Practical Privacy Preserving Cloud Resource-Payment for Constrained Clients

Martin Pirker[1], Daniel Slamanig[2], and Johannes Winter[1]

[1] Institute for Applied Information Processing and Communications (IAIK), Graz University of Technology (TUG), Inffeldgasse 16a, 8010 Graz, Austria
{martin.pirker,johannes.winter}@iaik.tugraz.at
[2] Department of Engineering and IT, Carinthia University of Applied Sciences, Primoschgasse 10, 9020 Klagenfurt, Austria
d.slamanig@cuas.at

Abstract. The continuing advancements in microprocessor technologies are putting more and more computing power into small devices. Today smartphones are especially popular. Nevertheless, for resource intensive tasks such devices are still too constrained. However, the simultaneous trend of providing computing resources as a commodity on a pay-as-you-go basis (cloud computing) combined with such mobile devices facilitates interesting applications: Mobile clients can simply outsource resource intensive tasks to the cloud. Since clients have to pay a cloud provider (CP) for consumed resources, e.g. instance hours of virtual machines, clients may consider it as privacy intrusive that the CP is able to record the activity pattern of users, i.e. how often and how much resources are consumed by a specific client. In this paper we present a solution to this dilemma which allows clients to anonymously consume resources of a CP such that the CP is not able to track users' activity patterns. We present a scenario which integrates up-to-date security enhanced platforms as processing nodes and a recent cloud payment scheme together with a concrete implementation supporting the practicality of the proposed approach.

1 Introduction

The sustained advancements in microprocessor technologies put more and more computing power into smaller and smaller devices. Today's *smartphones* essentially contain the computing "brainpower" of a desktop PC of not many years ago. This development, along with improved wireless connectivity, naturally fuels a shift in device usage patterns. It is no longer necessary to use a full-size desktop PC or laptop for certain tasks. Today, a smartphone is very well capable to perform many of the tasks desired by common end-users, such as browsing the web, or participation in social network services. Due to the small form factor, however, a smartphone's resources are limited in terms of battery power, storage, and consequently on available processing capacity.

A parallel trend is the rapid advancement of virtualisation technologies in conjuction with cheap storage and fast (wireless) Internet connections. This has

created a market for providing computing resources as a commodity – so called *cloud computing*. Large data centers can take advantage of the economics of scale and dynamically lease computing power or storage capacities to clients on demand. This trend promises to use IT resources more efficiently and to reduce costs.

Consequently, in the future the desktop PC may continue to loose importance and the *split processing model* may rise to dominance. In the split processing model small tasks are executed directly on end-user devices such as smartphones, tablets or other gadgets, while more complex and demanding jobs are delegated to remote cloud computing services where resources are leased on demand.

Contribution In this paper, we address the privacy concerns of the split processing model. We introduce a scenario where a low-resource client obtains cloud processing credits from a reseller and then uses them to pay for high-powered services at a remote cloud provider. Our main motivation for a "privacy-preserving by design" approach is that such cloud providers will be able to link data and information about resource consumption behaviour of their consumers (clients), allowing them to build dossiers. For many customers such transparency can be too intrusive. We want clients to be able to hide this information from cloud providers. As, for instance, argued in [9], activity patterns may constitute confidential business information and if divulged could lead to reverse-engineering of customer base, revenue size, and the like.

More precisely, we consider a setting where clients should be able to purchase a set of prepaid resources in form of cloud credits (CCs) from a reseller, e.g. represented as a scratch-off card. This card contains information which allows the client to obtain a single compact software token from a cloud provider that includes how many CCs a client is allowed to consume from this cloud provider. Then, clients should be able to consume their CCs from the cloud provider in an anonymous and unlinkable yet authorized fashion. For instance, if a client wants to consume n credits, he has to convince the CP that he possesses a valid token and he is still allowed to consume n credits. If this holds, the anonymous client is allowed to consume the resources and obtains an updated token corresponding to the remaining CCs in a privacy-preserving manner. Although the CP should be unable to track clients and determine how much credits a client has already consumed, it must be guaranteed that the client only consumes as much resources as CCs available in his token.

The novelty of our approach presented in this paper is that we use a recent *anonymous yet authorized and bounded cloud resource scheme* from [20] in a scenario which integrates *up-to-date security enhanced platforms as processing nodes*. We assume ARM *TrustZone* architecture enabled smartphone clients and *Trusted Execution Technology* attestable servers in the cloud. This scenario resembles a realistic use-case for a cloud computing environment and could be realised with today's mass-market hardware in the near future. We also provide experimental results from a prototypical implementation of our scheme on various current platforms.

Outline The remainder of the paper is structured into the following major sections. In Section 2 we present our scenario, in which we introduce all entities, their motivations as well as their privacy requirements. We then continue in Section 3 with preliminaries, including a brief background summary of the underlying *anonymous yet authorized and bounded cloud resource scheme* and the capabilities of trusted platform security technologies – in particular the *TrustZone* and *TXT* architectures. Section 4 introduces our concrete scheme and presents a description of all operations between the entities. We report practical implementation results in Section 5. We consider supplemental security, privacy implications, and trade-offs in Section 6. In Section 7 we present a brief overview of related work. Finally, Section 8 concludes the paper.

2 Scenario and High Level Description

In this section we present a practical scenario for deployment of our privacy preserving resource payment scheme. First we identify the core participants interacting and then describe the scenario. Then, we provide a high level description of the operations between the entities. We also discuss privacy issues for all involved entities.

2.1 Entities

In the scenario of this paper a small, resource limited *client* (C) wants to outsource computations to a powerful cloud datacenter.

The *cloud provider* (CP) professionally runs this large datacenter which takes advantage of the economics of scale. He rents computing power to anyone who can pay for the resources consumed. We make no assumption about what kind of resources are leased, we just define them to be accounted in discrete units of *cloud credits* (CC). Naturally, the client and the cloud provider must consent to a protocol so that the client can prove to the CP that he is eligible to consume a certain amount of resources. Obviously, client and cloud provider must interact directly when the client uses a CP service. However, before this step the client first has to obtain a certain amount of cloud credits.

As significant feature we thus introduce a third entity, the *credit reseller* (CR). The reseller obtains cloud credit units from the cloud provider in bulk and subsequently distributes and resells them through (small) distribution branches. This enables a wide range of use cases, such as cloud credits "gift-cards", and explicitly time-delayed, asynchronous interaction between the entities.

2.2 Scenario

We assume the client to be a state-of-the-art smartphone, based on an ARM platform with TrustZone capabilities. The smartphone is connected to the Internet, and thereby the cloud provider's servers, through a wireless connection to a mobile network. Figure 1 shows the major building blocks of our scenario and subsequently we reflect on the role of each:

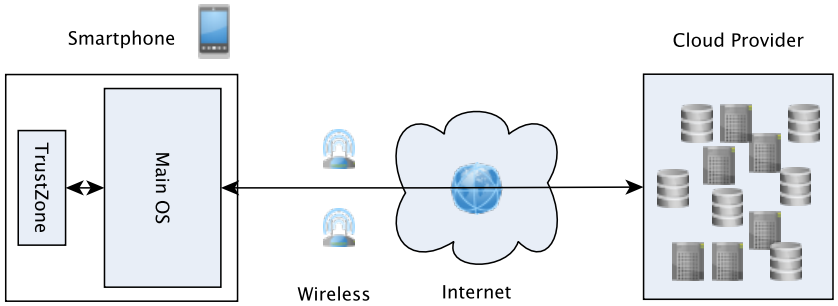

Fig. 1. Scenario in which a smartphone client connects to a cloud datacenter.

Cloud provider: Computing resources provided by the cloud provider are servers which were booted with a Trusted Execution Technology measured process (cf. Section 3.2). This enables the client to verify that he actually receives what he paid for.

Network: In our scenario the client is a smartphone in a mobile network, and thus the client is uniquely identifiable and his activities can be easily monitored by the mobile network provider. The network connection continues via the global Internet to the cloud provider. As privacy feature we assume the use of an anonymizing network like Tor [13] to hide the effective network address of the client from the cloud provider. Consequently, also the mobile provider is unable to log where the smartphone is connecting to[1].

Smartphone: The client device is essentially split into two processing worlds: The normal-world which hosts the mobile's OS, e.g. Android as is common today, and end user applications. There is also a small isolated execution area – the secure-world – for sensitive data and operations. The link between the two worlds is strictly monitored and only well-defined calls are allowed[2].
The client's cloud credits accounting and private data storage is done in the secure-world. Once the initial credit data structures (representing the token) are imported, they never leave again (only in the case when they are transferred to another client).

2.3 High Level Description of the Operations

We model the flows of cloud credits in our scenario as essentially a circle between our three entities. The cloud provider is the central entity. He is responsible for providing resources and requires clients to pay for the consumed resources by providing the necessary number of CCs to him. Clients who want to consume resources from a cloud provider need to acquire a certain number of CCs from a credit reseller beforehand. This transaction is carried out without involving the

[1] We ignore sophisticated global network surveillance scenarios and defer discussion of these security edge cases to the Tor community.
[2] See e.g. [24] for a practical realization of this approach.

cloud provider and thus can be carried out offline. After acquiring the credits from the reseller, clients can activate the credits at the cloud provider, maybe at some later point in time. Clients may not only consume acquired credits by themselves, but may also give or sell them to other clients. Furthermore, if clients pay too many credits at the cloud provider for a certain task, e.g. because they do not know how many resources the task actually requires beforehand, then a cloud provider can issue vouchers for unused credits. A graphical sketch of our high-level operations for the client, cloud provider, and credit reseller setting is given in Figure 2. Subsequently we present a high level description of the operations.

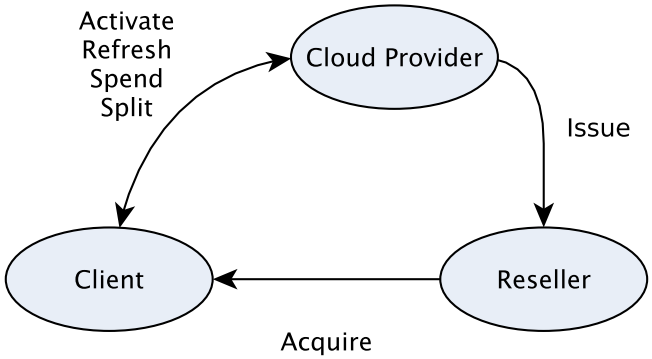

Fig. 2. Core Payment Scheme Operations

Issue: This transaction is an initial bulk transfer of cloud credits (CC) from the cloud provider (CP) to the credit reseller (CR). It is a personal business relationship operation and assumed to happen in a secure and reliable way.

Acquire: This transaction is carried out between a client (C) and a CR. At the end of this transaction C holds an amount of CCs, which cannot be used before they are activated.

Activate: This transaction is carried out between C and the CP. Essentially, C ends up holding an amount of activated CCs (in form of one compact token), which may have a limited validity.

Spend: This transaction is carried out between a client C and CP, where C pays a certain number of CCs and consumes an equivalent amount of resources from CP. The client ends up with the result of the task performed at the CP, and the remaining number of activated CCs.

SplitCredits: This is a transaction between C and CP in which C ends up with (at most) two valid tokens. Let us assume that C wants to split off n CCs from his collection of m CCs (if $m > n$)[3]. Then the first token contains $m - n$ CCs and the second token n CCs.

[3] Otherwise, if $m = n$ holds the first token will not be valid any more and if $m < n$ holds, the second token will only contain m CCs. Nevertheless, how such an operation

Refresh: This transaction is carried out between C and CP. It is required when CCs have a limited validity. When the validity period ends, C carries out this transaction and gets issued the same amount of activated credits he already holds, but valid within the new period.

Transfer (not shown in Figure)**:** This transaction is carried out between two clients C_1 and C_2, whereas C_1 transfers a number of activated CCs to C_2 and C_1's CCs are deleted, i.e. we explicitly want to support *transferability.*

2.4 Privacy Issues

With the entities known we now discuss the privacy requirements which are effective in our scenario. We assume that two entities can always connect to each other using a private, i.e., confidential and authenticated, channel[4]. Note that this does not mean that the entities need to mutually authenticate to each other using their identities. Essentially, for our scenario we will realize privacy by anonymity and unlinkability of conducted transactions. This presents a good compromise in which clients have privacy guarantees while at the same time the other entities can be sure that only authorized actions can be conducted.

Cloud provider From a privacy perspective the cloud provider is in the position to be honest, but curios. Thus, he has the means to sniff, log and track the cloud's activities and is subsequently able to figure out who his clients are and what they are doing. He must be able to demonstrate a certain level of security of his services-for-rent. Without such a proof, customers will eschew doing business with him as they cannot be sure what is happening with their data.

Credit reseller The relationship between the cloud provider and the credit reseller is a business one. By definition, the cloud provider wants a reliable and clearly identifiable reseller, and the reseller wants to ensure that he is dealing with the correct provider contact point. From this follows the requirement for secure communications and transactions between the two when they trade credits for money – and privacy is not important in this context.

The reseller is the entity who comes into real-life contact with the end-user. Consequently, the privacy of the client very much depends on the method of payment for the prepaid cloud credits offered and how the actual handover is organized. Similar to the cloud provider, the reseller has no motivation to invest extra effort to identify his clients without additional external pressure.

Client The client is the entity motivated to ensure that privacy is enforced at all steps in the transactions. First he has to obtain credits from the reseller. A straightforward solution is to pay with money in cash, an intuitive way to ensure

ends is application dependent and one may also wish to terminate the operation if $m < n$.

[4] e.g., using Transport Layer Security (TLS) [11] over a Tor [13] connection

privacy of payment. By definition the client already knows which cloud provider he wants to use, thus he must be able to verify that the offered credits from the reseller are genuine. Second, for establishment of an untraceable, i.e. anonymous and unlinkable, connection to the cloud provider the client uses an anonymous communication network, e.g. Tor.

The process of exchanging cloud credits for resources represents the most challenging issue. *How to consume them at the cloud provider without the cloud provider being able to identify and track clients?* The most straightforward non-privacy friendly solution for client payment is to require clients to register with the cloud provider and to have the cloud provider maintain an account for every client. The provider then bills all consumed resources to this account. In this scenario clients could pay for their consumed resources on a regular basis. However, this does not satisfy our privacy demands and we want to achieve the following privacy requirements for the payment process:

Anonymity Clients do not want the cloud provider to be able to identify them. Assuming that we have an anonymous channel between the client and the cloud provider, this means that none of the operations Activate, Spend, SplitCredits and Refresh must involve any information that can be used by the cloud provider to identify a client. When we present our approach in Section 4 it will be clear that anonymity solely depends on the underlying scheme for which it is shown in [20] that anonymity is given.

Unlinkability Clients do not want the cloud provider to be able to link different consumptions of resources (spending of credits), nor to discover how many credits a client still possesses. This gives the client the guarantee that he is unobservable from the cloud provider's perspective. As in case of anonymity, the operations Activate, Spend, SplitCredits and Refresh must be unlinkable and this follows from the underlying scheme as shown in [20].

3 Background and Preliminaries

This section is devoted to building blocks and technologies that are employed by our concrete scheme and deployment scenario. First we provide a compact description of the underlying cryptographic scheme and present some abstractions used throughout the paper. We then provide a brief overview of trusted platform security technologies, how they allow attestation of a platform's state, and how they support isolated, secured processing areas.

3.1 Anonymous Cloud Resource Scheme

In the following we represent all cloud credits (CCs) of a client as a single token, i.e. one may think of a wallet of CCs, and a consumption of CCs from this token can be thought of as spending some CCs out of the wallet.

We start by reviewing the *anonymous yet authorized and bounded cloud resource scheme* (AABCRS) introduced in [20], which is the most important building block for the work presented in this paper. The main idea behind such schemes is that clients are able to purchase a contingent of credits for resources, such as virtual machine instance hours, from a cloud provider (CP). While clients spend their credits for resources at the CP, the CP does not learn anything about the resource consumption behaviour of users. In particular, users can consume an arbitrary number of their credits as long as there are still enough credits from their purchased amount. Thereby, in any interaction with a client, the CP is convinced whether a client is allowed to consume resources, but cannot *identify* the client nor *link* any of the client's actions and thus cannot figure out their consumption patterns.

Below we present the definition of [20] and slightly modify it by discarding one of the protocols (the `Reclaim` protocol). This modification, however, has no impact on the concrete scheme. Hence, an AABCRS is a tuple (`ProviderSetup`, `ObtainCredits`, `Consume`) of polynomial time algorithms or protocols between clients C and a cloud provider CP and works as follows:

- `ProviderSetup`. On input a security parameter k, this algorithms outputs a private key sk and a public key pk of a suitable signature scheme and an empty blacklist BL which is required for double-spending detection.
- `ObtainCredits`. In this protocol a client c wants to obtain a token t for L credits (the credit limit) from the CP. The client's output is a token t with corresponding signature σ_t issued by CP. The token contains the credits L and the up to now consumed credits s. These values may be represented by a single value $L' := L - s$ within the token. Clearly, initially no credits have been spent. The output of CP is a transcript T_{OL} of the protocol.
- `Consume`. In this protocol a client c wants to consume n credits from the credits still available in his token. The client shows a value id (a unique token identifier) of a token t and convinces the CP that he holds a valid signature σ_t for token t and that t contains at least n credits. If the token was not already shown in a previous run of a `Consume` protocol, i.e. $t.id \notin BL$, the signature is valid and there are still enough credits available in the token, i.e. $s + n \leq L$ (or $L' - n \geq 0$ if both values are represented as a single one), then c's output is `accept` and an updated token t' (with new id) for credit limit L and up to now consumed credits $s + n$ (or $L' - n$ if both values are represented as a single one) with an updated signature $\sigma_{t'}$ from CP. Finally, CP includes id into BL. Otherwise the user's output is `reject`. The output of CP is a transcript T_C.

In [20] the author presents two variants of an AABCRS scheme based on the pairing based Camenisch-Lysyanskaya (CL) signature scheme [7]. In both variants, a token t is represented as an ordered sequence of values, whereas the number of elements depends on the variant. In the first Variant (V1), a token is of the form $t = (C(id), C(s), L)$, whereas $C(x)$ denotes an unconditionally hiding commitment to value x. The value id represents a unique identifier of

the token, s represents the number of CC's that have been consumed up to now and L represents the credit limit. The drawback of V1 is that L is included in plain in the token and thus is always visible to the CP. Hence, in the worst case, i.e. L is issued to exactly *one* client, the cloud provider can link actions of this client. However, if the set of clients associated to the same value L is reasonable large, unlinkability is no longer a problem in a practical setting[5]. In the second version (V2) even the value L is hidden from the cloud provider and a token is of the form $t = (C(id), C(s))$. Here, s represents the number of CCs that are still available from this token.

Intuition behind V1 [20] We briefly sketch the idea of V1 for simplicity, whereas the modifications for V2 are straightforward (see [20]): The main intuition of the construction is to let a client solely prove in each `Consume` protocol that enough cloud credits are still available in a token. Therefore, the cloud provider generates a key-pair (sk, pk) for the CL signature scheme, publishes pk and initializes an empty blacklist BL. Then, a client obtains a CL signature σ_t for a token $t = (C(id), C(s), L)$, whereas initially no CCs s are consumed. Let us assume that the client holds a token $t = (C(id), C(s), L)$ and corresponding signature σ_t. It is important to note, that id (a random token identifier) and s were signed as commitments and thus the CP is not aware of these values. If a user wants to consume n CCs from his token, he computes a commitment $C(id')$ representing the updated token identifier for the updated token, randomizes the signature σ_t to σ'_t (σ_t and σ'_t are then unlinkable) and proves in zero-knowledge that σ'_t is a valid signature for id and L. This includes showing the values id and L to the CP. Additionally, the client proves that the token includes an unknown value s, which satisfies $(s + n) \in [0, L]$ or equivalently $s \in [0, L - n]$. This proves convinces CP that at least n CCs are available from this token. If id is not contained in BL and all proofs succeed, then the client is eligible to consume n CCs. Consequently, the signature will be updated to a signature for $C(id + id')$, $C(s + n)$ and L in an interactive manner between the client and CP. Subsequently, the CP adds id to BL and the client obtains an updated signature for an updated token $t' = (C(id + id'), C(s + n), L)$. This signature can be randomized by the client (which makes t and t' unlinkable) and used for the next consumption of resources. Otherwise, CP will reject the client's request to consume n resources.

In the following we abstract from the details of the scheme, and use the following high level parameters and state information respectively:

- The public parameters $\mathsf{cpparams}_{pub}$ represent the public key pk of the CL signature scheme and are public knowledge.
- The private parameters $\mathsf{cpparams}_{priv}$ represent the private key sk of the CL signature scheme and the blacklist BL and are solely known to the cloud provider.

[5] Note that cloud resellers may sell cloud credits only in specific well-known amounts, similar to cards for prepaid mobile phones, e.g. 5$, 10$, etc.

– The state cstate_{pub} represents the actual token-signature pair of the client.
– The state cstate_{priv} represents the values id, s, (L in $\mathsf{V1}$) as well as the randomizers for the commitments and the randomization factors of the CL signature of the client's token. The state information is updated during every `Consume` operation, since only the current values are required.

We note that it is not necessary to keep the token-signature pair $\mathsf{cpparams}_{pub}$ secret, since without knowing cstate_{priv} no one will be able to convince CP that the signature σ_t is a valid signature for token t.

3.2 Trusted Platforms

In the last few years, mass-market computer platforms and devices have been enhanced with functions dedicated to support advanced security. In the following we give a short introduction to the features available in industry standard PCs as well as mobile platforms.

Trusted Platform Modules The concept of Trusted Computing as promoted by the Trusted Computing Group (TCG) extends the industry standard PC architecture with a specialised hardware component, the Trusted Platform Module (TPM) [22]. A TPM features cryptographic primitives similar to a smartcard, but is physically bound to its host platform.

An important concept of Trusted Computing is the measurement logging and reporting of the platform state. Upon platform hardware reset a special set of platform configuration registers (PCRs) in the TPM are reset to a well defined start value. PCRs cannot be directly written to, rather, a PCR with index i, $i \geq 0$, in state n is extended with input x by setting $PCR_i^{n+1} = \text{SHA-1}(PCR_i^n || x)$. This enables the construction of a chain-of-trust. From the BIOS onwards, every block of code is measured into a PCR before execution control is passed to it. Thus, the current values in the set of PCRs represent a log of what happend since system reboot, up to the current state of the system. The current state may then be TPM signed with the TPM *Quote* operation and reported in a so-called *remote attestation* protocol.

Intel Trusted Execution Technology Recent platforms from Intel[6] extend the basic TCG model of a *static* chain-of-trust from hardware reboot and trust rooted in early BIOS. They provide the option of a *dynamic* switch to a well-defined, measured system state [16], meaning at any point of execution after platform reboot. Consequently, this capability significantly cuts down the complexity of the chain-of-trust measurements to assess the platform state by excluding the early, messy bootup operations, leading to a simpler and practical implementation.

[6] We restrict our discussion to Intel's Trusted Execution Technology (TXT) as this is currently the dominant technology provider – comparable features are also available on e.g. AMD platforms.

ARM TrustZone Many computing devices, especially in the embedded and mobile domain, and more recently also in high-density data centers, do not use x86 microprocessors, but are rather powered by a processor of the ARM family. The ARM architecture follows a building-blocks approach, where the main processor design is developed and controlled by one company. Individual vendors select and license the intellectual property of the core and desired support components as needed, and then enhance them with their own functional units according to the needs of their customers.

The *TrustZone* architecture extension for ARM CPUs is an instruction set extension for security critical scenarios. Basically, it provides a separation of the memory resources and the CPU of a device, thereby creating two virtual domains which are the so-called *secure-world* (SW) and *normal-world* (NW) [3]. This approach is an improvement to the basic concept of privileged/unprivileged mode-split which can be found on many conventional architectures, including earlier ARM cores.

The normal-world is the containment for user programs or any kind of untrusted applications. Security critical code is executed in the secure-world. The isolation mechanisms of the TrustZone prohibit normal-world applications from accessing the secure-world. Consequently, the data flow between both worlds is controlled by a secure monitor entity, which is under control of the secure-world.

The total memory available for software inside the TrustZone is vendor dependent and ranges from 64 kBytes up to 256 kBytes on typical systems. This size enables the running of a small – hopefully evaluated and certified – core in the secure-world along with trusted executables.

Due to the ARM building-block approach there is no standardized way to report the genuinity of the TrustZone implementation of a certain vendor. However, typically TrustZone implementors provide a symmetric device-key for device identification and a asymmetric key for the purpose of secure boot. This allows construction of hardware authentication similar to that implemented by a TPM device and enforcement of a measured boot chain like with Intel TXT.

4 Practical Anonymous Payment

Building on the scenario, definitions and restrictions outlined in Section 2 we are now ready to present our resource payment scheme in more detail. When we write $\mathsf{A}(C(a_1,\ldots,a_n),CP(b_1,\ldots,b_m))$ we mean that operation A is run between entity C with private inputs $a_1,\ldots,a_n$ and entity CP with private inputs $b_1,\ldots,b_m$. All operations are conducted by the entities client (C), cloud provider (CP) and credit reseller (CR). Furthermore, we will use the algorithms of an AABCRS as defined in section 3.1 as subroutines.

We note, that the Acquire operation can be conducted by several means. One suitable and also privacy friendly scenario is to require the cloud provider to hand over scratch-off cards to the credit reseller. These scratch-off cards are of different monetary denominations representing some equivalent of cloud credits (CCs). If a user buys, e.g. with cash, such a card at a CR, he can scratch off the

opaque covering and a QR-Code is revealed. This QR-Code contains information about the denomination, a serial number and potentially a validity period along with a digital signature for those values. A client can scan this QR-Code with the built-in camera of his smartphone[7] and then holds all information necessary to conduct an Activate operation with the CP. We denote the information which is necessary for the activation as params_{act} subsequently. Note, that the aforementioned approach is advantageous from a privacy perspective, since CR does not learn the serial number of the card and thus cannot link (in cooperation with the cloud provider) the Acquire operation to the respective Activate operation.

Now, we present the remaining operations in a more formal manner, whereas we assume for simplicity that the CP provides *one* type of resource and has already conducted the `ProviderSetup` procedure. Furthermore, we denote by NW the normal-world and by SW the secure-world of the client's platform.

Activate$(C(\mathsf{params}_{act}, \mathsf{cpparams}_{pub}), CP(\mathsf{cpparams}_{pub,priv}))$**:** The NW sends params_{act} to the CP, who verifies them for validity and returns `true` or `false` to C. In case of `true`, C imports $\mathsf{cpparams}_{pub}$ into SW and C's SW runs an `ObtainLimit` protocol for CC limit L (contained in params_{act}) with CP. The client ends up with storing cstate_{pub} in the NW and cstate_{priv} in the SW. If the CP returns `false` in the first interaction, the operation terminates.

Spend$(C(\mathsf{cstate}_{pub,priv}, n), CP(\mathsf{cpparams}_{pub,priv}))$**:** The NW sends the number n of desired CCs along with cstate_{pub} to SW. If enough CCs are still available ("in" the token) then SW runs a `Consume` protocol to consume n CCs with CP, otherwise it returns `false` and the operation terminates. Thereby, CP obtains a proof that cstate_{pub} represents a valid token-signature pair, there are still enough CCs available and the token id not already contained in the blacklist BL. If any check fails, the operation terminates. If all checks succeed, the SW updates cstate_{priv} and obtains an updated token-signature pair cstate_{pub}, which is stored in NW.

SplitCredits$(C(\mathsf{cstate}_{pub,priv}, m), CP(\mathsf{cpparams}_{pub,priv}))$**:** The NW sends m along with cstate_{pub} to SW. Let us assume that $n > m$ whereas n represents the number of remaining CCs in cstate_{priv} for simplicity. Essentially, the operation works identical to the Spend operation, but the m resources are not consumed. Instead, an additional cstate'_{pub} is returned to C. At the end of this operation C holds cstate_{priv}, cstate'_{priv} (in SW) as well as cstate_{pub} and cstate'_{pub} (in NW), whereas the former token represents $n-m$ and the latter m CCs.

Transfer$(C_1(\mathsf{cstate}_{pub,priv}), C_2(\cdot))$**:** The SW of C_1 exports cstate_{priv} and the public and private state information $\mathsf{cstate}_{pub,priv}$ are transferred to C_2 who imports cstate_{priv} into his SW and cstate_{pub} into his NW. The SW of C_1 deletes cstate_{priv}. Note that C_1 transfers all n CCs represented by cstate_{priv} to C_2.

Refresh$(C(\mathsf{cstate}_{pub,priv}), CP(\mathsf{cpparams}_{pub,priv}, \mathsf{cpparams}'_{pub,priv}))$**:** C's NW sends cstate_{pub} to the SW and SW sends cstate_{pub} along with cstate_{priv} (rep-

[7] The use of a mobile phone camera to provide a trusted import path for cryptographic data was demonstrated viable in the Seeing-is-Believing effort [17].

resenting n CCs) to CP. Now, CP can check whether cstate_{pub} represents a valid token-signature pair for n CCs. If this is true, CP engages in an `ObtainLimit` protocol with respect to new parameters $\mathsf{cpparams}'_{pub,priv}$ with C and issues a token for n CCs.

We additionally observe the following:

SplitCredits: A client may have several motives to invoke a SplitCredits operation. For instance, a client may want to "split" off some CC's from his token to obtain a new token, in order to give one of the tokens to someone else, e.g. as a gift. Another scenario is that a client pays n CCs for some computation, but the computation actually only requires $m < n$ CCs. Then, after having conducted the computation, CP issues some kind of voucher in form of a new token to C for $n - m$ CCs.

Refresh: The version of the Refresh operation presented here is the simplest one. Essentially, the client is issued fresh CCs with respect to new CL signature parameters, i.e. every validity period is represented by distinct signature parameters. Note, that providing unlimited validity of tokens would not scale well, since CP would have to store the entire blacklist for double-spending detection. More flexibility can be achieved if validity periods are encoded directly into the tokens as it is proposed as an extension in [20].

Spend: For every Spend operation at least one CC is removed from circulation and at least one new entry in the blacklist BL is required to prevent double-spending. Naturally, the amount of CCs in circulation must be known to the CP as he must be able to manage a blacklist. Consequently, the maximum amount of credits issued is bound by the maximum size of the blacklist. The policy of the CP must enforce a periodical Refresh operation by the clients – in effect accounting periods – to allow periodical clearing of the blacklist (of the expired period).

5 Implementation

For practical evaluation we prototyped the core anonymous payment operations of our approach on multiple software and hardware platforms. Some are good approximations of the current generation of smartphones.

5.1 Specifications

On the software side, our scheme was implemented in the high-level language Java and uses the jPBC 1.2.0 library[8], a library for Pairing-Based Cryptography (PBC) in Java. As an alternative to the pure Java implementation there is also a C implementation of PBC[9], which can be called from Java via a Java-to-C wrapper. In the following we denote these two setups as Java "-J" and Native C

[8] `http://gas.dia.unisa.it/projects/jpbc/`
[9] `http://crypto.stanford.edu/pbc/`

accelerated "-C" variants. Our platforms ran either Linux (Li) or Android (An) as operating system. The platforms used to measure execution speed were as follows:

Pc* Laptop HP 8440p Elitebook, Intel i7M620 @2,67 Ghz, running Android for x86 2.3.5 (RC1 20110828) of the Android_x86 porting effort [1], or Ubuntu 11.10 with IcedTea6 1.11pre (OpenJDK 64-Bit Server VM (build 20.0-b11)

Ek* Freescale i.MX51 evaluation kit [15], Freescale MX515D @800Mhz[10], running Android 2.3.4 (build R10.3.2_3), or Ubuntu 10.04 LTS with IcedTea6 1.8.10 (OpenJDK Zero VM (build 14.0-b16))

SpGs Smartphone Google Nexus S, Samsung Exynos 1 GHz (ARM Cortex-A8), running Android 2.3.6.

SpS2 Smartphone Samsung Galaxy S2, Samsung Exynos 1.2 GHz dual-core (ARM Cortex-A9), running Android 2.3.3.

5.2 Results

For performance evaluation we focused on the Activate and Spend operations conducted by the client. This is due to the fact that the remaining protocols only require negligible computational resources or are based on one of the two aforementioned protocols, i.e. perform identically. The first one being run only once for initialisation of the credits token, the latter being run everytime credits are spend at the CP. Table 1 shows the results from our implementation on the platforms presented in Section 5.1. We measured from 4 to 16 bits for a practical cloud credits limit of $2^4 = 16$ to $2^{16} = 65536$ credits. The credits token is only valid at one specific provider and these limits enable many basic use cases. A larger limit is always possible, if the resulting additional computation time is acceptable for the client.

	PcLi-C	PcAn-C	PcLi-J	SpS2-C	SpGs-C	EkAn-C
Activate	0.06	0.15	0.35	0.64	0.86	1.12
Spend 4 bits	0.16	0.35	0.82	1.43	1.94	2.54
Spend 16 bits	0.34	0.77	1.72	2.99	4.11	5.32
	EkLi-C	PcAn-J	SpS2-J	SpGs-J	EkAn-J	EkLi-J
Activate	1.09	1.80	6.76	11.1	15.7	19.8
Spend 4 bits	2.53	3.60	13.3	20.9	29.6	39.7
Spend 16 bits	5.42	6.87	24.4	41.7	54.7	77.3

Table 1. Execution time of Activate and Spend 4 bits to 16 bits [s]

Figure 3 provides a more detailed analysis of the Spend protocol for tokens containing 2^x CCs. As can be seen from the figure, the time required for the

[10] The processor on this board is based on ARM's Cortex-A8 core and supports advanced security features such as ARM's TrustZone and secure boot facilities. Unfortunately, most parts of the documentation is only available under NDA from Freescale. For this prototype effort this is sufficient.

run of a Spend protocol grows linearly[11] in the number of bits x of CCs in the activated token, which is due to required zero-knowledge range proofs. We do not provide explicit timings for computations of the cloud provider, since he uses state-of-the-art servers and the computations are very efficient (the most expensive operation of the CP, i.e. `Consume` for a limit of 2^{30} CCs, reported in [20] takes about 1 second).

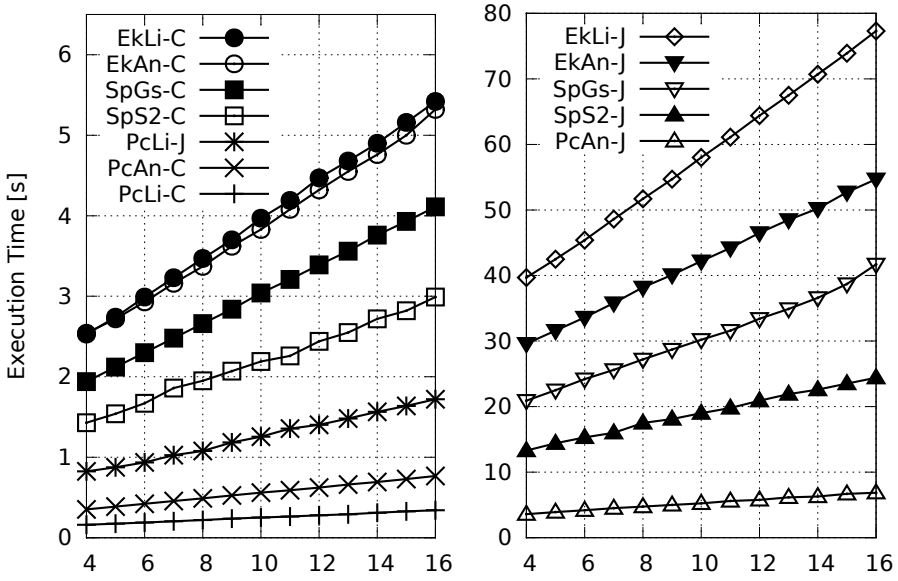

Fig. 3. Spend execution speed, x-axis shows credits token limit $L = 2^x$ [bits]

Our results clearly show that the *PC** versions dominate, the C versions take full advantage of the raw processor power. We interpret the difference between *PcLi-C* and *PcAn-C* due to the first being 64bit and the second a 32bit platform. They are closely followed by *PcLi-J*, a pure Java version executed by a server JVM optimized over many years. The next 4 places are claimed by the remaining C builds, as expected by their platform processor powers: 1.2GHz *SpS2-C* before 1.0GHz *SpGs-C* and 800MHz *Ek*-C*. The *EkLi-C* build runs almost identical to *EkAn-C*. The Java versions trail, again as expected by their processor speed. For the last platform, EkLi-J, the ARM JVM port appears to be quite unoptimized.

We must note that all Androids except *PcAn-J* and *SpS2-J* exhibited slight process memory leakage during execution. Despite continuous garbage collector runs memory usage grew. We assume this issue to add GC runtime overhead, therefore their execution time should actually be a little lower. So far we have been unable to determine the cause of this problem.

[11] With the non-linearities in the measurements caused by OS system services running in the background.

The main Java classes of our basic test code only consume 25 kB in size. The supporting Java libraries (*jpbc-api.jar*, *jpbc-plaf.jar*, *jpbc-pbc-jni.jar*) require 371 kB. The native C support libs (*libgmp.so*, *libjpbc-pbc.so*, *libpbc.so*) sizes vary according to the specific platform: Linux_x86_64 772 kB, Android_x86_32 688 kB, Linux_Arm 588 kB and Android_Arm 800 kB.

5.3 Discussion

The results of our practical evaluation support the feasibility of our scheme. While the Java numbers may appear to be very high at first glance, the C core accelerated versions are multiple times faster. To achieve this, first we replaced the default JNA wrapper for the Java-to-C bridge with our own custom JNI coded wrapper to allow the C code to be accessible on Android. Second, the ARM code was compiled to take advantage of ARM processor *Neon* SIMD and ARMv7 instructions features[12]. Consequently, an optimized, C core enhanced, Android build runs fast enough on current generation smartphones to allow Spend(ing) operations without long delays for end users.

Our prototype uses off-the-shelf libraries for crytography which were not optimised at all as initial evaluation of execution speed was our first objective. Consequently, our test code plus support libraries and JavaVM are way too large to fit into a TrustZone environment (Section 3.2). However, we expect a standalone implementation of the algorithm along with a small JavaVM to fit into a small TrustZone environment. This remains future work.

6 Discussion

In the following we reflect on supplemental aspects which were not discussed during the presentation of our scheme or our scenario.

Trusted Computing Attestation In Section 2.2 we use Trusted Execution Technology to attest the cloud servers provided to the client. This technology is mass-market available and TXT integration has already been demonstrated for Linux based servers [21]. Thus, if the software image to be rented to the client is agreed upon, a remote client can ask for a remote attestation proof (*TPM_Quote*) from the server to confirm what specific software image was actually booted. Attacks of the TXT components require physical intervention [25] and consequently raise the bar for manipulation[13].

The execution of multiple parallel running client VMs on a cloud server and subsequent attestation of the security of the system is an active topic of Trusted Computing research. The current TPM chip generation was not designed for this use case, an updated revision of the TPM hardware "V2" is expected to make this use case practical.

[12] GCC CFLAGS="-march=armv7-a -mfloat-abi=softfp -mfpu=neon"

[13] Or good old runtime software bugs which allow priviledge escalation.

Identification at credits purchase An obvious privacy problem is a potential camera at the reseller's place, which may collect a photo of the client's face. This threat is mitigated with the possibility of scratch-off cards and the ability to Transfer credits between trustworthy friends.

Isolated secure world By definition computation in the SW is isolated from the rest of the platform and only reachable via a well-defined, narrow interface. Consequently, an obvious problem is to decide from inside the SW whether a request from the NW is authorized – or not. This problem is non-trivial, but one solution would be to require a trusted input and display path which provides user-interaction (e.g. PIN entry) if explicit authorization for sensitive operations in the SW is required. We assume that users are acting in their own best interests when using their own smartphones, which is reasonable. Nevertheless, this still leaves the problem of potential malware on the client's platform which would be able to circumvent this feature. Note that this isolation also impacts the privacy provided by our network connection, as the Tor connection is currently anchored in the NW. Thus, how could the limited SW verify whether an anonymizing network connection to the CP is properly used?

Honesty of clients If two clients exchange cloud credits by means of a Transfer operation, say C_1 gives n CCs to C_2, then C_2 has no means to verify whether C_1 has properly deleted the transferred credits. Essentially, if C_1 is dishonest the first-come-first-served principle applies and whoever is the first one to spend credits from the token will be able to continue spending. We may, however, assume that only clients who trust each other exchange CCs, in which case this is not a problem. Furthermore, from the perspective of CP, even if C_1 does not delete his CCs, then this does not mean any harm to the CP, since only n CCs can be consumed in total and clients cannot create "extra" credits.

Forward Secrecy What happens when a smartphone is stolen, lost or seized? If Spend is not protected by additional secret information, e.g., a PIN, then someone in possession of the smartphone is able to spend all credits left within the currently activated token. Nevertheless, we note that even if the adversary is able to extract information from the secure world, he will not be able to link previous actions of the smartphone's holder – since no "history data" of the randomization processes of the underlying scheme is stored.

7 Related Work

To the best of our knowledge an approach related to the one presented in this paper has not been considered before. Nevertheless, there are three lines of work whose combination leads to the kind of work presented here. We will briefly elaborate on this below.

Privacy in trustworthy mobile platforms With the growing popularity of mobile computing, their ubiquitous application and the resulting privacy issues, there arises the necessity to provide functionalities of traditional privacy enhancing cryptographic protocols. However, due to limited storage capacities and processing power this task is non trivial and requires clever design. Recent works include the design of anonymous authentication for mobile devices by modifying direct anonymous attestation (DAA) and using hardware security features to prevent copying and sharing of private credentials [23]. Another implementation of DAA on mobile platforms with TPMs is presented in [12].

Privacy in cloud computing: Privacy is considered as one of the main issues in cloud computing. Besides known problems regarding user's privacy in traditional web applications, additional aspects imposed by the heavy use of virtualization seem to be novelties. For instance, sharing of resources among different users may potentially lead to the construction of covert or side channels which allows to infer activity patterns of other users (cf. [9]). User's access patterns represent privacy sensitive information that should also be hidden from the cloud. Recent works are mainly focusing on storing and sharing data in the cloud. In [14] an oblivious RAM (ORAM) based approach is presented, which allows users to outsource a set of data items to the cloud and provide read and write permission to users as follows. Users can only access (and read in plain) data items when accurate permissions were obtained and can learn nothing about other items. Additionally, users and even the cloud provider observing all accesses cannot infer which user is accessing which data items how often. Another approach based on dynamic accumulators was recently proposed in [19]. Here, no ORAM is employed and thus the cloud (and other users) may learn which data items are accessed, but each access is anonymous and unlinkable to each other. Hence, usage patterns of users can also not be inferred. Independent of the latter approach [18] also proposed a discretionary access control model for data outsourced in the cloud which hides access patterns.

Anonymous payments: Concepts for anonymous and untraceable electronic payments are around for quite a long time [8]. While these first schemes were based on blind signatures and the cut-and-choose paradigm, over the years several improvements, especially for off-line e-cash, e.g. compact e-cash [5] or divisible e-cash [4], allowing to spend 2^n coins from a "single coin" accumulating all coins, have been proposed. Recently, the use of anonymous payments for overlay networks like Tor [2,10], which use lightweight payment protocols for micropayments, have been proposed. Unlike all aforementioned schemes, which assume a bank, a set of payees and a set of payers, in our scenario we have one bank and one payee represented as the same entity. Thus, we do not need to employ offline schemes, but use a kind of online payment scheme. Since we do not need properties of e-cash schemes such as double-spender identification as well as spending with arbitrary payees (this usually adds a non trivial computational overhead), we employ a scheme tailored to payment for cloud resources in our work.

8 Conclusion and Outlook

In this paper we present a privacy preserving cloud resource payment scheme for resource constrained mobile devices such as smartphones. We discuss a concrete scenario for a setting which includes clients, credit resellers and a cloud provider. The client and cloud provider take advantage of the state-of-the-art security enhanced TrustZone and TXT hardware platforms. Besides theoretical considerations, we also prototype the core operations on state-of-the-art platforms. Our results suggest that an optimized C implementation of our scheme is already fast enough for deployment on the current generation of smartphones.

Future work includes the use of an instantiation of an AABCRS based on the strong RSA version of the CL signature scheme [6], which should provide a significant performance boost. Other interesting aspects are the extension of the scenario to multiple cloud providers such that credits can be spend at different CPs (potentially including a "bank" as within traditional payment systems) and the consideration of alternative (more efficient) payment mechanisms as for instance proposed in [10]. The problem of practically realizing such scenarios with smartphone secure processing and secure data storage technologies remain an active area of research.

Acknowledgements

We thank the anonymous reviewers for their helpful feedback on the paper. In particular we thank Thomas S. Benjamin for his many suggestions for improving this paper. This work has been supported by the European Commission through project FP7-SEPIA, grant agreement number 257433. The second author has been supported by an internal grant (zentrale Forschungsförderung – ZFF) of the Carinthia University of Applied Sciences.

References

1. Android x86 Team: Android-x86 - porting android to x86 (2011), http://www.android-x86.org/
2. Androulaki, E., Raykova, M., Srivatsan, S., Stavrou, A., Bellovin, S.M.: PAR: Payment for Anonymous Routing. In: Privacy Enhancing Technologies. LNCS, vol. 5134, pp. 219–236. Springer (2008)
3. ARM Ltd.: TrustZone Technology Overview. http://www.arm.com/products/esd/trustzone_home.html (2011)
4. Au, M.H., Susilo, W., Mu, Y.: Practical Anonymous Divisible E-Cash from Bounded Accumulators. In: Financial Cryptography and Data Security. LNCS, vol. 5143, pp. 287–301. Springer (2008)
5. Camenisch, J., Hohenberger, S., Lysyanskaya, A.: Compact E-Cash. In: EUROCRYPT. LNCS, vol. 3494, pp. 302–321. Springer-Verlag (2005)
6. Camenisch, J., Lysyanskaya, A.: A Signature Scheme with Efficient Protocols. In: SCN. LNCS, vol. 2576, pp. 268–289. Springer (2002)

7. Camenisch, J., Lysyanskaya, A.: Signature Schemes and Anonymous Credentials from Bilinear Maps. In: CRYPTO. LNCS, vol. 3152, pp. 56–72. Springer (2004)
8. Chaum, D.: Blind signatures for untraceable payments. In: CRYPTO. pp. 199–203. Plenum Press (1982)
9. Chen, Y., Paxson, V., Katz, R.H.: What's New About Cloud Computing Security? Tech. Rep. UCB/EECS-2010-5, University of California, Berkeley (2010)
10. Chen, Y., Sion, R., Carbunar, B.: XPay: Practical Anonymous Payments for Tor Routing and other Networked Services. In: WPES. pp. 41–50. ACM (2009)
11. Dierks, T., Rescorla, E.: The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246, IETF (2008), http://tools.ietf.org/html/rfc5246
12. Dietrich, K., Winter, J., Luzhnica, G., Podesser, S.: Implementation Aspects of Anonymous Credential Systems for Mobile Trusted Platforms. In: Communications and Multimedia Security. LNCS, vol. 7025, pp. 45–58. Springer (2011)
13. Dingledine, R., Mathewson, N., Syverson, P.F.: Tor: The Second-Generation Onion Router. In: USENIX Security Symposium. pp. 303–320 (2004)
14. Franz, M., Williams, P., Carbunar, B., Katzenbeisser, S., Peter, A., Sion, R., Sotáková, M.: Oblivious Outsourced Storage with Delegation. In: Financial Cryptography and Data Security. LNCS, vol. 7035, pp. 127–140. Springer (2011)
15. Freescale Semiconductor Inc.: i.MX51 evaluation kit. http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=MCIMX51EVKJ (2010)
16. Grawrock, D.: Dynamics of a Trusted Platform: A Building Block Approach. Intel Press (2009)
17. McCune, J.M., Perrig, A., Reiter, M.K.: Seeing-Is-Believing: Using Camera Phones for Human-Verifiable Authentication. In: IEEE Symposium on Security and Privacy (2005)
18. Raykova, M., Zhao, H., Bellovin, S.: Privacy Enhanced Access Control for Outsourced Data Sharing. In: Financial Cryptography and Data Security. LNCS, Springer (2012)
19. Slamanig, D.: Dynamic Accumulator based Discretionary Access Control for Outsourced Storage with Unlinkable Access. In: Financial Cryptography and Data Security. LNCS, Springer (2012)
20. Slamanig, D.: Efficient Schemes for Anonymous yet Authorized and Bounded Use of Cloud Resources. In: Selected Areas in Cryptography (SAC 2011). LNCS, vol. 7118, pp. 73–91. Springer (2012)
21. Toegl, R., Pirker, M., Gissing, M.: acTvSM: A Dynamic Virtualization Platform for Enforcement of Application Integrity. In: Chen, L., Yung, M. (eds.) Second International Conference on Trusted Systems (INTRUST 2010). LNCS, vol. 6802, pp. 326–345. Springer Verlag (2010)
22. Trusted Computing Group: TCG TPM Specification Version 1.2 (2007), https://www.trustedcomputinggroup.org/developers/
23. Wachsmann, C., Chen, L., Dietrich, K., Löhr, H., Sadeghi, A.R., Winter, J.: Lightweight Anonymous Authentication with TLS and DAA for Embedded Mobile Devices. In: ISC. LNCS, vol. 6531, pp. 84–98. Springer (2010)
24. Wiegele, P., Winter, J., Pirker, M., Toegl, R.: A flexible software development and emulation framework for ARM TrustZone. In: Proceedings of The Third International Conference on Trusted Systems (INTRUST2011). LNCS, Springer (2012)
25. Winter, J., Dietrich, K.: A Hijacker's Guide to the LPC Bus. In: EuroPKI. LNCS, Springer (2011)