfaces since the extracted keys are not present when devices are powered off.

III. ADVERSARY MODEL AND DESIGN PROPERTIES

In this section, we describe the adversary model and the desired secure properties of TrustTokenF.

A. Adversary Model

We give comprehensive considerations to our framework design assuming a very sophisticated adversary capable of launching various software attacks. The adversary has known the victim’s password, i.e., the first authentication factor and has full control ability over the OSes on the victim’s host and mobile devices and the communication channels among the host device, the mobile device and the server. The adversary cannot compromise the TEE of the victim’s mobile device. We don’t consider complex physical attacks like side-channel attacks, which are out of the scope of the protection provided by TrustZone. We don’t address denial-of-service attacks.
B. Design Properties

We consider TrustTokenF is secure if our presented adversary cannot be authenticated in the name of the victim, either via the victim’s mobile device or other ones under full physical control of the adversary. To achieve this goal, the following security properties need to be provided.

- **Device binding.** The authentication key can only be derived on a certain mobile device, so that any cloning attackers cannot impersonate normal users by using other devices.

- **App separation.** The authentication programs should be separated from the mobile OS to prevent various malware in normal world. Also, these programs should be separated from each other, both at runtime and for persistent stored data, so that a malicious or errant program cannot steal or tamper the resources of other programs in TEE, like authentication keys or runtime memory space.

- **Secure token interface.** The interface used for interaction between the tokens and outside entities must prevent leakage or manipulation of any token’s private information, which requires a secure authorization mechanism for the tokens.

- **Trusted user I/O.** The trusted user interaction of our framework must ensure that (1) the user input path is separated from normal world for reliable pin entering and user consent, (2) the authentication details are securely displayed for the user to judge if a certain authentication is intended by himself, (3) there exists a local attestation mechanism to prove the security status of current display and the token to the users, so that they won’t be cheated into entering their sensitive data to spoofing malware.

IV. **System Design**

The main motivation to design our framework is to provide comparable security assurance to hardware tokens for 2FA applications under the adversary model presented above while maximizing the compatibility with existing authentication services. And the adversary should be concerned through the whole life cycle of the tokens because the OS may have been compromised before token deployed. Adverse effects from compromised host device should also be taken into consideration. Our design should be lightweight to minimize the code size of the TCB (trusted computing base). As described above, the SMS based 2FA schemes may be easily compromised and will generate extra SMS expense to mobile network operators. We use a generic challenge/response 2FA protocol in our design, where the server sends a random challenge to the client, the client performs the authentication algorithm using his private authentication key and returns corresponding response.

A. TrustZone based Architecture Design

**Overview.** Architecture of TrustTokenF is illustrated in Figure 1. The standard PKCS#11 interface and token-specific interfaces are both supported, which are sent to secure world via TEE client API. An authentication token consists of four parts, the client APPs in normal world who receive authentication requests from host computers or local users, the authentication algorithm represented by TKService in secure world, an authentication key and an associated authorization policy. Token Manager is responsible for token access, deployment, migration and performs the authorization validation. I/O Switcher is responsible for building and disabling the trusted user I/O path. Storage Manager implements a seal/unseal function to provide token-specific secure storage, and is the only entity capable of reaching the device keys. The fuzzy extractor of PUF and device keys reside on-chip SRAM protected by TrustZone. The core components are detailed as follows.

**SRAM PUF.** We design the fuzzy extractor of SRAM PUF to extract a robust and device-specific ‘primary seed’ $s$ from the start up value $r$ of on-chip SRAM and public helper data $h$. Then a key derivation function (KDF) is performed using $s$ to obtain device keys, comprising a storage key $srk$ and a pair of identity keys $(sk, pk)$. The physical unclonable property of SRAM PUF ensures that the device keys cannot be derived from other devices using the same helper data.

1) **Generate.** This procedure is performed during the production process of mobile devices to generate the helper data and device keys.

   1) The manufacture select a large secret $s$ as a ‘primary seed’ for every device.
   2) A code value $v$ is extracted by encoding $s$ using BCH error correction algorithm, $v = BCH_{ENC}(s)$.
   3) Create public helper data $h, h = v \oplus r$.
   4) Derive device identity key pairs $pk, sk$ from $s$. Generate a device certificate $dcert$ by signing $pk$.
   5) Store $h$ and $dcert$ in the device’s external non-violate memory

1) **Reprocedure.** The device keys can be reconstructed every time the SRAM is initialized, i.e., when device is powered up. The steps are as follows.

   1) Record the start up values of SRAM $r'$.
   2) Compute a code value $v' = h \oplus r'$, $v'$ is a noisy value of $v$.
   3) Send $v'$ to BCH decoder to eliminate noisy and reproduce $s, s = BCH_{DEC}(v')$.
   4) Re-derive the device keys, $(pk, sk) = KDF_s('identity'), srk = KDF_s('storage')$. 

![Fig. 1. Architecture of TrustTokenF.](image-url)
**Seal/Unseal Function.** The seal/unseal function provides the security property of app separation described in section III. In the seal procedure, a token-specific key \( tsk \) is derived using \( srk \), token’s authentication code TKService and the corresponding policy, \( tsk = KDF_{srk}(Hash(TKService, policy, c)) \). This key is used to encrypt the corresponding authentication key. The unseal function performs the reverse procedure, \( tsk \) is derived to decrypt the authentication keys. The monotonic counter value \( c \) is used to prevent downgrading attackers, who roll back a token to an old version with vulnerable algorithms or defective policy. Each time a token is updated, \( c \) is increased by 1, so \( tsk \) can’t be derived correctly with an old version blob since \( c \) has been changed. The seal/unseal function and SRAM PUF ensure that the authentication key can only be derived from specific app with specific policy on specific mobile device.

**Trusted User Interaction Path.** The trusted user interaction path creates a reliable communication channel separated from the normal world. Interrupt handlers and I/O drivers, e.g., touch screen, keyboard, graphics card, are implemented in secure world. A trusted user path based on TrustZone need to satisfy two basic features. First, related interrupts should be routed to secure world and must not be masked or changed by normal world. Second, related I/O peripherals need to be configured as secure by setting related bits in TrustZone Protection Controllers to prevent illegal access from normal world.

In our design, we use the default secure FIQ as the interrupt source for the trusted user path, which is built as follows. During boot stage, related I/O interrupts are configured as secure in TZIC, ensuring that they can only be configured in secure world. When a trusted path is needed, I/O Switcher sets the F bit in CPSR (Current Program Status Register) to 0 to enable FIQ interrupt. Then the FW bit in SCR (Secure Configuration Register) is cleared ensuring the F bit in CPSR can’t be modified by normal world. The FW bit can only be changed in secure world. These two steps ensure that FIQ interrupts can’t be masked by normal world. Then I/O Switcher sets the FIQ bit in SCR to 1 to enforce an FIQ interrupt routed to the monitor mode. Finally related I/O peripherals are configured as secure in TZPC (TrustZone Protection Controller) ensuring that the peripherals can’t be illegally accessed by normal world.

The trusted user path should be dynamically configurable according to the specific program context, i.e., when interactions with local users are secure-critical, I/O peripherals and related interrupts are exclusively handled by secure world. Other times they are configured as shared by both worlds to support normal usages. Our design achieves this feature because all related configurations are software implemented by I/O Switcher in secure world. To close the trusted path, I/O switcher first clears the peripherals’ I/O buffers to prevent data leakage to normal world, then the FIQ bit in SCR is cleared thus routing I/O interrupts to normal world. Finally related I/O peripherals are configured as shared by both worlds.

**Policy Management.** To properly control access to the tokens, it is not enough to just isolate them from normal world, secure authorization should be provided to ensure that illegal access won’t be permitted. There are different levels of security requirements for authentication tokens in specific application scenarios. For online bank transactions, the authorization policy should be strict including e.g., a correct pin code entering and a correct measurement of platform states. While the authentication of a public transportation card need to be fast enough requiring no authorization policy. In order to provide a reasonable tradeoff between security, usability and efficiency, authorization policy for the tokens are freely determined by service providers in our scheme.

We introduce the thorough and flexible authorization mechanism of the Trusted Platform Module (TPM) 2.0 specification [12]. In TPM 2.0, every protected entity is allowed to be associated with an authorization policy, named authPolicy. The policy may be arbitrarily complex, but the authPolicy is reduced to a single unique hash value of all the policy assertions. Before using a protected key, the user need to send some policy assertions to TPM. TPM checks the policy assertions one by one and recomputes authPolicy. If every assertion is correct and the computing result matches authPolicy, meaning that every policy rule is satisfied and the authorization policy is exactly the one the protected key needs, the access is allowed. Each time a user sends a policy assertion to TPM, an interaction with TPM is required, meaning a world switch in our scheme. To reduce the switch counts, the caller sends all the policy assertions to secure world together. The associated authPolicy is imported with the token by service providers during deployment process. The Policy Manager is deployed in TrustZone secure world ensuring that the policy validation won’t be compromised or bypassed by malicious requesters.

An example of the authorization policy is illustrated in Figure 2. The policy contains eight assertions. Once received an authentication request with these authorization policy assertions, Token Manager first checks whether the hash values of TEE, mobile OS image and current TKService, TKClient match those presented in the assertions. Then it calls I/O Switcher to activate the trusted user path to obtain a pin code and checks its correctness. The command code assertion is used to limit the key’s usage. The policy also asserts the expiration date and the pin failure times.

### B. Protocol Design

Based on TrustTokenF architecture, we design a series of protocols for token management, which achieve good user experience. Five participants are involved: the Service Provider (SP), the Host Device (HD), the Mobile Device (MD), the MD’s TEE, and the Mobile Device Manufature (MDM).

**Assumptions.** For mobile 2FA schemes, a secure enrollment protocol is needed where SPs associate a user’s identity to the right mobile device. Note that for dedicated security tokens, the user-to-device binding happens within the service provider premises, before the token is shipped to the customer.
When mobile devices replace such tokens, SPs must establish the binding after the user has registered for the service. The purpose of the binding is to establish an authenticated channel between SPs and TEEs for secure token transferring. However, there are not any pre-shared secrets between SPs and TEEs before deployment, means that a TEE cannot prove that it belongs to a specific user, so the binding cannot be established securely. How to implement a secure enrollment protocol is an open problem. A method is proposed to build a secure out-of-band (OOB) channel requiring hardware changes to MD’s baseband architecture and collaboration with mobile network operators [13], which is impractical.

We assume the existence of a secure OOB channel during token deployment. With this assumption, SPs could bind the user with the right authentication key and the token will be securely transferred to the right TEE. TEE then re-encrypts the token’s code and policy using srk, then seals the authentication key by tsk as described above.

We also assume there is a pre-shared background image between the user and TEE for trust display, which is securely initialized when a user purchases his mobile device. The image serves as a security status indicator for the user to identify spoofing and phishing malware. However, without any pre-shared secret between the user and TEE, the initialization of the image may be intercepted or tampered. Our assumption is necessary from this point of view.

Authentication. The protocol details are illustrated in Figure 3. The user first launches the authentication session on HD, whose session id is sid. SP verifies the username and password, and sends a transaction-specific challenge nonce to MD. The connect between SP and MD can be various, like indirect connect via HD using USB, NFC or through the Internet using push notification feature [14]. The Internet method achieves better user experience with no need for physical connecting between HD and MD. So sid is also sent to bind the two authentication sessions of HD and MD with SP. The counter ctru,ctr_id are used to prevent cloning attacks, if an authentication procedure is passed using another illegal MD, the values of them will be different because only ctru will increase. The counter also ensures the freshness of the token state. MD generates the policy assertions and sends them to TEE. TEE first decrypts the authpolicy and checks whether it is satisfied, then decrypts the authentication algorithm algo for the token and unseals the authentication key usk, performs the authentication algorithm and returns the response to SP. If the signature is verified, SP notifies HD in the authentication session sid that the 2FA authentication for user u is passed. A pin code or an explicit user consent (a ‘yes’ button press) can be set in the policy to prevent relay attacks using our trusted I/O path. The picture img is used to authenticate the TEE to the user.

Migration. In case the user purchases a new mobile device, a token migration process is needed. Performing another deployment operation is impractical because the assumption of a secure out-of-band channel may be expensive or inconvenient. Since the two devices are both physically controlled by the user, the trusted user I/O path can be used to create a shared migration seed sec. The new device is denoted as NMD. The protocol procedures are detailed in Figure 4. Note that all communications between TEEs and SP are transferred by MDs, which are omitted in Figure 4. First TEE generates sec, which is transferred to NTEE by the user. The trusted user I/O path ensures that sec won’t be intercepted by untrusted OSes on each MD. The picture img will prevent impersonation of the importing app on NMD. Based on sec, the two TEEs generate the same migration keys mk,mk′ and migration requests req,req′. NTEE also generates its authentication key pairs (apk′,usk′), and sends the public portion apk′ to SP encrypted by mk′. Since SP has established the binding between the user and TEE, mk can be securely transferred to SP from TEE. SP first authenticates TEE using the user’s password and the origin authentication key apk, then authenticates NTEE using mk received from TEE. If the two authentications are validated, SP binds the user with apk′, sends the token’s code and the associated policy to NTEE, encrypted by upk′. NTEE decrypts them and stores them on NMD. Once completed, the user can be authenticated by either device. Our migration protocol eliminates the need of physically visits to SP for the user. The user interaction overhead is accepted as the migration operations are not often.

![Fig. 3. Procedures for the authentication protocol.](image-url)

![Fig. 4. Procedures for the migration protocol.](image-url)
Device Key Update. A significant advantage of secure storage based on SRAM PUF lies in the flexibility for device key updating, while a device key protected by secure non-volatile memory like eFuse can't be changed since it is one-time programmable. Upon receiving a key updating request, MDM first authenticates the device using the device certificate, then re-performs the generate procedure, sends the new helper data and new certificate to TEE. The next time MD reboots, TEE will generate new device keys. Tokens are re-encrypted using newly derived storage keys. Finally previous helper data and public key certificate are removed. The update process also enforces a physical user consent via the trusted user I/O path.

C. Discussion

We further discuss some design details concerned with mobile 2FA schemes to strengthen the security of our scheme.

2FA Deactivation. Current 2FA schemes typically provide an explicit deactivation mechanism. For browser based login verification on PC, a malware can wait until a user logs in, hijack the http session and disable 2FA in the name of the user [3]. The deactivation protocol has to be improved by requiring a 2FA authentication again though the user has logged in. Any secure-sensitive settings which can be configured by logged-in users should require an additional 2FA authentication.

Runtime Exploitations. The authorization policy based on the hash value of current process only ensure a clear state when the application is loaded. However, runtime exploitations such as code injection, code reuse attacks, return-oriented programming [15] may still perform malicious operations in the normal application’s address space, although they can’t compromise the tokens in secure world. Under this case, the authentication keys may be illegally access if an explicit user consent is not required. Antivirus software and TrustZone based Integrity Measurement Architecture (TIMA) [16] for dynamic kernel protection are complemented to our design.

V. Security Analysis

In this section, we discuss the security of TrustTokenF. We list all the possible software attacks aiming to broke the security property defined in section III.

TKService Compromise. The deployed tokens runs in the isolated execution environment provided by TrustZone, which can’t be broken by software attacks. An adversary controlling the whole normal OS cannot compromise these tokens.

Cloning Attacks. Cloning attackers aims to steal the authentication keys and restore them on their own devices to impersonate the victims. These attacks may happen in two situations. During the enrollment process before deployment, attackers may induce SP to bind their TEEs with the victim. This can be prevented by a secure out-of-band communication channel as described in section IV. After deployment, the attacker may copy the persistent stored authentication keys to his own device and try to restore them. The physical unclonable feature of SRAM PUF ensures the restoration can’t be success. The freshness counter used in the authentication protocol also helps to detect cloning attacks.

Relay Attacks. Relay attackers don’t have to compromise any applications or OS, nor do they need to obtain the authentication keys, they just monitor the communication interface to relay the messages between SP and victim’s TEE. The trusted user path ensures that any authentication attempts must be physically confirmed by the user, which cannot be tampered, emulated or masked by normal world. The thorough authorization policy also restricts accesses to deployed tokens.

Spoofing Attacks. The pre-shared image serving as a security indicator helps the user to distinguish a forged pin entering window created by malware. We argue that even without this image, these attacks won’t weaken the security for pin entering. First, the attackers can’t block or overlap the display from secure world as long as they send an authentication request to TEE, because I/O Switcher will activate the trusted user I/O path before I/O transactions. Second, the attackers cannot emulate a keystroke or screen touching interrupt event to confirm the authentication by writing forged data into peripherals’ I/O buffers, which are configured as secure by TrustZone. So they still cannot complete an authentication process without a physical user confirm although they has got the pin code.

VI. Implementations

We present our implementation on a TrustZone-enabled development board, Zynq-7000 AP SoC Evaluation Kit, which is equipped with dual ARM Cortex-A9 Core, 1GB external DDR3 RAM, and 256KB on-chip SRAM. The development board runs a linux 3.8 kernel in normal world and a secure OS, Open Virtuization SierraTEE [17], which is compliant with Global Platform TEE Specifications. SierraTEE provides basic OS-like features, including a simply Fat32 FileSystem support and a dynamically loading support for ELF (executable and linkable format) files, which provides runtime environment for the applications and enables flexible deployment for the tokens. It also provides a simplified C standard library. We use an Multi Media Card (MMC) as the insecure persistent storage. Our implementation is lightweight adding only 4.5K lines of source code to SierraTEE.

SRAM PUF. In Zynq-7000 development board, the BootRom loaded the First Stage BootLoader (FSBL) into 256KB on-chip SRAM once powered on. So we can’t read the init values of the SRAM. We use an ALTERA EP2CSF256 Core Board as the SRAM PUF generator. The SRAM init values of this board are transmitted to Zynq-7000 board via an UART port during the boot process and is stored in a protected memory region in secure world.

We port an open source BCH code [19] into SierraTEE. The BCH code’s parameters are [511,19,239], mean that a noisy 511 bits whose errors are less than [239/2=119] bits can obtain 19 correct bits without noisy. In case the primary seed is 256bits, the BCH code is needed to run [256/19=14] times requiring [14*511=7154] bits SRAM values. We modified SierraTEE to clear the primary seed once device keys are derived.

Crypto Libraries. We implement the KDF algorithm for asymmetric keys by porting the RSAREF library, and the one for symmetric keys according to the _cpri_KDFa function defined in TPM 2.0. We also port an OpenSSL-0.9.8y library for generic cryptography supports and to establish secure channels between trusted tokens and outside entities.
Token Manager. We implement our TrustZone based PKCS#11 library by modifying openCryptoki [21], an open source PKCS#11 standard library for linux. The library is modified to wrap all the PKCS#11 command into GP client API, and transfer them to the secure world using smc instruction. The Slot Manager daemon process of openCryptoki is ported into SierraTEE as our TokenManager. The caller could use the Data Object defined in PKCS#11 specification to transfer the authorization policy since PKCS#11 doesn’t provide corresponding interfaces. We add TPM2.0 policy based authorization to TokenManager. However, we didn’t find any open source TPM 2.0 software emulator written in C language. So we build a prototype system to support all the policy types mentioned in our paper. Other policy types can be added to our system which are detailed in TPM 2.0 specification.

Trusted User I/O Path. We implement a prototype trusted user path using an UART port on Zynq-7000 as the I/O interface, which can be configured as secure only or shared by both worlds. We used a PC to emulate the I/O peripherals connecting to Zynq-7000 via the UART port. We use Tera Term, a terminal tool for serial port debugging on PC to connecting to Zynq-7000 via the UART port. We use Tera Term, a terminal tool for serial port debugging on PC to transfer the I/O data between them. We implement a simple UART driver and an UART FIQ handler in the secure world. When a keyboard stroke is triggered on PC, Tera Term sends the stroked key and stores the related char in a text file. When sensitive data needs to be displayed to the user, the UART driver sends the data out via the UART port and Tera Term displays it on PC. We use the UART port to intercept any possible output data from the secure world. We perform several normal linux operations in the normal worlds and some authentication procedures including user I/O in secure world under two situations. When the trusted path is built, the malware doesn’t get any UART I/O data and any read attempts to the UART port fails. When the trusted path is configured closed, the malware records all the keyboard inputs and successfully intercepts some sensitive outputs from within secure world.

Though we don’t use real I/O peripherals like keyboard, touchscreen, we confirm that TrustZone has the ability to build a trusted user I/O path because they are treated as the same type of I/O peripherals with the UART port by CPU. The method to build a trusted user path for UART port can be applied to these peripherals straightforwardly.

Performance Evaluation We evaluate the computation performance using three hardware secure token versus using TrustTokenF. We implement a PKCS#11 compliant software token in SierraTEE and choose MARX CrypToken MX2048, VASCO DigiPass and a TPM 1.2 hardware chip produced by National Semiconductor as the hardware tokens. An RSA signing operation is performed on each device for 100 times. The experiment result is shown in Table 2.

Our software implemented token achieves high computation performance compared with other hardware tokens. The TPM chip we use only runs at 33M Hz, making it impractical for situations with demanding performance requirements. The other two dedicated tokens are even slower.

The extra operations during the authentication procedure also need to be evaluated, i.e., the world switch, the policy validation and the unseal function. We evaluate the world switching time by deploying an empty application in secure world. We denote the policy validation as three hash operations using SHA-256, and the unseal function as an AES decryption. The result shows that the total extra time is about 14 milliseconds, which is much less than the signing time of three hardware tokens.

VIII. Related Work

Current TEE-based security solutions focus on the runtime security provided by the isolated execution environment. The drawbacks about lacking a root of trust for persistent secure storage and the potential to build a trusted user interaction path have been overlooked.

Nokia’s On-board Credentials system (Obc) [22] provides an open security framework for credential deployment and storage based on the M-Shield secure hardware, which assumes the existence of a device key as the root of trust. Marforio et al. [23] propose a location based mobile 2FA architecture for the point of sale transactions. They build a secure GPS module in TrustZone’s secure world to generate a trusted location statement signed by the assumed device key for user authentication. However, they didn’t resolve the fraudulent issues in case the

<table>
<thead>
<tr>
<th>TABLE I. TCB SIZE OF TRUSTTOKENF.</th>
<th>TABLE II. PERFORMANCE EVALUATION (IN MS). AVG. OF 100 RUNS.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Code(LOC):</strong></td>
<td><strong>Average Time (ms)</strong></td>
</tr>
<tr>
<td>Fuzzy Extractor</td>
<td>Vasaco</td>
</tr>
<tr>
<td>UART driver</td>
<td>0.2K</td>
</tr>
<tr>
<td>I/O Switcher</td>
<td>0.1K</td>
</tr>
<tr>
<td>Token Manager</td>
<td>0.3K</td>
</tr>
<tr>
<td>Seal/Unseal</td>
<td>1.9K</td>
</tr>
<tr>
<td>KDF Function</td>
<td>1.7K</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>4.5K</td>
</tr>
</tbody>
</table>

Trusted User I/O Path. We implement a prototype trusted user path using an UART port on Zynq-7000 as the I/O interface, which can be configured as secure only or shared by both worlds. We used a PC to emulate the I/O peripherals connecting to Zynq-7000 via the UART port. We use Tera Term, a terminal tool for serial port debugging on PC to transfer the I/O data between them. We implement a simple UART driver and an UART FIQ handler in the secure world. When a keyboard stroke is triggered on PC, Tera Term sends it to the development board to generate an FIQ interrupt. The UART FIQ handler reads the stroked key and stores the related char in a text file. When sensitive data needs to be displayed to the user, the UART driver sends the data out via the UART port and Tera Term displays it on PC. We use the UART port connecting with a PC to emulate the I/O peripherals due to the convenience for program debugging and driver development.

VIII. Evaluation

In this section, we give some evaluations to TrustTokenF. We first present the TCB size of our implementation. Then we prove the security of the trusted user path we build. Finally we present a performance evaluation on our system.

TCB size. We evaluate the TCB size using the lines of source code. The results are shown in Table 1. The size of modules we implement is very small, except the KDF and seal/unseal function, which use the ported cryptography algorithms. The total size is 4.5K.

Security of the Trusted User I/O Path. We implement a proof-of-concept program in normal world acting as a ‘UART logger’ malware, who hooks the UART FIQ interrupt handler in normal world and records the UART inputs in a text file. Since multicore architecture is common seen in modern mobile devices like the develop board we use, malicious programs may still be running on other insecure cores while secure token services are running. So the malware also periodically queries

The experiment result is shown in Table 2.

Our software implemented token achieves high computation performance compared with other hardware tokens. The TPM chip we use only runs at 33M Hz, making it impractical for situations with demanding performance requirements. The other two dedicated tokens are even slower.

The extra operations during the authentication procedure also need to be evaluated, i.e., the world switch, the policy validation and the unseal function. We evaluate the world switching time by deploying an empty application in secure world. We denote the policy validation as three hash operations using SHA-256, and the unseal function as an AES decryption. The result shows that the total extra time is about 14 milliseconds, which is much less than the signing time of three hardware tokens.
attacker is physical proximity to the victim, due to lack of a trusted user confirmation method for transactions. Tamrakar et al. [24] propose a security framework for smart card emulation on smartphones, which leverage Obc for secure storage and isolated execution. Their work focuses on the compliant issues with the mainstream smart card standard. These solutions all rely on a device-unique key, which is not always available on current mobile devices. Moreover, the conventional hardware-backed secure storage like eFuse is expensive and inflexible for key updating.

Several works propose to use TEEs to implement general mobile 2FA schemes. Rijswijk-Deij and Poll [25] introduce a security model based on hardware tokens to investigate whether two TEE technologies, Intel’s IPT and ARM TrustZone can achieve the same level of security with hardware tokens. Ahmad et al. [20] integrate TrustZone with SIM card based security techniques to provide enhanced security for user authentication and content purchase. These works all argue the importance of a trusted user interaction path. However, they only describe their prototype design and don’t give the concrete implementation. Also, they both overlook the secure storage issue.

Areno et al. [18] present a PUF based method to protect the integrity of the TEE. Their idea for secure storage is similar to our work. However, they only concern about the secure boot process, and don’t present their implementation for the root of trust based on PUF.

IX. CONCLUSION

We propose TrustTokenF, a generic security framework for mobile 2FA authentication, which leverages ARM TrustZone to provide a runtime trusted execution environment and to build a dynamically configurable trusted user interaction path. Our framework also provides a cost-effective secure persistent storage mechanism based on SRAM PUF, which achieve comparable physical security and higher flexibility than non-volatile secure memory. We also introduce trusted computing secure primitives, seal/unseal and policy based authorization to strengthen our framework. TrustTokenF can achieve comparable level of security to dedicated hardware tokens and is more flexible, efficient and economical.

ACKNOWLEDGMENT

This research is sponsored by the National Natural Science Foundation of China (under grants No.91111801160) and the National 973 Program of China (under grants No.2013CB338003).

REFERENCES