AEP-M: Practical Anonymous E-Payment for Mobile Devices using ARM TrustZone and Divisible E-Cash (Full Version)*

Bo Yang[1], Kang Yang[1], Zhenfeng Zhang[1], Yu Qin[1], and Dengguo Feng[1,2]

[1] Trusted Computing and Information Assurance Laboratory
Institute of Software, Chinese Academy of Sciences, Beijing, China
[2] State Key Laboratory of Computer Science, Institute of Software
Chinese Academy of Sciences, Beijing, China
{yangbo,yangkang,zfzhang,qin_yu,feng}@tca.iscas.ac.cn

Abstract. Electronic payment (e-payment) has been widely applied to electronic commerce and has especially attracted a large number of mobile users. However, current solutions often focus on protecting users' money security without concerning the issue of users' privacy leakage. In this paper, we propose AEP-M, a practical anonymous e-payment scheme specifically designed for mobile devices using TrustZone. On account of the limited resources on mobile devices and time constraints of electronic transactions, we construct our scheme based on efficient divisible e-cash system. Precisely, AEP-M allows users to withdraw a large coin of value 2^n at once, and then spend it in several times by dividing it without revealing users' identities to others, including banks and merchants. Users' payments cannot be linked either. AEP-M utilizes bit-decomposition technique and pre-computation to further increase the flexibility and efficiency of spending phase for mobile users. As a consequence, the frequent online spending process just needs at most n exponentiations on elliptic curve on mobile devices. Moreover, we elaborately adapt AEP-M to TrustZone architecture for the sake of protecting users' money and critical data. The methods about key derivation and sensitive data management relying on a root of trust from SRAM Physical Unclonable Function (PUF) are presented. We implement a prototype system and evaluate AEP-M using Barreto-Naehrig (BN) curve with 128-bit security level. The security analysis and experimental results indicate that our scheme could meet the practical requirement of mobile users in respects of security and efficiency.

Keywords: E-Payment, Privacy, Mobile Devices, TrustZone, Divisible E-Cash, PUF.

1 Introduction

Depending on the development and achievements of wireless network as well as modern mobile devices, electronic commerce (e-commerce) is benefiting more and more people's daily lives. As e-commerce becomes a major component of business operations, e-payment, which builds up e-commerce, has become one of the most critical issues for successful business and financial services [16]. Defined as the transfer of an electronic value of payment from a payer to a payee through the Internet, e-payment has been already realized in different ways and applied to mobile devices by intermediaries such as PayPal, Google Wallet, Apple Pay and Alipay [18]. Unfortunately, with the widespread use of mobile e-payment, users are faced with the risk of privacy disclosure.

Although the intermediaries and online banks try the best to enhance the security of their e-payment solutions, the privacy-preserving scheme is often neglected or weakened in the implementation [25]. Generally, authenticating the user's legitimate identity is regarded as one prerequisite for withdrawing digital coins from the banks. The following spending

* An extended abstract of this paper appears in ISC 2016.

procedure is also associated with the authenticated identity, so that all the user's relevant consuming behaviors are identified and linked. In reality, the most of current deployed e-payment solutions unintentionally reveal user personal information, perhaps involving user real identity, billing and shopping records etc., to banks, intermediaries or payees [27,39,29]. Such sensitive information implies one's political view, location, religion or health condition. And what is worse, the personal information could be further shared with some third parties, for example, to send consumers behaviorally targeted advertisements [1]. Statistically, mobile users account for a high proportion among all the e-payment users [26]. Thus, the issue of information leakage is seriously threatening mobile e-payment users' personal privacy.

In theory, constructing anonymous e-payment scheme is able to effectively solve the above problem. Some anonymous protocols are the candidates here including direct anonymous attestation (DAA) [4] and U-Prove [23]. Based on DAA, Yang et al. [41] put forward LAMS for anonymous mobile shopping. However, these protocols hardly fulfill the anonymous e-payment from the perspectives of both anonymity and flexibility for payment. Acting as a targeted component for e-payment, electronic cash (e-cash), introduced by Chaum [10], allows users to withdraw digital coins from a bank and to spend them to merchants in an anonymous way, thus perfectly emulating conventional cash transactions. Derived from e-cash, divisible e-cash systems are proposed to address the issue of splitting coins of large values. Depending on it, users could withdraw a large coin of value 2^n at once and spend it in several times by dividing it. In practice, divisible e-cash makes the cash transactions more efficient and flexible. In regard to mobile devices, the limited resources along with the strong time constraints of electronic transactions indeed require the practical withdrawal and spending procedures. Therefore, it is advisable to build anonymous e-payment scheme upon efficient divisible e-cash for mobile devices.

It is commonly believed that good security and trust will ultimately increase the use of e-payment. Nevertheless, the direct application of anonymous e-payment scheme on mobile devices would bring potential security risks. Without the dedicated protection, the scheme's executing codes and sensitive data are easily either compromised or stolen by the malwares. In some cases, the attacks on mobile e-payment could cause user's great loss of property. The technique of Trusted Execution Environment (TEE) on mobile devices is able to lend us a helping hand. Isolated from a Rich Execution Environment (REE) where the Guest OS runs, TEE aims to protect sensitive codes execution and assets. As a prevalent example of providing TEE for embedded devices, ARM TrustZone [2] has been used to execute security-critical services [38,37]. Actually, TrustZone enables a single physical processor to execute codes in one of two possible isolated operating worlds: the *normal world* (NW) for REE and the *secure world* (SW) for TEE. The two worlds have independent memory address spaces and different privileges. As a hardware-based security extension of ARM architecture, TrustZone is widely supported and applied by mobile devices. But there is a fly in the ointment that TrustZone does not definitely provide the root of trust with inside root key for sensitive data management. To the best of our knowledge, there is no anonymous e-payment scheme specially designed for mobile devices using TrustZone.

1.1 Our Contribution

Based on ARM TrustZone and the divisible e-cash scheme with the best efficiency by Canard et al. [8], we propose AEP-M, a practical anonymous e-payment scheme for mobile devices, which enables a user to spend his digital coins securely and efficiently while preserving his privacy. This is the first complete work to design an efficient anonymous e-payment scheme integrated with TrustZone. We substantially modify the original e-cash scheme for adapting it to the executing mode of TrustZone and guaranteeing its security on mobile devices.

For device-centered design, we make following steps towards practical and secure usage:

– the sensitive codes on the user side of AEP-M are isolated and executed in TEE provided by TrustZone for the possibility that the guest OS is compromised;

– AEP-M utilizes some secret keys, which are derived from a root key seed reproduced via an on-chip SRAM PUF [13], to protect users' coins and data;
– in AEP-M, online banks could authenticate a user who holds a mobile device with available TrustZone and a valid account-password pair.

AEP-M elaborately protects the security of the user's passwords and coins even if the NW of his mobile device is corrupted while the SW still keeps honest. The pre-computation stage is carefully added into our scheme such that the computation amounts of the frequent online spending phase for mobile users are decreased. Furthermore, while the original divisible e-cash scheme [8] only allows one to spend a coin of value 2^ℓ for some $0 \leq \ell \leq n$ at once, our scheme supports that one spends a coin of value v for any $1 \leq v \leq 2^n$ at once by using the bit-decomposition technique, where the maximum denomination of a coin is 2^n.

We implement a prototype system of AEP-M and evaluate its efficiency using BN curve at the security level of 128-bit. The experimental results show that our scheme is efficient enough for practical usage, even from the perspective of mobile devices.

1.2 Related Work

E-Payment Scheme. Based on pre-paid cards, credit cards, debit cards and electronic checks, most current e-payment schemes attempt to ensure the user's data and money security without addressing the privacy protection [16,24]. Different from those schemes, e-cash system does a better job to construct anonymous e-payment. After Chaum first introduced e-cash [10], fair e-cash [24] was proposed to detect double-spending and identify the defrauders, which however weakened the anonymity of the scheme. Camenisch et al. [5] presented the compact e-cash system allowing users to withdraw wallets with 2^n coins at once. Unfortunately, its spending procedure should be done coin by coin. To deal with the problem of splitting coins of large values, some divisible e-cash schemes [21,9] were given without achieving high levels of anonymity. Afterwards, some truly anonymous divisible e-cash systems [6,7,14] were described. Nevertheless, they were quite inefficiency to implement, especially for resource-constrained devices. Recently, Canard et al. [8] proposed the first really efficient divisible e-cash system by defining one global binary tree that is common to all the coins. Our scheme takes this system as a reference and further increases its efficiency and security according to our architecture of trusted mobile device.

TrustZone Technology. As introduced before, e-payment needs to guarantee its codes integrity and data security. ARM TrustZone technology for the mobile devices can offer much help, which supports flexibly developing specified secure system. Relying on TrustZone, many practical mobile schemes are proposed. In [32], the public transport ticketing system on smartphone was designed exactly utilizing TrustZone. Also based on it, Yang et al. [41] presented an anonymous mobile shopping scheme. AdAttester [17] was described specially for secure mobile advertisement likewise on a TrustZone-enabled device. To date, TrustZone has been popularized and applied by many mainstream mobile manufacturers, such as Apple, Samsung and Huawei, to achieve secure applications [17,15]. However, no secure e-payment solution has ever been designed for mobile devices and particularly adapted to TrustZone.

2 Preliminaries

2.1 Notation

Throughout the paper, we use the notation shown in Table 1.

2.2 Bilinear Groups

Bilinear groups consist of three (multiplicatively written) groups $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ of prime order p equipped with a bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. Let g and $\widetilde{g}$ be generators of

Table 1. Notation used in this paper

Notation	Descriptions
λ	security parameter
$x \xleftarrow{\$} \mathbb{S}$	x chosen uniformly at random from a set $\mathbb{S}$
$y := x$	y assigned as x
$x \| \| y$	concatenation of x and y
$(y_1, ..., y_j) \leftarrow \mathsf{A}(x_1, ..., x_i)$	a (randomized) algorithm with input $(x_1, ..., x_i)$ and output $(y_1, ..., y_j)$
$1_{\mathbb{G}}$	the identity element of a group $\mathbb{G}$
$\mathbb{G}^*$	$\mathbb{G} \setminus \{1_{\mathbb{G}}\}$ for a group $\mathbb{G}$
$\Sigma_1 = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$	digital signature algorithm
$\Sigma_2 = (\mathsf{MAC})$	message authentication code
$\Sigma_3 = (\mathsf{Enc}_{\mathsf{asym}}, \mathsf{Dec}_{\mathsf{asym}})$	asymmetric (public key) encryption and decryption algorithm
$\Sigma_4 = (\mathsf{Enc}_{\mathsf{sym}}, \mathsf{Dec}_{\mathsf{sym}})$	symmetric encryption and decryption algorithm

$\mathbb{G}_1$ and $\mathbb{G}_2$ respectively. We define $\Lambda = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g, \widetilde{g})$ to be a description of bilinear groups parameters. The map e must satisfy the following properties:

1. **Bilinear.** for any $u \in \mathbb{G}_1$, $v \in \mathbb{G}_2$ and any $a, b \in \mathbb{Z}_p$, $e(u^a, v^b) = e(u, v)^{ab}$.
2. **Non-degenerate.** $e(g, \widetilde{g}) \neq 1_{\mathbb{G}_T}$.
3. **Computable.** the map e is efficiently computable.

In this paper, we only consider the Type-3 pairings [35], thus $\mathbb{G}_1 \neq \mathbb{G}_2$ and there is no known efficiently computable isomorphism between $\mathbb{G}_1$ and $\mathbb{G}_2$.

2.3 Cryptographic Assumptions

The security of AEP-M is based on the Decisional Diffie-Hellman assumption in $\mathbb{G}_1$ ($\mathrm{DDH}_{\mathbb{G}_1}$), Weak-EXDH assumption [8] and blind 4-LRSW (B-4-LRSW) assumption [3]. These three assumptions are stated as follows:

Assumption 1 ($\mathbf{DDH}_{\mathbb{G}_1}$). Given (g, g^a, g^b) for $a, b \xleftarrow{\$} \mathbb{Z}_p$, it is hard to distinguish g^{ab} from a random element of $\mathbb{G}_1$.

Assumption 2 (Weak-EXDH). Given the bilinear group parameters Λ, (g^a, g^b, g^{bc}) and $(\widetilde{g}^b, \widetilde{g}^c)$ for $a, b, c \xleftarrow{\$} \mathbb{Z}_p$, it is hard to distinguish g^{abc} from a random element of $\mathbb{G}_1$.

Assumption 3 (B-4-LRSW). Given the bilinear group parameters Λ, $(\widetilde{g}^x, \widetilde{g}^y)$ for $x, y \xleftarrow{\$} \mathbb{Z}_p$ and an oracle that on input of $g^m \in \mathbb{G}_1$ outputs $(A, A^y, A^{x+mxy}, A^{my})$ for $A \xleftarrow{\$} \mathbb{G}_1^*$, it is hard to output $(m^*, A^*, B^*, C^*, D^*)$ such that $m^* \in \mathbb{Z}_p^*$, $A^* \in \mathbb{G}_1^*$, $B^* := (A^*)^y$, $C^* := (A^*)^{x+m^*xy}$ and $D^* := (A^*)^{m^*y}$ where g^{m^*} was never queried to the oracle.

2.4 ARM TrustZone

ARM TrustZone [2] is a hardware-based security extension technology incorporated into ARM processors. The whole system is separated into two worlds and each world has banked registers and memory to run the domain-dedicated OS and software. The isolation mechanisms of TrustZone are well defined. As a result, access permissions are strictly under the control of the *secure world* that the *normal world* components cannot access the *secure world* resources. As the processor only runs in one world at a time, to run in the other world requires context switch. A secure monitor mode exists in the *secure world* to control the switch and migration between the two worlds. The time overhead of the switch is small enough to be ignored. Also, TrustZone leverages almost full power of the processor to run operations in one world at one time, which provides good performance for computing processes.

2.5 Physical Unclonable Functions

Physical Unclonable Functions (PUFs) [30] are functions where the relationship between input (or challenge) and output (or response) is decided by a physical system. Randomness and unclonability are two significant properties of PUFs. The unclonability originates from random variations in a device's manufacturing process. With the help of a fuzzy extractor that eliminates the noise from the response, PUFs are able to implicitly "store" a piece of secret data. PUFs provide much higher physical security by extracting the secret data from complex physical systems rather than directly reading them from non-volatile memory. Additionally, PUFs are cost-effective, since they take the advantage of the results from a preexisting manufacturing process.

Strictly speaking, TrustZone just provides an isolated environment. Only equipped with a root of trust, it becomes a real "trusted" execution environment (TEE) [43]. Because TrustZone almost does not internally install an available root key, it loses the capability to offer a root of trust. To cover this shortage, a PUF can be employed to properly act as the root of trust. In this paper, AEP-M takes the secret data extracted from the PUF as a root key seed to generate other keys. We adopt SRAM PUF [13] that leverages the relationship between an SRAM cell's address for the challenge and its power up value for the response.

3 System Model and Assumptions

3.1 System Model

The system model of AEP-M is composed of five kinds of entities: mobile device $\mathcal{D}$, merchant $\mathcal{M}$, trusted authority $\mathcal{T}$, central bank $\mathcal{B}$ and traditional commercial bank. In practice, there could be a number of mobile devices and merchants participating in our system. For the sake of brevity and clarity, we use $\mathcal{D}$ and $\mathcal{M}$ to represent an individual instantiation respectively. $\mathcal{D}$ is directly accessed by a user and equipped with ARM processor having TrustZone extension technology. $\mathcal{B}$ is responsible for issuing digital coins to legitimate (or trusted) $\mathcal{D}$ through **Withdraw** phase. $\mathcal{B}$ could be a bank card organization supporting e-payment or an intermediary serving electronic transactions. In the background, several commercial banks, where users actually deposit money, are in cooperation with $\mathcal{B}$ for dealing with money transfers in the real world. Service or product providers play the role of $\mathcal{M}$ in this interactive model. They collect digital coins from $\mathcal{D}$ via **Spend** phase and redeem them from $\mathcal{B}$ via **Deposit** phase. Note that $\mathcal{M}$ verifies the digital coins of some user without revealing user's identity to any entities including $\mathcal{M}$ itself. In various scenarios, $\mathcal{D}$ is able to pay $\mathcal{M}$ the coins via either Internet or other wireless technologies (e.g., NFC). Managed by the government or the industry administration, in **Identify** phase $\mathcal{T}$ performs revealing identity of the users who attempt to double-spend digital coins. From $\mathcal{B}$, $\mathcal{T}$ receives the reports and proofs about the double-spending. Figure 1 illustrates the system model for our proposed scheme.

3.2 Assumptions and Threat Model

To simplify our design in the system model, we assume that data communications between $\mathcal{B}$ and traditional bank, and between $\mathcal{B}$ and $\mathcal{T}$ build on secure transport protocols, such as TLS, which can provide confidentiality, authenticity and integrity protection for data transmission. Also, $\mathcal{M}$, $\mathcal{D}$ and $\mathcal{B}$ are able to acquire public parameters from $\mathcal{T}$ in the correct way. Public Key Infrastructure (PKI) is supposed to be already realized for authenticating $\mathcal{B}$ and $\mathcal{M}$. As a consequence, (1) $\mathcal{D}$ and $\mathcal{M}$ can accurately obtain the public key of $\mathcal{B}$ by verifying its certificate; (2) $\mathcal{D}$ and $\mathcal{B}$ can accurately obtain the public key of $\mathcal{M}$ similarly.

Actually, the establishment of the whole system requires some premised trust relationships. First, $\mathcal{B}$ is trusted not to issue counterfeit digital coins. Second, the manufacturers are considered to be so credible that they only embed device certificates into eligible $\mathcal{D}$s which

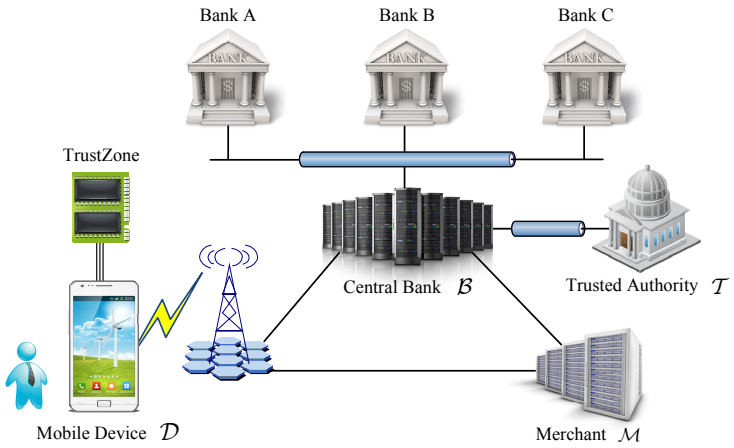

Fig. 1. System Model of AEP-M.

own available TrustZone. Another accepted fact is that $\mathcal{T}$ would never conspire maliciously with any other entities. Constrained by the market supervision and the force of law, the above-mentioned trust relationships are easily established and maintained.

Based on the assumptions, AEP-M protects against the following adversary:

- The adversary can attack the scheme itself by attempting to pretend entities, manipulate data transmission between entities and forge data.
- The adversary can perform software-based attacks which compromise the mobile Rich OS or existing applications running in REE. AEP-M interfaces in REE are also available for the adversary.
- The adversary can physically access the mobile device. He can reboot the device and gain access to data residing on persistent storage.

However, we ignore the malicious behaviors of tampering with the TrustZone hardware or mounting side-channel attacks on PUF [22].

4 AEP-M Scheme for Mobile Devices

In this section, we provide the specific architecture of trusted mobile device, and then present the key derivation and sensitive data management. Depending on these, the construction of AEP-M scheme is detailed next. Finally, the security properties of AEP-M is analyzed.

4.1 The Architecture of Trusted Mobile Device

Leveraging TrustZone and PUF technology, we design the architecture of trusted mobile device specifically for AEP-M based on our previous work [42]. The software-based implementation of AEP-M functionality on existing hardwares targets at economy, flexibility and extensibility. Meanwhile, our architecture is designed to be compatible with the conventional running model of secure applications using TrustZone. Figure 2 shows the detailed architecture with the way components interact with each other.

AEP-M functionality in the architecture contains two components: untrusted AEP-M Proxy in *normal world* (NW) and security-sensitive AEP-M Service in *secure world* (SW). In reality, SW instantiates TEE, while NW implements REE. Depending on the whitelist and integrity protection mechanism [28], only the trusted codes of programs in SW could be loaded and executed. Thus, AEP-M Service resides in a relatively secure environment isolated from other codes running in NW. The components are formally described as follows.

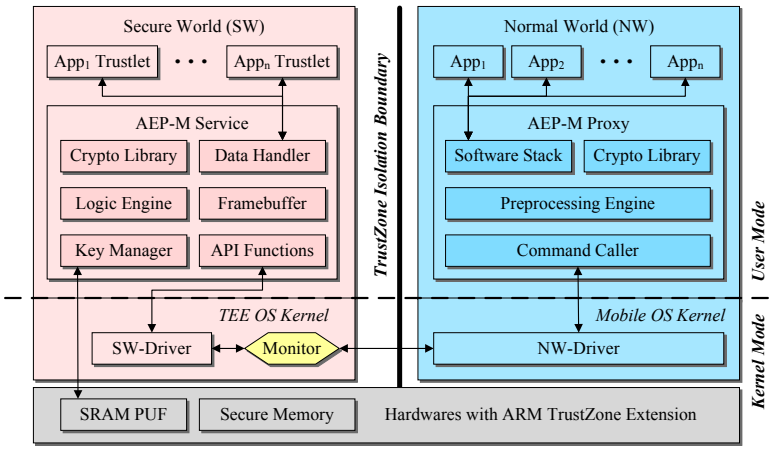

Fig. 2. Architecture of Trusted Mobile Device for AEP-M.

AEP-M Proxy. This is the component visible for mobile (e-payment) applications in NW. Waiting for their AEP-M service requests, the proxy handles the parameters and preprocesses them. According to the request type, the proxy would call AEP-M Service for substantive computations of the scheme and finally return the results. AEP-M Proxy consists of the following four subcomponents:

- **Software Stack**: provides top AEP-M interfaces for mobile applications. It parses the service requests and gives service response results.
- **Crypto Library**: offers cryptographic algorithm support for Preprocessing Engine. In NW, this library only supports exponentiations on elliptic curves.
- **Preprocessing Engine**: executes pre-computation for AEP-M after digital coins are successfully withdrawn from central bank to the mobile device.
- **Command Caller**: formats calling command and interacts with AEP-M Service. It sends the command through the GP TEE Client API [33], requests to switch NW to SW via NW-Driver and waits for the returned values.

AEP-M Service. This is the core component to perform AEP-M critical computations and operations. The execution of the component codes is under the well protection of TrustZone isolation mechanism. Six following subcomponents constitute AEP-M Service component:

- **API Functions**: receives a service request from AEP-M Proxy and parses the command. The functions transmit instructions to Logic Engine and wait for results that would be forwarded back to AEP-M Proxy.
- **Key Manager**: creates cryptographic keys using the unique root key seed extracted from SRAM PUF and provides keys to Data Handler.
- **Data Handler**: receives message (e.g., the amount of digital coins to spend) and seals or unseals sensitive data. To prevent adversary from forging message, Data Handler only receives message produced by App Trustlet in SW. Besides, using keys from Key Manager, Data Handler seals sensitive data to store them in the insecure non-volatile storage space of mobile device.
- **Framebuffer**: stores the image of confirmation message (e.g., the identity of merchant to be paid) to be securely displayed for the user. Different from the general frame buffer in NW, Framebuffer is devoted to the reliable graphical user interface (GUI) for SW.
- **Crypto Library**: offers cryptographic algorithms support for Logic Engine and Data Handler. In SW, it supports bilinear maps, computations on elliptic curves, and other cryptographic operations.
- **Logic Engine**: executes the computations of security-sensitive parts of AEP-M scheme. Engine reads parameters and data to run operations relying on scheme specification.

Application and Application Trustlet. The corresponding application should be launched if the user wants to enjoy e-payment service. For upper-level interaction, the application released by $\mathcal{B}$ consists of two parts: App for NW and App Trustlet for SW. App provides the general GUI and basic functions, while App Trustlet is securely loaded by SW and trusted for processing security-sensitive user inputs and data operations. When App has the need to execute AEP-M procedures, it calls AEP-M Proxy using its Software Stack. App could notify App Trustlet in SW to execute some sensitive operations through inter-domain communication mechanism supported by TrustZone [15].

Components in Kernels. SW-Driver in TEE OS Kernel and NW-Driver in Mobile OS Kernel handle the communication requests and responses with respect to switching the worlds. As secure monitor, the Monitor controls hardwares to fulfill the switching action.

Components in Hardwares. The hardwares of mobile device support ARM TrustZone extension technology. Protected by TrustZone mechanism, SRAM PUF component and Secure Memory component are only accessible for SW. Secure Memory contributes to temporally saving sensitive data.

4.2 Key Derivation and Sensitive Data Management

Prior to describing the concrete construction of our AEP-M scheme, we show how to derive various keys for different purposes using the root key seed extracted from SRAM PUF and how to utilize the derived keys to protect sensitive data.

Root Key Seed Extraction. We use the technique of SRAM PUF in [43] to extract the secret root key *seed*, which is a unique bit string picked randomly by the OEM who "stores" it in $\mathcal{D}$ through the physical features of one SRAM inside $\mathcal{D}$. From SRAM PUF component, *seed* is only reproduced and securely cached by Key Manager when $\mathcal{D}$ starts up every time in normal use. The confidentiality of *seed* is rigidly guaranteed by TrustZone.

Key Derivation. Key Manager has the deterministic key derivation function KDF: $\widetilde{\mathcal{S}} \times \{0,1\}^* \to \widetilde{\mathcal{K}}$, where $\widetilde{\mathcal{S}}$ is the key seed space, and $\widetilde{\mathcal{K}}$ is the derived key space. Using the KDF, the device key pair and the storage root key is derived as $(\mathsf{dsk}, \mathsf{dpk}) \leftarrow \mathsf{KDF}_{seed}(\texttt{"identity"})$ and $\mathsf{srk} \leftarrow \mathsf{KDF}_{seed}(\texttt{"storage\_root"})$ respectively. The unique device key pair is analogous to the endorsement key defined in trusted computing [36] but supports encryption and decryption. The storage root key srk is used for generating specific storage keys to preserve sensitive data. The hierarchical structure of storage keys enhances the security for key usage. Note that all the derived keys are never stored permanently. Instead, they are regained via KDF with *seed* at the same way when needed.

Sensitive Data Management. We can utilize the storage keys derived from the storage root key srk to seal the AEP-M's public parameters *params*, $\mathcal{D}$'s digital coin σ, the secret key m, and other related variables CT and δ. What these variables represent will be explained in Section 4.3. The sealed results of these data are stored in the insecure positions of $\mathcal{D}$.

- Protect integrity for *params*: $\mathsf{mk}_{params} \leftarrow \mathsf{KDF}_{\mathsf{srk}}(\texttt{"storage\_key"}||\texttt{"MAC"}||params)$, and $blob_{params} \leftarrow \mathsf{Data\_Seal}(\texttt{"MAC"}, \mathsf{mk}_{params}, params)$, where
$$blob_{params} := params||\,\mathsf{MAC}(\mathsf{mk}_{params}, params).$$
- Protect integrity for σ: $\mathsf{mk}_{\sigma} \leftarrow \mathsf{KDF}_{\mathsf{srk}}(\texttt{"storage\_key"}||\texttt{"MAC"}||\sigma)$, and $blob_{\sigma} \leftarrow \mathsf{Data\_Seal}(\texttt{"MAC"}, \mathsf{mk}_{\sigma}, \sigma)$, where
$$blob_{\sigma} := \sigma||\,\mathsf{MAC}(\mathsf{mk}_{\sigma}, \sigma).$$
- Protect both confidentiality and integrity for m, CT and δ with the aid of U:[3] $(\mathsf{sk}_m, \mathsf{mk}_m) \leftarrow \mathsf{KDF}_{\mathsf{srk}}(\texttt{"storage\_key"}||\texttt{"Enc+MAC"}||U)$, and

[3] In AEP-M scheme, $U = g^m$ for some fixed basis g.

$blob_m \leftarrow \mathsf{Data\_Seal}(\texttt{"Enc+MAC"}, \mathsf{sk}_m, \mathsf{mk}_m, m||CT||\delta, U)$, where

$$blob_m := \mathsf{Enc_{sym}}(\mathsf{sk}_m, m||CT||\delta)||U||\,\mathsf{MAC}(\mathsf{mk}_m, \mathsf{Enc_{sym}}(\mathsf{sk}_m, m||CT||\delta)||U).$$

Data Handler can use $\mathsf{Data\_Unseal}()$ to recover and verify the sensitive data from blobs with the related keys regained by Key Manager.

4.3 The Details of AEP-M Scheme

Following the divisible e-cash scheme [8], a unique and public global tree of depth n is used for all coins of value $V = 2^n$ as illustrated in Figure 3. So each leaf denotes the smallest unit of value to spend. We define $\mathcal{S}_n$ as the set of bit strings of size smaller than or equal to n and $\mathcal{F}_n$ as the set of bit strings of size exactly n. Thus, each node of the tree refers to an element $s \in \mathcal{S}_n$, the root to the empty string ϕ, and each leaf to an element $f \in \mathcal{F}_n$. For any node $s \in \mathcal{S}_n$, $\mathcal{F}_n(s) = \{f \in \mathcal{F}_n | s \text{ is a prefix of } f\}$ contains all the leaves in the subtree below s.

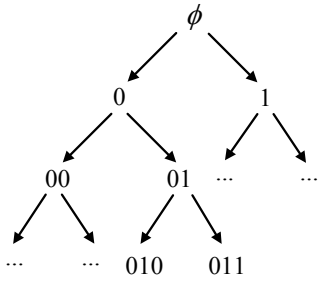

Fig. 3. Public Global Tree for All Coins.

Assume, before leaving the factory, $\mathcal{D}$ is initialized by the OEM in SW to generate the unique device key $(\mathsf{dsk}, \mathsf{dpk})$ which could uniquely identify $\mathcal{D}$. Then, the OEM issues a certificate $\mathsf{cert}_\mathcal{D}$ w.r.t. the public key dpk to indicate the OEM's recognition for $\mathcal{D}$. The certificate $\mathsf{cert}_\mathcal{D}$ also contains some $\mathcal{D}$'s configuration information (e.g., whether TrustZone is available).

AEP-M scheme consists of six phases: **Setup**, **KeyGen**, **Withdraw**, **Spend**, **Deposit** and **Identify**. First of all, **Setup** is executed to create the public parameters by $\mathcal{T}$. After that, $\mathcal{B}$ and $\mathcal{M}$ can execute **KeyGen** to generate their public-private key pairs according to the public parameters. Then, other phases are enabled to be executed according to requirements. The phases of the scheme are presented in detail as follows.

Setup. In this phase, the trusted authority $\mathcal{T}$ creates the public parameters. Given a security parameter λ, $\mathcal{T}$ picks the suitable bilinear groups parameters $\Lambda := (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g, \tilde{g})$ described in Section 2.2 such that $|p| \geq 2\lambda$. And then, according to the global tree, $\mathcal{T}$ generates (1) $r_s \xleftarrow{\$} \mathbb{Z}_p$ and $g_s := g^{r_s}$ for each $s \in \mathcal{S}_n$, and (2) $l_f \xleftarrow{\$} \mathbb{Z}_p$ and $\tilde{g}_{s \mapsto f} := \tilde{g}^{l_f / r_s}$ for each $s \in \mathcal{S}_n$ and each $f \in \mathcal{F}_n(s)$. $\mathcal{T}$ keeps $\mathbf{sck} = \{r_s | s \in \mathcal{S}_n\}$ as its secret keys to be used in **Identify** phase. Also, $\mathcal{T}$ determines a series of algorithms Ψ including the algorithms covering from Σ_1 to Σ_4 in Table 1, and four independent collision-resistant hash functions:

$$\mathsf{H}_1 : \{0,1\}^* \to \mathbb{Z}_p,\ \mathsf{H}_2 : \{0,1\}^* \to \mathbb{Z}_p,\ \mathsf{H}_3 : \{0,1\}^* \to \{0,1\}^{2\lambda},\ \mathsf{H}_4 : \{0,1\}^* \to \{0,1\}^{2\lambda}.$$

Finally, $\mathcal{T}$ sets $(\Lambda, n, \Psi, \{r_s | s \in \mathcal{S}_n\}, \{\tilde{g}_{s \mapsto f} | s \in \mathcal{S}_n \wedge f \in \mathcal{F}_n(s)\})$ as the public parameters, where $\mathcal{D}$ and $\mathcal{M}$ only need to know $params := (\Lambda, n, \Psi, \{r_s | s \in \mathcal{S}_n\})$, while $\mathcal{B}$ requires $params' := (\Lambda, n, \Psi, \{r_s | s \in \mathcal{S}_n\}, \{\tilde{g}_{s \mapsto f} | s \in \mathcal{S}_n \wedge f \in \mathcal{F}_n(s)\})$. After obtaining $params$, $\mathcal{D}$ calls $\mathsf{Data\_Seal}()$ to seal it and stores the output $blob_{params}$.

KeyGen. This phase initializes the public-private key pair for the central bank $\mathcal{B}$ and a merchant $\mathcal{M}$.

- *Key Generation for Central Bank.* First, given $params'$ as input, $\mathcal{B}$ picks $x, y \xleftarrow{\$} \mathbb{Z}_p^*$, and computes $X := \tilde{g}^x$ and $Y := \tilde{g}^y$. $\mathcal{B}$ sets (x, y) as the private key $\overline{\mathsf{sk}_\mathcal{B}}$ and publishes (X, Y) as the public key $\overline{\mathsf{pk}_\mathcal{B}}$. Second, $\mathcal{B}$ uses $\mathsf{KeyGen}()$ in Σ_1 to generate key pair for establishing sessions with $\mathcal{D}$: $(\mathsf{sk}_\mathcal{B}, \mathsf{pk}_\mathcal{B}) \leftarrow \mathsf{KeyGen}(1^\lambda)$, where $\mathsf{sk}_\mathcal{B}$ is the private key.
- *Key Generation for Merchant.* Similarly, $\mathcal{M}$ uses $\mathsf{KeyGen}()$ to generate key pair for establishing sessions with $\mathcal{D}$: $(\mathsf{sk}_\mathcal{M}, \mathsf{pk}_\mathcal{M}) \leftarrow \mathsf{KeyGen}(1^\lambda)$.

Accordingly, $\mathcal{D}$ could get the correct $\mathsf{pk}_{\mathcal{B}}$ and $\mathsf{pk}_{\mathcal{M}}$ from $\mathcal{B}$ and $\mathcal{M}$ via verifying their certificates [4]. And likewise, $\mathcal{M}$ and $\mathcal{B}$ could acquire the correct $\overline{\mathsf{pk}}_{\mathcal{B}}$ and $\mathsf{pk}_{\mathcal{M}}$ respectively as well as $\mathcal{T}$ obtains $\overline{\mathsf{pk}}_{\mathcal{B}}$.

Withdraw. In this phase, a user with mobile device $\mathcal{D}$ could withdraw some digital coins from the central bank $\mathcal{B}$ as follows.

1. The user operates App in NW of $\mathcal{D}$ to prepare for withdrawing some digital coins. $\mathcal{D}$ switches into SW and chooses a nonce $n_{\mathcal{D}} \xleftarrow{\$} \{0,1\}^{\lambda}$. $n_{\mathcal{D}}$ is saved in Secure Memory and delivered to AEP-M Proxy that sends $n_{\mathcal{D}}$, $\mathcal{D}$'s dpk with its certificate $\mathsf{cert}_{\mathcal{D}}$ to $\mathcal{B}$.
2. $\mathcal{B}$ checks whether dpk is valid with $\mathsf{cert}_{\mathcal{D}}$ and checks the configuration information on $\mathsf{cert}_{\mathcal{D}}$. If the check is passed, $\mathcal{B}$ chooses a nonce $n_{\mathcal{B}} \xleftarrow{\$} \{0,1\}^{\lambda}$, a key $\mathsf{k}_{\mathsf{mac}} \xleftarrow{\$} \{0,1\}^{\lambda}$ for MAC and a key $\mathsf{k}_{\mathsf{enc}} \xleftarrow{\$} \{0,1\}^{\lambda}$ for $\mathsf{Enc}_{\mathsf{sym}}$ and $\mathsf{Dec}_{\mathsf{sym}}$. Then, $\mathcal{B}$ encrypts $n_{\mathcal{B}}$, $\mathsf{k}_{\mathsf{mac}}$ and $\mathsf{k}_{\mathsf{enc}}$ using dpk to get a cipher text $C_{\mathcal{B}} \leftarrow \mathsf{Enc}_{\mathsf{asym}}(\mathsf{dpk}, n_{\mathcal{B}}||\mathsf{k}_{\mathsf{mac}}||\mathsf{k}_{\mathsf{enc}})$ and signs dpk, $n_{\mathcal{D}}$ and $C_{\mathcal{B}}$ using $\mathsf{sk}_{\mathcal{B}}$ to output a signature $\alpha \leftarrow \mathsf{Sign}(\mathsf{sk}_{\mathcal{B}}, \mathsf{dpk}||n_{\mathcal{D}}||C_{\mathcal{B}})$. Finally, $\mathcal{B}$ sends a commitment request $comm_{\mathsf{req}} := (C_{\mathcal{B}}, \alpha)$ to $\mathcal{D}$.
3. AEP-M Proxy invokes AEP-M Service with input $comm_{\mathsf{req}}$. In SW, App Trustlet waits for the user to input his bank account $account_{\mathcal{D}}$, the corresponding password pwd and the amount of digital coins to withdraw. For simplicity, we only describe how to withdraw one coin. The **Withdraw** phase could be easily extended to support withdrawing multiple coins at once. After the user finishes inputting, Logic Engine calls the API $\mathsf{AEPM\_SW\_Withdraw}()$ to generate a commitment response:

   $$comm_{\mathsf{res}} \leftarrow \mathsf{AEPM\_SW\_Withdraw}(blob_{params}, n_{\mathcal{D}}, \mathsf{pk}_{\mathcal{B}}, comm_{\mathsf{req}}, account_{\mathcal{D}}, pwd),$$

   where the API is executed as follows:

   1) Unseal the blob $blob_{params}$ to get $params$ by calling $\mathsf{Data\_Unseal}()$.
   2) Verify α using $\mathsf{pk}_{\mathcal{B}}$: $res \leftarrow \mathsf{Verify}(\mathsf{pk}_{\mathcal{B}}, \mathsf{dpk}||n_{\mathcal{D}}||C_{\mathcal{B}}, \alpha)$. If $res = false$, $comm_{\mathsf{res}} := \perp$ and return.
   3) Decrypt $C_{\mathcal{B}}$ using dsk: $(n'_{\mathcal{B}}, \mathsf{k}_{\mathsf{mac}}, \mathsf{k}_{\mathsf{enc}}) \leftarrow \mathsf{Dec}_{\mathsf{asym}}(\mathsf{dsk}, C_{\mathcal{B}})$.
   4) Choose $m \xleftarrow{\$} \mathbb{Z}_p^*$ as the secret key for a coin, and compute the commitment $U := g^m$.
   5) Set $\delta := V$ where δ denotes the current balance of the coin.
   6) Set CT as a string of $2^{n+1} - 1$ bits where each bit is 1. CT denotes the current tree structure of the unspent coin.
   7) Call $\mathsf{Data\_Seal}()$ to seal m, CT and δ, and generate $blob_m$(see Section 4.2).
   8) Choose a random number $r_{\mathcal{D}} \xleftarrow{\$} \mathbb{Z}_p^*$ and compute $R_{\mathcal{D}} := g^{r_{\mathcal{D}}}$.
   9) Compute $c_{\mathcal{D}} := \mathsf{H}_1(g||U||R_{\mathcal{D}}||C_{\mathcal{B}}||\alpha||n'_{\mathcal{B}})$.
   10) Compute $s_{\mathcal{D}} := r_{\mathcal{D}} + c_{\mathcal{D}} \cdot m \ (mod\ p)$.
   11) Generate a cipher context $C_{\mathcal{D}} \leftarrow \mathsf{Enc}_{\mathsf{sym}}(\mathsf{k}_{\mathsf{enc}}, account_{\mathcal{D}}||pwd)$.
   12) Generate $\tau_{\mathcal{D}} \leftarrow \mathsf{MAC}(\mathsf{k}_{\mathsf{mac}}, U||n'_{\mathcal{B}}||c_{\mathcal{D}}||s_{\mathcal{D}}||C_{\mathcal{D}})$, and output $comm_{\mathsf{res}} := (\tau_{\mathcal{D}}, U, n'_{\mathcal{B}}, c_{\mathcal{D}}, s_{\mathcal{D}}, C_{\mathcal{D}})$.

   AEP-M Service saves $n'_{\mathcal{B}}$ and $\mathsf{k}_{\mathsf{mac}}$ in Secure Memory as well as stores $blob_m$ in non-volatile storage. Then $\mathcal{D}$ switches back to NW and sends $comm_{\mathsf{res}}$ to $\mathcal{B}$.
4. On input $comm_{\mathsf{res}}$, $\mathcal{B}$ runs the following algorithm to generate a digital coin σ on m for $\mathcal{D}$:

   $$(\sigma, \tau_{\mathcal{B}}) \leftarrow \mathsf{Gen\_DC}(comm_{\mathsf{res}}, params', \mathsf{k}_{\mathsf{mac}}, \mathsf{k}_{\mathsf{enc}}, n_{\mathcal{B}}, \overline{sk}_{\mathcal{B}}).$$

   The algorithm has seven steps:

   1) Verify $\tau_{\mathcal{D}} = \mathsf{MAC}(\mathsf{k}_{\mathsf{mac}}, U||n'_{\mathcal{B}}||c_{\mathcal{D}}||s_{\mathcal{D}}||C_{\mathcal{D}})$, and check whether $n'_{\mathcal{B}} = n_{\mathcal{B}}$.
   2) Check whether U has not been used before by querying the database.
   3) Compute $R'_{\mathcal{D}} := g^{s_{\mathcal{D}}} \cdot U^{-c_{\mathcal{D}}}$ and $c'_{\mathcal{D}} := \mathsf{H}_1(g||U||R'_{\mathcal{D}}||C_{\mathcal{B}}||\alpha||n_{\mathcal{B}})$.
   4) Check whether $c'_{\mathcal{D}} = c_{\mathcal{D}}$.

[4] Utilizing PKI solution, a Certificate Authority (CA) issues public key certificates for the keys to $\mathcal{B}$ and $\mathcal{M}$ respectively.

5) Decrypt $C_{\mathcal{D}}$ using $\mathsf{Dec_{sym}}$ and $\mathsf{k_{enc}}$: $account_{\mathcal{D}}||pwd \leftarrow \mathsf{Dec_{sym}}(\mathsf{k_{enc}}, C_{\mathcal{D}})$, then check the plaintext's validness via communicating with the related commercial bank. If the account balance is enough, deduct money from the account and temporarily save it in $\mathcal{B}$.
6) Choose a random number $a \xleftarrow{\$} \mathbb{Z}_p^*$, compute $A := g^a$, $B := A^y$, $C := g^{ax} \cdot U^{axy}$ and $D := U^{ay}$, and generate $\sigma := (A, B, C, D)$.
7) Generate $\tau_{\mathcal{B}} \leftarrow \mathsf{MAC}(\mathsf{k_{mac}}, \sigma||n_{\mathcal{D}}||n_{\mathcal{B}})$.

In the above algorithm, if any check is failed, $\mathcal{B}$ aborts the process. If not, $\mathcal{B}$ sends $(\sigma, \tau_{\mathcal{B}})$ to $\mathcal{D}$, and sends $(U, \mathsf{dpk}, \mathsf{ID_{bank}}, \mathsf{ID_{user}})$ to $\mathcal{T}$ to backup for detecting possible double-spender. $\mathsf{ID_{bank}}$ is the identity of the commercial bank which the user account belongs to, and $\mathsf{ID_{user}}$ is the identity of the user.

5. Upon receiving $(\sigma, \tau_{\mathcal{B}})$, $\mathcal{D}$ switches into SW and verifies $\tau_{\mathcal{B}}$ using MAC, $\mathsf{k_{mac}}$ and $n'_{\mathcal{B}}$. Then, Data Handler calls $\mathsf{Data\_Seal}()$ to seal σ and generates $blob_\sigma$. Finally, Logic Engine deletes $n_{\mathcal{D}}$, $n'_{\mathcal{B}}$ and $\mathsf{k_{mac}}$ from Secure Memory.

Pre-Compute. After the above step, $\mathcal{D}$ returns back to NW. AEP-M Proxy executes pre-computation in the background (off-line) to prepare for the following **Spend** phase. Preprocessing Engine calls $\mathsf{AEPM\_NW\_PreCmpt}()$ to generate a blinded coin:

$$(l, R, S, T, W) \leftarrow \mathsf{AEPM\_NW\_PreCmpt}(blob_{params}, blob_\sigma),$$

where the algorithm consists of the following steps.

1) Get $params$ and digital coin σ by directly reading the plaintext part of $blob_{params}$ and $blob_\sigma$ respectively.
2) Parse σ as (A, B, C, D).
3) Choose $l \xleftarrow{\$} \mathbb{Z}_p^*$ and compute $(R, S, T, W) := (A^l, B^l, C^l, D^l)$.
4) Output (l, R, S, T, W).

Preprocessing Engine stores (l, R, S, T, W) together with $blob_\sigma$.

Spend. This is an interactive phase executed between a user with his mobile device $\mathcal{D}$ and a merchant $\mathcal{M}$, which enables $\mathcal{D}$ to anonymously pay some digital coins to $\mathcal{M}$.

1. App of $\mathcal{D}$ sends a nonce $\bar{n}_{\mathcal{D}} \xleftarrow{\$} \{0,1\}^\lambda$ to the merchant $\mathcal{M}$ for initiating a transaction.
2. Receiving $\bar{n}'_{\mathcal{D}}$, $\mathcal{M}$ chooses a nonce $n_{\mathcal{M}} \xleftarrow{\$} \{0,1\}^\lambda$ and generates a signature $\beta \leftarrow \mathsf{Sign}(\mathsf{sk}_{\mathcal{M}}, \texttt{"Spend"}||\mathsf{info})$ where $\mathsf{info} := (v, date, trans, \mathsf{pk}_{\mathcal{M}}, \bar{n}'_{\mathcal{D}}, n_{\mathcal{M}})$. info is the string collection containing the amount value v of coins to pay, transaction $date$, other necessary transaction information and the related nonce values. $\mathcal{M}$ sends $(\mathsf{info}, \mathsf{cert}_{\mathcal{M}}, \beta)$ to $\mathcal{D}$. In fact, issued by CA, $\mathsf{cert}_{\mathcal{M}}$ is $\mathcal{M}$'s certificate, containing $\mathsf{ID}_{\mathcal{M}}$, $\mathsf{pk}_{\mathcal{M}}$ and the signature $\mathsf{Sign}_{\mathsf{CA}}(\mathsf{ID}_{\mathcal{M}}||\mathsf{pk}_{\mathcal{M}})$, where $\mathsf{ID}_{\mathcal{M}}$ indicates the identitiy of $\mathcal{M}$.
3. When $\mathcal{D}$ receives the above data, AEP-M Proxy assembles the command to request AEP-M Service for payment. Without loss of generality, we assume that the user has a coin of value δ such that $\delta \geq v$. For the case that $\delta < v$, the user could spend another several coins in the same way in order that the sum amounts value of all coins equals v. On account of the request, $\mathcal{D}$'s environment is switched into SW. First, Logic Engine verifies β using Verify and $\mathsf{pk}_{\mathcal{M}}$ with $\mathsf{cert}_{\mathcal{M}}$. Then, $\mathcal{D}$ enters the secure GUI after authenticating the user's inputted PIN (or fingerprint). Relying on Framebuffer, the secure GUI displays $\mathsf{ID}_{\mathcal{M}}$ and the content of v, $date$ and $trans$. It is important for the user to confirm the exact $\mathsf{ID}_{\mathcal{M}}$ and transaction information in case an adversary falsifies the transaction. When the user presses the button of "OK", Logic Engine calls $\mathsf{AEPM\_SW\_Spend}()$ to create a master serial number $\boldsymbol{Z}$ of value v of coins together with a proof π of its validity, using the related pre-computation result as:

$$(\boldsymbol{Z}, \pi) \leftarrow \mathsf{AEPM\_SW\_Spend}(blob_{params}, blob_m, blob_\sigma||(l, R, S, T, W), \mathsf{info}),$$

where the detailed process is presented as follows:

1) Unseal the blobs to get $params$, (m, CT, δ) and (A, B, C, D) by calling $\mathsf{Data\_Unseal}()$.

2) Check whether $\bar{n}'_{\mathcal{D}} = \bar{n}_{\mathcal{D}}$.
3) Represent v by bits: $v = b_n b_{n-1} ... b_0$ and set $\Phi := \{i |\ 0 \leq i \leq n \wedge b_i = 1\}$.
4) For each $i \in \Phi$ from n to 0, based on CT, select uniformly at random an unspent node $s_i \in \mathcal{S}_n$ of level $n - i$ in the tree, and then mark it as the spent one.
5) For each chosen node s_i, compute $t_{s_i} := g_{s_i}^m$, and form three sets: $\boldsymbol{s} := \{s_i | i \in \Phi\}$, $\boldsymbol{g_s} := \{g_{s_i} | i \in \Phi\}$ and $\boldsymbol{t_s} := \{t_{s_i} | i \in \Phi\}$. Set $\boldsymbol{Z} := (\boldsymbol{s}, \boldsymbol{t_s})$.
6) Choose a random number $\bar{r} \xleftarrow{\$} \mathbb{Z}_p^*$, compute $L_i := g_{s_i}^{\bar{r}}$ for each $i \in \Phi$, form a set $\boldsymbol{L} := \{L_i | i \in \Phi\}$ and compute $\overline{L} := B^{l \cdot \bar{r}}$.
7) Compute $\bar{c} := \mathsf{H}_2(\boldsymbol{g_s} || \boldsymbol{t_s} || R || S || T || W || \boldsymbol{L} || \overline{L} || \mathsf{info})$.
8) Compute $\bar{z} := \bar{r} + \bar{c} \cdot m \ (mod\ p)$.
9) Set $\pi := (R, S, T, W, \bar{c}, \bar{z})$.
10) Delete (l, R, S, T, W) from the non-volatile storage.
11) Update CT and $\delta := \delta - v$. If $\delta > 0$, call $\mathsf{Data\_Seal}()$ again to regenerate $blob_m$ using the updated CT and δ, else delete $blob_m$ and $blob_\sigma$.

After the API finally returns, $\mathcal{D}$ switches back into NW and sends $(\boldsymbol{Z}, \pi)$ to $\mathcal{M}$.

4. $\mathcal{M}$ sets $\mathsf{Tr} := (\mathsf{info}, \boldsymbol{Z}, \pi)$ and verifies Tr by the means of calling the specialized verification algorithm $\mathsf{Tr\_Verify}()$ as:

$$res \leftarrow \mathsf{Tr\_Verify}(params, \overline{\mathsf{pk}_{\mathcal{B}}}, \mathsf{Tr}),$$

where the algorithm runs in detail as follows:

1) Parse Tr as $(\mathsf{info}, \boldsymbol{Z} = (\boldsymbol{s}, \boldsymbol{t_s}), \pi = (R, S, T, W, \bar{c}, \bar{z}))$.
2) For any two nodes in $\boldsymbol{s}$, check that the one does not belong to the subtree rooted at the other one (i.e., each node is not a prefix of any other one).
3) Compute $\overline{L}' := S^{\bar{z}} \cdot W^{-\bar{c}}$, $L'_i := g_{s_i}^{\bar{z}} \cdot t_{s_i}^{-\bar{c}}$ for each $s_i \in \boldsymbol{s}$, and set $\boldsymbol{L}' := \{L'_i | i \in \Phi\}$.
4) Compute $\bar{c}' := \mathsf{H}_2(\boldsymbol{g_s} || \boldsymbol{t_s} || R || S || T || W || \boldsymbol{L}' || \overline{L}' || \mathsf{info})$.
5) Check whether the relations $R \neq 1$, $W \neq 1$, $e(R, Y) = e(S, \tilde{g})$, $e(T, \tilde{g}) = e(R \cdot W, X)$ and $\bar{c}' = \bar{c}$ hold.
6) If all the above checks are passed, then $res := true$, else $res := false$.

According to the verification result res, $\mathcal{M}$ decides whether to accept the payment from $\mathcal{D}$ and provide services or goods for the user. If $\mathcal{M}$ accepts the transaction, he sends $\mathcal{D}$ a receipt $\theta_{\mathcal{M}} \leftarrow \mathsf{Sign}(\mathsf{sk}_{\mathcal{M}}, \texttt{"receipt"} || \mathsf{Tr})$ as the proof of accepting digital coins.

Pre-Compute. After Step 3 above, AEP-M Proxy of $\mathcal{D}$ in NW executes pre-computation again in the background to generate a new tuple (l', R', S', T', W') w.r.t. some $blob_\sigma$, if exists, for the next **Spend** use.

Deposit. In this phase, $\mathcal{M}$ could deposit money from Tr to his preferable bank $account_{\mathcal{M}}$ through the central bank $\mathcal{B}$.

1. $\mathcal{M}$ generates a signature $\gamma \leftarrow \mathsf{Sign}(\mathsf{sk}_{\mathcal{M}}, \texttt{"Deposit"} || \mathsf{Tr} || account_{\mathcal{M}})$. Then he sends Tr, $account_{\mathcal{M}}$ and γ together with $\mathsf{cert}_{\mathcal{M}}$ to $\mathcal{B}$.
2. $\mathcal{B}$ first verifies γ using Verify and $\mathsf{cert}_{\mathcal{M}}$. Secondly, $\mathcal{B}$ retrieves $\mathsf{pk}_{\mathcal{M}}$ from info and checks whether it is the same one inside $\mathsf{cert}_{\mathcal{M}}$. Thirdly, $\mathcal{B}$ computes $\mathsf{H}_3(\mathsf{Tr})$ and queries database $\mathsf{DB}_{\mathsf{Tr}}$ to check whether Tr has been used before. If not, $\mathcal{B}$ runs the verification algorithm $\mathsf{Tr\_Verify}()$ to verify the validity of Tr. If it is valid, $\mathcal{B}$ immediately transfers the exact amount v of real money to $account_{\mathcal{M}}$ with the help of some commercial bank.
3. $\mathcal{B}$ detects double-spending off-line after the above step. The detection process is presented as follows:

1) Retrieve $\boldsymbol{s}$ and $\boldsymbol{t_s}$ from Tr, and load $params'$.
2) For each $t_{s_i} \in \boldsymbol{t_s}$ and each $f \in \mathcal{F}_n(s_i)$, compute $d_{s_i \mapsto f} := e(t_{s_i}, \tilde{g}_{s_i \mapsto f})$ and $d_{s_i, f} := \mathsf{H}_4(d_{s_i \mapsto f})$.
3) Set $\boldsymbol{d} := \{d_{s_i, f} | s_i \in \boldsymbol{s} \wedge f \in \mathcal{F}_n(s_i)\}$.
4) Insert the item $(\mathsf{H}_3(\mathsf{Tr}), \mathsf{Tr}, \boldsymbol{d})$ into $\mathsf{DB}_{\mathsf{Tr}}$.

5) For each $d_{s_i,f}$, query $\mathsf{DB}_{\mathsf{Tr}}$ to check whether there exists a transaction Tr' that has the same $d_{s_i,f}$. If exists, send both Tr and Tr' to $\mathcal{T}$ through the secure channel for revealing the identity of the double-spender.

Identify. This phase endows $\mathcal{T}$ with the ability to reveal the identity of some double-spender.

1. When $\mathcal{T}$ receives the double-spending report $(\mathsf{Tr}, \mathsf{Tr}')$ from $\mathcal{B}$, it executes the verification algorithm $\mathsf{Tr\_Verify}()$ to verify the validity of Tr and Tr'. If both valid, $\mathcal{T}$ chooses one node $s_i \in \boldsymbol{s}$ from data of Tr and finds out the related r_{s_i} from its secret coin keys **sck** to recover U by computing $U := t_{s_i}^{1/r_{s_i}}$ (i.e. g^m). Likewise, $\mathcal{B}$ recovers U' from Tr'.
2. If $U = U'$, it indicates that Tr and Tr' lead to a double-spending. $\mathcal{T}$ would publish the spender's information $(\mathsf{Tr}, \mathsf{Tr}', U, \mathsf{dpk}, \mathsf{ID}_{\mathsf{bank}}, \mathsf{ID}_{\mathsf{user}})$. Then, some possible penalties on the user $\mathsf{ID}_{\mathsf{user}}$, for example deducting money from the user's account or temporally prohibiting the user from using e-payment system, would be triggered.

4.4 Optional Defense Mechanisms

Secure Non-Volatile Storage. Generally, our scheme could resist the conventional attacks. However, if there is a powerful adversary in NW of $\mathcal{D}$ to maliciously delete crucial data, such as $blob_m$ and $blob_\sigma$, from non-volatile storage, the user risks losing the digital coins withdrawn from $\mathcal{B}$. A possible solution here is to add the secure non-volatile storage which is only accessible to SW. Then, we can back-up the crucial data in it in case of lost. The specific technique has been implemented by Sun et al. [31].

Acknowledgement of receipt. During **Withdraw** phase, an adversary may maliciously impede $\mathcal{D}$ from receiving $(\sigma, \tau_{\mathcal{B}})$. Consequently, the user would lose the related digital coins even if anyone else cannot spend them. In fact, $\mathcal{B}$ could generate a receipt $\tau'_{\mathcal{D}} \leftarrow \mathsf{MAC}(\mathsf{k}_{\mathsf{mac}}, \texttt{"receipt"}||\sigma)$ and sends $(\texttt{"receipt"}, n'_{\mathcal{B}}, \tau'_{\mathcal{D}})$ back to $\mathcal{B}$ as the acknowledgement of receiving $(\sigma, \tau_{\mathcal{B}})$. If $\mathcal{B}$ does not immediately receive the acknowledgement, it would try to send $(\sigma, \tau_{\mathcal{B}})$ for several times, and eventually publish it on $\mathcal{B}$'s website for user downloading suppose $\mathcal{B}$ does not at last receive the acknowledgement.

4.5 Security Analysis

Based on the security definition and proof in [8], we describe the desired security properties of our anonymous e-payment scheme for mobile devices, and then analyze these properties.

Security Properties. Informally, AEP-M should satisfy the following properties:

- *Correctness* means that a user runs honestly the **Withdraw** protocol with the honest bank will obtain a divisible coin which is accepted by any honest merchant.
- *Unlinkability* requires that the bank, even colluding with malicious users and merchants, cannot decide whether two transcripts of the **Spend** protocol were produced by the same user even if the position of the spent parts (i.e., the spent nodes) in a coin can be revealed.
- *Traceability* requires that no coalition of users can spend more (and then accepted as deposit) than they withdrew, or double-spend a coin without revealing their identities.
- *Exculpability* requires that the bank, even colluding with malicious users and merchants, cannot falsely accuse an honest user for having double-spent a coin, even if it compromises the NW of the user's mobile device.
- *Confidentiality* requires that no coalition of malicious users and merchants can reveal any secret information (e.g., password) of a user from the **Withdraw** protocol, even if the NW of the user's device has been corrupted.
- *Authenticity* requires that 1) only the user in possess of a mobile device with TrustZone, a valid account and the corresponding password, can authenticate himself to the bank; 2) only a merchant accepted by the user can deposit the transaction from the user.

Security Analysis. As giving a formal security proof is outside the scope of this paper, we only provide an informal security analysis to argue that AEP-M satisfies the above security properties, if the $\text{DDH}_{\mathbb{G}_1}$, Weak-EXDH, B-4-LRSW assumptions hold, Σ_1 is EUF-CMA secure [12], Σ_2 is unforgeable against chosen message and chosen verification queries attack (uf-cmva) [11], Σ_3 is IND-CPA secure, and Σ_4 is IND-CPA secure. As Σ_2 is uf-cmva secure and Σ_4 is IND-CPA secure, $blob_{params}$, $blob_{\sigma}$ and $blob_m$ provide the integrity protection, and $blob_m$ provides the confidentiality of m. For the sake of simplicity, we omit the analysis of these blobs in the following analysis. Note that we could actually prove AEP-M achieves correctness, unlinkability, traceability and exculpability, based on the security proofs in [8] and [3]. Confidentiality and authenticity of AEP-M could also be proved in the standard way. Below, we interpret the intuition why AEP-M satisfies the above security properties.

- *Correctness:* This can be checked by working through all the phases.
- *Unlinkability:* In a transcript of the **Spend** protocol, only $(\boldsymbol{Z} = (\boldsymbol{s}, \boldsymbol{t_s}), \pi = (R, S, T, W, \bar{c}, \bar{z}))$ is associated with the spent coin (m, σ). Below, we show that $\boldsymbol{s}$ is the unique revealed information about the spent coin, and thus $(\boldsymbol{Z}, \pi)$ cannot be linked to other transactions. Recall that $(\bar{c}, \bar{z})$ is a non-interactive zero-knowledge proof on proving knowledge of m such that $t_{s_i} = g_{s_i}^m$ for each $s_i \in \boldsymbol{s}$ and $W = S^m$. Thus, $(\bar{c}, \bar{z})$ does not reveal any information of m. Note that (R, S, T, W) is randomized with a randomness l, $S = R^y$, $T = R^{x+mxy} = R^x \cdot (R^m)^{xy}$ and $W = S^m = (R^m)^y$. Following the fact that given g, $U = g^m$ and R, it is hard to decide if an element of $\mathbb{G}_1$ is random or equal to R^m under the $\text{DDH}_{\mathbb{G}_1}$ assumption, the tuple (R, S, T, W) does not reveal the information of (m, σ). Besides, based on the security proof in [8], g_s^m is indistinguishable from a random element in $\mathbb{G}_1$ under the Weak-EXDH assumption. Thus, using the standard hybrid argument, we have $\boldsymbol{t_s}$ is computationally indistinguishable from random elements. Note that σ must be produced by $\mathcal{B}$, as $\mathsf{k_{mac}}$ is only known by the SW of $\mathcal{D}$ and $\mathcal{B}$ (c.f., the security analysis of *Confidentiality*), and $\tau_{\mathcal{B}}$ provides the integrity protection of σ due to the uf-cmva security of Σ_2. In conclusion, $\boldsymbol{s}$ is the only revealed information on the spent coin.
- *Traceability:* No probabilistic polynomial time adversary can forge a coin that cannot be traced to an execution of the **Withdraw** protocol under the B-4-LRSW assumption. Furthermore, t_{s_i} for each $s_i \in \boldsymbol{s}$ and W must be generated using the same secret key m, following the zero-knowledge proof $(\bar{c}, \bar{z})$ is sound. According to the verification equations $e(R, Y) = e(S, \tilde{g})$ and $e(T, \tilde{g}) = e(R \cdot W, X)$, we have T contains the same m, i.e., m is certified by $\mathcal{B}$. Thus, malicious users cannot spend more than they have withdrawn, and any double-spending can be detected by $\mathcal{B}$ and identified by the trusted authority $\mathcal{T}$.
- *Exculpability:* In fact, $(\bar{c}, \bar{z})$ is a signature proof of knowledge on message info. Thus, an adversary attempting to spend a coin of an honest user must know the secret key m. However, given $U = g^m$, the corresponding zero-knowledge proof $(c_{\mathcal{D}}, s_{\mathcal{D}})$, and $(\boldsymbol{Z}, \pi)$ generated with the coin, it is hard to obtain m under the discrete-logarithm assumption. Therefore, exculpability is obtained.
- *Confidentiality:* A pair $(C_{\mathcal{B}}, \alpha)$ accepted by SW of a device $\mathcal{D}$ must be generated by $\mathcal{B}$, since Σ_1 is EUF-CMA secure and $n_{\mathcal{D}}$ collides with other nonce with negligible probability. Besides Σ_3 is IND-CPA secure, so $\mathsf{k_{mac}}$ and $\mathsf{k_{enc}}$ are only known by the SW and the bank. Moreover, Σ_2 is uf-cmva secure and Σ_4 is IND-CPA secure. Thus, the user's password pwd can only be known by $\mathcal{B}$. As a result, AEP-M satisfies the confidentiality.
- *Authenticity:* 1) The certificate $\mathsf{cert}_{\mathcal{D}}$ certifies that a mobile device owning dpk is equipped with ARM TrustZone. By the "Challenge-and-Response" paradigm (i.e., a ciphertext $C_{\mathcal{B}}$ is sent as challenge, and a nonce $n_{\mathcal{B}}$ and a tag $\tau_{\mathcal{D}}$ are sent back as response), the mobile device $\mathcal{D}$ can prove knowledge of the secret key dsk and provide the authenticity and integrity of U and $C_{\mathcal{D}} = \mathsf{Enc_{sym}}(\mathsf{k_{enc}}, account_{\mathcal{D}} || pwd)$ by generating a tag $\tau_{\mathcal{D}}$ with $\mathsf{k_{mac}}$. The central bank $\mathcal{B}$ is convinced that $\mathcal{D}$ supports TrustZone and U and $C_{\mathcal{D}}$ are created by $\mathcal{D}$. By checking the validity of $(account_{\mathcal{D}}, pwd)$, $\mathcal{B}$ is convinced that the user of device

$\mathcal{D}$ is the owner of $account_{\mathcal{D}}$. 2) The signature β must be produced by the merchant $\mathcal{M}$ with identity $\mathsf{ID}_{\mathcal{M}}$, as the digital signature algorithm Σ_1 is EUF-CMA secure. By displaying $(\mathsf{ID}_{\mathcal{M}}, v, date, trans)$, the user can decide whether $\mathcal{M}$ is the desired merchant. The public key $\mathsf{pk}_{\mathcal{M}}$ included in info is signed as $(\bar{c}, \bar{z})$. In the **Deposit** phase, a signature γ is generated, and thus only the merchant knowing the secret key $\mathsf{sk}_{\mathcal{M}}$ can deposit for $\mathsf{Tr} = (\mathsf{info}, \boldsymbol{Z}, \pi)$. In all, AEP-M has the authenticity.

5 Implementation and Evaluation

In this section, we first present the prototype system of AEP-M from both aspects of hardware and software. Afterwards, we show the efficiency of the proposed scheme. Finally, we give the performance evaluation and analysis based on our prototype system.

5.1 Implementation

Hardware Platform. To simulate real environment, we implement the role of merchant on one PC platform, and central bank as well as trusted authority on another one. For simulating mobile device, we leverage a development board Zynq-7000 AP Soc Evaluation Kit [40] to implement functions of AEP-M. It is TrustZone-enabled and equipped with ARM Cortex-A9 MPCore, 1GB DDR3 memory and On-Chip Memory (OCM) module including 256 KB SRAM. However, this SRAM is initialized by BootROM once the board is powered on, which prevents us from reading its initial data. Then we utilize an SRAM chip that is the type IS61LV6416-10TL [34] to act as our SRAM PUF. SRAM initial data is transferred to the Zynq development board by an FPGA implementation of Universal Asynchronous Receiver/Transmitter (UART) in Verilog hardware description language. A UART receiver in the Zynq board receives and stores the SRAM data in a RAM cache. Then the processor can fetch the SRAM data in the RAM cache via the bus. In addition, the methods given in [31] are applied to fulfill Secure Memory for data storage and Framebuffer for secure display.

Software Implementation. The software implementation on the development board for mobile device is divided into two parts. In *secure world*, we use Open Virtualization SierraTEE as the basic TEE OS which is compliant with GP's TEE Specifications [33]. For Key Manager of AEP-M Service, the fuzzy extractor of PUF is constructed on an open source BCH code [19]. For Crypto Library, we use OpenSSL-1.0.2g for general cryptographic algorithms, and Pairing-Based Cryptography (PBC) 0.5.14 library for computations of elliptic curves and bilinear maps. The security parameter λ is set to 128 (bits), so we choose SHA256 for H_3 and H_4, HMAC-SHA256 for MAC, 3072-bit RSA for $\mathsf{Enc_{asym}}$-$\mathsf{Dec_{asym}}$, 256-bit ECDSA for Sign-Verify and 128-bit AES-CBC for $\mathsf{Enc_{sym}}$-$\mathsf{Dec_{sym}}$. The implementation of KDF for generating different types of keys makes references to the relevant methods in PKCS #11 v2.40 [20]. 5268 lines of code (LOC) in C language totally make up our components and auxiliary functions in *secure world*. In *normal world*, we run a Linux as REE OS with kernel 3.8.6. The SierraTEE project provides the Linux with NW-Driver and GP TEE Client API. AEP-M Proxy totally comprises 2879 LOC. Besides we program one test application that could execute upon AEP-M scheme. It contains 1268 LOC for App running in NW and 661 LOC for App Trustlet in SW. Furthermore, there are several tens of thousands of LOC for implementing central bank, merchant and trusted authority.

5.2 Efficiency

AEP-M achieves some valuable and meaningful properties (e.g., sensitive data computing and management). Nevertheless, our customized construction makes it difficult to compare with other anonymous e-payment solutions which are mainly designed for PC platform. We therefore only list the main statistics in efficiency of our AEP-M for coins of value 2^n in

Table 2. Main statistics in efficiency of AEP-M for coins of value 2^n. The space and time complexities are given from three major entities' points of view. $E_{\mathbb{G}_1}$ refers to an exponentiation in $\mathbb{G}_1$, $E^2_{\mathbb{G}_1}$ to 2 simultaneous exponentiations in $\mathbb{G}_1$, and P to a pairing computation. The symbols of cryptographic functions, such as Verify, denote their corresponding computations respectively. $|\mathbb{G}_1|$ and $|\mathbb{G}_2|$ indicate the sizes of elements in $\mathbb{G}_1$ and $\mathbb{G}_2$. $|p|$ means the size of random number in $\mathbb{Z}^*_p$.

Entities	$\mathcal{D}$	$\mathcal{B}$	$\mathcal{M}$
Size of Public Parameters	$(2^{n+1}-1)\|\mathbb{G}_1\|$	$(2^{n+1}-1)\|\mathbb{G}_1\|$ $+(n+1)2^n\|\mathbb{G}_2\|$	$(2^{n+1}-1)\|\mathbb{G}_1\|$
Coin Size	$\|p\|+4\|\mathbb{G}_1\|$	–	–
Withdraw Computations	1 Verify $+1$ $\mathsf{Dec}_{\mathsf{asym}}$ $+2E_{\mathbb{G}_1}$	1 $\mathsf{Enc}_{\mathsf{asym}}$ $+1$ Sign $+2E^2_{\mathbb{G}_1}+3E_{\mathbb{G}_1}$	–
Pre-Compute Computations	$4E_{\mathbb{G}_1}$	–	–
Spend Computations	2 Verify $+\|\Phi\|E_{\mathbb{G}_1}$	–	2 Sign $+(\|\Phi\|+1)E^2_{\mathbb{G}_1}$ $+4P$
Double-Spending Detection Computations	–	$(\|\Phi\|+1)E^2_{\mathbb{G}_1}+(v+4)P$	–

Table 2. The statistics focus on the critical space and time complexities, during which the data sizes in constant level and the computations in low complexity (e.g., MAC and $\mathsf{Enc}_{\mathsf{sym}}$) are omitted. In the table, as some amounts of computations are related to consuming the coins of value v, we use $|\Phi|$ (where $1 \leq |\Phi| \leq n$) to denote the number of elements in set Φ.

From the perspective of $\mathcal{D}$, if we choose BN curve [35] to implement the scheme for $\lambda = 128$ and $n = 10$ (enabling users to divide one coin into 1024 parts), the storage space for public parameters is 66 KBytes and for coin size 160 Bytes. These can easily be handled by any mobile devices. $\mathcal{M}$ needs the same space for public parameters, while $\mathcal{B}$ requires additional 721 KBytes. Besides, the space required to store $\mathsf{DB}_{\mathsf{Tr}}$ of one million transactions is in a range from 320 MBytes to 17 GBytes, which is still practical for $\mathcal{B}$.

For computational complexity, except for some verifications and decryptions, $\mathcal{D}$ just needs several exportations in $\mathbb{G}_1$ during both **Withdraw** and **Spend**. Especially, there are only $|\Phi|$ exportations required by the most frequent execution, i.e. online **Spend**. The most expensive step for $\mathcal{B}$ is detecting double-spending because of $v+4$ pairing computations. Fortunately, this step is completely off-line, so that it would not block the normal online transactions and is quite feasible for the modern data center running behind $\mathcal{B}$. Likewise, $\mathcal{M}$ could undertake the computations during **Spend** without much pressure. Thus, in general, under high security level, the efficiency of AEP-M is theoretically feasible for current mainstream storage and computing power, even if provided by resource-constrained mobile devices.

5.3 Performance Evaluation

Since the resource-constrained mobile device is the performance bottleneck as well as the focus of our attention, we measure the performance of AEP-M on the prototype system revolving around mobile device through a series of experiments with different parameters. Referring to ISO/IEC 15946-5 standard [35], we select a kind of elliptic curve that is suitable for realizing Type-3 pairings. It is BN curve with embedding degree 12. For testing the security level of 128-bit, we conduct the experiments using BN256. Our experiments simulate the whole AEP-M running process covering instantiated scheme, TrustZone switch and sensitive data management. Each average experimental result is taken over 50 test-runs.

For coins of value 2^{10} and 2^{20} respectively, and spending 287 of them, Figure 4 illustrates the average time overheads of critical processes including the computations of **Withdraw**, *Pre-Compute* and **Spend** on mobile device for user side and **Spend** on PC for merchant side. The results show that using the curve with high security level for mobile device, the frequent computations about either *Pre-Compute* or **Spend** only take less than 450 milliseconds (ms), while infrequent and time-consuming **Withdraw** spends less than 660 ms. Even if the computation amount of **Spend** for merchant part is quite large, the time overhead is indeed

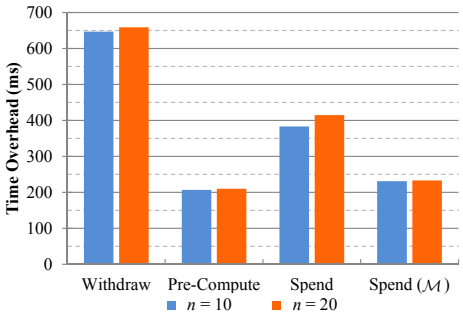

Fig. 4. Time overheads of the critical processes for coins of 2^n and $v = 287$.

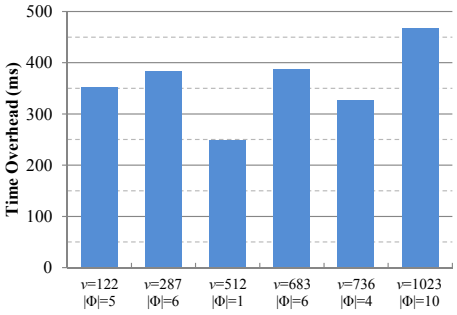

Fig. 5. Time overheads of **Spend** phase for $n = 10$ with different values of v.

low on PC platform. Additionally, the value of n has no significant influence on the time overhead for these critical processes.

Figure 5 shows the average time overheads of **Spend** phase on mobile device for user side using $n = 10$ and $v \in \{122, 287, 512, 683, 736, 1023\}$. Depending on different values of v, $|\Phi|$ takes corresponding values from v's representations by bits. From the figure, we can see that as the value of $|\Phi|$ increases, the time overheads of **Spend** have evident growth, which nearly has nothing to do with v itself, big or small. Encouragingly, under the worst-case scenario where $|\Phi| = 10$, the resulting overhead spends less than 500ms, which is completely acceptable for a mobile user.

Actually, a direct performance comparison of our competitive scheme to others is also meaningless because of some fundamental differences. Anyhow, according to our efficiency analysis and experimental results, AEP-M can be considered as a reasonably efficient scheme for mobile device. In regard to adopting modern mobile devices that are much more powerful than our development board, with a more optimal library to implement elliptic curves and parings, the time overhead of our scheme could be further decreased.

6 Conclusion

In this paper, we propose AEP-M, a complete and practical anonymous e-payment scheme using TrustZone and divisible e-cash. AEP-M tackles both security and privacy issues specially for mobile electronic payers. The scheme allows users to withdraw a coin of value 2^n and spend it in several times by dividing it. Pre-computation and the bit-decomposition technique for coin's representation are carefully taken into our consideration to raise scheme's efficiency and flexibility. What is more, TrustZone provides data and execution protection for AEP-M. The root of trust from SRAM PUF, key derivation and sensitive data management collectively further enhance the security of the scheme. Our implementation and evaluation convince that AEP-M is quite practical for payers using resource-constrained mobile devices.

References

1. Mobile privacy disclosures: Building trust through transparency. Tech. rep., Federal Trade Commission (February 2013)
2. ARM Limited: ARM Security Technology–Building a Secure System using TrustZone Technology (April 2009)
3. Bernhard, D., Fuchsbauer, G., Ghadafi, E., Smart, N.P., Warinschi, B.: Anonymous attestation with user-controlled linkability. International Journal of Information Security 12(3), 219–249 (2013)
4. Brickell, E., Camenisch, J., Chen, L.: Direct anonymous attestation. In: Proceedings of ACM CCS 2004. pp. 132–145. ACM (2004)

5. Camenisch, J., Hohenberger, S., Lysyanskaya, A.: Compact e-cash. In: EUROCRYPT 2005. pp. 302–321. Springer (2005)
6. Canard, S., Gouget, A.: Divisible e-cash systems can be truly anonymous. In: EUROCRYPT 2007. pp. 482–497. Springer (2007)
7. Canard, S., Gouget, A.: Multiple denominations in e-cash with compact transaction data. In: Financial Cryptography and Data Security (FC). pp. 82–97. Springer (2010)
8. Canard, S., Pointcheval, D., Sanders, O., Traore, J.: Divisible e-cash made practical. In: Public-Key Cryptography (PKC). pp. 77–100. Springer (2015)
9. Chan, Agnes, H., Frankel, Y., Tsiounis, Y.: Easy come - easy go divisible cash. Theory and Application of Cryptographic Techniques (1998)
10. Chaum, D.: Blind signatures for untraceable payments (1983)
11. Dodis, Y., Kiltz, E., Pietrzak, K., Wichs, D.: Message authentication, revisited. In: EUROCRYPT 2012, pp. 355–374. Springer (2012)
12. Goldwasser, S., Micali, S., Rivest, R.L.: A digital signature scheme secure against adaptive chosen-message attacks. SIAM Journal on Computing 17(2), 281–308 (1988)
13. Guajardo, J., Kumar, S.S., Schrijen, G.J., Tuyls, P.: FPGA intrinsic PUFs and their use for IP protection. Springer (2007)
14. Izabachene, M., Libert, B.: Divisible e-cash in the standard model. In: Proceedings of Pairing 2012. pp. 314–332. Springer (2013)
15. Jang, J., Kong, S., Kim, M., Kim, D., Kang, B.B.: SeCReT: Secure channel between rich execution environment and trusted execution environment. NDSS 2015 (2015)
16. Kim, C., Wang, T., Shin, N., Kim, K.S.: An empirical study of customers perceptions of security and trust in e-payment systems. Electronic Commerce Research and Applications 9(1), 84–95 (2010)
17. Li, W., Li, H., Chen, H., Xia, Y.: AdAttester: Secure online mobile advertisement attestation using trustzone. In: Proceedings of MobiSys 2015. pp. 75–88. ACM (2015)
18. Lim, A.S.: Inter-consortia battles in mobile payments standardisation. Electronic Commerce Research and Applications 7(2), 202–213 (2008)
19. Morelos-Zaragoza, R.: Encoder/decoder for binary bch codes in c (version 3.1)
20. OASIS Standard: PKCS #11 Cryptographic Token Interface Current Mechanisms Specification Version 2.40 (April 2015)
21. Okamoto, T., Ohta, K.: Universal electronic cash. In: CRYPTO 1991. pp. 324–337 (1992)
22. Oren, Y., Sadeghi, A.R., Wachsmann, C.: On the effectiveness of the remanence decay side-channel to clone memory-based pufs. In: CHES 2013, pp. 107–125. Springer (2013)
23. Paquin, C., Zaverucha, G.: U-Prove Cryptographic Specification V1.1. Microsoft (2013)
24. Plateaux, A., Coquet, V., Vernois, S., Lacharme, P., Murty, K., Rosenberger, C.: A privacy preserving e-payment architecture. In: FC 2013. p. 402. Springer (2013)
25. Preibusch, S., Peetz, T., Acar, G., Berendt, B.: Purchase details leaked to paypal. In: Financial Cryptography and Data Security (FC). pp. 217–226. Springer (2015)
26. Reaves, B., Scaife, N., Bates, A., Traynor, P., Butler, K.R.B.: Mo(bile) money, mo(bile) problems: analysis of branchless banking applications in the developing world. In: Proceedings of the 24th USENIX Conference on Security Symposium (2015)
27. Rial, A.: Privacy-Preserving E-Commerce Protocols. Ph.D. thesis, Faculty of Engineering Science, KU Leuven (March 2013)
28. Santos, N., Raj, H., Saroiu, S., Wolman, A.: Using ARM TrustZone to build a trusted language runtime for mobile applications. In: Proceedings of ASPLOS 2014. pp. 67–80. ACM (2014)
29. Stilgherrian: Apple pay isn't magic, and it isn't 'private'. http://www.zdnet.com/article/apple-pay-isnt-magic-and-it-isnt-private/ (2014), last accessed 20 February 2016
30. Suh, G.E., Devadas, S.: Physical unclonable functions for device authentication and secret key generation. In: 44th ACM/IEEE DAC 2007. pp. 9–14 (2007)
31. Sun, H., Sun, K., Wang, Y., Jing, J.: TrustOTP: Transforming smartphones into secure one-time password tokens. In: Proceedings of CCS 2015. pp. 976–988. ACM (2015)
32. Tamrakar, S., Ekberg, J.E.: Tapping and tripping with NFC. In: Trust and Trustworthy Computing (TRUST), pp. 115–132. Springer (2013)
33. GlobalPlatform: Tee client api specification version 1.0 (2010)
34. Integrated Silicon Solution Inc: IS61LV6416-10TL. http://www.alldatasheet.com/datasheet-pdf/pdf/505020/ISSI/IS61LV6416-10TL.html
35. ISO/IEC: 15946-5: 2009 information technology-security techniques: Cryptographic techniques based on elliptic curves: Part 5: Elliptic curve generation (2009)

36. ISO/IEC: 11889:2015 information technology – trusted platform module library. http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=66510 (2015), last accessed 1 March 2016
37. Proxama: http://www.proxama.com/platform/ (2015), last accessed 15 October 2015
38. Sansa Security: Discretix. https://www.sansasecurity.com/blog/discretix-becomes-sansa-security/ (2014), last accessed 22 June 2014
39. Santander: Privacy policy - Santander Apple Pay. http://www.santander.co.uk/uk/apple-pay/privacy-policy/ (2016), last accessed 15 February 2016
40. Xilinx: Zynq-7000 all programmable soc zc702 evaluation kit. http://www.xilinx.com/products/boards-and-kits/EK-Z7-ZC702-G.htm
41. Yang, B., Feng, D., Qin, Y.: A lightweight anonymous mobile shopping scheme based on daa for trusted mobile platform. In: IEEE TrustCom 2014. pp. 9–17. IEEE (2014)
42. Yang, B., Yang, K., Qin, Y., Zhang, Z., Feng, D.: DAA-TZ: An effcient DAA scheme for mobile devices using ARM TrustZone. In: Trust and Trustworthy Computing, pp. 209–227. Springer (2015)
43. Zhao, S., Zhang, Q., Hu, G., Qin, Y., Feng, D.: Providing root of trust for ARM TrustZone using on-chip SRAM. In: Proceedings of TrustED 2014. pp. 25–36. ACM (2014)