

Anonymous Transferable E-Cash

Foteini Baldimtsi^{1*}, Melissa Chase², Georg Fuchsbauer^{3**}, Markulf Kohlweiss²

¹ Computer Science Department, Brown University

`foteini@cs.brown.edu`

² Microsoft Research

`{melissac, markulf}@microsoft.com`

³ Institute of Science and Technology Austria

`georg.fuchsbauer@ist.ac.at`

Abstract. Cryptographic e-cash allows off-line electronic transactions between a bank, users and merchants in a secure and anonymous fashion. A plethora of e-cash constructions has been proposed in the literature; however, these traditional e-cash schemes only allow coins to be transferred once between users and merchants. Ideally, we would like users to be able to transfer coins between each other multiple times before deposit, as happens with physical cash.

“Transferable” e-cash schemes are the solution to this problem. Unfortunately, the currently proposed schemes are either completely impractical or do not achieve the desirable anonymity properties without compromises, such as assuming the existence of a trusted “judge” who can trace all coins and users in the system. This paper presents the first efficient and fully anonymous transferable e-cash scheme without any trusted third parties. We start by revising the security and anonymity properties of transferable e-cash to capture issues that were previously overlooked. For our construction we use the recently proposed malleable signatures by Chase et al. to allow secure and anonymous transferring of the coins. Finally, we propose an independent, efficient double-spending detection mechanism and discuss an instantiation of our construction.

Keywords: Electronic payments, transferable e-cash, malleable signatures, double-spending detection.

1 Introduction

Electronic payment systems are everywhere and average users take their two major properties, security and privacy, for granted even though they may be built on shaky foundations. Payments made with debit or credit cards do not provide any privacy guarantee for users since the corresponding financial institution can track all their transactions. Starting with Chaum [Cha83], the cryptographic

* Work done as an intern in Microsoft Research Redmond and as a student at Brown University, where supported by NSF grant 0964379.

** Supported by the European Research Council, ERC Starting Grant (259668-PSPC).

community has worked on electronic analogues to physical money (e-cash) that guarantee secure and private payments [Cha83,CFN88,Bra93,CHL05,BCKL09]. A typical e-cash system consists of three types of entities: the Bank, users and merchants. Users withdraw electronic coins from the Bank and spend them to merchants, who finally deposit them back to the Bank. E-cash systems should satisfy two main properties (1) *unforgeability*: an adversarial user cannot spend more e-coins than he withdrew; and (2) *anonymity*: nobody (including the Bank) can link spending transactions to each other or to specific withdrawal instances.

Unlike physical cash, electronic coins are easy to duplicate, so a mechanism ensuring that a user cannot spend one coin multiple times is needed. Two solutions were proposed in the literature: the first is *online* e-cash [Cha83], in which the merchants are constantly connected to the Bank and can therefore check whether a coin has already been deposited before accepting it. In order to overcome the strong requirement of a permanently online Bank, a second solution is to use a *double-spending* mechanism [CFN88]. The idea is that as long as a user is honest, his anonymity is guaranteed, but if he tries to cheat the system by spending one e-coin multiple times, his identity is revealed.

Unfortunately, in traditional e-cash users can only transfer their coins to merchants, who must then deposit them to the Bank. It would be natural to allow users to transfer coins to other users (or merchants), who should be able to further transfer the received coins, and so on. Moreover, it would be desirable if these transfers could be done without being connected to the bank, i.e., offline. One of the main advantages of such a transferability property is that it would decrease the communication cost between the Bank and the users. Moreover, it would allow for more real-world scenarios. Consider the example of coins of different denominations. A store, which is offline, wants to give back change to a customer, using coins previously received. In order to do so, coins need to be transferable multiple times. Transferability of e-cash was proposed in the 1990s and the desired security properties have been analyzed; however, all schemes proposed so far do not satisfy the proposed security and privacy requirements, or they are only of theoretical interest, such as [CG08].

Arguably, this was partly because e-cash fell out of fashion as it became clear that traditional banks were unlikely to support cryptographic currencies and that credit cards and centralized payment services offering little privacy are broadly accepted for online payments. Recently, with bitcoin [Nak08] demonstrating how to bypass the banks, there has been renewed interest in e-cash, as existing techniques from anonymous e-cash are likely to be applicable to the bitcoin world as well [MGGR13,BCG⁺14].

Related work. Transferable e-cash was originally proposed by Okamoto and Ohta [OO89,OO91], who gave e-cash schemes that satisfy various properties such as divisibility and transferability but only provide weak levels of anonymity. While an adversary cannot link a withdrawal to a payment, he can link two payments by the same user; this property was called *weak anonymity* (WA). Chaum and Pedersen [CP92] proved that (1) transferred coins have to grow in size and (2) an unbounded adversary can always recognize coins he owned

when seeing them spent later. Moreover, they extended the scheme due to van Antwerpen [vAE90] to allow coin transfer. The resulting scheme satisfies *strong anonymity* (SA), guaranteeing that an adversary cannot decide whether two payments were made by the same user. However, he can recognize coins he observed in previous transactions. Strong anonymity is also satisfied by the schemes constructed in [Bla08,CGT08].

Anonymity for transferable e-cash has been a pretty subtle notion to define. In 2008 Canard and Gouget [CG08] gave the first formal treatment of anonymity properties for transferable e-cash. In addition to weak and strong anonymity, which do not yield the guarantees one would intuitively expect, they defined *full anonymity* (FA): an adversary, impersonating the bank, cannot link a coin previously (passively) observed to a coin he receives as a legitimate user (Observe-then-Receive). They also define *perfect anonymity* (PA): an adversary, impersonating the bank, cannot link a coin previously owned to a coin he receives. They showed that $PA \Rightarrow FA \Rightarrow SA \Rightarrow WA$. Chaum and Pedersen [CP92] showed that perfect anonymity cannot be achieved against unbounded adversaries. Canard and Gouget [CG08] prove that it cannot be achieved against bounded adversaries either. They therefore introduce two modifications of perfect anonymity, which are incomparable to FA, namely PA1: an adversary, impersonating the bank, cannot link a coin previously owned to a coin he passively observes being transferred between two honest users (Spend-then-Observe); and PA2 (Spend-then-Receive): an adversary cannot link a coin previously owned to a coin he receives, assuming the bank is honest (If the adversary could impersonate the bank, the notion is not achievable due to the impossibility results mentioned above.) In the same paper they present a construction which satisfies all achievable anonymity properties, but is only of theoretical interest due to its inefficiency as it relies on metaproofs and thus Cook-Levin reductions.

The first practical scheme that satisfies FA, PA1 and PA2 is the scheme due to Fuchsbauer et al. [FPV09]; however, it has two main drawbacks: (1) the users have to store the data of all transactions they were involved in to prove innocence in case of fraud; and even worse (2) when a double-spending is detected, all users up to the double-spender lose their anonymity. Blazy et al. [BCF⁺11] addressed these problems and proposed a new scheme using commuting signatures [Fuc11], which overcomes the above drawbacks by assuming the existence of a trusted entity called the *judge*. This entity is responsible for the tracing of double-spenders, but can also trace all the coins and users in the system at any time. This clearly contradicts one of the main goals of e-cash: as long as users do not double-spend, they remain anonymous. (In addition, it is not clear whether their scheme satisfies PA2; see Section 4.4.)

Our contributions. We present the first transferable e-cash scheme that satisfies all of the anonymity properties from the literature (FA, PA1, PA2) and a new anonymity notion. Moreover, it does not assume any trusted party and does not rely on a Cook-Levin reduction or heuristics like the random-oracle model. Our contributions include new definitions, a construction based on malleable signatures and an independent double-spending detection mechanism.

Definitions. We provide a formal treatment of the security and anonymity properties of transferable e-cash in a game-based fashion, since many of the previous definitions were informal and/or incomplete. Moreover, we define a new anonymity requirement that was not captured before. Namely, we introduce a strengthening of Spend-then-Receive anonymity (a.k.a. PA2), which guarantees that an adversary, impersonating the bank—although able to link a coin that he previously owned to one he receives—should not be able to tell anything about the honest users who possessed the coin in between. This has not been guaranteed in any previous definitions.

Construction. In traditional e-cash systems a coin withdrawn from the bank typically consists of the bank’s signature on a unique serial number, SN. When spending the coin with a merchant, a double-spending tag DS is computed, which encodes the identity of the spender. The merchant then deposits $c = (\text{SN}, \sigma, \text{DS})$ at the bank. If two coins, c, c' , with the same serial number, but with different double-spending tags DS, DS’ are deposited, these tags together will reveal the identity of the user who double-spent. For transferable e-cash, the owner of a coin should be able to transfer the coin/signature he received from the bank to another user in such a way that the transferred coin is valid, carries all the information necessary to detect double-spending and preserves anonymity. Thus, we need a digital signature scheme that allows a user to compute a “fresh” version of a valid signature (unlinkable to the original one to ensure anonymity) and to extend the current signature to include more information (such as a double-spending tag for the new owner).

A recent proposal of a signature scheme that satisfies the above properties is due to Chase et al. [CKLM14]. They propose *malleable* signatures, an extension of digital signatures, where anyone can transform a signature on a message m into a signature on m' , as long as $T(m) = m'$ for some allowed transformation T . In our construction, a coin withdrawn by the bank is signed using a malleable signature scheme. Whenever a user wishes to transfer a coin to another user he computes a malleable signature on a valid transformation of the coin. A valid transformation guarantees that the transferred coin is valid, it is indeed owned by the sender (i.e. the sender’s secret key corresponds to the information encoded in the coin) and the new coin/signature created will encode the right information of the receiver. The serial number and the double-spending tags are encrypted under the bank’s public key, allowing it to check for double-spending on deposit. Moreover, the encryptions are re-randomized in every transfer, which ensures anonymity. We propose an instantiation in the full version that can be proved secure under standard assumptions (Decision Linear and the Symmetric External Decision Diffie-Hellman (SXDH)).

Double Spending Detection. Double spending detection for transferable e-cash is a complex issue, since it needs to ensure that the right user is accused while the anonymity of honest owners of the coin will be preserved. We propose an efficient double-spending detection mechanism, which is independent of our scheme and could be used by other transferable e-cash constructions, e.g., to provide an

offline payment mechanism for users holding a sufficient amount of bitcoins as a deposit.

2 Definitions for Transferable E-Cash

We adapt the definitions for transferable e-cash given by [CG08,BCF⁺11] and strengthen them in several aspects; in particular, we introduce a stronger anonymity notion.

In a transferable e-cash scheme there are two types of parties: the bank \mathcal{B} and users \mathcal{U}_i . Coins are denoted by c and each coin is uniquely identifiable via a serial number SN, which will be retrieved by the bank during deposit to check for double-spending. We let \mathcal{DCL} denote the list of deposited coins; if multiple coins with the same serial number were deposited, we keep all of them in \mathcal{DCL} .

Note that in contrast to previous versions of these definitions we add a protocol for user registration and we also merge the **Deposit** and **Identify** protocols. A transferable e-cash scheme consists of the following algorithms (which are probabilistic unless otherwise stated):

ParamGen(1^λ) on input the security parameter λ outputs the system parameters par (we assume that λ can be deduced from par , which is a default input to the remaining algorithms).

KeyGen() and **UKeyGen**() are executed by \mathcal{B} and a user \mathcal{U} respectively and output $(sk_{\mathcal{B}}, pk_{\mathcal{B}})$ and $(sk_{\mathcal{U}}, pk_{\mathcal{U}})$. The bank's key $sk_{\mathcal{B}}$ might be divided into two parts: $sk_{\mathcal{W}}$ for the registration and withdrawal phase and $sk_{\mathcal{D}}$ for the deposit phase. Thus, we define two extra algorithms **WKeyGen**() and **DKeyGen**() for the bank's key generation. We also assume that during the bank key generation the list \mathcal{DCL} is initialized to be empty.

Registration($\mathcal{B}[sk_{\mathcal{W}}, pk_{\mathcal{U}}], \mathcal{U}[sk_{\mathcal{U}}, pk_{\mathcal{B}}]$) is a protocol between the bank and a user. At the end the user receives a certificate $cert_{\mathcal{U}}$; both parties output either *ok* or \perp in case of error.

Withdraw($\mathcal{B}[sk_{\mathcal{W}}, pk_{\mathcal{U}}], \mathcal{U}[sk_{\mathcal{U}}, pk_{\mathcal{B}}]$) is a protocol between the bank and a user. The user either outputs a coin c or \perp . \mathcal{B} 's output is *ok* or \perp .

Spend($\mathcal{U}_1[c, sk_{\mathcal{U}_1}, cert_{\mathcal{U}_1}, pk_{\mathcal{B}}], \mathcal{U}_2[sk_{\mathcal{U}_2}, pk_{\mathcal{B}}]$) is a protocol in which \mathcal{U}_1 spends/transfers the coin c to \mathcal{U}_2 . At the end, \mathcal{U}_2 either outputs a coin c' and *ok* or it outputs \perp ; \mathcal{U}_1 either marks the coin c as spent and outputs *ok*, or it outputs \perp in case of error.

Deposit($\mathcal{U}[c, sk_{\mathcal{U}}, cert_{\mathcal{U}}, pk_{\mathcal{B}}], \mathcal{B}[sk_{\mathcal{D}}, pk_{\mathcal{U}}, \mathcal{DCL}]$) is a protocol where a user \mathcal{U} deposits a coin c at the bank. We split the deposit protocol into three sub-routines. First **CheckCoin** checks whether the coin c is consistent, and if not outputs \perp . Else, \mathcal{B} runs **CheckDS**, which outputs the serial number SN of the deposited coin. \mathcal{B} checks whether \mathcal{DCL} already contains an entry for SN. If not, \mathcal{B} adds SN to \mathcal{DCL} , credits \mathcal{U} 's account and returns "success" and \mathcal{DCL} . Otherwise, the coin was double-spent: the subroutine **DetectDS** is run on the two coins and outputs $(pk_{\mathcal{U}}, \Pi)$, where $pk_{\mathcal{U}}$ is the public key of the accused

user, and Π is a proof that the registered user who owns $pk_{\mathcal{U}}$ double-spent the coin. Note that Π should reveal nothing about the coin itself.

$\text{VerifyGuilt}(pk_{\mathcal{U}}, \Pi)$ is a deterministic algorithm that can be executed by anyone. It outputs 1 if the proof verifies and 0 otherwise.

Notice that in our definition a transferable e-cash scheme is *stateless* since there is no common state information shared between the algorithms. This means that a coin withdrawn will be the same, whether it was the first or the n -th coin the bank issues to a specific user. Moreover, when a user \mathcal{U}_2 receives a coin from a user \mathcal{U}_1 , then the transferred coin will only depend on the original coin (not on other coins received by \mathcal{U}_2 or coins transferred by \mathcal{U}_1). Thus, the bank and the users do not need to remember anything about past transactions—for transfer the coin itself must be sufficient.

In order to formally define the security properties of transferable e-cash, we first define some global variables and oracles, which will be used in the security games.

Global variables. In the **user list**, \mathcal{UL} , we store all information about users, keys and certificates. Its entries are of the form $(i, pk_i, sk_i, cert_i, uds_i)$, where uds indicates how many times user \mathcal{U}_i double-spent (this counter is used in the exculpability definition). If user i is corrupted (i.e. the adversary knows the secret key of this user) then $sk_i = \perp$; if it has not been registered then $cert_i = \perp$. We keep a counter, n , of the total number of generated/registered users which is initialized to 0.

In the **coin list**, \mathcal{CL} , we keep information about the coins created in the system. For each *original* coin withdrawn we store a tuple $(j, owner, c, fc, fd, cds, origin)$, where j is its index in \mathcal{CL} , $owner$ stores the index i of the user who withdrew the coin¹ and c is the coin itself. The flag fc indicates whether the coin has been corrupted² and the flag fd indicates whether the coin has been deposited. We also keep a counter, cds , of how many times this *specific instance* of the coin has been spent, which is initialized as $cds = 0$. In *origin* we write “ \mathcal{B} ” if the coin was issued by the honest bank and “ \mathcal{A} ” if the adversary issued it when impersonating the bank.

After a coin has been transferred, we add a new entry to \mathcal{CL} of the following format: $(j, owner, c, cds, pointer)$, where j is the position in \mathcal{CL} , $owner$ shows the current owner, c is the new, transferred coin and cds indicates how many times the coin has been spent. In *pointer* we store a pointer j' indicating which original coin this transferred coin corresponds to. Once a transferred coin is deposited or corrupted, we mark the original coin’s flags fc, fd appropriately. The last list is the **list of deposited coins**, \mathcal{DCL} .

We now define oracles used in the security definitions. If during the oracle execution an algorithm fails (outputs \perp) then the oracle also stops. Otherwise

¹ We do not store the coins withdrawn by the adversary.

² A *corrupted coin* is defined as a coin that was under the adversary’s control at some point. Once a coin is flagged as *corrupted*, it cannot be “un-flagged”, even if it is later under the control of an honest user.

the call to the oracle is considered *successful* (for the deposit oracles a successful call is one that also didn't detect any double-spending).

Creation, registration and corruption of users. The adversary can instruct the creation of honest users, and either play the role of the bank during registration, or passively observe registration:

Create() sets $n = n + 1$, executes $(sk_n, pk_n) \leftarrow \text{UKeyGen}()$, sets $\mathcal{UL}[n] = (n, pk_n, sk_n, \perp, 0)$ and outputs pk_n .

BRegister(pk) plays the bank side of the **Register** protocol and interacts with \mathcal{A} . Let $cert$ be the generated certificate. If $pk \notin \mathcal{UL}$ then set $n = n + 1$ and $\mathcal{UL}[n] = (n, pk, \perp, cert, 0)$; else abort.

URegister(i), for $i \leq n$, plays the user side of the **Register** protocol and adds $cert$ to the corresponding field of \mathcal{UL} .

Register(i), for $i \leq n$, simulates both sides of the **Register** protocol. If user i was not registered then add $cert$ to the corresponding field of \mathcal{UL} .

Corrupt(i, S), for $i \leq n$, allows the adversary to corrupt a subset, S , of user i 's coins. If $sk_i = \perp$ (i.e. this user is already corrupted) then abort. The set S consists of coin indices in \mathcal{CL} . For every $j \in S$ look up the j -th entry of \mathcal{CL} and if $owner \neq i$ then ignore this coin and remove it from S . The oracle first outputs sk_i and then updates \mathcal{UL} by setting $sk_i = \perp$ to mark this user as corrupted. Then, the *non-corrupted* coins in the set S are given to the adversary \mathcal{A} and are marked as corrupted i.e. the flag fc of the corresponding *original* coin is set $fc = 1$. Note that if \mathcal{A} tries to corrupt unregistered users, this doesn't give him any extra power. Also, once a user is corrupted he is considered to be an *adversarial* user and thus \mathcal{A} will be running instead of him³.

Withdrawal oracles. The adversary can either withdraw a coin from a trusted bank, play the role of the bank or passively observe a withdrawal.

BWith() plays the bank side of the **Withdraw** protocol. Note that coins belonging to \mathcal{A} are not added to the coin list \mathcal{CL} .

UWith(i) plays user i in a **Withdraw** protocol, where the bank is controlled by the adversary. Upon obtaining a coin c , it adds the entry $(j, owner = i, c, fc = 0, fd = 0, cds = 0, origin = \mathcal{A})$ to \mathcal{CL} .

With(i) simulates a complete **Withdraw** protocol execution playing both \mathcal{B} and user i . It adds $(j, owner = i, c, fc = 0, fd = 0, cds = 0, origin = \mathcal{B})$ to \mathcal{CL} and outputs the transcript.

Spend and deposit oracles.

Rcv(i) lets \mathcal{A} spend a coin to honest user i . It plays the role of \mathcal{U}_2 with user i 's secret key in the **Spend** protocol. A new entry $(j, owner = i, c, fc = 1, fd =$

³ This means that \mathcal{A} cannot run honest-user oracles on corrupted users: i.e. cannot run oracles **With**, **UWith**, **Rcv**, **S&R**.

$0, cds = 0, origin = \mathcal{A}$) is added to \mathcal{CL} . Coins received from the adversary are considered as *original* coins in \mathcal{CL} .

$\text{Spd}(j)$ enables \mathcal{A} to receive coin number j in \mathcal{CL} . If the coin belongs to a corrupted user it aborts. Otherwise, it plays the role of user \mathcal{U}_1 in the **Spend** protocol with the secret key of the owner i of the coin j in \mathcal{CL} . It increases the coin spend counter cds of entry j in \mathcal{CL} by 1. If cds was already greater than zero (i.e., this specific user has already spent this coin) then the double-spending counter, uds , of the owner of coin j is increased by one. Finally, whenever a coin is received by \mathcal{A} , we mark the *original* instance of this coin as corrupted, i.e., we set $fc = 1$.

$\text{S\&R}(i, j)$ is the Spend-and-Receive oracle that allows \mathcal{A} to passively observe the spending of coin j by its owner to user i (who must not be corrupted). It increases the current size ℓ of \mathcal{CL} by 1 and adds $(\ell, owner = i, c, cds = 0, pointer)$ to \mathcal{CL} , where $pointer = pointer_j$ if $pointer_j \notin \{\mathcal{A}, \mathcal{B}\}$, else $pointer = j$. It also increases the coin spend counter cds_j in entry j by 1. If cds_j was already greater than zero then the double-spending counter uds of the spender is also increased by 1.

$\text{BDepo}()$ simulates the bank in the **Deposit** protocol interacting with \mathcal{A} playing the role of a user. It updates \mathcal{DCL} accordingly.

$\text{UDepo}(j)$ simulates the role of the owner (who must not be corrupted) of coin j in the **Deposit** protocol, interacting with the adversary playing the bank. It increases the spend counter cds_j in entry j in \mathcal{CL} by 1. If cds_j was already greater than zero then the double-spending counter uds of the owner of coin j is increased by 1. It also marks $fd = 1$ for the original coin.

$\text{Depo}(j)$ simulates a **Deposit** of coin j between the bank and the owner of j (who must not be corrupted). It increases cds_j in entry j of \mathcal{CL} by 1. If cds_j was already greater than zero then uds of the owner of coin j is increased by one. It also marks $fd = 1$ in the original coin and adds the coin to \mathcal{DCL} .

Let $\text{size}(c)$ be a function that outputs the size of a coin. A withdrawn coin has size 1 and after a transfer the size increases by 1. We say that coins c_1 and c_2 are *compatible*, $\text{comp}(c_1, c_2) = 1$, if $\text{size}(c_1) = \text{size}(c_2)$. We need this property, since transferred coins necessarily grow in size [CP92] and thus an adversary may break anonymity by distinguishing coins of different sizes.

2.1 Security Properties

We define the security properties of transferable e-cash by refining previous definitions by [CG08] and [BCF⁺11]. In the beginning of security games the challenger typically runs $par \leftarrow \text{ParamGen}(1^\lambda)$ and $(sk_B, pk_B) \leftarrow \text{BKeyGen}()$, which we merge into one algorithm **AllGen**.

Unforgeability. This notion protects the bank in that an adversary should not be able to spend more coins than the number of coins he withdrew. In [BCF⁺11] an adversary can interact with honest users and wins the unforgeability game

if he withdrew fewer coins than the number of coins he successfully deposited, i.e. no double spending was detected. We simplify the definition noticing that is not necessary for the adversary to create, corrupt or instruct *honest* users to withdraw, spend, receive and deposit, since the adversary could simulate these users itself. An unforgeability definition *without* honest user oracles implies thus the definition *with* these oracles given in [BCF⁺11]. Consider the following experiment:

Experiment $\mathbf{Expt}_A^{\text{unforg}}(\lambda)$;
 $(par, sk_B, pk_B) \leftarrow \mathbf{AllGen}(1^\lambda)$;
 $\mathcal{A}^{\mathbf{BRegister}, \mathbf{BWith}, \mathbf{BDepo}}(par, pk_B)$;
Let q_W, q_D be the number of successful calls to $\mathbf{BWith}, \mathbf{BDepo}$ respectively;
If $q_W < q_D$ then return 1;
Return \perp .

Definition 1 (Unforgeability). *A transferable e-cash system is unforgeable if for any PPT adversary \mathcal{A} , we have $\mathbf{Adv}_A^{\text{unforg}}(\lambda) := \Pr[\mathbf{Expt}_A^{\text{unforg}}(\lambda) = 1]$ is negligible in λ .*

Identification of double spenders. No collection of users can spend a coin twice (double-spend) without revealing one of their identities. Consider the following experiment on the left where, analogously to the unforgeability definition, we do not give the adversary access to honest user oracles since he can simulate them itself.

<p>Experiment $\mathbf{Expt}_A^{\text{ident}}(\lambda)$ $(par, sk_B, pk_B) \leftarrow \mathbf{AllGen}(1^\lambda)$; $\mathcal{A}^{\mathbf{BRegister}, \mathbf{BWith}, \mathbf{BDepo}}(par, pk_B)$; Let (pk_{i^*}, Π_G) be the output of the last call to \mathbf{BDepo}; Return 1 if any of the following hold: - $\mathbf{VerifyGuilt}(pk_{i^*}, \Pi_G) = 0$; - $pk_{i^*} \notin \mathcal{UL}$; - $pk_{i^*} \in \mathcal{UL}$ and $cert_i = \perp$; Return \perp.</p>	<p>Experiment $\mathbf{Expt}_A^{\text{excul}}(\lambda)$ $par \leftarrow \mathbf{ParamGen}(1^\lambda)$; $(pk_B, st) \leftarrow \mathcal{A}(par)$; $(i^*, \Pi^*) \leftarrow \mathcal{A}^{\mathbf{Create}, \mathbf{URegister}, \mathbf{Corrupt}, \mathbf{UWith}, \mathbf{Rcv}, \mathbf{Spd}, \mathbf{S\&R}, \mathbf{UDepo}}(st)$; If $\mathbf{VerifyGuilt}(pk_{i^*}, \Pi^*) = 1$ & $sk_{i^*} \neq \perp$ & $uds_{i^*} = 0$ then return 1; Return \perp.</p>
--	--

Definition 2 (Double-spender identification). *A transferable e-cash system is secure against double-spending if for any PPT adversary \mathcal{A} , $\mathbf{Adv}_A^{\text{ident}}(\lambda) := \Pr[\mathbf{Expt}_A^{\text{ident}}(\lambda) = 1]$ is negligible in λ .*

Note that we cannot detect double-spending unless both of the double-spent coins are deposited.

Exculpability. Exculpability ensures that the bank, even when colluding with malicious users, cannot wrongly accuse honest users of double-spending. Specifically, it guarantees that an adversarial bank cannot output a double-spending proof Π^* for an honest user with public key $pk_{\mathcal{U}}$ that verifies even though user \mathcal{U} has not double-spent. Our definition follows the one from [BCF⁺11], but we allow the adversary to generate the bank keys himself, thus modeling a malicious bank. The adversary must output the index of the user being accused of double spending and the corresponding proof. The game is formalized as $\mathbf{Expt}^{\text{excul}}$ above.

Definition 3 (Exculpability). *A transferable e-cash system is exculpable if for any PPT adversary \mathcal{A} , we have $\mathbf{Adv}_{\mathcal{A}}^{\text{excul}}(\lambda) := \Pr[\mathbf{Expt}_{\mathcal{A}}^{\text{excul}}(\lambda) = 1]$ is negligible in λ .*

In the full version we also discuss a stronger version of exculpability that guarantees that a user cannot be accused of double spending *more* coins than the ones he did double-spend.

2.2 Anonymity Properties

We first consider the three anonymity notions given in [CG08,BCF⁺11]:

Observe-then-Receive Full Anonymity (OtR-FA). The adversary, impersonating the bank, cannot link a coin he receives as an adversarial user or as the bank to a previously (passively) observed transfer between honest users. This covers both the case where the adversary receives a coin during a transfer or receives a coin when impersonating the bank during deposit.

Spend-then-Observe Full Anonymity (StO-FA). The adversary, impersonating the bank, cannot link a (passively) observed coin transferred between two honest users to a coin he has already owned as a “legitimate” user.

Spend-then-Receive Full Anonymity (StR-FA). When the bank is honest, the adversary cannot recognize a coin he previously owned when he receives it again.

These three notions are incomparable as proved in [CG08]. Due to lack of space, we present the formal experiments in the full version since they are relatively similar to those in [BCF⁺11]. A difference is that we define *coin* indistinguishability, which implies the *user* indistinguishability properties considered in [BCF⁺11]. We also allow \mathcal{A} to pick the secret keys himself, in particular that of the bank when impersonating it (in contrast to [CG08,BCF⁺11], where the bank’s keys are created by experiment).

Moreover, we introduce a new, strong, user-indistinguishability notion of anonymity that we call *Spend-then-Receive**: although the adversary, when impersonating the bank, can tell whenever he receives a coin he owned before, he should not be able to learn anything about the identities of the users that owned the coin in between. We define this as an indistinguishability game in which the adversary picks a pair of users, to one of which (according to a random bit b) the coins are transferred. The goal is to guess this bit b .

Experiment $\mathbf{Expt}_{\mathcal{A},b}^{\text{StR}^*-\text{fa}}(\lambda)$

$par \leftarrow \text{ParamGen}(1^\lambda); pk_{\mathcal{B}} \leftarrow \mathcal{A}(par);$

$(i_0, i_1, 1^k) \leftarrow \mathcal{A}^{\text{Create, URegister, Corrupt, UWith, Rcv, Spd, S\&R, UDepo}};$

If $sk_{i_0} = \perp$ or $sk_{i_1} = \perp$ then return \perp ;

Run $\text{Rcv}(i_b)$ with \mathcal{A} ;

Let c_1 be the received coin and let j_1 be its index in \mathcal{CL} ;

Repeat the following two steps for $\alpha = 1, \dots, k-1$:

$(i_0, i_1) \leftarrow \mathcal{A}$; If $sk_{i_0} = \perp$ or $sk_{i_1} = \perp$ then return \perp ;

Run $\text{S\&R}(i_b, j_\alpha)$;

Let $c_{\alpha+1}$ be the received coin and let $j_{\alpha+1}$ be its index in \mathcal{CL} ;

Run $\text{Spd}(j_k)$ with \mathcal{A} ;

$b^* \leftarrow \mathcal{A}^{\text{Create, URegister, Corrupt, UWith, Rcv, Spd, S\&R, UDepo}};$

If for any of coins c_1, \dots, c_k we have $c_{ds} > 1$ then output \perp ;

If any of the owners of c_1, \dots, c_k is corrupted then output \perp ;

Return b^* .

Definition 4. (*Anonymity*) A transferable e-cash scheme is fully anonymous if for any stateful PPT adversary \mathcal{A} we have $\mathbf{Adv}_{\mathcal{A}}^{\text{StR}^*-\text{fa}}(\lambda) := \Pr[(\mathbf{Expt}_{\mathcal{A},1}^{\text{StR}^*-\text{fa}}(\lambda) = 1) - \Pr[(\mathbf{Expt}_{\mathcal{A},0}^{\text{StR}^*-\text{fa}}(\lambda) = 1)]$ is negligible in λ (and analogously for $\mathbf{Expt}_{\mathcal{A},b}^{\text{Otr-fa}}$, $\mathbf{Expt}_{\mathcal{A},b}^{\text{St0-fa}}$, and $\mathbf{Expt}_{\mathcal{A},b}^{\text{StR-fa}}$, defined in the full version).

3 Double-Spending Detection

In our construction every coin in the system contains a serial number $\text{SN} = \text{SN}_1 \| \dots \| \text{SN}_k$ where SN_1 was jointly generated by the bank and the user who withdrew the coin, SN_2 was generated by the second user who received the coin and so on. Moreover, a coin contains a set of double-spending tags $\text{DS} = \text{DS}_1 \| \dots \| \text{DS}_{k-1}$ which allows the bank to identify the user that double-spent whenever a coin is deposited twice. (To satisfy the Spend-then-Receive anonymity, these values will be encrypted so that only the bank can see them.)

We first describe the properties of the serial number and double-spending tags in order to construct our transferable e-cash construction. We then give concrete instantiations in Section 3.2.

3.1 Properties of Serial Numbers and Double-Spending Tags

As we will see in Section 3.2, for transferable e-cash it seems essential that the generation of SN_i uses both randomness chosen by the i -th receiver and the secret key of that user. We thus define a *Serial Number Function*, f_{SN} , which on input a random nonce and a secret key (n_i, sk_i) outputs the serial-number component SN_i of the coin. We require a form of collision-resistance, which guarantees that different (n_i, sk_i) generate different SN. Formally:

Definition 5 (Serial number function). A serial number function f_{SN} for parameters Gen_{SN} takes as input parameters $par_{\text{SN}} \leftarrow \text{Gen}_{\text{SN}}$, a nonce and a

secret key (n_i, sk_i) , and outputs a serial number SN_i . (We omit par_{SN} when it is clear from context.)

f_{SN} is called collision-resistant if given $par_{SN} \leftarrow \text{Gen}_{SN}$, it is hard to find $(sk_i, n_i) \neq (sk'_i, n'_i)$ such that $f_{SN}(par_{SN}, n_i, sk_i) = f_{SN}(par_{SN}, n'_i, sk'_i)$.

We also define a *Double Spending Tag Function*, f_{DS} , that takes as input the nonce n_i that the coin owner \mathcal{U}_i had picked when receiving the coin, \mathcal{U}_i 's secret key sk_i and SN_{i+1} , which was computed by the receiver of the coin. We also allow it to take as input some additional identifying information, ID_i , about \mathcal{U}_i . The output is a double-spending tag that reveals nothing about the owner, \mathcal{U}_i , unless she transfers the same coin to more than one users (i.e. double-spends). In that case, the bank can, given a database of public keys of all the users (and associated info ID for each one) identify the user that double-spent and produce a proof accusing her. More formally:

Definition 6 (Double spending tag). A double-spending tag function f_{DS} for parameters Gen_{SN} and key-generation algorithm KeyGen takes as input $par_{SN} \leftarrow \text{Gen}_{SN}$, $(ID_i, n_i, sk_i, SN_{i+1})$ and outputs the double-spending tag DS_i .

- f_{DS} is 2-show extractable if whenever we compute DS_i and DS'_i for the same $(par_{SN}, ID_i, n_i, sk_i)$ but different $SN_{i+1} \neq SN'_{i+1}$, there exists an efficient function f_{DetectDS} that on input DS_i and DS'_i and a list of identifiers \mathcal{I} such that $(ID_i, pk_i) \in \mathcal{I}$ for a pk_i corresponding to sk_i (according to KeyGen), efficiently extracts (pk_i, Π) where Π is an accepting proof for pk_i .
- f_{DS} is exculpable if, given pk_i produced by KeyGen and $par_{SN} \leftarrow \text{Gen}_{SN}$, it is computationally hard to compute an accepting proof for pk_i .

Finally, we want to be able to guarantee anonymity notions even against a malicious bank who gets to see the serial numbers and double-spending tags for deposited coins. Thus, we require that as long as the nonce n_i is fresh and random, these values reveal nothing about the other values, such as sk and ID , used to generate them.

Definition 7 (Anonymity of double spending tags). A double-spending tag function f_{DS} and a serial number function f_{SN} are anonymous if for all $ID_i, sk_i, SN_{i+1}, ID'_i, sk'_i, SN'_{i+1}$ if $par_{SN} \leftarrow \text{Gen}_{SN}$ and n_i is chosen at random, then $(par_{SN}, f_{SN}(par_{SN}, n_i, sk_i), f_{DS}(par_{SN}, ID_i, n_i, sk_i, SN_{i+1}))$ is computationally indistinguishable from $(par_{SN}, f_{SN}(par_{SN}, n_i, sk'_i), f_{DS}(par_{SN}, ID'_i, n_i, sk'_i, SN'_{i+1}))$.

3.2 A Double Spending Detection Mechanism

Here we propose a concrete instantiation for the functions f_{SN}, f_{DS} used to generate the serial numbers and double-spending tags. To give some intuition, we first consider the natural translation of traditional (non-transferable) e-cash double-spending techniques [CFN88], and show why it is not sufficient in the transferable setting. Assume that \mathcal{U}_i and \mathcal{U}_{i+1} execute the Spend protocol where the first user

transfers a coin to the second one. Let $\text{SN}_{i+1} = n_{i+1}$ be the nonce that the second user randomly picks and sends to \mathcal{U}_i . Then \mathcal{U}_i must compute the double-spending tag as follows: $\text{DS}_i = pk_i^{n_{i+1}} F(n_i)$, where $F(n_i)$ is hard to compute, except for the user that chose n_i .

Assume that \mathcal{U}_i double-spends the coin by transferring it to users \mathcal{U}_{i+1} and \mathcal{U}'_{i+1} and that both instances of the coin get eventually deposited at the bank. The bank receives two coins starting with SN_1 , so it looks for the first difference in the serial numbers SN and SN' , which is $\text{SN}_{i+1} \neq \text{SN}'_{i+1}$, pointing to \mathcal{U}_i as the double-spender. The bank now uses the tags DS_i and DS'_i to compute $pk_i = (\text{DS}_i (\text{DS}'_i)^{-1})^{1/(n_{i+1} - n'_{i+1})}$. But what if a coin was double-spent and the receivers picked the same nonce n_{i+1} ? We consider two cases:

Case 1: \mathcal{U}_i double-spends the coin to the *same* user \mathcal{U}_{i+1} and in both transactions \mathcal{U}_{i+1} picks the same nonce n_{i+1} . When the coins are deposited the first difference occurs at position $i + 2$ and the bank will therefore accuse \mathcal{U}_{i+1} of double-spending. However, user \mathcal{U}_{i+1} can easily avoid being wrongly accused of double-spending by picking a fresh nonce each time when receiving a coin.

Case 2: \mathcal{U}_i transfers the same coin to *different* users with pk_{i+1} and pk'_{i+1} who pick the same nonce n_{i+1} when receiving the coin. As before, the bank's serial numbers will diverge at position $i + 2$. However, in this case computation of a public key will fail, as DS_{i+1} and DS'_{i+1} contain different public keys.

The second scenario could be exploited by a collusion of U_i , U_{i+1} and U'_{i+1} to commit a double-spending without being traceable for it. We therefore need to ensure that different users cannot produce the same SN_{i+1} when receiving a coin. We ensure this by making SN_{i+1} dependent on the user's secret key, as formalized in Definition 5. We could easily achieve this by using a collision-resistant hash function, but in e-cash schemes users must prove well-formedness of SN and DS ; so we want to keep the algebraic structure of the above example in order to use efficient proof systems.

Our construction. The parameters par_{SN} are an asymmetric pairing group (q, G_1, G_2, G_T, e) of prime order q and six random generators of G_1 : $(g_1, g_2, h_1, h_2, \tilde{h}_1, \tilde{h}_2)$. We assume that secret keys and the info ID are elements of \mathbb{Z}_q . User \mathcal{U}_{i+1} chooses the nonce n_{i+1} randomly from \mathbb{Z}_q and computes SN_{i+1} as

$$f_{\text{SN}}(n_{i+1}, sk_{i+1}) = \{N_{i+1} = g_1^{n_{i+1}}, M_{i+1} = g_2^{sk_{i+1} \cdot n_{i+1}}\} .$$

When \mathcal{U}_i receives $\text{SN}_{i+1} = (N_{i+1}, M_{i+1})$, she forms the double-spending tags as:

$$f_{\text{DS}}(ID_i, n_i, sk_i, (N_{i+1}, M_{i+1})) = \left\{ \begin{array}{l} A_i = N_{i+1}^{ID_i} h_1^{n_i}, B_i = M_{i+1}^{ID_i} h_2^{n_i} \\ \tilde{A}_i = N_{i+1}^{sk_i} \tilde{h}_1^{n_i}, \tilde{B}_i = M_{i+1}^{sk_i} \tilde{h}_2^{n_i} \end{array} \right\}$$

We show that this construction satisfies the properties defined in Section 3.1. First, the function f_{SN} function is *collision-resistant*: in order to have $N_{i+1} = N'_{i+1}$ the adversary must pick $n_{i+1} = n'_{i+1}$, but then $M_{i+1} = M'_{i+1}$ can only be achieved if $sk_{i+1} = sk'_{i+1}$.

Next we consider double-spending. The bank stores a database of pairs (pk, ID) for all registered users with pk and ID unique to each user. When a coin is deposited, the bank retrieves the serial number $SN = SN_1 || \dots || SN_k$. If a coin was deposited before with $SN \neq SN'$ but $SN_1 = SN'_1$, the bank looks for the first pair such that $SN_{i+1} = (N_{i+1}, M_{i+1}) \neq SN'_{i+1} = (N'_{i+1}, M'_{i+1})$ in order to detect where the double-spending happened. Depending on whether the N -values or the M -values are different, the bank checks for which $ID \in \mathcal{DB}_B$ the following holds:

$$(A_i(A'_i)^{-1}) \stackrel{?}{=} (N_{i+1}(N'_{i+1})^{-1})^{ID} \quad \text{or} \quad (B_i(B'_i)^{-1}) \stackrel{?}{=} (M_{i+1}(M'_{i+1})^{-1})^{ID}$$

This is a relatively cheap operation that can be implemented efficiently. (In our e-cash construction in Section 4, ID will be the user's position in the registered user list.) In our scheme **KeyGen** outputs $pk_i = \hat{g}^{sk_i}$ for a fixed generator \hat{g} of G_2 . When the bank found an ID that satisfies the equation above, it looks up in its database the associated public key and checks whether the following pairing is satisfied:

$$e(\tilde{A}_i(\tilde{A}'_i)^{-1}, \hat{g}) = e(N_{i+1}(N'_{i+1})^{-1}, pk_i) \quad (1)$$

or similar for $\tilde{B}_i, \tilde{B}'_i, M_{i+1}, M'_{i+1}$ in case $N_{i+1} = N'_{i+1}$ (in which case we must have $M_{i+1} \neq M'_{i+1}$). If these checks fail for all pk, ID in the database, the bank outputs (\perp, \perp) , but this should never happen. Thus, the function f_{DetectDS} on input $DS_i, DS'_i, \mathcal{DB}_B$ outputs pk and $\Pi = (DS_i, DS'_i)$. The verification for this proof just checks equation (1). Thus, our f_{DS} function is 2-show extractable.

It remains to be shown that our system $(f_{\text{SN}}, f_{\text{DS}})$ is anonymous and exculpable. In the following lemma (whose proof is in the full version, p. ??) we show that both properties follow from SXDH:

Lemma 1. *The above constructions of a double-spending tag function f_{DS} and a serial number function f_{SN} are anonymous as defined in Definition 7 assuming that DDH holds in G_1 .*

Moreover, the double-spending function is exculpable if DDH holds in G_2 .

Discussion. Note that we could just use Equation 1 to detect double-spending (and discard the values A_i, B_i in f_{DS}). This would however be less efficient, since the bank would have to compute one pairing for every entry in its database. On the other hand, if exculpability is not required, we could discard the values \tilde{A}_i, \tilde{B}_i from f_{DS} .

4 Transferable E-Cash Based on Malleable Signatures

We now describe a generic construction of a transferable e-cash scheme using malleable signatures. Assume the existence of a malleable signature scheme (**MSGen**, **MSKeyGen**, **MSign**, **MSVerify**, **MSigEval**) with allowed transformation class \mathcal{T} as defined below, a signature scheme (**SignGen**, **SKeyGen**, **Sign**, **Verify**), a randomizable public-key encryption scheme (**EKeyGen**, **Enc**, **REnc**, **Dec**), a commitment scheme (**ComSetup**, **Com**), a zero knowledge proof system $\langle P, V \rangle$ and a

hard⁴ relation R_{pk} . We also assume the existence of the functions $f_{SN}, f_{DS}, f_{DetectDS}$ for Gen_{SN} as defined in Section 3.1.

The bank’s withdrawal key consists of $(vk_B^{(MS)}, sk_B^{(MS)}) \leftarrow \text{MSKeyGen}(1^\lambda)$ and $(vk_B^{(S)}, sk_B^{(S)}) \leftarrow \text{SKeyGen}(1^\lambda)$ while the deposit key is $(pk_{\mathcal{D}}, sk_{\mathcal{D}}) \leftarrow \text{EKeyGen}(1^\lambda)$. Users have keys $(pk_{\mathcal{U}}, sk_{\mathcal{U}})$ and when registering with the bank they receive a certificate $cert_{\mathcal{U}} = \text{Sign}_{sk_B^{(S)}}(pk_{\mathcal{U}}, I_{\mathcal{U}})$, where $I_{\mathcal{U}}$ is their joining order.

We recall the properties of malleable signatures, the central building block for our construction, and refer to the full version for the definitions of commitment schemes and re-randomizable encryption.

4.1 Malleable Signatures

A malleable (or homomorphic) signature scheme [ABC⁺12,ALP,CKLM14] allows anyone to compute a signature on a message m' from a signature on m as long as m and m' satisfy some predicate. Moreover, the resulting signature on m' reveals no extra information about the parent message m .

We adapt the definition by Chase et al. [CKLM14], who instead of a predicate consider a set of allowed *transformations*. A malleable signature scheme consists of the algorithms **KeyGen**, **Sign**, **Verify** and **SigEval**, of which the first three constitute a standard signature scheme. **SigEval** transforms multiple message/signature pairs into a new signed message: on input the verification key vk , messages $\vec{m} = (m_1, \dots, m_n)$, signatures $\vec{\sigma} = (\sigma_1, \dots, \sigma_n)$, and a transformation T on messages, it outputs a signature σ' on the message $T(\vec{m})$.

Definition 8 (Malleability). *A signature scheme (**KeyGen**, **Sign**, **Verify**) is malleable with respect to a set of transformations \mathcal{T} if there exists an efficient algorithm **SigEval** that on input $(vk, T, \vec{m}, \vec{\sigma})$, where $(vk, sk) \xleftarrow{\$} \text{KeyGen}(1^\lambda)$, $\text{Verify}(vk, \sigma_i, m_i) = 1$ for all i , and $T \in \mathcal{T}$, outputs a signature σ' for the message $m := T(\vec{m})$ such that $\text{Verify}(vk, \sigma', m) = 1$.*

In order to achieve stronger unforgeability and context-hiding notions, Chase et al. [CKLM14] provide simulation-based definitions for malleable signatures. *Simulatability* requires the existence of a simulator, which without knowing the secret key can simulate signatures that are indistinguishable from standard ones.⁵ Moreover, a simulatable and malleable signature scheme is *context-hiding* if a transformed signature is indistinguishable from a simulated signature on the transformed message. A malleable signature scheme is *unforgeable* if an adversary can only derive signatures of messages that are allowed transformations of signed messages. In the full version we present the corresponding formal definitions.

Chase et al. [CKLM14] describe a construction of malleable signatures based on controlled-malleable NIZKs [CKLM12] which they instantiate under the Decision Linear assumption [BBS04].

⁴ Informally, a relation R is said to be hard if for $(x, w) \in R$, a PPT adversary \mathcal{A} given x will output w_A s.t. $(x, w_A) \in R$ with only negligible probability.

⁵ This requires a trusted setup; for details see the full version.

4.2 Allowed Transformations

In a malleable signature scheme we define a class of allowed transformations, and then unforgeability must guarantee that all valid signatures are generated either by the signer or by applying one of the allowed transformations to another valid signature. In the following we will define two different types of transformations: one to be used when a coin is transferred from a user to another, \mathcal{T}_{CSpend} , and a second one that will be used when a user withdraws a coin from the bank $\mathcal{T}_{CWithdraw}$.

Coin spend transformation. A coin that has been transferred i times (counting withdrawal as the first transfer) will have the following format: $c = (par, (C_{\overline{SN_i}}, C_{\overline{DS_{i-1}}}), (n_i, R_{SN_i}), \sigma)$ where par denotes the parameters of the transferable e-cash scheme and $C_{\overline{SN_i}} = C_{SN_1} \parallel \dots \parallel C_{SN_i}$, $C_{\overline{DS_{i-1}}} = C_{DS_1} \parallel \dots \parallel C_{DS_{i-1}}$, for $C_{SN_j} = \text{Enc}(SN_j)$ and $C_{DS_j} = \text{Enc}(DS_j)$ respectively (all the encryptions are done under $pk_{\mathcal{D}}$). By DS_{i-1} we denote the double spending tag that was computed by the user \mathcal{U}_{i-1} when he transferred the coin to user \mathcal{U}_i , n_i is a nonce picked by \mathcal{U}_i when he received the coin⁶, and by R_{SN_i} the randomness used to compute the encryption of SN_i , i.e., $C_{SN_i} = \text{Enc}(SN_i; R_{SN_i})$. Finally, σ is a malleable signature on $(C_{\overline{SN_i}}, C_{\overline{DS_{i-1}}})$.

Assume now that the user \mathcal{U}_i wishes to transfer the coin c to \mathcal{U}_{i+1} . First, \mathcal{U}_{i+1} will pick a nonce n_{i+1} and will send $SN_{i+1} = f_{SN}(n_{i+1}, sk_{i+1})$ to \mathcal{U}_i . Then, \mathcal{U}_i will compute the new signature as (with T defined below):

$$\sigma' = \text{MSigEval}(par, vk_B^{(MS)}, T, (C_{\overline{SN_i}}, C_{\overline{DS_{i-1}}}), \sigma).$$

The transferred coin that \mathcal{U}_{i+1} will eventually obtain has the form:

$$c' = (par, (C_{\overline{SN_{i+1}}}, C_{\overline{DS_i}}), (n_{i+1}, R_{SN_{i+1}}), \sigma').$$

Note that the value n_{i+1} is only known to \mathcal{U}_{i+1} and he will have to use it when he wants to further transfer the coin, while the randomness $R_{SN_{i+1}}$, used to encrypt SN_{i+1} , was sent by \mathcal{U}_i . What is left is to define the transformation $T \in \mathcal{T}_{CSpend}$ which will take as input $m = (C_{\overline{SN_i}}, C_{\overline{DS_{i-1}}})$ and will output $\mathcal{T}(m) = (C_{\overline{SN_{i+1}}}, C_{\overline{DS_i}})$. A transformation of this type is described by the following values: (i.e. this is the information that one must “know” in order to apply the transformation)

$$\langle \mathcal{T} \rangle = ((sk_i, I_i, cert_i), (n_i, R_{SN_i}, R_{SN_{i+1}}, R_{DS_i}, R), SN_{i+1}).$$

where R is a random string that will be used in order to randomize $(C_{\overline{SN_i}}, C_{\overline{DS_{i-1}}})$ as part of the computation of the new signature. The output of \mathcal{T} , as defined by these values, on input $m = (C_{\overline{SN_i}}, C_{\overline{DS_{i-1}}})$ is then computed as follows:

1. If $SN_i \neq f_{SN}(n_i, sk_i)$ or $\text{Enc}(SN_i; R_{SN_i}) \neq C_{SN_i}$, then output \perp .

⁶ Depending on the instantiation, the nonce, n_i might be computed as the output of a function on a random number the user picks.

2. The new part of the serial number is encoded using randomness $R_{\text{SN}_{i+1}}$:
 $C_{\text{SN}_{i+1}} = \text{Enc}(\text{SN}_{i+1}; R_{\text{SN}_{i+1}})$.
3. The new part of the double spending tag is first computed using f_{DS} and then encrypted: $\text{DS}_i = f_{\text{DS}}(I_i, n_i, sk_i, \text{SN}_{i+1})$ and $C_{\text{DS}_i} = \text{Enc}(\text{DS}_i; R_{\text{DS}_i})$.
4. These encryptions are appended to the re-randomizations of C_{SN_i} and $C_{\text{DS}_{i-1}}$:

$$\begin{aligned} C_{\text{SN}_{i+1}} &= \text{REnc}(C_{\text{SN}_1}; R_1) \parallel \dots \parallel \text{REnc}(C_{\text{SN}_i}; R_i) \parallel C_{\text{SN}_{i+1}} \\ C_{\text{DS}_i} &= \text{REnc}(D_{\text{DS}_1}; R'_1) \parallel \dots \parallel \text{REnc}(C_{\text{DS}_{i-1}}; R'_{i-1}) \parallel C_{\text{DS}_i} \end{aligned}$$

where $R_1, \dots, R_i, R'_1, \dots, R'_{i-1}$ are all parts of the randomness R included in the description of the transformation.

We define $\mathcal{T}_{\text{CSpend}}$ to be the set of all transformations of this form such that:

1. The certificate $cert_i$ is valid (verifiable under the bank's verification key) and correspond to the secret key sk_i and some additional info I_i .
2. The random values $R_{\text{SN}_i}, R_{\text{SN}_{i+1}}, R_{\text{DS}_i}, R$ picked from \mathcal{U}_i belong to the correct randomness space as defined by the encryption scheme.

Coin withdrawal transformation. A coin that was just withdrawn has a different format from a coin that has already been transferred, as there is no need to include double spending tags for neither the bank or the user (we ensure that each coin withdrawn is a different coin). When a user \mathcal{U}_i withdraws a coin from the bank, she picks a nonce n_1 , computes a commitment $com = \text{Com}(n_1, sk_i; open)$ on n_1 and her secret key and sends it to the bank. (For the user to remain anonymous it is important that the bank does not learn n_1 .) The bank computes $\sigma = \text{MSign}(sk_B^{(MS)}, com)$ and sends it to the user. The latter computes $\text{SN}_1 = f_{\text{SN}}(n_1, sk_i)$, chooses randomness R_{SN_1} and sets $C_{\text{SN}_1} = \text{Enc}(\text{SN}_1; R_{\text{SN}_1})$ and computes a new signature $\sigma' = \text{MSigEval}(par, vk_B^{(MS)}, T, com, \sigma)$, which yields the coin defined as $c = (par, C_{\text{SN}_1}, (n_1, R_{\text{SN}_1}), \sigma')$. A transformation $T \in \mathcal{T}_{\text{CWithdraw}}$, which takes as input $m = com$ and outputs $T(m) = C_{\text{SN}_1}$ is described by $\langle T \rangle = ((sk_i, I_i, cert_i), (n_1, open), R_{\text{SN}_1}, \text{SN}_1)$. We define

$$T(com) = \begin{cases} C_{\text{SN}_1} = \text{Enc}(\text{SN}_1; R_1) & \text{if } \text{Com}(n_1, sk_i; open) = com \text{ and } \text{SN}_1 = f_{\text{SN}}(sk_i, n_1) \\ \perp & \text{otherwise.} \end{cases}$$

We define $\mathcal{T}_{\text{CWithdraw}}$ to be the set of all transformations of this form such that:

1. The certificate $cert_i$ is valid (verifiable under the bank's verification key) and correspond to the secret key sk_i and I_i .
2. Randomness R_{SN_1} belongs to the appropriate randomness space.

The class of allowed transformations: \mathcal{T}_{tec} We will allow users to apply a transformation in $\mathcal{T}_{\text{CWithdraw}}$ followed by any number of transformations in $\mathcal{T}_{\text{CSpend}}$. Thus, we define the allowed class of transformations for the malleable signatures used in our transferable e-cash to be the closure of $\mathcal{T}_{\text{tec}} = \mathcal{T}_{\text{CWithdraw}} \cup \mathcal{T}_{\text{CSpend}}$.

4.3 A Transferable E-Cash Construction

Below we describe a transferable e-cash scheme based on malleable signatures. For our construction we assume *secure channels* for all the communications, thus an adversary cannot overhear or tamper with the transferred messages.

ParamGen(1^λ): Compute $par_{MS} \leftarrow \text{MSGen}(1^\lambda)$, $par_{SN} \leftarrow \text{Gen}_{SN}(1^\lambda)$, and $par_{com} \leftarrow \text{ComSetup}(1^\lambda)$. Output $par := (1^\lambda, par_{MS}, par_{com}, par_{SN})$.

UKeyGen(par): Output a random pair (pk_U, sk_U) sampled from R_{pk} .

BKeyGen(par): Compute the withdrawal keys of the bank as $(vk_B^{(MS)}, sk_B^{(MS)}) \leftarrow \text{MSKeyGen}(1^\lambda)$ and $(vk_B^{(S)}, sk_B^{(S)}) \leftarrow \text{SKeyGen}(1^\lambda)$ and the deposit keys as $(pk_D, sk_D) \leftarrow \text{EKeyGen}(1^\lambda)$. Define $pk_W = (vk_B^{(MS)}, vk_B^{(S)})$ and $sk_W = (sk_B^{(MS)}, sk_B^{(S)})$ and output $((pk_W, sk_W), (pk_D, sk_D))$. The bank maintains a list \mathcal{UL} of all registered users and a list \mathcal{DCL} of deposited coins.

Registration($\mathcal{B}[sk_W, pk_U], \mathcal{U}[sk_U, pk_W]$): if $pk_U \notin \mathcal{UL}$, the bank computes $cert_U = \text{Sign}_{sk_B^{(S)}}(pk_U, ID_U)$, where $ID_U = |\mathcal{UL}| + 1$. Add $pk_U, cert, ID_U$ to the user list \mathcal{UL} and output $(cert_U, ID_U)$ or \perp .

Withdraw($\mathcal{B}[sk_W, pk_U], \mathcal{U}[sk_U, pk_W]$): The user picks a nonce n_1 and sends $com = \text{Com}(n_1, sk_U; open)$. The bank computes $\sigma \leftarrow \text{MSign}(par_{MS}, sk_B^{(MS)}, com)$, sends it to the user and outputs ok . If $\text{MSVerify}(par_{MS}, pk_B^{(MS)}, \sigma, com) = 0$, the user outputs \perp ; otherwise she sets $SN_1 = f_{SN}(n_1, sk_U)$, chooses randomness R_{SN_1} and computes $C_{SN_1} = \text{Enc}(SN_1; R_{SN_1})$. Then she sets $\langle T \rangle = ((sk_i, cert_i), (n_1, open), R_{SN_1}, SN_1)$ and computes the new signature $\sigma' = \text{MSigEval}(par_{MS}, vk_B^{(MS)}, T, com, \sigma)$. The output is the coin $c = (par, C_{SN_1}, (n_1, R_{SN_1}), \sigma')$.

Spend($\mathcal{U}_1[c, sk_{U_1}, cert_{U_1}, pk_W], \mathcal{U}_2[sk_{U_2}, pk_W]$) Parse the coin as

$$c = (par, \overline{C_{SN_i}}, \overline{C_{DS_{i-1}}}), (n_i, R_{SN_i}), \sigma) .$$

\mathcal{U}_2 picks a nonce n_{i+1} , computes $SN_{i+1} = f_{SN}(n_{i+1}, sk_{U_2})$ and sends it to \mathcal{U}_1 . \mathcal{U}_1 computes the double spending tag $DS_i = f_{DS}(ID_U, n_i, sk_{U_1}, SN_{i+1})$ and defines the transformation $\langle T \rangle = ((sk_{U_1}, cert_{U_1}), (n_i, R_{SN_i}, R_{SN_{i+1}}, R_{DS_i}, R), SN_{i+1})$. Next, he computes $C_{SN_{i+1}} = \text{Enc}(SN_{i+1}; R_{SN_{i+1}})$ and $C_{DS_i} = \text{Enc}(DS_i; R_{DS_i})$, which he appends to the randomized ciphertext contained in c :

$$\begin{aligned} \overline{C_{SN_{i+1}}} &= \text{REnc}(C_{SN_1}; R_1) \parallel \dots \parallel \text{REnc}(C_{SN_i}; R_i) \parallel C_{SN_{i+1}} \\ \overline{C_{DS_i}} &= \text{REnc}(D_{DS_1}; R'_1) \parallel \dots \parallel \text{REnc}(C_{DS_{i-1}}; R'_{i-1}) \parallel C_{DS_i} \end{aligned}$$

\mathcal{U}_1 computes $\sigma' = \text{MSigEval}(par, vk_B^{(MS)}, T, (\overline{C_{SN_{i+1}}}, \overline{C_{DS_i}}), \sigma)$ and sends $(\sigma', R_{i+1}, (\overline{C_{SN_{i+1}}}, \overline{C_{DS_i}}))$ to \mathcal{U}_2 . If $\text{MSVerify}(par_{MS}, pk_B^{(MS)}, \sigma', (\overline{C_{SN_{i+1}}}, \overline{C_{DS_i}})) = 0$ then \mathcal{U}_2 aborts. Otherwise, \mathcal{U}_2 outputs $c' = (par, (\overline{C_{SN_{i+1}}}, \overline{C_{DS_i}}), (n_{i+1}, R_{SN_{i+1}}), \sigma')$.

Deposit($\mathcal{U}[c, sk_U, cert_U, pk_B], \mathcal{B}[sk_D, pk_U, \mathcal{DCL}]$): First, \mathcal{U} runs a **Spend** protocol with the bank playing the receiver: **Spend**($\mathcal{U}[c, sk_U, cert_{U_1}, pk_W], \mathcal{B}[\perp, pk_W]$) (the bank can set the secret key to \perp , as it will not transfer this coin). If the protocol did not abort, \mathcal{B} holds a valid coin $c = (par, (\overline{C_{SN_i}}, \overline{C_{DS_{i-1}}}), (n_i,$

$R_{\text{SN}_i}), \sigma)$. Next, using $sk_{\mathcal{D}}$, \mathcal{B} decrypts the serial number $\overline{\text{SN}}_i = \text{SN}_1 \parallel \dots \parallel \text{SN}_i$ and the double spending tags $\overline{\text{DS}}_{i-1} = \text{DS}_1 \parallel \dots \parallel \text{DS}_{i-1}$. It checks if in \mathcal{DCL} there exists another coin c' with $\text{SN}'_1 = \text{SN}_1$; if not then adds coin to \mathcal{DCL} . Otherwise, a double spending must have happened and the bank looks for the first position d , where $\text{SN}'_d \neq \text{SN}_d$. (Except with negligible probability such a position exists, since SN_i was chosen by the bank.) It applies the double-spending detection function f_{DetectDS} on the corresponding double spending tags DS_{d-1} and DS'_{d-1} . If f_{DetectDS} outputs \perp then \mathcal{B} aborts. Otherwise, it outputs $(pk_U, \Pi) = f_{\text{DetectDS}}(\text{DS}_{d-1}, \text{DS}'_{d-1}, \mathcal{UL})$.

$\text{VerifyGuilt}(pk_U, \Pi)$: it outputs 1 if the proof Π verifies and 0 otherwise.

Withdraw and Spend are depicted in the full version, where, due to space constraints, the proof of the following was also deferred.

Theorem 1. *If the malleable signature scheme (MSGen, MSKeyGen, MSign, MSVerify, MSigEval) is simulatable, simulation unforgeable and simulation hiding with respect to \mathcal{T} , the signature scheme (SKeyGen, Sign, Verify) is existentially unforgeable, the randomizable public-key encryption scheme (EKeyGen, Enc, REnc, Dec) is semantically secure and statistically re-randomizable, and the commitment scheme (ComSetup, Com) is computationally hiding and perfectly binding, then the construction in Section 4.3 describes a secure and anonymous transferable e-cash scheme as defined in Section 2.*

4.4 Why Malleable Signatures

Let us discuss why our construction requires the use of this powerful primitive. Malleable signatures satisfy a strong notion of unforgeability, called *simulation unforgeability* (see the full version). In brief, it requires that an adversary who can ask for simulated signatures and then outputs a valid message/signature pair (m^*, σ^*) must have derived the pair from received signatures. This is formalized by requiring that there exists an extractor that from (m^*, σ^*) extracts messages \vec{m} that were all queried to the signing oracle and a transformation T such that $m^* = T(\vec{m})$.

Among the anonymity notions considered in the literature, Spend-then-Receive (StR) anonymity (formally defined in the full version) is the hardest to achieve. Recall that it formalizes that an adversary should not be able to recognize a coin he had already owned before. Intuitively, our scheme satisfies it, since a coin only consists of ciphertexts, which are re-randomized, and a malleable signature, which can be simulated. However, when formally proving the notion we have to provide a Deposit oracle, which we cannot simulate when reducing to the security of the encryptions. Here we make use of the properties of malleable signatures, which allow us to extract enough information to check for double-spending—even after issuing simulated signatures (see the proof of Lemma ??).

The scheme by Blazy et al. [BCF⁺11] also claims to achieve StR anonymity. In their scheme a coin contains Groth-Sahai (GS) commitments \vec{c} to the serial number, additional (ElGamal) encryptions \vec{d} of it and a GS proof that guarantees

that the values in \vec{c} and \vec{d} are the same. The bank detects double-spending by decrypting \vec{d} . In their proof of StR anonymity by game hopping, they first replace the GS commitments and proofs by perfectly hiding ones and then simulate the proofs. (Double-spending can still be checked via the values \vec{d} .) Finally they argue that in the “challenge spending via Spd in the experiment, we replace the commitments/encryptions d_{n_i} [...] by random values.”

It is not clear how this can be done while still simulating the **Deposit** oracle, which must check for double-spendings: a simulator breaking security of the encryptions would not know the decryption key required to extract the serial number from \vec{d} . (One would have to include additional encryptions of the serial number and use them for extraction—however, for this approach to work, the proof guaranteeing that the encryptions contain the same values would have to be simulation-sound (cf. [Sah99]), which however contradicts the fact that they must be randomizable.)

5 Instantiation

In order to instantiate our scheme we need to make concrete choices for a malleable signature scheme which supports the allowable transformations \mathcal{T}_{CSpend} and \mathcal{T}_{CWith} , a signature scheme for the signing of certificates, a randomizable public key encryption scheme, a commitment scheme (**ComSetup**, **Com**) and a zero knowledge proof system $\langle P, V \rangle$.

We can use the malleable signature construction given by Chase et al. [CKLM14]. They provide a generic construction of malleable signatures that is based on cm-NIZKs [CKLM12]. There exist two constructions of cm-NIZKs, both due to Chase et al.: the first [CKLM12] is based in Groth-Sahai proofs [GS08], the second [CKLM13], less efficient but simpler one is based on succinct non-interactive arguments of knowledge (SNARGs) and fully homomorphic encryption. The SNARG based construction directly gives a feasibility result, as long as there is some constant maximum on the number of times a given coin can be transferred. To achieve an efficient instantiation, one could instead use the Groth-Sahai instantiation.

In the full version we present an instantiation of our construction based on Groth-Sahai. We first explain that we have to extend the \mathcal{T}_{CSpend} , \mathcal{T}_{CWith} transformations to include the identity. Then, we show that our relation and transformations are *CM-friendly* (Def. ??) which means that all of the objects (instances, witnesses and transformations) can be represented as elements of a bilinear group so that the system is compatible with Groth-Sahai proofs. To achieve that we need to slightly modify our construction very slightly, to map elements of Z_p (like n_i, sk_i, I_i) into the pairing group for the transformation – this happens in a straightforward way without affecting security. Finally, for the rest of the building blocks, we use the structure preserving signature due to Abe et al. [ACD⁺12] and El Gamal encryption scheme [EG85] for both encryption and commitments.

References

- ABC⁺12. Jae Hyun Ahn, Dan Boneh, Jan Camenisch, Susan Hohenberger, abhi she-lat, and Brent Waters. Computing on authenticated data. In *TCC*, 2012.
- ACD⁺12. Masayuki Abe, Melissa Chase, Bernardo David, Markulf Kohlweiss, Ryo Nishimaki, and Miyako Ohkubo. Constant-size structure-preserving signatures: Generic constructions and simple assumptions. In *ASIACRYPT*, 2012.
- ALP. Nuttapon Attrapadung, Benoît Libert, and Thomas Peters. Computing on authenticated data: New privacy definitions and constructions. In *ASIACRYPT*.
- BBS04. Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures. In *CRYPTO*, 2004.
- BCF⁺11. Olivier Blazy, Sébastien Canard, Georg Fuchsbauer, Aline Gouget, Hervé Sibert, and Jacques Traoré. Achieving optimal anonymity in transferable e-cash with a judge. In *AFRICACRYPT*, 2011. Available at <http://crypto.rd.francetelecom.com/publications/p121>.
- BCG⁺14. Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. 2014.
- BCKL09. M. Belenkiy, M. Chase, M. Kohlweiss, and A. Lysyanskaya. Compact e-cash and simulatable VRFs revisited. In *Pairing*, Incs. SV, 2009.
- Bla08. Marina Blanton. Improved conditional e-payments. In *ACNS*, 2008.
- Bra93. Stefan Brands. Untraceable off-line cash in wallets with observers (extended abstract). In *CRYPTO*, 1993.
- CFN88. David Chaum, Amos Fiat, and Moni Naor. Untraceable electronic cash. In *CRYPTO*, 1988.
- CG08. Sébastien Canard and Aline Gouget. Anonymity in transferable e-cash. In *ACNS*, 2008.
- CGT08. Sébastien Canard, Aline Gouget, and Jacques Traoré. Improvement of efficiency in (unconditional) anonymous transferable e-cash. In *FC*. 2008.
- Cha83. David Chaum. Blind signature system. In *CRYPTO*, 1983.
- CHL05. Jan Camenisch, Susan Hohenberger, and Anna Lysyanskaya. Compact e-cash. In *EUROCRYPT*, 2005.
- CKLM12. Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Sarah Meiklejohn. Malleable proof systems and applications. In *EUROCRYPT*, 2012.
- CKLM13. Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Sarah Meiklejohn. Succinct malleable NIZKs and an application to compact shuffles. In *TCC*, 2013.
- CKLM14. Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Sarah Meiklejohn. Malleable signatures: New definitions and delegatable anonymous credentials. In *IEEE Computer Security Foundations Symposium*, 2014.
- CP92. David Chaum and Torben Pryds Pedersen. Transferred cash grows in size. In *EUROCRYPT*, 1992.
- EG85. Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *CRYPTO*, 1985.
- FPV09. Georg Fuchsbauer, David Pointcheval, and Damien Vergnaud. Transferable constant-size fair e-cash. In *CANS*. Springer-Verlag, 2009.
- Fuc11. Georg Fuchsbauer. Commuting signatures and verifiable encryption. In *EUROCRYPT*, 2011.

- GS08. Jens Groth and Amit Sahai. Efficient non-interactive proof systems for bilinear groups. In *EUROCRYPT*, 2008.
- MGGR13. Ian Miers, Christina Garman, Matthew Green, and Aviel D. Rubin. Zero-coin: Anonymous distributed e-cash from bitcoin. In *2013 IEEE Symposium on Security and Privacy, SP 2013, Berkeley, CA, USA, May 19-22, 2013*, pages 397–411, 2013.
- Nak08. S. Nakamoto. Bitcoin: A peer-to-peer electronic cash. bitcoin.org/bitcoin.pdf, 2008.
- OO89. Tatsuaki Okamoto and Kazuo Ohta. Disposable zero-knowledge authentications and their applications to untraceable electronic cash. In *CRYPTO*, 1989.
- OO91. Tatsuaki Okamoto and Kazuo Ohta. Universal electronic cash. In *CRYPTO*, 1991.
- Sah99. Amit Sahai. Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In *FOCS*, 1999.
- vAE90. H. van Antwerpen and Technische Universiteit Eindhoven. *Off-line Electronic Cash*. Eindhoven University of Technology, 1990.