Stronger Definitions for Anonymous Electronic Cash

Mårten Trolin
marten@nada.kth.se

Royal Institute of Technology (KTH), Stockholm, Sweden

Abstract In this paper we investigate definitions of security for previously proposed schemes and point out that they can be strengthened so that the bank does need to be trusted to the same extent. We give experiment-based definitions for our stronger notion and show that they imply security in the framework for Universal Composability. Finally we propose a scheme secure under our definitions in the common reference string (CRS) model under the assumption that trapdoor permutations exist.

1 Introduction

Electronic payments is an interesting cryptographic task. The concept was introduced by Chaum et al. [11]. As was common at the time, the claimed security properties were not defined in a precise way. Many schemes for anonymous electronic followed [7,13,27,24,21,20,19,8],

In the recent years, several papers have focused on giving precise security definitions for tasks such as group signatures [2,3] and ring signatures [4]. In this paper we suggest definitions of security of schemes for electronic cash. In addition to experiment based definitions we construct an ideal functionality for electronic cash and show that security in the experiment setting implies simulation-based security in the framework for universal composability [9] using the ideal functionality.

We construct a scheme using general methods, which is secure in our model in the common reference string (CRS) model. The scheme is not intended for practical use, but it is rather a proof of concept.

We point out that the definitions current schemes do not rule out a corrupt bank cheating a user. The scenario is that the bank claims a user has withdrawn a coin, but the user denies this. We argue that the protocol should include a mechanism to solve such an issue. Our definitions address this issue by requiring a proof a withdrawal also from the bank.

2 Notation and Definitions

We write $[a, b]$ to denote the set $\{x \in \mathbb{Z} \mid a \leq x \leq b\}$, and we let $[b] = [1, b]$. We say that an element is chosen "randomly" instead of the more cumbersome

"independently and uniformly at random". By $r \leftarrow_R S$ we mean that r is chosen randomly in S. Throughout the paper, κ denotes the security parameter. A function $\varepsilon : \mathbb{N} \to \mathbb{R}^+$ is said to be negligible if for each $c > 0$ there exists a $\kappa_0 \in \mathbb{N}$ such that $\varepsilon(\kappa) < \kappa^{-c}$ for $\kappa_0 < \kappa \in \mathbb{N}$. We say that a function $f : \mathbb{N} \to \mathbb{R}^+$ is non-negligible whenever it is not negligible.

We write $\emptyset$ to denote both the empty set and the empty string, and we let $\perp$ be a special symbol. All adversaries in this paper are modeled as polynomial time Turing machines with *non-uniform* auxiliary advice string. We denote the set of such adversaries by PPT*.

Informally a family of functions is called a family of *trapdoor permutations* if the permutation is hard to invert unless a trapdoor is known, in which case it can be efficiently inverted. The precise definition is given in Appendix A.

3 Protocol Definitions and Security Model

Here we give definitions for a scheme for electronic cash as well as for its security.

While some security properties are obvious and dealt with from the very first scheme, others are more subtle. Naturally a scheme must not allow for a user to forge coins, and a double-spender must be detected. The schemes [8,28] require that a corrupt bank cannot accuse an honest user of double-spending, whereas this requirement is not explicit in, e.g., [19,18]. However, to our knowledge, no scheme discusses the possibility of a corrupt bank falsely claiming that an honest user has withdrawn a coin, or rejecting a deposition from a merchant of a legally spent coin. The tendency seem to be to, apart from anonymity, protect the interests of the bank rather than those of the user.

We give definitions requiring that the bank be able to prove withdrawals. Thus, after executing the withdrawal protocol, the output of the bank should be a proof of withdrawal and the output of the user should be a valid coin. However, the neither part should be able to benefit by aborting the protocol prematurely. While there exist protocols that release information piece by piece, so called fair exchange [5], they are either based on gradual release of information and thus not very practical, or require the presence of a third trusted party. Since we would like definitions that can be instantiated with a practical protocol, we use a different approach. After the execution, the bank receives a withdrawal proof, and the user receives a coin secret key. The honest bank would send the proof to the user, who can use it as a coin. Should the bank fail to do this, the user can challenge the transaction and require the bank to prove that a coin has indeed been withdrawn. Since the proof can be used a coin, the scheme is fair also from the point of view of the user.

We require that spent coins be publicly verifiable to avoid the possibility of the bank rejecting a deposition and to ensure that a merchant cannot deny having received a payment. In particular the bank can verify a spent coin. Therefore there is no need for an interactive deposition protocol. The merchant simply hands the spent coin to the bank.

The merchants do not have a secret key in our setting. Instead the receiving merchant's identity mid is encoded in the spent coin together with a transaction identifier tid. Thus the merchant's consent is not necessary in order to spend a coin. We use this approach to make the definitions cleaner. In practice, a user would require some sort of contract before handing the merchant a coin, but we feel this is best handled outside of the protocol. In standard banking systems it is indeed possible to wire money without the recipient's approval, although it is not very sensible to do so. Since there is no secret for the merchant and the resulting spent coin is publicly verifiable, the spending protocol is non-interactive.

We assume the existence of a PKI, i.e., given a public key there exists a method to obtain the identity of the holder of the key. We also assume the existence of secure and authenticated communication. We do not explicitly define a protocol to register a user. If the protocol requires some secret information to be passed from the bank to the user, this can be done in the withdrawal protocol. Therefore there is no loss of generality in this.

Here we discuss payment schemes containing all basic properties, but there are many possible extensions. Examples of such alternate definitions include the presence of a trusted third party which can identify coins, even when they have not been double-spent. Such schemes are called *fair*. Another extension is the possibility to transfer a coin between users in several steps before it is deposited at the bank, and divisible coins. We leave it as an open problem to adjust the definitions to handle also such cases.

3.1 Participants

The participants are the bank $\mathcal{B}$, merchants $\mathcal{M}_i$, and users $\mathcal{U}_i$.

3.2 Algorithms and Protocols

We now define the algorithms and the protocols that a scheme for electronic cash consists of. For non-interactive algorithms the definitions are straight-forward. We define two-party protocols as a pair of algorithms, where each participant executes one algorithm. The algorithms take as input a message, a state, and the private input of the party. On startup of a protocol the initiating party executes the algorithm with $\emptyset$ as message. Each algorithm outputs a pair $(\mathsf{msg}, \mathsf{state})$, where msg is handed as message to the other party's algorithm, and state is passed as input by the executing party the next round. When a party's algorithm outputs $p = \perp$ the protocol is finished, and the final value of each party's state is parsed as private output. The *transcript* of a protocol is defined as the list of messages exchanged.

The below algorithm illustrates the execution between two parties using algorithms A and B with private input sk_A, sk_B, respectively.

Note that the only protocol we use involve exactly two parties.

There is an algorithm for creating a bank key pair and a user key pair. After the user has generated its key, it is inserted into the PKI and hence tied to the user's identity. Each merchant has an identity $\mathsf{mid} \in \{0,1\}^{\kappa/2}$, but no secret key.

$\mathsf{state}_A \leftarrow \mathsf{sk}_A$
$\mathsf{state}_B \leftarrow \mathsf{sk}_B$
while $(\mathsf{msg}_A \neq \perp) \wedge (\mathsf{msg}_B \neq \perp)$ **do**
  $(\mathsf{msg}_B, \mathsf{state}_A) \leftarrow A(\mathsf{msg}_A, \mathsf{state}_A)$
  if $\mathsf{msg}_B \neq \perp$ **then**
    $(\mathsf{msg}_A, \mathsf{state}_B) \leftarrow B(\mathsf{msg}_B, \mathsf{state}_B)$
  end if
end while
return $(\mathsf{state}_A, \mathsf{state}_B)$

Algorithm Header 1 (Bank Key Generation BKg).
INTERFACE: $\mathsf{BKg}(1^\kappa)$, where κ is the security parameter.
OUTPUT: $(\mathsf{bpk}, \mathsf{bsk})$, where bpk is a bank public key and bsk is a bank secret key.

Algorithm Header 2 (User Key Generation UKg).
INTERFACE: $\mathsf{UKg}(1^\kappa)$, where κ is the security parameter.
OUTPUT: $(\mathsf{upk}, \mathsf{usk})$, where upk is a user public key and usk is a user secret key.

Protocol Header 1 (Coin Withdrawal, $(\mathsf{UWithdraw}, \mathsf{BWithdraw})$).
PARTIES: Bank $\mathcal{B}$, User $\mathcal{U}$.
PRIVATE INPUT OF $\mathcal{B}$: Bank public key bpk, bank secret key bsk.
PRIVATE INPUT OF $\mathcal{U}$: Bank public key bpk, user public key upk, user secret key usk.
PRIVATE OUTPUT OF $\mathcal{B}$: Coin coin.
PRIVATE OUTPUT OF $\mathcal{U}$: Coin secret key csk.

$(\mathsf{UWithdraw}, \mathsf{BWithdraw})$ is the protocol used when a user withdraws a coin. The private input of the user is a user private key usk, a user public key upk, and a bank public key bpk. The private input of the bank is a bank secret key bsk and a bank public key bpk. The user's private output is a coin secret key csk used when spending the coin, whereas the bank's output is interpreted as a withdrawn coin (coin). Normally the bank would hand the withdrawn coin to the user, but we do not address this in the protocol.

Algorithm Header 3 (Coin Spending Spend).
INTERFACE: $\mathsf{Spend}(\mathsf{coin}, \mathsf{upk}, \mathsf{usk}, \mathsf{csk}, \mathsf{mid}, \mathsf{tid}, \mathsf{bpk})$, where coin is a coin, upk is a user public key, usk is a user secret key, csk is a coin secret key, mid is a merchant identity, $\mathsf{tid} \in \{0,1\}^{\kappa/2}$ is a transaction identity, and bpk is a bank public key.
OUTPUT: $\mathsf{spentcoin}$, where $\mathsf{spentcoin}$ is a (publicly verifiable) spent coin.

$\mathsf{VfDoubleSpent}(\mathsf{spentcoin}_1, \mathsf{spentcoin}_2, \mathsf{bpk})$ returns the public key upk of the double-spender if $\mathsf{spentcoin}_1$ and $\mathsf{spentcoin}_2$ are transcripts of two spendings of the same coin. Otherwise it returns $\perp$.

We require that the spent coins handed to $\mathsf{VfDoubleSpent}$ has been verified, or the output of the algorithm is undefined. This requirement could be removed by including $(\mathsf{mid}, \mathsf{tid})$ for each coin in the call, but this would make the interface unnecessarily complex.

Algorithm Header 4 (Identifying a Double-Spender, VfDoubleSpent).
INTERFACE: $\mathsf{VfDoubleSpent}(\mathsf{spentcoin}_1, \mathsf{spentcoin}_2, \mathsf{bpk})$, where the parameters $\mathsf{spentcoin}_1$ and $\mathsf{spentcoin}_2$ are two spent coins and bpk is a bank public key.
OUTPUT: (upk), a (possibly empty) user public key.

In addition to the above, there are two algorithms which verify the validity of coins produced during withdrawal and spending, VfCoin(coin, upk, bpk), VfSpentCoin(spentcoin, mid, tid, bpk), The algorithms output 1 if the proof is valid with regards to the additional input parameters and 0 otherwise.

Algorithm Header 5 (Verifying a Withdrawal, VfCoin).
INTERFACE: $\mathsf{VfCoin}(\mathsf{coin}, \mathsf{upk}, \mathsf{bpk})$, where coin is a withdrawn coin, upk a user public key, and bpk a bank public key.
OUTPUT: $b \in \{0,1\}$.

Algorithm Header 6 (Verifying a Spent Coin, VfSpentCoin).
INTERFACE: $\mathsf{VfSpentCoin}(\mathsf{spentcoin}, \mathsf{tid}, \mathsf{mid}, \mathsf{bpk})$, where spentcoin is a spent coin, $\mathsf{tid} \in \{0,1\}^{\kappa/2}$ is a transaction identity, mid is a merchant identity, and bpk is a bank public key.
OUTPUT: $b \in \{0,1\}$.

3.3 Correctness

By correctness we mean that the scheme works as expected when all participants are honest. Proving correctness is often straight-forward, and this property is sometimes not stated explicitly. Here we define correctness for a scheme as defined above.

Experiment 1 (Correctness, $\mathbf{Exp}^{\mathsf{correct}}_{\mathcal{EC}}(\kappa)$).

$(\mathsf{bpk}, \mathsf{bsk}) \leftarrow \mathsf{BKg}(1^\kappa)$
$(\mathsf{upk}, \mathsf{usk}) \leftarrow \mathsf{UKg}(1^\kappa)$
$(\mathsf{coin}, \mathsf{csk}) \leftarrow \mathsf{Withdraw}(\mathsf{bpk}, \mathsf{bsk}, \mathsf{usk})$
if $\mathsf{VfCoin}(\mathsf{coin}, \mathsf{upk}, \mathsf{bpk}) = 0$ **then**
  return 0
end if
$\mathsf{mid} \leftarrow_R \{0,1\}^{\kappa/2}$
$\mathsf{tid} \leftarrow_R \{0,1\}^{\kappa/2}$
$\mathsf{spentcoin} \leftarrow \mathsf{Spend}(\mathsf{usk}, \mathsf{coin}, \mathsf{csk}, \mathsf{mid}, \mathsf{tid}, \mathsf{bpk})$
if $\mathsf{VfSpentCoin}(\mathsf{spentcoin}, \mathsf{mid}, \mathsf{bpk}) = 0$ **then**
  return 0
end if
return 1

Definition 1 (Correctness). *A scheme for electronic cash $\mathcal{EC}$ is* correct *if* $\Pr[\mathbf{Exp}^{\mathsf{correct}}_{\mathcal{EC}}(\kappa) = 0]$ *is negligible as a function of* κ.

Detection of double-spenders is not included in the definition of correctness. This may seem strange at first, but as we will see later the definition of unforgeability implies that double-spenders are detected. Another way to see it is that correctness only stipulates how the protocol works with honest parties, and an honest party does not double-spend.

An alternative definition of correctness is that no polynomial-time machine can find input that makes the correctness experiment fail. Correctness by such a definition would clearly imply correctness in our definition.

3.4 Security

We describe four experiments, or games, to define security for a scheme for electronic cash.

In each experiment the adversary has access to a number of oracles defined below. They operate on the following global parameters.

- $\mathfrak{U}$ contain all public keys inserted into the PKI.
- $\mathfrak{C}$ contain the public keys of corrupt users. Obviously $\mathfrak{C}$ is a subset of $\mathfrak{U}$.
- $(\mathsf{upk}_i, \mathsf{usk}_i)$ is the public and private key of the ith honest user.
- l is the number of coins withdrawn from the bank using the withdrawal oracle.
- ds_i is the number of double-spendings that has been made on behalf of user upk_i using the spend oracle $\mathsf{HonestSpend}$. It is initialized to 0.
- CSK_i is the set of coin keys for user upk_i produced when the withdrawal oracle is used. For a new user it is initiated as the empty set.

$\mathsf{HonestUKg}(1^\kappa)$ calls $\mathsf{UKg}(1^\kappa)$ to generate a key pair $(\mathsf{upk}, \mathsf{usk})$. The public key upk is inserted into the PKI. The key pair $(\mathsf{upk}, \mathsf{usk})$ is stored in the key list. The public key upk is returned.

$\mathsf{CorruptUKg}(\mathsf{upk})$ inserts the key upk into the PKI and into the set $\mathfrak{C}$.

$\mathsf{HonestUWithdraw}(i, j, \mathsf{msg})$ lets the adversary interact in the withdrawal protocol with user i in session j. More precisely, if session j has not been instantiated, i.e., state_j is not defined, then $\mathsf{state}_j \leftarrow (\mathsf{upk}_i, \mathsf{usk}_i, \mathsf{bpk})$. Then a call is made to $\mathsf{UWithdraw}(\mathsf{msg}, \mathsf{state}_j)$ with output $(\mathsf{msg}', \mathsf{state}_j)$. The message msg' is returned, and state_j is stored. After the session has finished, state_j is parsed as a coin secret key csk_j and stored. The key set for user i is updated $\mathsf{CSK}_i \leftarrow \mathsf{CSK}_i \cup \{\mathsf{csk}_j\}$.

$\mathsf{HonestBWithdraw}(j, \mathsf{msg})$ lets the adversary interact in the withdrawal protocol with the bank in session j. More precisely, if session j has not been instantiated, i.e., state_j is not defined, then $\mathsf{state}_j \leftarrow (\mathsf{bpk}, \mathsf{bsk})$. Then a call is made to $\mathsf{BWithdraw}(\mathsf{msg}, \mathsf{state}_j)$ with output $(\mathsf{msg}', \mathsf{state}_j)$. The message msg' is returned, and state_j is stored. After the session has finished, state_j is parsed as a coin coin and returned. Each time a coin is returned the counter l is incremented.

$\mathsf{HonestSpend}(i, j, \mathsf{coin}, \mathsf{mid}, \mathsf{tid})$ first checks if $\mathsf{csk}_j \notin \mathsf{CSK}_i$ and returns $\perp$ if this is the case. Then it checks if (i, coin) has been stored and sets $\mathsf{ds}_i \leftarrow \mathsf{ds}_i + 1$ if this is the case. Then it stores (i, coin), calls $\mathsf{Spend}(\mathsf{coin}, \mathsf{upk}_i, \mathsf{usk}_i, \mathsf{csk}_j, \mathsf{mid}, \mathsf{tid}, \mathsf{bpk})$, and returns the output.

It can be noted that the construction of the HonestSpend oracle prevents the adversary from using it to double-spend a coin.

We let $\mathfrak{Q}_O$ be the set of queries to oracle O and we let $\mathfrak{R}_O$ be the set of responses.

Concurrency The adversary is given oracle access to the withdrawal protocol without any restrictions on how to access it. Therefore the scheme must be secure also under concurrent use to pass our definitions.

Unforgeability The property of *unforgeability* informally says that one cannot create valid coins by other means than withdrawing them from the bank. A little more precisely it says that if a coalition of users spend more than they have legally withdrawn, then at least one of them will get caught as a double-spender. Recall that l is the number of withdrawn coins using the withdrawal oracle. Unforgeability corresponds to the property *balance* of [8].

Experiment 2 (Unforgeability, $\mathbf{Exp}^{\mathsf{unforge}}_{\mathcal{EC},A}(\kappa)$).

$(\mathsf{bpk}, \mathsf{bsk}) \leftarrow \mathsf{BKg}(1^\kappa)$
$(\mathsf{spentcoin}_1, \ldots, \mathsf{spentcoin}_k) \leftarrow A^{\mathsf{CorruptUKg}(\cdot),\mathsf{HonestBWithdraw}(\cdot,\cdot)}(\mathsf{bpk})$
if $k \leq l$ **then**
  return 0
end if
if $\exists i \in [k] : \mathsf{VfCoin}(\mathsf{spentcoin}_i, \mathsf{bpk}) = 0$ **then**
  return 0
end if
if $\exists (i,j) \in [k]^2 : \mathsf{VfDoubleSpent}(\mathsf{spentcoin}_i, \mathsf{spentcoin}_j, \mathsf{bpk}) \in \mathfrak{C}$ **then**
  return 0
end if
return 1

The advantage of the adversary is defined as

$$\mathbf{Adv}^{\mathsf{unforge}}_{\mathcal{EC},A}(\kappa) = \Pr[\mathbf{Exp}^{\mathsf{unforge}}_{\mathcal{EC},A}(\kappa) = 1] \ .$$

Definition 2 (Unforgeability). *A scheme for electronic cash $\mathcal{EC}$ has* unforgeability *if for any $A \in$ PPT the advantage $\mathbf{Adv}^{\mathsf{unforge}}_{\mathcal{EC},A}(\kappa)$ is negligible as a function of κ.*

Non-Frameability A potential problem could be that a coalition of users and possibly the bank could accuse an honest user of double-spending. We say that a scheme has *non-frameability* if it is infeasible to frame an honest user in such a way.

In the experiment below, the parameter ds is the number of double-spending that has performed using the spending oracle HonestSpend, and the $\mathcal{DS}$ is the set of (indices of) double-spent coins. The adversary wins if it creates more

double-spendings than ds, the number of double-spendings the adversary has made using the spending oracle. Intuitively this means that a user can only be accused of the actual number of double-spending she has performed.

Non-frameability corresponds to *strong exculpability* of [8]. The weak variant would guarantee that a user that has never double-spent cannot be accused of double-spending, but it would not prevent a double-spending user from being set up for additional double-spendings. Which variant to prefer is a matter of taste. One could argue that a user that double-spends has already breached her part of the contract, and the protocol should not protect her anymore. On the other hand, a double-spending could occur due to a technical malfunction rather than intentional misconduct, and in such a case it would be unreasonable for the protocol to allow an adversary to create additional double-spendings on behalf of the user.

Since not even the bank should be able to frame a user, we allow the bank key to be chosen in an adversial way.

Experiment 3 (Non-Frameability, $\mathbf{Exp}^{\mathsf{non-frame}}_{\mathcal{EC},A}(\kappa)$).

$(\mathsf{bpk}, \mathsf{state}) \leftarrow A(\mathsf{setup}, 1^\kappa)$
$(\mathsf{spentcoin}_i, \mathsf{tid}_i, \mathsf{mid}_i)_{i=1}^{k} \leftarrow$
$\quad A^{\mathsf{HonestUKg}(1^\kappa),\mathsf{HonestUWithdraw}(\cdot,\cdot,\cdot),\mathsf{HonestSpend}(\cdot,\cdot,\cdot,\cdot)}(\mathsf{guess}, \mathsf{state})$
if $\exists i : \mathsf{VfSpentCoin}(\mathsf{spentcoin}_i, \mathsf{tid}_i, \mathsf{mid}_i 1, \mathsf{bpk}) = 0$ **then**
  return 0
end if
$\mathcal{DS} \leftarrow \{i : \exists j > i : \mathsf{VfDoubleSpent}(\mathsf{spentcoin}_i, \mathsf{spentcoin}_j, \mathsf{bpk}) \in \mathfrak{U}\}$
if $|\mathcal{DS}| > \mathsf{ds}$ **then**
  return 1
else
  return 0
end if

The advantage of the adversary is defined as

$$\mathbf{Adv}^{\mathsf{non-frame}}_{\mathcal{EC},A}(\kappa) = \Pr[\mathbf{Exp}^{\mathsf{non-frame}}_{\mathcal{EC},A}(\kappa) = 1] \ .$$

Definition 3 (Non-Frameability). *A scheme for electronic cash $\mathcal{EC}$ has* non-frameability *if for any $A \in \mathrm{PPT}$ the advantage $\mathbf{Adv}^{\mathsf{non-frame}}_{\mathcal{EC},A}(\kappa)$ is negligible as a function of κ.*

Anonymity A scheme for electronic cash is *anonymous* if it is infeasible for any player, including the bank, to decide the identity of a spender. We define anonymity in a very strong sense, namely that not even knowing the private key of the spender helps revealing the identity of the user. We cannot, however, give the adversary the coin secret keys, since in such a case the adversary could double-spend the coin and reveal the identity. For the same reason the adversary may not use the $\mathsf{HonestSpend}$ oracle to double-spend one of the challenge coins.

In the experiment we let the adversary choose the bank public key and use oracles to create users and withdraw coins before it selects two coins, one of

which will be spent as the challenge. Together with the challenge $\mathsf{spentcoin}$ the adversary is given the private keys of all users. This corresponds to the scenario where the private key of a user is exposed. The privacy of the user should be kept also in such a case.

If the keys were given to the adversary in the first stage, it could withdraw coins itself using the protocol, and it would trivially win the experiment by double-spending the challenge coins.

Experiment 4 (Anonymity, $\mathbf{Exp}^{\mathsf{anon}-b}_{\mathcal{EC},A}(\kappa)$).

$(\mathsf{bpk}, \mathsf{state}) \leftarrow A(\mathsf{setup}, 1^\kappa)$
$(i_0, i_1, \mathsf{mid}, \mathsf{tid}) \leftarrow$
  $A^{\mathsf{HonestUKg}(1^\kappa),\mathsf{HonestUWithdraw}(\cdot,\cdot,\cdot),\mathsf{HonestSpend}(\cdot,\cdot,\cdot,\cdot)}(\mathsf{choose}, \mathsf{state})$
$\mathsf{spentcoin} \leftarrow \mathsf{Spend}(\mathsf{coin}_{i_b}, \mathsf{usk}_{i_b}, \mathsf{csk}_{i_b}, \mathsf{mid}, \mathsf{tid}, \mathsf{bpk})$
$d \leftarrow A^{\mathsf{HonestSpend}(\cdot,\cdot,\cdot,\cdot)}(\mathsf{guess}, \mathsf{state}, \mathsf{spentcoin}, (\mathsf{usk}_i)_{i=1}^{|\mathfrak{U}|})$
if $\exists \mathsf{mid}, \mathsf{tid} : (\{\mathsf{coin}_{i_0}, i_0, \mathsf{mid}, \mathsf{tid}), (\mathsf{coin}_{i_1}, i_1, \mathsf{mid}, \mathsf{tid})\} \cap \mathfrak{Q}_{\mathsf{HonestSpend}} \neq \emptyset$ **then**
  return 0
end if
if $d = b$ **then**
  return 1
else
  return 0
end if

The advantage of the adversary is defined as

$$\mathbf{Adv}^{\mathsf{anon}}_{\mathcal{EC},A}(\kappa) = |\Pr[\mathbf{Exp}^{\mathsf{anon}-0}_{\mathcal{EC},A}(\kappa) = 1] - \Pr[\mathbf{Exp}^{\mathsf{anon}-1}_{\mathcal{EC},A}(\kappa) = 1]| \ .$$

Definition 4 (Anonymity). *A scheme for electronic cash $\mathcal{EC}$ has* anonymity *if for any $A \in \mathrm{PPT}$ the advantage $\mathbf{Adv}^{\mathsf{anon}}_{\mathcal{EC},A}(\kappa)$ is negligible as a function of κ.*

Exculpability Exculpability states that the bank should not be able to create proofs of withdrawal, i.e., coins, which the user cannot spend.

Experiment 5 (Exculpability, $\mathbf{Exp}^{\mathsf{exculp}}_{\mathcal{EC},A}(\kappa)$).

$(\mathsf{bpk}, \mathsf{state}) \leftarrow A(\mathsf{setup}, 1^\kappa)$
$(\mathsf{coin}, i, \mathsf{mid}, \mathsf{tid}) \leftarrow$
  $A^{\mathsf{HonestUKg}(1^\kappa),\mathsf{HonestUWithdraw}(\cdot,\cdot,\cdot),\mathsf{HonestSpend}(\cdot,\cdot,\cdot,\cdot,\cdot)}(\mathsf{guess}, \mathsf{state})$
if $\mathsf{VfCoin}(\mathsf{coin}, \mathsf{upk}_i, \mathsf{bpk}) = 0$ **then**
  return 0
end if
if $\exists \mathsf{csk} \in \mathsf{CSK}_i : \mathsf{Spend}(\mathsf{coin}, \mathsf{upk}_i, \mathsf{usk}_i, \mathsf{csk}, \mathsf{mid}, \mathsf{tid}, \mathsf{bpk}) \neq \perp$ **then**
  return 0
end if
return 1

The advantage of the adversary is defined as

$$\mathbf{Adv}^{\mathsf{exculp}}_{\mathcal{EC},A}(\kappa) = \Pr[\mathbf{Exp}^{\mathsf{exculp}}_{\mathcal{EC},A}(\kappa) = 1] \ .$$

Definition 5 (Exculpability). *A scheme for electronic cash $\mathcal{EC}$ has* exculpability *if for any $A \in$ PPT the advantage $\mathbf{Adv}^{\mathsf{exculp}}_{\mathcal{EC},A}(\kappa)$ is negligible as a function of κ.*

Finally we make the following definition.

Definition 6 (Secure Scheme for Electronic Cash). *A scheme for electronic cash is* secure *if it has unforgeability, non-frameability, anonymity, and exculpability.*

3.5 Comparison to Group Signatures

Electronic cash resembles group signatures in many ways. When compared to [17,3], certain similarities and differences can be identified.

Unforgeability Our definition of unforgeability resembles the misidentification attack of [17] and traceability of [3].

Non-Frameability Non-frameability is similar of both group signature definitions in that it requires that a user cannot be framed even if the complete system conspires against her. Although the adversary is given the secret keys of the group manager in [17,3], our definition is stronger, since we allow the adversary to construct the key itself.

Anonymity Anonymity of a group signature is different than that of a spent coin, since the opening key can always be used to open group signature. This is reflected in the experiments, which otherwise are quite similar.

Exculpability The exculpability property is not defined for any group signature scheme to our knowledge. The corresponding property would be that a group manager cannot falsely claim that it has included a certain member into the group. A scenario where this might pose a problem is if group members are allowed to download certain information and there is a price to join the group. Group signatures do not address the potential issue when a member claims that the group manager has not issued him a key.

4 Security in the Framework for Universal Composability

We now consider the relation between experiment-based security of a scheme for electronic cash as defined above and security in the framework for universal composability (UC) [9]. We describe an ideal functionality, discuss why it captures the notion of anonymous electronic cash, and show that a scheme that is secure according to Definition 6 also securely realizes the ideal functionality.

The ideal *anonymous electronic cash* functionality $\mathcal{F}_{\text{AnonEC}}$ running with parties $\mathcal{B}$, $\mathfrak{U} = \{\mathcal{U}_1, \mathcal{U}_2, \ldots, \mathcal{U}_k\}$ is given in Figure 1. The ideal adversary $\mathcal{S}$ corrupts a subset of the users and possibly the bank. We let $\mathfrak{C}$ be the set of corrupted parties.

We do not differentiate between users and merchants. In fact, any party (except for the bank $\mathcal{B}$) can act both as a merchant and as a user. In addition we assume every user is uniquely identified with an identifier $\mathsf{mid} \in \{0,1\}^{\kappa/2}$.

The ideal functionality stores the following values.

T_{coins} is the table of coins that the bank has issued.
cc is the number of coins that have been withdrawn by corrupt users. It is initialized to 0.
$T_{\text{spent}-\text{coins}}$ contains the coins that have been honestly spent.
$T_{\text{corrupt}-\text{coins}}$ holds the coins that have been verified to be correct, but which the bank has not issued.
T_{unres} is a list of unordered pairs of spent coins that have been spent by corrupts parties, but for which it has still not been decided whether they should be considered double-spendings or not.
$T_{\text{not}-\text{double}}$ are the coin pairs that have been determined not to be double-spendings.
T_{double} are the coin pairs that have been determined to be double-spendings.

4.1 About the Functionality

Let us discuss why $\mathcal{F}_{\text{AnonEC}}$ captures what one would expect from a secure scheme for anonymous electronic cash.

Withdrawal For an honest user, the coin is created as in the protocol to ensure correct distribution. The coin is stored in the table for withdrawn coins T_{coins} and returned to the bank. For a corrupt user, the bank engages in the withdrawal protocol (via the simulator). If the result is indeed a coin, then it is stored in the coins table and the counter for coins withdrawn by corrupt users is incremented.

Coin Verification Coins are deemed valid when the protocols says so. This may seem overly simplified, but since the `Spend` algorithms requires valid coins to be spendable, this covers what one would expect from a valid coin. By the correctness of $\mathcal{EC}$, honestly withdrawn coins always pass the verification.

Spending Before a coin can be spent, it is verified that it is valid and the spender owns the coin. If so, then a spent coin is created by creating a new user, withdraw a coin and spend it. Thus the spent coin has no information about the owner to ensure anonymity. The coin is stored in the table for spent coins $T_{\text{spent}-\text{coins}}$.

Functionality 1 (Anonymous Electronic Cash).

1. Wait for the message $(\mathcal{S}, \texttt{Keys}, (\mathsf{bpk}, \mathsf{bsk}), (\mathsf{upk}_i, \mathsf{usk}_i)_{i=1}^{k})$ where $\mathsf{usk}_i = \perp$ if $\mathcal{U}_i \in \mathfrak{C}$ and $\mathsf{bsk} = \perp$ if $\mathcal{B} \in \mathfrak{C}$. Store $(\mathsf{bpk}, \mathsf{bsk})$ and $(\mathsf{upk}_i, \mathsf{usk}_i)$.
2. Then handle incoming messages as follows.
   - **Withdraw.** Upon reception of $(\mathcal{B}, \texttt{AccWithdrawal}, \mathcal{U}_i)$ act as follows.
     - If $\mathcal{U}_i \notin \mathfrak{C}$, then compute $(\mathsf{coin}, \mathsf{csk}) \leftarrow \mathsf{Withdraw}(\mathsf{bpk}, \mathsf{bsk}, \mathsf{usk}_i)$. Store $(\mathcal{U}_i, \mathsf{coin}, \mathsf{csk})$ in T_{coins}. Then hand $((\mathcal{B}, \texttt{Coin}, \mathcal{U}_i, \mathsf{coin}), (\mathcal{S}, \texttt{AccWithdrawal}, \mathcal{U}_i))$ to $\mathcal{C}_{\mathcal{I}}$.
     - If $\mathcal{U}_i \in \mathfrak{C}$, then hand $(\mathcal{S}, \texttt{AccWithdrawal}, \mathcal{U}_i)$ to $\mathcal{C}_{\mathcal{I}}$. Upon reception of response $(\mathcal{S}, \texttt{Coin}, \mathcal{U}_i, \mathsf{coin})$, store $(\mathcal{U}_i, \mathsf{coin}, \perp)$ in T_{coins}. Set $\mathsf{cc} \leftarrow \mathsf{cc}+1$. Hand $(\mathcal{B}, \texttt{Coin}, \mathcal{U}_i, \mathsf{coin})$ to $\mathcal{C}_{\mathcal{I}}$.
   - **Verify Coin.** Upon reception of $(P, \texttt{VfCoin}, \mathsf{coin}, \mathcal{U}_i)$, set $b \leftarrow \mathsf{VfCoin}(\mathsf{coin}, \mathsf{bpk}, \mathsf{upk}_i)$ and hand $(P, \texttt{VfCoin}, \mathsf{coin}, \mathcal{U}_i, b)$ to $\mathcal{C}_{\mathcal{I}}$.
   - **Spend.** Upon reception of $(\mathcal{U}_i, \texttt{Spend}, \mathsf{coin}, \mathsf{mid}, \mathsf{tid})$, execute $(\cdot, \texttt{VfCoin}, \mathsf{coin}, \mathcal{U}_i)$. If the result is 0, then hand $(\mathcal{U}_i, \texttt{Spend}, \mathsf{coin}, \mathsf{mid}, \mathsf{tid}, \perp)$ to $\mathcal{C}_{\mathcal{I}}$. Otherwise compute $(\mathsf{usk}, \mathsf{upk}) \leftarrow \mathsf{UKg}(1^{\kappa})$, $(\mathsf{coin}', \mathsf{csk}') \leftarrow \mathsf{Withdraw}(\mathsf{bpk}, \mathsf{bsk}, \mathsf{usk})$, $\mathsf{spentcoin} \leftarrow \mathsf{Spend}(\mathsf{coin}, \mathsf{usk}, \mathsf{csk}, \mathsf{mid}, \mathsf{tid}, \mathsf{bpk})$. Store $(\mathcal{U}_i, \mathsf{coin}, \mathsf{mid}, \mathsf{tid}, \mathsf{spentcoin})$ in $T_{\text{spent-coins}}$.
   - **Verify Spent Coin.** Upon reception of $(P, \texttt{VfSpentCoin}, \mathsf{spentcoin}, \mathsf{mid}, \mathsf{tid})$ proceed as follows.
     - If $\mathcal{B} \in \mathfrak{C}$, then set $b \leftarrow \mathsf{VfSpentCoin}(\mathsf{spentcoin}, \mathsf{mid}, \mathsf{tid}, \mathsf{bpk})$.
     - If $(\cdot, \mathsf{mid}, \mathsf{tid}, \mathsf{spentcoin})$ has been stored in $T_{\text{spent-coins}}$, then set $b \leftarrow 1$.
     - If $(\cdot, \mathsf{coin}, \mathsf{mid}, \mathsf{tid}, \mathsf{spentcoin}) \notin T_{\text{spent-coins}}$, then do as follows.
       * If there exists $\mathcal{U}_j \in \mathfrak{C}$ such that $(\mathcal{U}_j, \mathsf{coin}, \perp) \in T_{\text{coins}}$, then set $b \leftarrow \mathsf{VfSpentCoin}(\mathsf{spentcoin}, \mathsf{mid}, \mathsf{tid}, \mathsf{bpk})$. If $b = 1$, then store $(\mathcal{U}_j, \mathsf{mid}, \mathsf{tid}, \mathsf{spentcoin})$ in $T_{\text{corrupt-coins}}$ and add $\{\mathsf{spentcoin}, \mathsf{spentcoin}'\}$ to T_{unres} for each $\mathsf{spentcoin}'$ such that $(\cdot, \cdot, \cdot, \mathsf{spentcoin}') \in T_{\text{corrupt-coins}}$.
       * If no such user exists, then set $b \leftarrow 0$.

     Hand $(P, \texttt{VfSpentCoin}, \mathsf{spentcoin}, \mathsf{mid}, \mathsf{tid}, b)$ to $\mathcal{C}_{\mathcal{I}}$.
   - **Verify Double-Spending.** Upon reception of $(P, \texttt{VfDblSpent}, \mathsf{spentcoin}_1, \mathsf{spentcoin}_2)$, proceed as follows.
     - If $(\mathcal{U}_j, \mathsf{coin}, \mathsf{mid}, \mathsf{tid}, \mathsf{spentcoin}_1)$ and $(\mathcal{U}_j, \mathsf{coin}, \mathsf{mid}', \mathsf{tid}', \mathsf{spentcoin}_2)$ for $(\mathsf{mid}, \mathsf{tid}) \neq (\mathsf{mid}', \mathsf{tid}')$ exist in $T_{\text{spent-coins}}$, then set $\mathcal{U} \leftarrow \mathcal{U}_j$.
     - If $(\mathcal{U}_j, \{\mathsf{spentcoin}_1, \mathsf{spentcoin}_2\}) \in T_{\text{double}}$, then let $\mathcal{U} \leftarrow \mathcal{U}_j$.
     - If $\{\mathsf{spentcoin}_1, \mathsf{spentcoin}_2\} \in T_{\text{not-double}}$, then let $\mathcal{U} \leftarrow \perp$.
     - Otherwise set $\mathsf{upk} \leftarrow \mathsf{VfDoubleSpent}(\mathsf{spentcoin}_1, \mathsf{spentcoin}_2, \mathsf{bpk})$. Set $\mathcal{U}_j$ such that $\mathsf{upk}_j = \mathsf{upk}$ unless $\mathsf{upk} = \perp$.
       * If $\mathcal{U}_j \in \mathfrak{C}$, then insert $(\mathcal{U}_j, \{\mathsf{spentcoin}_1, \mathsf{spentcoin}_2\})$ into T_{double} and remove $\{\mathsf{spentcoin}_1, \mathsf{spentcoin}_2\}$ from T_{unres}.
       * If $\mathcal{U}_j \notin \mathfrak{C}$ and $|T_{\text{corrupt-coins}}| - \mathsf{cc} - |T_{\text{double}}| = |T_{\text{unres}}|$, then let $\mathcal{U}_j \leftarrow_R \mathfrak{C}$, insert $(\mathcal{U}_j, \{\mathsf{spentcoin}_1, \mathsf{spentcoin}_2\})$ into T_{double} and remove $\{\mathsf{spentcoin}_1, \mathsf{spentcoin}_2\}$ from T_{unres}.
       * Otherwise let $\mathcal{U}_j \leftarrow \perp$, insert $\{\mathsf{spentcoin}_1, \mathsf{spentcoin}_2\}$ into $T_{\text{not-double}}$ and remove $\{\mathsf{spentcoin}_1, \mathsf{spentcoin}_2\}$ from T_{unres}.

     Hand $(P, \texttt{VfDblSpent}, \mathsf{spentcoin}_1, \mathsf{spentcoin}_2, \mathcal{U}_j)$ to $\mathcal{C}_{\mathcal{I}}$.

Figure 1. The definition of $\mathcal{F}_{\text{AnonEC}}$.

Verification of Spent Coin If the bank is honest and the coin exists in the table for spent coins, then it is valid. If it does not exist in the table, it may still be valid, but only if it has been spent by a corrupt party and would implicate a corrupt party if double-spent. This is handled by including the spent coin in the list of potential double-spending by corrupt parties.

If the bank is corrupt, then any coin deemed valid by the protocol is accepted.

Identification of Double-Spenders The algorithm is constructed so that it may only point out an honest user if it has actually double-spent a coin by sending a `Spend` command twice for the same coin.

The algorithm also ensures that if more coins are spent than withdrawn by corrupt parties, then a double-spender will be revealed. If the two coins have been spent by corrupt parties, then they may only be cleared from double-spending if there are enough potential double-spendings to cover the surplus of spent coins against withdrawn coins. As a special case a double-spending is never required to be exposed if the corrupt parties have not spent more coin than they have withdrawn.

4.2 On the Possibility of Simplifying the Functionality

TODO: Finns kanske bättre exempel. Lägg gärna till bild med graf.

The functionality $\mathcal{F}_{\text{AnonEC}}$ is rather complex, and it is natural question to ask whether it could be simplified. Let us consider how double-spenders are identified. Let $\mathcal{Z}$ be an environment proceeding as follows:

- $\mathcal{Z}$ runs with one corrupt user $\mathcal{U}_1$.
- $\mathcal{U}_1$ withdraws three coins.
- $\mathcal{U}_1$ spends four times, creating $\mathsf{spentcoin}_1, \mathsf{spentcoin}_2, \mathsf{spentcoin}_3, \mathsf{spentcoin}_4$.
- $\mathcal{Z}$ lets a player check the five coin-pairs ($\mathsf{spentcoin}_1$, $\mathsf{spentcoin}_2$), ($\mathsf{spentcoin}_1$, $\mathsf{spentcoin}_3$), ($\mathsf{spentcoin}_2$, $\mathsf{spentcoin}_3$), ($\mathsf{spentcoin}_2$, $\mathsf{spentcoin}_4$), ($\mathsf{spentcoin}_3$, $\mathsf{spentcoin}_4$) for double-spendings.

There is a consistent scenario where none of these are double-spendings, namely if ($\mathsf{spentcoin}_1, \mathsf{spentcoin}_4$) form a double-spending. Therefore the functionality should not force any of the checks to reveal $\mathcal{U}_1$. As a matter of fact, once these combinations have been checked, further queries must return consistent answers, hence the need for the table $T_{\text{not}-\text{double}}$. However, if the pair ($\mathsf{spentcoin}_1, \mathsf{spentcoin}_4$) is checked, it must be deemed a double-spending with a corrupt user as double-spender. Also in this case, a table T_{double} must be used to ensure further queries are answered in the same way.

As the above example shows, the functionality needs to be complex to handle double-spendings correctly, and it seems hard to make it considerably simpler.

4.3 The Real Protocol π_{AnonEC}

We describe how the protocol π_{AnonEC} is built from the algorithms of the scheme.

The Bank

The bank $\mathcal{B}$ generates $(\mathsf{bpk}, \mathsf{bsk}) \leftarrow \mathsf{BKg}(1^\kappa)$ and broadcasts bpk. Then it waits for a message $(\mathtt{Keys}, \mathsf{upk}_i)$ for every user $\mathcal{U}_i$. Incoming messages are handled as follows:

- Upon reception of $(\mathtt{AccWithdrawal}, \mathcal{U}_i)$, the bank engages in the withdrawal protocol with $\mathcal{U}_i$. After the protocol has terminated, it outputs ($\mathtt{Coin}$, $\mathcal{U}_i$, coin).

Users

The user $\mathcal{U}_i$ generates $(\mathsf{upk}_i, \mathsf{usk}_i) \leftarrow \mathsf{UKg}(1^\kappa)$ and broadcasts upk_i. Then it waits for a message $(\mathtt{Keys}, \mathsf{upk}_j)$ for every user $\mathcal{U}_j$ and $(\mathtt{Keys}, \mathsf{bpk})$ from $\mathcal{B}$. Incoming messages are handled as follows:

- When challenged in the withdrawal protocol, run it according to the algorthm $\mathsf{UWithdraw}$. After the protocol has terminated, store the output csk_j.
- Upon reception of $(\mathtt{Spend}, \mathsf{coin}, \mathsf{mid}, \mathsf{tid})$, set $\mathsf{spentcoin} \leftarrow \mathsf{Spend}(\mathsf{upk}_i, \mathsf{usk}_i, \mathsf{csk}_j, \mathsf{mid}, \mathsf{tid}, \mathsf{bpk})$ for the corresponding coin secret key csk_j. Output ($\mathtt{Spend}$, coin, mid, tid, $\mathsf{spentcoin}$).

All Parties

Incoming messages are handled as follows:

- Upon reception of the message $(\mathtt{VfCoin}, \mathsf{coin}, \mathcal{U}_j)$, set $b \leftarrow \mathsf{VfCoin}(\mathsf{coin}, \mathsf{upk}_j, \mathsf{bpk})$ and return $(\mathtt{VfCoin}, \mathsf{coin}, \mathcal{U}_j, b)$.
- Upon reception of the message $(\mathtt{VfSpentCoin}, \mathsf{spentcoin}, \mathsf{mid}, \mathsf{tid})$, set $b \leftarrow \mathsf{VfSpentCoin}(\mathsf{spentcoin}, \mathsf{mid}, \mathsf{tid}, \mathsf{bpk})$ and return $(\mathtt{VfSpentCoin}, \mathsf{spentcoin}, \mathsf{mid}, \mathsf{tid}, b)$.
- Upon reception of the message $(\mathtt{VfDblSpent}, \mathsf{spentcoin}_1, \mathsf{spentcoin}_2)$, set $\mathsf{upk} \leftarrow \mathsf{VfDoubleSpent}(\mathsf{spentcoin}_1, \mathsf{spentcoin}_2, \mathsf{bpk})$. If $\mathsf{upk} = \perp$, then return $(\mathtt{VfDblSpent}, \mathsf{spentcoin}_1, \mathsf{spentcoin}_2, \perp)$. Otherwise let $\mathcal{U}_j$ be such that $\mathsf{upk}_j = \mathsf{upk}$, and return $(\mathtt{VfDblSpent}, \mathsf{spentcoin}_1, \mathsf{spentcoin}_2, \mathcal{U}_j)$

4.4 Proof of Security

Theorem 1. *Let* $\mathcal{EC} = (\mathsf{BKg}, \mathsf{UKg}, \mathsf{UWithdraw}, \mathsf{BWithdraw}, \mathsf{VfCoin}, \mathsf{Spend}, \mathsf{VfSpentCoin}, \mathsf{VfDoubleSpent})$ *be a secure scheme for anonymous electronic cash. Then* π_{AnonEC} *securely realizes* $\mathcal{F}_{\mathrm{AnonEC}}$.

Proof. Defining the Hybrids. We prove the theorem with a hybrid argument. We build a polynomial-size chain of protocols $\pi_0, \pi_1, \ldots, \pi_t$ such that $\pi_0 = \mathcal{F}_{\mathrm{AnonEC}}$ and $\pi_t = \pi_{\mathrm{AnonEC}}$. Then we show that if there exists an adversary A which can distinguish between π_t and π_{t+1} for some t, then A can be used to break one of the underlying assumptions.

1. We define π_1^t to be π_1^{t-1} with the difference that the tth call to $\mathtt{Spend}$ produces a spent coin according to the protocol rather than using a dummy user as in the functionality.

2. We define π_2^t to be π_2^{t-1} with the difference that the tth call to `Spend` there is no call to `VfCoin` before the spent coin is constructed.
3. Let $\pi_3^0 = \pi_2^m$, and define π_3^t to be π_3^{t-1} with the difference that the tth call to `VfSpentCoin` is executed according to the protocol rather than to the functionality.
4. Let $\pi_4^0 = \pi_3^m$, and define π_4^t to be π_4^{t-1} with the difference that the tth call to `VfDblSpent` is executed according to the protocol rather than to the functionality.

Building the Simulator. By assumption $\mathcal{Z}$ distinguishes between $\mathcal{F}_{\text{AnonEC}}$ and π_{AnonEC} for any ideal adversary. In particular it distinguishes between the two protocols for the adversary defined as follows.

TODO: Borde vara paragraph

For each player P_i that the real-world adversary A corrupts, the ideal adversary $\mathcal{S}$ corrupts the corresponding dummy player $\tilde{P}_i$. When a corrupted dummy player $\tilde{P}_i$ receives a message m from $\mathcal{Z}$, the simulator $\mathcal{S}$ lets $\mathcal{Z}'$ send m to P_i. When a corrupted P_i outputs a message m to $\mathcal{Z}'$, then $\mathcal{S}$ instructs the corrupted $\tilde{P}_i$ to output m to $\mathcal{Z}$. This corresponds to P_i being linked directly to $\mathcal{Z}$.

The simulated real-world adversary $\mathcal{A}$ is connected to $\mathcal{Z}$, i.e., when $\mathcal{Z}$ sends m to $\mathcal{S}$, $\mathcal{Z}'$ hands m to $\mathcal{A}$, and when $\mathcal{A}$ outputs m to $\mathcal{Z}'$, $\mathcal{S}$ hands m to $\mathcal{Z}$. All non-corrupted players are simulated honestly. The corrupted players run according to their respective protocols.

When all parties have broadcasted their keys, $\mathcal{S}$ inspects the internal state of honest parties and intercepts the broadcast of corrupt parties to construct $(\mathsf{bpk}, \mathsf{bsk})$ and $(\mathsf{upk}_i, \mathsf{usk}_i)_{i=1}^k$ where $\mathsf{bsk} = \perp$ and $\mathsf{usk}_i = \perp$ only if $\mathcal{B}$ and $\mathcal{U}_i$, respectively, is corrupt. It hands $(\mathcal{F}_{\text{AnonEC}}, \texttt{Keys}, (\mathsf{bpk}, \mathsf{bsk}), (\mathsf{upk}_i, \mathsf{usk}_i)_{i=1}^k)$ to $\mathcal{C}_{\mathcal{I}}$.

When $\mathcal{S}$ receives the message $(\texttt{AccWithdrawal}, \mathcal{U}_i)$, it instructs $\mathcal{Z}'$ to send $(\texttt{AccWithdrawal}, \mathcal{U}_i)$ to $\mathcal{B}$. If $\mathcal{U}_i \in \mathfrak{C}$, then on output $(\texttt{Coin}, \mathcal{U}_i, \mathsf{coin})$ from $\mathcal{B}$, $\mathcal{S}$ hands $(\mathcal{F}_{\text{AnonEC}}, \texttt{Coin}, \mathcal{U}_i, \mathsf{coin})$ to $\mathcal{C}_{\mathcal{I}}$. All other functions are local and need not be simulated for $\mathcal{A}$.

We now handle the cases when $\mathcal{Z}$ distinguishes between π_i^t and π_i^{t+1} for $i = 1, 2, 3, 4$.

1. Assume $\mathcal{Z}$ can distinguish between π_1^t and π_1^{t+1} with non-negligible probability. Then we construct A' breaking the anonymity of $\mathcal{EC}$ as follows. A' simulates the protocol to $\mathcal{Z}$ by using its HonestUKg to set up keys for honest users, interact with HonestBWithdraw to simulate withdrawals, and use the HonestSpend oracles to construct coins. Let the $(t+1)$th `Spend` request be on behalf of user $\mathcal{U}_i$ for data $(\mathsf{mid}, \mathsf{tid})$. When executing Experiment 4, A' requests a spent coin by either $\mathcal{U}_i$ or a user $\mathcal{U}_j$, which has never before spent a coin. The challenge coin spentcoin is returned on the `Spend` request.
   If the challenge coin is by $\mathcal{U}_i$, then A' has run π_1^{t+1}, and if the coin is by $\mathcal{U}_j$, then the protocol simulated is π_1^t. Since, by assumption $\mathcal{Z}$ can distinguish between the two, it wins the anonymity experiment with non-negligible probability.

2. Assume $\mathcal{Z}$ can distinguish between π_2^t and π_2^{t+1} with non-negligible probability.

   We construct A' breaking the exculpability property of $\mathcal{EC}$ by running in Experiment 5 and simulating the protocol for $\mathcal{Z}$. A' uses its oracles to create keys for the users, withdraw coins, and create spent coins. Since $\mathcal{Z}$ can distinguish between π_2^t and π_2^{t+1}, the coin to be spent in call $t+1$ does not pass the `VfCoin`, but can still be spent. Then A' outputs this coin in the guess phase of the experiment. Since the coin cannot be spent, A' wins the experiment with non-negligible probability.

3. Assume $\mathcal{Z}$ can distinguish between π_3^t and π_3^{t+1} with non-negligible probability. We can assume $\mathcal{B} \notin \mathfrak{C}$, since otherwise `VfSpentCoin` is run identically in the protocol and the functionality. By the correctness of $\mathcal{EC}$, a coin issued by the bank and honestly spent is always accepted.

   By the construction of the functionality, a spentcoin created by corrupt user will be detected as a double-spending if more coins are spent than has been withdrawn. Therefore $\mathcal{Z}$ can distinguish between the protocol and the functionality only if the tth call to `Spend` will be revealed as a double-spending by the functionality but not by the protocol.

   We use $\mathcal{Z}$ to break the unforgeability of $\mathcal{EC}$ as follows. We construct A' running in Experiment 2. The keys of the users are created honestly and then "registered" using the CorruptUKg oracle. Withdrawals are simulated by interacting with the HonestBWithdraw oracle. When asked to output forged coins, it outputs $T_{\text{corrupt}-\text{coins}}$. By the assumption, the table contains more coins than have been withdrawn, thus breaking the unforgeability property of $\mathcal{EC}$.

4. Assume $\mathcal{Z}$ can distinguish between π_4^t and π_4^{t+1} with non-negligible probability.

   For double-spendings that point out a corrupt user as double-spender, the protocol and the functionality are identical. Therefore $\mathcal{Z}$, if able to distinguish between the functionality and the protocol, has found ($\mathsf{spentcoin}_1$, $\mathsf{spentcoin}_2$) such that the functionality does not consider them a double-spending, but the protocol points out an honest party as double-spender.

   We let A' interact in Experiment 3 and simulate the protocol for $\mathcal{Z}$ as follows. The bank keys are constructed honestly and the keys of the honest users are created using the HonestUKg oracle. Withdrawals are performed by interacting with the HonestUWithdraw oracle, and spent coins are constructed with the HonestSpend oracle. By construction of the functionality, $\mathcal{Z}$ has found ($\mathsf{spentcoin}_1, \mathsf{spentcoin}_2$) such that they were not both constructed using the `Spend`, but still form a double-spending. Using this pair, we can construct A' running in Experiment 3 breaking the non-frameability of $\mathcal{EC}$.

As shown, for each hybrid pair we can construct an adversary breaking a security assumption of $\mathcal{EC}$. Therefore it follows that if $\mathcal{EC}$ is secure, then the theorem follows.

5 A Construction

In this section we describe a secure scheme for electronic cash based on general methods. We first define the primitives, then we give the algorithms, and finally we prove that our scheme is secure according to our definitions.

5.1 Primitives

Our construction uses a signature scheme, a commitment scheme, and simulation sound non-interactive zero-knowledge proofs of knowledge (NIZK). Here we briefly describe these notions, and refer to Appendix A for precise definitions of these well-known concepts.

Digital Signatures A signature scheme $\mathcal{SS} = (\mathsf{Kg}, \mathsf{Sig}, \mathsf{Vf})$ is *correct* if for $(\mathsf{pk}, \mathsf{sk})$ generated by Kg and any message m it holds that $\mathsf{Vf}_{\mathsf{pk}}(m, \mathsf{Sig}_{\mathsf{sk}}(m)) = 1$. $\mathcal{SS}$ is secure against chosen-message attacks, *CMA*-secure [16], if it is infeasible to produce valid message-signature pair for *any* message, even if the adversary has access to a signing oracle $\mathsf{Sig}_{\mathsf{sk}}(\cdot)$.

Commitment Schemes A commitment scheme $\mathcal{COM} = (\mathsf{Commit}, \mathsf{Reveal})$ for messages of length κ is secure if a commitment is both hiding and binding, i.e., that the adversary gains any useful information about the committed value from $(c, r)\mathsf{Commit}(m)$, and that given (c, r) it is infeasible to find $(m', r') \neq (c, r)$ such that $\mathsf{Reveal}(c, m', r') = 1$.

Non-interactive Proofs of Knowledge We use non-interactive zero-knowledge proofs of knowledge, or NIZKs, in our construction. Given a language $L \in \mathbf{NP}$ with witness relation R and $x \in L$, a NIZK (P, V) enables a prover P to prove to a verifier V that she knows a witness w such that $(x, w) \in R$.

A proof system is said to be zero-knowledge if there exists a simulator which produces proofs indistinguishable from real proofs, and the condition for it to be called non-interactive should be obvious. A NIZK is complete if for any $(x, w) \in R$ it holds that $V(x, P(x, w)) = 1$ and sound if for any algorithm A the probability that $V(x, \pi) = 1$ and $x \notin L$ is negligible, where $(x, \pi) \leftarrow A(\xi)$. A NIZK is a proof of knowledge (NIZK-PK) if there exists an extractor which, if allowed to choose the CRS, can extract a witness.

In the experiments we give the adversary access to oracles which sometimes produce simulated proofs. Potentially this could help the adversary in producing false proofs. The stronger notion of simulation sound NIZKs requires that no adversary can break the soundness even if given access to a simulator.

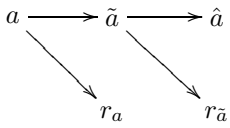
It has been shown [23,25] that a simulation sound NIZK exists for any **NP**-relation if trapdoor permutations exist. Combined with the standard method of [1] it can be turned into a NIZK-PK under the assumption that dense encryption schemes exist [1,26]. We detail the construction in Appendix A.

We need NIZKs for languages on the form $L = \{x \in \mathrm{Im}(f)\}$. Here the obvious witness relation is $R = \{(x, w) : f(w) = x\}$. For such a relation we use the notation NIZK($\omega : f(\omega) = x$) to denote a NIZK. We use Greek letters to denote variables in the witness, i.e., known only to the prover, and Latin letters for variables known both to the prover and to the verifier. We denote the verifier by Vf. It will be clear from context for which relation the proof is.

5.2 The Protocol

Here we give the definitions for algorithms and protocols that form a scheme for electronic cash in the CRS-model. We begin by giving an informal description. In order to identify double-spenders, we use Ferguson's [13] trick of letting each coin contain a line $y = ax + \mathsf{upk}$, such that the coordinate of its intersection with the y-axis coincides with the identity upk of the owner. When spending the coin, one point on the line is revealed. Thus one spending of a coin gives no information about its owner, whereas the identity can be computed from two spendings of the same coin.

When withdrawing a coin, the user randomly selects the slope a. It computes a commitment $\tilde{a}$ of a and a commitment $\hat{a}$ of $\tilde{a}$ with associated randomness r_a and $r_{\tilde{a}}$.



A two-step commitment $\hat{b}$ of the user public key upk is also computed. Then $\hat{a}, \hat{b}$ is sent to the bank and signed, and when a coin is spent, $\tilde{a}$ is revealed together with a proof of knowledge that it is correctly formed. Intuitively this gives anonymity, since $\tilde{a}, \tilde{b}$ cannot be linked to $\hat{a}, \hat{b}$. It also assures that double-spenders are detected, since it is infeasible to open $\hat{a}, \hat{b}$ in more than one way. It can be noted that the signing mechanism is similar to the blind signature scheme found by Fischlin [14].

We let $\mathcal{SS} = (\mathsf{Kg}, \mathsf{Sig}, \mathsf{Vf})$ be a CMA-secure signature scheme and we let $\mathcal{COM} = (\mathsf{Commit}, \mathsf{Reveal})$ be a binding and hiding commitment scheme. Such signature schemes and commitment schemes exist if one-way functions exist [22,15], and thus certainly if trapdoor permutations exist.

We use NIZKs for two different relations in the withdrawal and the spending protocols. The NIZKs work in the common reference string model. We will denote the reference string ξ. Each NIZK needs its own CRS, so we divide ξ into two parts so that $\xi = \xi_0 || \xi_1$ such that both ξ_0 and ξ_1 are long enough. We let $S(\mathsf{setup}, 1^\kappa)$ create both ξ_0 and ξ_1, store the two secrets in $\mathsf{simstate}$, and return $\xi_0 || \xi_1$. Now S can simulate and extract proofs for both relations.

We start with the key generation algorithms. Key generation for the bank consists of generating a key for the signature scheme. The key for the user is created by drawing a value at random and computing a commitment to the value.

The private key is the random value and coin tosses used in the commitment, and the public key is the commitment. As noted above, we do not include user registration at the bank as part of the protocol.

Definition 7 ($\mathsf{BKg}(1^\kappa)$)**.**
$(\mathsf{bpk}, \mathsf{bsk}) \leftarrow \mathsf{Kg}(1^\kappa)$

Definition 8 ($\mathsf{UKg}(1^\kappa)$)**.**
$t \leftarrow_R \{0,1\}^\kappa$
$(\mathsf{upk}, r_t) \leftarrow \mathsf{Commit}(t)$
$\mathsf{usk} \leftarrow (t, r_t)$
return $(\mathsf{upk}, \mathsf{usk})$

Merchant registration is straight-forward. Since our protocol does not use a merchant secret key, registration simply consists of handing the merchant identity to the bank, which registers the merchant.

The coin withdrawal protocol is a two-round protocol with the following steps.

1. The user draws a value a at random and commits to a in two steps, i.e., it computes a commitment $\tilde{a}$ to a, and a commitment $\hat{a}$ to $\tilde{a}$. In the same way it computes a two-step commitment $\hat{b}$ to its public key upk. It also constructs a proof $\pi_{\mathcal{U}}$ of knowledge of a a and coin tosses used in the commitments. It hands $\hat{a}, \hat{b}$ and $\pi_{\mathcal{U}}$ to the bank and stores a together with the coin tosses as the coin secret key csk.
2. The bank verifies that the user is allowed to withdraw a coin and that the proof of knowledge is valid. It then signs the user's public key concatenated with $(\hat{a}, \hat{b})$. The coin consists of the signature, $\hat{a}$,$\hat{b}$, the user's public key upk, and the proof $\pi_{\mathcal{U}}$.

More precisely, the withdrawal protocols consists of the following two algorithms.

Definition 9 ($\mathsf{UWithdraw}(\mathsf{msg}, \mathsf{state})$)**.**
Parse state *as* $(\mathsf{upk}, \mathsf{usk})$.
$a \leftarrow_R \{0,1\}^\kappa$
$(\tilde{a}, r_a) \leftarrow \mathsf{Commit}(a)$
$(\hat{a}, r_{\tilde{a}}) \leftarrow \mathsf{Commit}(\tilde{a})$
$(\tilde{b}, r_{\mathsf{upk}}) \leftarrow \mathsf{Commit}(\mathsf{upk})$
$(\hat{b}, r_{\tilde{b}}) \leftarrow \mathsf{Commit}(\tilde{b})$
$\pi_{\mathcal{U}} \leftarrow \mathrm{NIZK}(\alpha, \rho_\alpha, \tilde{\alpha}, \rho_{\tilde{\alpha}}, \tau, \rho_\tau, \rho_{\mathsf{upk}}, \tilde{\beta}, \rho_{\tilde{\beta}}$:
$\quad \mathsf{Reveal}(\tilde{a}, \alpha, r_\alpha) = 1 \wedge \mathsf{Reveal}(\hat{a}, \tilde{\alpha}, \rho_{\tilde{\alpha}}) = 1 \wedge \mathsf{Reveal}(\mathsf{upk}, \tau, \rho_\tau) = 1 \wedge$
$\quad \mathsf{Reveal}(\tilde{\beta}, \mathsf{upk}, \rho_{\mathsf{upk}}) = 1 \wedge \mathsf{Reveal}(\hat{b}, \tilde{\beta}, \rho_{\tilde{\beta}}) = 1)$
return $((a, \tilde{a}, r_a, r_{\tilde{a}}, \tilde{b}, r_{\mathsf{upk}}, r_{\tilde{b}}), (\mathsf{upk}, \hat{a}, \hat{b}, \pi_{\mathcal{U}}))$

Definition 10 ($\mathsf{BWithdraw}(\mathsf{msg}, \mathsf{state})$)**.**
Parse state *as* (bsk).

Parse msg *as* $(\mathsf{upk}, \hat{a}, \hat{b}, \pi_{\mathcal{U}})$.
Quit if user with public key upk *is not allowed to withdraw a coin.*
if $\mathsf{Vf}(\pi_{\mathcal{U}}) = 1$ **then**
  return $(\mathsf{reject}, \emptyset)$
end if
$s \leftarrow_R \mathsf{Sig}_{\mathsf{bsk}}(\mathsf{upk}, \hat{a}, \hat{b})$
$\mathsf{coin} \leftarrow (s, \hat{a}, \hat{b}, \mathsf{upk}, \pi_{\mathcal{U}})$
return $(\mathsf{coin}, \emptyset)$

We also need to be able to verify whether or not a coin has been withdrawn by a certain user by verifying the coin's signature and the user's proof of knowledge.

Definition 11 ($\mathsf{VfCoin}(\mathsf{coin}, \mathsf{upk}, \mathsf{bpk})$)**.**
Parse coin *as* $(s, \hat{a}, \hat{b}, \mathsf{upk}, \pi_{\mathcal{U}})$.
return $\mathsf{Vf}_{\mathsf{bpk}}((\mathsf{upk}, \hat{a}, \hat{b}), s) \wedge \mathsf{Vf}(\pi_{\mathcal{U}})$

To spend a coin the user first checks that the coin is valid. Then it lets (x, y) be a point on the line $y = ax + \mathsf{upk}$, where a is the coin secret key and upk the public key of the user. The point x is chosen as the concatenation of the transaction identity and the merchant identity. The user reveals the values $\tilde{a}$ and $\tilde{b}$. The spent coin consists of (p, x, y), and a proof of knowledge of a and upk such that (x, y) is indeed a point on the line and of a bank signature on $(\mathsf{upk}, \hat{a}, \hat{b})$ as well as of coin tosses such that $\hat{a}$ is a commitment of $\tilde{a}$ and $\hat{b}$ of $\tilde{b}$.

Definition 12 ($\mathsf{Spend}(\mathsf{coin}, \mathsf{usk}, \mathsf{csk}, \mathsf{mid}, \mathsf{tid}, \mathsf{bpk})$)**.**
Parse coin *as* $(s, \hat{a}, \hat{u}, \mathsf{upk}, \pi_{\mathcal{U}})$
Parse usk *as* (t, r_t)
Parse csk *as* $(a, \tilde{a}, r_a, r_{\tilde{a}}, \tilde{b}, r_{\mathsf{upk}}, r_{\tilde{b}})$
if $(\mathsf{Vf}_{\mathsf{bpk}}((\mathsf{upk}, \hat{a}, \hat{b}), s) = 0) \vee (\mathsf{Reveal}(\tilde{a}, a, r_a) = 0) \vee (\mathsf{Reveal}(\hat{a}, \tilde{a}, r_{\tilde{a}}) = 0) \vee$
$(\mathsf{Reveal}(\mathsf{upk}, t, r_t) = 0) \vee (\mathsf{Reveal}(\tilde{b}, \mathsf{upk}, r_{\mathsf{upk}}) = 0) \vee (\mathsf{Reveal}(\hat{b}, \tilde{b}, r_{\tilde{b}}) = 0)$ **then**
  return $\perp$
end if
$x \leftarrow \mathsf{mid}||\mathsf{tid}$
$y \leftarrow ax + \mathsf{upk}$
$\pi \leftarrow \mathrm{NIZK}(\iota, \alpha, \rho_\alpha, \hat{\alpha}, \rho_{\tilde{a}}, \rho_{\mathsf{upk}}, \hat{beta}, \rho_{\tilde{b}}, \sigma, \tau, \rho_\tau :$
  $y = \alpha x + \iota \wedge \mathsf{Reveal}(\tilde{a}, \alpha, \rho_\alpha) = 1 \wedge \mathsf{Reveal}(\hat{\alpha}, \tilde{a}, \rho_{\tilde{a}}) = 1 \wedge \mathsf{Reveal}(\tilde{b}, \iota, \rho_{\mathsf{upk}}) \wedge$
  $\mathsf{Reveal}(\hat{\beta}, \tilde{b}, \rho_{\tilde{b}}) = 1 \wedge \mathsf{Vf}_{\mathsf{bpk}}((\iota, \tilde{\alpha}, \tilde{\beta}), \sigma) = 1 \wedge \mathsf{Reveal}(\iota, \rho_\tau, \tau) = 1)$
$\mathsf{spentcoin} \leftarrow (\tilde{a}, \tilde{b}, x, y, \pi)$
return $\mathsf{spentcoin}$

Verification of a spent coin is straight-forward:

Definition 13 ($\mathsf{VfSpentCoin}(\mathsf{spentcoin},\mathsf{tid},\mathsf{mid},\mathsf{bpk})$)**.**
Parse $\mathsf{spentcoin}$ *as* $(\tilde{a}, \tilde{b}, x, y, \pi)$.
if $x \neq \mathsf{mid}||\mathsf{tid}$ **then**
  return 0

end if
return $\mathsf{Vf}(\pi)$

Finally we give the algorithm to identify a double-spender. A coin is double-spent if the value b appears twice with different values of x. Finding the double-spender is then simply a task of solving the two equations for upk.

Definition 14 ($\mathsf{VfDoubleSpent}(\mathsf{spentcoin}_1, \mathsf{spentcoin}_2, \mathsf{bpk})$)**.**
Parse $\mathsf{spentcoin}_1$ *as* $(\tilde{a}_1, \tilde{b}_1, x_1, y_1, \pi_1)$.
Parse $\mathsf{spentcoin}_2$ *as* $(\tilde{a}_2, \tilde{b}_2, x_2, y_2, \pi_2)$.
if $((\tilde{a}_1, \tilde{b}_1) \neq (\tilde{a}_2, \tilde{b}_2)) \vee (x_1 = x_2)$ **then**
  return $\perp$
end if
$\mathsf{upk} \leftarrow \frac{x_1 y_2 - x_2 y_1}{x_1 - x_2}$
return upk

6 Proof of Security

In this section we prove the following theorem about the scheme $\mathcal{EC}$ = (BKg, UKg, $\mathsf{UWithdraw}$, $\mathsf{BWithdraw}$, VfCoin, Spend, $\mathsf{VfSpentCoin}$, $\mathsf{VfDoubleSpent}$) as defined in Section 5.

Theorem 2. *The scheme for electronic cash $\mathcal{EC}$ is correct and secure in the common reference string model if there exists a family of trapdoor permutations.*

From Theorem 1 this implies the following, where $\mathcal{F}_{\mathcal{CRS}}$ in the common reference string functionality.

Theorem 3. *The protocol π_{AnonEC} using algorithms as defined in Section 5 securely realizes $\mathcal{F}_{\mathrm{AnonEC}}$ in the $\mathcal{F}_{\mathcal{CRS}}$-hybrid model if there exists a family of trapdoor permutations.*

We prove the theorem by showing the five properties. Each lemma holds in the CRS-model under the assumption that a family of trapdoor permutations exists, although this is not stated explicitly.

Lemma 1 (Correctness). *The scheme $\mathcal{EC}$ is correct.*

Proof. Follows by the construction of the algorithms.

Lemma 2 (Unforgeability). *The scheme $\mathcal{EC}$ has unforgeability.*

Proof. Let A be an adversary that is successful in Experiment 2 with non-negligible probability. We show how to use A to construct either a machine A_{cma} breaking the CMA-security of the signature scheme $\mathcal{SS} = (\mathsf{Kg}, \mathsf{Sig}, \mathsf{Vf})$, a machine A_{binding} breaking the binding property of the commitment scheme $\mathcal{COM}$, or a machine $A_{\mathsf{sim-sound}}$ attacking the simulation soundness of the NIZK-PK.

A_{cma} is given a public key pk for the signature scheme as input. The CRS is created using the simulator $(\xi, \mathsf{simstate}) \leftarrow S(\mathsf{setup}, 1^\kappa)$. As in the experiment

for CMA security, A_{cma} has access to a signature oracle. It passes pk as parameter bpk to A. The $\mathsf{BWithdraw}$ oracle is run honestly, except that the signature $\mathsf{Sig}_{\mathsf{bsk}}(\cdot)$ is produced by calling the signature oracle.

Let k be the number of $\mathsf{spentcoin}$ produced by A. Recall A has made l withdrawals using its oracle. This implies A_{cma} has made l calls to the CMA oracle. For each $\mathsf{spentcoin}_i = (\tilde{a}_i, \tilde{b}_i, x_i, y_i, \pi_i)$, A_{cma} calls $S(\mathsf{extract}, (\hat{a}_i, \hat{b}_i, x_i, y_i), \pi_i, \xi, \mathsf{simstate})$ to extract (among other parameters) σ_i, ι, $\hat{\alpha}$, $\hat{\beta}$, ρ_β such that $\mathsf{Vf}_{\mathsf{bpk}}((\iota, \hat{\alpha}, \hat{\beta}), \sigma) = 1$. We now have the following different cases:

1. Signatures on more than l distinct messages are extracted. In this case A_{cma} is successful in breaking the CMA-security of $\mathcal{SS}$.
2. At least one proof π_i cannot be extracted. In this case $A_{\mathsf{sim-sound}}$ uses π_i to break the extractable simulation soundness of the NIZK-PK.
3. All proofs can be extracted, but two proofs yield signatures on the same message $(\iota, \hat{\alpha}, \hat{\beta})$. Since no double-spending is detected, all $(\tilde{\alpha}, \tilde{\beta})$ are distinct. Hence there are two commitments with associated randomness $(\tilde{\alpha}_i, \rho_i)$ and $(\tilde{\alpha}_j, \rho_j)$ such that $\mathsf{Reveal}(\hat{\alpha}, \tilde{\alpha}_i, \rho_i) = \mathsf{Reveal}(\hat{\alpha}, \tilde{\alpha}_j, \rho_j) = 1$. Using these values A_{binding} breaks the binding property of the commitment scheme $\mathcal{COM}$.

Thus we have shown that an adversary which breaks unforgeability can be used to either break the CMA security of $\mathcal{SS}$, the extractable simulation soundness of the NIZK-PK, or break the binding property of $\mathcal{COM}$. Hence $\mathcal{EC}$ has unforgeability.

Lemma 3 (Non-Frameability). *The scheme $\mathcal{EC}$ has non-frameability.*

Proof. Let A be an adversary that succeeds in Experiment 3 with non-negligible probability. We show how to use A to construct either a machine A_{secrecy} breaking the secrecy property of the commitment scheme $\mathcal{COM}$, a machine A_{binding} breaking the binding property, or a machine $A_{\mathsf{ext-sim-sound}}$, which breaks the extractable simulation soundness of the NIZK-PK.

The machine A_{secrecy} takes part in Experiment 7. It creates a CRS using the simulator $(\xi, \mathsf{simstate}) \leftarrow S(\mathsf{setup}, 1^\kappa)$. It randomly draws two message msg_0 and msg_1 which it returns to its experiment, receiving a challenge commitment c. Let the polynomial $p(\kappa)$ be an upper bound on the number of calls to $\mathsf{HonestUKg}$ by A. Since A runs in polynomial time, there exists such a polynomial. A_{secrecy} randomly selects $t \in [p(\kappa)]$. Intuitively A_{secrecy} guesses that A will frame user $\mathcal{U}_t$. All queries to $\mathsf{HonestUKg}$ are executed honestly except for query t, to which A_{secrecy} response c.

When A queries $\mathsf{HonestUWithdraw}$ or $\mathsf{HonestSpend}$ for a user different from $\mathcal{U}_t$, the query is answered honestly. For $\mathcal{U}_t$, the NIZK-PK is constructed by invoking the simulator $S(\mathsf{simulate}, \cdot, \xi, \mathsf{simstate})$.

A outputs a list of coins $(\mathsf{spentcoin}_i)_{i=1}^k$. Let $\mathsf{spentcoin}_i, \mathsf{spentcoin}_j$ be coins such that $\mathsf{VfDoubleSpent}(\mathsf{spentcoin}_i, \mathsf{spentcoin}_j, \mathsf{bpk}) \notin \mathfrak{C}$ and at least one coin has not been produced by $\mathsf{HonestSpend}$. Since A outputs more double-spent coins than was created by $\mathsf{HonestSpend}$, such a pair of coins exists by the pigeon-hole principle. Let $\mathsf{spentcoin}_i = (\tilde{a}_i, \tilde{b}_i, x_i, y_i, \pi_i)$ and $\mathsf{spentcoin}_j = (\tilde{a}_j, \tilde{b}_j, x_j, y_j, \pi_j)$.

By the assumption that they form a double-spending, we have that $(\tilde{a}_i, \tilde{b}_i) = (\tilde{a}_j, \tilde{b}_j)$ and $x_i \neq x_i$. With probability $1/p(\kappa)$, i.e., non-negligible, it holds that $\mathsf{VfDoubleSpent}(\mathsf{spentcoin}_i, \mathsf{spentcoin}_j, \mathsf{bpk}) = \mathsf{upk}_t$. From now on, we will assume that this is the case.

From π_i and π_i the machine A tries to extract $\bar{a}, \bar{r}_a, \bar{\mathsf{upk}}, \bar{r}_{\mathsf{upk}}$ such that $\mathsf{Reveal}(\tilde{a}, \bar{a}, \bar{r}_a) = 1$ and $\mathsf{Reveal}(\tilde{b}, \bar{\mathsf{upk}}, \bar{r}_{\mathsf{upk}}) = 1$. We now have the following cases and subcases:

1. None of the proofs were created by the simulator.
   (a) At least one extraction fails. In such case we can construct a machine $A_{\mathsf{ext-sim-sound}}$ using A and breaking the extractable simulation soundness of the NIZK-PK as follows. $A_{\mathsf{ext-sim-sound}}$ takes part in Experiment 13. When A asks for a NIZK-PK, $A_{\mathsf{ext-sim-sound}}$ uses its simulation oracle. The un-extractable NIZK-PK of A is output by $A_{\mathsf{ext-sim-sound}}$, which wins the extractable simulation soundness experiment with non-negligible probability.
   (b) Both extractions succeed but return $(\bar{a}_i, \bar{r}_a^{(i)}, \bar{\mathsf{upk}}_i, \bar{r}_{\mathsf{upk}}^{(i)}) \neq (\bar{a}_j, \bar{r}_a^{(j)}, \bar{\mathsf{upk}}_j, \bar{r}_{\mathsf{upk}}^{(j)})$. In such a case the extracted values can be used by the machine A_{binding} to break the binding property of $\mathcal{COM}$.
   (c) Both extractions succeed and return consistent values. Since the NIZK-PK also proves that $y_l = ax_l + \mathsf{upk}$, it follows that $\bar{\mathsf{upk}} = \mathsf{upk}$. By extracting τ, ρ_τ such that $\mathsf{Reveal}(\mathsf{upk}, \tau, \rho_\tau) = 1$, the machine A_{secrecy} breaks the secrecy of $\mathcal{COM}$ with non-negligible probability.
2. One proof was created by the simulator. Without loss of generality we assume that the simulator created π_j, and let $a_j, r_a^{(j)}, \mathsf{upk}_j, r_{\mathsf{upk}}^{(j)}$ be the values used when responding to the oracle query.
   (a) The extraction of the proof π_i fails. If this is the case, then $A_{\mathsf{ext-sim-sound}}$ proceeds as in Step 1a to break the simulation soundness of the NIZK-PK.
   (b) The extraction of π_i succeeds but yields $(\bar{a}_i, \bar{r}_a^{(i)}, \bar{\mathsf{upk}}_i, \bar{r}_{\mathsf{upk}}^{(i)}) \neq (a_j, r_a^{(j)}, \mathsf{upk}_j, r_{\mathsf{upk}}^{(j)})$. Then, as in Step 1b, the binding property of $\mathcal{COM}$ is broken.
   (c) The extraction π_i succeeds and gives consistent values. Then, as in Step 1c, the secrecy of $\mathcal{COM}$ is broken.
3. Both proofs were created by the simulator. Since, by assumption, at least one coin was not created by an oracle query, this cannot happen.

We have shown that if A breaks the non-frameability property, then at least one of the machines A_{secrecy}, A_{binding}, and $A_{\mathsf{ext-sim-sound}}$ is successful with non-negligible probability. Since this breaks the assumption, the scheme $\mathcal{EC}$ has non-frameability.

Lemma 4 (Anonymity). *The scheme $\mathcal{EC}$ has anonymity.*

Proof. Assume A wins in the anonymity experiment 4 with non-negligible probability. We show how to construct either A_{secrecy} breaking the secrecy of the

commitment scheme $\mathcal{COM}$ or a machine $A_{\mathsf{ad-ind}}$ breaking the adaptive indistinguishability of the NIZK-PK.

We define two variants of the scheme $\mathcal{EC}$. We let $\mathcal{EC}'$ be $\mathcal{EC}$ with the modification that the CRS is created by the simulator for the NIZK-PK in Spend, $(\xi, \mathsf{simstate}) \leftarrow S(\mathsf{setup}, 1^\kappa)$ and the NIZK-PK is generated by the simulator, and we let $\mathcal{EC}''$ be $\mathcal{EC}'$ with the difference that the commitment scheme of UWithdraw used to produce $\tilde{a}$ is replaced by a commitment scheme with perfect secrecy.

Since a spentcoin in $\mathcal{EC}''$ contains no information about the spender of a coin, the advantage of A when attacking $\mathcal{EC}''$ is 0. We now have the following two cases.

1. The advantage of A when attacking $\mathcal{EC}'$ is non-negligible. We show how to use A to construct A_{secrecy} which successfully attacks the secrecy of the commitment scheme $\mathcal{COM}$. A_{secrecy} takes part in Experiment 7. All calls to HonestUKg and HonestSpend are answered honestly. When A outputs $(i_0, i_1, \mathsf{mid}, \mathsf{tid})$, A_{secrecy} outputs $(\mathsf{upk}_{i_0}, \mathsf{upk}_{i_1})$ to its experiment, receiving a commitment c in response. Then A_{secrecy} uses c as $\tilde{a}$ when creating the challenge spentcoin and constructs the rest of the coin honestly. (Since $\mathcal{EC}'$ only uses simulated NIZK-PKs, not knowing the message of c is not a problem.) A outputs a bit d, which A_{secrecy} outputs in its experiment.
   From the construction it follows that A_{secrecy} is successful when A is, and hence breaks the secrecy of $\mathcal{COM}$ with non-negligible probability.
2. The advantage of A when attacking $\mathcal{EC}'$ is negligible. In such case we can use A to construct $A_{\mathsf{ad-ind}}$ breaking the adaptive indistinguishability of the NIZK-PK. $A_{\mathsf{ad-ind}}$ takes part in Experiment 10 and 11 while executing Experiment 4 for A. All parts of the experiment are executed honestly, except that NIZK-PKs are created by using the oracle. If A is successful, $A_{\mathsf{ad-ind}}$ responds that it is interacting with Experiment 10, and otherwise that it is interacting with Experiment 11. Since A is successful only when NIZK-PKs are genuine, $A_{\mathsf{ad-ind}}$ has a non-negligible advantage.

We have shown that a machine breaking the anonymity of $\mathcal{EC}$ can be made into a machine either breaking the secrecy of the commitment scheme or a machine breaking the adaptive indistinguishability of the proof system. Since such machines contradicts the assumptions, we conclude that $\mathcal{EC}$ has anonymity.

Lemma 5 (Exculpability). *The scheme $\mathcal{EC}$ has exculpability.*

Proof. Let A be an adversary which wins in Experiment 5 with non-negligible probability. We use A to construct either a machine A_{secrecy} breaking the secrecy property of the commitment scheme $\mathcal{COM}$, A_{binding} breaking the binding property of the $\mathcal{COM}$, a machine $A_{\mathsf{ext-sim-sound}}$ breaking the simulation soundness of the NIZK-PK, or a machine $A_{\mathsf{ad-ind}}$ breaking the adaptive indistinguishability of the NIZK-PK.

We define the scheme $\mathcal{EC}'$ being equal to $\mathcal{EC}$ with the difference that the CRS is setup using the simulator $(\xi, \mathsf{simstate}) \leftarrow S(\mathsf{setup}, 1^\kappa)$ and the NIZK-PK of UWithdraw is created using the simulator.

First assume A has negligible probability of breaking the exculpability property of $\mathcal{EC}'$. Then we can use A to construct $A_{\mathsf{ad-ind}}$ in the following way. $A_{\mathsf{ad-ind}}$ takes part in Experiment 10 or 11. It invokes A, answering all queries honestly except that the NIZK-PK is created using its oracle. Hence, if $A_{\mathsf{ad-ind}}$ is run in Experiment 10, it will run $\mathcal{EC}$ for A, but if it is run in Experiment 11, it will run $\mathcal{EC}'$. If A is successful, then $A_{\mathsf{ad-ind}}$ returns 0, and otherwise it returns 1. From the construction of $A_{\mathsf{ad-ind}}$ it follows that it breaks the adaptive indistinguishability of the NIZK-PK.

Now assume A has non-negligible probability in winning the exculpability experiment against $\mathcal{EC}'$. Then there are two possiblities.

1. The NIZK-PK $\pi_{\mathcal{U}}$ of coin output by A has been constructed by the simulator. This implies that $\pi_{\mathcal{U}}$ was created by $\mathsf{HonestUWithdraw}$ for a certain user and coin secret key $\mathsf{usk}_i, \mathsf{csk}_i$. Hence the coin can be spent using $\mathsf{usk}_i, \mathsf{csk}_i$.
2. The NIZK-PK $\pi_{\mathcal{U}}$ of coin output by A has not been constructed by the simulator. In this case we can construct A_{secrecy} breaking the secrecy of the commitment scheme as follows. The $p(\kappa)$ be an upper bound on the number of calls to $\mathsf{HonestUKg}$. Since A is polynomial, such a bound exists. Let $t \leftarrow_R [p(\kappa)]$. Informally A_{secrecy} guesses that A will frame $\mathcal{U}_t$. A_{secrecy} randomly chooses τ_0, τ_1. It answers queries honestly, except that when asked to generate the public key for $\mathcal{U}_t$, it returns the challenge commitment c created by its experiment.
   With probability $1/p(\kappa)$ A produces a coin that can be verified to belong to $\mathcal{U}_t$. Assume this is the case. Then A_{secrecy} uses the extractor to extact τ, r_τ such that $\mathsf{Reveal}(\mathsf{upk}_t, \tau, r_\tau) = 1$. We now have three cases.
   (a) There exists d such that $\tau_d = \tau$. Then A_{secrecy} returns d hence breaks the secrecy of the commitment scheme $\mathcal{COM}$
   (b) No such d exists. Then two openings of commitment upk_t has been found, violating the binding property of $\mathcal{COM}$.
   (c) The extraction fails, breaking the extractable simulation soundness property of the NIZK-PK.

We can conclude that a machine breaking the exculpability property of $\mathcal{EC}$ implies a machine breaking one of the assumptions. Therefore $\mathcal{EC}$ has exculpability.

7 Conclusions

We have given definitions of security that are stronger than what has previously been suggested. We also show that the requirements are realistic by giving a scheme fulfilling them under the assumption of existence of a family of trapdoor permutations.

It remains an open problem to construct a practical scheme which is secure in our sense under some well-established number-theoretical assumptions.

We would like to thank Johan Håstad for helpful discussions and Douglas Wikström for pointing out the similarities to [14].

References

1. Giuseppe Persiano A. De Santis. Zero-knowledge proofs of knowledge without interaction (extended abstract). In *33rd IEEE Symposium on Foundations of Computer Science – FOCS*, pages 427–436. IEEE Computer Society Press, 1992.
2. M. Bellare, D. Micciancio, and B. Warinschi. Foundations of group signatures: Formal definitions, simplified requirements, and a construction based on general assumptions. In *Advances in Cryptology – EUROCRYPT 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 614–629. Springer Verlag, 2003.
3. M. Bellare, H. Shi, and C. Zhang. Foundations of group signatures: The case of dynamic groups. In *RSA Conference 2005, Cryptographers' Track 2005*, volume 3376 of *Lecture Notes in Computer Science*, pages 136–153. Springer Verlag, 2005. Full version at `http://eprint.iacr.org/2004/077`.
4. A. Bender, J. Katz, and R. Morselli. Ring signatures: Stronger definitions, and constructions without random oracles. In *Theory of Cryptography Conference – TCC 2006*, volume 3876 of *Lecture Notes in Computer Science*, pages 60 – 79. Springer Verlag, 2006. Full version at `http://eprint.iacr.org/2005/304`.
5. M. Blum. How to exchange (secret) keys. *ACM Transactions on Computer Systems – TOCS*, 1(2):175–193, 1983.
6. M. Blum, P. Feldman, and S. Micali. Non-interactive zero-knowledge and its applications. In *20th ACM Symposium on the Theory of Computing – STOC*, pages 103–118. ACM Press, 1988.
7. S. Brands. Untraceable off-line cash in wallets with observers. In *Advances in Cryptology – CRYPTO'93*, volume 773 of *Lecture Notes in Computer Science*, pages 302–318. Springer Verlag, 1994.
8. J. Camenisch, S. Hohenberger, and A. Lysyanskaya. Compact e-cash. In *Advances in Cryptology – EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 302–321. Springer Verlag, 2005. Full version at `http://eprint.iacr.org/2005/060`.
9. R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd IEEE Symposium on Foundations of Computer Science – FOCS*. IEEE Computer Society Press, 2001. Full version at `http://eprint.iacr.org/2000/067`.
10. M. Chase and A. Lysyanskaya. On signatures of knowledge. Cryptology ePrint Archive, Report 2006/184, 2006. `http://eprint.iacr.org/2006/184`.
11. D. Chaum, A. Fiat, and M. Naor. Untraceable electronic cash. In *Advances in Cryptology – CRYPTO'88*, volume 403 of *Lecture Notes in Computer Science*, pages 319–327. Springer Verlag, 1990.
12. U. Feige, D. Lapidot, and A. Shamir. Multiple non-interactive zero-knowledge proofs under general assumptions. *SIAM Journal on Computing*, 29(1):1–28, 1999.
13. N.T. Ferguson. Single term off-line coins. In *Advances in Cryptology – EUROCRYPT'93*, volume 765 of *Lecture Notes in Computer Science*, pages 318–328. Springer Verlag, 1993.
14. M. Fischlin. Round-optimal composable blind signatures in the common reference string model. In *Advances in Cryptology – CRYPTO2006*, volume 4117 of *Lecture Notes in Computer Science*, pages 60–77. Springer Verlag, 2006.
15. O. Goldreich and L. A. Levin. A hard-core predicate for all one-way functions. In *21st ACM Symposium on the Theory of Computing – STOC*, pages 25–32. ACM Press, 1989.

16. S. Goldwasser, S. Micali, and R. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, 1988.
17. A. Kiayias and M. Yung. Efficient secure group signatures with dynamic joins and keeping anonymity against group managers. In *Mycrypt 2005*, volume 3715 of *Lecture Notes in Computer Science*, pages 151–170. Springer Verlag, 2005.
18. M. Liskov and S. Micali. Amortized e-cash. In *Financial Cryptography 2001*, volume 2339 of *Lecture Notes in Computer Science*, pages 1–20. Springer Verlag, 2001.
19. T. Nakanishi, M. Shiota, and Y. Sugiyama. An efficient online electronic cash with unlinkable exact payments. In *Information Security Conference – ISC 2004*, volume 3225 of *Lecture Notes in Computer Science*, pages 367–378. Springer Verlag, 2004.
20. T. Nakanishi and Y. Sugiyama. Unlinkable divisible electronic cash. In *Information Security Workshop – ISW 2000*, volume 1975 of *Lecture Notes in Computer Science*, pages 121–134. Springer Verlag, 2000.
21. T. Okamoto and K. Ohta. Universal electronic cash. In *Advances in Cryptology – CRYPTO'91*, volume 576 of *Lecture Notes in Computer Science*, pages 324–337. Springer Verlag, 1992.
22. J. Rompel. One-way functions are necessary and sufficient for secure signatures. In *22nd ACM Symposium on the Theory of Computing – STOC*, pages 387–394. ACM Press, 1990.
23. A. Sahai. Non-malleable non-interactive zero-knowledge and adaptive chosen-ciphertext security. In *40th IEEE Symposium on Foundations of Computer Science – FOCS*, pages 543–553. IEEE Computer Society Press, 1999.
24. T. Sander and A. Ta-Shma. Auditable, anonymous electronic cash. In *Advances in Cryptology – CRYPTO'99*, volume 1666 of *Lecture Notes in Computer Science*, pages 555–572. Springer Verlag, 1999.
25. A. De Santis, G. Di Crescenzo, R. Ostrovsky, G. Persiano, and A. Sahai. Robust non-interactive zero knowledge. In *Advances in Cryptology – CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 566–598. Springer Verlag, 2001.
26. A. De Santis, G. Di Crescenzo, and G. Persiano. Necessary and sufficient assumptions for non-interactive zero-knowledge proofs of knowledge for all np relations. In *27th International Colloquium on Automata, Languages and Programming – ICALP*, volume 1853 of *Lecture Notes in Computer Science*, pages 451–462. Springer Verlag, 2000.
27. V. Varadharajan, K.Q. Nguyen, and Y. Mu. On the design of efficient RSA-based off-line electronic cash schemes. *Theoretical Computer Science*, 226:173–184, 1999.
28. V. Wei. More compact e-cash with efficient coin tracing. Cryptology ePrint Archive, Report 2005/411, 2005. `http://eprint.iacr.org/2005/411`.

A Definitions

A.1 Trapdoor Permutations

Definition 15 (Trapdoor Permutation Family). *A* trapdoor permutation family *is a tuple of probabilistic polynomial time Turing machines* $\mathcal{F} = (\mathsf{Gen}, \mathsf{Eval}, \mathsf{Invert})$ *such that:*

1. $\mathsf{Gen}(1^\kappa)$ *outputs a pair* (f, f^{-1}) *such that* f *is a permutation of* $\{0,1\}^\kappa$.

2. $\mathsf{Eval}(1^\kappa, f, x)$ *is a deterministic algorithm which on input* f*, where* $(f, f^{-1}) \in \mathsf{Gen}(1^\kappa)$*, and* $x \in \{0,1\}^\kappa$ *outputs* $y = f(x)$*.*
3. $\mathsf{Invert}(1^\kappa, f^{-1}, y)$ *is a deterministic algorithm which on input* f^{-1}*, where* $(f, f^{-1}) \in \mathsf{Gen}(1^\kappa)$*, and* $y \in \{0,1\}^\kappa$ *outputs some* $x = f^{-1}(y)$*.*
4. *For all* κ*,* $(f, f^{-1}) \in \mathsf{Gen}(1^\kappa)$*, and* $x \in \{0,1\}^\kappa$ *we have* $f^{-1}f(x) = x$*.*
5. *For all adversaries* $A \in \mathrm{PPT}^*$*, the following is negligible*

$$\Pr[(f, f^{-1}) \leftarrow \mathsf{Gen}(1^\kappa), \quad x \leftarrow \{0,1\}^\kappa, \quad A(f, f(x)) = f^{-1}(y)] \ .$$

A.2 Signature Schemes

A signature scheme $\mathcal{SS} = (\mathsf{Kg}, \mathsf{Sig}, \mathsf{Vf})$ is secure against chosen-message attacks, *CMA*-secure [16], if it is infeasible to produce valid message-signature pair for *any* message, even if the adversary has access to a signing oracle $\mathsf{Sig}_{\mathsf{sk}}(\cdot)$. Formally we use the following experiment for the definition

Experiment 6 (CMA, $\mathbf{Exp}^{\mathsf{cma}}_{\mathcal{SS},A}(\kappa)$).
$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Kg}(\kappa)$
$(m, \sigma) \leftarrow A^{\mathsf{Sig}_{\mathsf{sk}}(\cdot)}(\mathsf{pk})$
return $\mathsf{Vf}_{\mathsf{pk}}(m, s) = 1$

The advantage of the adversary is defined as

$$\mathbf{Adv}^{\mathsf{cma}}_{\mathcal{SS},A}(\kappa) = \Pr[\mathbf{Exp}^{\mathsf{cma}}_{\mathcal{SS},A}(\kappa) = 1] \ .$$

A signature scheme $\mathcal{SS}$ is *CMA*-secure if $\mathbf{Adv}^{\mathsf{cma}}_{\mathcal{SS},A}(\kappa)$ is negligible for all polynomial-time adversaries A.

A.3 Commitment Schemes

A (non-interactive) commitment scheme $\mathcal{COM} = (\mathsf{Commit}, \mathsf{Reveal})$ consists of two algorithms, the commitment algorithm and the reveal algorithm. The commitment algorithm takes as input a message $\mathsf{msg} \in \{0,1\}^\kappa$ and outputs a pair (c, r). The reveal algorithm takes a commitment c, a message msg, and the secret r and determines whether or not c is a commitment to msg under commitment secret r.

The following two experiments defines secrecy and binding of a commitment scheme.

Experiment 7 (Secrecy, $\mathbf{Exp}^{\mathsf{secrecy}-b}_{\mathcal{COM},A}(\kappa)$).
$(\mathsf{msg}_0, \mathsf{msg}_1, \mathsf{state}) \leftarrow A(\mathsf{choose}, 1^\kappa)$
$(c, r) \leftarrow \mathsf{Commit}(\mathsf{msg}_b)$
$d \leftarrow A(\mathsf{guess}, c, \mathsf{state})$
return d

The advantage of an adversary A is

$$\mathbf{Adv}^{\mathsf{secrecy}}_{\mathcal{COM},A}(\kappa) = |\Pr[\mathbf{Exp}^{\mathsf{secrecy}-0}_{\mathcal{COM},A}(\kappa) = 1] - \Pr[\mathbf{Exp}^{\mathsf{secrecy}-1}_{\mathcal{COM},A}(\kappa) = 1]| \ .$$

The commitment scheme $\mathcal{COM}$ has secrecy if $\mathbf{Adv}^{\mathsf{secrecy}}_{\mathcal{COM},A}(\kappa)$ is negligible for any polynomial-time adversary A.

Experiment 8 (Binding, $\mathbf{Exp}^{\mathsf{binding}}_{\mathcal{COM},A}(\kappa)$).
$(c, r_0, \mathsf{msg}_0, r_1, \mathsf{msg}_1) \leftarrow A(\mathsf{guess})$
if $(\mathsf{Reveal}(c, \mathsf{msg}_0, r_0) = \mathsf{Reveal}(c, \mathsf{msg}_1, r_1) = 1) \wedge (\mathsf{msg}_0 \neq \mathsf{msg}_1)$ **then**
  return 1
else
  return 0
end if

The advantage of an adversary A is

$$\mathbf{Adv}^{\mathsf{binding}}_{\mathcal{COM},A}(\kappa) = \Pr[\mathbf{Exp}^{\mathsf{binding}}_{\mathcal{COM},A}(\kappa) = 1] \ .$$

The commitment scheme $\mathcal{COM}$ is binding if $\mathbf{Adv}^{\mathsf{binding}}_{\mathcal{COM},A}(\kappa)$ is negligible for any polynomial-time adversary A.

One could give stronger definitions, but in our case the above experiments suffice. As an example, they do not rule out the existence of an adversary which, after seeing one commitment, creates another commitment to the same value, which can be opened after the first commitment has been opened.

It is known that secret and binding commitment schemes exist if there exists a family of one-way permutations [15]. The construction even gives a perfectly binding scheme, i.e., even an unbounded adversary cannot decommit to more than one value.

A.4 Indistinguishable Encryption Schemes

Informally an encryption scheme $\mathcal{CS} = (\mathsf{Kg}, E, D)$ is called indistinguishable if it is infeasible to distinguish between the encryptions of two plaintexts of the same length. The experiment below formalizes this assumption.

Experiment 9 (Indistinguishability, $\mathbf{Exp}^{\mathsf{ind}-b}_{\mathcal{CS},A}(\kappa)$).
$(\mathsf{msg}_0, \mathsf{msg}_1, \mathsf{state}) \leftarrow A(\mathsf{choose}, 1^\kappa)$
$c \leftarrow E(\mathsf{msg}_b)$
$d \leftarrow A(\mathsf{guess}, c, \mathsf{state})$
return d

The advantage of an adversary A is

$$\mathbf{Adv}^{\mathsf{ind}}_{\mathcal{CS},A}(\kappa) = |\Pr[\mathbf{Exp}^{\mathsf{ind}-0}_{\mathcal{CS},A}(\kappa) = 1] - \Pr[\mathbf{Exp}^{\mathsf{ind}-1}_{\mathcal{CS},A}(\kappa) = 1]| \ .$$

The encryption scheme $\mathcal{CS}$ is indistinguishable if $\mathbf{Adv}^{\mathsf{ind}}_{\mathcal{COM},A}(\kappa)$ is negligible for any polynomial-time adversary A.

The notion of indistinguishability is equivalent to the well-known definition of semantic security, which informally says that no information about the plain-text can be efficiently computed from the cipher-text. We use the terms "indistinguishable encryption scheme" and "semantically secure encryption scheme" interchangeably in this text.

A.5 Proofs of Knowledge

Non-interactive zero-knowledge proofs (NIZK) were introduced by Blum, Feldman, and Micali [6]. Several works have since refined and extended the notion in various ways. Following [2] we employ the definition of adaptive zero-knowledge for NIZK introduced by Feige, Lapidot, and Shamir [12] and we use the notion of simulation soundness introduced by Sahai [23]. The notion of simulation soundness is strengthened by De Santis et al. [25]. In contrast to [2], the NIZK we use must be adaptive zero-knowledge for polynomially many statements, and not only for a single statement. The requirement on simulation soundness is in fact unchanged compared with [2], i.e. single statement simulation soundness suffices.

Definition 16 (NIPS). *A triple $(p(\kappa), P, V)$ is an* efficient adaptive non-interactive proof system *(NIPS) for a language $L \in \mathbf{NP}$ with witness relation R if $p(\kappa)$ is a polynomial and P and V are probabilistic polynomial time machines such that*

1. *Completeness. $(x, w) \in R$ and $\xi \in \{0,1\}^{p(\kappa)}$ implies $V(x, P(x, w, \xi), \xi) = 1$.*
2. *Soundness. For all computable functions A, $\Pr_{\xi \in \{0,1\}^{p(\kappa)}}[A(\xi) = (x, \pi) \wedge x \notin L \wedge V(x, \pi, \xi) = 1]$ is negligible in κ.*

We suppress p in our notation of a NIPS and simply write (P, V).

Loosely speaking a non-interactive zero-knowledge proof system is a NIPS, which is also zero-knowledge, but there are several flavors of zero-knowledge. We need a NIZK which is adaptive zero-knowledge (for a single statement) in the sense of Feige, Lapidot, and Shamir [12].

Experiment 10 (Adaptive Indistinguishability, $\mathbf{Exp}^{\mathsf{ad-ind-0}}_{(P,V,S),A}(\kappa)$).

$\xi \leftarrow_R \{0,1\}^{f(\kappa)}$
$(\mathsf{state}, x, w) \leftarrow A(\mathsf{setup}, \xi)$
while $(x, w) \in R$ **do**
  $(\mathsf{state}, x, w) \leftarrow A(\mathsf{choose}, P(x, w, \xi))$
end while
return $A(\mathsf{guess}, \mathsf{state})$

Experiment 11 (Adaptive Indistinguishability, $\mathbf{Exp}^{\mathsf{ad-ind-1}}_{(P,V,S),A}(\kappa)$).

$(\xi, \mathsf{simstate}) \leftarrow S(1^\kappa)$
$(\mathsf{state}, x, w) \leftarrow A(\mathsf{setup}, \xi)$
while $(x, w) \in R$ **do**
  $(\mathsf{state}, x, w) \leftarrow A(\mathsf{choose}, S(x, \xi))$

end while
return $A(\mathsf{guess}, \mathsf{state})$

The advantage in the experiment is defined

$$\mathbf{Adv}^{\mathsf{ad-ind}}_{(P,V,S),A}(\kappa) = |\Pr[\mathbf{Exp}^{\mathsf{ad-ind}-0}_{(P,V,S),A}(\kappa) = 1] - \Pr[\mathbf{Exp}^{\mathsf{ad-ind}-1}_{(P,V,S),A}(\kappa) = 1]|$$

and the notion of adaptive zero-knowledge is given below.

Definition 17 (Adaptive Zero-Knowledge (cf. [12])). *A NIPS (P, V) is adaptive zero-knowledge (NIZK) if there exists a polynomial time Turing machine S such that $\mathbf{Adv}^{\mathsf{ad-ind}}_{(P,V,S),A}(\kappa)$ is negligible for all $A \in \mathcal{A}$.*

In cryptographic proofs one often performs hypothetic experiments where the adversary is run with simulated NIZKs. If the experiment simulates NIZKs to the adversary, the adversary could potentially gain the power to compute valid proofs of false statements. For a simulation sound NIZK this is not possible.

Experiment 12 (Simulation Soundness, $\mathbf{Exp}^{\mathsf{sim-sound}}_{(P,V,S),A}(\kappa)$ (cf. [25])).
$(\xi, \mathsf{simstate}) \leftarrow S(\mathsf{setup}, 1^\kappa)$
$(x, \pi) \leftarrow A^{S(\mathsf{simulate},\cdot,\xi,\mathsf{simstate})}(\mathsf{guess}, \xi)$
if $(\pi \notin \mathfrak{R}_S) \wedge (x \notin L) \wedge (V(x, \pi, \xi) = 1)$ **then**
  return 1
else
  return 0
end if

Definition 18 (Simulation Soundness (cf. [23,25])). *A NIZK (P, V) with polynomial time simulator S for a language L is unbounded simulation sound if*

$$\mathbf{Adv}^{\mathsf{sim-sound}}_{(P,V,S),A}(\kappa) = \mathbf{Exp}^{\mathsf{sim-sound}}_{(P,V,S),A}(\kappa)$$

is negligible for all $A \in \mathcal{A}$.

De Santis et al. [25] extend the results in [12] and [23] and prove the following result.

Theorem 4. *If there exists a family of trapdoor permutations, then there exists a simulation sound NIZK for any language in* **NP** *in the CRS-model.*

In the this paper we abbreviate "efficient non-interactive adaptive zero-knowledge unbounded simulation sound proof" by NIZK.

It is important to note that the above definitions do not require that it is possible to extract the witness, i.e., they are not proofs of knowledge. To our knowledge, there are no results on the existence of simulation-sound proofs of knowledge, although signatures of knowledge [10] are similar.

One must be careful when defining the experiment for extractability. As for simulation-soundness, we want to give the adversary the ability to request simulated proofs for theorems of its choice, and if it outputs a valid proof, the

extractor should be able to extract a witness. In the original definitions of NIZK proofs of knowledge [1,25], *soundness* and *validity*, i.e., the requirement on extractability, are two separate properties. Such a definition would be hard to use when designing protocols. In a protocol, we need to produce a single CRS which is used both for simulation and for extraction. Therefore it makes sense to combine the two properties in a single experiment.

Experiment 13 (Extractable Simulation Soundness, $\mathbf{Exp}^{\mathsf{ext-sim-sound}}_{(P,V,S),A}(\kappa)$ (cf. [25])).

$(\xi, \mathsf{simstate}) \leftarrow S(\mathsf{setup}, 1^\kappa)$
$(x, \pi) \leftarrow A^{S(\mathsf{simulate},\cdot,\xi,\mathsf{simstate})}(\mathsf{guess}, \xi)$
$w \leftarrow S(\mathsf{extract}, x, \pi, \xi, \mathsf{simstate})$
if $(\pi \notin \mathfrak{R}_S) \wedge ((x, w) \notin R) \wedge (V(x, \pi, \xi) = 1)$ **then**
  return 1
else
  return 0
end if

Definition 19 (Extractable Simulation Soundness). *A NIZK (P, V) with polynomial time simulator S for a language L is unbounded extractable simulation sound if*

$$\mathbf{Adv}^{\mathsf{ext-sim-sound}}_{(P,V,S),A}(\kappa) = \mathbf{Exp}^{\mathsf{ext-sim-sound}}_{(P,V,S),A}(\kappa)$$

is negligible for all $A \in \mathcal{A}$.

We now give a construction of an extractable simulation sound proof system based on an unbounded simulation sound proof system. The idea behind the construction is the same as for [1] which is also used in [10], namely to encrypt the witness using a semantically secure encryption scheme where the public key is derived from the common reference string. Extraction is performed by letting the extractor choose the CRS in such a way that it knows the private key.

Let L be a language with witness relation R, i.e., $x \in L$ exactly when there exists w such that $(x, w) \in R$. We define a proof system (P, V) with simulator S and prove that it is an unbounded simulation sound zero-knowledge proof of knowledge. Note that S plays the role both of the simulator and the extractor.

In all the below experiments, we let the common reference string ξ consist of two parts, ξ_0 and ξ_1. We let pk be the public key defined by ξ_0 for the encryption scheme $\mathcal{CS} = (\mathsf{Kg}, E, Dec)$, and let sk be the corresponding secret key . We also let $L' = \{(x, c) \mid x \in L \wedge (x, D_{\mathsf{sk}}(c)) \in R$ with witness relation $R' = \{((x, c), (w, r)) \mid (x, w) \in R \wedge E_{\mathsf{pk},r}(w) = c\}$. Let (P', V') be a unbounded simulation sound proof system for L', and let S' be its simulator. .

Definition 20 (Prover $P(x, w)$).

$(c, r) \leftarrow E_{\mathsf{pk}}(w)$
$\pi' \leftarrow P'((x, c), (w, r), \xi_1)$
$\pi \leftarrow (c, \pi')$
return π

Definition 21 (Verifier $P(x, \pi)$).

Parse π *as* (c, π')
return $V'((x, c), \pi', \xi_1)$

Definition 22 (Simulator $S(\mathsf{tag}, \mathsf{params})$).

if $\mathsf{tag} = \mathsf{setup}$ **then**
  Parse params *as* 1^κ
  $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Kg}(1^\kappa)$
  $\xi_0 \leftarrow \mathsf{pk}$
  $(\xi_1, \mathsf{simstate}') \leftarrow S'(\mathsf{setup}, 1^\kappa)$
  $\mathsf{simstate} \leftarrow (\mathsf{sk}, \mathsf{simstate})$
  $\xi \leftarrow (\xi_0, \xi_1)$
  return $(\xi, \mathsf{simstate})$
else if $\mathsf{tag} = \mathsf{simulate}$ **then**
  Parse params *as* $(x, \xi, \mathsf{simstate})$
  Parse $\mathsf{simstate}$ *as* $(\mathsf{sk}, \mathsf{simstate}')$
  $c \leftarrow E_{\mathsf{pk}}(0)$
  $\pi' \leftarrow S'(\mathsf{simulate}, (x, c), \xi_1, \mathsf{simstate}')$
  $\pi \leftarrow (c, \pi')$
  return π
else if $\mathsf{tag} = \mathsf{extract}$ **then**
  Parse params *as* $(\pi, x, \xi, \mathsf{simstate})$
  Parse $\mathsf{simstate}$ *as* $(\mathsf{sk}, \mathsf{simstate}')$
  Parse π *as* (c, π')
  $w \leftarrow D_{\mathsf{sk}}(c)$
  return w
else
  return $\perp$
end if

We now prove the following theorem.

Theorem 5. *The proof system (P, V) is an unbounded simulation-sound non-interactive zero-knowledge proof of knowledge for the language L with witness relation R.*

Proof. We prove, in order, the properties adaptive indistinguishability and extractable simulation-soundness.

Adaptive Indistinguishability Assume (P, V) is not adaptively indistinguishable. Let A be an adversary such that $\mathbf{Adv}^{\mathsf{ad-ind}}_{(P,V,S),A}(\kappa)$ is non-negligible. We will use A to construct $A_{\mathsf{ad-ind}}$, breaking either the adaptive indistinguishability of (P', V'), or $A_{\mathsf{sem-sec}}$, breaking the semantic security of $\mathcal{CS}$. Consider the proof system $(\tilde{P}, \tilde{V})$, which is identical to (P, V) except that instead of outputting (c, π'), the prover outputs $(E_p k(0), \pi')$. If A wins the indistinguishability experiment in this case with non-negligible probability, then $A_{\mathsf{ad-ind}}$ can use A by prepending $E_{\mathsf{pk}}(0)$ to each query and distinguish between interaction with P' and the simulator.

If A does not win with non-negligible probability against $(\tilde{P}, \tilde{V})$, then it can be used by $A_{\mathsf{sem-sec}}$ in the following way. $A_{\mathsf{sem-sec}}$ chooses a theorem and a witness $(x, w) \in R$ and requests and encryption c of either w or 0. It runs $\mathbf{Exp}^{\mathsf{ad-ind}-0}_{(P,V,S),A}(\kappa)$ and $\mathbf{Exp}^{\mathsf{ad-ind}-1}_{(P,V,S),A}(\kappa)$ which the modification that P returns (c, π') instead of $(E_{\mathsf{pk}}(w), \pi')$. If A is able to distinguish between the experiments, then $A_{\mathsf{sem-sec}}$ concludes that it received an encryption of w, and otherwise that in received an encryption of 0.

Since, by assumption, (P', V') has adaptive indistinguishability and $\mathcal{CS}$ is semantically secure, the existence of either $A_{\mathsf{ad-ind}}$ or $A_{\mathsf{sem-sec}}$ with the above properties is a contradiction. Hence (P, V) has adaptive indistinguishability.

Extractable Simulation-Soundness Assume (P, V) does not have extractable simulation-soundness, and let A be an adversary which wins in Experiment 13 with non-negligible probability. We describe how to construct an adversary $A_{\mathsf{sim-sound}}$ which breaks the simulation soundness of (P', V').

$A_{\mathsf{sim-sound}}$ runs A in Experiment 13, while taking part in Experiment 12 itself. It answers queries to S by algorithm in Definition 22, using its oracle S' where necessary. When A outputs (x, π) on the call $A(\mathsf{guess}, \xi)$, $A_{\mathsf{sim-sound}}$ parses π as (c, π') and outputs $((x, c), \pi')$ on its call $A_{\mathsf{sim-sound}}(\mathsf{guess}, \xi)$. If $w \leftarrow D_{\mathsf{sk}}(c)$ is not a witness of x, then $(x, c) \notin L'$. Thus $A_{\mathsf{sim-sound}}$ wins in its experiment exactly when A wins. Thus $A_{\mathsf{sim-sound}}$ breaks the simulation-soundness of (P', V'), which is a contradiction. We conclude that (P, V) is extractable simulation sound.

In this paper we write NIZK-PK for unbounded simulation sound non-interactive zero-knowledge proof of knowledge.