Fair E-cash: Be Compact, Spend Faster*

Sébastien Canard[1], Cécile Delerablée[2], Aline Gouget[3], Emeline Hufschmitt[4], Fabien Laguillaumie[5], Hervé Sibert[6], Jacques Traoré[1], and Damien Vergnaud[7]

[1] Orange Labs R&D, 42 rue des Coutures, BP6243, F-14066 Caen Cedex, France.
[2] UVSQ, 45 Avenue des Etats-Unis, 78035 Versailles Cedex, France.
[3] Gemalto, 6 rue de la Verrerie, 92190 Meudon, France.
[4] Thalès Communications, 160 boulevard de Valmy, 92704 Colombes, France.
[5] GREYC - Université de Caen-Basse Normandie, France.
[6] ST-Ericsson, 9-11 rue Pierre-Felix Delarue, 72100 Le Mans Cedex 9, France.
[7] École normale supérieure – C.N.R.S. – I.N.R.I.A., France.

Abstract. We present the first *fair e-cash system* with a compact wallet that enables users to spend efficiently k coins while only sending to the merchant $\mathcal{O}(\lambda \log k)$ bits, where λ is a security parameter. The best previously known schemes require to transmit data of size at least linear in the number of spent coins. This result is achieved thanks to a new way to use the Batch RSA technique and a tree-based representation of the wallet. Moreover, we give a variant of our scheme with a less compact wallet but where the computational complexity of the spend operation does not depend on the number of spent coins, instead of being linear at best in existing systems.

Keywords. Fair e-cash, privacy-preserving, batch RSA, blind signature.

1 Introduction

Electronic cash systems allow users to withdraw electronic coins from a bank, and then to pay merchants using these coins preferably in an off-line manner, i.e. with no need to communicate with the bank or a trusted party during the payment. Finally, the merchant deposits the coins he has received to the bank.

An e-cash system should provide user anonymity against both the bank and the merchant during a purchase in order to emulate the perceived anonymity of regular cash. However, it seems that the necessity to fight against money laundering encourages the design of fair e-cash systems where a trusted party can, at any time when it's needed, revoke the anonymity of users. We thus focus on the design of fair e-cash systems. In order to reach the privacy target while being reasonably practical, it is necessary to focus on the efficiency of the most repeated protocol, namely the spending one between the user and the merchant. It should also be possible to withdraw or spend several coins more efficiently than repeating a single withdrawal or spending protocol. At last, we must pay attention to the compactness of the data that are exchanged in all protocols.

Related Works. The compact e-cash system [8] has recently aroused a new interest in e-cash by proposing the first e-cash system permitting a user to efficiently withdraw a wallet with 2^L coins such that the space required to store these coins, and the complexity of the withdrawal protocol, are proportional to L rather than to 2^L. Another possibility of efficient withdrawal is also given in [1]. These schemes fulfill all security properties usually required in the non-fair setting but do not consider the efficiency of the spending phase. One solution to improve it is to manage a wallet that contains coins with several monetary values [12]. The main drawback of this solution is that the user must choose during the withdrawal protocol how many coins he

* This work has been financially supported by the French Agence Nationale de la Recherche and the TES Cluster under the PACE project while 2nd author was working at Orange Labs and 4th author at ENS.

wants for each monetary value. In [2], the initial compact e-cash scheme is modified to improve the spending phase; however, the overall cost is still linear in the number of spent coins and, again, the paper only consider non-fair e-cash. Consequently, there exists no privacy-preserving fair e-cash system allowing the user to both (i) withdraw compact wallets and (ii) spend several coins while the transmitted data size is less than linear in the number of spent coins.

Our Contributions. This paper presents a fair e-cash system with a compact wallet that allows users to spend efficiently k coins while sending to the merchant only $\mathcal{O}(\lambda \log k)$ bits, with λ a security parameter, while preserving the privacy of the users. Our proposal makes use of two main cryptographic building blocks: *blind signatures* [13] and *batch cryptography* [17]. The concept of blind signature is the essence of many e-cash systems [15, 6, 26]. However, many of these suffer from a lack of efficiency since they usually use the cut-and-choose method in order to identify double-spenders [15]. The Batch RSA method makes it possible to efficiently obtain multiple RSA signatures of multiple messages. Batch cryptography has been used to build several e-cash systems, in order to get additional properties [16, 7], to decrease the amount of processing done by the merchant [22], or to improve the efficiency of the withdrawal process at the cost of the linkability of coins withdrawn together [5].

To the best of our knowledge, our proposal is the most efficient (fair) e-cash system in terms of wallet storage size, computational complexity of spending and spending transfer size, which is strongly unforgeable. Note that the level of anonymity achieved by our scheme is strong but it is not perfect. Indeed it is strong because it is impossible to link (i) a withdrawal protocol with a user identity, (ii) a spending protocol to a withdrawal protocol, and (iii) two spending protocols but only under specific constraints. The anonymity property achieved by our scheme cannot be perfect since some information related to the coin number (with respect to the wallet) leaks during the spending phase.

2 Security Model

2.1 Algorithms

A fair e-cash system involves four kinds of players: a user $\mathcal{U}$, a bank $\mathcal{B}$, a merchant $\mathcal{M}$ and a judge $\mathcal{J}$. Each user is able to withdraw a wallet with ℓ coins. Such wallet consists of an identifier and a proof of validity. A fair e-cash scheme is defined by the following algorithms, where λ is a security parameter.

- $\mathsf{ParamGen}(1^\lambda)$ is a probabilistic algorithm that outputs the parameters of the system $params$. In the sequel, all algorithms take as input 1^λ and $params$.
- $\mathsf{JKeyGen}()$, $\mathsf{BKeyGen}()$ and $\mathsf{UKeyGen}()$ are key generation algorithms for $\mathcal{J}$, $\mathcal{B}$ and $\mathcal{U}$, respectively. The key pairs are denoted by $(sk_\mathcal{J}, pk_\mathcal{J})$, $(sk_\mathcal{B}, pk_\mathcal{B})$, and $(sk_\mathcal{U}, pk_\mathcal{U})$. Note that $\mathsf{UKeyGen}()$ also provides the keys of merchants that can be seen as users in e-cash systems.
- $\mathsf{Register}(\mathcal{J}(sk_J, pk_\mathcal{U}), \mathcal{U}(sk_\mathcal{U}, pk_\mathcal{J}))$ is an interactive protocol whose outcome is a notification decision of $\mathcal{J}$ together with a certificate of validity of $\mathcal{U}$'s public key which guarantee that $\mathcal{U}$ knows his secret key.
- $\mathsf{Withdraw}(\mathcal{U}(pk_\mathcal{B}, sk_\mathcal{U}, \ell), \mathcal{B}(pk_\mathcal{U}, sk_\mathcal{B}))$ is an interactive protocol that allows $\mathcal{U}$ to withdraw a wallet W of ℓ coins. The output of $\mathcal{U}$ is a wallet W, i.e. an identifier I and a proof of validity Π, or an error message $\perp$. The output of $\mathcal{B}$ is its view $\mathcal{V}_\mathcal{B}^{\mathsf{Withdraw}}$ of the protocol.
- $\mathsf{Spend}(\mathcal{U}(W, pk_\mathcal{M}, pk_\mathcal{B}, k), \mathcal{M}(sk_\mathcal{M}, pk_\mathcal{B}))$ is an interactive protocol enabling $\mathcal{U}$ to spend k coins. $\mathcal{M}$ outputs the serial numbers $S_0, \cdots, S_{k-1}$ and a proof of validity π. $\mathcal{U}$'s output is an updated wallet W' or an error message $\perp$.

- $\mathsf{Deposit}(\mathcal{M}(sk_{\mathcal{M}}, (S_0, \ldots, S_{k-1}), \pi, pk_{\mathcal{B}}), \mathcal{B}(pk_{\mathcal{M}}, sk_{\mathcal{B}}))$ is an interactive protocol allowing $\mathcal{M}$ to deposit the coins, i.e. $S_0, \ldots, S_{k-1}$ and π. $\mathcal{B}$ adds the coins to the list of spent coins or outputs an error message $\perp$.
- $\mathsf{Identify}(S, \pi_1, \pi_2, sk_{\mathcal{J}})$ is an algorithm executed by $\mathcal{J}$ which outputs a proof Π_G and either a registered public key $pk_{\mathcal{U}}$ or $\perp$.
- $\mathsf{VerifyGuilt}(S, pk_{\mathcal{U}}, \Pi_G, pk_{\mathcal{J}})$ is an algorithm allowing to publicly verify the proof Π_G that the $\mathsf{Identify}$ has been done correctly.

2.2 Security Properties

We informally describe the security statements of a fair e-cash scheme.

Unforgeability. From the bank point of view, what matters is that no coalition of users can ever spend more coins than they have withdrawn:
  - let $\mathcal{A}$ be an adversary that has access to the public key $pk_{\mathcal{B}}$ of the system;
  - $\mathcal{A}$, playing a user, executes in a concurrent manner $\mathsf{Withdraw}$ and $\mathsf{Deposit}$ protocols with the bank. $\mathcal{A}$ can legitimately withdraw f wallets; we denote by w_f the number of coins withdrawn during these executions.
  - the adversary $\mathcal{A}$ wins the game if, at any time, the honest bank accepts more than w_f coins (without detecting a double-spending).

  We require that no PPT adversary succeeds in this game with non-negligible probability.

Anonymity. From the user privacy point of view, the bank, even when cooperating with malicious users and merchants, should not learn anything about a user's spending other than from the environment. We capture a weaker notion of anonymity by assuming that the targeted users withdraw and spend the same number of coins (see discussion in Section 5.2):
  - let $\mathcal{A}$ be an adversary that has access to the secret key $sk_{\mathcal{B}}$ of the bank;
  - $\mathcal{A}$ executes $\mathsf{Withdraw}$ (as the bank) and Spend (as the merchant) protocols any number of times. $\mathcal{A}$ can also corrupt players;
  - at any time of the game, $\mathcal{A}$ chooses two honest users $\mathcal{U}_0$ and $\mathcal{U}_1$ such that both $\mathcal{U}_0$ and $\mathcal{U}_1$ has withdrawn and spent the *same* number of coins. Then, a bit $b \in \{0, 1\}$ is chosen and a Spend protocol is played between $\mathcal{U}_b$ and $\mathcal{A}$. At the same time, we assume that $\mathcal{U}_{\bar{b}}$ also plays a Spend protocol that is not observed by $\mathcal{A}$. Next, $\mathcal{A}$ can again executes $\mathsf{Withdraw}$ (as the bank) and Spend (as the merchant) protocols;
  - the adversary $\mathcal{A}$ finally outputs a bit b'.

  We require that for any PPT adversary, the probability that $b' = b$ differs significantly from $1/2$ is negligible.

Identification of double-spenders. From the bank's point of view, no collection of users should be able to double-spend a coin without revealing one of their identities:
  - let $\mathcal{A}$ be a an adversary that has access to $pk_{\mathcal{B}}$;
  - $\mathcal{A}$ executes, as a user, $\mathsf{Withdraw}$ and Spend protocols as many time as it wishes;
  - $\mathcal{A}$ wins the game if, at any time, the bank outputs $\perp$ while the merchant executes the $\mathsf{Deposit}$ protocol and $\mathsf{Identify}$ outputs $\perp$.

  We require that no PPT adversary succeeds with non-negligible probability.

Exculpability. The bank, even cooperating with malicious users, cannot falsely accuse honest users from having double-spent a coin, and only users who double-spent a coin can be convicted:
  - let $\mathcal{A}$ be an adversary that has access to both the secret key $sk_{\mathcal{B}}$ of the bank and the one $sk_{\mathcal{J}}$ of the judge;

– the adversary $\mathcal{A}$ can create as many users as he wants and corrupt some of them. All along the game, $\mathcal{A}$ plays the bank side of the Withdraw and Deposit protocols, $\mathcal{A}$ can play either the role of the user (as a corrupted user) or the role of the merchant during Spend protocols;
– the adversary $\mathcal{A}$ wins the game if, at any time, the Identify algorithm outputs the public key of an honest user together with a valid proof Π_G.

We require that no PPT adversary succeeds with non-negligible probability.

3 Useful Tools, Notations and Conventions

In the sequel, λ is the general security parameter. In a withdrawal protocol, the user withdraws $\ell \leq K = 2^L$ coins from the bank, and every coin is labeled with a serial number $S_j, 0 \leq j < \ell$. In a spending protocol, the number of remaining coins in the wallet before spending and the number of coins to be spent is denoted by K' and k, respectively.

3.1 Batch RSA Method

The Batch RSA method [17] makes it possible, for a given RSA modulus, to efficiently obtain multiple RSA signatures whose public exponents are coprime pairwise.

Let n be an RSA modulus for which the factorization is only known by the signer. Let $e_0, \ldots, e_{\ell-1}$ be ℓ exponents, coprime both pairwise and with $\phi(n)$, with $\ell \leq K = 2^L$. As the efficiency of the Batch RSA depends on the size of these exponents, a generic suitable choice is the ℓ first odd prime numbers. Let $E = \prod_{i=0}^{\ell-1} e_i$. Given messages $S_0, S_1, \ldots, S_{\ell-1}$, it is possible to generate the ℓ roots $S_0^{1/e_0} \pmod n, \ldots, S_{\ell-1}^{1/e_{\ell-1}} \pmod n$ in $\mathcal{O}(\log K \log E + \log n)$ modular multiplications and $\mathcal{O}(K)$ divisions. We sketch the steps of the Batch RSA description and complexity proof described in [17]:

– (B1) compute the product $M = \prod_{i=0}^{\ell-1} S_i^{E/e_i}$ along a binary tree as shown in Figure 1 for the case $\ell = 5$. Every complete binary tree with ℓ leaves is suitable. However, for efficiency purpose, we suppose the height of the tree is $\mathcal{O}(\log K) = \mathcal{O}(L)$. Each node in the tree contains a value $M_{[i_1 \ldots i_2]} = \prod_{i=i_1}^{i_2} S_i^{E_{[i_1 \ldots i_2]}/e_i}$ with $E_{[i_1 \ldots i_2]} = \prod_{i=i_1}^{i_2} e_i$. In order to compute this tree, the number of operations is $\mathcal{O}(\log K \log E + \log n)$ multiplications;
– (B2) compute the batch signature $M^{1/E} = \prod_{i=0}^{\ell-1} S_i^{1/e_i}$, as a usual RSA signature with public exponent E;
– (B3) decompose $M^{1/E}$ in order to obtain the values S_i^{1/e_i}. In this step, the binary tree built at the first step is parsed down, and at each node of the tree the value $M_{[i_1 \ldots i_2]}^{1/E_{[i_1 \ldots i_2]}} = \prod_{i=i_1}^{i_2} S_i^{1/e_i}$ is computed and broken into two factors (one for each son) by using the Chinese remainder theorem and the values computed in (B1). The cost of this last step is $\mathcal{O}(K)$ modular divisions and $\mathcal{O}(\log E \log K)$ operations.

Use of Batch RSA in our proposal. The messages signed using Batch RSA are the serial numbers of coins. For efficiency purpose, the Batch RSA exponents e_i are the K first prime numbers. Therefore, we have $\log E = \mathcal{V}(e_{K-1})$, where $\mathcal{V}$ is the Chebyshev function[1]. This yields $\log E \sim K \ln K$.

During the withdrawal, the user has to perform steps (B1) and (B2) (see Section 3.2) in order to receive an aggregated signature on all the serial numbers that he has chosen. The aggregated value $M^{1/E}$ represents his wallet.

[1] We recall that the Chebyshev function is $\mathcal{V}(x) = \sum_{p \leq x \text{ prime}} \log(p)$.

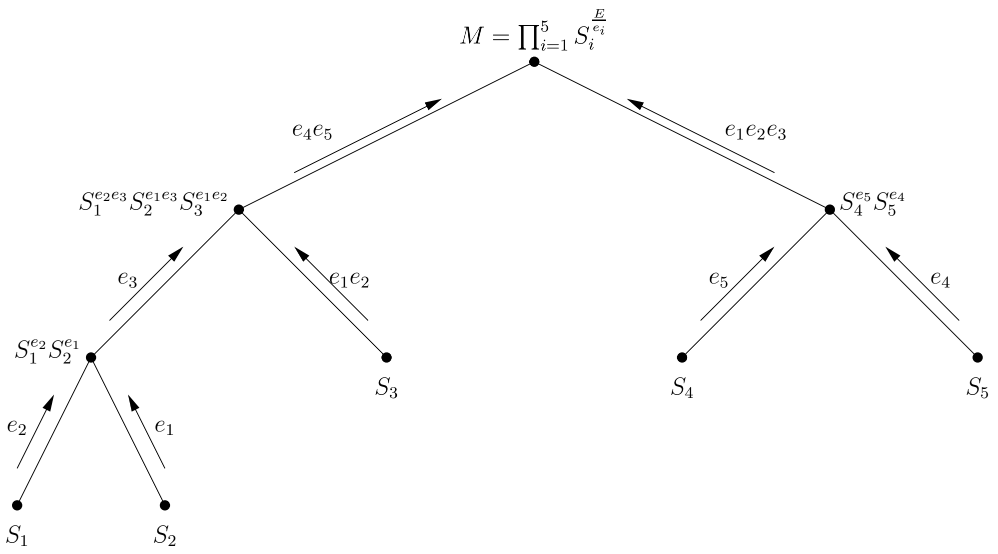

Fig. 1. Withdrawal binary tree for the computation of M

One novel aspect of our scheme is that it is never necessary to fully decompose the aggregated signature into all the signatures of spent coins during the spending phase. Indeed, at each spending, the current aggregated signature is split into two parts following a single node operation from step (B3), the first part being the aggregated signature of the coins to be spent, and the second part being the new wallet signature representing the remaining coins. Suppose that a user still owns an aggregated signature $M_F^{1/E'} = \prod_{i \in F} S_i^{1/e_i}$, with $F \subset \{0, \ldots, \ell - 1\}$ and $E' = \prod_{i \in F} e_i$. This user wants to spend a subset F_1 of the coins in F. Let $F_2 = F \setminus F_1$. In order to compute the aggregated signature $M_{F_1}^{1/E'_1} = \prod_{i \in F_1} S_i^{1/e_i}$, the user creates two binary trees, corresponding to the subsets F_1 and F_2, respectively, and connects them at the root of a new binary tree. Then, the user computes the resulting tree as in step (B1) above in order to obtain the two factors M_{F_1} and M_{F_2}. The cost is $\mathcal{O}(\log \#F \log E' + \log n)$. Using the values computed for the roots of each subset F_i, the user can now retrieve the aggregated signature to be spent and the remainder as another aggregated signature. The cost of this operation is 2 modular divisions and $\mathcal{O}(\log E')$ multiplications. An example is shown in Figure 2.

This technique allows a user to carry a very small amount of data and to transfer reduced signature data. Indeed, in this case, only the non-spent interval and the remaining aggregated signature must be stored in the wallet, while a single aggregated signature is sent to the merchant. There are several trade-offs related to how we use the Batch RSA signatures. We detail them in Section 6.

3.2 RSA Blind Signature Scheme

A blind signature [13] is a protocol between a user and a signer where the user gets a signature from the signer in a way that the signer does not know the content of the message he is signing. Furthermore, the signer cannot link afterward his views of the protocol to the resulting signatures.

A common blind signature is the RSA blind signature scheme from Chaum [13, 14]. This three-move blind signature scheme is defined by a set of five algorithms BS=(KeyGen, Blind, Sign, $\mathsf{UnBlind}$, Verif), where Blind corresponds to the computation of $\tilde{M} = r^e.\mathcal{H}(M) \pmod n$ where r is a secret random value, M is the message to be blindly signed and $\mathcal{H}$ is a one-way

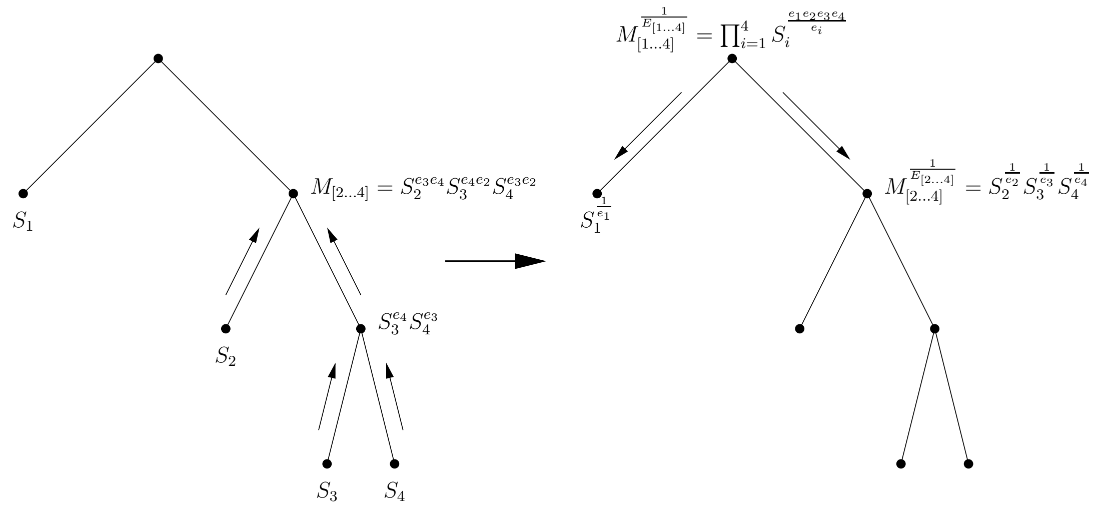

Fig. 2. Binary tree built to spend coins $2, 3, 4$ from a wallet with 4 remaining coins

collision-resistant hash function, while Unblind consists in computing $\sigma = \tilde{\sigma}/r \pmod n$, where $\tilde{\sigma}$ is a classical RSA signature on the message $\tilde{M}$. Thus, it is obvious that σ is also a classical RSA signature of the message M.

Use of the RSA blind signature scheme in our proposal. Our scheme relies on blind RSA signatures using the Batch RSA technique, for which we choose a modulus n, where $\log n$ is polynomial in λ. The messages signed using the RSA blind signature are serial numbers of coins. During step (B2), the batch signature is replaced by a blind signature process. Thus, for $M = \prod_{i=0}^{\ell-1} \mathcal{H}(S_i)^{E/e_i}$, instead of simply computing the message $M^{1/E} = \prod_{i=0}^{\ell-1} \mathcal{H}(S_i)^{1/e_i}$, the signer obtains from the user $\tilde{M} = r^E M \pmod n$ and computes $\tilde{\sigma} = \tilde{M}^{1/E} = r \prod_{i=0}^{\ell-1} \mathcal{H}(S_i)^{1/e_i} \pmod n$. The user finally computes, as for the traditional RSA blind signature scheme, $\sigma = \tilde{\sigma}/r \pmod n$, which corresponds to $\prod_{i=0}^{\ell-1} \mathcal{H}(S_i)^{1/e_i}$, as desired.

3.3 Signature of Knowledge

Zero-knowledge proofs of knowledge (ZKPK) are interactive protocols between a verifier and a prover allowing a prover to assure the verifier his knowledge of a secret, without any leakage of it. In the following, we use proofs of knowledge of a discrete logarithm [25, 19], of a representation, proof of equality of two known representations in the same or in different groups [4]. In the following, we denote by $PK(\alpha_1, \ldots, \alpha_q : \mathsf{R}(\alpha_1, \ldots, \alpha_q))$ a proof of knowledge of the secrets $\alpha_1, \ldots, \alpha_q$ verifying the relation R. Note that the combination of these proofs and the underlying security have been studied in [21, 11] and refined in [9].

These interactive proofs can also be used non interactively (a.k.a. *signatures of knowledge*) by using the Fiat-Shamir heuristic [18].

3.4 Camenisch-Lysyanskaya Type Signature Schemes.

Camenisch and Lysyanskaya have proposed in [10] various signature schemes which include new features. These signatures, called CL signatures for short, are based on Pedersen's commitment scheme which allows a user to commit some values without revealing them. CL signatures should satisfy the unforgeability property and have the following protocols.

– KeyGen: a key generation algorithm which outputs a key pair (sk, pk).

– Sign: an efficient protocol between a user and a signer that permits the user to obtain from the signer a signature Σ of some commitment $C = \mathsf{Commit}(x_1, \ldots, x_k)$ such that $(x_1, \ldots, x_k)$ are unknown from the signer. The latter uses the CLSign algorithm on input C and the user obtains a signature Σ on the messages $(x_1, \ldots, x_k)$, such that $\mathsf{Verif}(\Sigma, (x_1, \ldots, x_k)) = 1$.
– ZKPK: an efficient ZKPK of a signature of some values that are moreover (may be independently) committed.
– Verif: a procedure verifying the signature Σ on the messages $(x_1, \ldots, x_k)$.

One possible choice is to take the construction from [10], which is secure under the flexible RSA assumption (a.k.a. strong RSA assumption), and where the signature on values $(x_0, \ldots, x_k)$ is (A, e, s) such that $A^e = a_0 a_1^{x_1} \cdots a_k^{x_k} b^s$, where the a_i's and b are public.

4 Compact spending

In this section, we first give a high level description of our proposal before describing the procedure and protocols of our scheme.

4.1 Overview of our scheme

In e-cash systems, a withdrawal protocol allows a user to get from the bank, a wallet of coins that can be represented by a set of *serial numbers* and a signature of the bank that will allow him to prove the validity of the coins. The spending protocol of a fair e-cash system usually includes the generation of ℓ valid serial numbers $S_0, \ldots, S_{\ell-1}$ (to allow the detection of double-spending by the bank during the deposit protocol), a verifiable encryption of the spender public key, and a proof of validity of the S_i's and of the encryption of the user public key without revealing any information about his identity.

Serial numbers. As we have seen, the Batch RSA technique can be used to obtain compact spendings by aggregating signatures. However, the transmission of the serial numbers also has to get more compact in order to decrease the overall spending complexity. In order to compact data related to serial numbers, we use a tree with a derivation mechanism from the root to the leaves which represent the serial numbers of the coins. In our scheme, the maximal number of coins that can be withdrawn during a protocol is a fixed parameter of the system $K = 2^L$. Each wallet of monetary value $\ell \leq K = 2^L$ withdrawn from the bank is mapped to a binary tree of $L+1$ levels[2]. The tree root is assigned a *compact* serial number $S_{0,0}$. For every level i, $0 \leq i < L$, the 2^i nodes are assigned each a *compact* serial number denoted by $S_{i,j}$ with $0 \leq j < 2^i$. The values $S_{L,j}$ with $0 \leq j < 2^L$ related to the leaves of the tree are called the *serial numbers* of the purse and denoted S_j.

The derivation is illustrated by Figure 3 and it works as follows: the descendants from a node $S_{i,j}$ are given by a public function $\mathcal{F}(\cdot, \cdot)$ that, on input a compact serial number $S_{i,j}$ and a bit $b \in \{0, 1\}$ to indicate *left* or *right*, outputs the (compact) serial number $S_{i+1,2j+b}$ of the left or right descendant of $S_{i,j}$ in the tree. Thus, from the tree root $S_{0,0}$, it is possible to compute all the serial numbers $S_{i,j}$, $0 \leq i \leq L$, $0 \leq j < 2^i$. The idea used to obtain compact spendings with serial numbers is that it is possible to send the serial number of a node $S_{i,j}$ instead of the serial numbers of all the leaves that come from him. Conversely, once a node $S_{i,j}$ is revealed, none of its descendants or ascendants can be spent, and no node can be spent more than once. This rule is necessary to protect against over-spending. It must also be impossible to compute a serial number without the knowledge of one of its ascendants. Finally, for security reasons, function $\mathcal{F}$ must be collision-free.

[2] The user may withdraw less than 2^L coins, but still has to work with a tree of depth $L+1$, because the number of derivations to get the serial number of a coin must be the same for all users in order to prevent linking.

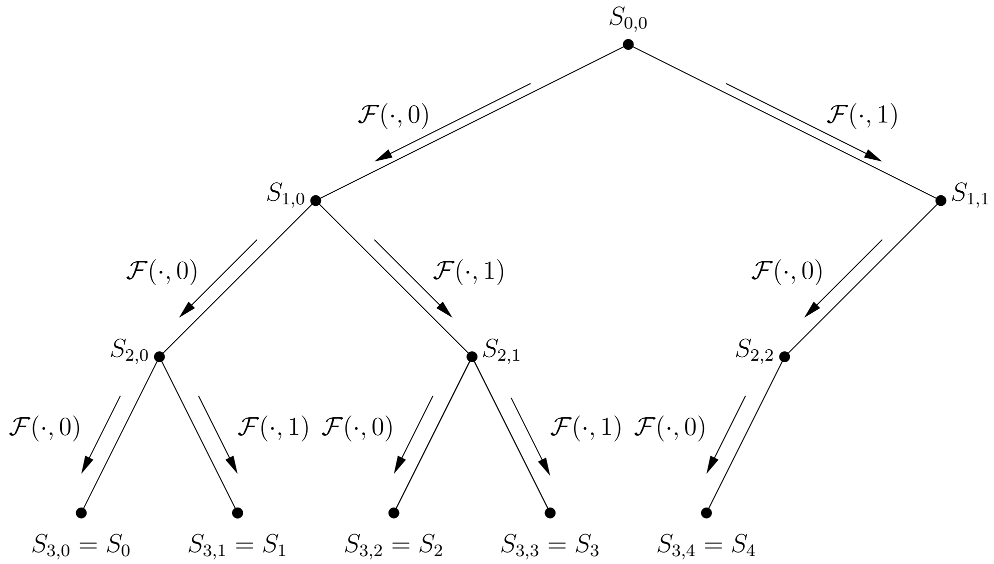

Fig. 3. Serial number binary tree for $\ell = 5$ and $K = 2^3$

Withdrawal. During the withdrawal protocol, the user chooses a number $\ell \leq K = 2^L$ of coins to withdraw. For every j, $0 \leq j \leq \ell - 1$, the serial number S_j is the message related to the exponent e_j (see Section 3.1). The user computes the ℓ serial numbers $S_0, \ldots, S_{\ell-1}$ from a compact serial number $S_{0,0} = s$, where s is a random value known only by the user but computed jointly by the bank and the user, so as to prevent an attack where two users use the same compact serial number. The user at last obtains from the bank both a blind Batch RSA signature on the serial numbers $S_0, \ldots, S_{\ell-1}$ with exponents $e_0, \ldots, e_{\ell-1}$ and a CL signature on s and her identity u.

Spending. When a user wants to spend k coins, she does not need to send k serial numbers and k proofs of validity but only one batch signature (see Section 3.1) and $\mathcal{O}(\lambda \log(k))$ nits corresponding to "compact serial numbers", assuming that the user spends the coins by increasing (or decreasing) exponents. As the size of the remaining values transmitted during spending is at most $\mathcal{O}(\lambda \log k)$ bits, this is also the overall size of the data transmitted during the spending protocol.

Finally, the merchant can verify the correctness of the serial numbers (w.r.t. the bank) using a ZKPK of the CL signature on the values s and u done by the user, following a technique given in [26] which permits us not to prove that the spent serial numbers are indeed generated from the value s signed by the bank.

4.2 Setup Procedure

The $\mathsf{ParamGen}$ procedure first sets $2^L = K$ as the maximum number of coins in a wallet and $e_0, \ldots, e_{K-1}$ as K distinct small prime numbers. For all $i \in [1, K]$, $E_i = \prod_{j=0}^{i-1} e_j$. Next $\mathsf{Enc}_{\mathcal{J}}(\cdot)$ is an encryption function of the judge's IND-CPA public key cryptosystem (e.g. the El Gamal encryption scheme), $\mathcal{H}(\cdot)$ and $\mathcal{F}(\cdot, \cdot)$ are two one-way collision resistant (hash) functions, g is a generator of a cyclic group G of prime or unknown order (the structure of the group depends on the chosen CL signature scheme). Next, the bank $\mathcal{B}$ (resp. the judge $\mathcal{J}$) executes the $\mathsf{BKeyGen}$ (resp. $\mathsf{JKeyGen}$) procedure by executing the KeyGen algorithms of the CL and blind signature schemes (resp. of the encryption scheme).

During the $\mathsf{UKeyGen}$ procedure, each user $\mathcal{U}$ is finally associated to a long-term private key $sk_{\mathcal{U}} = u$ and a corresponding public key $pk_{\mathcal{U}} = g^u$, where g is a public parameter.

4.3 Withdrawal Protocol

Let $\mathcal{U}$ be a user who wants to withdraw ℓ (with $0 < \ell \leq K$) coins to the bank $\mathcal{B}$. The protocol between $\mathcal{U}$ and $\mathcal{B}$ is described in Figure 4. Note that $\mathcal{B}$ can compute the commitment C on u, $s = s' + s''$ and w using only C' and s'' and without needing to know s' and thus s. Next, the computation of E_ℓ and the serial numbers $S_0, \ldots, S_{\ell-1}$ is done using the tree structure we described above with $\mathcal{F}$ as function and $S_{0,0} = s$ as the tree root (see Sections 3.1 and 4.1 for details). The user $\mathcal{U}$ now possesses a wallet determined by the set $(s, u, w, \Sigma, \sigma)$.

$\mathcal{U}$		$\mathcal{B}$
Choose s', w at random $C' = \mathsf{Commit}(s', u, w)$ $U = PK(\alpha, \beta, \gamma : pk_{\mathcal{U}} = g^\alpha \wedge C' = \mathsf{Commit}(\beta, \alpha, \gamma))$		
	$\xrightarrow{C', U, pk_{\mathcal{U}}}$	Verify U Choose s'' at random $C = \mathsf{Commit}(s = s' + s'', u, w)$ $\Sigma = \mathtt{CLSign}(C)$
	$\xleftarrow{s'', \Sigma}$	
$\mathtt{Verif}(\Sigma, (s = s' + s'', u, w)) \stackrel{?}{=} 1$ Compute E_ℓ and $S_0, \cdots, S_{\ell-1}$ using $\mathcal{F}$ and $S_{0,0} = s$ $M = \mathcal{H}(S_0)^{E_\ell/e_0} \cdots \mathcal{H}(S_{\ell-1})^{E_\ell/e_{\ell-1}}$ Choose r at random $\tilde{M} = r^{E_\ell} M \pmod n$		
	$\xrightarrow{\tilde{M}, E_\ell}$	$\tilde{\sigma} = \tilde{M}^{1/E_\ell} \pmod n$
	$\xleftarrow{\tilde{\sigma}}$	
$\sigma = \tilde{\sigma}/r \pmod n$ $W = (s, u, w, \Sigma, \sigma)$		$\mathcal{V}_{\mathcal{B}}^{\mathsf{Withdraw}} = (C, pk_{\mathcal{U}}, U, s'', \Sigma, E_\ell, \tilde{M}, \tilde{\sigma})$

Fig. 4. Withdrawal Protocol

4.4 The Spend Protocol

Assume that a user $\mathcal{U}$ owns a wallet $(s, u, w, \Sigma, \sigma)$ and wants to spend k coins to a merchant $\mathcal{M}$. The spend protocol works as follows:

1. $\mathcal{M}$ sends some public information *info* concerning the transaction (typically the time and date of the ongoing transaction);
2. $\mathcal{U}$ knows the smallest i such that $S_i, \cdots, S_{i+k-1}$ are unspent serial numbers;
3. $\mathcal{U}$ does not need to compute the values of the serial numbers $S_i, \cdots, S_{i+k-1}$. Indeed, she only needs to compute the smallest number of master serial numbers necessary to allow the computation by the merchant of $S_i, \cdots, S_{i+k-1}$. In the worst case, we need $2\lfloor \log k \rfloor$ values $S_{i_1,j_1}, \ldots, S_{i_n,j_n}$, $0 \leq i_1, \ldots, i_n$ and $0 \leq j_1 \leq 2^{i_1} - 1, \ldots, 0 \leq j_n \leq 2^{i_n} - 1$. $\mathcal{U}$ sends to the merchant $S_{i_1,j_1}, \ldots, S_{i_n,j_n}$ and the index value i;
4. using the batch RSA signature described in Section 3.1, $\mathcal{U}$ computes the batch signature $\sigma_{[i,i+k-1]}$ on $S_i, \cdots, S_{i+k-1}$ (further denoted σ_k);
5. $\mathcal{U}$ computes $R = \mathcal{H}(info||pk_{\mathcal{M}}||\sigma_k)$ which is used as a freshness indicator;
6. next $\mathcal{U}$ computes two values $C_1 = \mathsf{Enc}_{\mathcal{J}}(pk_{\mathcal{U}})$ and $C_2 = \mathsf{Enc}_{\mathcal{J}}(s)$;
7. $\mathcal{U}$ produces a signature of knowledge Π which proves that:
   - C_1 and C_2 are well-formed, that is C_1 is an encryption of $pk_{\mathcal{U}} = g^u$ and C_2 is an encryption of s under the judge's public key encryption scheme, without revealing $pk_{\mathcal{U}}$ nor s;
   - $\mathcal{U}$ knows a CL bank's signature Σ on u, s and w without revealing u, s, w nor σ.

She uses $c = \mathcal{H}(S_{i_1,j_1}\| \dots \|S_{i_n,j_n}\|\sigma_k\|R\|C_1\|C_2)$ as a challenge;

8. at the end, the user has sent $(i, S_{i_1,j_1}, \dots, S_{i_n,j_n}, \sigma_k, C_1, C_2, \Pi, R)$;
9. the merchant $\mathcal{M}$ computes $S_i, \cdots, S_{i+k-1}$ from $S_{i_1,j_1}, \dots, S_{i_n,j_n}$ and checks the validity of the coin by verifying the validity of σ_k and Π;

4.5 Deposit Protocol

During this step, a merchant $\mathcal{M}$ sends to the bank $\mathcal{B}$ the values $(i, S_i, \dots, S_{i+k-1}, \sigma_k, C_1, C_2, \Pi, R)$. The bank checks the validity of the spending by verifying the batch signature σ_k on the values $S_i, \dots, S_{i+k-1}$ using the index i, and the validity of the proof Π using R, C_1 and C_2. If the spending is valid, the bank checks whether at least one of the serial numbers $S \in \{S_i, \dots, S_{i+k-1}\}$ is already in its database. If not, $\mathcal{B}$ adds them into the database. Otherwise, the bank verifies the freshness of the spending using the value R. If it is fresh, the bank asks the judge to execute the identification of double spender procedure. Otherwise, the merchant is a cheater and the bank rejects the deposit.

4.6 Identification of Double Spender and Verification of Guilt

In this procedure, the bank sends to the judge two spendings $(i, S_i, \dots, S_{i+k-1}, \sigma_k, C_1, C_2, \Pi, R)$ and $(i', S'_{i'}, \dots, S'_{i'+k'-1}, \sigma'_{k'}, C'_1, C'_2, \Pi', R')$ such that there exists i_0 and i'_0 with $i \le i_0 \le i+k-1$ and $i' \le i'_0 \le i'+k'-1$ with $S_{i_0} = S'_{i'_0} = S$. This latter verifies the validity of both spendings, decrypts C_2 and C'_2 to retrieve s and s', and next decrypts C_1 and/or C'_1 if necessary.

- If S cannot be computed from s (resp. s'), then the judge decrypts C_1 (resp. C'_1) and concludes that $pk_{\mathcal{U}}$ (resp. $pk_{\mathcal{U}'}$) is guilty.
- Else, with high probability $s = s'$ (since $\mathcal{H}$ and $\mathcal{F}$ are collision-free) and $pk_{\mathcal{U}} = pk_{\mathcal{U}'}$ (since it is unlikely that two different users obtain the same wallet secret s in the withdrawal phase and since $\mathcal{F}$ is collision-free). Thus, the judge concludes that $pk_{\mathcal{U}} = pk_{\mathcal{U}'}$ is guilty. Note that if the case $s = s'$ and $pk_{\mathcal{U}} \neq pk_{\mathcal{U}'}$ happens, that means that user $\mathcal{U}$ has proven the knowledge of a bank's signature on the values (s, u) and user $\mathcal{U}'$ has proven the knowledge of a bank's signature on the values (s, u'). In this case, the two spendings are valid and the judge sends back a false alarm message since there is no double-spending.
- At the end, the judge produces a proof Π_G that the public key of the guilty user has been correctly decrypted. The proof consists of the values (s and $pk_{\mathcal{U}}$) related to the cheater and of a ZKPK that the secret key $sk_{\mathcal{J}}$ embedded in $pk_{\mathcal{J}}$ has correctly been used to decrypt s and $pk_{\mathcal{U}}$.

The verification of guilt consists in verifying the judge's proof Π_G on $pk_{\mathcal{U}}$ and s.

5 Security Analysis

In this section, we give the security arguments for our construction. We first detailed the security assumptions we use and next give the security theorem; security proofs are not included in the paper due to space restrictions.

5.1 Security Assumptions

One-More Unforgeability. In 2001, Bellare et al. [3] introduced the notion of *one-more one-way function*, and showed how it leads to a proof of security of Chaum's RSA-based blind signature scheme [14] in the random oracle model. We now introduce a variant of the one-more RSA problem in order to prove the security of the Batch variant of Chaum's blind signatures. The one-more flexible (or strong) RSA-problem is defined by the following game for an algorithm $\mathcal{A}$.

- the adversary $\mathcal{A}$ gets an RSA modulus n and a public exponent E made of the product of ℓ prime numbers $E = e_0 \dots e_{\ell-1}$;
- it is given access to an *inversion* oracle that given $y \in \mathbb{Z}_n^*$ returns $x \in \mathbb{Z}_n^*$ such that $x^E = y \mod N$;
- it is given access to a *challenge* oracle that returns ℓ random challenges point from $\mathbb{Z}_n^*$;
- eventually, $\mathcal{A}$ wins the game if it succeeds in inverting $q \cdot \ell + 1$ points output by the challenge oracle using less than q queries to the inversion oracle[3].

The *strong one-more RSA assumption* states that no probabilistic polynomial-time algorithm $\mathcal{A}$ may win the previous game with non-negligible probability.

Following, Bellare *et al.*'s technique from [3], it is readily seen that in the random oracle model, the Batch-RSA blind signature scheme is one-more unforgeable under the *strong one-more RSA assumption*:

Lemma 1. *If the one-more flexible RSA problem is hard, then the Batch-RSA blind signature scheme is polynomially-secure against one-more forgery in the random oracle model.*

Proof. It is almost identical to the one of [3, Theorem 16]. □

Strong Blindness Property. In the security proof of our e-cash system, we need a *Strong Blindness* property for this Batch-RSA blind signature scheme. More precisely, we have the following experiment:

- let $\mathcal{A}$ be a PPT Turing Machine having access to the signer's key pair and being able to participate to the blind process from the signer's point of view, obtain resulting message/signature (M, σ) and obtain chosen partial pairs message/signature, that is all $S_i \in F$ and the signature $\prod_{i \in F} \mathcal{H}(S_i)^{1/e_i}$ for any $F \subset \{0, \cdots, \ell - 1\}$ of the adversary's choice (see Section 3.1 for details);
- at any time of the game, the adversary outputs two transcripts I_0 and I_1 of a blind signature process (from the signer's point of view) and a challenge $\tilde{F} \subset \{0, \cdots, \ell-1\}$. The challenger next chooses at random a bit $b \in \{0, 1\}$ and outputs the messages and the signature corresponding to the transcript I_b and the set $\tilde{F}$;
- the adversary finally outputs a bit b'.

The *Strong Blindness* property says that the probability that $b' = b$ differs significantly from 1/2 is negligible.

Lemma 2. *The Batch-RSA Blind signature scheme unconditionally verifies the Strong Blindness property.*

Proof. Straightforward as the proof is similar to the security proof of the initial RSA blind signature scheme, which is unconditionally blind. □

[3] Using q times the inversion oracle and the batch RSA technique given in Section 3.1, the adversary can easily invert $q \cdot \ell$ points.

Unforgeability of signature of knowledge. In our construction, we use the Fiat-Shamir heuristic to make non-interactive traditional interactive zero-knowledge proofs of knowledge. In [24], Pointcheval and Stern prove that this transformation is secure in the random oracle model.

Camenisch-Lysyanskaya type signature schemes. We need the CL type signature scheme to be unforgeable, saying that even if an adversary has oracle access to the signing algorithm which provides signatures on messages of the adversary's choice, the adversary cannot create a valid signature on a message not explicitly queried. If we choose the CL signature scheme in [10], we need to assume that the flexible RSA problem is hard.

The One-more discrete logarithm assumption. The one-more discrete logarithm problem [3] is the following one. Given $l + 1$ values and having access to a discrete logarithm oracle at most l times, find the discrete logarithm of all these values.

5.2 Security Statement

Theorem 1. *Our e-cash system is a secure fair e-cash system:*

- *unforgeability under the one-more unforgeability of the Batch-RSA blind signature scheme and the non-malleability of the signature of knowledge, in the random oracle model;*
- *anonymity under the strong blindness of the Batch-RSA blind signature scheme and the indistinguishability of the encryption scheme, in the random oracle model;*
- *identification of double-spenders under the unforgeability of the CL signature scheme, in the random oracle model;*
- *exculpability under the one-more discrete logarithm assumption, in the random oracle model.*

Note that our construction does not provide a perfect anonymity property since it is possible to know which leaves in the serial number binary tree are used during the spending. For example, if two spendings are from the same part of the tree, everyone can conclude that the spendings are from different wallets.

6 Efficiency Considerations

In order to simplify the complexity statements, we consider $\ell = K$, so that the exponents used for a wallet are the first $K = 2^L$ prime numbers; we have $\log E \sim K \ln K$. The coins are spent following the decreasing order of exponents. We denote by E' the product of exponents corresponding to the number K' of coins remaining in the wallet. As seen in Section 4, the data transfer size is always at least $\mathcal{O}(\lambda \log k)$.

Using Batch RSA as described in Section 3.1 as our default variant ($V0$) for the scheme yields the following efficiency trade-off: only the highest remaining exponent and one aggregated signature have to be stored in the wallet, with storage size $\mathcal{O}(\log n)$. During the spending phase, a binary tree has to be rebuilt, requiring $\mathcal{O}(\log K' \log E') = \mathcal{O}(K' \log^2 K' + \log n)$ multiplications, and the current signature has to be broken up in two pieces, which costs $\mathcal{O}(1)$ modular divisions plus $\mathcal{O}(\log E') = \mathcal{O}(K' \log K')$ modular multiplications. At last, a single aggregated signature is sent to the merchant, together with the number of coins and the biggest exponent, thus requiring transfer of $\mathcal{O}(\log n)$ bits. As this variant is targeted at reduced storage, it is relevant to store also the root serial number only and compute the needed serial numbers at each spending, thus minimizing the storage cost.

Instead of reducing the storage cost, we can also manage the Batch RSA tree similarly to the tree of serial numbers. This yields variant $(V1)$: we store the initial withdrawal binary tree so that, during the spending, the user sends the aggregated signatures corresponding to the nodes of the tree closest to the root and such that all the corresponding leaves are in the spending set. The whole binary tree is stored, hence the initial storage size is $\mathcal{O}(K \log n)$. During the spending phase, the user needs to send at most $2\lfloor \log_2(k+1) \rfloor$ aggregated signatures corresponding to tree nodes to the merchant, hence a data transfer of size $\mathcal{O}(\log n \log k)$. The computational cost for the user is the cost of retrieving the aggregated signatures corresponding to the nodes spent and to their remaining counterparts. At most, this requires $\mathcal{O}(\log K)$ signature break-ups (in case single coins must be retrieved), each of which costs $\mathcal{O}(1)$ modular divisions plus at most (for nodes closest to the tree root) $\mathcal{O}(\log E') = \mathcal{O}(K' \log K')$ modular multiplications. However, these values can be pre-computed off-line after the withdrawal of the wallet, and stored in the tree, thus achieving a $\mathcal{O}(1)$ on-line computational cost. This variant aims at reducing computations during spending, so it is relevant to store also the whole serial number tree in order to retrieve the needed serial numbers at each spending in $\mathcal{O}(1)$.

The relative storage, spending computational complexity and data transfer size of our schemes are summed up in Table 1; M and D are the respective costs of exponentiation, multiplication and division modulo n, F is the cost of derivation with function $\mathcal{F}$, λ is a security parameter, K is the number of withdrawn coins, k the number of spent coins and K' the number of remaining coins in the wallet after spending. They take into account the complexities related to the serial numbers mentioned in Section 4, which provides the overall picture as the proof Π and the remaining data only have a constant complexity.

	Default variant (V0)	Variant (V1)
Wallet storage size	$\mathcal{O}(\lambda + \log n)$	$\mathcal{O}(K(\lambda + \log n))$
Computational complexity of spending	$\mathcal{O}(K' \log^2 K' + \log n)M$ $+\mathcal{O}(1)D + \mathcal{O}(\log k)F$	$\mathcal{O}(1)$
Spending transfer size	$\mathcal{O}(\lambda \log k + \log n)$	$\mathcal{O}((\lambda + \log n) \log k)$

Table 1. Efficiency trade-offs.

References

1. M. H. Au, W. Susilo, and Y. Mu, *Practical Anonymous Divisible E-Cash from Bounded Accumulators.*, Financial Cryptography (G. Tsudik, ed.), Lect. Notes Comput. Sci., vol. 5143, Springer, 2008, pp. 287–301.
2. M. H. Au, Q. Wu, W. Susilo, and Y. Mu, *Compact E-Cash from Bounded Accumulator.*, Topics in Cryptology - CT-RSA 2007 (M.Abe, ed.), Lect. Notes Comput. Sci., vol. 4377, Springer, 2007, pp. 178–195.
3. M. Bellare, C. Namprempre, D. Pointcheval, and M. Semanko, *The One-More-RSA-Inversion Problems and the Security of Chaum's Blind Signature Scheme.*, J. Cryptology **16** (2003), no. 3, 185–215.
4. F. Boudot and J. Traoré, *Efficient Publicly Verifiable Secret Sharing Schemes with Fast or Delayed Recovery.*, in Varadharajan and Mu [27], pp. 87–102.
5. C. Boyd, E. Foo, and C. Pavlovski, *Efficient Electronic Cash Using Batch Signatures.*, in Pieprzyk et al. [23], pp. 244–257.
6. S. Brands, *Untraceable Off-line Cash in Wallets with Observers (Extended Abstract).*, Advances in Cryptology - Crypto'93 (D. R. Stinson, ed.), Lect. Notes Comput. Sci., vol. 773, Springer, 1994, pp. 302–318.
7. S. Brands and D. Chaum, *Distance-Bounding Protocols (Extended Abstract).*, in Helleseth [20], pp. 344–359.
8. J. Camenisch, S. Hohenberger, and A. Lysyanskaya, *Compact E-Cash.*, Advances in Cryptology - Crypto 2005 (V. Shoup, ed.), Lect. Notes Comput. Sci., vol. 3621, Springer, 2005, pp. 302–321.

9. J. Camenisch and M. Kiayias, A.and Yung, *On the Portability of Generalized Schnorr Proofs.*, Advances in Cryptology - Eurocrypt 2009 (Antoine Joux, ed.), Lect. Notes Comput. Sci., vol. 5479, Springer, 2009, pp. 425–442.
10. J. Camenisch and A. Lysyanskaya, *A Signature Scheme with Efficient Protocols.*, Third Conference on Security in Communication Networks, SCN 2002 (S. Cimato, C. Galdi, and G. Persiano, eds.), Lect. Notes Comput. Sci., vol. 2576, Springer, 2003, pp. 268–289.
11. S. Canard, I. Coisel, and J. Traoré, *Complex Zero-Knowledge Proofs of Knowledge Are Easy to Use.*, Provable Security, First International Conference, ProvSec 2007 (W. Susilo, J. K. Liu, and Y. Mu, eds.), Lect. Notes Comput. Sci., vol. 4784, Springer, 2007, pp. 122–137.
12. S. Canard, A. Gouget, and É Hufschmitt, *A Handy Multi-coupon System.*, Applied Cryptography and Network Security, ACNS 2006 (J. Zhou, M. Yung, and F. Bao, eds.), Lect. Notes Comput. Sci., vol. 3989, 2006, pp. 66–81.
13. D. Chaum, *Blind Signatures for Untraceable Payments.*, Advances in Cryptology - Crypto'82 (D. Chaum, R. L. Rivest, and A. T. Sherman, eds.), Plenum Press, New York, 1983, pp. 199–203.
14. ———, *Blind Signature System.*, Advances in Cryptology - Crypto'83 (D. Chaum, ed.), Plenum Press, New York, 1984, p. 153.
15. D. Chaum, A. Fiat, and M. Naor, *Untraceable Electronic Cash.*, Advances in Cryptology - Crypto'88 (S. Goldwasser, ed.), Lect. Notes Comput. Sci., vol. 403, Springer, 1990, pp. 319–327.
16. N. Ferguson, *Single Term Off-Line Coins.*, in Helleseth [20], pp. 318–328.
17. A. Fiat, *Batch RSA.*, J. Cryptology **10** (1997), no. 2, 75–88.
18. A. Fiat and A. Shamir, *How to Prove Yourself: Practical Solutions to Identification and Signature Problems.*, Advances in Cryptology - Crypto'86 (A. M. Odlyzko, ed.), Lect. Notes Comput. Sci., vol. 263, Springer, 1987, pp. 186–194.
19. M. Girault, P. Poupard, and J. Stern, *On the Fly Authentication and Signature Schemes Based on Groups of Unknown Order.*, J. Cryptology **19** (2006), no. 4, 463–487.
20. T. Helleseth (ed.), *Advances in Cryptology - Eurocrypt'93, Workshop on the Theory and Application of of Cryptographic Techniques, Lofthus, Norway, May 23-27, 1993, Proceedings*, Lect. Notes Comput. Sci., vol. 765, Springer, 1994.
21. A. Kiayias, Y. Tsiounis, and M. Yung, *Traceable Signatures.*, Advances in Cryptology - Eurocrypt 2004 (C. Cachin and J. Camenisch, eds.), Lect. Notes Comput. Sci., vol. 3027, Springer, 2004, pp. 571–589.
22. C. Pavlovski, C. Boyd, and E. Foo, *Detachable Electronic Coins.*, in Varadharajan and Mu [27], pp. 54–70.
23. J. Pieprzyk, R. Safavi-Naini, and J. Seberry (eds.), *Information security and privacy, 4th australasian conference, acisp'99, wollongong, nsw, australia, april 7-9, 1999, proceedings*, Lect. Notes Comput. Sci., vol. 1587, Springer, 1999.
24. D. Pointcheval and J. Stern, *Security Arguments for Digital Signatures and Blind Signatures.*, J. Cryptology **13** (2000), no. 3, 361–396.
25. C. P. Schnorr, *Efficient signature generation by smart cards.*, J. Cryptology **4** (1991), no. 3, 161–174.
26. J. Traoré, *Group Signatures and Their Relevance to Privacy-Protecting Off-Line Electronic Cash Systems.*, in Pieprzyk et al. [23], pp. 228–243.
27. V. Varadharajan and Y. Mu (eds.), *Information and Communication Security, Second International Conference, ICICS'99, Sydney, Australia, November 9-11, 1999, Proceedings*, Lect. Notes Comput. Sci., vol. 1726, Springer, 1999.