Automatic Teller Machines for Offline E-cash

Anrin Chakraborti[1][0009−0006−2095−6992], Qingzhao Zhang[2][0000−0003−2598−5988], Jingjia Peng[3][0009−0007−3570−2212], Morley Mao[3][0000−0002−9844−2055], and Michael K. Reiter[4][0000−0001−7007−8274]

[1] University of Illinois Chicago, Chicago IL 60607, USA
[2] University of Arizona, Tucson AZ 85721, USA
[3] University of Michigan, Ann Arbor MI 48109, USA
[4] Duke University, Durham NC 27708, USA

Abstract. Electronic cash (e-cash) is a digital alternative to physical currency that allows anonymous transactions between users and merchants. Typically, coins in an e-cash scheme are only dispensed through a central bank. A drawback of this approach is that the bank is always on the critical path during withdrawals, and if a reliable connection to the bank is temporarily unavailable, users may be unable to withdraw coins in a timely fashion. As with physical currency, there are benefits to supporting a decentralized infrastructure where withdrawals can be performed without involving the bank on the critical path.

We propose the design of a new cryptographic bearer token that can be dispensed by *automatic teller machines* (ATM) in a fully offline e-cash scheme. Such bearer tokens provide anonymity, unforgeability and untraceability, i.e., users cannot be tracked by their spending activities or the locations of withdrawal. We formalize the requirements of an e-cash scheme with multiple issuers and propose an efficient design building on top of the compact e-cash protocol of Camenisch et al. [10]. Our construction leverages an unforgeable and *doubly-anonymous* voucher that allows a one-time transfer of coins between an ATM and a user, while hiding their identities from parties not involved in the transaction.

1 Introduction

Electronic cash (e-cash) [18] entitles a *user* holding a *bearer token* (coin) to certain privileges, which she exercises by presenting the token to a *gatekeeper* (merchant). As such, to be useful, a party should be unable to forge bearer tokens for herself; instead, a user must get a bearer token from a sanctioned *issuer* (bank). Furthermore, coins should also ensure *anonymity* during transactions, that is, they should not divulge any information regarding the user spending a coin, either to the bank or to the party that receives it in a transaction. Typically, there is only one issuer, namely the bank, and most e-cash schemes cannot be trivially extended (unless the bank shares secrets) to support multiple issuers.

Nonetheless, even in the context of e-cash, there are benefits to supporting multiple issuers and using them in a role similar to *automatic teller machines* (ATMs) for physical currency. For example, to withdraw coins in a single-issuer

system, a user must always connect to the bank, which makes the process vulnerable to network disruptions. As a viable alternative, we can envision a setup where ATMs act as physical proxies of the bank and facilitate fully *offline* operations. The user-to-ATM connections are supported through fast short-range communication channels, e.g., bluetooth, while the ATM-to-bank communication is over a standard wireless/wired network.

Introducing multiple issuers, which we will refer to as ATMs henceforth, pose new technical challenges for e-cash designs. First, if ATMs are physical entities, then a coin bearing the ATM's identity, e.g., in the form of a signature, will violate the user's anonymity guarantees. Therefore, coins should be *untraceable* to their issuers, as in case of real physical currency. Second, some of the ATMs can behave maliciously and the necessary safeguards need to be incorporated in the design; indeed instances of ATM hijacking are known in reality [1]. Third, the bank must be able to enforce rate-limiting policies on the ATMs and identify malicious behavior. Finally, it is desirable to support offline withdrawals if connectivity to the bank is temporarily disrupted.

Thus, this papers asks the following question:

> *Can we design e-cash schemes with multiple (potentially malicious) issuers under the control of an honest-but-curious bank that guarantees unforgeability, untraceability and anonymity?*

To answer this question affirmatively, we formalize the requirements of an e-cash scheme with multiple issuers and propose an efficient design. In our model, ATMs are stocked with coins *independent of user requests*, and subsequently a coin(s) is transferred between an authorized ATM and a verified user during a withdrawal from the ATM. To support such (limited) coin transfers, we introduce the notion of an unforgeable *voucher* that links a coin to the identities of the issuing ATM and the user who receives the coin. The voucher is included as part of the coin transaction and allows the bank to identify misbehaving users and ATMs. Crucially, the vouchers are *doubly anonymous* in the sense that they do not reveal the user's and issuing ATM's identities to the bank unless the coin has been double-spent (or double-issued). This general framework can be instantiated with several existing e-cash schemes by integrating vouchers into their withdrawal and spending mechanisms. We provide an instantiation building on top of the compact e-cash protocol of Camenisch et al. [10].

1.1 Problem Definition

Functionality: An e-cash scheme with multiple issuers (ATMs) must support three types of operations: i) ATMs can *withdraw* coins from the bank, ii) ATMs can *issue* previously withdrawn coins to users on demand, and iii) users can *spend* a coin(s) acquired previously at a merchant (Fig. 1). Merchants deposit coins obtained in transactions to the bank for account credits. In our model, only the bank is authorized to mint coins; the ATMs are intermediaries who facilitate fast withdrawals. However, ATMs cannot accept coin deposits because

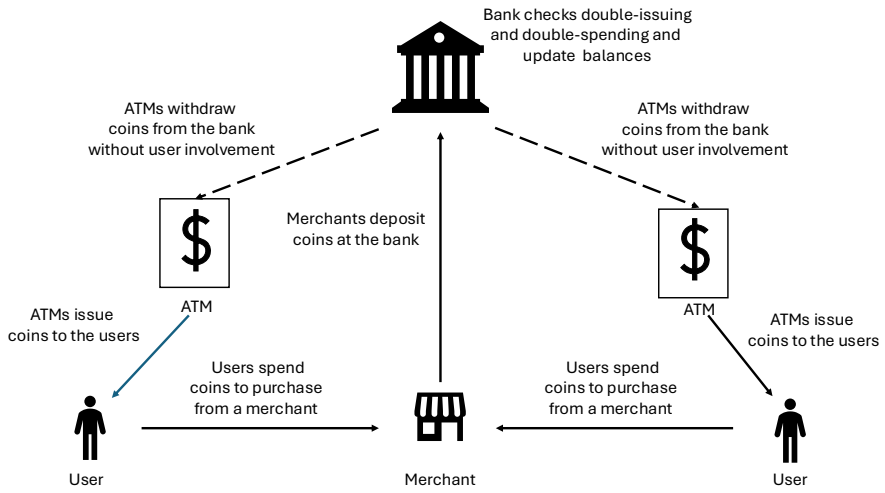

Fig. 1: A description of the parties and operations in a multi-issuer e-cash scheme. The ATMs requests coins from the bank (the dashed line indicates that this process is performed in the background and periodically) which are then issued to users. Coins are spent at merchants using transactions. Once the merchant verifies a transaction locally, it can deposit it at the bank.

they do not maintain account balances and/or have the ability to check for double-spending. Thus, deposits must always be to the bank.

For security, informally speaking, we assume that some of the ATMs can behave maliciously and potentially *double-issue* coins to honest users, and/or attempt to debit an honest user's account without issuing a commensurate number of coins. Similarly, malicious users can attempt to *double-spend* coins. Thus, all operations between users and ATMs need verifiability; e.g., the merchant can verify that the coin spent in a transaction was minted by the bank and issued to the user by an authorized ATM. The bank is honest-but-curious in our model and detects both double-spending by users and double-issuing by ATMs.

Existing definitions: transferable e-cash: Before proceeding further to formally define a multi-issuer e-cash system, we remark that all of the functions we require are already provided by transferable e-cash schemes. Introduced first by Okamoto and Ohta [30], transferable e-cash allows an arbitrary number of anonymous coin transfers between users and merchants. Indeed any transferable e-cash scheme will support ATMs almost trivially, e.g., ATMs can blindly withdraw coins from the bank, and then transfer them to the users.

However, transferable e-cash is a strictly stronger primitive than multi-issuer e-cash. Specifically, as noted by Canard et al. [14], any scheme supporting arbitrary coin transfers between users must satisfy a stronger notion of anonymity called *coin transparency*: once a user transfers a coin to another user, she can no longer recognize this coin even if she receives it in a subsequent transfer from another user. This is necessary because without this property, a merchant can

trace users by "'marking" a coin obtained in a transaction and then transferring the marked coin to another user. Subsequently, if the same coin is used again in a transaction, it will allow the merchant to easily correlate the two transactions, violating anonymous-spending guarantees.

Unfortunately, coin transparency comes with steep performance penalties: each transfer essentially randomizes the coin and the entire transcript pertaining to any previous transfer(s), thus ensuring that even if a coin reaches a party that has previously seen the same coin, he cannot recognize it. The transcript size is typically linear in the number of transfers, and it proves authenticity of the transfers while also allowing double-spending detection by the bank. To support this randomization, transferable schemes either use randomizable non-interactive zero-knowledge proofs [5] to generate the transcript, and/or malleable signatures [3]. Both of these tools are computationally expensive, making the state-of-the-art transferable e-cash scheme [5] concretely less efficient than any standard e-cash scheme, which can be typically constructed from cheaper primitives. Another major drawback of supporting arbitrary transfers is an inherently large coin size. For example, in the scheme of Bauer et al. [5], a single coin comprises of more than 272 pairing group elements, and each transfer adds 104 new elements to the coin (see [5, Table 1]). When the pairing group is instantiated with the BLS12-381 curve [4], a coin is roughly 19KB in size.

In contrast to transferable e-cash, *coin transparency is not necessary* for multi-issuer e-cash since, unlike in transferable schemes, after a user spends a coin, the merchant can only deposit the coin to the bank. This alleviates fears that the merchant will "mark" the coin and correlate any subsequent transactions. As such, we do not need to randomize the coin (and any related transcript), and more efficient constructions matching the performance of standard e-cash designs are possible. The size of a coin in our scheme is 1KB, more than 18× smaller than the coin size in the scheme of Bauer [5], and *all* operations are highly efficient, taking less than 100ms even on resource-constrained setups. While the performance of Bauer, et al.'s scheme [5] is not reported, we estimate higher computational costs, since it relies on the Groth-Sahai proof framework [22] for the randomizable non-interactive zero-knowledge proof system. We have no such requirement and only rely on standard Sigma proofs.

New requirements: In addition to the model relaxation, a multi-issuer scheme also enforces new requirements that are not directly addressed by a transferable e-cash scheme. We detail these requirements below.

(1) **Fair exchange:** Since some of the ATMs and users are malicious, we need to ensure that a *fair exchange* of coins takes place before an account is debited at the bank. That is, we must ensure that an ATM cannot falsely effectuate an update to a user's balance at the bank, unless it has issued a commensurate number of coins to the user, and conversely, a user cannot falsely accuse an ATM of reneging on a promise to issue coins, despite receiving the coin from the ATM. It is known that fair exchange with malicious parties is impossible without a trusted third party [31, 35]. Thus, a multi-issuer scheme must in-

corporate new mechanisms for such fair exchange by involving the bank as a TTP in case of disputes, without violating anonymity.

(2) **Compactness:** For storage efficiency, we want to ensure that coin withdrawals by the ATM can be performed offline. This will make the issuing process more efficient, as the bank needs to be consulted only for checking the user's balance. However, to support this feature, the ATMs would need to store a sufficient stock of coins to serve user requests as they arrive. Thus, optimizing coin storage becomes critical if the ATMs run on resource-constrained devices. One way to optimize storage is by making the scheme *compact* [10], i.e., the cost of storing n coins scales sub-linearly in n. This feature is missing from the state-of-the-art transferable e-cash scheme.

New definitions: In light of the discussion above, we define the security of a multi-issuer e-cash scheme by extending the formal definitions of unforgeability and anonymity of coins used in standard e-cash schemes to tolerate the presence of multiple issuers using a game-based model [7]. Since some of the ATMs and users can be malicious, our security definitions account for both double issuing and double spending. If the bank detects either double spending or double issuing, then there is a mechanism to identify the offending party and enforce penalties. In addition, we define two new properties:

- **Untraceability:** When a user obtains a coin from one of the ATMs, the corresponding transcript of issue does not reveal the identity of the issuer to either the merchant where the coin is spent, or to the bank. A single-issuer e-cash scheme does not need such untraceability guarantees because it is implicit that *all* coins must have been directly withdrawn from the bank.
- **Fair exchange:** Informally, this property requires that if either a user or an ATM reneges on its promise during a coin issue — as we will discuss later, a user provides a receipt upon receiving a coin from an ATM — the coin cannot be later spent without violating anonymity of the guilty party.

1.2 Construction Overview

Towards a solution for a multi-issuer e-cash scheme, we can start with a standard e-cash scheme such that the ATMs withdraw coins from the bank and store them locally. When a user requests a coin from an ATM, the ATM invokes the spending procedure of the e-cash scheme, with the user acting as the merchant, to issue a coin(s). Then, the user may spend the coin at a merchant and provide the transcript pertaining to the coin issue, i.e., the proof of ATM-to-user transaction, to demonstrate that it has acquired the coin from a valid ATM. The coins will be unforgeable and preserve anonymity during spending. It is worth noting that ensuring the validity of the coin is not enough because the merchant must also verify that the user presenting the coin is the owner of the coin. Otherwise, a malicious ATM may be able to *double-issue* coins.

The problem with this construction is that if issuing a coin simply follows the procedure of a standard transaction, and the users provide the corresponding transcript to the merchant/bank for verification, then the user does not remain

anonymous. Indeed, in e-cash, the identity of a merchant (the user in case of a ATM-to-user transaction) reporting a transaction is part of the transcript *in the clear*, so as to enable the bank to credit the merchant's account.

Doubly anonymous vouchers: We address this problem by introducing an unforgeable and doubly anonymous *voucher* that records a transaction of a valid coin between a spending party (an ATM) and receiving party (a user) while hiding both of their identities from any party that is not involved in the transaction, including the bank. More specifically, the voucher + coin allows any party to verify that the coin was minted by the bank, and was issued to the user claiming ownership, *without revealing the identities of either the user or the ATM that issued the coin.* In an e-cash scheme, such a voucher serves no purpose since at least the identity of the receiving party must be known to credit an account. However, a doubly anonymous voucher will allow for a *single* transfer of a coin between an ATM and a user, while also enabling the user to later spend the coin at a merchant. We remark that such a voucher can be computed *only* by the party who withdrew the coin — in our construction, only an ATM can compute a voucher — and so vouchers cannot be used for arbitrary number of transfers, as in a transferable e-cash scheme. Therefore, a doubly-anonymous voucher is a strictly weaker primitive than a randomizable *tag* proposed in previous work on transferable e-cash [3].

Instantiation with compact e-cash: Introducing a doubly anonymous and unforgeable voucher provides a blueprint to turn any single-issuer e-cash scheme into an e-cash scheme with multiple issuers. We demonstrate feasibility of this idea using the compact e-cash scheme of Camenisch et al. [10]. Informally, their protocol can be described *without compactness* as follows.

- **Withdrawal:** A coin is a key k for a pseudorandom function PRF uniformly-sampled by the user, that is blindly signed by the bank using a CL signature [11].
- **Spending:**
  (1) The user spends the coin by computing a *verification token* $X \leftarrow \mathsf{PRF}_k(R)$, where R is a unique coin-specific identifier (defined later).
  (2) Double spending is detected using a *double-spending token* of the form $Z \leftarrow pk_i \cdot \mathsf{PRF}_k(R)^{r_t}$ where $r_t \leftarrow h(pk_m, \dots)$ is derived from the hash of the merchant's public key pk_m and other transaction-specific information. Here, pk_i is the spending user's public key.
  (3) The user proves in zero-knowledge that it knows a signature on k under the bank's private key. The user also proves that the verification token and the double-spending token have been correctly computed. Thus, each transaction reported to the bank includes X, Z, r_t and the zero-knowledge proofs. The coin is not revealed in the clear as part of the transcript.
- **Deposit & double-spending:** If there are two transactions spending the same coin, which means that both transactions have matching values of the verification token X but different double-spending tokens Z and Z', then the

bank obtains: i) $Z \leftarrow pk_i \cdot \mathsf{PRF}_k(R)^{r_t}$, ii) $Z' \leftarrow pk_i \cdot \mathsf{PRF}_k(R)^{r'_t}$, iii) r_t and iv) r'_t. Using these values the bank derives the misbehaving user's identity

$$\mathsf{PRF}_k(R) = \left(\frac{Z}{Z'}\right)^{r'_t - r_t} \; ; \; pk_i = \frac{Z}{\mathsf{PRF}_k(R)^{r_t}}$$

Design overview: We adapt this design to a multi-issuer system. At a high-level, our scheme can be viewed as a two-layered e-cash scheme where each spent coin is part of two transactions: the first transaction is when the coin is issued to a user by an ATM, and the second transaction is when it is spent by the user at a merchant. The caveat is that both these transaction must be cryptographically linked to each other; i.e., the receiver in the first transaction is the spender in the second transaction. Since we cannot reveal the receiver's (and spender's) identity, we use the unforgeable and doubly anonymous voucher to tie the two transactions together. In the following, we informally the describe this idea.

We begin by describing the keys in the possession of the users and the ATMs. Each user has: i) a *identity key* pair $\langle pk_i, sk_i \rangle$, ii) a *spending* key that is a fixed, uniformly sampled PRF key s_i, and iii) a key pair for a digital signature scheme. The private identity key and the PRF key are blindly signed by the bank using a CL-signature. Similarly, each ATM has: i) a identity key pair $\langle pk_j^A, sk_j^A \rangle$, and ii) a key pair for a digital signature scheme. The private identity key is signed blindly by the bank using a CL-signature. ATMs withdraw coins from the bank following Step 1 of the compact e-cash scheme described above; i.e., each coin is a PRF key k that is uniformly sampled by the ATM and blindly signed by the bank. When an ATM issues the coin corresponding to k to a user, the ATM additionally creates a voucher and provides it to the user as proof of ownership.

(1) The voucher includes a unique identifier $X \leftarrow \mathsf{PRF}_k(P)$ where P is a commitment to the user's private identity key sk_i. The voucher also includes P.
(2) The voucher includes a *double-issuing token* $Y \leftarrow pk_j^A \cdot \mathsf{PRF}_k(\dots)^P$ that embeds the identity of the ATM. Here, pk_j^A is the issuing ATM's public key. (The inputs to the PRF computation is a constant and will be defined later.)
(3) The voucher includes non-interactive zero-knowledge proofs to show that X and Y have been correctly computed under a PRF key signed by the bank.

The voucher is doubly anonymous because it does not include the issuing ATM's or user's identity in the clear. Unforgeability is ensured because the issuing party must prove knowledge of the bank's signature on the key k. Similarly, the voucher is bound to the user because of the computationally binding commitment on the user's private key that ties all the components of the voucher together. Moreover, if an ATM issues the same coin twice, then the bank will be able to recover pk_j^A from the double-issuing token, assuming that with overwhelming probability, the commitments in two separate withdrawals will not be the same. As we will see, in order to spend the coin, the user must prove that it knows the opening of the commitment.

1.3 Design Optimizations

Fair exchange: We ensure an *optimistic* fair exchange of coins between users and ATMs by using the bank as a trusted third party [2, 9, 28]. Optimistic, as defined by previous work [9], in this context means that the bank is contacted only when there is a dispute during an exchange

The idea is that after a user is issued a coin, it provides a signed receipt to the ATM that indicates that it has withdrawn a coin(s) from that ATM; importantly, the receipt does not include any information that identifies the coins. The receipt is subsequently provided by the ATM to the bank to debit the user's account. However, during this exchange, it is possible that one of the parties reneges on its promise: the user may not provide a receipt after receiving a coin, or conversely, the ATM may not provide the coin to the user after receiving the receipt. Either way, this is problematic: if the user does not provide a receipt, then it essentially gets a coin for "free" because the ATM cannot prove to the bank that the user has withdrawn the coin. And, if the ATM misbehaves, then it may issue the coin to another (malicious) user, but debit the honest user's account instead.

To mitigate this problem, we facilitate an optimistic fair exchange as follows.

(1) The ATM first provides a signed (under its private key) *promise* to the user, which includes: i) a computationally hiding and binding commitment to the coin, ii) the voucher linked to the coin in the clear, and iii) a randomly chosen nonce that uniquely identifies the withdrawal.
(2) After verifying the signature, the user signs a receipt (under its private key) which includes: i) the public key of the user, ii) the public key of the ATM, and iii) the nonce, and provides it to the ATM.
(3) After verifying the receipt, the ATM opens the commitment to the coin. The user verifies the opening and ensures that the coin is consistent with all the information signed by the ATM in Step 1.
(4) If the user does not receive the coin from the ATM after sending the receipt, or the coin is inconsistent with all the other information, the user informs the bank about the ATM's misbehavior using an AbortWithdrawal message. In that message, the user includes the coin commitment, the voucher, the nonce and the ATM's signature on all the information obtained in Step 1.
(5) If the bank receives an AbortWithdrawal message from a user, it stores all the corresponding information in a log for future investigation. Moreover, the bank does *not* debit the user's account, *even if a contradictory receipt is received from the ATM signifying that the user's balance should be debited.*

This mechanism ensures fair exchange as follows. Suppose first that a misbehaving user does not provide the receipt after receiving the first message from the ATM. In that case, the user does not get the coin, unless it is able to derive the coin from the commitment, which happens with negligible probability. Of course, a user may try to misuse the mechanism by sending an AbortWithdrawal message even after it receives the coin. But this is not a problem, because since the message contains the signed commitment to the coin, the bank can later identify if the coin is spent by the same user, and catch the user's lie.

Conversely, suppose that an ATM misbehaves by not opening the commitment after receiving the receipt, and the user sends the AbortWithdrawal message along with the voucher and the coin commitment. The idea is that since the voucher ties the coin to the user, and includes the double-issuing token, if the ATM tries to double-issue the coin to a different user later, then upon receiving the coin after it is spent, the bank will be able to identify the ATM for double-issuing. Thus, a (committed) coin that is reported as part of AbortWithdrawal effectively becomes void because it cannot be spent by a misbehaving user or issued by a misbehaving ATM. We remark that since both the user's and ATM's identities are available to the bank, this mechanism cannot be misused for a denial of service since : i) the bank will identify if too many reports are submitted by the same user, and ii) the ATM will exhaust all its coins as a reported coin becomes unusable, and the bank can deny further withdrawals by the ATM.

Spending & deposits: A user spends a coin by providing both the coin and the voucher to the merchant. Recall that the commitment to the user's private key P binds the coin to the voucher, and the merchant verifies that the coin was indeed issued to the user using the commitment. Specifically, to spend a coin,

(1) The user with public key pk_i computes the double-spending token $Z \leftarrow pk_i \cdot \mathsf{PRF}_{s_i}(P)^{r_t}$ using its signed spending key s_i (where r_t is derived from the merchant's public key pk_m and other information specific to the transaction).
(2) The user proves in zero-knowledge the bank's signature on the opening of P, and that the public key pk_i in the double-spending token is linked to the private key in the opening of the commitment. The user also proves in zero-knowledge the bank's signature on s_i. The transaction with the merchant includes the coin, the voucher, Z, and the zero-knowledge proofs.

As we will describe later, the PRF and the commitment scheme can be instantiated such that all the zero-knowledge proofs can be implemented with standard Sigma protocols. Once the merchant verifies the coin and the voucher, it deposits all the information to the bank. The bank checks for double spending and double issuing, following a procedure similar to the compact e-cash protocol. The bank also check if there is a matching commitment to the coin in any of the AbortWithdrawal messages it has received in the past. If so, the bank identifies the misbehaving user/ATM.

Compactness: The scheme can be made compact by leveraging a technique due to Camenisch et al. [10] whereby the same PRF key k is used multiple times to issue new coins. Each coin has a unique serial number $\mathsf{ctr} \in [1, n]$, and both the verification token $X \leftarrow \mathsf{PRF}_k(\mathsf{ctr})$ and double-issuing token $Y \leftarrow pk_j^A \cdot \mathsf{PRF}_k(\mathsf{ctr})^P$ are tied together using ctr. The issuing ATM computes a non-interactive zero-knowledge range proof showing that the same value of $\mathsf{ctr} \in [1, n]$ is used in both evaluations. We describe the scheme in detail in Sec. 4.

Fully offline withdrawals: Our scheme supports fully offline withdrawals such that an ATM does not need to check a user's balance before issuing coins. This is particularly helpful if the connection between the ATMs and the bank is prone to disruptions, or the round-trip latency is high. To support this feature, we

share with the ATMs a set of accounts that do not have enough balance to support the maximum amount that can be withdrawn from an ATM; typically this amount is much smaller than an average user's balance. The set is shared in a concise fashion by periodically broadcasting to the ATMs a Bloom filter with these account numbers. When a user requests a withdrawal, the ATM checks from the Bloom filter whether the user has sufficient balance. We note that while the Bloom filter reveals the accounts with low balance to the ATMs, in practice this information (or parts of it) may be available to the ATMs anyway during withdrawals. Sharing the Bloom filter does not violate any of the security guarantees of an e-cash scheme (anonymity during spending and unforgeability). The ATM updates the bank about the withdrawal using the users' receipts as proof, either periodically or under stable network connection.

A potential risk of fully offline withdrawals is that the user may attempt to overdraw her account by visiting multiple ATMs in between the times that the user withdraws from an ATM and her account is debited by the bank. However, in this case, the user will be eventually identified by the signed receipts, and the bank can enforce appropriate penalties. Nonetheless, we also discuss mitigation strategies to reduce the risk of such behavior in Sec. 5.

Collusion & enforcing penalties: As with most *offline* e-cash schemes, our scheme also detects user/ATM misbehavior *after the fact*, and thus we expect that the bank will enforce appropriate penalties. We do not stipulate how such penalties are decided, but we note that in real-world deployments, economic penalties are typical, e.g., overdraft charges when a user overdraws her account. Another point of concern is that if an ATM misbehaves, it may collude with a malicious user and issue a large number of coins. Since every coin that is issued by an ATM must be accounted for in a corresponding receipt, the bank will detect if there is a discrepancy in the amounts. One obvious precaution that the bank can implement is ensuring that ATMs can only stock a small amount of coins at a time. However, beyond these measures, purely cryptographic solutions are not enough to handle collusions, because if the ATM ceases all communication with the bank after issuing an arbitrary number of coins to a malicious user and deletes all local state, then the bank cannot learn any new information to identify the malicious users. Therefore, non-cryptographic safeguards are necessary. For example, it is common practice to have cameras inside ATMs to track user misbehavior; we envision similar measures if the ATMs are implemented by physical entities, e.g., edge devices, in our scheme. Of course, physical attacks against such mechanisms are possible but we consider them out of scope.

1.4 Related Work

Electronic cash (e-cash) is a digital alternative to physical currency. In typical schemes, users *withdraw* digitally signed coins from a bank and *spend* them at merchants, who verify the signature and report valid, unspent coins. Anonymity is preserved by preventing the bank from linking withdrawals to spending. E-cash is *online* when the bank participates in transactions [17], and *offline* when

it only detects (but cannot prevent) double spending [18]. Numerous variants address specific needs [3, 8, 10, 13, 15, 34, 36]. We refer readers to broader surveys for more details, and focus on schemes with multiple issuers (ATMs).

One class of scheme is delegated e-cash, which uses proxy blind signatures [25, 26, 37, 39] to let a central bank authorize branches to issue coins. However, these schemes assume full trust in branches and lack oversight mechanisms such as *rate-limiting* to prevent over-issuance. Another option is transferable e-cash [3, 5, 14], where coins can be passed between users arbitrarily. While meeting our requirements, such schemes are inefficient in practice and support more general transferability than needed here, as discussed before.

2 Model

2.1 Threat Model

Our e-cash system has three types of participants.

Bank: The bank issues currency and maintains user accounts. The bank also certifies users that are registered to the system, e.g., by signing their public keys. The bank is *honest but curious*.

ATM: Similar to the real-world analogue of physical currency, ATMs serve as cash dispensing points. Users can withdraw coins at the ATMs, *but deposits must be directly at the bank*. This is because an account credit can occur only after a coin(s) has been checked for double-spending, which cannot be performed by an ATM. Some ATMs are malicious, though we assume that adversarial behaviors such as denial-of-service (DoS) attacks, where the ATM refuses to serve a user's withdrawal request, are considered beyond the scope of this study.

Users: Users withdraw and spend coins either by interacting directly with the bank or via ATMs. A subset of the users are also merchants who provide services in exchange for coins. Users can behave maliciously, i.e., they may violate the protocol to overdraw their balance or double-spend coins.

2.2 Syntax

We define the syntax of our e-cash scheme (see Fig. 2), starting from the items that are exchanged in the scheme.

– **Coin:** A coin is the currency exchanged in the scheme. A coin can be only issued by a bank, directly to a user or to an ATM. Each such coin is unique (with overwhelming probability). To spend a coin, it needs to be first included in a voucher (described next), which ties the coin to a user identity (necessary for double spending detection). When a user directly withdraws a coin from a bank, the voucher is created by the bank. Otherwise, it is created by the ATM from where the user withdraws the coin.

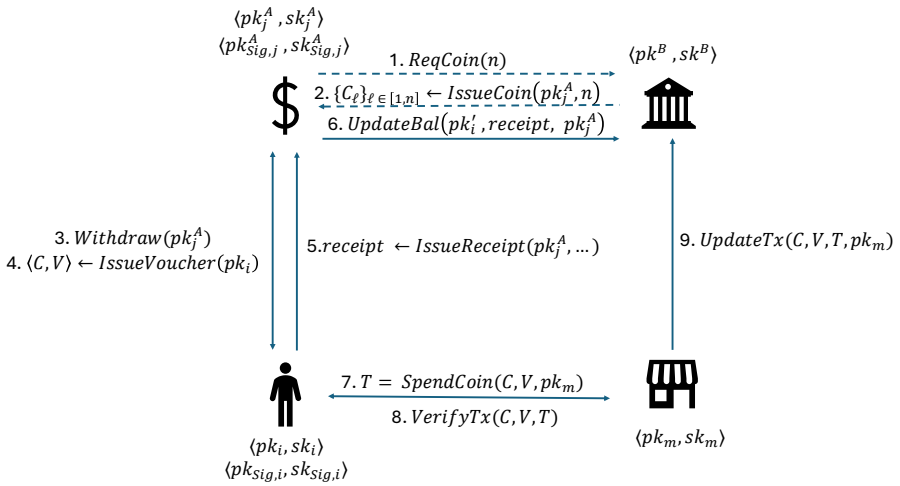

Fig. 2: Formal interfaces in a multi-issuer e-cash scheme. Each withdrawal from an ATM produces a coin and a linked voucher. After withdrawal, the user provides a signed receipt to the ATM as proof of withdrawal that is used to debit the user's account. The merchant verifies the coin and voucher in a transaction before depositing to the bank.

- **Voucher**: A voucher ties a specific coin to a user's identity, e.g., public key, when the user withdraws that coin. The rationale behind having a voucher (in addition to coins) is that unlike traditional online e-cash, where the user must directly interact with the bank to withdraw a coin, in our scheme the user can be issued coins that have been acquired *a priori* by an ATM without the user's identity being known. Consequently, mechanisms that require the user's identity, e.g., double-spending detection, cannot be directly applied to the coin. A voucher enables these additional mechanisms. Vouchers are not transferable between users. We also assume for simplicity that a voucher is tied to a single coin, although this can be relaxed with additional mechanisms.
- **Transaction:** A transaction records the spending of a coin. It uniquely identifies the coin, and the merchant that received the coin for a service. Each transaction has a unique, random transaction ID, and may also include other fields that are application-specific. Additionally, the identity of the user spending the coin is embedded in a transaction, *but is only available in the clear to the bank when the coin in the transaction is double spent in another transaction.* Once the merchant verifies the coin and the voucher, it deposits the transaction to the bank, which updates balances and records the coin in a log.
- **Receipt:** A receipt records an interaction between a user and an ATM. In particular, this is a message signed under the user's private key indicating the amount that has been withdrawn and other relevant information. Receipts act as a safeguard against misbehavior by malicious ATMs.

2.3 Algorithms

Bank: The bank implements the following algorithms (see Fig. 2).

- $\langle pk^{\mathcal{B}}_{\mathsf{sig}}, sk^{\mathcal{B}}_{\mathsf{sig}} \rangle \leftarrow \textsc{InitBank}(1^\lambda)$: When this interface is invoked, the bank generates the secrets necessary to issue coins, and authenticate users/ATMs. This includes a key pair for a blind signature scheme. It also initializes a key-value store IDS where the values are user accounts (or derived information) that will be used in the scheme to track double spending, as well as a list of already deposited coins CS, initially empty. The bank tracks account balances for the users, which is updated during withdrawals and deposits.
- $\{\mathsf{C}_\ell\}_{\ell \in [n]} / \perp \leftarrow \textsc{IssueCoin}(pk^A_j, n)$: This interface takes as input the identity of an ATM, e.g., in the form of a certificate with a public key, and the number of coins requested n (and other auxiliary information), and either issues the specified number of coins, or aborts. If the request is directly from a user, the bank also issues a voucher tying the coin to the user's identity.
- $0/1 \leftarrow \textsc{UpdateBal}(pk'_i, \mathsf{receipt}, pk^A_j)$: This interface takes as input the a receipt signed by the user who has requested a withdrawal from ATM_j. This invocation is made by ATM_j to update the balance of the indicated user after a withdrawal. The interface returns a 0 to update the balance was updated correctly, or returns a 1 if the invocation fails, e.g., the account is overdrawn.
- $pk_i / pk^A_j / 0 \leftarrow \textsc{UpdateTx}(\mathsf{C}, \mathsf{V}, \mathsf{T}, pk_m)$: This interface takes as input the identity of the merchant where a transaction has taken place, and all the artifacts related to the transaction, which includes the coin C, the corresponding voucher, V, and the transaction record, T, linking the coin to the merchant. The algorithm checks whether the coin is valid; the voucher records the assignment of this coin to a user; and the transaction record pertains to the spending of this coin (by that user). If the coin is not valid, then the interface aborts. Otherwise, the invocation first checks whether the coin is already recorded in the log, in which case it triggers a double-spending detection algorithm and *outputs the identity (a public key) of the user or the ATM who double-spent/double-issued the coin.* If there is no double-spending, the invocation adds the coin to the log, updates the balance of the merchant and outputs 0 to signify that the transaction has been recorded.

ATM: Each ATM implements the following algorithms.

- $\langle pk^A_j, sk^A_j \rangle, \langle pk^A_{\mathsf{Sig},j}, sk^A_{\mathsf{Sig},j} \rangle \leftarrow \textsc{InitATM}(1^\lambda)$: This interface sets up the secrets that will be used by the ATM to dispense coins to users. In our scheme, this corresponds to an identity key pair, $\langle pk^A_j, sk^A_j \rangle$, where the private key is blindly signed by the bank, and a key pair for a digital signature $\langle pk^A_{\mathsf{Sig},j}, sk^A_{\mathsf{Sig},j} \rangle$.
- $\{\mathsf{C}_\ell\}_{\ell \in [n]} / \perp \leftarrow \textsc{ReqCoin}(n)$: This interface requests n coins from the bank, and may take other request-specific inputs. The bank may decline the request if, e.g., the bank would like to *rate-limit* the ATM.
- $\mathsf{C}, \mathsf{V} \leftarrow \textsc{IssueVoucher}(pk_i)$: This interface takes as input a user $\mathcal{U}_i$'s information, and outputs a voucher that is (unforgeably) linked to a coin; we indicate that a coin is linked to a voucher with the notation $\mathsf{C} \rightleftharpoons \mathsf{V}$. The voucher has a

serial number that is tied to the user's identity; we do not allow transferring of vouchers between users. The voucher also includes relevant information to verify authenticity.

Users Each user implements the following algorithms:

- $\langle pk_i, sk_i \rangle, \langle pk'_i, sk'_i \rangle \leftarrow \textsc{InitUser}(1^\lambda)$: This interface generates a key pair to establish $\langle pk_i, sk_i \rangle$ and a signing key pair $\langle pk'_i, sk'_i \rangle$ for a digital signature scheme. The private key sk_i is blindly signed by the bank. The interface may also generate other secrets which will be described later.
- $\mathsf{C}, \mathsf{V} \leftarrow \textsc{Withdraw}(pk_j^A)$: This interface takes as input an ATM's public identity key and other auxiliary information, and requests a coin(s) withdrawal from the ATM (or from the bank).
- $\mathsf{T} \leftarrow \textsc{SpendCoin}(\mathsf{C}, \mathsf{V}, pk_m)$: This interface takes as input a coin C, the linked voucher V, the identity of a merchant in the form of a public key pk_m, and other auxiliary information, and produces a transaction signifying that the coin has been spent at that merchant.
- $0/\perp \leftarrow \textsc{VerifyTx}(\mathsf{C}, \mathsf{V}, \mathsf{T})$: This interface takes as input a transaction that has taken place between a user and a merchant, and outputs 0 if the transaction is valid (defined later), otherwise it aborts. The interface is invoked by a merchant to verify the coin and the voucher, and other relevant information, before depositing it to the bank.
- $\mathsf{receipt} \leftarrow \textsc{IssueReceipt}(pk_j^A, \dots)$: This interface takes as input the identity of an ATM ATM_j from where a withdrawal has taken place, and other information, and produces a signed (using the user's private key) message indicating that the user withdrew a coin(s) from the ATM.

2.4 Security

We informally describe the security guarantees provided by our e-cash scheme. Formal definitions are available in the full version of the paper [16]

Unforgeability: Only the bank can mint new coins. We define this property with a security experiment, where the adversary represents the malicious users and ATMs, and is able to invoke the bank, the honest ATMs and the honest users as oracles. It eventually produces a coin C, a voucher V, possibly linked to the coin, and a transaction T that spends the coin. The adversary wins if the bank accepts the transaction, and the coin was not issued by the bank.

Anonymity: We will ensure that the coins spent at a merchant and the transactions submitted to the bank do not reveal any information about the spender, unless the coin has been double spent. We define this property with an indistinguishability experiment, where a distinguisher – signifying the bank, a merchant or an ATM – receives transactions deposited by two honest users and attempts to distinguish between them.

Authenticity of balance: We will ensure that an adversary, representing dishonest users and ATMs, cannot dishonestly withdraw from an honest user's

account. They also cannot forge unauthorized transactions to credit their own accounts using coins that have been withdrawn by honest users. (This also ensures that an honest user cannot be falsely implicated for double spending.)

Fair exchange: We will ensure that a user that receives a coin during a withdrawal must provide a receipt to the ATM; otherwise it is unable to spend the coin without being detected by the bank. We will ensure that the receipts are unforgeable, and can be used by ATMs and users if there are conflicts regarding account updates. Combined with authenticity of balance where a user does not provide a receipt until it receives a coin from the ATM, this property implies a fair exchange enabled through the bank acting as the trusted third party.

Identification of double spenders: We will ensure that two valid transactions double spending the same coin will identify the user responsible. Alternatively, if the same coin is issued to two different users by a malicious ATM, then the identity of the offending ATM will be disclosed to the bank.

Untraceability: We will introduce a new security property that is important for schemes such as ours that can dispense cash through intermediaries. If a voucher includes information about the ATM that issued it, e.g., in the form of a signature, then the voucher will divulge information regarding the whereabouts of the user to the merchant and the bank. e-cash systems typically do not need to protect this information since they only allow withdrawals directly from the bank. However, as we are assuming that ATMs can be physical entities, protecting the user's locations from untrusted merchants becomes critical.

We will ensure that information regarding the honest ATM where an honest user withdrew a coin is not available to the merchant where it deposits coins and to the bank. We formalize this using an indistinguishability experiment as an inability to distinguish a coin + voucher dispensed by one of two honest ATMs to an honest user of the adversary's choice

Non-goals: The system does not address denial-of-service attacks where a malicious ATM does not provide well-formed vouchers to honest users, or overloads the bank by flooding withdrawal/deposit requests.

2.5 Cryptographic Tools

Pseudorandom function: We will use a pseudorandom function (PRF) family. A uniformly sampled PRF produces outputs that are indistinguishable from a random function. In our constructions, we particularly use the function family $\mathsf{PRF} : \mathbb{Z}_p \times \mathbb{Z}_p \to \mathbb{G}$ over the group $\mathbb{G}$ with order p, due to Dodis and Yampolskiy [19]. Given a key $a \leftarrow\!\!\$\ \mathbb{Z}_p$ and an input $x \in \mathbb{Z}_p$, the PRF computes $\mathsf{PRF}_a(x) = g^{\frac{1}{1+a+x}}$, where g is a generator of $\mathbb{G}$.

Commitment scheme: We will use a cryptographic commitment scheme $\mathsf{Comm} : \mathbb{Z}_p^{n+1} \to \mathbb{G}$, where $\mathbb{G}$ is a group with order p. The algorithm $\mathsf{Comm}(m; r)$ is used to commit to a set of messages $m_1, \ldots, m_n \in \mathbb{Z}_p$ with a random coin $r \leftarrow\!\!\$\ \mathbb{Z}_p$. Wherever it is not necessary, we will drop the random coin in the notation for

brevity. We will require the scheme provides computational hiding and binding properties. Intuitively, the hiding property implies that a commitment to a message m hides the message against an adversary. The binding property ensures that an adversary cannot produce two messages that produce the same commitment. We will use generalized Pedersen commitments [32]. The public parameters include $\mathbb{G}$, and $n+1$ generators $g_1, \ldots, g_n, g'$. To commit to a set of values $m_1, \ldots, m_n \in \mathbb{Z}_p$, $\mathsf{Comm}(m_1, \ldots, m_n; r) = g_1^{m_1} \ldots g_n^{m_n} (g')^r$, where $r \leftarrow\!\!\$\, \mathbb{Z}_p$.

Digital signature: We will use a signature scheme $\pi_{\mathsf{sgn}} = \langle \mathsf{SKGen}, \mathsf{Sign}, \mathsf{Vrfy} \rangle$ where SKGen generates a public-private key pair $\langle pk, sk \rangle$. The signing algorithm Sign on input m produces a signature $\sigma \leftarrow \mathsf{Sign}_{sk}(m)$. The verification algorithm $\mathsf{Vrfy}_{pk}(m, \sigma)$ outputs 1 if the input is a valid signature under sk and outputs 0 otherwise. For correctness we require that $\mathsf{Vrfy}_{pk}(\mathsf{Sign}_{sk}(m), m) = 1$ for any $\langle pk, sk \rangle$ output from SKGen. For security, we require unforgeability, which implies that an adversary cannot produce a signature on a message of its own choice under sk, for which it has not queried a signing oracle before.

Blind signature: A blind signature scheme is a signature scheme in which the signer can create signatures on unknown messages [17]. The signature scheme is defined by a tuple of algorithms $\pi_{\mathsf{bsgn}} = \langle \mathsf{BKGen}, \mathsf{BBlind}, \mathsf{BSign}, \mathsf{BUnblind}, \mathsf{BVrfy} \rangle$. Here, $\mathsf{BKGen}(1^\lambda)$ generates a public-private key pair, $\mathsf{BBlind}(m; r)$ blinds the message m to be signed using a random blind r, $\mathsf{BSign}_{sk}(\mathsf{BBlind}(m; r))$ computes a signature σ on the blinded message, $\mathsf{BUnblind}(\sigma; r)$ unblinds the signature to obtain a signature on the original message, and $\mathsf{BVrfy}_{pk}(m, \sigma)$ verifies a signature. We will use a standard RSA blind signature [33].

Blind signature with proofs of knowledge: We will use a blind signature scheme where the holder can prove in zero-knowledge that it has a signature on a committed value. Specifically, such a signature scheme is defined by the tuple of algorithms $\pi_{\mathsf{zsgn}} = \langle \mathsf{ZkGen}, \mathsf{ZSign}, \mathsf{ZProve}, \mathsf{ZVrfy} \rangle$. The algorithm ZkGen creates a public-private key pair. The algorithm $\sigma \leftarrow \mathsf{ZSign}_{sk}(com)$ takes as input a commitment to a value, and produces a signature on the value that has been committed to. The algorithm $\Psi \leftarrow \mathsf{ZProve}_{pk}(com, \sigma)$ takes as input a commitment to a value and a signature on the committed value, and produces a NIZK proof of knowledge of a signature on the opening of the commitment. Finally, $\mathsf{ZVrfy}_{pk}(com, \Psi)$ takes as input a commitment and a NIZK proof produced by ZProve, and returns 1 if the proof shows that the party knows a signature on the committed value. Several schemes provide these properties, such as CL signatures [11] and BBS signatures [12].

Non-interactive zero-knowledge proofs of knowledge: Our protocol leverages a noninteractive zero-knowledge (NIZK) proof of knowledge $\Pi = \langle \mathsf{pGen}, \mathsf{pVerify}, \mathsf{pSim}, \mathsf{pEx} \rangle$ in the random oracle model [6]. Let $\mathcal{R}_{\mathcal{L}}$ be the witness relation for language $\mathcal{L} \subseteq \{0,1\}^*$, and let $\mathcal{H}$ denote the set of all functions from $\{0,1\}^*$ to $\{0,1\}^\lambda$. On input $\langle x, w \rangle \in \mathcal{R}_{\mathcal{L}}$ and with access to a random oracle $h \leftarrow\!\!\$\, \mathcal{H}$, $\mathsf{pGen}_w^h(x)$ produces a proof Ψ (using the witness w) that $x \in \mathcal{L}$, so that if $\Psi \leftarrow \mathsf{pGen}_w^h(x)$ then $\mathsf{pVerify}^h(x, \Psi)$ returns true. As is typical, we require

Table 1: Table of notations

Cryptographic schemes	
π_{sgn}	standard digital signature scheme
π_{bsgn}	blind signature scheme
π_{zsgn}	blind signature scheme with proofs of knowledge (e.g., CL-signature [11])
PRF	Dodis-Yampolinsky PRF scheme over group $\mathbb{G}$ with generator g
Comm	generalized Pedersen commitment over group $\mathbb{G}$
RO	Random oracle $\{0,1\}^* \to \{0,1\}^\lambda$
Keys	
$\langle pk^{\mathcal{B}}_{\mathsf{sig}}, sk^{\mathcal{B}}_{\mathsf{sig}} \rangle$	bank's key pair for a digital signature π_{sgn}
$\langle pk^{\mathcal{B}}_{\mathsf{b}}, sk^{\mathcal{B}}_{\mathsf{b}} \rangle$	bank's key pair for a blind signature π_{bsgn}
$\langle pk^{\mathcal{B}}_{\mathsf{z}}, sk^{\mathcal{B}}_{\mathsf{z}} \rangle$	bank's key pair for a blind signature with proofs of knowledge π_{zsgn}
$\langle g^{sk^A_j}, sk^A_j \rangle$	ATM_j's identity key pair; $sk^A_j \leftarrow\!\!\$\ \mathbb{Z}_p$
$\langle g^{sk_i}, sk_i \rangle$	user $\mathcal{U}_i$'s identity key pair; $sk_i \leftarrow\!\!\$\ \mathbb{Z}_p$
$\langle pk_{\mathsf{Sig},i}, sk_{\mathsf{Sig},i} \rangle$	user $\mathcal{U}_i$'s signing key pair for the signature scheme π_{sgn}
$\langle pk^A_{\mathsf{Sig},j}, sk^A_{\mathsf{Sig},j} \rangle$	ATM_j's signing key pair for the signature scheme π_{sgn}
σ_{sk_i}	bank's blind signature under $sk^{\mathcal{B}}_{\mathsf{z}}$ on $\langle sk_i, s_i \rangle$
$\sigma_{sk^A_j}$	bank's blind signature under $sk^{\mathcal{B}}_{\mathsf{z}}$ on sk^A_j
Components of a coin	
A	commitment to PRF key $a \leftarrow\!\!\$\ \mathbb{Z}_p$ sampled by ATM
B	commitment to PRF key $b \leftarrow\!\!\$\ \mathbb{Z}_p$ sampled by ATM
Q	commitment to ATM's private identity key sk^A_j
$\langle h(A,B,Q), \sigma_c \rangle$	coin format; σ_c is a signature on $h(A,B,Q)$ under $sk^{\mathcal{B}}_{\mathsf{b}}$
Components of vouchers and transactions (except NIZK proofs)	
P	commitment to user $\mathcal{U}_i$'s secrets $\langle sk_i, s_i \rangle$
X	unique unforgeable identifier tying a specific coin to a user's identity
Y	double-issuing token to identity ATMs who double-issue coins
Z	double-spending token to identify users who double spend coins

two properties from the proofs of knowledge: i) the existence of a knowledge extractor pEx, and ii) a proof simulator pSim.

3 Protocol Design

W describe two versions of our e-cash protocol. The first protocol π_{nc} sacrifices compactness for faster withdrawals and deposits. The protocol π_{comp} introduces compactness but at the cost of higher withdrawal and spending latencies. In the following, $\mathbb{G}$ is a prime-order group with order p, g is a random generator of $\mathbb{G}$, $\mathsf{PRF} : \mathbb{Z}_p \times \mathbb{Z}_p \to \mathbb{G}$ is the Dodis-Yampolinsky PRF, and Comm is the generalized Pedersen commitment scheme. For better readability, we summarize the key notations in Table 1. The proofs of security are available in the full version [16].

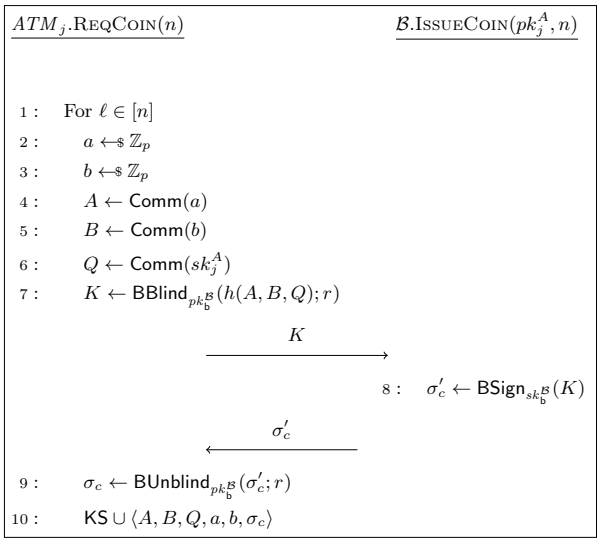

Fig. 3: π_{nc}: Coin requests by ATM

3.1 Description of Non-Compact Protocol

Initialization The bank initializes by selecting signing keys for three types of signature schemes: a key pair for π_{sgn}, a key pair for a blind signature scheme π_{bsgn} and a key pair for a blind signature scheme with proofs of knowledge π_{zsgn} (see Table 1). ATMs initialize by selecting a public-private key for establishing identity $\langle sk_j^A, g^{sk_j^A} \rangle$ and a signing key pair $\langle pk_{\mathsf{Sig},j}^A, sk_{\mathsf{Sig},j}^A \rangle$ for the signature scheme π_{sgn}. The bank signs the private identity key using π_{zsgn}. The bank stores $g^{sk_j^A}, pk_{\mathsf{Sig},j}^A$ and other information about the ATM.

Each user initializes by registering with the bank. For this, the user samples a random PRF key s_i, a key pair $\langle sk_i, g^{sk_i} \rangle$ as its identity keys, and a signing key pair $\langle pk_{\mathsf{Sig},i}, sk_{\mathsf{Sig},i} \rangle$ for the signature scheme π_{sgn}. The bank signs both s_i and sk_i as a pair using π_{zsgn}. During signing, the user proves in zero-knowledge that the public key g^{sk_i} is correctly derived from the private key. The bank stores $g^{sk_i}, pk_{\mathsf{Sig},i}$ and other information about the user.

ATM coin requests: An ATM periodically requests new coins from the bank using the interface ReqCoin(n) interface (Fig. 3). This interface requires as input the number of coins n. (We implicitly assume that the bank can verify that the request is from an authorized ATM, e.g, using certificates).

The ATM randomly selects n keys $\langle a, b \rangle_{\ell \in [n]}$ for PRF and commits to them (Step 2). The ATM also commits to its private identity key. The bank blindly signs the hash of the committed keys (Steps 7–9), and *each pair of signed commitments + blind signature constitutes a coin.* The ATM stores the coins and the corresponding PRF keys in a local key store CS. These coins are later dispensed to the users. The bank may refuse the ATM's request, e.g., if the bank wants to rate-limit the ATM from issuing more coins.

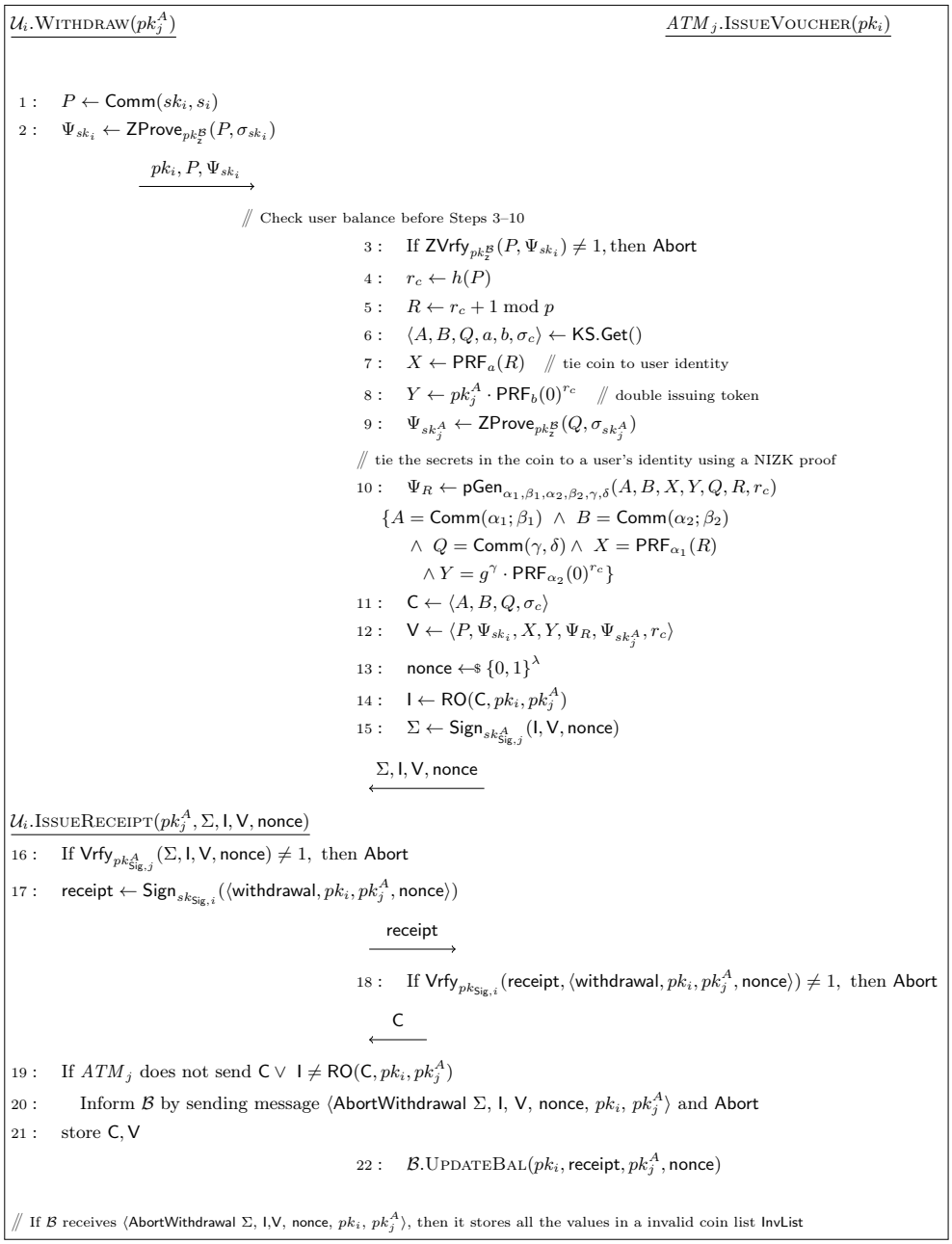

Fig. 4: π_{nc}: Withdrawing a coin

Withdrawing from ATM: When a user request a withdrawal from an ATM, it is issued a coin(s) and a voucher. This is processed using the IssueVoucher

interface (Fig. 4). At a high level, the algorithm spends a coin that has been requested by the ATM before using the ReqCoin interface, where the user acts as the merchant. The transaction record becomes the voucher.

In more details, the user commits to its identity private key in P, and proves in zero-knowledge that is has previously obtained a signature from the bank on the key (Steps 1–2). After verifying the signature and ensuring that the user has enough balance for a withdrawal, the ATM maps P to an element in $\mathbb{Z}_p$ by using a collision-resistant hash function, and computes a voucher ID R derived from the the commitment. Then, the ATM retrieves metadata from CS which includes: i) the commitments to two PRF keys and the ATM's private identity key, A, B, Q respectively, ii) the PRF keys committed to a, b, and iii) the bank's signature σ_c. The commitments to the PRF keys and the ATM's private key along with the bank's signature constitutes a coin that will issued to the user. The ATM computes a voucher that is linked to the coin, and primarily includes two tokens X, Y that i) tie the user's private key to the coin using a (Step 7), and ii) tie the issuing ATM's identity to the coin + voucher in a double-issuing token computed using b (Step 8). Finally, the ATM computes two NIZK proofs showing that Q is in fact a commitment to the ATM's private key that has been previously (blindly) signed by the bank, and that the tokens X, Y have been correctly computed (Step 10)

Note that the bank's signature on the (hash of) signed commitments in the coin shows that the voucher has been computed by an ATM that has been authorized to issue the coin by the bank. By including the double-issuing token X in the voucher, we ensure that if the same coin is issued by the ATM twice, then with overwhelming probability the bank will learn the identity of the offending ATM. Evaluating the PRF under the committed keys on the user's private key commitment ensures that the coin + voucher is cryptographically linked to the user's identity. This is violated only if the user can open the commitment to multiple values, or modify the tokens in the voucher and forge a NIZK proof without the witnesses, i.e., the PRF keys committed to in A, B by the ATM. Both these cases should happen with only negligible probability.

Fair exchange of coins: The IssueReceipt interface is invoked by the user when receiving a coin from ATM_j to produce a receipt. However, to ensure an optimistic fair exchange, we follow a three-step process. First, the ATM commits to the coin by computing a hash of the coin, I, using a random oracle RO, and public keys of the user and the ATM. The ATM signs the hash, the voucher and a random nonce which acts as a unique identifier for the withdrawal and sends the signature to the user (Step 14). The user verifies the signature and then sends to the ATM a receipt which is essentially a signature on the nonce and the public keys of the user and the ATM (Step 17). After verifying the receipt, the ATM sends the coin to the user. If the coin is not well-formed, or the user does not receive the coin after sending the receipt, then the user sends all the information received from the ATM to the bank in an $\mathsf{AbortWithdrawal}$ message.

Upon receiving the $\mathsf{AbortWithdrawal}$ message, the bank stores the commitment and identities of the user and the ATM in an invalid coin list $\mathsf{InvList}$. If

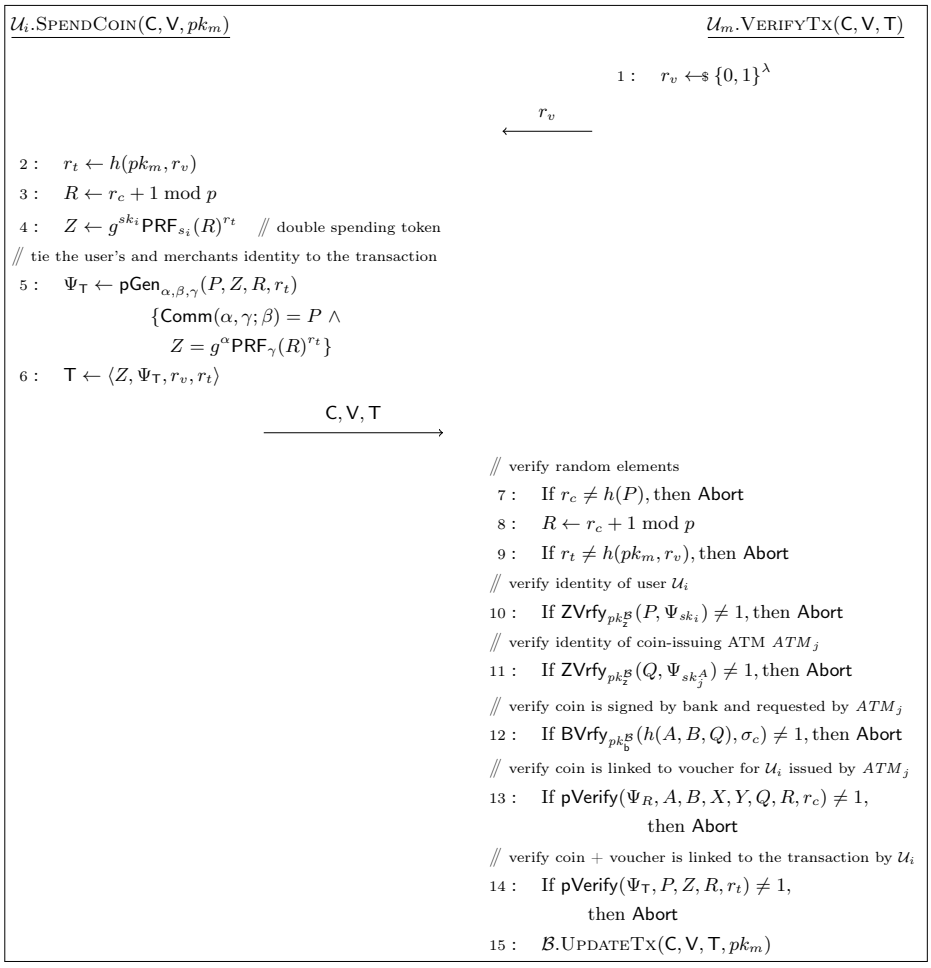

Fig. 5: π_{nc}: Spending a coin

a coin is added to the invalid list, then it should not be spent in any transaction later. Thus, for every transaction submitted, the bank checks whether the coin C' that is part of the transaction is in the invalid list by checking whether $\mathsf{I} = \mathsf{RO}(\mathsf{C}', pk_i, pk_j^A)$. If there is a match, the bank can distinguish whether the coin was issued by the ATM to a different user by checking if the transaction has a different voucher than the one in InvList. If so, it can verify that both of the vouchers were issued by the same ATM by using the double-issuing token (as will be discussed later). Otherwise, the bank identifies that the user sent a false AbortWithdrawal message, and takes appropriate action.

Spending a coin: User $\mathcal{U}_i$ invokes SpendCoin to spend a coin at merchant. Intuitively, this process ties the coin to the merchant's identity using a transaction. For this, the user first computes a "double-spending token" using the PRF

key signed by the bank during initialization. In particular, the token ties the voucher ID R to the private key of the user and the public key of the merchant (Step 4). The user also generates proofs of knowledge to show that the private key embedded in the token corresponds to the private key in the commitment P that is included as part of the voucher. The merchant is provided the coin, the voucher, the double spending token and the proof of knowledge. The merchant verifies the transaction, and if successful, reports the transaction to the bank.

Identifying double spenders: Upon being reported a transaction, the bank invokes $\mathcal{B}.\textsc{UpdateTx}(\mathsf{C}, \mathsf{V}, \mathsf{T}, \mathcal{U}_k)$ and verifies that the coin is valid, the voucher is linked to the coin, and the transaction is valid. If the transaction is valid, the bank checks if the coin is already recorded in the log CS, in which case it invokes the double spending detection algorithm. Specifically, the bank checks for two cases: i) the user has double-spent the coin, or ii) an ATM has double-issued the coin. The process for the former is nearly identical to the compact e-cash design, and computes $\mathsf{PRF}_{s_i}(R)$ from the double spending tokens Z and Z' in the transactions that spend the same coin (Steps 9–15). Then, the bank is able to obtain the public key of the offending user. The latter also follows the same process but uses the double-issuing tokens Y and Y' for obtaining the public key of the offending ATM. We show in the full version [16] that the algorithm fails if an adversary produces an incorrectly computed voucher/transaction that the bank verifies as valid, which happens with negligible probability.

4 Description of Protocol with Compactness

Fig. 7 describes the e-cash protocol with compactness. The main differences are that the an ATM can reuse the same PRF key pair a, b more than once, but at most n times. For this, the bank blindly signs these keys with a scheme that supports proof of knowledge of a signature. When the ATM issues a coin to a user it first commits to the keys, and proves that it knows a signature on the openings of the commitments (Steps 4–7). To show that the key hasn't been used more than n times, the ATM uses the value of a counter ctr as a serial number, and evaluates the PRF on the serial number (Steps 11–14). The ATM also proves in zero knowledge that the value of the counter does not exceed n (Step 15). Note that the value of the counter is not provided in the clear.

We remark here the notable differences from the non-compact protocol described in Fig. 4. First, the two PRF evaluations X and Y are on the *same* value, namely the serial number ctr. This is because we need to ensure that an adversary cannot reuse the values of X and Y as part of different coins. Thus, evaluating the PRF on ctr, whose value is not known to the adversary, and tying the evaluations using the NIZK proof Ψ_R ensures that each pair of evaluations X and Y can be used only once and as part of the same coin. This is not necessary in the non-compact version because the commitments to the keys used for evaluating X and Y are tied together in a coin using the hash function $h(A, B, Q)$. Thus, if the adversary provides X evaluated under the opening of A in a voucher,

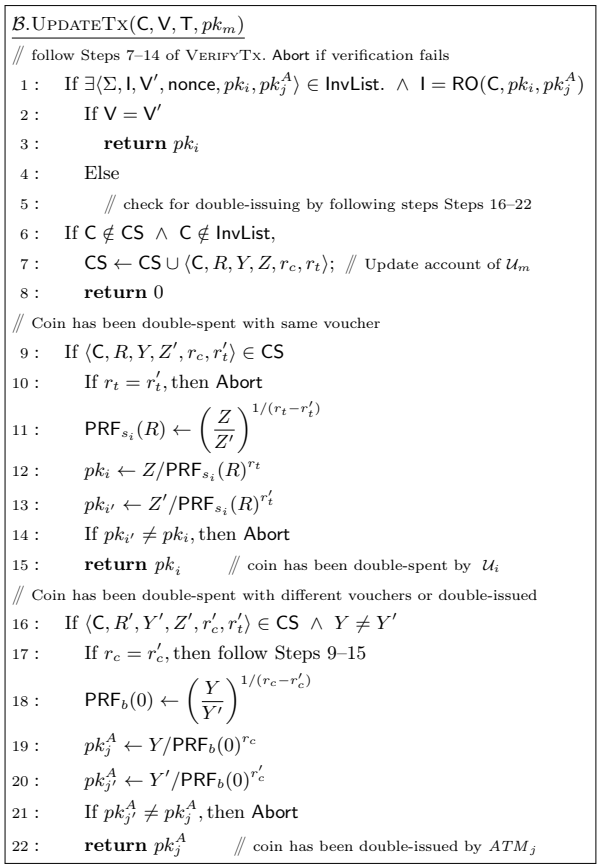

Fig. 6: π_{nc}: Verifying transactions

then the same voucher must also include Y evaluated under the opening of B, unless there is a collision in h.

Second, in the compact version $X = \mathsf{PRF}_a(\mathsf{ctr})$, *independent of the user identity*. In the non-compact version (Fig. 4), the evaluation $X = \mathsf{PRF}_a(P)$ is on the commitment to the user's private key; this ensures that coins cannot be transferred between users without sharing keys, or an adversary computing $X = \mathsf{PRF}_a(P')$ without knowing a. In the compact version, to transfer a coin + voucher, the double spending token $Y = \mathsf{PRF}_b(\mathsf{ctr})^{r_c}$ would need to be recomputed by an adversary, which also entails computing the NIZK proof Ψ_R without the secrets a, b, Q, ctr. Also note that, similar to the non-compact version, the user still needs to tie the coin to its identity by using the token $R \leftarrow h(P, J)$ for using the PRF, and proving that its knows the opening of P in Ψ_{T}.

Fig. 7: π_{comp}: Withdrawing a coin

5 Fully Offline Withdrawals

We have assumed so far that the ATMs will always check (and update) balances from the bank before dispensing coins to a user, although the coins that are dispensed are withdrawn offline. We can allow an ATM to defer checking a user's account balance until later, i.e., off the critical path of dispensing coins, provided that (i) the user had a sufficiently large balance when the ATM last checked, and (ii) users are globally rate-limited in their abilities to withdraw.

The first condition can be fulfilled if the bank broadcasts a set of accounts that do not have the minimum withdrawal balance. To make this list concise, the bank will represent this set as a Bloom filter. When a user requests cash,

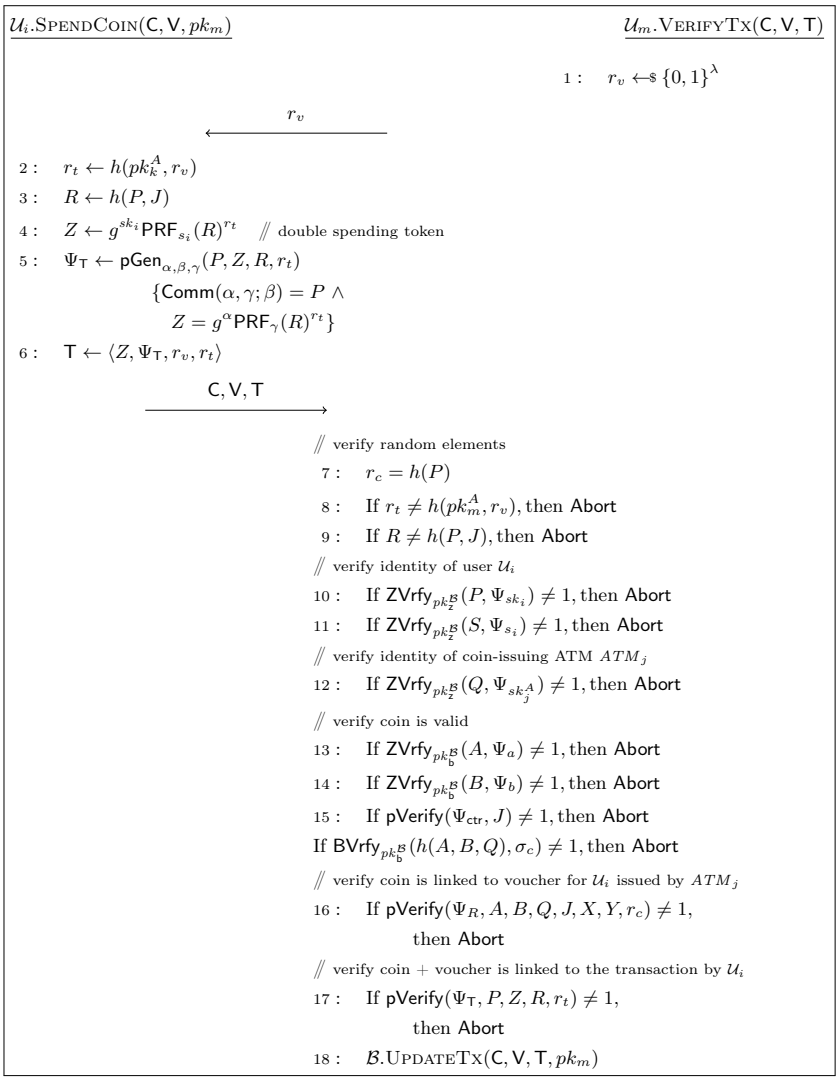

Fig. 8: π_{comp}: Spending a coin

the ATM will first check whether the user's account number is in the Bloom filter; if not, it will dispense the required cash. The Bloom filter can be tuned for negligible false positives, since the set of all possible queries is known [24].

For the second condition, we need to prevent a user from withdrawing simultaneously from multiple ATMs. If the ATMs are edge devices and users are required to present some unforgeable credentials to access them, then implicitly the user may not be able to visit multiple ATMs to overdraw their account, unless they are able to forge their credentials. In such cases, standard authentication systems that measure physical property, like biometrics can be used to

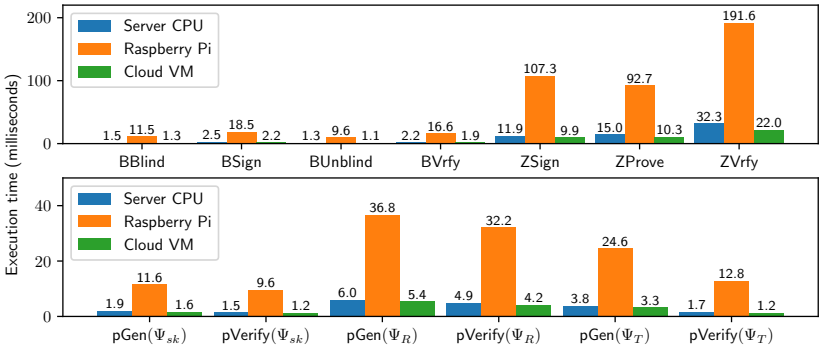

Fig. 9: Computation overhead of cryptographic functions.

ensure that a user is physically present during withdrawals. We will ensure that the time period between two subsequent updates from an ATM must be bounded so that a user cannot visit a large number of ATMs in that time period.

Alternatively, if the ATMs are digital entities that users connect to, then globally rate-limiting a user's coin withdrawals might be accomplished using a public random beacon (e.g., as implemented by NIST [23]). If each user holds a private key for a verifiable random function family (VRF) [28], for which the bank (and its ATMs) holds the corresponding public key, then an ATM can permit coin withdrawal by this user only if the user's VRF, applied to the current beacon value, designates this ATM as the one at which the user is currently eligible to withdraw. In this way, the user will be rate-limited to withdraw at only one ATM per beacon pulse and, then, only the number of coins that ATM allows.

6 Microbenchmarks

We implemented our e-cash scheme with blind signatures on a 2048-bit RSA group, blind signature with proofs of knowledge on the elliptic curve BLS12_381 (CL signature [12]), and non-interactive zero-knowledge proofs of knowledge on a 2048-bit RSA groups. BLS12_381 is a pairing-friendly elliptic curve providing 128-bit security, and is widely adopted in ZCash [38] and Ethereum [20].

The scheme is implemented in 1,694 lines of C++. The CL signature [12] and pseudorandom functions (PRFs) [19] are implemented using the MIRACL library [29], the NIZK is implemented on ZKPDL [27], while the other cryptographic functions are implemented on OpenSSL [21].

We benchmarked the performance on a single core of three CPU platforms:

- Intel Xeon Silver 4110 8-core CPU @ 2.10GHz of around $1,400 representing the typical computation power on servers.
- Raspberry Pi 4 with quad-core Cortex-A72 64-bit SoC @ 1.5GHz costing $40, as the most affordable choice.
- Amazon AWS virtual machine (t2.micro, $0.28 per day) for benchmarking the performance on a low-resource cloud system.

Table 2: Size and execution time of each protocol step in the ATM e-cash scheme.

(a) Protocol object sizes

Object	KB
Coin C	1
Voucher V	54
Transaction T	2

(b) Communication cost per protocol step

Messgage direction	KB
Coin request from bank (ReqCoin/IssueCoin)	
ATM $\rightarrow$ bank (K)	0.3
Bank $\rightarrow$ ATM (σ_c')	0.3
Coin withdrawal (Withdraw/ IssueVoucher)	
User $\rightarrow$ ATM (pk_i, P, Ψ_{sk_i})	25
ATM $\rightarrow$ User (C, V)	55
Coin spending (SpendCoin/ VerifyTx)	
User $\rightarrow$ Merchant (C, V, T)	57

(c) Execution time (ms)

CPU	Withdraw/ IssueVoucher		SpendCoin/ VerifyTx	
	User	ATM	User	Merchant
Server CPU	15.52	58.57	4.33	67.77
Raspberry Pi	94.56	414.13	26.46	397.62
Cloud VM	10.80	46.86	3.80	64.55

Execution time in milliseconds.

Cryptographic functions: Fig. 9 illustrates the execution time of cryptographic functions. Across various hardware platforms, the server CPU and cloud VM exhibit similar performance, whereas the Raspberry Pi is approximately 8$\times$ slower. The most expensive function is the verification of CL signature, while it can be finished in 200 ms on Raspberry Pi 4. Overall, the benchmarking results demonstrate that our e-cash scheme is well-suited for real-time applications.

Object sizes & communication: Table 2 presents the concrete sizes of the protocol objects alongside the execution time of each protocol step across the three hardware platforms for the non-compact version of the protocol. The coin C consists of three Pedersen commitments and one blind signature, giving a size of 1KB. The voucher V is dominated by two CL signature proofs (Ψ_{sk_i} and $\Psi_{sk_j^A}$), which together account for $\approx$93% of its 55KB size. The transaction T is roughly 2KB, containing the double-spending token, a NIZK proof with 3 witnesses, and two random elements. Importantly, the coin size is *constant* regardless of the number of users, ATMs, or prior transactions — only the voucher carries the zero-knowledge proofs needed for verification.

Introducing compactness increases the latency for withdrawal since the voucher now includes an additional zero-knowledge range proof. The spending time also slightly increases due to the additional proof verification time. While the compact scheme can be used when storage resources are limited, we believe that

for most deployment scenarios, ATMs will have enough storage to stock coins without compactness, due to the small size of the coins in our protocol. Without compactness, the ATMs will also need more interaction with the bank to fetch coins (invoking ReqCoin/ IssueCoin), but the total communication per coin is ≈0.6KB. Thus, fetching around 100,000 coins in a batch requires roughly 66MB of total communication, and storing them requires roughly 100MB of storage.

End-to-end compute latency: Coin withdrawal from an ATM (Withdraw + IssueVoucher) is the most expensive online step for the user, as its proves in zero-knowledge the possession of a valid CL signature. During the withdrawal, the ATM computes the voucher for the user. On the server CPU, user-side computation takes 15.52 ms and ATM-side computation takes 58.57 ms. On the Raspberry Pi, these increase to 94.56 ms and 414.13 ms respectively, approximately 7× slower, consistent with the per-function overhead observed in Fig. 9.

Spending a coin (SpendCoin) is lightweight on the user side: 4.33 ms on the server CPU and 26.46 ms even on the Raspberry Pi, as it only requires computing the double-spending token and the transaction proof Ψ_{T}. Verification at the merchant (VerifyTx) takes 67.77 ms on the server CPU and 397.62 ms on the Raspberry Pi, dominated by the CL signature verifications of user and ATM credentials, the blind signature check (BVrfy) on the coin, and the NIZK proof verifications. The cloud VM performs comparably to the server CPU. Overall, the total latency for the online withdrawal path (user + ATM) is 74.09 ms on the server CPU and 508.69 ms on the Raspberry Pi, while the spending path (user + merchant) completes in 72.10 ms and 424.08 ms respectively. These results confirm that our ATM e-cash protocol is practical for real-time applications

7 Conclusion

We presented a new cryptographic bearer token design for offline e-cash systems, enabling anonymous, unforgeable, and untraceable withdrawals from decentralized ATMs. Our construction leverages an unforgeable and *doubly-anonymous* voucher that allows a one-time transfer of coins between an ATM and a user, while hiding their identities from parties not involved in the transaction.

8 Acknowledgements

This work was supported in part through NSF award 2112562, 2425891 and 2425892. We thank our shepherd and the anonymous ACNS reviewers for their excellent suggestions and feedback. The views and conclusions in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the National Science Foundation.

Bibliography

[1] Atm malware. https://www.cyber.nj.gov/threat-landscape/malware/atm-malware

[2] Asokan, N., Shoup, V., Waidner, M.: Optimistic fair exchange of digital signatures. In: Advances in Cryptology – EUROCRYPT. Springer (1998)

[3] Baldimtsi, F., Chase, M., Fuchsbauer, G., Kohlweiss, M.: Anonymous transferable e-cash. In: Public Key Cryptography — PKC. Springer (2015)

[4] Barreto, P.S., Naehrig, M.: Pairing-friendly elliptic curves of prime order. In: Workshop on Selected Areas in Cryptography. Springer (2005)

[5] Bauer, B., Fuchsbauer, G., Qian, C.: Transferable e-cash: A cleaner model and the first practical instantiation. In: Public Key Cryptography — PKC. Springer (2021)

[6] Bellare, M., Rogaway, P.: Random oracles are practical: A paradigm for designing efficient protocols. In: 1st ACM Conference on Computer and Communications Security (Nov 1993)

[7] Bellare, M., Rogaway, P.: Code-based game-playing proofs and the security of triple encryption (2004)

[8] Brands, S.: Untraceable off-line cash in wallet with observers. In: Advances in Cryptology – CRYPTO. Springer (1994)

[9] Bürk, H., Pfitzmann, A.: Value exchange systems enabling security and unobservability. Computers & Security **9**(8) (1990)

[10] Camenisch, J., Hohenberger, S., Lysyanskaya, A.: Compact e-cash. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. Springer (2005)

[11] Camenisch, J., Lysyanskaya, A.: A signature scheme with efficient protocols. In: International Conference on Security in Communication Networks. Springer (2003)

[12] Camenisch, J., Lysyanskaya, A.: Signature schemes and anonymous credentials from bilinear maps. In: Advances in Cryptology – CRYPTO (2004)

[13] Camenisch, J., Lysyanskaya, A., Meyerovich, M.: Endorsed e-cash. In: IEEE Security and Privacy. IEEE (2007)

[14] Canard, S., Gouget, A.: Anonymity in transferable e-cash. In: International Conference on Applied Cryptography and Network Security. Springer (2008)

[15] Canard, S., Pointcheval, D., Sanders, O., Traoré, J.: Divisible e-cash made practical. In: Public Key Cryptography — PKC. Springer (2015)

[16] Chakraborti, A., Zhang, Q., Peng, J., Mao, M., Reiter, M.K.: Automatic teller machines for offline e-cash (2026), https://arxiv.org/abs/2604.10380

[17] Chaum, D.: Blind signatures for untraceable payments. In: Public Key Cryptography — PKC. Springer (1983)

[18] Chaum, D., Fiat, A., Naor, M.: Untraceable electronic cash. In: Advances in Cryptology – CRYPTO '88. Lecture Notes in Computer Science, vol. 403. Springer-Verlag (1990)

[19] Dodis, Y., Yampolskiy, A.: A verifiable random function with short proofs and keys. In: Public Key Cryptography — PKC. Springer (2005)
[20] Ethereum: Ethereum official website. https://ethereum.org/en/ (2024), accessed: 2024-06-02
[21] Foundation, O.S.: OpenSSL: The open source toolkit for SSL/TLS (2024), https://www.openssl.org/
[22] Groth, J., Sahai, A.: Efficient non-interactive proof systems for bilinear groups. In: Advances in Cryptology – EUROCRYPT. Springer (2008)
[23] Kelsey, J., Brandão, L.T., Peralta, R., Booth, H.: A reference for randomness beacons: Format and protocol version 2. Tech. rep., National Institute of Standards and Technology (2019)
[24] Larisch, J., Choffnes, D., Levin, D., Maggs, B.M., Mislove, A., Wilson, C.: Crlite: A scalable system for pushing all tls revocations to all browsers. In: IEEE Security and Privacy. IEEE (2017)
[25] Liu, J., Hu, Y.: A new off-line electronic cash scheme for bank delegation. In: International Conference on Information Science and Technology (2015)
[26] Liu, J., Liu, J., Qiu, X.: A proxy blind signature scheme and an off-line electronic cash scheme. Wuhan University Journal of Natural Sciences (2013)
[27] Meiklejohn, S., Erway, C.C., Küpçü, A., Hinkle, T., Lysyanskaya, A.: Zkpdl: A language-based system for efficient zero-knowledge proofs and electronic cash. In: USENIX Security Symposium. vol. 10 (2010)
[28] Micali, S., Rabin, M., Vadhan, S.: Verifiable random functions. In: IEEE Symposium on Foundations of Computer Science. IEEE (1999)
[29] MIRACL Ltd.: Miracl cryptographic sdk. https://github.com/miracl/MIRACL (2021), accessed: 2026-04-19
[30] Okamoto, T., Ohta, K.: Disposable zero-knowledge authentications and their applications to untraceable electronic cash. In: Advances in Cryptology – CRYPTO. Springer (1989)
[31] Pagnia, H., Gärtner, F.C., et al.: On the impossibility of fair exchange without a trusted third party. Tech. rep. (1999)
[32] Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. In: Advances in Cryptology – CRYPTO. Springer (1991)
[33] Pointcheval, D., Stern, J.: Provably secure blind signature schemes. In: Advances in Cryptology – ASIACRYPT. Springer (1996)
[34] Sander, T., Ta-Shma, A.: Auditable, anonymous electronic cash. In: Advances in Cryptology – CRYPTO. Springer (1999)
[35] Sandholm, T., Wang, X.: (im) possibility of safe exchange mechanism design. In: Eighteenth national conference on Artificial intelligence (2002)
[36] Shi, L., Carbunar, B., Sion, R.: Conditional e-cash. In: International Conference on Financial Cryptography and Data Security. Springer (2007)
[37] Tan, Z.: An off-line electronic cash scheme based on proxy blind signature. The computer journal
[38] Zcash: Zcash official website. https://z.cash/ (2024), accessed: 2024-06-02
[39] Zhang, F., Safavi-Naini, R., Susilo, W.: Efficient verifiably encrypted signature and partially blind signature from bilinear pairings. In: International Conference on Cryptology in India (INDOCRYPT) (2003)