Light-SPD: A Platform to Prototype Secure Mobile Applications

Sileshi Demesie Yalew[†‡], Gerald Q. Maguire Jr.[†], Miguel Correia[‡]
[†]School of Information and Communication Technology, KTH Royal Institute of Technology, Sweden
[‡]INESC-ID, Instituto Superior Técnico, Universidade de Lisboa, Portugal
sdyalew@kth.se maguire@kth.se miguel.p.correia@ist.utl.pt

ABSTRACT

Securely storing sensitive personal data is critical for protecting privacy. Currently, many persons use smartphones to store their private data. However, smartphones suffer from many security issues. To overcome this situation, the PCAS project is designing a secure personal storage device called the Secure Portable Device (SPD), to be attached to a smartphone for securely storing sensitive personal data. However, this device is unavailable, closed, and expensive to deploy for prototyping applications. We propose a platform that emulates the SPD and the smartphone using a board with an ARM processor with the TrustZone security extension. This platform is open, inexpensive, and secure. A payment application is used as an example to show the platform's capabilities. As a proof-of-concept, we implemented this platform and provide a performance evaluation using a i.MX53 board.

CCS Concepts

•Security and privacy → Tamper-proof and tamper-resistant designs; Mobile platform security;

Keywords

Mobile Computing; Privacy; Hardware Security; Trusted Computing; ARM TrustZone

1. INTRODUCTION

In order to protect privacy, many believe that individuals should control their own personal data and take responsibility for how this data is accessed. For this reason, many people carry their personal digital data with them. Increasingly, people rely on smartphone applications to store sensitive private data (e.g., passwords, contact information, and credit card numbers). Unfortunately, smartphones suffer from many security issues that can put this data in jeopardy [41, 13, 22]. Moreover, the increasing popularity of smartphones coupled with the fact that these devices store a large amount of sensitive personal data has made them an attractive target for malware writers, other attackers, and even law enforcement. Researchers have shown that Android-based mobile devices are vulnerable to a number of different attacks: malicious applications and libraries that misuse their privileges [17, 42]; root exploits that extract sensitive data [41]; exploit of unprotected interfaces [8, 39]; confused deputy attacks [9]; and collusion attacks [25].

To improve this situation, the *Personal Centralized Authentication System* (PCAS) project[1] is developing a trustworthy environment for secure storage and management of personal data. PCAS proposed a secure hardware component called the *Secured Portable Device* (SPD) that aims to be attached to a smartphone as if it was a sleeve to physically protect the device [34, 33].[2] This SPD can securely store privacy-critical data, such as personal medical records and financial data. It has also biometric sensors to authenticate its owner. The SPD is connected to the smartphone via an USB connection.

The ARM TrustZone is a hardware security extension incorporated into recent ARM processors [5]. This extension compartmentalizes the device in two *worlds* (or zones) and provides isolated execution of applications and secure storage. Specifically, the *secure world* runs trusted applications on top of a small trusted operating system (OS), whereas the *normal world* runs normal applications on top of a mobile OS, such as Android. The secure world's memory, peripherals, and interrupts are isolated from the mobile OS in order to ensure that the trusted application and the secure storage are protected from software attacks.

This paper presents the design of a platform called *Light-SPD*, that allows to prototype (experiment with) applications for the PCAS environment (i.e., for smartphones with SPDs), providing some degree of security to allow real deployment if that is desired. Light-SPD leverages a prototyping board with an ARM processor with TrustZone. The SPD is emulated in the secure world, whereas the smartphone's functionalities run in the normal world.

The original SPD is proprietary, only a few prototypes exist, and they are unavailable to the research community. In contrast, Light-SPD is open and easy to deploy, as it is implemented with readily available boards, such as the NXP Semiconductors (formerly Freescale Semiconductor) i.MX53 Quick Start Board (QSB). Moreover, Light-SPD is secure, so

PAMCO'16, July 05 2016, Paderborn, Germany
© 2016 ACM. ISBN 978-1-4503-4346-6/16/07...$15.00
DOI: http://dx.doi.org/10.1145/2940343.2940349

[1]https://www.pcas-project.eu

[2]The market of sleeves for smartphones and tablets is huge today. The SPD is expected to be cheaper than some of these sleeves that can cost hundreds of euros.

private data is protected from software and hardware attacks to the extent provided by the underlying secure hardware. The Light-SPD is less secure than the original SPD that aims to resist strong attacks such as using side-channels or X-rays to steal encryption keys. However, the purpose of Light-SPD is to provide a prototyping platform for developing applications rather than a physically secure product.

We demonstrate the capabilities of our design with a payment application in which persons use their smartphone to make payments at a point of sale (POS). This application leverages the SPD to make payments securely, despite the operating systems of the buyer's smartphone or of the POS being malicious. The SPD authenticates the owner and communicates securely with the backend application.

The contribution of this paper is the design, implementation, and experimental evaluation of a platform for secure mobile applications that emulates PCAS's SPD working together with a smartphone. The platform is open, inexpensive, and secure. An example payment application is presented and evaluated experimentally.

2. BACKGROUND

This section provides background information about the technologies underlying the design and implementation of Light-SPD.

2.1 ARM TrustZone

ARM TrustZone is an extension supported by recent ARM processors, including the ARM Cortex A8, A9, and A15 [5]. The TrustZone technology provides two trust domains. The first is called the *secure world* and is supposed to contain a small secure OS and security-critical services, whereas the untrusted side or *normal world* executes a full-fledged traditional OS (such as Android) and all user applications. The secure world should not run arbitrary code (such as an application downloaded from the Internet), hence a scheme based on signatures and a public-key infrastructure might be used to enforce this (e.g., a scheme similar to Java's [31]).

The context switch between these two worlds is controlled by a high privileged mode called TrustZone monitor mode and by a special *secure monitor call* (SMC) instruction. Each world has access to its own *memory management unit* (MMU) that maintains separate page table translations. Cache memories are TrustZone aware, thus the cache lines are tagged as secure or non-secure. As a result access to secure cached content from the normal world is denied.

Certain hardware peripherals and memory can be reserved to be exclusively accessible by the secure world. As a result, it is possible to secure peripherals such as memory, keyboard, and screen to ensure they can be protected from software attacks. The secure world provides code and data integrity and confidentiality because untrusted code running in the normal world cannot access protected resources, devices, or memory pages within the secure world.

2.2 The PCAS SPD

The PCAS SPD is a secure hardware storage device that works connected to a smartphone using USB [34, 33]. Currently, a few prototypes have been implemented, but it is not a commercial product yet.

The SPD is essentially a system-on-chip with its own battery. It has a large amount of memory so it can securely store a considerable amount of personal data, e.g., personal medical records. It has also biometric sensors, e.g., a front camera to implement face recognition, to authenticate its owner, in order to authorize access to the stored data on the SPD and enforce secure communication with service providers in the cloud when accessing and uploading data. The SPD has no network interfaces so it cannot connect to the Internet directly; instead it uses the smartphone's communication services (e.g., 3G or 4G) as a gateway. The SPD has no screen, only a few LEDs.

The SPD is designed to allow the implementation of several secure mobile applications. An example application involves personal medical records, which may be stored in the SPD for user convenience (e.g., to be able to provide them to different hospitals). Another example that we explore in the paper is a payment application. In this case, a SPD acts as credit/debit card in transactions to purchase products at POS or vending machines. In such a transaction, the user passes the SPD connected to her smartphone in front of an NFC reader, rather than using a credit card. Next the SPD asks the user to authenticate herself and to approve the amount to pay. The SPD contacts the backend of the application in the cloud that deducts the amount from the user's account. Finally, after the backend confirms the payment the vending machine releases the product.

3. LIGHT-SPD DESIGN

In this section, we describe the design of the Light-SPD platform. The main idea is to emulate both the SPD and smartphone system shown in Figure 1(a) using the virtual system shown in Figure 1(b) by taking advantage of the ARM TrustZone that provides a hardware-assisted isolated environment.

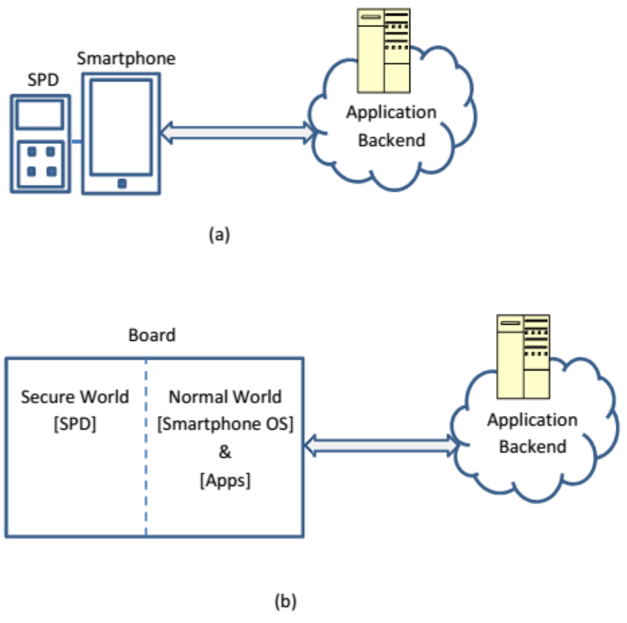

Figure 1: System architecture: (a) the SPD plus smartphone system; (b) the emulation system (Light-SPD).

Figure 1(a) depicts a system in which the SPD is physically connected to a smartphone using a USB cable and all data exchange between the two is carried over this communication link. The smartphone communicates with the cloud for backend processing. In Figure 1(b), a board which supports ARM TrustZone extension is used to emulate a smartphone. The smartphone's OS and applications are implemented in the normal world of the board, whereas the SPD functionalities are emulated in the secure world. Note that such a system might be implemented in smartphones with an ARM processor with TrustZone, but these devices are currently locked in such a way that it is not possible to use the secure world (without special arrangements with the device manufacturer). Samsung's Knox technology might also be used, but it is extremely hard to get access to it [3]. Therefore, we opted for an open board such as the i.MX53 QSB.

Figure 2 illustrates a more detailed architecture of the Light-SPD platform. Light-SPD provides three execution environments: (1) the *normal world*, an untrusted domain for running the mobile's OS and most applications' graphical user interface (GUI); (2) the *secure world*, a trusted domain for running the trusted OS, security sensitive application components (*trustlets*), and for storing security critical data as does the original SPD; and (3) an application backend where the core application services run. Sometimes we will abuse the expression and use *Light-SPD* to mean the software that emulates the SPD – the software inside the secure world – instead of meaning the whole platform.

3.1 Threat Model and Assumptions

We assume that in the normal world the mobile OS and the applications it executes are untrusted and potentially malicious. In contrast, we assume that the software running in the secure world is not compromised.

We leverage the ARM TrustZone hardware protection to protect data and code residing in the secure world from the potentially compromised mobile OS. Moreover, we assume that only trusted code is executed in the secure world. In order to reduce the size of the trusted computing base (TCB) [29], we reduce as much as possible the software installed in the secure world and, specifically, we do not include a network stack. We also reduce the size of the API the secure world exposes and carefully validate inputs to that API.

These two features make software attacks against the secure world hard, so for the purposes of this paper we assume they cannot be successful. We also assume that hardware attacks cannot be successful due to the protection provided by the TrustZone.

3.2 Light-SPD Components

As shown in Figure 2, the Light-SPD platform provides the following main components.

TrustZone Driver.

The SPD is a passive device that is used by the smartphone. Therefore, in the Light-SPD platform the communication follows the same pattern and it is the normal world that *calls* the secure world. This communication mechanism is similar to a remote procedure call or, more closely, to a call from a computer to a Java Card [28].

The TrustZone driver (`TZ_Driver`) is a kernel driver in the mobile OS, which supports cross-world calls from the normal world to the secure world. In this simple design it allocates a shared memory zone that is used for the normal world to pass parameters to the secure world, and for the secure world to pass results of calls to the normal world.

SPD API.

Light-SPD provides an application programming interface (API) in the normal world (`SPD_API`). This API allows untrusted application components running in the normal world to call trusted functions in the secure world. The API has a single function that requests an operation by a trusted application (trustlet) in the secure world:

```
operation_call(int AppID,          // in
               char *op_code,      // in
               char *buffer,       // in & out
               int buffer_size,    // in
               int *success_code)  // out
```

This function calls an operation with operation code `op_code` in the trustlet identified by `AppID`. The arguments to be passed to the trustlet are encoded into the `buffer`. If this function call is successful, it returns `success_code` with the value of zero. On an error, it returns -1. The result from this call will be placed back into the `buffer`.

An example of the `operation_call` function being called is presented in the code snippet shown in Figure 3.

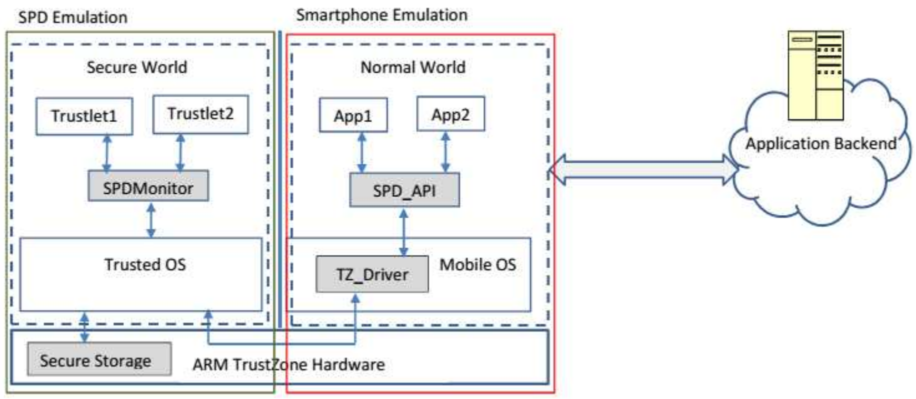

Figure 2: Light-SPD detailed architecture. Grey boxes are components that are specific of the Light-SPD platform.

```
AppID = ...
op_code = ...
size_of_inputs=compute_size(input_descriptors);
size_of_outputs=compute_size(output_descriptors);
buffer_size=max(size_of_inputs, size_of_outputs);
buffer=allocate_buffer(buffer_size);
success_code=0:

encode_into_buffer(buffer, input_descriptors,
                        argument1, argument2, ...);

operation_call(AppID, op_code, buffer, buffer_size,
                                     success_code);

if (success_code < 0) {
    process_error();
}

decode_from_buffer(buffer, output_descriptors,
                        output1, output2, ...);
```

Figure 3: An example API call.

SPDMonitor.

This component implements the core functionality of the SPD. It is responsible for receiving a request from an application in the normal world and determining the corresponding trustlet in the secure world. It also validates all the incoming parameters from the normal world in order to prevent attacks such as buffer overflows, format string attacks, or command injection. In addition, it may perform user authentication using the biometric sensors before it starts the trusted application partition in the secure world. This prevents unauthorized access to the user's private data. The form of authentication performed depends on a policy stored in the Light-SPD, similarly to what happens in the PCAS SPD.

3.3 Light-SPD Mechanisms

In this section, we discuss the security mechanisms provided by the Light-SPD platform.

Secure Connectivity.

The secure world provides the ability to communicate with a remote server using an end-to-end secure channel, hence securing this communication even against applications running in the normal world (that emulates the smartphone). In order to limit the size of the TCB, no network driver is included in the secure world, only in the normal world. Although the network driver is in the normal world, its role is only to forward packets (it works as a gateway) and communication is protected end-to-end, so it does need to be trusted. If compromised, it may only block the communication, not do other attacks.

The secure channel protects communication using 4 shared secret keys established using the Internet Key Exchange protocol [20]. Protection is implemented using the usual mechanisms: confidentiality is guaranteed by encrypting the data using the sender's encryption key and AES; integrity and authenticity are guaranteed using message authentication codes (MACs) obtained with the sender's MAC key; replay is prevented using sequence numbers. Messages always cross the normal world protected this way (they are only encrypted, decrypted, etc. inside the secure world).

Secure Storage.

The Light-SPD platform provides a secure storage component accessible only to the secure world. This secure storage is isolated in the secure world and cannot be accessed by the mobile OS or applications running in the normal world. Therefore, this secure memory can be used to store sensitive data (such as credit card details and personal health records) and this data can be protected from attacks despite the mobile OS being compromised.

Secure World User Interface.

Unlike the SPD, the Light-SPD has a small screen and a keyboard. The objective of this modification of the original idea is to provide more versatility when prototyping applications. Use of this screen and keyboard is optional.

While a screen and keyboard might be shared by the two worlds, if input/output (I/O) devices are accessible from both the worlds, then the data from/to the secure world could be accessed or modified by untrusted applications or the mobile OS (which itself is vulnerable to malware attacks) in the normal world. Gilad, Herzberg, and Trachtenberg [16] proposed using a hardware indicator on the device to shows the status, i.e., if the device is running in the normal or secure world (for example, a LED that only lights up when the secure world is running). However, OSs are designed to run in a CPU without (long) interruptions and it seems infeasible to dedicate the CPU to the secure world during the entire time the user needs to interact with it, e.g., seconds. Moreover, if an interrupt occurs a compromised OS could modify inputs/outputs.

Therefore, for Light-SPD we opted to use separate peripherals, i.e., its own small screen and keyboard. These input/output devices are controlled exclusively by the secure world, thus enabling Light-SPD applications to have a trusted GUI with which to securely interact with the user.

Biometric Authentication.

Light-SPD utilizes the biometric sensors of the device to implement user authentication mechanisms in order to subsequently authorize operations by using `SPDMonitor` in the secure world. This requires secure sensor data reading and processing by the secure world. However, malicious code in the normal world could create fake sensor data if these sensors are accessible to a compromised OS.

Research on trusted GPS [24], addresses this problem by resetting the sensor's configuration to a known good state whenever the secure world starts reading the sensor's data. However, this solution would lead to very high delay when using sensors. Therefore, similar to the separate I/O devices used for the trusted GUI, we use biometric sensors controlled exclusively by the secure world for user authentication.

4. MOBILE APPLICATIONS

In order to leverage Light-SPD to protect the user's privacy, a *mobile application* has to be partitioned into three components:

- an untrusted component implementing most of the application's GUI running in the normal world;
- a small application component that handles security sensitive computations running in the secure world (*trustlet*); and
- a (larger) component implementing most of the application's functionality running as service in a remote back-end server.

This partitioning minimizes the size of the TCB in the secure world, while enhancing scalability and reducing power consumption of the mobile device, as computationally intensive services are offloaded to the cloud.

As depicted in Figure 2, the untrusted application (*App*) in the normal world interacts with the trusted application (*trustlet*) by invoking operations. Since the *App* resides in a separate domain, it uses the `SPD_API` to communicate with the *trustlet* in the secure world.

Trustlets implement the SPD logic. They are responsible, e.g., for performing computation on security-sensitive data stored in the secure storage. They may also securely transfer data from/to remote services running in a remote server to perform computationally intensive operations. *Trustlets* may be generic (e.g., a *trustlet* for data storage) or application-specific. We assume a service provider validates and signs trustlets before they are inserted in the Light-SPD, which verifies the signature using the public-key of the provider.

Every Light-SPD *App* is assigned a unique identity number (`AppID`), that identifies each Light-SPD application installed on a device. Thus, the Light-SPD maintains a manifest file which contains a list of Light-SPD applications represented as 3-tuples <$\texttt{AppID}_i$, $\texttt{T}_i$, $\texttt{S}_i$>, where $\texttt{AppID}_i$ is the unique ID of the App_i in the normal world. This $\texttt{AppID}_i$ is associated with a *trustlet* $\texttt{T}_i$ in the secure world and a *service* $\texttt{S}_i$ (`URL`) for the corresponding service in the cloud. The manifest file is stored in the secure storage component accessible only by the secure world.

When App_i sends a request with $\texttt{AppID}_i$ to the `SPDMonitor`, the `SPDMonitor` uses the $\texttt{AppID}_i$ to determine the corresponding *trustlet* $\texttt{T}_i$ from the manifest file for secure computation in the secure world. Next, the `SPDMonitor` may communicate with the *service* $\texttt{S}_i$ in the cloud for backend processing.

4.1 Request Processing Example

Figure 4 shows an example of how an operation request could be processed in the Light-SPD platform. An untrusted application partition (`App`) in the normal world initiates a request containing an `AppID` and an operation code to the `SPDMonitor` in the secure world (step 1). The `SPDMonitor` checks for an entry for this `AppID` in the manifest file and asks the user for authentication via the trusted GUI (step 2).

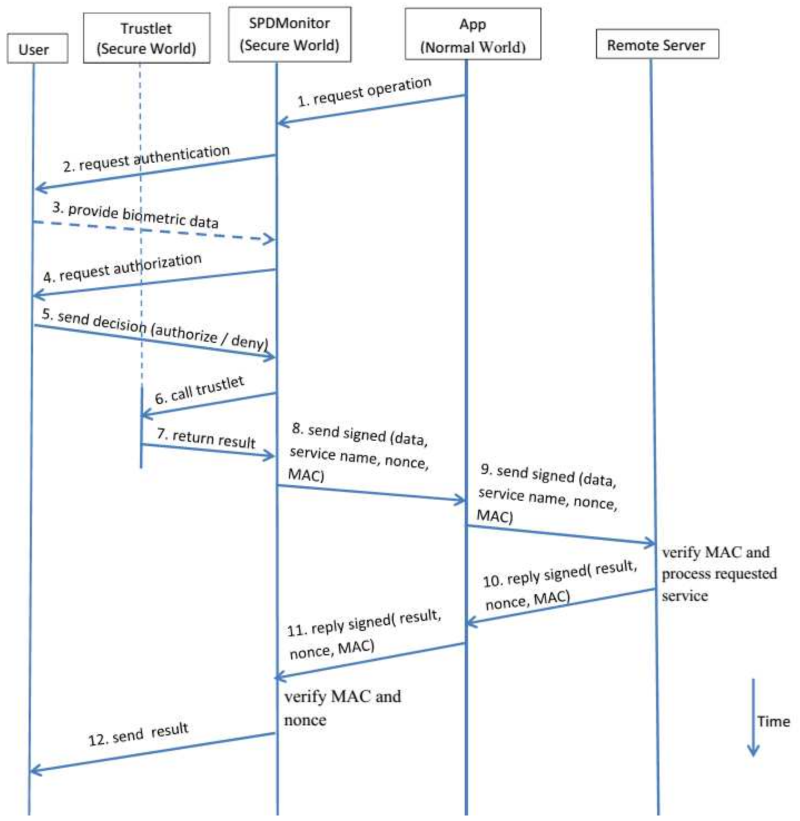
User
Trustlet (Secure World)
SPDMonitor (Secure World)
App (Normal World)
Remote Server
1. request operation
2. request authentication
3. provide biometric data
4. request authorization
5. send decision (authorize / deny)
6. call trustlet
7. return result
8. send signed (data, service name, nonce, MAC)
9. send signed (data, service name, nonce, MAC)
verify MAC and process requested service
10. reply signed(result, nonce, MAC)
11. reply signed(result, nonce, MAC)
verify MAC and nonce
12. send result
Time

Figure 4: Time diagram of an example processing of a request in Light-SPD. The user interacts via the trusted GUI.

The user provides biometric data via the secure biometric sensor (step 3). The `SPDMonitor` sends again a confirmation request to the user with information about the requested operation via the trusted GUI (step 4). The user replies with a decision (authorize or deny) and optionally enters user inputs through the trusted GUI (step 5).

If the user authorizes, the `SPDMonitor` then calls the corresponding trusted application partition (`trustlet`) (step 6). The `trustlet` returns the result back to the `SPDMonitor` (step 7).

The `SPDMonitor` encrypts user data, service name, and nonce (for replay protection) together with message authentication code (MAC) using the appropriate shared key, and sends this message to the remote server via the `App` in the normal world for backend processing (steps 8-9).

After receiving the data, the server decrypts it, checks the MAC, and performs the requested service. The server then encrypts the result, and nonce together with MAC using the shared key, and sends the ciphertext back to the `SPDMonitor` through the `App` in the normal world (steps 10-11). Finally, the `SPDMonitor` decrypts and checks the nonce and MAC, and displays the response data to the user (step 12). Note that this final display can be done either via the trusted GUI or the normal GUI depending upon the App's preference.

This scheme exemplifies how to provide data confidentiality, integrity, and authenticity in the Light-SPD platform. It is not part of the actual SPD or Light-SPD.

5. PAYMENT APPLICATION

Currently, many mobile payment applications store and process private data (such as credit card details and passwords/pin codes) in an untrusted environment, in a smartphone. Therefore, this sensitive data is vulnerable to various attacks from malicious applications in the smartphone.

Figure 5(a) shows a secure payment scenario at a point of sale (POS) terminal that leverages the PCAS SPD to protect user personal data and provide strong authentication with biometric data. The SPD is connected to a smartphone via USB and can be used to securely store the cardholder's data and other security-sensitive data (an the obvious use of SPDs for payments).

In this section we will consider a different, richer, payment scenario, represented in Figure 5(b). The secure payment application works between the customer and the cashier to showcase Light-SPD functionality by having each of the persons involved (customer and cashier) use their own devices with a (emulated) SPD. The two devices can communicate, e.g., with NFC, Bluetooth Low Energy, or Wi-Fi. The advantage of this scenario in relation to the previous one is that it allows secure payments even if the cashier and the customer devices are temporarily disconnected from the Internet.

Before any payment occurs, the customer and the cashier must have installed the payment application(s). To initiate a payment, the cashier starts the payment application (`cashier_App`) in her device, in the normal world. The `cashier_App` sends its `AppID` and operation code to the Light-SPD (i.e., to the `SPDMonitor`) in the secure world of the cashier's device by using the Light-SPD's API (i.e., the `SPD_API`). The `SPDMonitor` asks the cashier for authorization to request this payment via the device's trusted GUI and authentication via the secure biometric sensor. After a successful authentication and authorization, the `SPDMonitor` calls the trustlet (`cashier_trustlet`) that returns a challenge. The `SPDMonitor` then sends the challenge together with transaction details to the `customer_App` in the normal world of the customer's device via a secure channel.

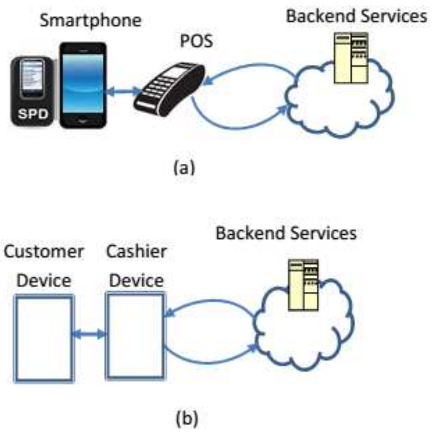

Figure 5: Secure payment scenarios: (a) smarphone with SPD connect to conventional POS; (b) customer and cashier with Light-SPD devices emulating smarphones and SPDs.

Similarly to the cashier, to do the payment the customer uses the `customer_App` to initiate an operation that involves contacting the `SPDMonitor` in the secure world of her device. The `SPDMonitor` displays the amount to pay and asks the customer for authorization to make this payment via its trusted GUI and authentication via the secure biometric sensor. After a successful authentication and authorization, the `SPDMonitor` calls the `customer_trustlet`. The `customer_trustlet` answers the challenge and performs the transaction by deducting from a local balance of the customer against a line of credit available for local payments. This method can work offline and reduce the device's energy consumption; otherwise the device can communicate with this app's remote server to get a payment token or directly make the payment to the cashier's account.

At this stage, the `SPDMonitor` also sends the answer for the challenge and if necessary the customer's credit card details to the `SPDMonitor` in the cashier's device which may confirm the availability of credit or funds by interacting with its remote server. Finally, the payment confirmation and details are displayed on the customer's trusted GUI (customer side) and on the cashier's trusted GUI (cashier side).

A brief explanation of why this process is secure is the following. If the user's smartphone was malicious it might modify the payment process to pay a different value and/or to a different vendor (stealing from the user). However, with our payment application this is not possible because every transaction is performed by the `customer_trustlet` after a successful authentication and authorization by the `SPDMonitor` in the secure world. If the cashier was malicious she might also modify the amount to request for a payment. In Light-SPD, this is not possible because the amount details are processed and sent securely by using the `SPDMonitor` and the `cashier_trustlet` isolated in the secure world.

6. EXPERIMENTS

This section presents the implementation of the Light-SPD and its experimental evaluation.

6.1 Implementation

We implement a prototype using a i.MX53 QSB.[3] The board is equipped with a Cortex-A8 single core with 1 GHz application processor, 1 GB DDR memory, and a 4GB MicroSD card.

Genode is a framework for building special-purpose operating systems [15]. It is a collection of small building blocks including applications and system software components (e.g. kernels, device drivers, and protocol stacks). Since requirements vary, Genode can reduce system complexity for each security-sensitive scenario. Due to its small TCB, Genode is an appealing foundation for an OS designated to run on the secure secure world. Genode Labs has released a TrustZone virtual machine monitor (VMM) demo for this board, which enables the execution of Genode in the secure world, while a guest OS (such as Linux) monitored by a Genode hypervisor runs in the normal world. We used this demo as a starting point to run Genode in the secure world.

In the secure world we run the Genode kernel. On top of the kernel runs a VMM that implements the `SPDMonitor` that handles hypercalls from the normal world. In the normal world we run Linux. We incorporated a driver (`TZ_Driver`) in the Linux kernel to issue an hypercall to exit the normal world, and trap to the secure world using the secure monitor call (`SMC`) instruction of the ARM security extensions. A shared buffer in normal world RAM is implemented to pass selected data across worlds. General purpose registers are used to store information about the shared buffer between the two worlds, including the physical address of the buffer and the length of the buffer. By doing this, application components that resides in the two worlds can interact.

We have ported the OpenSSL library (*libcrypto*) to run in the secure world on top of Genode in order to support a variety of cryptographic operations. Thus, the secure world can establish secure channels with remote servers via the network stack in the normal world. For secure display and user input, we use a serial console to simulate the secure display and a hardware keyboard as the secure input. To do this we setup the `UART` device as a secure world only peripheral, which means that software running in the normal world cannot access that device. For secure storage, we use the Genode partition manager (`part_blk`), which supports `MSDOS` as well as `GPT` partition tables and provides a block session for each partition on a SD card. This allows the partitions to be addressable as separate block sessions and makes it is easy to grant or deny access to them. In our platform, we use an SD card partition for storing the security sensitive data which is accessible exclusively by the secure word. In the current prototype, we allocated 2GB of secure storage space from a total of 4GB space available on the MicroSD card.

[3]We plan to release the prototype but it is not publicly available yet.

6.2 Performance

In this section we evaluate the prototype via both micro-benchmarks and a single macro benchmark.

6.2.1 *Micro-benchmarks*

We used a set of micro-benchmarks to evaluate the performance of our platform. We measured 4 latencies: context switch between worlds; data passing using the shared buffer; data encryption and decryption in the secure world. All the results we present are the average of 1000 operations.

To evaluate the time for context switching between the two worlds, we make `SMC` calls from the normal world and the secure world returns immediately without executing any trustlet (we modified `SPDMonitor` for this purpose). We measured an average context switching time of 0.045 *ms*.

We evaluate the overhead due to data transfer between the two worlds using the shared buffer, copying different sized chunks of data into the shared buffer and sending them to the secure world. Next the secure world retrieves the data from the buffer and switches execution back to the normal world. The results of these experiments are shown in the second line (Data Transfer) of Table 1.

In the payment application and generically in the Light-SPD platform when secure channels are used, a trustlet running in the secure world encrypts data before it sends it to the remote server and decrypts incoming data. We measured the time for these operations using different sizes of data and AES with keys of 256 bits. Note the key is previously stored in the secure world. We show these results in the two bottom lines of Table 1.

6.2.2 *Macro-benchmark - Payment application*

To evaluate the performance of the payment application, we measured the latency of two operations of the payment application: `pay()` to make a payment and `show_balance()` to check the available balance. We did not consider the user authentication and authorization times. We conducted this experiment with two payment methods: a local payment where all transactions are performed on the local machine/our board, and a remote payment, where all

Table 2: Time of the payment application operations (milliseconds).

Payment \ Operation	pay	show_balance
Local	0.16	0.15
Remote	1.58	1.52

Table 1: Time to transfer data between the two worlds and to encryt/decrypt data in the secure world (milliseconds).

Operation \ Data Size	10B	100B	1KB	10KB	100KB
Data Transfer	0.061	0.07	0.085	0.311	2.5
Data Encryption	7.1	8.0	8.2	9.4	27
Data Decryption	6.4	7.0	8.0	8.1	24.6

transactions are performed on a remote server. For the remote payment, we used a LAN network and the average round trip time (RTT) between the board and the remote server is 0.373 *ms*. We show these results in Table 2.

In the payment application the messages exchanged have less than 1 KB in size. Therefore, if we do not consider the user authentication, communication, and processing times, our platform introduces a delay of a few centiseconds, which is negligible in operations involving humans. Note that these measurements reflect preliminary measurements. As future work we expect to examine the details of caching between the normal world and the secure world in order to improve this performance.

7. RELATED WORK

There is a large amount of related work, of which we highlight four major areas: Android extensions to protect sensitive data, trusted execution environment, secure storage of such data, and secure mobile payments (the example used in this paper).

Extending Android to protect sensitive data.

Many approaches have been proposed to protect private data in Android smartphones. TaintDroid [11], AppFence [19], and AppIntent [40] control real-time flow of privacy-sensitive data through applications and expose possible privacy leaks. Saint [32], Aurasium [38], and Apex [30] enable applications to specify runtime constraints on the use of their sensitive data, e.g., an application can specify that another application with network access cannot read that data. Several systems, including FlowDroid [6] and ComDroid [8], detect privacy-sensitive data leaks by scanning the source code (or bytecode) without executing it. Another line of research protects sensitive data in Android devices by classifying applications along with their data into separate domains with different trust levels and denying data exchange between different domains. TrustDroid [7] provides a framework to isolate applications that are divided into corporate and private applications. All of the above solutions assume that the Android OS is trusted, i.e., part of the TCB, which is often not true in reality. In contrast, Light-SPD enables sensitive data to remain protected despite Android being compromised.

Trusted execution environment.

Another area of related work is on ensuring that sensitive code and data are stored, processed, and protected in an isolated, trusted environment. Researchers have provided trusted execution environments (TEEs) based on virtualization, e.g., Terra [14], that rely on the hypervisor to create and manage TEEs without any modification to existing hardware. Others leveraged the Trusted Computing Group's Trusted Platform Module (TPM) to provide higher assurance, e.g., [36, 26].

More recently hardware extensions appeared that support the creation of TEEs. The Intel software guard extensions (SGX) [4, 18, 27] is an Intel architecture designed to support the concept of an enclave, an isolated execution environment, similar to ARM's TrustZone [5]. An enclave is a protected area of code and data within an application. Unlike ARM TrustZone, SGX can have multiple protected enclaves in a system and their memory areas are separately encrypted. The access to an enclave's memory area by untrusted code including the OS, is protected by the processor. Iso-X [12] offers higher memory allocation flexibility than SGX, but is an academic work, not available in current processors.

Secure storage.

Recent research on secure storage of sensitive data on mobile devices has begun to leverage the ARM TrustZone extension. Several works have proposed mechanisms to protect secrets such as private keys, that are only accessible to small security critical programs, by using an isolated environment protected using the TrustZone [10, 21, 16]. Light-SPD is not restricted to storage of secrets, but allows biometric authentication and storage of large files, besides supporting the execution of hybrid applications with an untrusted and a trusted part. It also supports secure channels for communication with external backends running in the cloud. DroidVault [23] provides a secure data vault on Android devices using the TrustZone. DroidVault encrypts the private data downloaded from a remote server and stores the ciphertext in the untrusted Android file system. As a result, the data may be deleted by a compromised Android OS. In contrast, it is up to the specific trustlet whether to store encrypted data in the file system of the mobile OS or to store it in the secure storage component, with the latter providing increased protection against data loss. Unlike previous work, the Light-SPD can exploit biometric authentication mechanisms to authorize access to privacy sensitive data stored in the secure storage component.

Secure mobile payments.

Marforio, et al. [24] use a trusted application that runs in the ARM TrustZone secure world to verify the location of a cardholder during payments at the POS by using the phone's location. Light-SPD enables secure payments without using physical credit/debit cards. The Trusted Language Runtime (TLR) [37] and Pirker and Slamanig [35] provide a framework to separate application security logic, called a trustlet, from the rest of the application and run it in the ARM TrustZone secure world. This trustlet alone is responsible for ensuring a secure transaction during mobile payment. In contrast, Light-SPD can exploit computationally intensive services running in the cloud to reduce the size the TCB and the power consumption of the mobile devices.

There a few real payment services, which have to protect payment information during transactions. Android Pay [1] and Apple Pay [2] generate a virtual account number unique to each credit or debit card added to a user device. They use this virtual account number along with a payment token, a one-time security code, when making a transaction. Moreover, the payment information including the actual credit or debit card number are encrypted and stored on secure servers, not on users' devices.

8. CONCLUSIONS

In order to protect privacy, the PCAS project proposed the SPD to be connected to a smartphone via USB for securely storing sensitive personal data. However, this device is closed and currently unavailable. This paper presents the design, implementation, and experimental evaluation of the Light-SPD platform that allows experimenting with mobile applications for the PCAS SPD and smartphones, and provides sufficient security to allow deployment of such applications.

We have demonstrated the capabilities of the platform using a payment application. Light-SPD is open, easy to deploy, and secure with both an acceptable cost and performance.

9. ACKNOWLEDGMENTS

This work was partially supported by the European Commission through contract No. 610713 (PCAS), the Erasmus Mundus Doctorate Programme under Grant Agreement No. 2012-0030 (EMJD-DC), and by national funds through Fundação para a Ciência e a Tecnologia (FCT) with reference UID/CEC/50021/2013 (INESC-ID).

10. REFERENCES

[1] Android Pay. https://www.android.com/pay/.

[2] Apple Pay. http://www.apple.com/apple-pay/.

[3] Samsung KNOX. http://www.samsung.com/my/business-images/resource/white-paper/2013/11/Samsung_KNOX_whitepaper_An_Overview_of_Samsung_KNOX-0.pdf, 2013.

[4] I. Anati, S. Gueron, S. Johnson, and V. Scarlata. Innovative technology for CPU based attestation and sealing. In *Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy*, 2013.

[5] ARM. ARM security technology, building a secure system using TrustZone technology. http://www.arm.com,2009.

[6] S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, Y. Le Traon, D. Octeau, and P. McDaniel. Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for Android apps. In *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 259–269, 2014.

[7] S. Bugiel, L. Davi, A. Dmitrienko, S. Heuser, A.-R. Sadeghi, and B. Shastry. Practical and lightweight domain isolation on Android. In *Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices*, pages 51–62, 2011.

[8] E. Chin, A. P. Felt, K. Greenwood, and D. Wagner. Analyzing inter-application communication in Android. In *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services*, pages 239–252, 2011.

[9] L. Davi, A. Dmitrienko, A.-R. Sadeghi, and M. Winandy. Privilege escalation attacks on Android. In *Proceedings of the 13th International Conference on Information Security*, pages 346–360. 2011.

[10] J.-E. Ekberg, N. Asokan, K. Kostiainen, and A. Rantala. Scheduling execution of credentials in constrained secure environments. In *Proceedings of the 3rd ACM Workshop on Scalable Trusted Computing*, pages 61–70, 2008.

[11] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. Sheth. Taintdroid: An information-flow tracking system for realtime privacy monitoring on smartphones. In *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, pages 1–6, 2010.

[12] D. Evtyushkin, J. Elwell, M. Ozsoy, D. Ponomarev, N. Abu Ghazaleh, and R. Riley. Iso-x: A flexible architecture for hardware-managed isolated execution. In *Proceedings of International Symposium on Microarchitecture*, pages 190–202, 2014.

[13] A. P. Felt, M. Finifter, E. Chin, S. Hanna, and D. Wagner. A survey of mobile malware in the wild. In *Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices*, pages 3–14, 2011.

[14] T. Garfinkel, B. Pfaff, J. Chow, M. Rosenblum, and D. Boneh. Terra: A virtual machine-based platform for trusted computing. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles*, pages 193–206, 2003.

[15] Genode Labs. ARM TrustZone, an exploration of ARM TrustZone technology. http://genode.org/news/an-exploration-of-arm-trustzone-technology, 2014.

[16] Y. Gilad, A. Herzberg, and A. Trachtenberg. Securing smartphones: a micro-TCB approach. In *arXiv preprint arXiv:1401.7444*, 2014.

[17] M. C. Grace, W. Zhou, X. Jiang, and A.-R. Sadeghi. Unsafe exposure analysis of mobile in-app advertisements. In *Proceedings of the 5th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, pages 101–112, 2012.

[18] M. Hoekstra, R. Lal, P. Pappachan, V. Phegade, and J. Del Cuvillo. Using innovative instructions to create trustworthy software solutions. In *Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy*, 2013.

[19] P. Hornyack, S. Han, J. Jung, S. Schechter, and D. Wetherall. These aren't the droids you're looking for: retrofitting Android to protect data from imperious applications. In *Proceedings of the 18th ACM Conference on Computer and Communications Security*, pages 639–652, 2011.

[20] C. Kaufman. Internet key exchange (IKEv2) protocol. IETF Request for Comments: RFC 4306, Dec. 2005.

[21] K. Kostiainen, J.-E. Ekberg, N. Asokan, and A. Rantala. On-board credentials with open provisioning. In *Proceedings of the 4th International Symposium on Information, Computer, and Communications Security*, pages 104–115, 2009.

[22] M. La Polla, F. Martinelli, and D. Sgandurra. A survey on security for mobile devices. *IEEE Communications Surveys & Tutorials*, pages 446–471, 2013.

[23] X. Li, H. Hu, G. Bai, Y. Jia, Z. Liang, and P. Saxena. DroidVault: A trusted data vault for Android devices. In *Proceedings of the 19th International Conference on Engineering of Complex Computer Systems*, pages 29–38, 2014.

[24] C. Marforio, N. Karapanos, C. Soriente, K. Kostiainen, and S. Čapkun. Smartphones as practical and secure location verification tokens for payments. In *Proceedings of the Network and Distributed System Security Symposium*, 2014.

[25] C. Marforio, H. Ritzdorf, A. Francillon, and S. Capkun. Analysis of the communication between colluding applications on modern smartphones. In *Proceedings of the 28th Annual Computer Security Applications Conference*, pages 51–60, 2012.

[26] J. M. McCune, Y. Li, N. Qu, Z. Zhou, A. Datta, V. Gligor, and A. Perrig. TrustVisor:Efficient TCB reduction and attestation. In *Proceedings of International Symposium on Security and Privacy*, pages 143–158, 2010.

[27] F. McKeen, I. Alexandrovich, A. Berenzon, C. V. Rozas, H. Shafi, V. Shanbhogue, and U. R. Savagaonkar. Innovative instructions and software model for isolated execution. In *Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy*, 2013.

[28] W. Mostowski and E. Poll. Malicious code on Java Card smartcards: Attacks and countermeasures. In *Smart Card Research and Advanced Applications*, pages 1–16. Springer, 2008.

[29] National Computer Security Center. *Trusted Computer Systems Evaluation Criteria*. Number CSC-STD-001-83. Aug. 1983.

[30] M. Nauman, S. Khan, and X. Zhang. Apex: extending Android permission model and enforcement with user-defined runtime constraints. In *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, pages 328–332, 2010.

[31] S. Oaks. *Java Security*. O'Reilly, 2nd edition, 2001.

[32] M. Ongtang, S. McLaughlin, W. Enck, and P. McDaniel. Semantically rich application-centric security in Android. In *Proceedings of the 2009 Annual Computer Security Applications Conference*, pages 340–349, 2009.

[33] PCAS. Deliverable D5.2, SPD configuration and UI. https://www.pcas-project.eu/index.php/deliverables/, 2015.

[34] PCAS. Deliverable D5.3, secure data access. https://www.pcas-project.eu/index.php/deliverables/, 2015.

[35] M. Pirker and D. Slamanig. A framework for privacy-preserving mobile payment on security enhanced ARM TrustZone platforms. In *Proceedings of the IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications*, pages 1155–1160, 2012.

[36] N. Santos, K. Gummadi, and R. Rodrigues. Towards trusted cloud computing. In *Proceedings of the Workshop on Hot Topics in Cloud Computing*, June 2009.

[37] N. Santos, H. Raj, S. Saroiu, and A. Wolman. Using ARM TrustZone to build a trusted language runtime for mobile applications. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 67–80, 2014.

[38] R. Xu, H. Saïdi, and R. Anderson. Aurasium: Practical policy enforcement for Android applications. In *Proceedings of the 21st USENIX conference on Security Symposium*, pages 27–27, 2012.

[39] Z. Xu, K. Bai, and S. Zhu. Taplogger: Inferring user inputs on smartphone touchscreens using on-board motion sensors. In *Proceedings of the 5th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, pages 113–124, 2012.

[40] Z. Yang, M. Yang, Y. Zhang, G. Gu, P. Ning, and X. S. Wang. Appintent: Analyzing sensitive data transmission in Android for privacy leakage detection. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, pages 1043–1054, 2013.

[41] Y. Zhou and X. Jiang. Dissecting Android malware: Characterization and evolution. In *Proceedings of the 2012 IEEE Symposium on Security and Privacy*, pages 95–109, 2012.

[42] Y. Zhou, Z. Wang, W. Zhou, and X. Jiang. Hey, you, get off of my market: Detecting malicious apps in official and alternative Android markets. In *Proceedings of the 19th Annual Network and Distributed System Security Symposium*, pages 5–8, 2012.