Practical Compact E-Cash with Arbitrary Wallet Size

Patrick Märtens

Mathematisches Institut, Justus-Liebig-Universität Gießen
patrickmaertens@gmx.de

February 4, 2015

Abstract. Compact e-cash schemes allow users to withdraw a wallet containing K coins and to spend each coin unlinkably. We present the first compact e-cash scheme with arbitrary wallet size $k \leq K$ while the spending protocol is of constant time and space complexity. Known compact e-cash schemes are constructed from either verifiable random functions or bounded accumulators. We use both building blocks to construct the new scheme which is secure under the q-SDH, the y-DDHI and the SXDH assumptions in the random oracle model.

Keywords: E-Cash, compact, constant-size, arbitrary wallet size, accumulator, pairings

1 Introduction

Electronic cash (e-cash) was introduced by Chaum [19] in 1982. Basically, an (offline) e-cash scheme consists of three parties (the bank $\mathcal{B}$, user $\mathcal{U}$ and merchant $\mathcal{M}$) and three protocols ($\mathsf{WithdrawalProtocol}$, $\mathsf{SpendProtocol}$ and $\mathsf{DepositProtocol}$). A user withdraws coins from the bank and applies the coins to pay a merchant (without involving the bank during spending). Then the merchant deposits the coins to the bank. In a *compact* e-cash scheme a user can withdraw a wallet $\mathcal{W}$ containing K coins and can spend each coin unlinkably.

From the bank's point of view, the most important security property is that no one is able to forge a valid coin or can double-spend a coin without being identified (*balance*). Further, the bank should be able to identify a double-spender without a third party. From the users point of view, the most important security properties are that honest users are anonymous (*anonymity*) and cannot be accused to have performed a double-spending (*exculpability*).

1.1 Related Work

CHL's Compact E-Cash. Camenisch, Hohenberger and Lysyanskaya proposed the first compact e-cash scheme [13]. There are two versions of the scheme, but we only describe the first one (which we title as CHL scheme) since it is simpler and much more efficient than the other.

When a user with secret key $\mathsf{sk}_{\mathcal{U}}$ withdraws a wallet, she obtains a signature σ from the bank on a committed message $(\mathsf{sk}_{\mathcal{U}}, s, t)$ for two secret random numbers s and t. Broadly speaking, the number s is used to generate a *serial number* S, which is required to detect a double-spending.

The number t is used to generate a *blinding value* B. By using this blinding value the user computes a double-spending equation T, which is required to identify a double-spender.

To spend K coins, the user has to generate K serial numbers. For $1 \leq J \leq K$, each serial number S_J is computed by the number s and a counter J. To spend the resulting coins unlinkably, a pseudorandom function $F_{(\cdot)}(\cdot)$ is used to generate this serial numbers and the corresponding blinding values. The resulting values are $S_J = F_s(J)$ and $B_J = F_t(J)$. Each double-spending equation is computed as $T_J = \mathsf{pk}_{\mathcal{U}} \cdot B_J^R$, where $\mathsf{pk}_{\mathcal{U}}$ is the user's public key and R is some hash value. For the counter J running from 1 to K, the wallet can be spent K times unlinkably.

To spend a coin for some $1 \leq J \leq K$, the user has to (1) compute S_J and T_J and has to prove in zero-knowledge (2) that S_J and T_J are correctly formed from $(\mathsf{sk}_{\mathcal{U}}, s, t, J)$, (3) that the user is in possession of a signature σ on the message $(\mathsf{sk}_{\mathcal{U}}, s, t)$ and (4) that $1 \leq J \leq K$.

In the CHL scheme, a wallet containing K coins can be stored using $O\left(\lambda + \log(K)\right)$ bits and the complexity of the withdrawal and spend operations is $O\left(\lambda + \log(K)\right)$, where λ is the security parameter. The size of the bank's public key is just $O\left(\lambda\right)$.

ASM's Compact E-Cash. Au, Susilo and Mu [4] modified the CHL scheme to construct a generic compact e-cash scheme with a more efficient spending protocol. Further, they introduced two additional protocols, namely, batch spending and compact spending. Batch spending allows to spend several coins in one single execution, but is only slightly better than repeating the protocol several times. In compact spending, the user can spend the whole wallet at once. Au, Susilo and Mu constructed two instantiations, one scheme using BBS+ signature [3] and one scheme using CL+ signature [14, 16]. We only describe how the usual spend protocol is formed from the CHL scheme using BBS+ signature (scheme 1 in [4]) and we title the resulting e-cash scheme as ASM scheme.

The ASM scheme is a modification of the CHL scheme. The essential difference is the manner of proving that the counter J is within 1 to K. In the CHL scheme, the user directly proves that J lies in the interval $[1, K]$ with a zero-knowledge proof of knowledge proposed by Boudot [12]. This proof is of complexity $O(\log(K))$ and works in groups of unknown order. In the ASM scheme, the bank signs each value from 1 to K and publishes the corresponding signatures $\sigma_1, \ldots, \sigma_K$. Instead of proving that J lies in the interval $[1, K]$, a user proves knowledge of a signature σ_J on counter J. Since the bank only publishes signatures on the values $1, \ldots, K$, proving knowledge of such a signature on J indirectly proves that $1 \leq J \leq K$. Due to this technique, the complexity of the spend operation is $O(\lambda)$ instead of $O\left(\lambda + \log(K)\right)$. The tradeoff is that the size of the bank's public key becomes $O\left(\lambda \cdot K\right)$.

ASM's Compact E-Cash with Arbitrary Wallet Size. In the same paper, the authors extended their compact e-cash scheme to support arbitrary wallet size. They modified the system in such a way that the wallet size $k \leq K$ can be chosen by the user arbitrarily during the withdrawal protocol, while coins from wallets of different size are indistinguishable during spending. The idea is that during withdrawal protocol, the value k is also signed by the bank. To spend a coin, the user additionally has to prove in zero-knowledge that the counter J lies in the interval $[1, k]$ (with the same technique as in the CHL scheme) and that k is signed by the bank, where k is hidden from the merchant. However, inefficient exact range proof [12] has to be employed (see [4]). Due to this fact, the complexity of the spend protocol becomes $O\left(\lambda + \log(k)\right)$.

1.2 Our Contribution

We first modify the ASM scheme to construct a compact e-cash scheme 1 and after that we extend this scheme to construct compact e-cash scheme 2 that provides arbitrary wallet size. In scheme 1, we change the manner of proving that the counter J is within 1 to K. We make use of an accumulator [9, 23]. Instead of publishing signatures on the values 1 to K, the bank publishes the accumulator V of the values $1, \ldots, K$. Further, the bank publishes the corresponding witness W_J for each value $1 \leq J \leq K$. To prove that the counter J is in the interval $[1, K]$, the user proves that the counter J is accumulated in V. Thus, instead of proving knowledge of a signature σ_J, the user proves knowledge of a witness W_J. The resulting scheme is as efficient as the ASM scheme (using the short BB signature [10], as proposed by Au [2]). So, the complexity of the spend operation is $O(\lambda)$ and the size of the bank's public key is $O(\lambda \cdot K)$.

To extend this compact e-cash scheme to a scheme that provides arbitrary wallet size, the bank generates K accumulators. For $1 \leq k \leq K$, each element V_k is the accumulator of the values $1, \ldots, k$. To withdraw a wallet containing k coins, the user obtains a signature σ from the bank on the committed message $(V_k, \mathsf{sk}_{\mathcal{U}}, s, t)$. To generate those signatures, we make use of the extended special signature (ESS) proposed by Au, Wu, Susilo and Mu [6]. Thus, during spending, we don't need inefficient exact range proof like in the ASM scheme to support arbitrary wallet size.

Further, we can made a tradeoff between the size of the bank's public key and the computational cost during the spending phase. If the bank only publishes the generators to compute the accumulators and the witnesses, the bank's public key still is $O(\lambda \cdot K)$ as in scheme 1. To compute an accumulator or a witness, the user has to perform one multi-base exponentiation regarding k or $k-1$ public generators, respectively. Nonetheless, a multi-base exponentiation takes a similar time as a single-base exponentiation (see [25, 7]). On the other hand, if the bank publishes all accumulators and all witnesses, the bank's public key becomes $O(\lambda \cdot K^2)$.

However, it is also possible not to support every wallet size $1, \ldots, K$, but n different wallet sizes $k_1, \ldots, k_n$ where $k_n = K$. This will reduce the size of the bank's public key in the latter case. For $n = O(1)$, the size of the bank's public key still is $O(\lambda \cdot K)$. In section 4.3, we also achieve a bank public key size of $O(\lambda \cdot K)$ where $n = O(\log(K))$.

1.3 Organization

We discuss technical preliminaries such as mathematical assumptions and cryptographic building blocks in the next section. In section 3 we define the security model. The constructions and security analyses are shown in Section 4. Finally we conclude in Section 5.

2 Preliminaries

2.1 Bilinear Maps

A bilinear map, also called pairing, maps two group elements to one group element. Let $\hat{e}$ be a bilinear map $\hat{e}\colon \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ such that the following properties hold.

- $\mathbb{G}_1$ and $\mathbb{G}_2$ are cyclic multiplicative groups of prime order p.
- Each element of $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$ has a unique binary representation.

- g_0 is a generator of $\mathbb{G}_1$ and h_0 is a generator of $\mathbb{G}_2$.
- (Bilinear:) $\forall\, g \in \mathbb{G}_1, h \in \mathbb{G}_2$ and $a, b \in \mathbb{Z}_p$, $\hat{e}\left(g^a, h^b\right) = \hat{e}\left(g, h\right)^{ab}$.
- (Non-degenerate:) $\hat{e}\left(g_0, h_0\right) \neq 1$.
- The group action in $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ and the bilinear map $\hat{e}$ are all efficiently computable.

We call $(\mathbb{G}_1, \mathbb{G}_2)$ a bilinear group pair. Let $\mathcal{G} := (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \hat{e}, g_0, h_0)$ be the global parameters of a pairing.

Galbraith, Paterson and Smart [22] separated different possible pairing instantiations into three basic types:

Type 1: $\mathbb{G}_1 = \mathbb{G}_2$.

Type 2: $\mathbb{G}_1 \neq \mathbb{G}_2$ but there is an efficiently computable homomorphism $\psi \colon \mathbb{G}_2 \to \mathbb{G}_1$.

Type 3: $\mathbb{G}_1 \neq \mathbb{G}_2$ and there are no efficiently computable homomorphisms between $\mathbb{G}_1$ and $\mathbb{G}_2$.

2.2 Mathematical Assumptions

The security of our construction depends on the following assumptions.

Definition 1 (**Decisional Diffie-Hellman**). *The Decisional Diffie-Hellman (DDH) problem in $\mathbb{G}$ is the following: On input a quadruple $\left(g, g^a, g^b, g^c\right) \in \mathbb{G}^4$, output 1 if $g^c = g^{ab}$ and 0 otherwise. We say that the DDH assumption holds in $\mathbb{G}$ if no PPT algorithm has non-negligible advantage over random guessing in solving the DDH problem in $\mathbb{G}$.*

Definition 2 (**Symmetric External Diffie-Hellman [1, 6]**). *The Symmetric External Diffie-Hellman (SXDH) assumption in $(\mathbb{G}_1, \mathbb{G}_2)$ states that the DDH assumption holds in $\mathbb{G}_1$ and $\mathbb{G}_2$. It implies that there are no efficiently computable isomorphisms between $\mathbb{G}_1$ and $\mathbb{G}_2$ (see [6]).*

Definition 3 (**q-Strong Diffie-Hellman [10, 4, 26]**). *The q-Strong Diffie-Hellman (q-SDH) problem in $(\mathbb{G}_1, \mathbb{G}_2)$ is the following: On input a $(q+3)$-tuple $(g_0, g_0^x, g_0^{x^2}, \ldots, g_0^{x^q}, h_0, h_0^x) \in \mathbb{G}_1^{q+1} \times \mathbb{G}_2^2$, output a pair $(g_0^{1/(x+c)}, c) \in \mathbb{G}_1 \times \mathbb{Z}_p^*$. We say that the q-SDH assumption holds in $(\mathbb{G}_1, \mathbb{G}_2)$ if no PPT algorithm has non-negligible advantage in solving the q-SDH problem in $(\mathbb{G}_1, \mathbb{G}_2)$.*

Definition 4 (**y-Decisional Diffie-Hellman Inversion [20, 13, 4]**). *The y-Decisional Diffie-Hellman Inversion (y-DDHI) problem in a prime order group $\mathbb{G}$ is the following: On input a $(y+2)$-tuple $(g, g^x, g^{x^2}, \ldots, g^{x^y}, g^c) \in \mathbb{G}^{y+2}$, output 1 if $g^c = g^{1/x}$ and 0 otherwise. We say that the y-DDHI assumption holds in $\mathbb{G}$ if no PPT algorithm has non-negligible advantage over random guessing in solving the y-DDHI problem in $\mathbb{G}$.*

2.3 Building Blocks

Zero-Knowledge Proofs of Knowledge. A zero-knowledge proof of knowledge (PoK) is an interactive protocol during which a prover proves to a verifier that he knows a secret that verifies a given relation. We need those proofs for proving statements related to knowledge of discrete logarithms constructed over cyclic groups of prime order p. We follow the notation by [17], for example, $PoK\left\{(a,b,c) : A = g_1^a g_2^b \ \wedge\ B = g_3^a g_4^c\right\}$ denotes a PoK such that the prover knows the secret $(a,b,c) \in \mathbb{Z}_p^3$ such that $A = g_1^a g_2^b$ and $B = g_3^a g_4^c$. Using the Fiat-Shamir heuristic [21] these proofs can be made non-interactive and are secure in the random oracle model [8]. This is referred to as a signature of knowledge (SoK). Back to the above example, the corresponding SoK is denoted as $SoK\left\{(a,b,c) : A = g_1^a g_2^b \ \wedge\ B = g_3^a g_4^c\right\}(m)$, where m is some message.

Pseudorandom Functions. An important building block of the CHL scheme and the ASM scheme is the pseudorandom function proposed by Dodis and Yampolskiy [20]. The function F is defined by a quadruple $(\mathbb{G}, p, g, s)$, where $\mathbb{G}$ is a cyclic group of prime order p, g is a generator of $\mathbb{G}$ and s is a seed in $\mathbb{Z}_p$. For any input $x \in \mathbb{Z}_p$, the function $F_{(\mathbb{G},p,g,s)}(\cdot)$ is defined as $F_{(\mathbb{G},p,g,s)}(x) = g^{1/(s+x+1)}$. Under the y-DDHI assumption, the output of this function is indistinguishable from random elements in $\mathbb{G}$.

Signature Schemes with Efficient Protocols. Camenisch and Lysyanskaya [15] presented a signature scheme with two efficient protocols for (1) issuing a signature on a committed message and for (2) proving knowledge of a message-signature pair.

Boneh, Boyen and Shacham [11] proposed a group signature scheme which can be extended to a signature scheme with efficient protocols (see [16]). Au, Susilu and Mu [3] gave a formal security proof of the extended signature, implemented with a type 2 pairing, which they named BBS+ signature. A tight security reduction for all pairing types can be found in [26]. BBS+ allows to sign a committed message $m_1, \ldots, m_L \in \mathbb{Z}_p$ and is secure (unforgeable against adaptively chosen message attack) in the standard model under the q-SDH assumption. We briefly describe the construction of BBS+, which will be used in scheme 1.

Let $(\mathbb{G}_1, \mathbb{G}_2)$ be a bilinear group pair with parameters $\mathcal{G}$ and additional generators $g_1, \ldots, g_{L+1} \in \mathbb{G}_1$. The secret signing key is $\mathsf{sk}_{\mathsf{BBS+}} = y \in \mathbb{Z}_p^*$ and the public key is $\mathsf{pk}_{\mathsf{BBS+}} = Y = h_0^y \in \mathbb{G}_2$. To sign a message $m = (m_1, \ldots, m_L) \in \mathbb{Z}_p^L$, the signer randomly chooses $e, s \in_R \mathbb{Z}_p^*$ and computes $A = \left(g_0 g_1^s g_2^{m_1} \cdots g_{L+1}^{m_L}\right)^{1/(y+e)}$. The signature on the message is $\sigma = (A, e, s)$. Everyone can verify the equation $\hat{e}\left(A, Y h_0^e\right) \stackrel{?}{=} \hat{e}\left(g_0 g_1^s g_2^{m_1} \cdots g_{L+1}^{m_L}, h_0\right)$.

The protocol for issuing a signature on a committed message is the following (see [3]): The user randomly chooses $s' \in_R \mathbb{Z}_p^*$, computes the perfectly hiding Pedersen Commitment [24] $C' = g_1^{s'} g_2^{m_1} \cdots g_{L+1}^{m_L}$ and sends C' to the signer. The user also need to prove that C' is correctly formed by the following proof of knowledge: $PoK\left\{(s', m_1, \ldots, m_L) : C' = g_1^{s'} g_2^{m_1} \cdots g_{L+1}^{m_L}\right\}$. After successful verification of the proof, the signer randomly chooses $e, s'' \in_R \mathbb{Z}_p^*$, computes $A = \left(g_0 g_1^{s''} C'\right)^{1/(y+e)}$ and returns (A, e, s''). The user computes $s = s' + s''$ to obtain $\sigma = (A, e, s)$.

The protocol for proving knowledge of a message-signature pair is the following (see [3]): The user randomly chooses $r_1, r_2 \in_R \mathbb{Z}_p^*$, computes $A_1 = g_1^{r_1} g_2^{r_2}, A_2 = A g_2^{r_1}$ and generates the following signature of knowledge:

$$
\begin{aligned}
SoK\Big\{&(r_1, r_2, \delta_1, \delta_2, e, s, m_1, \ldots, m_L) : A_1 = g_1^{r_1} g_2^{r_2} \ \wedge\ 1 = A_1^e g_1^{-\delta_1} g_2^{-\delta_2} \ \wedge\ \hat{e}\left(A_2, Y\right) \hat{e}\left(g_0, h_0\right)^{-1} \\
&= \hat{e}\left(g_1, h_0\right)^s \hat{e}\left(g_2, h_0\right)^{m_1 + \delta_1} \hat{e}\left(g_3, h_0\right)^{m_2} \cdots \hat{e}\left(g_{L+1}, h_0\right)^{m_L} \hat{e}\left(g_2, Y\right)^{r_1} \hat{e}\left(A_2, h_0\right)^{-e}\Big\}(R)
\end{aligned}
$$

where $\delta_1 = r_1 e, \delta_2 = r_2 e$ and R is some message.

ESS is an extension of BBS+, proposed in [6] and extended to ESS+ in [5]. ESS allows signing a committed message $m_1, \ldots, m_L \in \mathbb{Z}_p$ together with a group element $M \in \mathbb{G}_1$ and is secure (unforgeable against adaptively chosen message attack) in the generic group model under the SXDH assumption. So, ESS is not secure when it is implemented with type 1 or type 2 pairing. We briefly describe the construction of ESS, which will be used in scheme 2.

Again, let $(\mathbb{G}_1, \mathbb{G}_2)$ be a bilinear group pair with parameters $\mathcal{G}$ and additional generators $g_1, \ldots, g_{L+1} \in \mathbb{G}_1$ such that there are no efficient isomorphisms between $\mathbb{G}_1$ and $\mathbb{G}_2$. The secret signing key is $\mathsf{sk}_{\mathsf{ESS}} = (X, y) \in \mathbb{G}_1 \times \mathbb{Z}_p^*$ and the public key is $\mathsf{pk}_{\mathsf{ESS}} = (Z, Y) = (\hat{e}(X, h_0), h_0^y) \in \mathbb{G}_T \times \mathbb{G}_2$. To sign a message $m = (M, m_1, \ldots, m_L) \in \mathbb{G}_1 \times \mathbb{Z}_p^L$, the signer randomly chooses $e, s \in_R \mathbb{Z}_p^*$ and computes $A = XM^e, B = \left(g_0 g_1^s g_2^{m_1} \cdots g_{L+1}^{m_L}\right)^{1/(y+e)}$ and $C = h_0^e$. The signature on the message is $\sigma = (A, B, C, s)$. Everyone can verify the equations $\hat{e}(A, h_0) \stackrel{?}{=} Z\hat{e}(M, C)$ and $\hat{e}(B, CY) \stackrel{?}{=} \hat{e}\left(g_0 g_1^s g_2^{m_1} \cdots g_{L+1}^{m_L}, h_0\right)$.

The protocol for issuing a signature on a committed message is the following (see [6]): The user randomly chooses $s' \in_R \mathbb{Z}_p^*$, computes the perfectly hiding Pedersen Commitment [24] $C' = g_1^{s'} g_2^{m_1} \cdots g_{L+1}^{m_L}$ and sends M, C' to the signer. The user also need to prove that C' is correctly formed by the following proof of knowledge: $PoK\left\{(s', m_1, \ldots, m_L) : C' = g_1^{s'} g_2^{m_1} \cdots g_{L+1}^{m_L}\right\}$. After successful verification of the proof, the signer randomly chooses $e, s'' \in_R \mathbb{Z}_p^*$, computes $A = XM^e, B = \left(g_0 g_1^{s''} C'\right)^{1/(y+e)}, C = h_0^e$ and returns (A, B, C, s''). The user computes $s = s' + s''$ to obtain $\sigma = (A, B, C, s)$.

The protocol for proving knowledge of a message-signature pair is the following (see [6]): Let $h_1 \in_R \mathbb{G}_2$ be an additional generator of $\mathbb{G}_2$. The user randomly chooses $r_1, r_2, r_3, r_4, r_5 \in_R \mathbb{Z}_p^*$, computes $A_1 = Ag_1^{r_1}, A_2 = Bg_2^{r_2}, A_3 = Ch_1^{r_3}, A_4 = Mg_3^{r_4}, A_5 = g_1^{r_3} g_2^{r_5}$ and generates the following signature of knowledge:

$$
\begin{aligned}
SoK\Big\{&(r_1, r_2, r_3, r_4, r_5, \delta_{2,3}, \delta_{2,5}, \delta_{3,4}, \delta_{4,5}, s, m_1, \ldots, m_L) : \\
&A_5 = g_1^{r_3} g_2^{r_5} \ \wedge\ 1 = A_5^{r_2} g_1^{-\delta_{2,3}} g_2^{-\delta_{2,5}} \ \wedge\ 1 = A_5^{r_4} g_1^{-\delta_{3,4}} g_2^{-\delta_{4,5}} \ \wedge \\
&Z\hat{e}(A_4, A_3)\, \hat{e}(A_1, h_0)^{-1} = \hat{e}(A_4, h_1)^{r_3}\, \hat{e}(g_3, A_3)^{r_4}\, \hat{e}(g_1, h_0)^{-r_1}\, \hat{e}(g_3, h_1)^{-\delta_{3,4}} \ \wedge \\
&\hat{e}(A_2, A_3 Y)\, \hat{e}(g_0, h_0)^{-1} = \hat{e}(g_1, h_0)^{s}\, \hat{e}(g_2, h_0)^{m_1} \cdots \hat{e}(g_{L+1}, h_0)^{m_L} \\
&\qquad \hat{e}(A_2, h_1)^{r_3}\, \hat{e}(g_2, A_3 Y)^{r_2}\, \hat{e}(g_2, h_1)^{-\delta_{2,3}}\Big\}(R)
\end{aligned}
$$

where $\delta_{i,j} = r_i r_j$ and R is some message.

(Bounded) Accumulators. An accumulator scheme is a method for accumulating several values into one element, called the accumulator. The notion, bounded accumulator, was introduced in [6] as an accumulator with a limit q of values that can be accumulated. Au *et al.* [6] observed that the construction due to Nguyen [23], whose security relies on the q-SDH assumption, is a bounded accumulator. However, this observation is unproved and our schemes won't rely on any boundedness property. We briefly describe Nguyen's construction.

Let $(\mathbb{G}_1, \mathbb{G}_2)$ be a bilinear group pair with parameters $\mathcal{G}$. The generation algorithm randomly selects $\alpha \in_R \mathbb{Z}_p^*$ and computes the elements $h_\alpha := h_0^\alpha$ and $g_0^{\alpha^i}$ for $1 \leq i \leq q$. The public parameters of the accumulator scheme are $\left(\mathcal{G}, g_0^\alpha, g_0^{\alpha^2}, \ldots, g_0^{\alpha^q}, h_\alpha\right)$. The accumulator of a set of values $x_1, \ldots, x_q \in \mathbb{Z}_p \setminus \{-\alpha\}$ is computed as $V = g_0^{\prod_{j=1}^{q}(\alpha + x_j)}$, what does not require knowledge of the secret α since the elements $g_0, g_0^{\alpha}, \ldots, g_0^{\alpha^q}$ are publicly known. A witness W_i, to prove that the value x_i is accumulated in V, is computed as $W_i = g_0^{\prod_{j=1, j \neq i}^{q}(\alpha + x_j)}$. Thus, the witness-value pair (W_i, x_i) satisfies the equation $\hat{e}(V, h_0) = \hat{e}(W_i, h_\alpha h_0^{x_i})$.

Under the q-SDH assumption, the described accumulator fulfills the security property *collision-resistance*. That is, it is infeasible on input $\left(\mathcal{G}, g_0^{\alpha}, g_0^{\alpha^2}, \ldots, g_0^{\alpha^q}, h_\alpha\right)$ to output $(x_1, \ldots, x_{q'}, W, x)$ where $q' \leq q$, such that $x \notin \{x_1, \ldots, x_{q'}\}$ and $\hat{e}\left(g_0^{\prod_{j=1}^{q'}(\alpha + x_j)}, h_0\right) = \hat{e}(W, h_\alpha h_0^{x})$. Broadly speaking, it is infeasible to compute a valid witness for a value which has not been accumulated.

The following proofs of knowledge have been proposed in [6]: Let $g_1, g_2 \in_R \mathbb{G}_1$ be additional generators of $\mathbb{G}_1$. To prove that a hidden value x_i is accumulated in an accumulator V, the user randomly chooses $r_1, r_2 \in_R \mathbb{Z}_p^*$, computes $A_1 = g_1^{r_1} g_2^{r_2}, A_2 = W_i g_2^{r_1}$ and generates the following proof of knowledge:

$$PoK\Big\{(r_1, r_2, \delta_1, \delta_2, x_i) : A_1 = g_1^{r_1} g_2^{r_2} \ \wedge \ 1 = A_1^{x_i} g_1^{-\delta_1} g_2^{-\delta_2} \ \wedge \\ \hat{e}(V, h_0)\, \hat{e}(A_2, h_\alpha)^{-1} = (A_2, h_0)^{x_i}\, \hat{e}(g_2, h_0)^{-\delta_1}\, \hat{e}(g_2, h_\alpha)^{-r_1}\Big\}$$

where $\delta_1 = r_1 x_i$ and $\delta_2 = r_2 x_i$. This proof will be used in scheme 1.

To prove that a hidden value x_i is accumulated in a hidden accumulator V, the user randomly chooses $r_1, r_2, r_3 \in_R \mathbb{Z}_p^*$, computes $A_1 = g_1^{r_1} g_2^{r_2}, A_2 = W_i g_2^{r_1}, A_3 = V g_1^{r_3}$ and generates the following proof of knowledge:

$$PoK\Big\{(r_1, r_2, r_3, \delta_1, \delta_2, x_i) : A_1 = g_1^{r_1} g_2^{r_2} \ \wedge \ 1 = A_1^{x_i} g_1^{-\delta_1} g_2^{-\delta_2} \ \wedge \\ \hat{e}(A_3, h_0)\, \hat{e}(A_2, h_\alpha)^{-1} = (A_2, h_0)^{x_i}\, \hat{e}(g_2, h_0)^{-\delta_1}\, \hat{e}(g_2, h_\alpha)^{-r_1}\, \hat{e}(g_1, h_0)^{r_3}\Big\}$$

where $\delta_1 = r_1 x_i$ and $\delta_2 = r_2 x_i$. This proof will be used in scheme 2.

3 Syntax

Let K be the maximum size of a wallet and let λ be the security parameter. A compact e-cash scheme can be defined by the following polynomial time algorithms and protocols between the bank $\mathcal{B}$, the user $\mathcal{U}$ and the merchant $\mathcal{M}$:

- $\mathsf{BankSetup}(1^\lambda, K)$ is a probabilistic algorithm that outputs a key pair $(\mathsf{pk}_\mathcal{B}, \mathsf{sk}_\mathcal{B})$ for $\mathcal{B}$. Further, an empty database $\mathcal{D}$ is set up. In the following, 1^λ and K are implicitly in the input of all algorithms and protocols.

- $\mathsf{UserSetup}(\mathsf{pk}_\mathcal{B})$ is a probabilistic algorithm that outputs a key pair $(\mathsf{pk}_\mathcal{U}, \mathsf{sk}_\mathcal{U})$ for $\mathcal{U}$. A merchant $\mathcal{M}$ executes the same algorithm to get $(\mathsf{pk}_\mathcal{M}, \mathsf{sk}_\mathcal{M})$.

- $\mathsf{WithdrawalProtocol}\left(\mathcal{U}(\mathsf{pk}_\mathcal{B}, \mathsf{sk}_\mathcal{U}, k), \mathcal{B}(\mathsf{pk}_\mathcal{B}, \mathsf{sk}_\mathcal{B}, k)\right)$ is a protocol where $\mathcal{U}$ withdraws a wallet $\mathcal{W}$ containing $k \leq K$ coins. $\mathcal{B}$ outputs its protocol view view.

- $\mathsf{SpendProtocol}\left(\mathcal{U}(\mathsf{pk}_\mathcal{B}, \mathsf{pk}_\mathcal{M}, \mathsf{sk}_\mathcal{U}, \mathcal{W}), \mathcal{M}(\mathsf{pk}_\mathcal{B}, \mathsf{sk}_\mathcal{M})\right)$ is a protocol where $\mathcal{U}$ spends one coin coin from the wallet $\mathcal{W}$. $\mathcal{U}$ outputs an updated wallet $\mathcal{W}'$ and $\mathcal{M}$ outputs the coin coin.

- $\mathsf{DepositProtocol}\left(\mathcal{M}\left(\mathsf{pk}_{\mathcal{B}}, \mathsf{sk}_{\mathcal{M}}, \mathsf{coin}\right), \mathcal{B}\left(\mathsf{sk}_{\mathcal{B}}, \mathsf{pk}_{\mathcal{M}}\right)\right)$ is a protocol where $\mathcal{M}$ deposits a coin coin. If it concerns a double-spending, $\mathcal{B}$ executes the $\mathsf{RevokeDoubleSpender}$ algorithm. Otherwise, $\mathcal{B}$ outputs 1 and stores the coin coin in the database $\mathcal{D}$.

- $\mathsf{RevokeDoubleSpender}(\mathsf{coin}, \mathsf{coin}')$ is a deterministic algorithm that outputs the public key $\mathsf{pk}_{\mathcal{U}}$ of a fraudulent user or merchant.

- $\mathsf{VerifyGuilt}(\mathsf{pk}_{\mathcal{B}}, \mathsf{pk}_{\mathcal{U}}, \mathsf{coin}, \mathsf{coin}')$ is a deterministic algorithm that outputs 1 if the two coins are valid and the output of $\mathsf{RevokeDoubleSpender}(\mathsf{coin}, \mathsf{coin}')$ is $\mathsf{pk}_{\mathcal{U}}$, and 0 otherwise.

For correctness, we require that whenever an honest user obtains a wallet $\mathcal{W}$ from $\mathcal{B}$, an honest merchant shall accept the coin coin from $\mathcal{W}$. Whenever an honest merchant obtains a coin coin, the deposit of coin will be accepted by the honest bank.

3.1 Security Definitions

We first informally describe the security properties of a compact e-cash system.

- *Balance* guarantees that no collusion of users and merchants can deposit more coins than they have withdrawn from the bank. We require that a collusion of users and merchants, having run the withdrawal protocol for n times, cannot deposit more than nk coins. If they deposit $nk + 1$ coins, at least one of the colluder must be identified.

- *Anonymity* guarantees that no collusion of users, merchants and the bank can link a spent coin to its owner or to other coins from the same wallet.

- *Exculpability* guarantees that no collusion of users, merchants and the bank can falsely accuse an honest user from having double spent a coin.

For a formal definition of the security, we use a game-based approach. Since our scheme is a modified version of the ASM scheme, we apply the same security definitions. However, the ASM scheme also fulfills the security definition given by Au [2]. The adversary's capabilities are modeled by arbitrary and adaptive queries to the following oracles:

- *Withdrawal Oracle.* The adversary $\mathcal{A}$ presents a public key $\mathsf{pk}_{\mathcal{A}}$ and engages in the $\mathsf{WithdrawalProtocol}$ as user to obtain a wallet. The oracle stores $\mathsf{pk}_{\mathcal{A}}$ in a set $\mathcal{U}_{\mathcal{A}}$.

- *Spend Oracle.* The adversary $\mathcal{A}$ presents a public key $\mathsf{pk}_{\mathcal{U}} \notin \mathcal{U}_{\mathcal{A}}$ and engages in the $\mathsf{SpendProtocol}$ as merchant to request the user $\mathsf{pk}_{\mathcal{U}}$ to spend a coin.

- *Hash Oracle.* The adversary $\mathcal{A}$ can ask for the output of the hash functions for any input.

Definition 5 (Game Balance)

- (Initialization Phase.) *The challenger $\mathcal{C}$ takes a sufficiently large security parameter λ and runs* $\mathsf{BankSetup}$ *to generate the key pair* $(\mathsf{pk}_{\mathcal{B}}, \mathsf{sk}_{\mathcal{B}})$. *$\mathcal{C}$ sends $\mathsf{pk}_{\mathcal{B}}$ to $\mathcal{A}$.*

- (Probing Phase.) *The adversary $\mathcal{A}$ can perform a polynomially bounded number of queries to the oracles in an adaptive manner.*

- (End Game Phase.) *Let q_W be the number of queries to the Withdrawal Oracle where k_i is the wallet size of the i-th query and let q_S be the number of queries to the Spend Oracle. The adversary $\mathcal{A}$ wins the game if it can deposit at least $\sum_{i=1}^{q_W} k_i + q_S + 1$ coins such that* $\mathsf{RevokeDoubleSpender}$ *does not output any* $\mathsf{pk}_{\mathcal{A}} \in \mathcal{U}_{\mathcal{A}}$.

The advantage of $\mathcal{A}$ is defined as the probability that $\mathcal{A}$ wins.

Definition 6 (Game Anonymity)

- (Initialization Phase.) *The challenger $\mathcal{C}$ takes a sufficiently large security parameter λ and sends λ to $\mathcal{A}$. The adversary generates a key pair $(\mathsf{pk}_{\mathcal{B}}, \mathsf{sk}_{\mathcal{B}})$ and returns $\mathsf{pk}_{\mathcal{B}}$. Since $\mathcal{A}$ is in possession of the secret key $\mathsf{sk}_{\mathcal{B}}$, only Hash Oracle query is allowed.*

- (Challenge Phase.) *The challenger $\mathcal{C}$ chooses two public keys pk_0 and pk_1 and presents them to $\mathcal{A}$. The challenger $\mathcal{C}$ runs the* $\mathsf{WithdrawalProtocol}$ *with $\mathcal{A}$ acting as the bank to obtain several wallets $\mathcal{W}_{0,1}, \ldots, \mathcal{W}_{0,n}$ and $\mathcal{W}_{1,1}, \ldots, \mathcal{W}_{1,m}$ on behalf of the two public keys, where n and m are specified by $\mathcal{A}$. Then $\mathcal{C}$ runs the* $\mathsf{SpendProtocol}$ *with $\mathcal{A}$ acting as a merchant and asking for spending from $\mathcal{C}$, where $\mathcal{A}$ is allowed to specify which wallet $\mathcal{C}$ uses, with the restriction that it cannot ask $\mathcal{C}$ to over-spend any of the wallets. Finally, $\mathcal{A}$ chooses one wallet $\mathcal{W}_0 \in \{\mathcal{W}_{0,1}, \ldots, \mathcal{W}_{0,n}\}$ and one wallet $\mathcal{W}_1 \in \{\mathcal{W}_{1,1}, \ldots, \mathcal{W}_{1,m}\}$ from the set of wallets that contain at least one unspent coin. $\mathcal{C}$ then randomly chooses a bit $b \in_R \{0, 1\}$ and uses wallet $\mathcal{W}_b$ for the challenge spending.*

- (End Game Phase.) *The adversary $\mathcal{A}$ outputs a bit b' and wins the game if $b' = b$.*

The advantage of $\mathcal{A}$ is defined as the probability that $\mathcal{A}$ wins minus $1/2$.

Definition 7 (Game Exculpability)

- (Initialization Phase.) *The challenger $\mathcal{C}$ takes a sufficiently large security parameter λ and sends λ to $\mathcal{A}$. The adversary generates a key pair $(\mathsf{pk}_{\mathcal{B}}, \mathsf{sk}_{\mathcal{B}})$ and returns $\mathsf{pk}_{\mathcal{B}}$. Since $\mathcal{A}$ is in possession of the secret key $\mathsf{sk}_{\mathcal{B}}$, only Hash Oracle query is allowed.*

- (Challenge Phase.) *The challenger $\mathcal{C}$ runs the* $\mathsf{WithdrawalProtocol}$ *for q times with $\mathcal{A}$ acting as the bank to obtain several wallets $\mathcal{W}_1, \ldots, \mathcal{W}_q$. Then $\mathcal{C}$ runs the* $\mathsf{SpendProtocol}$ *with $\mathcal{A}$ acting as a merchant and asking for spending from $\mathcal{C}$, where $\mathcal{A}$ is allowed to specify which wallet $\mathcal{C}$ uses, with the restriction that it cannot ask $\mathcal{C}$ to over-spend any of the wallets. $\mathcal{A}$ can also ask to corrupt any of the user in the above* $\mathsf{WithdrawalProtocol}$. *A corrupted user needs to surrender its private key as well as the wallet to $\mathcal{A}$.*

- (End Game Phase.) *The adversary $\mathcal{A}$ runs two deposit protocols with $\mathcal{C}$ and wins the game if* $\mathsf{RevokeDoubleSpender}$ *on this two deposit protocols points to a user in* any *of the* $\mathsf{WithdrawalProtocol}$ *and that user has not been corrupted.*

The advantage of $\mathcal{A}$ is defined as the probability that $\mathcal{A}$ wins.

Definition 8. *A compact e-cash scheme is* secure *if no PPT adversary $\mathcal{A}$ can win in Game Balance, Game Anonymity and Game Exculpability with non-negligible advantage.*

4 Our Constructions

4.1 Compact E-Cash Scheme 1

Since our scheme is a modification of the ASM scheme, there are only a few differences. The crucial difference is the manner of proving that the counter J is within 1 to K. So, the bank's public key doesn't contain signatures $\sigma_1, \dots, \sigma_J$ on the values $1, \dots, K$, but an accumulator V and witnesses $W_1, \dots, W_K$. UserSetup and WithdrawalProtocol are the same as in the ASM scheme. During SpendProtocol, a user proves knowledge of W_J rather then proving knowledge of σ_J. To explain our modification, we only describe the normal SpendProtocol. Batch spending and compact spending can be done analogously. DepositProtocol, RevokeDoubleSpender and VerifyGuilt are the same as in the ASM scheme.

Since the ASM scheme applies the BBS+ [3] signature scheme, which security has been proven using an efficiently computable isomorphism $\psi\colon \mathbb{G}_2 \to \mathbb{G}_1$, it uses a type 2 pairing. However, BBS+ signature implemented with a type 3 pairing is also secure (see [18, 26]). So, the parameters of the ASM scheme can be easily set to type 3 and we don't need the isomorphism ψ.

BankSetup. Let λ be the security parameter. Let $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ be cyclic groups of order p such that p is a λ-bit prime and such that $\hat{e}\colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a bilinear map. Let $\mathbb{G}_p$ be an additional cyclic group of order p where the y-DDHI assumption holds and let $H\colon \{0,1\}^* \to \mathbb{Z}_p^*$ be a secure cryptographic hash function. Let $g_0, g_1, g_2, g_3, g_4, g_5$ be random generators of $\mathbb{G}_1$, h_0 be a random generator of $\mathbb{G}_2$ and u_0, u_1 be random generators of $\mathbb{G}_p$ such that related discrete logarithm of the generators are unknown. This can be done by setting these generators to be output by a hash function of some publicly known seed (see [4]).

The bank generates a BBS+ key pair $(\mathsf{pk}_{\mathsf{BBS+}}, \mathsf{sk}_{\mathsf{BBS+}}) = (Y, y)$. Next, the bank randomly chooses $\alpha \in_R \mathbb{Z}_p^*$, computes $h_\alpha = h_0^\alpha, V = g_0^{\prod_{j=1}^{K}(\alpha+j)}$ and $W_J = g_0^{\prod_{j=1 j\neq J}^{K}(\alpha+j)}$ for $1 \le J \le K$. V is the accumulator of the values $1, \dots, K$ and W_J is the witness for the value J. The bank publishes $\mathsf{pk}_{\mathcal{B}} = (g_0, g_1, g_2, g_3, g_4, g_5, V, W_1, \dots, W_K, h_0, h_\alpha, Y, u_0, u_1)$ and saves the secret key $\mathsf{sk}_{\mathcal{B}} = y$. α can be safely deleted (see [6]). Let be $E_i := \hat{e}(g_i, h_0)$ for $i = 0, \dots, 5$, $E_V := \hat{e}(V, h_0), E_Y := \hat{e}(g_2, Y)$ and $E_\alpha := \hat{e}(g_2, h_\alpha)$. For efficiency consideration, these elements can also be published as part of the public key.

Further, $\mathcal{B}$ sets up a database $\mathcal{D}$.

UserSetup. To join the system, the user randomly selects a secret $x \in_R \mathbb{Z}_p^*$ and computes the public key $\mathsf{pk}_{\mathcal{U}} = u_0^x$. $\mathcal{U}$ sends $\mathsf{pk}_{\mathcal{U}}$ together with a proof of correctness to $\mathcal{B}$. The bank stores $\mathsf{pk}_{\mathcal{U}}$ as the identity of $\mathcal{U}$ in its database and the user stores the key pair $(\mathsf{pk}_{\mathcal{U}}, \mathsf{sk}_{\mathcal{U}}) = (u_0^x, x)$.

WithdrawalProtocol. The user randomly chooses $s', t, y, r \in_R \mathbb{Z}_p^*$, computes the commitment $C' = g_1^{s'} g_2^t g_3^x g_4^y g_5^r$ and sends C' to $\mathcal{B}$. Then, the user generates the following proof of knowledge:

$$\Pi_0 = PoK\left\{(s', t, x, y, r) : C' = g_1^{s'} g_2^t g_3^x g_4^y g_5^r \;\wedge\; \mathsf{pk}_{\mathcal{U}} = u_0^x\right\}.$$

The bank verifies the proof, randomly chooses $e, s'' \in_R \mathbb{Z}_p^*$, computes $A = \left(g_0 g_1^{s''} C'\right)^{1/(y+e)}$ and sends (A, e, s'') to $\mathcal{U}$.

The user computes $s = s' + s''$ and checks if $\hat{e}\left(A, Yh_0^e\right) = \hat{e}\left(g_0 g_1^s g_2^t g_3^x g_4^y g_5^r, h_0\right)$. Then, $\mathcal{U}$ stores the wallet $\mathcal{W} = (A, e, s, t, x, y, r, J)$ and sets counter $J = 1$.

SpendProtocol. Let the wallet be $\mathcal{W} = (A, e, s, t, x, y, r, J)$ such that $J \leq K$. First, the user and the merchant with public key $\mathsf{pk}_{\mathcal{M}}$ agree on transaction information info. Then, both parties compute the hash value $R = H\left(\mathsf{pk}_{\mathcal{M}}||\mathsf{info}\right)$.

The user computes $S = u_1^{1/(s+J+1)}$ and $T = \mathsf{pk}_{\mathcal{U}} u_1^{1/(t+J+1)}$. The user also computes $A_1 = g_1^{r_1} g_2^{r_2}, A_2 = A g_2^{r_1}, A_3 = g_1^J g_2^t g_3^{r_3}, A_4 = g_1^{r_4} g_2^{r_5}, A_5 = W_J g_2^{r_4}$ for $r_1, r_2, r_3, r_4, r_5 \in_R \mathbb{Z}_p^*$. Then, $\mathcal{U}$ generates the following signature of knowledge:

$$\begin{aligned}\Pi_1 = SoK\Big\{&(r_1, r_2, r_3, r_4, r_5, \delta_1, \delta_2, \delta_3, \delta_4, \delta_5, \delta_J, \delta_t, e, s, t, x, y, r, J) : \\ &A_1 = g_1^{r_1} g_2^{r_2} \ \wedge \ 1 = A_1^e g_1^{-\delta_1} g_2^{-\delta_2} \ \wedge \\ &\hat{e}\left(A_2, Y\right) E_0^{-1} = E_1^s E_2^{t+\delta_1} E_3^x E_4^y E_5^r E_Y^{r_1} \hat{e}\left(A_2, h_0\right)^{-e} \ \wedge \\ &u_1 S^{-1} = S^{J+s} \ \wedge \ A_3 = g_1^J g_2^t g_3^{r_3} \ \wedge \ 1 = A_3^x g_1^{-\delta_J} g_2^{-\delta_t} g_3^{-\delta_3} \ \wedge \\ &u_1^R T^{-1} = T^{J+t} u_0^{-\delta_J - \delta_t - x} \ \wedge \ A_4 = g_1^{r_4} g_2^{r_5} \ \wedge \ 1 = A_4^J g_1^{-\delta_4} g_2^{-\delta_5} \ \wedge \\ &E_V \hat{e}\left(A_2, h_\alpha\right)^{-1} = \hat{e}\left(A_5, h_0\right)^J E_2^{-\delta_4} E_\alpha^{-r_4} \Big\}(R)\end{aligned}$$

where $\delta_1 = r_1 e, \delta_2 = r_2 e, \delta_3 = r_3 x, \delta_4 = r_4 J, \delta_5 = r_5 J, \delta_J = Jx, \delta_t = tx$.

The user sends $S, T, A_1, A_2, A_3, A_4, A_5$ along with Π_1 to the merchant. The merchant verifies Π_1. If the proof is valid, the merchant accepts the payment and stores the coin $\mathsf{coin} = (S, T, R, A_1, A_2, A_3, A_4, A_5, \Pi_1, \mathsf{info})$.

Finally, the user increases the counter J of his wallet by 1.

DepositProtocol. The merchant sends a coin $\mathsf{coin} = (S, T, R, A_1, A_2, A_3, A_4, A_5, \Pi_1, \mathsf{info})$ to the bank. The bank checks $R \overset{?}{=} H\left(\mathsf{pk}_{\mathcal{M}}||\mathsf{info}\right)$ to verify that $\mathsf{pk}_{\mathcal{M}}$ is the real merchant. After verification of Π_1, the bank checks if the serial number S is already in its database $\mathcal{D}$. If so, the bank runs the identification algorithm $\mathsf{RevokeDoubleSpender}$ to identify the double-spender. Else, $\mathcal{B}$ stores coin in $\mathcal{D}$.

RevokeDoubleSpender. Let $\mathsf{coin} = (S, T, R, A_1, A_2, A_3, A_4, A_5, \Pi_1, \mathsf{info})$ and $\mathsf{coin}' = (S, T', R', A_1', A_2', A_3', A_4', A_5', \Pi_1', \mathsf{info}')$ be two coins where a double spending occurred and let $\mathsf{pk}_{\mathcal{M}}$ be the merchant who deposited the coin coin. The bank checks if $R \overset{?}{=} R'$. If so, the merchant $\mathsf{pk}_{\mathcal{M}}$ tried to deposit the same coin twice since the hash function H is collision-resistant. Else, $\mathcal{B}$ computes the public key of the double-spender as follows: $\mathsf{pk}_{\mathcal{U}} = \left(T^{R'}/T'^{R}\right)^{1/(R'-R)}$.

VerifyGuilt. The bank publishes the two double-spent coins and the identity $\mathsf{pk}_{\mathcal{U}}$ of the double-spender. Everyone can verify the two coins and execute $\mathsf{RevokeDoubleSpender}$ to check if $\mathsf{pk}_{\mathcal{U}}$ is the double-spender.

4.1.1 Security Analysis

We have the following theorem for scheme 1.

Theorem 1. *Compact e-cash scheme 1 is secure under the q-SDH assumption in the random oracle model, if the ASM scheme is secure.*

Proof. The security of scheme 1 directly follows from the security of the ASM scheme and the collision-resistance of the accumulator scheme.

Balance: Let $\mathcal{A}$ be a PPT adversary that makes q_W withdrawal queries and q_S spend queries. We show that the success probability of $\mathcal{A}$ is negligible under the q-SDH assumption in the random oracle model, if the ASM scheme [4] is secure.

We construct a simulator $\mathcal{S}$ which acts as challenger $\mathcal{C}$.

$\mathcal{S}$ simulates the oracle queries as in [4]: For each withdrawal query, $\mathcal{A}$ sends $\mathsf{pk}_{\mathcal{U}}, C'$ and proves the knowledge of representation of $\mathsf{pk}_{\mathcal{U}}$ and C' in Π_0. $\mathcal{S}$ acts as if an honest bank would, except during the proof of knowledge where $\mathcal{S}$ runs a rewind simulation and extracts (s', t, x, y, r). After each execution, $\mathcal{S}$ computes $\mathcal{W} = (S_1, \ldots, S_K, s, t, x, y, r)$ where $S_J = u_0^{1/(s+J+1)}$ for $1 \leq J \leq K$. Let $\mathbb{S}_W = (S_{i,J} | 1 \leq i \leq q_W, 1 \leq J \leq K)$ be the set of all serial numbers after q_W executions of the withdrawal protocol.

For each spend query, $\mathcal{A}$ presents a public key $\mathsf{pk}_{\mathcal{U}}$, a transaction information info, a merchant public key $\mathsf{pk}_{\mathcal{M}}$ and request a legal number of spend protocol. For each query, $\mathcal{S}$ randomly chooses $s, t \in_R \mathbb{Z}_p^*, J \in_R [1, K]$ and sends $S = u_1^{1/(s+J+1)}, T = \mathsf{pk}_{\mathcal{U}} u_1^{R/(t+J+1)}$ such that $R = H(\mathsf{pk}_{\mathcal{M}} || \mathsf{info})$. Further, $\mathcal{S}$ randomly chooses $A_1, A_2, A_3, A_4, A_5 \in_R \mathbb{G}_1$. Thus, $A_1, \ldots, A_5$ are perfectly simulated. $\mathcal{S}$ then generates a simulated signature of knowledge Π_1. Let $\mathbb{S}_S = (S_j | 1 \leq j \leq q_S)$ be the set of all serial numbers after q_S executions of the spend protocol.

Finally, $\mathcal{A}$ runs $\sum_{i=1}^{q_W} k_i + q_S + 1 = Kq_W + q_S + 1$ deposit protocols with $\mathcal{S}$.

As in [4], $\mathcal{A}$ wins the game either if (1) all $Kq_W + q_S + 1$ serial numbers are unique or (2) some of the serial numbers are duplicated but the $\mathsf{RevokeDoubleSpender}$ algorithm does not output any public key $\mathsf{pk}_{\mathcal{A}} \in \mathcal{U}_{\mathcal{A}}$ of a corrupted user. We analyze these two cases separately.

Case (1): Since only q_S serial numbers are given to $\mathcal{A}$ during the spend queries, $\mathcal{A}$ must have produce another $Kq_W + 1$ serial numbers. Thus, $\mathcal{A}$ can only win in case (1) by convincing $\mathcal{S}$ to accept a serial number $S \notin \mathbb{S}_W \cup \mathbb{S}_S$. As in [4], $\mathcal{A}$ must have generated a *false* proof as part of the signature of knowledge such that one of the following is fake:

1. Possession of BBS+ signature $\sigma = (A, e, s)$ on message $m = (t, x, y, r)$.
2. $S = u_1^{1/(s+J+1)}$.
3. Possession of witness W_J for counter J and accumulator V (instead: possession of BBS+ signature σ_J on counter J).

The first two cases are the same as in ASM scheme and happens with negligible probability under the q-SDH assumption and the discrete logarithm assumption.

Now we have to be intent on the last case.

Assume, $\mathcal{S}$ extracts a valid witness-value pair (W_J, J) such that $J \notin \{1, \ldots, K\}$. Then, $\mathcal{S}$ can output $(1, \ldots, K, W_J, J)$ such that $J \notin \{1, \ldots, K\}$ and $\hat{e}\left(g_0^{\prod_{j=1}^{K}(\alpha+j)}, h_0\right) = \hat{e}\left(W_J, h_\alpha h_0^J\right)$. Thus, $\mathcal{S}$ breaks the collision-resistance of the Nguyen accumulator scheme. This is negligible under the q-SDH assumption (see [23]). Thus, $\mathcal{A}$'s success probability is negligible in case (1).

As in [4], case (2) is negligible. Thus, $\mathcal{A}$'s total success probability is negligible.

Anonymity: Anonymity directly follows from the ASM scheme which ensures anonymity under the y-DDHI assumption (see [4]). The only difference of a coin is the computation of A_5. Instead of computing $A_5 = B_J g_2^{r_4}$ where B_J is part of the public signature on counter J, the user computes $A_5 = W_J g_2^{r_4}$ where W_J is the witness for counter J. Thus, the counter J is still perfectly hidden. Consequently, the information leaked about a user or wallet during $\mathsf{SpendProtocol}$ are the same as in the ASM scheme which fulfills anonymity.

Exculpability: Since the algorithms $\mathsf{RevokeDoubleSpender}$ and $\mathsf{VerifyGuilt}$ are the same as in the ASM scheme, exculpability directly follows from [4]. $\square$

4.2 Compact E-Cash Scheme 2

Now we extend scheme 1 to scheme 2 that provides arbitrary wallet size $k \leq K$. In distinction from scheme 1, we need K accumulators where for $1 \leq k \leq K$ each element V_k is the accumulator of the values $1, \ldots, k$. Let $W_{k,J}, 1 \leq k \leq K, 1 \leq J \leq k$ be the witness for counter J and accumulator V_k. To spend a coin, the user has to prove knowledge of $W_{k,J}$ without revealing k or J. Unlike scheme 1, it is not sufficient that the bank just publishes V_k and $W_{k,J}$ for $1 \leq k \leq K$ and $1 \leq J \leq k$. This is because a user always can prove knowledge of $W_{K,J}$ for all $1 \leq J \leq K$, independent of the wallet size. So, to withdraw a wallet containing $k \leq K$ coins, the user obtains a signature σ from the bank on the accumulator V_k and (t, x, y, r). We use ESS to generate these signatures. During spending, a user has to prove knowledge of $W_{K,J}$ and knowledge of a signature on the corresponding accumulator V_k.

Since we make use of ESS which is only secure under the SXDH assumption, we have to implement our scheme with a type 3 pairing.

BankSetup. Let λ be the security parameter. Let $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ be cyclic groups of order p such that p is a λ-bit prime and such that $\hat{e}\colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a bilinear map and there are no efficient isomorphisms between $\mathbb{G}_1$ and $\mathbb{G}_2$. Let $\mathbb{G}_p$ be an additional cyclic group of order p where the y-DDHI assumption holds and let $H\colon \{0,1\}^* \to \mathbb{Z}_p^*$ be a secure cryptographic hash function. Let $g_0, g_1, g_2, g_3, g_4, g_5$ be random generators of $\mathbb{G}_1$, h_0, h_1 be random generators of $\mathbb{G}_2$ and u_0, u_1 be random generators of $\mathbb{G}_p$ such that related discrete logarithm of the generators are unknown. This can be done by setting these generators to be output by a hash function of some publicly known seed.

The bank generates an ESS key pair $(\mathsf{pk}_{\mathsf{ESS}}, \mathsf{sk}_{\mathsf{ESS}}) = ((Z, Y), (X, y))$. Next, the bank randomly chooses $\alpha \in_R \mathbb{Z}_p^*$ and computes $h_\alpha = h_0^\alpha$. The bank publishes $\mathsf{pk}_{\mathcal{B}} = (g_0, g_1, g_2, g_3, g_4, g_5, h_0, h_1, h_\alpha, Y, u_0, u_1, Z)$ and saves the secret key $\mathsf{sk}_{\mathcal{B}} = (X, y)$.

To compute the accumulators $V_k = g_0^{\prod_{j=1}^{k}(\alpha+j)}$ and witnesses $W_{k,J} = g_0^{\prod_{j=1, j\neq J}^{k}(\alpha+j)}$ without the secret α, the bank either additionally publishes K elements $g_0^{\alpha^i}, 1 \leq i \leq K$ or directly generates all K accumulators and all $K(K+1)/2$ witnesses. Note, that since $W_{k,k} = V_{k-1}$ for $1 \leq k \leq K$ and $V_0 := g_0$ only $K(K-1)/2$ witnesses have to be published. In the following, we assume that the bank publishes the accumulators and witnesses.

Again, α can be safely deleted. Let be $E_i := \hat{e}(g_i, h_0)$ for $i = 0, \ldots, 5, E_\alpha := \hat{e}(g_2, h_\alpha)$ and $E_{i,1} := \hat{e}(g_i, h_1)$ for $i = 2, 3$. For efficiency consideration, these elements can also be published as part of the public key.

Further, $\mathcal{B}$ sets up a database $\mathcal{D}$.

UserSetup. This is the same as in scheme 1.

WithdrawalProtocol. The user randomly chooses $s', t, y, r \in_R \mathbb{Z}_p^*$, computes the commitment $C' = g_1^{s'} g_2^t g_3^x g_4^y g_5^r$ and sends C' to $\mathcal{B}$. Then, the user generates the following proof of knowledge:

$$\Pi_0 = PoK\left\{(s', t, x, y, r) : C' = g_1^{s'} g_2^t g_3^x g_4^y g_5^r \ \wedge\ \mathsf{pk}_{\mathcal{U}} = u_0^x\right\}.$$

The bank verifies the proof, randomly chooses $e, s'' \in_R \mathbb{Z}_p^*$, computes $A = XV_k^e, B = \left(g_0 g_1^{s''} C'\right)^{1/(y+e)}, C = h_0^e$ and sends (A, B, C, s'') to $\mathcal{U}$.

The user computes $s = s' + s''$ and checks if $\hat{e}(A, h_0) = Z\hat{e}(V_k, C)$ and $\hat{e}(B, CY) = \hat{e}(g_0 g_1^s g_2^t g_3^x g_4^y g_5^r, h_0)$. Then, $\mathcal{U}$ stores the wallet $\mathcal{W} = (A, B, C, s, t, x, y, r, J, k)$ and sets counter $J = 1$.

SpendProtocol. Let the wallet be $\mathcal{W} = (A, B, C, s, t, x, y, r, J, k)$ such that $J \leq k$ and let V_k and $W_{k,J}$ be the corresponding accumulator and witness (either computed by the user or published by the bank). First, the user and the merchant with public key $\mathsf{pk}_{\mathcal{M}}$ agree on transaction information info. Then, both parties compute the hash value $R = H(\mathsf{pk}_{\mathcal{M}} || \mathsf{info})$.

As in scheme 1, the user computes $S = u_1^{1/(s+J+1)}$ and $T = \mathsf{pk}_{\mathcal{U}} u_1^{1/(t+J+1)}$. Now, the user has to hide the elements $A, B, C, V_k, W_{k,J}$. So, $\mathcal{U}$ also computes $A_1 = Ag_1^{r_1}, A_2 = Bg_2^{r_2}, A_3 = Ch_1^{r_3}, A_4 = V_k g_3^{r_4}, A_5 = W_{k,J} g_2^{r_5}, A_6 = g_1^J g_2^t g_3^{r_6}, A_7 = g_1^{r_3} g_2^{r_7}, A_8 = g_1^{r_5} g_2^{r_8}$ for $r_1, r_2, r_3, r_4, r_5, r_6, r_7, r_8 \in_R \mathbb{Z}_p^*$. Then, $\mathcal{U}$ generates the following signature of knowledge (cf. [6, 2]):

$$\begin{aligned}
\Pi_1 = SoK\Big\{&(r_1, r_2, r_3, r_4, r_5, r_6, r_7, r_8, \delta_{2,3}, \delta_{2,7}, \delta_{3,4}, \delta_{4,7}, \delta_6, \delta_J, \delta_t, \delta_{5,J}, \delta_{8,J}, s, t, x, y, r, J) : \\
&A_7 = g_1^{r_3} g_2^{r_7} \ \wedge\ 1 = A_7^{r_2} g_1^{-\delta_{2,3}} g_2^{-\delta_{2,7}} \ \wedge\ 1 = A_7^{r_4} g_1^{-\delta_{3,4}} g_2^{-\delta_{4,7}} \ \wedge \\
&Z\hat{e}(A_4, A_3)\,\hat{e}(A_1, h_0)^{-1} = \hat{e}(A_4, h_1)^{r_3}\,\hat{e}(g_3, A_3)^{r_4}\, E_1^{-r_1} E_{3,1}^{-\delta_{3,4}} \ \wedge \\
&\hat{e}(A_2, A_3 Y)\, E_0^{-1} = E_1^s E_2^t E_3^x E_4^y E_5^r \hat{e}(A_2, h_1)^{r_3}\,\hat{e}(g_2, A_3 Y)^{r_2}\, E_{2,1}^{-\delta_{2,3}} \ \wedge \\
&u_1 S^{-1} = S^{J+s} \ \wedge\ A_6 = g_1^J g_2^t g_3^{r_6} \ \wedge\ 1 = A_6^x g_1^{-\delta_J} g_2^{-\delta_t} g_3^{-\delta_6} \ \wedge \\
&u_1^R T^{-1} = T^{J+t} u_0^{-\delta_J - \delta_t - x} \ \wedge\ A_8 = g_1^{r_5} g_2^{r_8} \ \wedge\ 1 = A_8^J g_1^{-\delta_{5,J}} g_2^{-\delta_{8,J}} \ \wedge \\
&\hat{e}(A_4, h_0)\,\hat{e}(A_5, h_\alpha)^{-1} = \hat{e}(A_5, h_0)^J E_2^{-\delta_{5,J}} E_\alpha^{-r_5} E_3^{r_4}\Big\}(R)
\end{aligned}$$

where $\delta_{i,j} = r_i r_j, \delta_6 = r_6 x, \delta_J = Jx, \delta_t = tx, \delta_{i,J} = r_i J$.

The user sends $S, T, A_1, A_2, A_3, A_4, A_5, A_6, A_7, A_8$ along with Π_1 to the merchant. The merchant verifies Π_1. If the proof is valid, the merchant accepts the payment and stores the coin $\mathsf{coin} = (S, T, R, A_1, A_2, A_3, A_4, A_5, A_6, A_7, A_8, \Pi_1, \mathsf{info})$.

Finally, the user increases the counter J of his wallet by 1.

DepositProtocol, RevokeDoubleSpender and VerifyGuilt proceed analogously to scheme 1.

4.2.1 Security Analysis

We have the following theorem for scheme 2.

Theorem 2. *Compact e-cash scheme 2 is secure under the q-SDH assumption and the SXDH assumption in the random oracle model, if the ASM scheme is secure.*

Proof. The security of scheme 2 directly follows from the security of scheme 1 and the unforgeability of ESS.

Balance: Let $\mathcal{A}$ be a PPT adversary that makes q_W withdrawal queries and q_S spend queries. We show that the success probability of $\mathcal{A}$ is negligible under the q-SDH assumption and the SXDH assumption in the random oracle model, if the ASM scheme [4] is secure.

We construct a simulator $\mathcal{S}$ which acts as challenger $\mathcal{C}$.

For each withdrawal query, $\mathcal{A}$ sends $\mathsf{pk}_{\mathcal{U}}, C'$ and proves the knowledge of representation of $\mathsf{pk}_{\mathcal{U}}$ and C' in Π_0. $\mathcal{S}$ acts as if an honest bank would, except during the proof of knowledge where $\mathcal{S}$ runs a rewind simulation and extracts (s', t, x, y, r). After each execution, $\mathcal{S}$ computes $\mathcal{W} = (S_1, \ldots, S_{k_i}, s, t, x, y, r)$ where $S_J = u_0^{1/(s+J+1)}$ for $1 \leq J \leq k_i$ and $k_i \leq K$ is the requested wallet size of the i-th query. Let $\mathbb{S}_W = (S_{i,J} | 1 \leq i \leq q_W, 1 \leq J \leq k_i)$ be the set of all serial numbers after q_W executions of the withdrawal protocol.

For each spend query, $\mathcal{A}$ presents a public key $\mathsf{pk}_{\mathcal{U}}$, a transaction information info, a merchant public key $\mathsf{pk}_{\mathcal{M}}$ and request a legal number of spend protocol. For each query, $\mathcal{S}$ randomly chooses $s, t \in_R \mathbb{Z}_p^*, k \in_R [1, K], J \in_R [1, k]$ and sends $S = u_1^{1/(s+J+1)}, T = \mathsf{pk}_{\mathcal{U}} u_1^{R/(t+J+1)}$ such that $R = H(\mathsf{pk}_{\mathcal{M}} || \mathsf{info})$. Further, $\mathcal{S}$ randomly chooses $A_1, A_2, A_4, A_5, A_6, A_7, A_8 \in_R \mathbb{G}_1$ and $A_3 \in_R \mathbb{G}_2$. Thus, $A_1, \ldots, A_8$ are perfectly simulated. $\mathcal{S}$ then generates a simulated signature of knowledge Π_1. Let $\mathbb{S}_S = (S_j | 1 \leq j \leq q_S)$ be the set of all serial numbers after q_S executions of the spend protocol.

Finally, $\mathcal{A}$ runs $\sum_{i=1}^{q_W} k_i + q_S + 1$ deposit protocols with $\mathcal{S}$.

As in scheme 1, $\mathcal{A}$ wins the game either if (1) all $\sum_{i=1}^{q_W} k_i + q_S + 1$ serial numbers are unique or (2) some of the serial numbers are duplicated but the $\mathsf{RevokeDoubleSpender}$ algorithm does not output any public key $\mathsf{pk}_{\mathcal{A}} \in \mathcal{U}_{\mathcal{A}}$ of a corrupted user.

Case (1): Since only q_S serial numbers are given to $\mathcal{A}$ during the spend queries, $\mathcal{A}$ must have produce another $\sum_{i=1}^{q_W} k_i + 1$ serial numbers. Thus, $\mathcal{A}$ can only win in case (1) by convincing $\mathcal{S}$ to accept a serial number $S \notin \mathbb{S}_W \cup \mathbb{S}_S$. Analogous to scheme 1, $\mathcal{A}$ must have generated a *false* proof as part of the signature of knowledge such that one of the following is fake:

1. Possession of ESS signature $\sigma = (A, B, C, s)$ on message $m = (V_k, t, x, y, r)$ (instead: possession of BBS+ signature $\sigma = (A, e, s)$ on message $m = (t, x, y, r)$).
2. $S = u_1^{1/(s+J+1)}$.
3. Possession of witness $W_{k,J}$ for counter J and accumulator V_k (instead: possession of witness W_J for counter J and accumulator V).

The last two cases are the same as in scheme 1 and happens with negligible probability under the discrete logarithm assumption and the q-SDH assumption.

Fake proof of possession of ESS signature happens with negligible probability under the SXDH assumption.

As in scheme 1, case (2) is negligible. Thus, as in scheme 1, $\mathcal{A}$'s total success probability is negligible.

Anonymity: Anonymity directly follows from scheme 1. During spending, the used accumulator V_k and witness $W_{k,J}$ are perfectly hidden. Thus, the counter J and the wallet size k

are also perfectly hidden. Consequently, the information leaked about a user or wallet during SpendProtocol are the same as in scheme 1 which fulfills anonymity.

Exculpability: Since the algorithms RevokeDoubleSpender and VerifyGuilt are the same as in scheme 1, exculpability directly follows from scheme 1. □

4.3 Efficiency and Modification of Compact E-Cash Scheme 2

To achieve a spend protocol of complexity $O(\lambda)$, the bank publishes a set of all accumulators and witnesses. The size of this set is

$$|\{V_k : 1 \leq k \leq K\} \cup \{W_{k,J} : 1 \leq k \leq K, 1 \leq J \leq k-1\}| = \frac{K(K+1)}{2},$$

because we need K accumulators and $K(K-1)/2$ additional witnesses, since $W_{k,k} = V_{k-1}$ for $V_0 := g_0$ as mentioned above. So, the size of the bank's public key is $O\left(\lambda \cdot K^2\right)$.

As already mentioned in the introduction, it is also possible not to support every wallet size $1, \ldots, K$, but n different wallet sizes $k_1, \ldots, k_n$ where $k_n = K$. Let $I = \{k_1, \ldots, k_n\}$ be the set of all possible wallet sizes. In this case, the size of the set of all needed accumulators and witnesses is

$$|\{V_{k_i} : 1 \leq i \leq n\} \cup \{W_{k_i,J} : 1 \leq i \leq n, 1 \leq J \leq k_i\} \setminus \{g_0\}| = n + \sum_{i=1}^{n} k_i - |\{k_i : k_i - 1 \in I \cup \{0\}\}|,$$

because we only need n accumulators and k_i witnesses for each accumulator V_{k_i} for $1 \leq i \leq n$. However, as above, we have $W_{k,k} = V_{k-1}$ for $V_0 := g_0$. For $n = O(1)$, the size of the bank's public key is reduced to $O\left(\lambda \cdot K\right)$.

For example, let $K = 1\,000$ be the maximum wallet size. Then, the size of the set of all accumulators and witnesses is $500\,500$. If, for example, a user only can choose a wallet size of $1, 2, 5, 10, 20, 50, 100, 200, 500$ and $1\,000$ coins, the size of the set of all needed accumulators and witnesses is only $1\,896 < 2K$.

We give an other example. If we require $K = 2^L$ as in [13] and want to support every wallet size 2^ℓ for $0 \leq \ell \leq L$, the size of the set of all needed accumulators and witnesses is

$$\begin{aligned} &\left|\{V_{2^\ell} : 0 \leq \ell \leq L\} \cup \left\{W_{2^\ell,J} : 0 \leq \ell \leq L, 1 \leq J \leq 2^\ell\right\} \setminus \{g_0, V_1\}\right| \\ &\qquad = L + 1 + \sum_{\ell=0}^{L} 2^\ell - 2 = 2^{L+1} + L - 2 = 2K + L - 2, \end{aligned}$$

because we only need $L+1$ accumulators and 2^ℓ witnesses for each accumulator V_{2^ℓ} for $0 \leq \ell \leq L$, but again we have $W_{1,1} = V_0 := g_0$ and $W_{2,2} = V_1$. Thus, again the size of the bank's public key is $O\left(\lambda \cdot K\right)$ where now $n = O(\log(K))$. For $L = 10$ we have $K = 1\,024$ and the size of the set of all accumulators and witnesses is $2\,056 \approx 2K$.

5 Concluding Remarks

We introduced a new idea of a compact e-cash scheme that provides arbitrary wallet size without the need of inefficient exact range proof like in [4] and gave an efficient and secure construction.

Analogous to [4], this construction can be extended to support batch spending and compact spending. Further, it also can be extended to support full coin tracing of double-spenders as proposed in [13] or [4].

References

[1] G. Ateniese, J. Camenisch, S. Hohenberger, and B. de Medeiros. Practical group signatures without random oracles. Cryptology ePrint Archive, Report 2005/385, October 2005.

[2] M. H. Au. *Contribution to Privacy-Preserving Cryptographic Techniques*. PhD thesis, School of Computer Science and Software Engineering, University of Wollongong, May 2009.

[3] M. H. Au, W. Susilo, and Y. Mu. Constant-size dynamic k-taa. In *Security and Cryptography for Networks – SCN '06*, volume 4116 of *Lecture Notes in Computer Science*, pages 111–125. Springer, 2006.

[4] M. H. Au, W. Susilo, and Y. Mu. Practical compact e-cash. In *Information Security and Privacy – ACISP '07*, volume 4586 of *Lecture Notes in Computer Science*, pages 431–445. Springer, 2007.

[5] M. H. Au, W. Susilo, and Y. Mu. Practical anonymous divisible e-cash from bounded accumulators. In *Financial Cryptography and Data Security – FC '08*, volume 5143 of *Lecture Notes in Computer Science*, pages 287–301. Springer, 2008.

[6] M. H. Au, Q. Wu, W. Susilo, and Y. Mu. Compact e-cash from bounded accumulator. In *Topics in Cryptology – CT-RSA '07*, volume 4377 of *Lecture Notes in Computer Science*, pages 178–195. Springer, 2007.

[7] M. Bellare, J. A. Garay, and T. Rabin. Fast batch verification for modular exponentiation and digital signatures. In *Advances in Cryptology – EUROCRYPT '98*, volume 1403 of *Lecture Notes in Computer Science*, pages 236–250. Springer, 1998.

[8] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *1st ACM Conference on Computer and Communications Security – CCS '93*, CCS, pages 62–73. ACM, 1993.

[9] J. Benaloh and M. de Mare. One-way accumulators: A decentralized alternative to digital signatures (extended abstract). In *Advances in Cryptology – EUROCRYPT '93*, volume 765 of *Lecture Notes in Computer Science*, pages 274–285. Springer, 1994.

[10] D. Boneh and X. Boyen. Short signatures without random oracles. In *Advances in Cryptology – EUROCRYPT '04*, volume 3027 of *Lecture Notes in Computer Science*, pages 56–73. Springer, 2004.

[11] D. Boneh, X. Boyen, and H. Shacham. Short group signatures. In *Advances in Cryptology – CRYPTO '04*, volume 3152 of *Lecture Notes in Computer Science*, pages 41–55. Springer, 2004.

[12] F. Boudot. Efficient proofs that a committed number lies in an interval. In *Advances in Cryptology – EUROCRYPT '00*, volume 1807 of *Lecture Notes in Computer Science*, pages 431–444. Springer, 2000.

[13] J. Camenisch, S. Hohenberger, and A. Lysyanskaya. Compact e-cash. In *Advances in Cryptology – EUROCRYPT '05*, volume 3494 of *Lecture Notes in Computer Science*, pages 302–321. Springer, 2005.

[14] J. Camenisch and A. Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. In *Advances in Cryptology – CRYPTO '02*, volume 2442 of *Lecture Notes in Computer Science*, pages 61–76. Springer, 2002.

[15] J. Camenisch and A. Lysyanskaya. A signature scheme with efficient protocols. In *Security in Communication Networks – SCN '02*, volume 2576 of *Lecture Notes in Computer Science*, pages 268–289. Springer, 2002.

[16] J. Camenisch and A. Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In *Advances in Cryptology – CRYPTO '04*, volume 3152 of *Lecture Notes in Computer Science*, pages 56–72. Springer, 2004.

[17] J. Camenisch and M. Stadler. Efficient group signature schemes for large groups. In *Advances in Cryptology – CRYPTO '97*, volume 1294 of *Lecture Notes in Computer Science*, pages 410–424. Springer, 1997.

[18] S. Chatterjee and A. Menezes. On cryptographic protocols employing asymmetric pairings – the role of ψ revisited. Cryptology ePrint Archive, Report 2009/480, September 2009.

[19] D. Chaum. Blind signatures for untraceable payments. In *Advances in Cryptology – CRYPTO '82*, pages 199–203. Plenum Press, 1983.

[20] Y. Dodis and A. Yampolskiy. A verifiable random function with short proofs and keys. In *Public-Key Cryptography – PKC '05*, volume 3386 of *Lecture Notes in Computer Science*, pages 416–431. Springer, 2005.

[21] A. Fiat and A. Shamir. How to prove yourself: Practical solutions of identification and signature problems. In *Advances in Cryptology – CRYPTO '86*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer, 1986.

[22] S. Galbraith, K. Paterson, and N. Smart. Pairings for cryptographers. Cryptology ePrint Archive, Report 2006/165, May 2006.

[23] L. Nguyen. Accumulators from bilinear pairings and applications to id-based ring signatures and group membership revocation. In *Topics in Cryptology – CT-RSA '05*, volume 3376 of *Lecture Notes in Computer Science*, pages 275–292. Springer, 2005.

[24] T. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Advances in Cryptology – CRYPTO '91*, volume 576 of *Lecture Notes in Computer Science*, pages 129–140. Springer, 1991.

[25] B. Qin, Q. Wu, J. Domingo-Ferrer, and L. Zhang. Preserving security and privacy in large-scale vanets. In *ICICS '11*, volume 7043 of *Lecture Notes in Computer Science*, pages 121–135. Springer, 2011.

[26] S. Schäge. Tight proofs for signature schemes without random oracles. In *Advances in Cryptology – EUROCRYPT '11*, volume 6632 of *Lecture Notes in Computer Science*, pages 189–206. Springer, 2011.