TrustOTP: Transforming Smartphones into Secure One-Time Password Tokens

He Sun[1,2,3], Kun Sun[1], Yuewu Wang[2], and Jiwu Jing[2]

[1]Department of Computer Science, College of William and Mary
{hsun01,ksun}@wm.edu
[2]Data Assurance and Communication Security Research Center, Chinese Academy of Sciences
State Key Laboratory of Information Security, Institute of Information Engineering of CAS
[3]University of Chinese Academy of Sciences
{sunhe, wangyuewu, jingjiwu}@iie.ac.cn

ABSTRACT

Two-factor authentication has been widely used due to the vulnerabilities associated with traditional text-based password. One-time password (OTP) plays an indispensable role on authenticating mobile users to critical web services that demand a high level of security. As the smartphones are increasingly gaining popularity nowadays, software-based OTP generators have been developed and installed into smartphones as software apps, which bring great convenience to the users without introducing extra burden. However, software-based OTP solutions cannot guarantee the confidentiality of the generated passwords or even the seeds when the mobile OS is compromised. Moreover, they also suffer from denial-of-service attacks when the mobile OS crashes. Hardware-based OTP tokens can solve these security problems in the software-based OTP solutions; however, it is inconvenient for the users to carry physical tokens with them, particularly, when there are more than one token to be carried. In this paper, we present TrustOTP, a secure one-time password solution that can achieve both the flexibility of software tokens and the security of hardware tokens by using ARM TrustZone technology. TrustOTP can not only protect the confidentiality of the OTPs against a malicious mobile OS, but also guarantee reliable OTP generation and trusted OTP display when the mobile OS is compromised or even crashes. It is flexible to integrate multiple OTP algorithms and instances for different application scenarios on the same smartphone platform without modifying the mobile OS. We develop a prototype of TrustOTP on Freescale i.MX53 QSB. The experimental results show that TrustOTP has small impacts on the mobile OS and its power consumption is low.

CCS'15, October 12–16, 2015, Denver, Colorado, USA.

DOI: http://dx.doi.org/10.1145/2810103.2813692.

Categories and Subject Descriptors

D.4.6 [**Operating Systems**]: Security and Protection

General Terms

Security

Keywords

One-Time Password; ARM TrustZone; Secure GUI

1. INTRODUCTION

There is an ever increasing number of enterprise employees who need to remotely access the corporate networks. By the end of 2015, more than 1.3 billion workers worldwide will routinely work beyond the traditional office environment [35]. Around the same time, more mobile devices are being widely used to perform business transactions by mobile workers. Enterprises have traditionally used two-factor authentication to secure employee's remote access to corporate resources. Due to its ease of use, one-time password (OTP) is widely adopted by enterprises in their two-factor authentication solutions [24, 44, 52].

An OTP is an automatically generated numeric or alphanumeric string of characters that authenticates the user for a single transaction or session to an authentication server. OTP enhances the traditional user ID and password authentication by adding an extra level of dynamic password that changes each authentication. Two most popular types of OTP solutions are the time-based OTP (TOTP) and HMAC-based OTP (HOTP) that is event-based. OTP mechanism consists of a token - either hardware-based (e.g., pocket-size fobs) or software-based (e.g., a soft token) - that can generate an OTP using a built-in clock (or counter) and a factory-encoded secret key. The secret key is known as the "seed", which is different for each token and also stored in the related authentication server.

As the smartphone gains its popularity nowadays, software-based OTP generators have been developed and installed on smartphones as software apps [2]. Due to the popularity of smartphone usages, the software-based OTP solutions bring no extra burden to the users and are economical to use. However, they may suffer from a couple of security problems. First, when the mobile operating system is compromised, it cannot guarantee the confidentiality of the generated OTPs or even the seeds. For instance, the attacker may steal the OTP by taking screenshots that contain the OTP displayed on the screen [42]. If the OTP app has been instrumented [16], the instrumented code can stealthily send the OTP out to the attacker. Moreover, rootkits [57] in mobile operating system can steal the OTP seed through inspecting system memory and easily duplicate the soft token. Second, the mobile OS may face denial-of-service attacks, so there is no guarantee of the OTP availability when the OTP is required. A malicious OS can tamper with or simply delete the OTP code in the memory or the permanent storage, so that the OTP generator cannot successfully run. Also, when the OS crashes, the OTP is not accessible either.

Since hardware OTP tokens can solve all the security problems of the software OTP tokens, they have been widely used in the scenarios that demand a higher level of security [24, 44, 52]. A small piece of self-contained software runs in a tamper-resistant environment, which makes it difficult to compromise the seeds in the physical token for duplication. The battery in the hardware tokens can commonly last for 5 years. However, compared to software token, the hardware tokens also have some limitations. First, it is not easy to upgrade the software in the physical tokens. Second, a physical token usually costs tens of dollars [26], which is much more expensive than a software token that is usually free. Third, it is inconvenient for the users to carry physical tokens with them, particularly, when there are more than one fob to be carried all the time.

In this paper, we present TrustOTP, a secure one-time password solution that can achieve both the flexibility of software tokens and the security of hardware tokens by using ARM TrustZone technology [15]. TrustOTP can not only protect the confidentiality of the OTPs against a malicious mobile OS, but also guarantee reliable OTP generation and trusted OTP display to the users when the mobile OS is compromised or even crashes. Our solution can accommodate various OTP algorithms and support multiple OTP instances on the same smartphone platform. TrustOTP targets at combining the benefits of both hardware tokens and software tokens.

First, TrustOTP can achieve a secure OTP token that ensures confidentiality, integrity, and availability of the OTPs generated on smartphones. All the code and data of TrustOTP in both volatile and non-volatile memory storage are securely isolated from the mobile OS (called Rich OS) running in the normal domain. TrustZone ensures that the Rich OS cannot compromise the confidentiality and integrity of the generated OTPs or the seeds, which are only accessible in the secure domain. Moreover, TrustOTP can ensure the availability of the OTPs even if the Rich OS is compromised or totally crashes. The static code image of TrustOTP is stored on a secure permanent storage that can only be accessed in the secure domain, so the Rich OS cannot delete or modify the code image. Users can trigger a non-maskable interrupt (NMI) to guarantee that the system will be switched into the secure domain when the OTP is required.

Second, TrustOTP is flexible to integrate various OTP algorithms and able to support multiple OTP instances on the same smartphone. It can support both event-based and time-based OTP. For the time-based OTP, a secure real-time clock is used by the OTP and protected from the Rich OS. For the event-based OTP, a secure counter, which increments as the OTP is updated, can only be accessed by the secure domain. Given an OTP algorithm on the smartphone, users can easily add new OTP instances by copying the new OTP seeds into the system.

Third, we provide a trusted user input/output for users to input new seeds, select needed OTP instances, and display the OTPs. Since the Rich OS and the trusted OTP application share the same touchscreen, we must ensure that no sensitive information will be leaked into the Rich OS through the shared display device. When the input (e.g., OTP seed) can be mediated by the Rich OS, it can be manipulated by malware. We solve this problem by integrating a self-contained touchscreen driver in the secure domain for users to register new OTP instances. Also, we achieve a trusted output by isolating a secure framebuffer that cannot be accessed by the Rich OS and providing a secure display controller in the secure domain. When the selected OTP instance is displayed on one corner of the screen, the users can still see the screen contents of the Rich OS and continue their operations in the Rich OS.

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

- We propose a new design of secure OTP Tokens using smartphones. Our design can achieve the same level of security as the hardware token and the flexibility of the software token. It can prevent all types of attacks from the malicious Rich OS and is capable of showing the OTP even if the mobile OS crashes. It is flexible to support various OTP algorithms and multiple OTP instances on one smartphone.

- We provide a trusted graphical user interface that displays the OTP on the same screen shared with the Rich OS. Our user-friendly display allows users to read the OTP and input it into an app in the Rich OS simultaneously. We ensure that the Rich OS cannot directly read the OTPs and seeds from the framebuffer memory or obtain the current OTP by capturing the screenshots. Moreover, we provide a trusted touchscreen driver for users to input new OTP seeds and choose the OTP instances to display.

- We implement a TrustOTP prototype and the evaluation results show that TrustOTP can work efficiently with small power consumption. TrustOTP requires no modification of the Rich OS and has small impacts on user experience of using the Rich OS. In our prototype, after the user presses a physical button, the password can be shown on the screen in less than 80 ms.

The remainder of the paper is organized as follows. Section 2 introduces background knowledge. Section 3 describes the threat model and assumptions. We present the TrustOTP framework in Section 4. A prototype implementation is detailed in Section 5. Section 6 discusses the experimental results. We discuss the limitation of our system in Section 7. We describe the related work in Section 8 and conclude the paper in Section 9.

2. BACKGROUND

2.1 One-Time Password

One-time password (OTP) is the kind of password that is only valid for one transaction. Different from the traditional passwords, OTP is resistant to replay attacks. The OTPs can be generated in three ways: the time-synchronization OTP is calculated based on current time, the previous-password-based OTP is calculated using the previous OTP, and the challenge-based OTP is calculated with a challenge from the server. The OTP can be generated respectively by the client and an authentication server with shared secret key, or it can be generated on the server side and then sent to the client through SMS or Email. A number of OTP-related standards have been developed, including RFC 1760 (S/KEY) [10], RFC 2289 (OTP) [11], RFC 4226 (HOTP) [12] and RFC 6238 (TOTP) [13].

2.2 ARM TrustZone

ARM TrustZone technology [4] is a system-wide security approach to provide hardware-level isolation between two execution domains: *normal domain* and *secure domain*, which share the CPU in a time-sliced fashion. The secure domain has a higher access privilege than the normal domain, so it can access the resources of the normal domain such as memory, CPU registers, and peripherals, but not vice versa. TrustZone includes an *NS* bit in the CPU processor to control and indicate the state of the CPU. The memory address is partitioned into a number of memory regions, which are marked as either secure or non-secure, by the TrustZone Address Space Controller (TZASC). TrustZone supports secure and non-secure interrupts. The normal domain cannot access the interrupt source of the secure domain. Each interrupt is marked either secure or non-secure, and a Generic Interrupt Controller (GIC) only signals an IRQ interrupt for the non-secure interrupt and signals either IRQ or FIQ for the secure interrupt. The device privilege is configured in the TrustZone Protection Controller (TZPC) that dedicates one bit to each independent device.

A system supporting TrustZone is always shipped with additional supportive hardware security modules to better work with TrustZone, such as secure non-volatile keys [30], secure storage [18, 28], secure real-time clock [30, 28] and cryptographic accelerators [30, 18, 28]. These security modules work together with TrustZone to enhance system security and performance. For instance, to prevent rollback attacks in time-sensitive protocols such as DRM and PKI, secure real-time clock (SRTC) is introduced. The SRTC is usually powered by a coin-cell to keep running even when the platform is powered off.

3. THREAT MODEL AND ASSUMPTIONS

We assume the Rich OS running in the normal domain may be compromised by malicious code, and further the compromised OS may be manipulated to attack the one-time password generator. First, it may attempt to read (steal) the OTPs directly from the memory or the display device. Second, it may target at obtaining the seeds that are used to generate the OTPs by searching both volatile and non-volatile storage. Third, it may tamper with either the static code image or the control flow of TrustOTP to steal the generated OTPs. Lastly, the malicious OS can launch denial-of-service attacks to prevent the user from successfully obtaining the OTPs. The Rich OS may suspend TrustOTP from being either executed or displayed correctly. It can even make the system crash or unbootable, though this does not happen frequently.

Our OTP solution aims at achieving the same level of security as the traditional hardware OTP token. The attacks against the hardware tokens are out of the scope of this paper; however, we discuss those limitations in Section 7. We assume the attacker may have physical access to the mobile devices. We trust the TrustZone hardware security supports on the ARM processors and assume the code in the ROM and the secure domain can be trusted. We assume the cell battery that powers the real-time clock on the board can last for a long time.

4. TrustOTP DESIGN

4.1 System Overview

We design a secure OTP framework called TrustOTP that uses ARM TrustZone hardware security extension to transform smartphones into secure one-time password tokens. The framework of TrustOTP is shown in Figure 1. A mobile operating system called Rich OS runs in the normal domain to execute normal mobile apps. TrustOTP is installed in the secure domain and consists of three major components: *OTP generator*, *secure display controller*, and *secure touchscreen driver*. The OTP generator is responsible for continuously generating one-time passwords even if the Rich OS is malicious or crashes. Since TrustOTP and the Rich OS share the same set of physical display device, TrustOTP requires a secure display controller to guarantee that the one-time passwords can be and only be seen by the user, but not the Rich OS. A secure framebuffer is reserved to save the data to be displayed by the touchscreen. In order to support multiple OTP instances, TrustOTP must provide a trusted touchscreen driver for users to choose the OTPs they need.

4.2 Secure OTP Generator

The OTP Generator is responsible for supporting various OTP algorithms to compute OTPs. It supports two most popular categories of OTP: the time-based OTP (TOTP) and the event-based OTP (HOTP). The TOTP generator reads the current clock time from a secure clock when generating a time-based OTP, and the HOTP generator maintains one growing counter to calculate the OTP. Thus, a clock and a counter are indispensable to the OTP generator.

4.2.1 OTP Generation Code

TrustOTP is flexible to accommodate various OTP algorithms. Since TrustOTP and the authentication server share the same OTP generation algorithm and the same secret key, the two parties will generate the same OTP when the generation code and the keying materials are well protected. The static code and data of the OTP generator should be stored and protected on a secure non-volatile storage, which cannot be accessed or flushed by the Rich OS. The keying materials used in the OTP algorithms are also stored in the secure permanent storage. Otherwise, if the static code can be manipulated by the Rich OS, it may disclose the keying material and the OTPs to the Rich OS. Moreover, the Rich OS can compromise the availability of the OTP by flushing

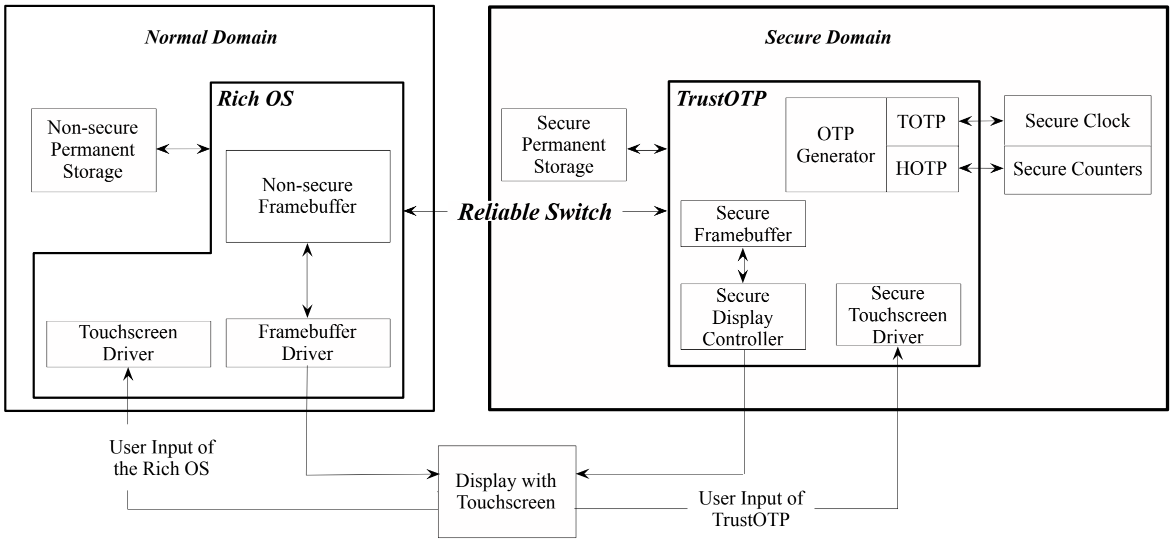


Figure 1: TrustOTP Framework.

the code from the storage. Thus it requires to store the image of the Rich OS in a separate non-volatile storage, which also serves as the filesystem of the Rich OS. When the system boots up, the code and data of the OTP generator will be loaded into a secure memory region that can only be accessed in the secure domain. Therefore, the Rich OS cannot access the sensitive information directly from the memory.

4.2.2 *Secure Clock and Counters*

Besides protecting the OTP code integrity and the seed secrecy, we also need to protect the timer and the counter that are used as input to TOTP and HOTP, respectively. Since the secure clock serves as the time source of TOTP, its value should not be modified by the untrusted Rich OS. Moreover, the clock should keep ticking even if the smartphone runs out of power. Typically, a secure clock is powered by an independent power source such as a dedicated cell battery. Each event-based HOTP has a corresponding counter, which increments as the HOTP is updated. Since only the secure domain has the privilege to access the counters, the normal domain cannot modify these counters. Moreover, the counter value is well maintained after the smartphone is powered down. The secure counters can be real physical modules that run on independent power source and increment by one on demand or just numbers stored on the secure non-volatile storage that are updated each time the corresponding HOTPs are updated.

4.3 Secure OTP Display

TrustOTP requires a trusted graphical user interface (GUI) to display the continuously changing OTPs to the user immediately, while the Rich OS cannot compromise the GUI itself and the data that the interface processes. We integrate a secure Display Controller in TrustOTP to securely copy the image from a secure framebuffer to the display device, where the framebuffer stores the image of the OTP to be displayed. To prevent potential OTP leakage, the secure framebuffer is different from the framebuffer used by the Rich OS and reserved in the secure domain. Since one smartphone usually has only one video card and one display screen, those peripheral devices are shared between the Rich OS and TrustOTP. We must ensure a reliable OTP display controller to program the video card and the display device, so that they can work correctly no matter what states they are before the system switching. Moreover, TrustOTP should save the states of the video card and the display device for the Rich OS before resetting them to display the OTPs. Afterwards, TrustOTP cleans up its states and restores the Rich OS's states before switching back to the normal domain.

4.4 Secure Touchscreen for OTP Registration

TrustOTP can support multiple OTP instances that use the same OTP algorithm but different seeds. To dynamically add a new OTP instance, the user first registers in the corresponding authentication system and obtains a shared secret key for TOTP or a counter plus a key for HOTP. Next, the user needs to input the shared keying information to the mobile devices. Since the Rich OS cannot be trusted, we must provide a trusted input interface in the secure domain to initiate the OTP. Thus, we include a self-contained secure touchscreen driver in the secure domain for the user to input into the secure domain. Moreover, TrustOTP can be dynamically extended to accommodate new OTP algorithms or upgraded to a new version. It also requires a trusted input interface that can authenticate the user before the installation.

4.5 Secure Booting and Reliable Switch

TrustOTP is loaded into the secure domain when the system boots up and remains in the secure memory unless the system restarts or powers down. The boot sequence is depicted in Figure 2. The code on the ROM runs first after the smartphone powers on. It loads the secure bootloader from the secure permanent storage to the memory of the secure domain. Then the secure bootloader gains control and initializes the secure domain. Next it loads TrustOTP from the secure permanent storage to the memory of the secure domain, and also loads the non-secure bootloader into the memory of the normal domain. Finally, the secure bootloader changes the CPU from the secure state to the non-secure state and jumps to the non-secure bootloader.

The non-secure bootloader initializes the normal domain and boots the Rich OS. After that, the Rich OS is running in the normal domain. Since TrustOTP is loaded before the Rich OS, it will stay in the memory no matter what the status of the Rich OS is.

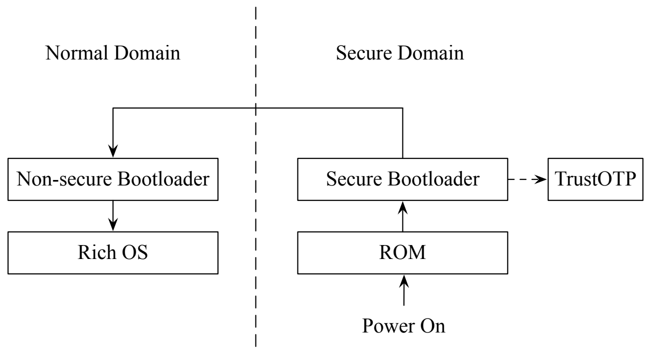


Figure 2: Booting Sequence of TrustOTP.

The OTP is usually demanded when the user performs an online transaction or logins an authentication system. Therefore, when this happens, the Rich OS should be suspended for a short time to allow the system switch into the secure domain to generate and display the OTPs. A reliable switch ensures that TrustOTP can be triggered on demand even if the Rich OS is malicious or crashes. In other words, the interrupt triggering the switch cannot be disabled or intercepted by the Rich OS. We cannot use any switching instruction or a software interrupt in the Rich OS to initiate the switching, since they can be easily disabled or intercepted by the Rich OS. Instead, we can use a non-maskable interrupt (NMI) that is triggered by a hardware interrupt to initiate the switching. Since the secure domain is non-reentrant, after the system enters the secure domain, the system will switch back to the Rich OS only when TrustOTP initiates the switching back to the normal domain.

4.6 Security Analysis

Our design can protect the confidentiality, integrity, and availability of TrustOTP even if the Rich OS is malicious or crashes. We manage to achieve the same security level of hardware tokens by using the smartphones, while keeping the flexible usage of software tokens.

Information Leakage. Since the OTPs are generated in the secure domain, the Rich OS in the normal domain has no rights to access any resources of the OTPs. The keying materials used in the OTP algorithms are stored on the secure non-volatile storage. When the system boots up, they are loaded into the secure memory region of the secure domain by the secure bootloader. Thus, the Rich OS cannot access the keys from either the permanent storage or the RAM memory. Moreover, before the system switches back to the Rich OS, the TrustOTP cleans up its footprint including CPU registers that may contain sensitive data for the OTPs. The Rich OS may target at tampering with the control flow of TrustOTP; however, since the code of TrustOTP runs in the secure domain, the Rich OS cannot modify TrustOTP's code. Moreover, since all the secure interrupts triggered by TrustOTP are handled in the secure domain, the Rich OS cannot intercept the execution of TrustOTP through interrupts.

Though TrustOTP shares the same display device with the Rich OS, TrustOTP has its own independent display controller to guarantee that the OTPs displayed on the screen cannot be accessed by the Rich OS. We allocate a dedicated framebuffer for TrustOTP in the secure memory region, which is not accessible to the normal domain. Moreover, a dedicated display controller maintains a secure context switching on the display device, so that the Rich OS's display states are saved and stored during the system switching and TrustOTP's display states are flushed before returning to the Rich OS.

Denial-of-Service Attacks. Since we attempt to use smartphones as physical hardware OTP tokens and the mobile OS may have been compromised, denial-of-service (DoS) attacks are the largest threat on the usability of our solution. First, when the user needs to trigger TrustOTP and get the OTP, the Rich OS may intercept and discard the requests. Our reliable switching mechanism can prevent this type of DoS attack by using a hardware-based non-maskable interrupt, which cannot be bypassed by the Rich OS. The interrupt handler of the NMI is in the secure domain, so the Rich OS cannot manipulate the handler either. Therefore, we can guarantee that the system will switch into the secure domain as soon as the NMI is triggered.

Second, a malicious Rich OS may delete the binary code of TrustOTP from the permanent storage. We solve this problem by saving the TrustOTP code in an isolated storage medium that can only be accessed by the secure domain. Note we cannot use the solution that saves the encrypted TrustOTP code on the same storage medium as the Rich OS and decrypts/verifies the TrustOTP code during the system booting, since the Rich OS has the privilege to delete the TrustOTP files so that TrustOTP cannot be loaded after the system reboots. Similarly, since the TrustOTP code in the memory is protected by TrustZone, the Rich OS cannot tamper with it. Moreover, the inputs of the OTP algorithms including the clock and counters cannot be manipulated by the Rich OS either, since we use a dedicated secure clock only accessible in the secure domain and store the counters in the secure storage, respectively.

Third, the Rich OS may prevent the OTP results from being displayed to the user. When the Rich OS is running, it may disable the display device, misconfigure the video card, or even powering down the display device. However, since TrustOTP includes a self-contained display controller, we can guarantee that the OTP will be shown on the display device even if the Rich OS is malicious.

5. IMPLEMENTATION

We implement a TrustOTP prototype using Freescale *i.MX53 QSB*, a TrustZone-enabled mobile System on Chip (SoC) [31]. i.MX53 QSB has an ARM Cortex-A8 1 GHz application processor with 1 GB DDR3 RAM memory. i.MX53 QSB is inserted a SanDisk 4GB MicroSD memory card and a Kingston DTSE9 32GB USB 2.0 Flash Drive. We use a 4.3 inch seiko touchscreen LCD. We deploy an Android 2.3.4 source code from Adeneo Embedded [14] in the normal domain. We use a Thinkpad T430 laptop with Ubuntu 12.04 LTS installed to debug and test the developing board.

5.1 Secure Non-volatile Storage

It is a key feature for TrustZone to protect a secure permanent storage from the non-secure domain; however, Trust-

Zone leaves it to the SoC vendors to decide which storage peripherals are subject to this mechanism. Thus, each vendor may have different designs on assigning or partitioning storage resources for the access of either world. In cases when the vendor does not provide a dedicated secure permanent storage, we can convert one available permanent storage to secure storage.

It is common for a smartphone to have more than one type of permanent storage. For instance, a smartphone usually uses a NAND flash as its primary permanent storage and has a MicroSD card as an extended permanent storage. Thus, we can dedicate one storage medium for the secure domain and another one for the normal domain. However, on our development board i.MX53 QSB, it has no NAND flash memory, but equipped with a MicroSD card port and a USB port. Therefore, we build a prototype by assigning the MicroSD card to the secure domain and the USB flash drive to the normal domain. On the secure MicroSD card, we store the static images of the secure bootloader, TrustOTP, the non-secure bootloader, and the kernel of the Rich OS. On the non-secure USB flash, we only store the filesystem of the Rich OS. By saving the static images of the non-secure bootloader and the Rich OS kernel on the MicroSD card, we can guarantee that even if the Rich OS is compromised when running in the normal domain, its static kernel image is still protected, so the system can be recovered after rebooting.

The privilege of peripherals are set in the Configure Slave Level (CSL) Registers of CSU [30]. Each peripheral has four access modes: *non-secure user*, *non-secure supervisor*, *secure user* and *secure supervisor*. The secure mode only allows the access from the secure domain, and the supervisor mode only allows the access from the privileged mode of CPU. Therefore, in our prototype, the MicroSD port is set to secure user mode while the USB port is set to non-secure user mode. We set the Rich OS to mount the USB flash drive as the filesystem in the *init.rc* file of Android.

5.2 Memory Isolation

i.MX53 QSB provides a watermark mechanism to isolate secure memory regions from non-secure memory ones. This mechanism is managed by Multi Master Multi Memory Interface (M4IF) [30]. There are two banks of RAM on i.MX53 QSB, and each bank is 512 MB. M4IF can watermark one continuous region of up to 256 MB on each bank. The start and end address of one watermark are stored in the *Watermark Start Address Register* and *Watermark End Address Register*, respectively. The watermark region cannot be accessed by the normal domain. The watermark controller inside M4IF is not accessible to the normal domain, either. In our prototype, we reserve the highest 1 MB RAM as the secure memory. In the 1 MB secure memory, 750 KB is reserved as the secure framebuffer.

The framebuffer contains 800*480 pixels, and each pixel is expressed by a 2-byte RGB565 value. To display the OTP, we save the pictures of number from 0 to 9 that occupy 156.25 KB memory. Each number picture has 100*80 pixels. The number pictures are preloaded into memory when the system boots. Since there's no need to copy the picture data from the MicroSD card to memory, it shortens the displaying latency when TrustOTP is running. Though the required memory space is small compared to the 1 GB memory space of i.MX53 QSB, we can further reduce it by using small-sized number pictures.

5.3 TrustOTP Booting

Figure 3 shows the detailed booting process of TrustOTP. When the system powers on, the code in the ROM runs first, and it loads the secure bootloader from the MicroSD card into the secure memory. Then, the secure bootloader is responsible for loading TrustOTP from the MicroSD card into the secure memory. Next, the secure bootloader also loads the images of the non-secure bootloader and the Rich OS kernel from the MicroSD card into the non-secure memory. At last, the secure bootloader switches the system into the normal domain to run the non-secure bootloader, which sets the execution environment in the normal domain and then runs the already loaded Rich OS kernel to mount the USB flash drive as filesystem. High Assurance Boot (HAB) of i.MX53 QSB [30] is used to ensure a secure booting through checking the integrity of the secure bootloader and the TrustOTP image before loading them. Therefore, our system can detect any malicious attempts on tampering with the static image on the MicroSD card.

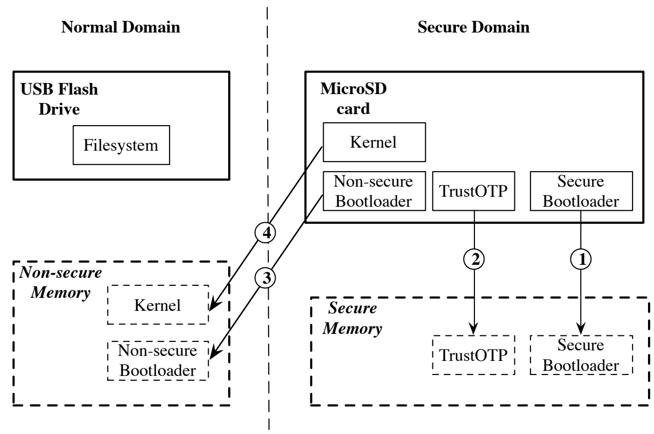


Figure 3: TrustOTP Booting Process.

5.4 Reliable Switch

Our reliable switch is based on the non-maskable interrupt (NMI) mechanism; however, since i.MX53 QSB does not explicitly provide any NMI, we propose a method to construct an NMI on i.MX53 QSB. First, we set the interrupt type of the targeted NMI source as secure in the Interrupt Security Register (TZIC_INTSEC), which prevents the Rich OS from configuring the NMI in the TZIC [30]. Second, we enable FIQ exception by setting the F bit of Current Program Status Register (CPSR) to 0. To ensure the F bit of CPSR cannot be tampered with by the Rich OS, we then set the FW bit of Secure Configuration Register (SCR) to 0. After these two bits are set, the Rich OS cannot disable or block the FIQ requests to the ARM processor. Third, we enforce the ARM processor to switch to the monitor mode on an FIQ exception by setting the FIQ bit in SCR to 1. This enforcement ensures that the FIQ request to the secure domain does not need to go through the normal domain, and thus cannot be intercepted or blocked by the Rich OS. Finally, we configure the NMI source as a secure peripheral in the CSU that cannot be accessed by the Rich OS.

In our prototype, we choose the user-defined button 1 on i.MX53 QSB to trigger the reliable switch. Since the button is connected to the fifteenth pin of GPIO-2, we use GPIO-2 as our NMI source. After the above steps, the system will switch into the secure domain as soon as the button is pressed.

5.5 OTP Generator

In our prototype, we implement two types of OTP generation algorithms: the event-based HOTP based on RFC4226 [12] and the time-based TOTP based on RFC6238 [13]. We adopt the source code from OATH Toolkit [46] that provides components for building one-time password authentication systems. It supports OTP generation, validation, and control through a command line. We only port the OATH Toolkit parts that implement OTP generation to the secure domain.

To reduce the system's trusted computing base (TCB), there is no operating system in the secure domain, so the OTP code must be self-contained and run directly on the bare-metal environment. Since the source code from OATH Toolkit relies on the C standard library to call the output functions such as `printf` and the memory operation functions such as `memcpy`, we port these functions in the secure domain to eliminate the dependency on the C library.

Listing 1: OTP Generation Functions

```
int oath_hotp_generate (const char *secret,
                        size_t secret_length,
                        uint64_t moving_factor,
                        unsigned digits,
                        char *output_otp)

int oath_totp_generate (const char *secret,
                        size_t secret_length,
                        time_t now,
                        unsigned time_step_size,
                        unsigned digits,
                        char *output_otp)
```

The two major functions are `oath_hotp_generation()` and `oath_totp_generation()`, whose declarations are shown in Listing 1. The explanation of the function parameters is listed in Table 1. `Secret` is the shared key and `secret_length` is the key's length. The algorithms are able to support 6-digit, 7-digit and 8-digit OTP, and the `digits` defines the length of the OTP. The generated OTP output is stored in `output_otp` in the form of characters. In the event-based HOTP generation, the `moving_factor` is the counter used for calculating the one-time password. In the TOTP, `now` is the current Unix time value and a new TOTP is generated every `time_step_size` seconds.

Table 1: Parameters of HOTP and TOTP.

Parameter	Explanation
`secret`	the secret key
`secret_length`	the length of secret Key
`moving_factor`	the secure counter in HOTP
`now`	the secure clock in TOTP
`time_step_size`	time period between two TOTPs
`digits`	the length of the generated OTP
`output_otp`	the generated OTP

5.5.1 Secure Key Management

The secret key (seed) must be protected both in the permanent storage and in the RAM memory. We store the key for one OTP on the secure MicroSD card, so that the Rich OS cannot read, write, or delete it. Furthermore, we can add another level of protection by storing the keys in ciphertext, which is encrypted by hardware SCC-AES support on i.MX53 QSB, where the encryption key is protected in the e-FUSE based secure storage. Since the key will only be loaded into the secure memory by the secure bootloader, the Rich OS cannot access it. Moreover, to prevent information leakage, all the CPU registers are flushed to remove the residual keying materials before switching back from the secure domain to the normal domain.

5.5.2 Clock and Counter Protection

For the time-based TOTP, a synchronized clock time is required to generate the OTP, so the clock on the smartphone should continue to run even if the system is powered off. Moreover, the clock cannot be manipulated by the Rich OS. On i.MX53 QSB, a secure real-time clock (SRTC) is included. The SRTC clock is served by a 32.768KHz crystal, and it consists of two domains: low power domain and high power domain. The low power domain of SRTC (SRTC LP) is powered by a coin cell battery. Therefore, the SRTC LP is always powered up as long as the battery works. Moreover, the SRTC LP can only be accessed by the secure domain. Therefore, we use the SRTC LP as the secure clock for our time-based OTP solutions.

For the event-based HOTP, the counter changes each time an OTP is generated. Since the value of the counter should be maintained even after the system is reset, we use the Low Power Secure Monotonic Counter Register (*SRTC_LPSMCR*) on our board to store the counter. The counter cannot be accessed by the normal domain and is able to retain its value after the system reboots.

5.5.3 Multiple OTPs

TrustOTP is flexible to accommodate different OTP algorithms. Due to the OTP standardization work, most OTP instances use the same algorithms but different seeds and other parameters such as counters. Thus, we can support a large number of OTP instances in our system. When a user wants to add a new OTP instance, the user usually first requests a new key from the corresponding authentication system or through other secure communication channels. Then, the user can use a trusted user input interface using touchscreen to upload the new key into the secure storage and update TrustOTP with the new instance. Since the secret key is usually 20 bytes, it may be tedious and error-prone to input 40 hexadecimals manually. Instead, the user can use a MicroSD card reader to upload the new key. Moreover, an alias is given when a new OTP instance is created. Thus, the user can choose the needed instance according to its alias shown on the screen.

For the TOTP instances, they can share the same SRTC clock time; while the HOTP instances may require to protect different counters besides the different secret keys. In our prototype, since i.MX53 QSB only provides one secure counter register, to support multiple HOTP instances, we choose to save those counters on the secure MicroSD card. To minimize the counter synchronization error, we update the counter on the MicroSD card each time an OTP is generated.

5.6 Secure Touchscreen Driver

Since the user needs to input the key in the secure domain to register an OTP and choose which OTP to update both on the touchscreen, a secure touchscreen drivers must be included to ensure the input will not be intercepted or blocked by the Rich OS. In our prototype, a 4-wire resistive touchscreen is connected to the Power Management Integrated Circuit (PMIC) [27]. The PMIC collects the voltage of the touchscreen and converts the analog value to digital one that represents the X-Y coordinate of the touch on the screen. The value of each coordinate is stored in an Analog-to-Digital Converter (ADC) register in PMIC and is zero when there is no touch event.

When the touchscreen is touched, a PMIC interrupt arises. The interrupt handler calls the touchscreen driver to read the ADC registers to get the location of the touch. Next the driver parses the user's intention based on the input and the display. Since the touchscreen is used by both the Rich OS and TrustOTP, it has two interrupt handlers: one for the normal domain and the other for the secure domain. After TrustOTP is triggered by NMI, the interrupt of the touchscreen is set to secure. Then if there is a touch on the screen, an interrupt will arise in the secure domain. After all the activities end in the secure domain, the interrupt of the touchscreen is configured back to non-secure.

5.7 Secure Display Controller

Since the LCD is shared between the secure domain and the normal domain, TrustOTP shares the same display device with the Rich OS. Therefore, we must prevent the Rich OS from stealing the keying materials from TrustOTP through inspecting the shared display device. Moreover, since the Rich OS may be compromised or simply crashes, we cannot rely on the Rich OS's framebuffer driver to display the OTP to the users.

In our prototype, we include a self-contained secure display controller in the secure domain to reset the IPU in the secure domain and program the IPU to display the secure framebuffer. We adopt a stand-alone IPU driver from an on-board diagnostics suite of i.MX53 QSB [32]. Our display system consists of two major components: the Image Processing Unit (IPU) and the LCD. The IPU is responsible for sending the data flow from the display framebuffer to the external display device. It can also conduct certain image processing and manipulation on the data flow. The size of the framebuffer, the pixel format, and the location of the framebuffer are all set in the Channel Parameter Memory (CPMEM). The LCD is the display device and it requires initialization before being used.

When the Rich OS is running, the IPU is set as a non-secure device and can transfer data from the non-secure framebuffer to the LCD. When the system switches to the secure domain, the secure display controller saves the state of the IPU and uses the self-contained touchscreen driver to reset the IPU. In i.MX53 QSB, the normal domain can choose to use either one framebuffer or three framebuffers in a round-robin fashion to improve the display performance. By default, the Rich OS uses all three framebuffers. There is a control interface between the GPU and the IPU to synchronize the usage of the framebuffers. After switching into the secure domain, the secure display controller only uses the secure framebuffer to transfer pixel data to the LCD. Since the IPU is set as a non-secure peripheral when working in the normal domain, the controller needs to set the IPU as secure to transfer the data from the secure framebuffer to LCD. Before returning to the Rich OS, the controller erases the footprint in the device to prevent the information leakage, and then restores the device states for the Rich OS. An alternative method to achieve a secure display is to reuse the Rich OS's IPU driver after verifying its integrity. However, it cannot guarantee to provide a reliable display for OTP when the Rich OS crashes. Therefore, we choose to provide a self-contained display controller to show the OTPs in our prototype.

We reserve the highest 1 MB RAM as the secure memory. In the 1 MB secure memory, 750 KB is reserved as the secure framebuffer. Since the secure framebuffer contains the information of the displayed OTP, we protect it in the secure domain. The framebuffer contains 800*480 pixels, and each pixel is expressed by a 2-byte RGB565 value. To display the OTP, we save pictures of numbers from 0 to 9 that occupy 156.25 KB memory, which may be further reduced by using smaller pictures. These pictures are preloaded into the memory when the system boots.

5.8 User Friendly Display

When TrustOTP is running, the Rich OS will be suspended until TrustOTP exits. Therefore, if the OTPs are being displayed for a long time, the user cannot perform any operations in the Rich OS. It is fine for some usage scenarios. For instance, if the user wants to input an OTP on a laptop, he can keep showing the OTP on the smartphone's screen. However, if the user needs to input the OTPs into apps running on the smartphones, she has to remember the OTP before TrustOTP exits and then input it in the Rich OS. To make it convenient for the user to use the OTP locally on the smartphone, we develop a user-friendly display method. The basic idea is to display the OTP only for a very short time and repeat this display operation with a time interval until the user stops the display by pressing the user-defined button 1 again. The OTP needs only to be calculated once at the first entry of the secure domain. In next entries, TrustOTP just needs to display the generated OTP if no new OTPs are generated.

i.MX53 QSB provides two identical Watchdog Timer (*Wdog*) modules with different privilege: one for normal use and the other for TrustZone use. The Wdog 2 on our board is used for periodically displaying OTP. After being activated, a Wdog starts the time-out counter with a configurable period from 0.5 second to 128 seconds in a resolution of 0.5 second. The Wdog will generate an interrupt after the time period expires unless the counter is reset before the time out. The interrupt of Wdog 2 is configured as an NMI and Wdog 2 starts to count since the system boots up; however, the interrupt is disabled by TrustOTP. Hence, the Wdog 2 will not raise interrupt when it times out in the normal domain. After the user-defined button 1 is pressed, TrustOTP draws the generated OTPs into the framebuffer and enables the interrupt of Wdog 2. Then TrustOTP returns to the normal domain to run the Rich OS. When the Wdog 2 times out, it raises an NMI that suspends the Rich OS and switches the system into the secure domain unconditionally. Then the generated OTPs are displayed on the screen for a while. After the display ends, TrustOTP resets the counter of Wdog 2 to make it continue running and switches back to the Rich OS. In this manner, TrustOTP and the Rich OS runs in turn until the user-defined button 1 is pressed again. Then, TrustOTP disables the interrupt of Wdog 2 to stop switching into the secure domain after TrustOTP transfers the control back to the Rich OS.

The time interval between two rounds of OTP display is configurable. It should be short enough so the user will not waste time waiting for the next display. The OTP display time should be short enough so the user will not waste time waiting for the Rich OS to run. However it should be long enough for the user to recognize the OTP clearly. OTP display time should be longer than 1/24 second due to *phi phenomenon* [9]. In our prototype, the time interval between two displays is set to 1.5 seconds. The length of OTP display time is determined by a `while` loop. In the loop an integer `i` increases by one from `0` until `i` reaches a threshold, which is set to `0x4ffffff` in our prototype.

6. PERFORMANCE EVALUATION

We measure the performance overhead and the power consumption for generating and displaying the OTPs in TrustOTP. We use the performance monitor in the Cortex-A8 processor to count the CPU cycles and then convert the cycles to time by multiplying 1 *ns* / *cycle*. We conduct each experiment 50 times and report the average value. We use Monsoon Power Monitor [45] to evaluate the power consumption of TrustOTP.

6.1 Time Breakdown

In our prototype, when the user presses the physical button, an NMI is triggered and the Rich OS is suspended. OTPs are generated and displayed periodically (e.g. 1 second) until the user presses the button again. Before the OTPs are displayed, there is preparation time when the Rich OS is suspended but TrustOTP is not ready for use. We break down the preparation time between the NMI triggered and the OTP displayed on the monitor into 7 parts, as shown in Table 2.

It takes less than 2 *us* to switch from the normal domain to the secure domain. We measure the switching times using both the user-defined button 1 and the Wdog 2. When the button is pressed, it takes 1.7 *us* for the system to enter the secure domain; it takes 1.8 *us* to enter the secure domain after Wdog 2 is triggered. TrustOTP saves the context information of the Rich OS in 0.6 *us*. It saves all the 33 general-purpose registers of the processor into the secure memory. It takes 48 *us* to generate a TOTP and 44 *us* to generate a HOTP. Then to create a smooth user friendly display of the OTPs, TrustOTP copies the non-secure framebuffer to the secure framebuffer. It takes 49.85 *ms* to finish the background matching. This step is the largest overhead in the preparation of the display. Next, TrustOTP draws the numbers of OTP in the secure framebuffer. It takes 8.029 *ms* to copy twelve 100*80-pixel pictures in the secure framebuffer to show two OTPs (one HOTP and one TOTP) at the same time. Then TrustOTP takes 2.22 *ms* to check all the internal registers and memory of IPU. At last, it takes 0.28 *ms* to change the framebuffer pointer of IPU from the non-secure framebuffer to the secure framebuffer.

The overall time for preparing the OTP display is 60.4716 *ms*. After displaying the OTP, the time to recover the Rich OS is 7.52 *ms*, which consists of flushing the FIFO buffer of IPU and changing the framebuffer pointer back to the non-secure framebuffer. The flushing is done by replacing the generated OTPs with value `000000` (6-digit OTP is chosen in our prototype) in the secure framebuffer. In our prototype, it takes 7.47 *ms* to drawing two all-zero 6-digit value in the secure framebuffer. Since the picture of the zero number will be cached due to frequent usage, it is shorter than OTP drawing in the preparation stage. We can see these two time periods are small and the OTP can be shown quickly after being requested.

Table 2: Time breakdown of OTP Display.

Step	Operation	Time (ms)
1	Domain Switching	0.002
2	Context Saving	0.0006
3	TOTP/HOTP Generation	0.048/0.044
4	Background Matching	49.85
5	OTP Drawing	8.029
6	IPU Check	2.22
7	Framebuffer Replacement	0.28

6.2 Impacts on Rich OS

TrustOTP resides in the secure domain and does not run most of time. We evaluate the impacts on the Rich OS when TrustOTP is triggered by running Android benchmark tools including Antutu [3] and Vellamo [47]. Then, we compare the results to those when TrustOTP is not triggered in Figure 4. Antutu measures the performance of CPU, RAM, GPU and Database, while Vellamo integrates five benchmarks and runs every benchmark to get an overall score in one test. The vertical axis is the score of each item, and higher scores indicates better performance. We can see that the performance of the Rich OS decreases during the short time when TrustOTP is running. It is reasonable because the same operation takes more time to complete in the Rich OS when TrustOTP is running. Particularly, the performance of CPU and RAM decrease more than that of the peripherals. This is because when TrustOTP is running, CPU and RAM are fully controlled by the secure domain, but the peripheral can continue the tasks of the Rich OS .

6.3 Power Consumption

The Monsoon Power Monitor and Power Tool Software [45] enable a robust power measurement solution for mobile devices rated at 4.5 volts (maximum 3 amps) or lower. The monitor is able to measure the instant and average voltage, current, and power. We use the power monitor to first measure the power of the board when only the Rich OS is running. We run the experiment 5 times. In each experiment the board is restarted and runs for ten minutes. When only the Rich OS is running, the average power of the board is $2,128\ mW$. The sample of the instant power data is shown in Figure 5a. The vertical axis is the value of the power in the unit of watt. When only the Rich OS is running, the LCD is on but there is no input from the user. When TrustOTP runs together with the Rich OS, the average power of the board increases a little bit to $2,230\ mW$. In our experiments, we see that the power consumption goes up when the system switches into the secure domain. The sample of the power data when TrustOTP is running is in Figure 5b.

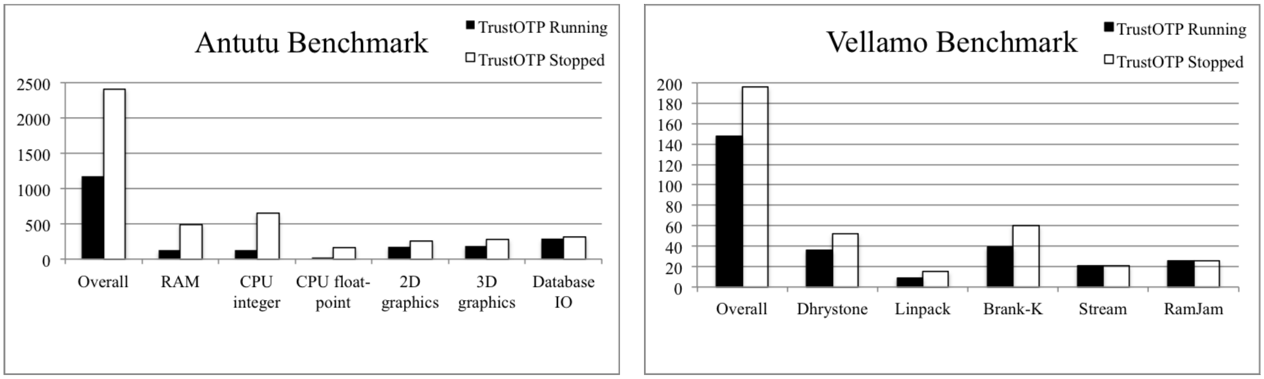
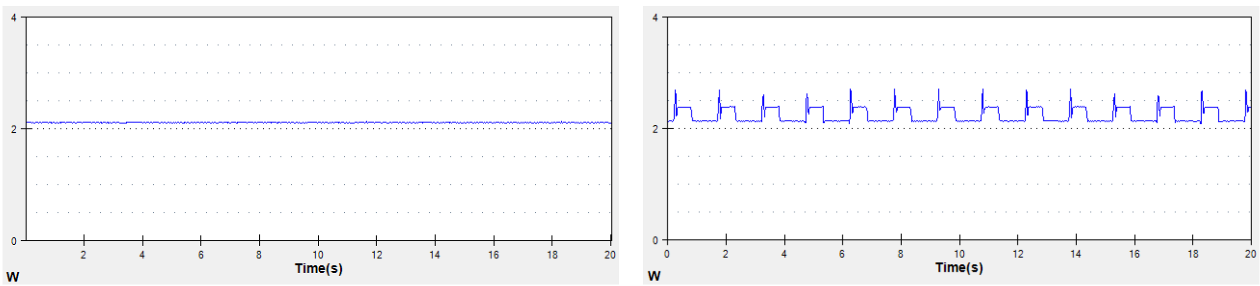


Figure 4: Performance Impacts on Rich OS.



(a) TrustOTP Stopped

(b) TrustOTP Running

Figure 5: Samples of Power Usage Data.

In Figure 5b, the power curve remains the same in each display cycle since it performs the similar OTP operations. In each cycle TrustOTP goes through the above seven steps in Section 6.1. There is a peak at the start of each cycle and then the power keeps at a high level due to a busy `while` loop until TrustOTP exits. After removing the loop, the sample of the power data is depicted in Figure 6, which only shows a peak in the beginning of every cycle.

Since the power is mainly subject to the ratio of the time when OTPs are displayed in each cycle, the time period when OTPs are displayed determines the power consumption when the display cycle is fixed. When the time period decreases, the power consumption of TrustOTP also decreases. The time period of the `while` loop can be adjusted by changing the threshold of the integer counter. A suitable time could be chosen through experimental tests to both save energy and remain the user perception. We test the length of the `while` loop from `0xffffff` to `0x8ffffff` in the resolution of `0xffffff`. Finally we pick `0x4ffffff`, which guarantees that the user can see the OTPs clearly and the Rich OS is suspended for only a minimal time period. According to the OTP display time, we choose the time between two displays as 1.5 seconds. It leaves the user approximate 1 second to input 2 to 3 of the OTP numbers into the Rich OS.

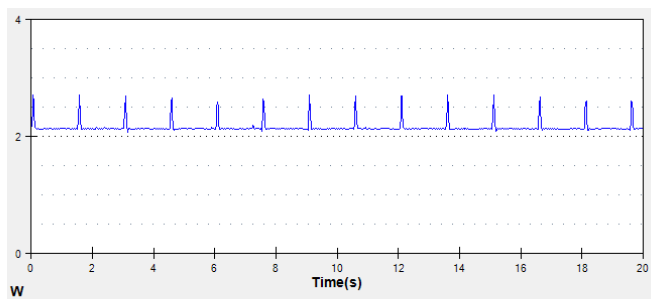


Figure 6: TrustOTP without Display

7. LIMITATIONS AND DISCUSSION

We aim at using commercial off-the-shelf (COTS) smartphones to achieve a secure one-time password solution with the same security level as hardware OTP tokens. However, currently the major smartphone vendors like Samsung lock the TrustZone in their commercial products to protect their OEM software in the TrustZone, so it is difficult for third parties to develop and deploy Trustzone-based mechanisms on real products. Instead, researchers choose to build their system using development boards [29, 7] or emulators [56] that have TrustZone support enabled. Our prototype is built on a Freescale i.MX53 development board [29].

When comparing our smartphone-based token to traditional hardware tokens, one major difference is the battery. For most hardware tokens, the life of battery is typically 5-8 years [52]. TrustOTP depends on the battery of the smartphone, which needs to be recharged every one or two days. When the smartphone is out of battery, TrustOTP cannot work until being charged by a computer or a power outlet. However, for most working scenarios such as when the user needs to retrieve the OTP from the smartphone and then input it into a computer, it is easy to find a power supply to the smartphone even if it is out of power. Hopefully, wireless charging (i.e., inductive charging) solutions can further mitigate this problem. On the other hand, it is complicated to replace the batteries in hardware tokens due to the tamper-resistant design.

Hardware OTP tokens are designed to be tamper-resistant to protect the seed (secret key). In TrustOTP, we save all the encrypted OTP secret keys on the MicroSD card. Even if attackers may have access to the MicroSD card, they still cannot decrypt the OTP secret keys without knowing the key in the tamper-resistant e-Fuse register that is responsible for decrypting the secret keys when loading them into the secure memory. As portable password containers, both hardware tokens and the smartphones have the same practical vulnerability – they may be lost or stolen. For a hardware token, a user will typically wait more than one day before reporting the missing, which gives the attacker plenty of time to breach the unprotected system. However, when a smartphone is stolen, the users may discover it at an early stage and report it quickly to reduce the risk of misuse.

Similar to the usage of physical tokens, our TrustOTP may suffer from man-in-the-middle attacks when used alone. If the attacker can intercept the current password and block the authorized user from being authenticated by the server until the next token code becomes valid, the attacker is able to log into the server. We can prevent this attack by adopting the risk-based authentication mechanism from RSA SecurID [38]. Moreover, both hardware tokens and TrustOTP cannot work correctly if the authentication server's clock is out of sync with the clocks built into the tokens or smartphones. Normal clock drift on hardware tokens and smartphones can be accounted for automatically by the server through adjusting a stored "Drift" value over time. In TrustOTP, even if the server clock or the smartphone clock had drifted or been changed, the dedicated SRTC clock on the smartphone can be resynchronized manually.

Each hardware token typically only provides one instance of one-time password, so a user may bring multiple physical tokens to authenticate to different servers. In TrustOTP, if two OTP instances share the same OTP algorithm such as the open OATH HOTP standard, we can support them by simply adding the keying materials for the new instance. Also, it is easy to integrate various OTP algorithms in the secure domain after converting them into self-contained mode. One remaining challenge is to isolate all OTP instances from each other, so one malicious OTP algorithm cannot compromise other OTP instances. It could be solved by running each OTP in a lightweight container [5]. It is a general problem on how to install and upgrade secure apps into TrustZone, since the TrustZone is controlled by phone manufactures. With careful code review and PKI certification chain technology, OTP service providers may collaborate with phone manufactures to customize a secure API for installing and upgrading OTP software in the secure domain. We leave it as our future work when commercializing our system.

8. RELATED WORK

There are several types of OTP tokens in use, and they can be divided into software-based token and hardware-based token. Software-based OTP tokens have been widely used on desktops and laptops, incorporating the convenient OS supports and functionalities [50, 44, 52] to provide OTPs for user authentication. Software tokens have been widely ported to mobile devices too. For instance, Google develops a software OTP token called *Google Authenticator* [34] as an app supporting both iOS and Android. Moreover, smartphones can receive OTPs from the authentication server through an out-of-band channel such as SMS or email. In general, software-based OTP tokens are vulnerable to malicious apps and compromised OS running on the same machine. Hardware virtualization supports on x86 and ARM processors are promising to help protect the OTP generator from other malicious software [20, 39, 54, 1].

Hardware-based OTP tokens use a dedicated physical fob to calculate and display the OTP on an integrated screen [24, 58]. Contrast to RSA SecurID that provides proprietary hardware OTP solution [24], OATH Token is an open-source software token that implements the RFC 4226 HOTP/OATH algorithm standard and is not tied to any proprietary server software. A number of commercial hardware tokens support the OATH standards [8], including Vasco Digipass GO 6 OATH [6]. Yubikey [58] has a USB interface with customized software to provide OTP on a laptop. The major drawback of these hardware tokens is that the user needs to carry an extra hardware that is prone to being lost or stolen. It is not convenient to upgrade the software in the physical tokens either. Moreover, a hardware token can cost around 100 dollars. Intel Identity Protection Technology with One-Time Password [36] integrates a built-in OTP hardware token into the CPU core. The integrated hardware token is tamper-proof and isolated from the operating system. However, such OTP solution is based on Intel Core processors and is not available on ARM processors. TrustOTP achieves both the flexibility of the software tokens and the security of the hardware tokens.

Alexandra et al. [21] analyzed potential attacks on mobile two-factor authentication (2FA) schemes and provided a number of valuable high-level research directions against those attacks. Our work falls into the countermeasure category of leveraging secure hardware on mobile platforms to provide a trusted user input/output to a trusted OTP app when the Rich OS and the trusted OTP application share the same touchscreen and display. Our solution leverages ARM TrustZone to isolate TrustOTP from an untrusted Rich OS.

ARM TrustZone can isolate a secure OS from a Rich OS into two computing domains. Thus, untrusted applications in a compromised Rich OS cannot access secure applications in the secure OS [55, 15, 53, 33]. Several TrustZone-based systems (e.g., Mobicore/Trustonics [33], Trusted Logic [53], ObCs [40, 23], and KNOX [48]) have been developed to enhance the security of mobile devices. For instance, MobiCore/Trustonics [33] is a secure Operating System for TrustZone-enabled ARM controllers including ARM1176 or CortexA8/A9. It provides development tools called Trustlets for third-party application developer.

A number of research efforts have been done on TrustZone. Jang et al. [37] build a secure channel between the normal domain and the secure domain. Azab et al. [17] leverage the secure domain to protect the integrity of the Rich OS kernel in real time. Sun et al. [51] perform reliable memory dump of the Rich OS in the secure domain. Santos et al. [49] use TrustZone to build a trusted language runtime that support .NET Framework in the secure domain. Marforio et al. [43] propose a new location-based second-factor authentication solution by using smartphone as location verification token for payments in point of sale transaction. Pawel et al. [22] combine TrustZone with a customized PANTA display processor [25] to provide trusted input and output. It can ensure a strong I/O isolation between the two execution environments; however, since it depends on integrating a dedicated co-processor, it may have compatible issues and increase the cost. Li et al. [41] provide a verifiable mobile ad framework to detect and prevent advertisement frauds. Their verifiable display technqiue can achieve a trusted display as TrustOTP does.

Winter et al. [56] develop a flexible software emulation framework for TrustZone development. Since the major smartphone vendors lock the TrustZone in their commercial products to protect their OEM software, researchers choose to develop their prototypes using development boards [29, 7, 19] or emulators [56] that have TrustZone support enabled. Our prototype is built on a Freescale i.MX53 development board [29].

9. CONCLUSIONS

We design a secure OTP token solution called TrustOTP using smartphones, with the goal to achieve both the security of the hardware tokens and the flexibility of the software tokens. Our design can prevent all types of attacks from the malicious mobile OS and continue to display the OTP even if the mobile OS crashes. It is flexible to support various OTP algorithms and multiple OTP instances on one smartphone. It requires no changes of the mobile OS and has small impacts on the mobile OS's performance. A prototype shows that TrustOTP works efficiently with tiny extra power consumption.

10. ACKNOWLEDGMENT

The authors would like to thank the shepherd, Brent Byunghoon Kang from Korea Advanced Institute of Science and Technology (KAIST), and the anonymous reviewers for their valuable comments and suggestions. He Sun and Kun Sun are supported by U.S. Office of Naval Research under Grant N00014-15-1-2396 and N00014-15-1-2012. He Sun, Yuewu Wang and Jiwu Jing are supported by National 973 Program of China under Award No. 2013CB338001 and Strategy Pilot Project of Chinese Academy of Sciences under Award No. XDA06010702. Yuewu Wang is the corresponding author.

11. REFERENCES

[1] AMD Virtualization. http://www.amd.com/en-us/solutions/servers/virtualization.

[2] Android OATH Token. https://code.google.com/p/androidtoken/.

[3] Antutu Benchmark. http://www.antutu.com/en/Ranking.shtml.

[4] ARM. http://www.arm.com/.

[5] Booting the Android LXC container. https://wiki.ubuntu.com/Touch/ContainerArchitecture.

[6] DIGIPASS GO 6. https://www.vasco.com/products/client_products/single_button_digipass/digipass_go6.aspx.

[7] Juno ARM Development Platform. http://www.arm.com/products/tools/development-boards/versatile-express/juno-arm-development-platform.php.

[8] OATH Compatible Hardware Tokens. http://www.rcdevs.com/tokens/?type=hardware.

[9] Phi phenomenon. http://en.wikipedia.org/wiki/Phi_phenomenon.

[10] RFC1760. https://tools.ietf.org/html/rfc1760.

[11] RFC2289. https://tools.ietf.org/html/rfc2289.

[12] RFC4226. https://tools.ietf.org/html/rfc4226.

[13] RFC6238. https://tools.ietf.org/html/rfc6238.

[14] Adeneo Embedded. Reference BSPs for Freescale i.MX53 Quick Start Board. http://www.adeneo-embedded.com/en/Products/Board-Support-Packages/Freescale-i.MX53-QSB.

[15] T. Alves and D. Felton. Trustzone: Integrated hardware and software security. *ARM white paper*, 3(4), 2004.

[16] S. Arzt, S. Rasthofer, and E. Bodden. Instrumenting android and java applications as easy as abc. In *Runtime Verification - 4th International Conference, RV 2013, Rennes, France, September 24-27, 2013. Proceedings*, pages 364–381.

[17] A. M. Azab, P. Ning, J. Shah, Q. Chen, R. Bhutkar, G. Ganesh, J. Ma, and W. Shen. Hypervision across worlds: Real-time kernel protection from the ARM trustzone secure world. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014*, pages 90–102.

[18] J. Azema and G. Fayad. M-shield mobile security technology: making wireless secure. *Texas Instruments Whitepaper*, 2008.

[19] O. Board. Origen exynos4 quad evaluation board. http://www.origenboard.org/wiki/index.php/Introduction.

[20] C. Dall and J. Nieh. KVM/ARM: the design and implementation of the linux ARM hypervisor. In *Architectural Support for Programming Languages and Operating Systems, ASPLOS '14, Salt Lake City, UT, USA, March 1-5, 2014*, pages 333–348.

[21] A. Dmitrienko, C. Liebchen, C. Rossow, and A. Sadeghi. Security analysis of mobile two-factor authentication schemes. *Intel Technology Journal*, 18(4), 2014.

[22] P. Duc. Secure Mobile Payments - Protecting display data in TrustZone-enabled SoCs with the Evatronix PANTA Family of Display Processors. http://www.design-reuse.com/articles/30675.

[23] J. Ekberg, K. Kostiainen, and N. Asokan. Trusted execution environments on mobile devices. In *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*, pages 1497–1498.

[24] EMC^2. RSA SecureID Hardware Tokens. http://www.emc.com/security/rsa-securid/rsa-securid-hardware-tokens.htm.

[25] Evatronix. Evatronix Launches Display Processor based on Latest ARM Security Technology. http://www.electronicsweekly.com/noticeboard/general/evatronix-launches-display-processor-based-on-latest-arm-security-technology-2012-05/.

[26] Fortinet. FortiToken. http://www.fortinet.com/products/fortitoken/index.html.

[27] Freescale. Hardware Reference Manual for i.MX53 Quick Start-R. http://cache.freescale.com/files/32bit/doc/ref_manual/IMX53RQSBRM-R.pdf?fr=g.

[28] Freescale. i.MX 6Solo/6DualLite Applications Processor Reference Manual. http://cache.freescale.com/files/32bit/doc/ref_manual/IMX6SDLRM.pdf?fpsp=1&WT_TYPE=Reference%20Manuals&WT_VENDOR=

FREESCALE&WT_FILE_FORMAT=pdf&WT_ASSET=Documentation.

[29] Freescale. i.MX53 Processors. http://www.freescale.com/webapp/sps/site/taxonomy.jsp?code=IMX53_FAMILY.

[30] Freescale. i.MX53 Reference Manual with fusemap addendum. http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=i.MX537&fpsp=1&tab=Documentation_Tab.

[31] Freescale. Imx53qsb: i.mx53 quick start board. http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=IMX53QSB&tid=vanIMXQUICKSTART.

[32] Freescale. On Board Diagnose Suit (OBDS). http://www.freescale.com/webapp/sps/download/license.jsp?colCode=IMX53QSBOBDS&location=null&fasp=1.

[33] Giesecke & Devrient. MobiCore. http://www.gi-de.com/en/trends_and_insights/mobicore/trusted-mobile-services.jsp.

[34] Google. Google Authenticator. http://en.wikipedia.org/wiki/Google_Authenticator.

[35] IDC. Worldwide Mobile Worker Population 2011-2015 Forecast. http://cdn.idc.asia/files/5a8911ab-4c6d-47b3-8a04-01147c3ce06d.pdf, Dec 2011.

[36] Intel. Intel identity protection technology with one-time password. http://ipt.intel.com/Home/How-it-works/network-security-identity-management/ipt-with-one-time-password.

[37] J. Jang, S. Kong, M. Kim, D. Kim, and B. B. Kang. Ssecret: Secure channel between rich execution environment and trusted execution environment. In *21st Annual Network and Distributed System Security Symposium, NDSS 2015, San Diego, California, USA, February 8-11, 2015*.

[38] Jeff Carpenter, EMC. Did You Know: Trends in RSA SecurID Two-Factor Authentication. http://www.emc.com/collateral/rsa/eventpresentations/04-10-12-Two-Factor_Auth.pdf.

[39] S. Kalkowski. Virtualization Dungeon on ARM. In *Free and Open Source Software Developers' European Meeting, FOSDEM 2014, Brussels, Belgium, February 1-2, 2014*.

[40] K. Kostiainen, J. Ekberg, N. Asokan, and A. Rantala. On-board credentials with open provisioning. In *Proceedings of the 2009 ACM Symposium on Information, Computer and Communications Security, ASIACCS 2009, Sydney, Australia, March 10-12, 2009*, pages 104–115.

[41] 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, MobiSys 2015, Florence, Italy, May 19-22, 2015*, pages 75–88.

[42] C. Lin, H. Li, X. Zhou, and X. Wang. Screenmilker: How to milk your android screen for secrets. In *21st Annual Network and Distributed System Security Symposium, NDSS 2014, San Diego, California, USA, February 23-26, 2014*.

[43] C. Marforio, N. Karapanos, C. Soriente, K. Kostiainen, and S. Capkun. Smartphones as practical and secure location verification tokens for payments. In *21st Annual Network and Distributed System Security Symposium, NDSS 2014, San Diego, California, USA, February 23-26, 2014*.

[44] McAfee. Mcafee one time password. http://www.mcafee.com/us/products/one-time-password.aspx.

[45] Monsoon Solutions. Monsoon Power Monitor. https://www.msoon.com/LabEquipment/PowerMonitor/.

[46] Open AuTHentication. OATH Toolkit. http://www.nongnu.org/oath-toolkit/.

[47] Qualcomm Innovation Center. Vellamo Mobile Benchmark. https://play.google.com/store/apps/details?id=com.quicinc.vellamo&hl=en.

[48] Samsung Electronics. White Paper: An Overview of Samsung KNOX. http://www.samsung.com/global/business/business-images/resource/white-paper/2013/06/Samsung_KNOX_whitepaper_June-0.pdf.

[49] N. Santos, H. Raj, S. Saroiu, and A. Wolman. Using ARM trustzone to build a trusted language runtime for mobile applications. In *Architectural Support for Programming Languages and Operating Systems, ASPLOS '14, Salt Lake City, UT, USA, March 1-5, 2014*, pages 67–80.

[50] SolidPass. Desktop soft token. http://www.solidpass.com/authentication-methods/one-time-password-generator-otp-token.html.

[51] H. Sun, K. Sun, Y. Wang, J. Jing, and S. Jajodia. Trustdump: Reliable memory acquisition on smartphones. In *Computer Security - ESORICS 2014 - 19th European Symposium on Research in Computer Security, Wroclaw, Poland, September 7-11, 2014. Proceedings, Part I*, pages 202–218.

[52] Symantec. Whitepaper: Two-factor Authentication: A TCO Viewpoiont. https://www4.symantec.com/mktginfo/whitepaper/user_authentication/whitepaper-twofactor-authentication.pdf.

[53] Trusted Logic. Trusted foundations by trusted logic mobility. http://www.arm.com/community/partners/display_product/rw/ProductId/5393/.

[54] R. Uhlig, G. Neiger, D. Rodgers, A. L. Santoni, F. C. Martins, A. V. Anderson, S. M. Bennett, A. Kagi, F. H. Leung, and L. Smith. Intel Virtualization Technology. *Computer*, 38(5):48–56, 2005.

[55] J. Winter. Experimenting with ARM trustzone - or: How I met friendly piece of trusted hardware. In *11th IEEE International Conference on Trust, Security and Privacy in Computing and Communications, TrustCom 2012, Liverpool, United Kingdom, June 25-27, 2012*, pages 1161–1166.

[56] J. Winter, P. Wiegele, M. Pirker, and R. Tögl. A flexible software development and emulation framework for arm trustzone. In *INTRUST*, pages 1–15. 2011.

[57] D. You and B. Noh. Android platform based linux kernel rootkit. In *6th International Conference on Malicious and Unwanted Software, MALWARE 2011, Fajardo, Puerto Rico, USA, October 18-19, 2011*, pages 79–87.

[58] Yubico. Yubikey. https://www.yubico.com/.