AEP-M: Practical Anonymous E-Payment for Mobile Devices Using ARM TrustZone and Divisible E-Cash

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

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 · 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

M. Bishop and A.C.A. Nascimento (Eds.): ISC 2016, LNCS 9866, pp. 130–146, 2016.
DOI: 10.1007/978-3-319-45871-7_9

business operations, e-payment, which builds up e-commerce, has become one of the most critical issues for successful business and financial services. 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 [8]. Unfortunately, with the widespread use of 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 [9]. Generally, the spending procedure is associated with the authenticated identity to indicate who withdraws digital coins from banks, 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 [11]. Such sensitive information implies one's political view, location, religion or health condition. Statistically, mobile users account for a high proportion among all the e-payment users [10]. Thus, the issue of information leakage is 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) and U-Prove. Based on DAA, Yang et al. [19] 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 [5], 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 [1] has been used to execute security-critical services [17]. 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. [4], 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 [6], 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, 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, while the original scheme [4] cannot, where the maximum denomination of a coin is 2^n.

We implement a prototype 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. Different from pre-paid cards, credit cards and electronic checks, e-cash system does a better job to construct anonymous e-payment. After

Chaum first introduced e-cash [5], Camenisch et al. [2] 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. Afterwards, some truly anonymous divisible e-cash systems [3] were described, but quite inefficiency. Recently, Canard et al. [4] proposed the first really efficient divisible e-cash system by defining one global binary tree. Our scheme takes it as a reference and further increases its efficiency and security according to our architecture of trusted mobile device.

TrustZone Technology. ARM TrustZone technology for the mobile devices can guarantee codes integrity and data security. Relying on TrustZone, many practical mobile schemes are proposed. For instance, AdAttester [7] was presented specially for secure mobile advertisement 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 [7].

2 Preliminaries

2.1 Notation

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

Let $\Lambda = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g, \widetilde{g})$ be a description of bilinear groups which 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 $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$, where g and $\widetilde{g}$ is the generator of $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively. In this paper, we only consider the Type-3 pairings [16].

2.2 ARM TrustZone

ARM TrustZone [1] 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. 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.

2.3 Physical Unclonable Functions

Physical Unclonable Functions (PUFs) [12] 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. 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.

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

Strictly speaking, only equipped with a root of trust, TrustZone becomes a real "trusted" execution environment (TEE) [22]. Because TrustZone almost does not internally install an available root key, it loses the capability to offer a root of trust. Employing a PUF can cover this shortage. 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 [6] 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 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

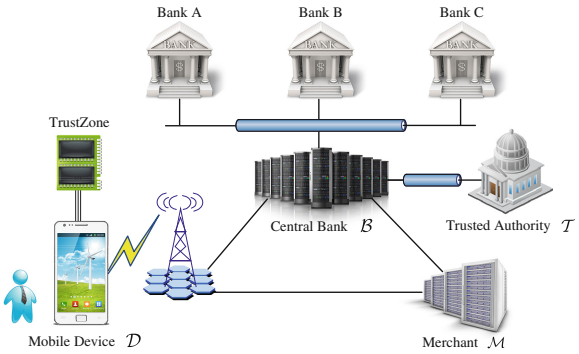

Fig. 1. System model of AEP-M.

user without revealing user's identity to any entities including $\mathcal{M}$ itself. 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. Figure 1 illustrates the system model for our 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.

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.

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

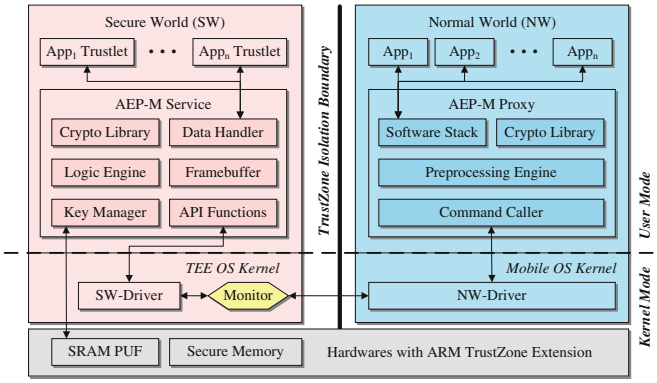

Fig. 2. Architecture of trusted mobile device for AEP-M.

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 [20]. 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, 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 different components from [20] are formally described as follows.

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. **Preprocessing Engine** executes precomputation for AEP-M after digital coins are withdrawn from central bank to the mobile device.

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. **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.

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.

Components in Hardwares. 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 [22] 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.

Key Derivation. Key Manager has the deterministic key derivation function $\mathsf{KDF}\colon \widetilde{\mathcal{S}} \times \{0,1\}^* \to \widetilde{\mathcal{K}}$, 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 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 Sect. 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: $(\mathsf{sk}_m, \mathsf{mk}_m) \leftarrow \mathsf{KDF}_{\mathsf{srk}}(\texttt{"storage\_key"}||\texttt{"Enc+MAC"}||U)$, and $blob_m \leftarrow \mathsf{Data\_Seal}(\texttt{"Enc+MAC"}, \mathsf{sk}_m, \mathsf{mk}_m, m||CT||\delta, U)$, where

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

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 [4], a unique and public global tree of depth n is used for all coins of value $V = 2^n$ as illustrated in Fig. 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.

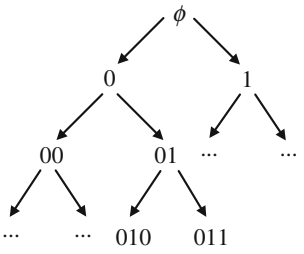

Fig. 3. Public global tree for all coins.

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 Sect. 2.1 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. 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 Sect. 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 $c'_{\mathcal{D}} = c_{\mathcal{D}}$.
(5) Decrypt $C_{\mathcal{D}}$ using $\mathsf{Dec}_{\mathsf{sym}}$ and $\mathsf{k}_{\mathsf{enc}}$: $account_{\mathcal{D}}||pwd \leftarrow \mathsf{Dec}_{\mathsf{sym}}(\mathsf{k}_{\mathsf{enc}}, C_{\mathcal{D}})$, 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 $\tau_{\mathcal{B}} \leftarrow \mathsf{MAC}(\mathsf{k}_{\mathsf{mac}}, \sigma||n_{\mathcal{D}}||n_{\mathcal{B}})$.

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 $(U, \mathsf{dpk}, \mathsf{ID}_{\mathsf{bank}}, \mathsf{ID}_{\mathsf{user}})$ to $\mathcal{T}$ to backup for detecting possible double-spender. $\mathsf{ID}_{\mathsf{bank}}$ is the identity of the commercial bank which the user account belongs to, and $\mathsf{ID}_{\mathsf{user}}$ 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, $\mathsf{k}_{\mathsf{mac}}$ 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 $\mathsf{k}_{\mathsf{mac}}$ 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*, $(m,\ CT,\ \delta)$ 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 $\mathbf{sck}$ 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 and Security Analysis

AEP-M satisfies the desired security properties such as unlinkability, traceability, exculpability, confidentiality and authenticity. Additional defense mechanisms could further enhance our scheme's security. The detailed description of these properties, mechanisms and the analysis can be found in the full paper [21].

5 Implementation and Evaluation

In this section, we first present the prototype 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 [18] to implement functions of AEP-M. It is TrustZone-enabled and equipped with ARM Cortex-A9 MPCore, 1 GB DDR3 memory and On-Chip Memory (OCM) module. We utilize an SRAM chip that is the type IS61LV6416-10TL [15] to act as our SRAM PUF. The processor can fetch the SRAM data in the RAM cache via the bus. In addition, the methods given in [13] are applied to fulfill 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 [14]. For Crypto Library, we use OpenSSL-1.0.2 g 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 $\mathsf{Enc}_{\mathsf{asym}} - \mathsf{Dec}_{\mathsf{asym}}$, 256-bit ECDSA for $\mathsf{Sign\text{-}Verify}$ and 128-bit AES-CBC for $\mathsf{Enc}_{\mathsf{sym}} - \mathsf{Dec}_{\mathsf{sym}}$. 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. 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 other entities.

5.2 Efficiency and Performance Evaluation

The specific analysis of AEP-M's efficiency also appears in the full paper [21]. 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. We select BN curve with embedding degree 12. For testing the security level of 128-bit, we conduct the experiments using BN256. 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, Fig. 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 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. Moreover, the time overhead is indeed low on PC platform.

Figure 5 shows the average time overheads of **Spend** phase on mobile device for user side using $n = 10$ and different v. $|\Phi|$ takes corresponding values from v's representations by bits. We can see that as the value of $|\Phi|$ increases, the time

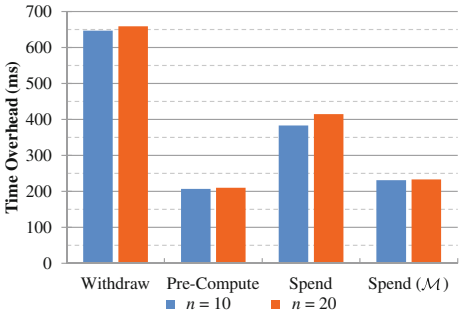

Fig. 4. Time overheads of the critical processes for coins of 2^n and $v = 287$.

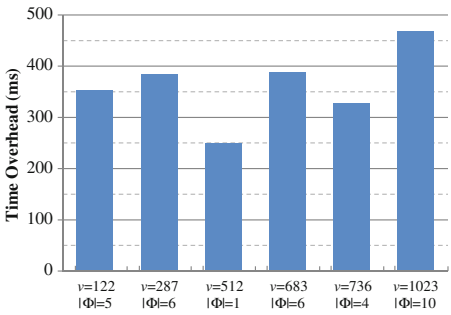

Fig. 5. Time overheads of **Spend** phase for $n = 10$ with different values of v.

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 500 ms, which is completely acceptable for a mobile user. According to our efficiency analysis and experimental results, AEP-M can be considered as a reasonably efficient scheme for mobile device.

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. Our implementation and evaluation convince that AEP-M is quite practical for payers using resource-constrained mobile devices.

Acknowledgment. This work was supported in part by grants from the National Natural Science Foundation of China (No. 91118006 and No. 61402455).

References

1. Limited ARM: ARM security technology-building a secure system using TrustZone technology, April 2009
2. Camenisch, J.L., Hohenberger, S., Lysyanskaya, A.: Compact e-cash. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 302–321. Springer, Heidelberg (2005)
3. Canard, S., Gouget, A.: Divisible e-cash systems can be truly anonymous. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 482–497. Springer, Heidelberg (2007)

4. Canard, S., Pointcheval, D., Sanders, O., Traoré, J.: Divisible e-cash made practical. In: Katz, J. (ed.) PKC 2015. LNCS, vol. 9020, pp. 77–100. Springer, Heidelberg (2015)
5. Chaum, D.: Blind signatures for untraceable payments. In: Chaum, D., Rivest, R.L., Sherman, A.T. (eds.) Advances in Cryptology, pp. 199–203. Springer, New York (1983)
6. Guajardo, J., Kumar, S.S., Schrijen, G.J., Tuyls, P.: FPGA intrinsic PUFs and their use for IP protection. In: Paillier, P., Verbauwhede, I. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2007. LNCS, vol. 4727, pp. 63–80. Springer, Heidelberg (2007)
7. 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)
8. Lim, A.S.: Inter-consortia battles in mobile payments standardisation. Electron. Commer. Res. Appl. **7**(2), 202–213 (2008)
9. Preibusch, S., Peetz, T., Acar, G., Berendt, B.: Purchase details leaked to PayPal (short paper). In: Böhme, R., Okamoto, T. (eds.) FC 2015. LNCS, vol. 8975, pp. 217–226. Springer, Heidelberg (2015)
10. 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)
11. Rial, A.: Privacy-preserving e-commerce protocols. Ph.D. thesis, Faculty of Engineering Science, KU Leuven, March 2013
12. 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)
13. Sun, H., Sun, K., Wang, Y., Jing, J.: Trust OTP: transforming smartphones into secure one-time password tokens. In: Proceedings of CCS 2015, pp. 976–988. ACM (2015)
14. GlobalPlatform: Tee client API specification version 1.0 (2010)
15. Integrated Silicon Solution Inc, IS61LV6416-10TL. http://www.alldatasheet.com/datasheet-pdf/pdf/505020/ISSI/IS61LV6416-10TL.html
16. ISO/IEC: 15946–5: 2009 Information Technology-Security Techniques: Cryptographic Techniques based on Elliptic Curves: Part 5: Elliptic Curve Generation (2009)
17. Proxama (2015). http://www.proxama.com/platform/. Accessed 15 Oct 2015
18. Xilinx: Zynq-7000 all programmable soc zc702 evaluation kit. http://www.xilinx.com/products/boards-and-kits/EK-Z7-ZC702-G.htm
19. 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)
20. Yang, B., Yang, K., Qin, Y., Zhang, Z., Feng, D.: DAA-TZ: an efficient DAA scheme for mobile devices using ARM TrustZone. In: Conti, M., Schunter, M., Askoxylakis, I. (eds.) TRUST 2015. LNCS, vol. 9229, pp. 209–227. Springer, Heidelberg (2015)
21. Yang, B., Yang, K., Zhang, Z., Qin, Y., Feng, D.: AEP-M: practical anonymous e-payment for mobile devices using ARM Trust Zone and divisible e-cash (full version). ePrint (2016)
22. Zhao, S., Zhang, Q., Hu, G., Qin, Y., Feng, D.: Providing root of trust for ARM trust zone using on-chip SRAM. In: Proceedings of TrustED 2014, pp. 25–36. ACM (2014)