Platypus: A Central Bank Digital Currency with Unlinkable Transactions and Privacy-Preserving Regulation

Karl Wüst[*]
CISPA Helmholtz Center
for Information Security
Germany

Kari Kostiainen
Department of Computer Science
ETH Zurich
Switzerland

Noah Delius
Department of Computer Science
ETH Zurich
Switzerland

Srdjan Capkun
Department of Computer Science
ETH Zurich
Switzerland

ABSTRACT

Due to the popularity of blockchain-based cryptocurrencies, the increasing digitalization of payments, and the constantly reducing role of cash in society, central banks have shown an increased interest in deploying central bank digital currencies (CBDCs) that could serve as a digital cash-equivalent. While most recent research on CBDCs focuses on blockchain technology, it is not clear that this choice of technology provides the optimal solution. In particular, the centralized trust model of a CBDC offers opportunities for different designs.

In this paper, we depart from blockchain designs and instead build on ideas from traditional e-cash schemes. We propose a new style of building digital currencies that combines the transaction processing model of e-cash with an account-based fund management model. We argue that such a style of building digital currencies is especially well-suited to CBDCs. We also design the first such digital currency system, called Platypus, that provides strong privacy, high scalability, and expressive but simple regulation, which are all critical features for a CBDC. Platypus achieves these properties by adapting techniques similar to those used in anonymous blockchain cryptocurrencies like Zcash to fit our account model and applying them to the e-cash context.

1 INTRODUCTION

Recent research on digital currencies has mostly focused on blockchains such as Bitcoin [32] instead of traditional e-cash systems such as [14]. This is mostly due to the popularity of blockchains for permissionless digital currencies, i.e., digital currencies that do not rely on a trusted central authority.

Inspired by the popularity of blockchains, several central banks, such as Swedish central bank [39] and the Bank of England [9], have expressed interest in creating a digital version of their currency. The People's Bank of China [44] has already deployed a digital yuan into trial use. Recently, several central banks, together with the Bank of International Settlements, have outlined the principles and core features of such a central bank digital currency (CBDC) [8].

A CBDC has a different trust model and different requirements from permissionless cryptocurrencies. Namely, the central bank is generally a trusted authority and the consensus process should not be open to everyone. Nevertheless, decentralized ledgers have often been proposed for such central bank digital currencies [5, 17, 42], since they offer benefits over traditional e-cash such as increased robustness due to the distributed consensus, as well as transferability due to the ledger-based system. While traditional e-cash [14] provides privacy for the sender, it leaks the transaction amounts since the coins need to be deposited immediately for double-spending protection. In ledger-based systems, coins are not deposited, but instead value is transferred, which allows for private transactions such as in Zerocash [38].

However, e-cash systems have several advantages compared to ledger-based systems. Namely, e-cash systems are easier to scale, mainly because they do not require byzantine agreement between independent parties due to their centralized nature. This has particular implications on their sharding potential. For example, depositing coins can easily be sharded based on the serial number of the deposited coin. Since the coins are signed by the central authority, there is no need to check (potentially cross-shard) if the coin was produced as an output of a previous transaction (such as in ledger-based systems), and instead it suffices to check that the coin is signed by the central authority and that the serial number has not been seen before. Further, the requirements for clients can potentially be reduced compared to ledger-based systems, since in ledger-based systems, clients keep up to date with the whole ledger or use a lightweight client, which reduces their privacy [24] without the use of additional mechanisms that require additional trust assumptions [30, 43] or expensive cryptographic protocols [29].

We want to leverage the different trust model of central bank digital currencies and combine the benefits of ledger-based digital currencies and traditional e-cash schemes. Namely, we assume an authority that is trusted for the integrity of the currency (e.g. double-spending protection) but is not trusted for privacy, a setting that has been proposed by several central banks [8, 16]. We want to make use of the performance benefits from traditional e-cash schemes, but combine them with a transaction mechanism inspired by anonymous ledger-based cryptocurrencies like Zerocash [38] that provides anonymity for the sender and recipient as well as secrecy of the transaction amounts. In addition, the mechanism should be easy to extend with regulation mechanisms for e.g. money laundering protection similar to [23, 42].

To achieve these goals, as the first main contribution of this paper, we propose a new style of building digital currencies that combines the transaction processing model of e-cash payments

[*] Work partially done while at ETH Zurich

with an account-based model for managing users' funds (which is also used in some ledger-based systems like Ethereum [41]). We argue that this style of building digital currencies is particularly well-suited to CBDCs and allows us to achieve strong privacy, high scalability, and simple but expressive regulation, which are all desirable features for a CBDC.

As the second main contribution of this paper, we design the first digital currency system, called Platypus[1], that follows this design pattern. Platypus is also inspired by previous anonymous blockchain-based cryptocurrencies such as Zerocash [38]. In Platypus, each participant owns an account that is represented by a commitment, called *account state commitment*, to a serial number and a balance and which is signed by the central bank. A transaction then consists of updating the commitments of both, the sender and recipient. The sender and recipient reveal the serial number of their current account states, prove in zero-knowledge that they are in possession a corresponding state commitment signed by the central bank, and that the sum of their balances remains invariant.

Such a design provides advantages over anonymous ledger-based designs as well as over UTXO-based designs (e.g. Zcash). The main advantage over ledger-based designs lies in the scalability of such an approach. Since transactions do not need to be ordered on a ledger and the system is centralized, transaction validation can be sharded almost arbitrarily using standard database techniques such as two-phase commit [28] and thus there is no inherent limit on its throughput. In addition, the requirements for clients are reduced significantly. In systems like Zcash, clients need to download and decrypt every transaction stored on the ledger if they want to benefit from Zcash's privacy guarantees. If a currency is to be widely used, as expected from a CBDC, downloading and decrypting every transaction quickly becomes infeasible for most users, who may want to use this currency on a mobile device.

Another advantage of the account-based design is simplified enforcement of regulatory rules. If enrollment in the system is bound to real identities, this account-based design can simplify regulatory rules similar to those proposed by Garman et al. [23] and Wüst et al. [42]. In particular, it enables enforcement of regulatory rules that, e.g., limit the amount of funds that a particular user can posses at a time (as mentioned e.g. by the Bank of England [9]), or that require disclosure of the user's identity if a certain limit for receiving funds within a period of time is exceeded. However, in contrast to [23], this enforcement is more flexible since it can be applied on the sender or recipient side (instead of only the sender) and it can be done more efficiently since it does not require aggregation over multiple transaction outputs. In contrast to [42], it preserves full transaction unlinkability.

This design also provides advantages over more traditional e-cash schemes like the original proposal by Chaum [14] and optimizations using similar principles [11, 12], in which a bank issues blinded coins to a user, who then spends them at one or more merchants, who deposit them back in the bank. Namely, one of the main advantages is that our account-based approach does not require spending of individual coins, which has two important effects.

First, this leads to a more compact scheme, since the transaction size and the verification cost do not increase with the transaction value. In traditional e-cash schemes, each coin is spent individually which means that the transaction size depends on the transaction value whereas the size can stay constant in an account-based design. Even in divisible e-cash [13] – which shrinks the transaction size to logarithmic in the value – each coin is still deposited individually, i.e. the verification cost at the bank is linear in the transaction value.

Second, and more importantly, this improves privacy: In traditional e-cash systems, the amount that a merchant receives always leaks, since they need to deposit the received coins at the bank. In the case of an online e-cash system, this immediately leaks all transaction values of the merchant, in an offline system, this leaks the amount that is received between two deposits. Thus, traditional e-cash systems only provide *payer privacy*, but not *recipient privacy* or *value privacy*. With our account-based design, the size of a transaction is independent of its value and the funds are immediately deposited in the blinded account of the recipient. This ensures the anonymity of the sender and recipient, the confidentiality of the transaction value, and the unlinkability of transactions.

Contributions. In this paper, we make the following contributions:

- A *new pattern of building digital currencies* that combines the transaction processing model of e-cash with an account-based fund management model.
- A *new digital currency design* called Platypus that provides unlinkable transactions, high scalability, and privacy-preserving regulation mechanisms which are all critical features for a CBDC.
- A *security analysis* that shows that Platypus provides integrity and strong privacy guarantees.
- *An implementation and evaluation* that show that transaction creation is fast and Platypus can be easily scaled.

2 OVERVIEW

In this section we provide an overview of Platypus. We start by explaining our motivation and goals, followed by the trust assumptions and our system model. After that, we explain the main ideas of Platypus.

2.1 Motivation & Goals

Recently, multiple central banks together released a report detailing the principles, motivations and risks of CBDCs [8]. This serves as a good basis for technical decisions in the design of a CBDC since it directly provides the view of the involved central banks.

One of the main motivations outlined in [8] is continued access to central bank money, i.e. the function of a CBDC as a form of a "digital banknote". Currently, both, access and the use of cash are declining in many jurisdictions, which creates the risk that some businesses and households lose access to risk-free central bank money. A CBDC could step in to fill this void to ensure the confidence in a currency.

Cash does not only provide risk-free central bank money, but it also provides very strong privacy guarantees. In a cash payment, third parties neither learn the identities of the parties nor the value. This is a property that should also be mirrored by a CBDC [5, 8]. A

[1] Similar to how its namesake combines features from different animals, Platypus combines ideas from e-cash, blockchains, and bank accounts.

working paper from the Swiss National Bank [16] explicitly mentions "mass surveillance" as one of the potential threats of a CBDC, which exemplifies the need for strong privacy guarantees and a consultation from the European Central Bank [21] showed that privacy is the most important feature of a CBDC for the survey respondents.

A CBDC could also increase resilience and the diversity of payment systems, improve financial inclusion, and simplify cross-border payments if the CBDCs of multiple countries are interoperable [8]. Lastly, even though this is not stated as one of the main motivations of [8], a CBDC could be used for "programmable monetary policy" to e.g. provide so-called "helicopter drops" that distribute funds to the public combined with an expiration date for spending these funds.

CBDCs also create some risks for financial stability [5, 8]. In particular, it can lead to a form of bank runs, since it provides a convenient way (in contrast to paper money) of storing their funds as central bank money. One of the potential mitigations for this risk is to explicitly design the currency as a cash-like system that, e.g., enforces limits on how much currency can be held by a single party at a time. Because of this, allowing the enforcements of such limits is one of the central regulatory goals for such a digital currency.

Another regulatory requirement for CBDCs is the enforcement of anti-money-laundering (AML) legislation [5, 8]. However, this partially conflicts with the goal of improved payment privacy. This conflict can be solved by allowing anonymous payments up to a given limit per unit of time above which the recipient needs to disclose their identity to a regulator. This idea has also been proposed by the European Central Bank in the form of "anonymity vouchers" [20] as well as by previous work [23, 42].

Based on these motivations and ideas, we focus on a *retail CBDC* that can be used as a digital equivalent of cash since this is the main use case considered by central banks [8]. Other settings for CBDCs exist, e.g., replacement of a settlement layer or to replace bank transfers. In such settings other properties may become relevant, such as offline receiving that we discuss in Appendix B.

Given our focus, our main goal is to provide a digital currency that is maintained by a central bank and provides fully anonymous transactions, i.e., where the transaction values are secret, the sender and recipient cannot be identified and transactions are unlinkable to previous or future transactions. In addition, this solution should make use of the benefits allowed by the trust model in which a central authority is trusted for integrity (as proposed, e.g., in [16]) and should provide significant performance benefits over other anonymous digital currencies such as Zerocash [38].

As a secondary goal, we want this digital currency to be easily and efficiently extendable with regulation mechanisms similar to those described by Garman et al. [23] and Wüst et al. [42] to make it viable for the use as fiat currency.

2.2 System Model & Trust Assumptions

Motivated by the considerations in Section 2.1, we consider the setting in which a *central bank* wants to issue and maintain a digital currency, as shown in Figure 1. Such a centralized design is proposed by a working paper of the Swiss National Bank [16] and suggested as one possible option in a report from a group of central banks [8].

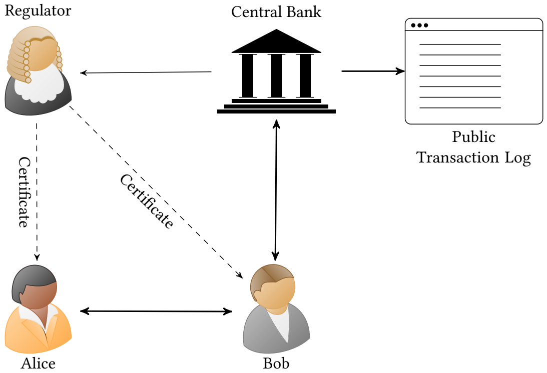

Figure 1: Platypus System model. Platypus consists of a central bank that is responsible for transaction validation, a regulator that issues certificates to clients and receives transaction information relevant for compliance with regulatory rules, as well as clients that participate in the system. The central bank also publishes a log of all transactions.

In addition to this central bank, we assume that there exists a *regulator* (e.g., a government agency), which is responsible for enforcing regulatory requirements, such as anti-money-laundering (AML) legislation. While such a regulator is not necessary for the functioning of the core protocol, it would likely be an integral part in any deployment of a CBDC in practice (see Section 4).

Our system also contains *clients* that can act as payment senders and payment recipients. We assume that these clients are considered untrusted, i.e., they may behave arbitrarily.

Since central banks are responsible for monetary policy, we assume that the central bank is trusted for the integrity of the currency and the regulator trusts the central bank to comply with regulatory requirements. The central bank is responsible for the issuance of new money and preventing double-spending is in its own interest as double-spending would effectively increase the currency in the system.

However, based on our considerations in Section 2.1 and the potential threat of mass surveillance [16], we assume that the central bank is not trusted for privacy, i.e., it might be interested in deanonymizing the sender or recipient of a transaction or recover transaction values.

We consider full protection against network-based deanonymization attacks (e.g., linking an IP address to multiple transactions) to be out of scope of this paper. To protect against such attacks, clients can use protections such as anonymity networks like Tor [3] if desired. However, to provide some resilience against such attacks, we design the system such that the recipient and sender of a transaction cannot be easily linked together by the central bank, even without the use of anonymity networks and provide some discussion about network-based transaction linking in Appendix B. In the normal case (cooperating recipient) this is achieved by only having the recipient communicate with the central bank. For other cases, and to simplify account recovery (see Appendix B), the central bank publishes transactions in a publicly accessible log, which clients can use to look up their previous transactions. This log can be mirrored by third parties, similar to block explorers in blockchain systems.

Finally, we assume that clients communicate with each other through secure channels and that all cryptographic primitives used are secure according to the standard definitions for their security: we assume that commitments are computationally binding and hiding, that signatures are unforgeable, that the zero-knowledge proof systems are zero-knowledge and provide soundness, and that encryption is CPA-secure.

2.3 Platypus Design

Platypus uses a hybrid between an account model and an e-cash design, in which each participant is responsible for keeping track of their own *account state* which is kept as objects similar to coins in an e-cash system. However, in contrast to e-cash, where a client usually has multiple coins that can be used in a transaction, a client has a single account state which is consumed in every transaction and a replaced with a new one. This account state is represented by an *account state commitment* state_i to the account balance bal_i and to a serial number serial_i. The account state commitment is produced by a previous transaction and is signed by the central bank. To sign these state commitments, the central bank uses its secret key sk_C (corresponding to public key pk_C). For enforcement of regulatory policies, the account state may contain additional information as described in Section 4.

Figure 2 shows how a transaction is processed in the normal case where both the sender and recipient have already participated in the system. In step ①, Alice initiates a transaction, in which she sends a value of v_{Tx} to the recipient, Bob, by creating a *sender account update*. Alice creates a commitment, called *transaction commitment*, to the value v_{Tx} using a random blinding factor $\mathsf{blind}_{\mathsf{Tx}}$, denoted by $\mathsf{comm}_{\mathsf{Tx}} = \mathsf{comm}(v_{\mathsf{Tx}}, \mathsf{blind}_{\mathsf{Tx}})$. She then creates a new state commitment state^A_{i+1} that commits to a fresh pseudorandomly (based on her longterm key) chosen serial number serial^A_{i+1} and a value $\mathsf{bal}^A_{i+1} = \mathsf{bal}^A_i - v_{\mathsf{Tx}}$ where bal^A_i is the balance committed to in her current state commitment state^A_i. Alice then creates a non-interactive zero-knowledge proof zkp^A_{i+1} which proves that she performed these steps correctly.

Note that, for this zero-knowledge proof, both the previous account state commitment state^A_i and the central bank's signature are secret values, i.e. they are not revealed in this transaction. This ensures that this transaction is not linkable to the previous transaction in which state^A_i was created and which contains state^A_i and the bank's signature. The zero-knowledge proof potentially also needs to prove compliance with regulatory rules, if a regulation mechanism as described in Section 4 is in place.

This zero-knowledge proof zkp^A_{i+1}, as well as the transaction commitment $\mathsf{comm}_{\mathsf{Tx}}$, the serial number of the old state serial^A_i, and the new account state commitment state^A_{i+1} are then sent to the recipient, Bob. Alice also provides the random value $\mathsf{blind}_{\mathsf{Tx}}$ required to open the commitment $\mathsf{comm}_{\mathsf{Tx}}$, such that Bob can use it to create a zero-knowledge proof for his own account update.

To complete the transaction (step ②), Bob then creates a creating a *receiver account update*, for which he proceeds similarly to Alice, with the difference that his zero-knowledge proof zkp^B_{j+1} reuses the transaction commitment $\mathsf{comm}_{\mathsf{Tx}}$ and proves that his account balance in his new state state^B_{j+1} increases by exactly v_{Tx} compared to his previous state state^B_j with serial number serial^B_j.

Once Bob has created the proof zkp^B_{j+1}, he sends the transaction commitment $\mathsf{comm}_{\mathsf{Tx}}$, Alice' and his serial numbers (serial^A_i, serial^B_j), both of their new state commitments (state^A_{i+1}, state^B_{j+1}), and both zero-knowledge proofs (zkp^A_{i+1}, zkp^B_{j+1}) to the central bank.

The central bank then executes the transaction (step ③) by verifying both zero-knowledge proofs and checking that neither of the serial numbers (serial^A_i, serial^B_j) have been used in previous transactions. If this is the case, the central bank adds both serial numbers to the set of used serial numbers, signs the two new state commitments (state^A_{i+1}, state^B_{j+1}) with their private key sk_C and sends the signatures $\sigma^A_{i+1} = \mathsf{Sign}(sk_C, \mathsf{state}^A_{i+1})$ and $\sigma^B_{j+1} = \mathsf{Sign}(sk_C, \mathsf{state}^B_{j+1})$ back to Bob, who checks if the signatures are valid and, if so, accepts the payment (step ④). Bob then forwards σ^A_{i+1} to Alice, who verifies the signature and updates her stored state information, which completes the payment (step ⑤).

The central bank keeps a record of all recent (i.e., for some specified time interval chosen by the central bank) transactions, which they publish in a publicly accessible way. In particular, for each transaction, the bank publishes all values received from Bob, as well as the bank's signatures on the new account state commitments. This allows Alice to check the set of recently published transactions for the serial number of her old account state to find and receive the transaction containing the signed new state commitments, even if Bob does not forward this information to her. Note that the published set of transactions does not need to be ordered (in contrast to a ledger or blockchain) and can be mirrored by arbitrary parties. To enable efficient account backups and recovery, additional encrypted (with a key of the owner) information about the contents of the transaction can be included in the transaction. We describe backups and account recovery in Appendix B.

The centralized design of Platypus also simplifies sharding, since standard database sharding techniques can be used for checking and updating the serial numbers of the used account states. We describe the sharding potential in more detail in Appendix B.

3 PLATYPUS BASE TRANSACTION DETAILS

In this section, we describe the details of the base transactions in Platypus, i.e., the creation of transactions without regulation mechanisms. We defer the explanation of regulation mechanisms to Section 4 to improve readability and to make the system design easier to understand.

Platypus makes use of zero-knowledge proofs in its transactions. These zero-knowledge proofs can be instantiated with different proof techniques, but the statements that are proven are independent of these techniques. In Section 6, we implement and evaluate Platypus using zk-SNARKs.

Some of these proof-techniques (including that by Groth [26] used in our implementation) require a *trusted setup* to generate a common reference string, which has been criticized in the context of decentralized cryptocurrencies like Zcash, and extraordinary efforts were made to keep it secure when Zcash was originally launched [34]. It is important to note here that, at least in most

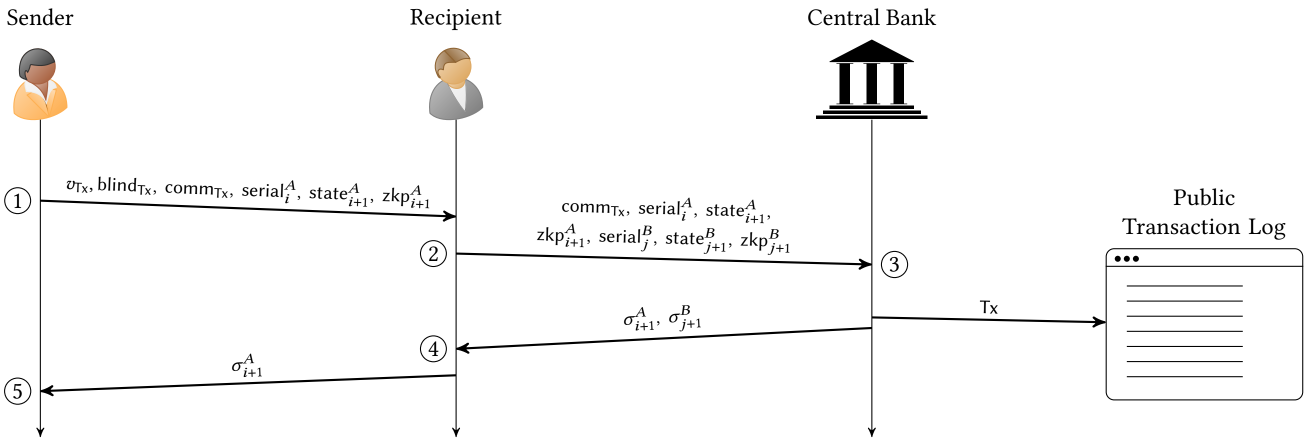
Sender
Recipient
Central Bank
Public Transaction Log
①
v_{Tx}, blind_{Tx}, comm_{Tx}, serial_i^A, state_{i+1}^A, zkp_{i+1}^A
②
comm_{Tx}, serial_i^A, state_{i+1}^A,
zkp_{i+1}^A, serial_j^B, state_{j+1}^B, zkp_{j+1}^B
③
Tx
σ_{i+1}^A, σ_{j+1}^B
④
σ_{i+1}^A
⑤

Figure 2: Platypus Base Transaction. ① *Transaction Initiation.* The sender, Alice, creates the transaction commitment as well as the proofs for the update of her account state commitment and sends all of this to the recipient Bob, together with the blinding value of the transaction commitment. ② *Transaction Completion.* Based on this, Bob creates the proofs for the update of his account state commitment and sends the new account state commitments and both proofs to the central bank. ③ *Transaction Execution.* The central bank verifies the proofs, checks that the revealed serial numbers have not been used before, and based on this either accepts or rejects the transaction. If the bank accepts, it signs the new account states and sends the signatures back to Bob. Simultaneously, the central bank also publishes the full transaction (i.e. everything received from Bob plus the new signatures) on a public transaction log. ④ *Payment Acceptance.* If the signatures are valid, Bob accepts the payment and forwards the signature on Alice' state to her, which completes the payment (⑤ *Payment Completion.*).

constructions including [26], a compromise of this trusted setup does not affect the zero-knowledge property of the proofs. Instead it "only" affects soundness, which in the context of digital currencies allows the creation of money, but does not affect privacy. In our context, i.e., a CBDC, the central bank is the entity that creates money and is trusted for the integrity of the currency, which means that it can therefore be trusted to perform the setup without any additional assumptions. Of course, in practice, it could be preferable to distribute the setup between multiple parties or to use a proof system that does not require a trusted setup.

3.1 System Setup

To set up the system, the central bank creates a private/public key pair (sk_C, pk_C) that is used for signing account state commitments and publishes its public key pk_C. In addition, if a proof system is used that requires the setup of a common reference string (see above), the central bank runs the trusted setup procedure (possibly in conjunction with other parties). In addition, the central bank sets a parameter bal_{max} which is a maximum limit on account balances to prevent value overflows and can be set to a value larger than all realistic values for account balances. Similarly, the entity responsible for regulation generates the parameters required by the regulation, which we describe in Section 4.

User Enrollment. When a user U enrolls in the system, they create a secret key $sk_U = (sk_{U1}, sk_{U2})$, consisting of two randomly chosen keys sk_{U1}, sk_{U2}. These keys can later be used to pseudorandomly derive serial numbers and blinding values for their account states using pseudorandom functions $f_{sk_{U1}}$ and $g_{sk_{U2}}$. Pseudorandom serial numbers prevent possible attacks that could destroy funds [37]. Blinding values could instead also be chosen randomly. However, using pseudorandom values for both simplifies the creation of backups for an account (see Appendix B). If regulation is in place, users may also need to register their identity with the regulator (see Section 4).

To create an enrollment transaction, U derives pseudorandom values $\text{serial}_1^U, \text{blind}_1^U$ from their secret key as $\text{serial}_1^U = f_{sk_{U1}}(0)$ and $\text{blind}_1^U = g_{sk_{U2}}(0)$ and uses them to create a new state commitment $\text{state}_1^U = \text{comm}(\text{serial}_1^U, \text{bal}_1^U, \text{blind}_1^U)$ for an account with no balance, i.e., $\text{bal}_1^U = 0$. U then creates a non-interactive zero-knowledge proof zkp_1^U which proves that the account state commitment corresponds to an account with balance zero, i.e., zkp_1^U proves the following statement:

Given public value

$$\text{state}_1^U$$

I know secret values

$$sk_{U1}, \text{serial}_1^U, \text{blind}_1^U$$

such that

$$\text{state}_1^U = \text{comm}(\text{serial}_1^U, 0, \text{blind}_1^U)$$
$$\text{serial}_1^U = f_{sk_{U1}}(0)$$

U then sends state_1^U and zkp_1^U to the central bank. The bank checks if the proof is correct and then signs the account state commitment state_1^U and sends the signature back to U.

3.2 Transaction Creation

Here, we describe how a transaction between a sender (Alice) and a recipient (Bob) is created. We assume that clients keep all values secret unless mentioned otherwise and that they communicate through secure channels.

Alice' current account state is represented by a commitment $\mathsf{state}_i^A = \mathsf{comm}(\mathsf{serial}_i^A, \mathsf{bal}_i^A, \mathsf{blind}_i^A)$, and similarly Bob's current account state is represented by $\mathsf{state}_j^B = \mathsf{comm}(\mathsf{serial}_j^B, \mathsf{bal}_j^B, \mathsf{blind}_j^B)$ if he already has an account. In addition, both are in possession of a signature from the central bank on their account state commitment, denoted by $\sigma_i^A = \mathsf{Sign}(sk_C, \mathsf{state}_i^A)$ and $\sigma_j^B = \mathsf{Sign}(sk_C, \mathsf{state}_j^B)$, respectively. The commitment can be created using any hiding and binding commitment scheme. The steps correspond to the steps shown in Figure 2.

① **Transaction Initiation:**

(i) To create a transaction to Bob with value v_{Tx}, Alice chooses a fresh random value $\mathsf{blind}_{\mathsf{Tx}}$ and creates a commitment $\mathsf{comm}_{\mathsf{Tx}} = \mathsf{comm}(v_{\mathsf{Tx}}, \mathsf{blind}_{\mathsf{Tx}})$.

(ii) Alice also derives pseudorandom values $\mathsf{serial}_{i+1}^A, \mathsf{blind}_{i+1}^A$ from her secret key as $\mathsf{serial}_{i+1}^A = f_{sk_{A1}}(\mathsf{serial}_i^A)$ and $\mathsf{blind}_{i+1}^A = g_{sk_{A2}}(\mathsf{blind}_i^A)$ and creates a new account state $\mathsf{state}_{i+1}^A = \mathsf{comm}(\mathsf{serial}_{i+1}^A, \mathsf{bal}_i^A - v_{\mathsf{Tx}}, \mathsf{blind}_{i+1}^A)$.

(iii) Alice then creates a non-interactive zero-knowledge proof zkp_{i+1}^A that proves the following statement:
Given public values

$$\mathsf{serial}_i^A, \mathsf{comm}_{\mathsf{Tx}}, \mathsf{state}_{i+1}^A, \mathsf{bal}_{\mathsf{max}}, pk_C$$

I know secret values

$$sk_{A1}, \mathsf{bal}_i^A, \mathsf{bal}_{i+1}^A, \mathsf{blind}_i^A, \sigma_i^A, v_{\mathsf{Tx}}, \mathsf{blind}_{\mathsf{Tx}}, \mathsf{serial}_{i+1}^A, \mathsf{blind}_{i+1}^A$$

such that

$$\begin{aligned} \mathsf{True} &= \mathsf{Vrfy}(pk_C, \mathsf{comm}(\mathsf{serial}_i^A, \mathsf{bal}_i^A, \mathsf{blind}_i^A), \sigma_i^A) \\ \mathsf{comm}_{\mathsf{Tx}} &= \mathsf{comm}(v_{\mathsf{Tx}}, \mathsf{blind}_{\mathsf{Tx}}) \\ \mathsf{state}_{i+1}^A &= \mathsf{comm}(\mathsf{serial}_{i+1}^A, \mathsf{bal}_{i+1}^A, \mathsf{blind}_{i+1}^A) \\ \mathsf{bal}_{\mathsf{max}} &\geq \mathsf{bal}_{i+1}^A \\ \mathsf{bal}_{i+1}^A &= \mathsf{bal}_i^A - v_{\mathsf{Tx}} \\ \mathsf{serial}_{i+1}^A &= f_{sk_{A1}}(\mathsf{serial}_i^A) \end{aligned}$$

(iv) Alice then sends $v_{\mathsf{Tx}}, \mathsf{blind}_{\mathsf{Tx}}, \mathsf{comm}_{\mathsf{Tx}}, \mathsf{serial}_i^A, \mathsf{state}_{i+1}^A, \mathsf{zkp}_{i+1}^A$ to Bob.

② **Transaction Completion:**

(i) After receiving the partial transaction from Alice, Bob derives pseudorandom values $\mathsf{serial}_{j+1}^B, \mathsf{blind}_{j+1}^B$ from his secret key as $\mathsf{serial}_{j+1}^B = f_{sk_{B1}}(\mathsf{serial}_j^B)$ and $\mathsf{blind}_{j+1}^B = g_{sk_{B2}}(\mathsf{blind}_j^B)$ and uses them to create a new account state $\mathsf{state}_{j+1}^B = \mathsf{comm}(\mathsf{serial}_{j+1}^B, \mathsf{bal}_j^B + v_{\mathsf{Tx}}, \mathsf{blind}_{j+1}^B)$.

(ii) If Bob already has an account, Bob creates a non-interactive zero-knowledge proof zkp_{j+1}^B that, similar to Alice' proof (with the difference of proving that his balance *increased* by the transaction value), proves the following statement:
Given public values

$$\mathsf{serial}_j^B, \mathsf{comm}_{\mathsf{Tx}}, \mathsf{state}_{j+1}^B, \mathsf{bal}_{\mathsf{max}}, pk_C$$

I know secret values

$$sk_{B1}, \mathsf{bal}_j^B, \mathsf{bal}_{j+1}^B, \mathsf{blind}_j^B, \sigma_j^B, v_{\mathsf{Tx}}, \mathsf{blind}_{\mathsf{Tx}}, \mathsf{serial}_{j+1}^B, \mathsf{blind}_{j+1}^B$$

such that

$$\begin{aligned} \mathsf{True} &= \mathsf{Vrfy}(pk_C, \mathsf{comm}(\mathsf{serial}_j^B, \mathsf{bal}_j^B, \mathsf{blind}_j^B), \sigma_j^B) \\ \mathsf{comm}_{\mathsf{Tx}} &= \mathsf{comm}(v_{\mathsf{Tx}}, \mathsf{blind}_{\mathsf{Tx}}) \\ \mathsf{state}_{j+1}^B &= \mathsf{comm}(\mathsf{serial}_{j+1}^B, \mathsf{bal}_{j+1}^B, \mathsf{blind}_{j+1}^B) \\ \mathsf{bal}_{\mathsf{max}} &\geq \mathsf{bal}_{j+1}^B \\ \mathsf{bal}_{j+1}^B &= \mathsf{bal}_j^B + v_{\mathsf{Tx}} \\ \mathsf{serial}_{j+1}^B &= f_{sk_{B1}}(\mathsf{serial}_j^B) \end{aligned}$$

(iii) Finally, Bob sends the values $\mathsf{comm}_{\mathsf{Tx}}, \mathsf{serial}_i^A, \mathsf{state}_{i+1}^A, \mathsf{zkp}_{i+1}^A, \mathsf{serial}_j^B, \mathsf{state}_{j+1}^B, \mathsf{zkp}_{j+1}^B$ to the central Bank.

③ **Transaction Execution:**

(i) The central Bank checks that none of the serial numbers serial_i^A, serial_j^B appear in its stored set of previously used serial numbers and that both zero-knowledge proofs zkp_{i+1}^A, zkp_{j+1}^B verify. If this is not the case, then the central bank rejects the transaction and informs Bob.

(ii) Otherwise, the central bank accepts the transaction and adds both serial numbers to the set of previously used serial numbers, signs the new state commitments as $\sigma_{i+1}^A = \mathsf{Sign}(sk_C, \mathsf{state}_{i+1}^A)$ and $\sigma_{j+1}^B = \mathsf{Sign}(sk_C, \mathsf{state}_{j+1}^B)$ and sends them to Bob. In addition, the central bank publishes the transaction (i.e., all values received from Bob plus σ_{i+1}^A and σ_{j+1}^B) on a publicly available log.

④ **Payment Acceptance:** Bob checks that the signatures received from the central bank are valid, accepts the payment and stores σ_{j+1}^B to update his account if this is the case, and forwards σ_{i+1}^A to Alice. Otherwise, he rejects the payment and informs Alice.

⑤ **Payment Completion:** Alice checks that the signature received from Bob is valid. Otherwise, or if she has not received a signature from Bob after a timeout, she inspects the central bank's public transaction log to retrieve the transaction and the signature on her new account state commitment. She then stores σ_{i+1}^A to update her account and the payment is completed.

4 REGULATION IN PLATYPUS

As described in Section 2.1, a CBDC requires the possibility to enforce regulatory policies. In particular, a CBDC should enable rules that ensure the financial stability of a system, e.g., to prevent bank runs, as well as rules that allow enforcement of anti-money-laundering legislation or allow the detection of tax evasion [5, 8].

The design of Platypus explicitly simplifies the implementation of such compliance policies through its account-based design. This account-based design allows storing additional information within an account state, which enables efficient zero-knowledge proofs through which the account holder can prove compliance with a given rule. In particular, it improves efficiency over previous designs such as that of Garman et al. [23] that require proofs over the state of the whole system (inclusion of several UTXO in a Merkle tree) instead of a proof of a signature. In contrast to the design by Garman et al., which only allows proofs on the state of the sender, it also allows proofs about the state of the recipient. In addition, it improves privacy compared to designs like PRCash [42] that require linking several transactions together for efficiency.

In this section, we describe a generic framework for enabling such regulatory policies. In Appendix A, we describe in detail two examples for such policies that are in line with the goals of a CBDC as stated by several central banks [8]. Namely, the first example puts limits on how much currency a user can hold without declaring it to authorities. The second example limits how much currency a user can receive anonymously within a given time period. The goals of this second example are similar to that of the *anonymity vouchers* as proposed by the European Central Bank [20] as well as the previous proposals by Garman et al. [23] and Wüst et al. [42]. In Appendix A, we also show that our regulation mechanism ensures compliance with the enforced policies.

4.1 Regulation Framework

Meaningful compliance rules need to be bound to a recognized identity. Otherwise, a user could establish a large number of pseudonymous identities to circumvent these rules. This requires an entity responsible for establishing these identities.

In addition, many practical rules do not simply prevent someone from taking an action but instead require them to disclose information under certain conditions. We therefore also assume the existence of a government agency that is responsible for receiving such information and operating on it, e.g., within the legal system. For simplicity, we assume that these roles are taken on by a single entity that we call the *regulator*. However, in practice, the responsibilities could be split, e.g., one entity could be responsible for establishing identities and a separate agency could hold the responsibility for each compliance rule. As part of the system setup, the regulator creates one key pair for issuing certificates (sk_{RC}, pk_{RC}), another key pair that is used for encryption (sk_{RE}, pk_{RE}), and publishes both public keys.

Enrollment. To enable regulation, users need to explicitly enroll in the system and establish identities. To do this, and to later be able to prove their identity, each user U generates a random secret value, called *secret identity* si_U from which their *public identity* $pi_U = \mathsf{PubID}(si_U)$ is derived, i.e., essentially a private/public key pair with the sole purpose of identifying the user. The user then needs to receive a *certificate* $\sigma_R^U = \mathsf{Sign}(sk_{RC}, (pi_U, params))$, i.e., a signature from the regulator on the user's public identity, as well as potentially some other individual parameters ($params$) that can be used for different rules on an individual basis. For example, the certificate could contain a holding limit that is individual to each user. This can be useful to, e.g., allow retail businesses to hold a larger amount of currency than users can hold in private accounts.

To issue this certificate, the user proves knowledge of the secret identity corresponding to their public identity pi_U, which is then, together with the individual regulation parameters $params$, signed by the regulator after confirming the real identity of the user (e.g., by the user physically going to an office of the responsible government agency). The user's certificate and secret identity can then later be used in zero-knowledge proofs for anonymous identification. To ensure that this identity cannot be used for multiple accounts, the public identity is always included in the account state commitment and the user proves equality of the public identities committed to in the old and new state commitments.

Structure of regulated transactions. For the enforcement of some regulatory rules, it can be useful to keep track of information involving the user's transaction history, which then allows the user to create proofs involving this information when creating a transaction. To enable this, such *auxiliary information* can be committed to in the account state commitment of the user. The account state of a user U is thus represented by a commitment $\mathsf{state}_i^U = \mathsf{comm}(\mathsf{serial}_i^U, \mathsf{bal}_i^U, pi_U, \mathsf{aux}_i^U, \mathsf{blind}_i^U)$ where aux_i^U denotes the required auxiliary information.

The zero-knowledge proof of the base protocol (see Section 3), is then extended such that the user also proves in zero-knowledge that they comply with the regulation rules. This includes proving that they know a private identity for which they have a certificate from the regulator. The regulator can require the user to disclose some information (e.g., the user's identity, balance, transaction value etc.) under certain specified conditions. We denote the computation of this information with a function $\mathsf{RegInfo}$ which takes the user's state information as input and either outputs either a fixed dummy value (if the condition is not triggered) or the information that is required to be disclosed (if the condition is triggered), e.g., the user's identity pi_U and account balance bal_{i+1}^U.

The user encrypts the output of this function with the regulator's public encryption key pk_{RE}, resulting in a ciphertext E_{i+1}^U and proves that the computation of $\mathsf{RegInfo}$ and the encryption was correctly performed. Encrypting a dummy value if the transaction is fully compliant ensures that all transactions are indistinguishable to parties other than the regulator independent of triggering the condition. The ciphertext E_{i+1}^U is sent to the bank as part of the transaction, which forwards it to the regulator, who can then decrypt it (and discard it, if it is the dummy value).

Below, we show the general structure of the updated proof statement for regulated transactions. The function $\mathsf{updateAux}$ is used to update the auxiliary information aux and the function $\mathsf{checkOther}$ is a predicate that can contain additional checks that would cause the generation of the proof to fail. E.g., this could be used to impose hard limits on the amount that a user can hold in their account (see Appendix A). These functions, as well as $\mathsf{RegInfo}$ are dependent on the policies that are enforced. All of these functions can also depend on additional public information aux_{pub} such as the current date. To improve readability, the differences to the proof of the base transaction (see Section 3.2) are shown in purple.

Given public values

$$\mathsf{serial}_i^U, \mathsf{comm}_{\mathsf{Tx}}, \mathsf{state}_{i+1}^U, E_{i+1}^U, \mathsf{bal}_{\max}, pk_C, pk_{RC}, pk_{RE}, \mathsf{aux}_{pub}$$

I know secret values

$$si_U, pi_U, \sigma_R^U, params, \mathsf{aux}_i^U, \mathsf{aux}_{i+1}^U$$
$$sk_{U1}, \mathsf{bal}_i^U, \mathsf{bal}_{i+1}^U, \mathsf{blind}_i^U, \sigma_i^U, v_{\mathsf{Tx}}, \mathsf{blind}_{\mathsf{Tx}}, \mathsf{serial}_{i+1}^U, \mathsf{blind}_{i+1}^U$$

such that

$$\begin{aligned}
\mathsf{True} &= \mathsf{Vrfy}(pk_C, \mathsf{comm}(\mathsf{serial}_i^U, \mathsf{bal}_i^U, pi_U, \mathsf{aux}_i^U, \mathsf{blind}_i^U), \sigma_i^U) \\
\mathsf{comm}_{\mathsf{Tx}} &= \mathsf{comm}(v_{\mathsf{Tx}}, \mathsf{blind}_{\mathsf{Tx}}) \\
\mathsf{state}_{i+1}^U &= \mathsf{comm}(\mathsf{serial}_{i+1}^U, \mathsf{bal}_{i+1}^U, pi_U, \mathsf{aux}_{i+1}^U, \mathsf{blind}_{i+1}^U) \\
\mathsf{bal}_{\max} &\geq \mathsf{bal}_{i+1}^U \\
\mathsf{bal}_{i+1}^U &= \mathsf{bal}_i^U + v_{\mathsf{Tx}} \\
\mathsf{serial}_{i+1}^U &= f_{sk_{U1}}(\mathsf{serial}_i^U) \\
pi_U &= \mathsf{PubID}(si_U)
\end{aligned}$$

$$\begin{aligned}
\mathsf{True} &= \mathsf{Vrfy}(pk_{RC}, (pi_U, params), \sigma_R^U)\\
\mathsf{aux}_{i+1}^U &= \mathsf{updateAux}(\mathsf{aux}_i^U, \mathsf{aux}_{pub}, pi_U, params, v_{\mathsf{Tx}}, \mathsf{bal}_{i+1}^U)\\
E_{i+1}^U &= \mathsf{Enc}(pk_{RE}, \mathsf{RegInfo}(\mathsf{aux}_i^U, \mathsf{aux}_{pub}, pi_U, params, v_{\mathsf{Tx}}, \mathsf{bal}_{i+1}^U))\\
\mathsf{True} &= \mathsf{checkOther}(\mathsf{aux}_i^U, \mathsf{aux}_{pub}, pi_U, params, v_{\mathsf{Tx}}, \mathsf{bal}_{i+1}^U)
\end{aligned}$$

Depending on the type of compliance rule in place, not all parts are necessary. For example, a holding limit (see Appendix A) does not require committing to any auxiliary data aux_i^U. We show the proof statement for a recipient here, but this can be equally applied to the sender (with the only difference being an increase vs. decrease of the balance). Two example policies that illustrate regulation proofs are discussed in Appendix A.

5 SECURITY ANALYSIS

In this section, we analyze the security of Platypus, in particular its integrity and privacy guarantees.

5.1 Transaction Integrity

We first discuss the integrity of our system. Since Platypus is a digital currency system, this entails that only authorized parties should be able to spend funds or create funds and funds should not be spendable more than once. In particular, the system should provide *transaction unforgeability* and *balance invariance*. We define these two properties below and show that our system provides them.

Transaction unforgeability essentially ensures that only authorized parties can create transactions that spend their respective funds and that the transaction values and intended recipients cannot be changed by an adversary. Balance invariance ensures that an adversary cannot spend funds multiple times or increase the supply of the currency. We capture the first of these properties with the following *transaction forgery* game:

Definition 5.1 (Transaction Forgery Game)**.** Given our system, the game consists of an interaction between an adversary $\mathcal{A}$ and a challenger C with access to an oracle O that simulates honest parties in the system. The game proceeds as follows:

(1) C initializes the system with a security parameter λ, which is used by the system to in turn initialize all used primitives, such as the signature scheme or the zero-knowledge proof system. C also initializes the oracle O.
(2) $\mathcal{A}$ can then generate arbitrary private keys and associated accounts with a balance chosen by $\mathcal{A}$, which O enrolls in the system by signing the associated account state commitments.
(3) $\mathcal{A}$ can also ask O to initialize additional clients with balances chosen by $\mathcal{A}$. O initializes them with the specified balance by signing an according account state commitment and then sends the signed account state commitment and serial number for each of them to the adversary.
(4) $\mathcal{A}$ can use his accounts to create arbitrary transactions, interact arbitrarily (i.e. send or receive transactions) with any account managed by the oracle, or can ask the oracle to create transactions between accounts managed by the oracle which are created and forwarded to the adversary. All transactions created in interaction with O are added to a query set Q.
(5) For each of these transactions, the adversary can then decide to submit them to O for execution, where O acts as central bank, performs the same checks as the central bank and either accepts or rejects the transaction.
(6) The adversary wins the game if they can create a transaction that is accepted by the oracle (simulating the central bank) in the *transaction execution step* that does not appear in the query set Q and is either
   - a transaction in which $\mathcal{A}$ controls neither the sender nor the recipient account
   - a transaction in which $\mathcal{A}$ controls the recipient account, but not the sender account and no transaction with the same sender serial number and the same transaction value, and for which the adversary controls the recipient account, exists in Q

Claim 5.1 (Transaction Unforgeability)**.** *No computationally bounded adversary $\mathcal{A}$ without access to the simulation trapdoor of the zero-knowledge proof system can win the transaction forgery game with non-negligible probability.*

Proof Sketch. Assume such an adversary $\mathcal{A}$ exists. Then there are two possible cases to distinguish: Either 1) the adversary forges a valid account update for the sender that is not part of any transaction in Q, or 2) he reuses a valid sender account update from a transaction $\mathsf{Tx}_Q \in Q$.

In Case 1, $\mathcal{A}$ either a) creates a valid account update for an account not controlled by $\mathcal{A}$ without knowing the respective secret values, b) gains knowledge of the secret values, or c) creates a valid account update for a non-existing account.

In case 1a), $\mathcal{A}$ must be able to create a zero-knowledge proof that is accepted by the central bank without knowing the secrets, thus violating our assumption that the zero-knowledge proof system is sound. In 1b) $\mathcal{A}$ must be able to compute the sender's secret values based on previously seen transactions, in particular also the blinding value used to create the previous account state. Since this blinding value is only used for the account state commitment, which is never opened, such an adversary could be used to distinguish commitments to two different pairs of serial numbers and account balances, which violates our assumption that the commitment scheme is hiding.

In case 1c), $\mathcal{A}$ either needs to produce a signature from the central bank on a forged account state commitment or they need to produce a proof of knowledge of such a signature without having knowledge of it. If $\mathcal{A}$ can produce either of them, then this adversary $\mathcal{A}$ can also be used to either break soundness of the zero-knowledge proof system or to win the signature forgery game, which violates our assumptions.

Now consider case 2. Then $\mathcal{A}$ either a) does not control the recipient account for the transaction Tx_Q from Q, or b) controls the recipient account for Tx_Q. In case 2a) $\mathcal{A}$ does not know the blinding value used to create the transaction commitment and needs to either find a transaction $\mathsf{Tx}'_Q \in Q$ for which the transaction commitment is the same as in Tx_Q (to reuse its recipient state update) which is negligible, or $\mathcal{A}$ needs to create a recipient account update that uses the transaction commitment from Tx_Q which is analogous to case 1.

In case 2b) $\mathcal{A}$ controls the recipient account of Tx_Q and therefore needs to create a transaction Tx' with a different recipient account update that changes the transaction value. In this case, the adversary knows the blinding value used to create the transaction commitment since he controls the recipient account used in Tx_Q. However, since the commitment scheme is binding, $\mathcal{A}$ cannot open the commitment to any value other than the originally committed value, and since we assume the proof system to be sound, $\mathcal{A}$ can therefore not create any recipient account update that changes the recipient's balance by any other value. Thus, $\mathcal{A}$ cannot create any such transaction Tx' without violating either the binding property of the commitment scheme or soundness of the proof system.

Since all possible cases violate at least one assumption, Platypus provides transaction unforgeability. □

Claim 5.2 (Balance Invariance)**.** *No computationally bounded adversary without access to the simulation trapdoor of the zero-knowledge proof system can create a transaction that increases the available funds in the system or spends funds more than once.*

Proof Sketch. There are multiple cases to distinguish. An adversary can either 1) attempt to use the same sender account state in multiple transactions, 2) attempt to use a sender account state that never resulted from a transaction accepted by the central bank, or 3) attempt to create a transaction that increases the balance of the recipient by more than it decreases the balance of the sender.

First, let us consider the case where an adversary attempts to use the same account state multiple times as sender in a transaction. Similar to traditional e-cash schemes like [14] as well as Zerocash [38], double spending is prevented using serial numbers that uniquely define an account state and can only be used once. Once the serial number serial_i^A has been revealed for one account state commitment state_i^A, the same account state can no longer be used for future updates, since reusing the account state would require proving that the same account state commitment opens to a different serial number $\mathsf{serial'}_i^A$. If the adversary can create such a proof, then either the proof system is not sound or the commitment scheme used to create the state commitment is not binding, both of which contradict our assumptions. No client can therefore use the same account state for more than one transaction.

Now consider the case where an adversary creates a transaction that uses a sender account state that has never been the result of a transaction accepted by the central bank. This would immediately allow the adversary to win the transaction forgeability game and thus violates at least one of our assumptions.

Lastly, consider the case where an adversary attempts to create a transaction that increases the account balance of the receiver by more than the value subtracted from the account balance of the sender. Since the value of each transaction is committed to using the transaction commitment $\mathsf{comm}_{\mathsf{Tx}}$, which is created using a hiding and binding commitment scheme, no computationally bounded party can open the commitment to a transaction value other than what was committed to originally. Since the proof of the transaction sender proves that their account balance was decreased by exactly the committed value and the proof of the transaction recipient proves that their balance was increased by exactly this value, any adversary that could increase the recipients balance by a different value could be used to either break soundness of the zero-knowledge proof system or to break the binding property of the commitment scheme. Thus, the account balance of the recipient is increased by exactly the amount that the balance of the sender is decreased and the transaction does not increase the total amount of funds available in the system. □

5.2 Transaction Privacy

Here, we consider the privacy guarantees provided by Platypus. In particular, we consider privacy towards parties other than the regulator and show that accepted transactions in our system are indistinguishable. In Appendix A we discuss what additional information the regulator receives. We do not consider network-level attacks on anonymity here, as they are out of scope of this paper, but we provide a short discussion of such attacks in Appendix B.

We capture the privacy guarantees with the following *transaction indistinguishability* game:

Definition 5.2 (Transaction Indistinguishability Game)**.** Given our system, the game consists of an interaction between an adversary $\mathcal{A}$ and a challenger C with access to an oracle O that simulates honest parties in the system. The game proceeds as follows:

(1) C initializes the system with a security parameter λ, which is used by the system to in turn initialize all used primitives, such as the signature scheme or the zero-knowledge proof system. C also initializes the oracle O.
(2) $\mathcal{A}$ can then generate arbitrary private keys and associated accounts with a balance chosen by $\mathcal{A}$, which O enrolls in the system by signing the associated account state commitments.
(3) $\mathcal{A}$ can also ask O to initialize additional clients with balances chosen by $\mathcal{A}$. O initializes them with the specified balance by signing an according account state commitment and then sends the state commitment and serial number for each of them to the adversary.
(4) $\mathcal{A}$ can use his accounts to create arbitrary transactions, interact arbitrarily (i.e. send or receive transactions) with any account managed by the oracle, or can ask the oracle to create transactions between accounts managed by the oracle which are created and if they result in a valid transaction, they are executed (i.e. the states of the involved parties are updated) and forwarded to the adversary.
(5) In the challenge phase, $\mathcal{A}$ chooses parameters (i.e. sender, recipient, value) for two transactions Tx_0 and Tx_1, such that the adversary controls neither the sender nor the recipient account and the transaction value does not exceed the sender's balance and sends these parameters to C.
(6) C chooses a bit $b \in \{0, 1\}$ u.a.r., executes the transaction Tx_b and sends the resulting transaction to $\mathcal{A}$
(7) $\mathcal{A}$ then outputs a bit b' and wins the game if $b = b'$

Claim 5.3 (Transaction Indistinguishability)**.** *No computationally bounded adversary $\mathcal{A}$ can win the transaction indistinguishability game with non-negligible advantage.*

Proof Sketch. As stated in Section 2.2, we assume that all used cryptographic primitives are secure according to their respective notions. In particular this includes that the pseudorandom function is indistinguishable from a truly random function, the commitments to different values are indistinguishable, the zero-knowledge proof system provides zero-knowledge (i.e. we have access to a simulation oracle $\mathcal{S}$ that can simulate indistinguishable proofs for any statement), and that the encryption scheme provides CPA-indistinguishability.

We now show that no efficient adversary $\mathcal{A}$ can succeed in winning the game with non-negligible advantage using a hybrid argument. To that end consider two set of distributions $T_0^0, T_0^1, \dots, T_0^9$ and $T_1^0, T_1^1, \dots, T_1^9$ for the challenge transactions Tx_0 and Tx_1, respectively in which we gradually replace fields in the transactions through an idealized version. That is, T_k^0 (for $k \in \{0, 1\}$) is the distribution for the real transaction Tx_k, T_k^1 replaces the sender zero-knowledge proof zkp_{i+1}^A with a simulated proof (from $\mathcal{S}$), T_k^2 additionally replaces the sender serial number serial_i^A with the output of a truly random function, T_k^3 also replaces the sender's state commitment state_{i+1}^A with a commitment to randomly chosen account parameters, and T_k^4 replaces the encrypted regulation information E_{i+1}^A with the encryption of a random value. The same is repeated for the recipient's part of the transactions for the distributions $T_k^5, \dots, T_k^8$, and finally T_k^9 also replaces the transaction commitment c_{Tx} with a commitment to a random value.

T_0^9 and T_1^9 are therefore distributions in which all fields in the transaction have been replaced with random values (sampled according to the distribution resulting from truly random inputs to the respective functions) and the zero-knowledge proofs are simulated based on these random values. A special case is the serial number, which is replaced by the output of a truly random function with a previous serial number as input. However, since all previous serial numbers are unique for transactions accepted by the central bank, the output is also truly random. Therefore T_0^9 and T_1^9 are the same distributions and thus indistinguishable for any adversary.

Assume that we have an arbitrary adversary $\mathcal{A}$ that wins our game with non-negligible advantage, i.e., that can successfully distinguish T_0^0 and T_1^0. Thus, for some non-negligible function p, we have $\left|\Pr\left[\mathcal{A}(T_0^0) = 1\right] - \Pr\left[\mathcal{A}(T_1^0) = 1\right]\right| \geq p(\lambda)$. Due to the triangle inequality, we also have:

$$\begin{aligned}
&\left|\Pr\left[\mathcal{A}(T_0^0) = 1\right] - \Pr\left[\mathcal{A}(T_1^0) = 1\right]\right| \\
&\leq \sum_{i=1}^{9} \left|\Pr\left[\mathcal{A}(T_0^{i-1}) = 1\right] - \Pr\left[\mathcal{A}(T_0^i) = 1\right]\right| \\
&+ \sum_{i=1}^{9} \left|\Pr\left[\mathcal{A}(T_1^{i-1}) = 1\right] - \Pr\left[\mathcal{A}(T_1^i) = 1\right]\right| \\
&+ \left|\Pr\left[\mathcal{A}(T_0^9) = 1\right] - \Pr\left[\mathcal{A}(T_1^9) = 1\right]\right|
\end{aligned}$$

Since the last term is zero (as T_0^9 and T_1^9 are the same distribution), at least one of the other terms must be non-negligible, i.e. $\left|\Pr\left[\mathcal{A}(T_k^{i-1}) = 1\right] - \Pr\left[\mathcal{A}(T_k^i) = 1\right]\right| \geq p'(\lambda)$ for some $i \in \{1, \dots, 9\}, k \in \{0, 1\}$ and some non-negligible function p'. Since the only difference between these two distributions is that one of them replaces one of the fields with a value that is indistinguishable (according to the respective notion of the used primitive), this leads to a contradiction. Therefore, Platypus provides transaction indistinguishability.

□

5.3 Availability of Funds

While we do not consider network-level attacks on availability, our system should ensure that a client cannot be prevented from using their funds by a third party. For example, Ruffing et al. [37] described an attack on Zerocoin [31], in which an attacker invalidates coins from another user by creating and immediately spending coins with the same serial number as that of an honest user, which prevents the honest user from using their funds. Since Platypus also uses serial numbers to prevent double-spending, we need to consider similar attacks. In particular, we make the following claim:

Claim 5.4. *No computationally bounded adversary can invalidate the account state of another client.*

Proof Sketch. First, note that in order to prevent a client from creating a transaction that updates their account state, either some information necessary to create the account state update needs to be withheld from the client, or the adversary needs to cause the central bank to reject the transaction. We assume that the client does not lose access to their long term keys and private information and thus they can always retrieve all necessary information from the central bank's transaction log.

Since the central bank will always accept a valid transaction unless it reuses a previously seen serial number, the adversary can only make the central bank reject an account update from a client by creating a transaction that uses the same serial number as used by the honest client (as in [37]).

To invalidate a user's account state with serial number serial_i^U, the adversary needs to create an account update that reveals the same serial number and they need to prove that this serial number was committed to in a valid state commitment for which they know the corresponding secret key. Thus, the adversary needs to create a series of account states that at some point results in the same serial number serial_i^U, i.e. they need to find a secret key sk' and an index j, such that $f_{sk'}^{\circ j}(0) = \mathsf{serial}_i^U = f_{sk_{U1}}^{\circ i}(0)$ (where $f_x^{\circ k}$ is the k-times iterated composition of f_x and k is bounded by an arbitrary but fixed value n (polynomial in the security parameter)).

Since f_x is a pseudorandom function, so is $h_{(x,k)} = f_x^{\circ k}$ for a randomly chosen key (x, k) where $k \in \mathbb{Z}_n^+$ (by induction). A successful adversary as described above would therefore need to find a key for the pseudorandom function family h that produces the given input/output pair which is infeasible. □

6 EVALUATION

6.1 Implementation

We implemented Platypus using the gnark[1] library for the zero-knowledge proofs with the BN256 curve and the Groth16 proof system [26]. Our implementation covers benchmarks for the creation and verification of the zero-knowledge proofs, as well as a simple 'end-to-end' system to measure throughput.

For the signatures and commitments, we use the gadgets as provided by the library for EdDSA [10] signatures and MiMC [4] hashes. Our prototype also uses MiMC for the pseudorandom function to generate serial numbers. Blinding values for commitments are randomly chosen in our prototype. To provide public key encryption for our regulation mechanism, our implementation uses Elgamal encryption [18]. Our implementation covers the base transaction as well as transactions with two regulation enforcement rules. These rules can be toggled individually and put limits on the amount of money that can be received within a given time interval or held in

Table 1: Performance of Platypus. This table shows proving and verification, as well as the time required for the trusted setup and the number of R1CS constraints. All measurements are averaged over 100 runs and rounded to two significant figures.

	Trusted Setup [s]	Proving [s]	Proving iPhone [s]	Verification [s]	# R1CS constraints	Tx Size [B]
Base Tx	0.73	0.11	0.19	0.000 89	11 728	672
Tx with holding limit	2.8	0.37	0.69	0.000 93	47 356	800
Tx with receiving limit	2.9	0.37	0.69	0.000 94	48 631	800
Tx with both limits	3.4	0.43	0.80	0.000 92	61 344	864

the account before the user is required to report this information to the regulator (see Section 4 and Appendix A). The proof generation and verification in the gnark library is parallelized.

Throughput Benchmark. Our throughput benchmark consists of a simple (non-optimized) server and a client that generates the full transaction, i.e., simulates both sender and recipient. The *server* exposes the functionality of the central bank as a simple REST interface with JSON payloads. Serial numbers submitted by clients in valid transaction proofs are stored in a local SQLite [2] database.

The *client library* handles all client-side functionality, i.e., generation of keys and transaction proofs, account state management, and communication with the server via HTTP.

To measure the transaction throughput that the server can handle, we use our client library to first prepare a large number of transactions (10k for our measurement). For each transaction, the client generates and enrolls two accounts and then creates a transaction between these pairs of accounts. In the measurement phase, the client library then submits these transactions to the server and measures the time required for all submitted transactions to complete.

6.2 Results

We measured the proving and verification time of our implementation for the base transaction and regulated transactions with receiving and holding limits (see Appendix A for more detail on the policies). We also measured the time required for the trusted setup, which is a one-time operation only run during system setup. As can be seen in Table 1, this setup is quite fast, taking less than ten seconds for all configurations.

Table 1 shows the results of our measurements as well as the number of R1CS constraints for our zero-knowledge proofs and the transaction sizes. R1CS is an intermediate format, used in many zero-knowledge proof systems, which represents the proof constraints and thus provides a system-independent measure of the proof complexity. We performed these measurements on a machine with an Intel® Core™ i7-7700 CPU (3.60GHz) with 4 cores and 16GB of RAM. We also measured proving time on an Apple iPhone 13 mini to benchmark performance on a mobile device. The results for proof generation and proof verification are provided per proof, i.e., both the sender and the recipient have to perform a proof generation and the central bank needs to perform two proof verifications per transaction. The client-side proof generation could be done in parallel after first communicating the values used to create the transaction commitment ($\mathsf{blind}_{\mathsf{Tx}}$ and v_{Tx}), i.e., the transaction sender and recipient can compute their proofs simultaneously to reduce the total transaction creation time. Our results show that this can be done efficiently. Transaction sizes are based on 256-bit serial numbers and commitments and show the size of the transaction before execution, i.e., when they are submitted to the central bank. After execution (in the log), they additionally include two signatures, thus, an additional 128 bytes.

With both regulation mechanisms in place, the proof generation takes 0.4 seconds on our test machine and 0.8 seconds on the iPhone (see Table 1). This makes it feasible to perform the complete transaction within one second, which is often considered an important limit for usability [33] and which makes it usable for retail payments. In addition, our numbers show that the overhead of adding additional regulation mechanisms is small. Concretely, while adding any regulation adds a significant overhead in the proving time compared to the base transaction, the additional impact of enforcing a second rule is small and only increases the proving time by a tenth of a second on the iPhone and a twentieth on our larger test machine (while verification time stays constant), which shows that Platypus can easily support enforcement of multiple regulatory rules.

The fast transaction verification is constant independent of the size of the proof statement (i.e., regardless of the regulation mechanisms in use) and allows a single machine to process a large number of transactions. In our throughput benchmark, our machine achieves a throughput of 922 transactions per second.

Even though CBDCs are not intended to replace all other forms of payments, only to complement them [8], it is interesting to consider the feasibility of such a system for all payments in a large economic area. Data from the European Union show that in 2016, the EU population performed 163 billion payments [19] for a population of just below 450 million people [22]. This corresponds to a volume of slightly more than five thousand transactions per second on average, or if we assume that all of these payments take place within only 8 hours of each day (to exclude times with a low transaction volume), a volume of about 15.5 thousand transactions per second.

Thus, to handle all transactions in the EU, a deployment of Platypus would require the equivalent of approximately 17 of our test machines (with its 922 transactions per second), which is a modest requirement for such a large economic area. Put differently, assuming the same transaction volume per person and again assuming that all transactions are concentrated on 8 hours per day, a single machine would be able to easily handle the transactions of a small country like Switzerland ($\approx$ 300 transactions per second), Israel ($\approx$ 320 transactions per second), or Sweden ($\approx$ 350 transactions per second).

7 RELATED WORK

E-Cash Systems. With e-cash [14], Chaum introduced the first design for an anonymous digital currency, in which a user can withdraw a coin from a bank by generating a coin identified by a serial number and receiving a blind signature on it, which ensures that the bank does not see the serial number. The user later unblinds this signature, which allows them to use the coin for payments. A merchant receiving a payment deposits the coin at the bank, at which point the bank checks if the serial number has already been used. If that is the case, the bank rejects the payment, otherwise it is accepted.

E-cash makes withdrawal and spending of a coin unlinkable, but it reveals to the bank the total transaction volume of a client (based on their withdrawals) and the value of each transaction for every merchant (based on their deposits). It also requires users to store information linear in the number of coins that they own. Later designs [11, 13] reduce the overhead. Camenisch et al. later also proposed an e-cash system that offers a form of regulation [12], limiting the amount that can be spent anonymously by a user per merchant. However, in all previous e-cash designs, the merchant still reveals the value of their received coins to the bank when depositing them.

Baldimtsi et al. [7] used techniques for double-spending detection for a transferable e-cash design, in which a coin can be transferred to different users without interaction with the bank. Once a coin gets deposited, the bank then checks for double-spending and identifies the offending party. This removes the issue that the merchant needs to reveal the transaction value to the bank for all received funds. Unfortunately, such a transferable e-cash scheme necessitates that coins grow in size depending on how often they were used, which makes spending less efficient than other e-cash schemes. This also affects linkability, since coins of a different size (i.e. coins that have been used a different number of times) are distinguishable.

Blockchain-based Systems. Several proposals for anonymous cryptocurrencies exist in the blockchain space. Zerocash [38] and its instantiation Zcash is currently considered to provide the strongest privacy guarantees. All of the transaction information is completely hidden and transactions are unlinkable, similar to the guarantees provided by Platypus. Garman et al. later showed how Zerocash can be extended with accountability mechanisms [23] that put restrictions on the transaction sender. One of the main drawbacks of Zerocash and the proposal by Garman et al. are the heavy client requirements which are difficult to remove or reduce in a decentralized setting [43]. This is particular due to the transaction receiving mechanism, which requires decrypting every transaction included in the blockchain as well as the requirement to prove knowledge of the path of a transaction output in a Merkle tree, which requires clients to keep this tree up to date. The second also makes scaling more difficult, since adding new transaction outputs to this tree requires all transactions to be serialized. In contrast, Platypus can take advantage of the changed trust assumptions to provide better scalability and to reduce the requirements for clients.

Other recent research has proposed schemes to provide regulation in a semi-centralized blockchain setting. PRCash [42] provides a design that uses lightweight zero-knowledge proofs to efficiently enable a receiving limit per time interval (*epoch*) for anonymous transactions. However, PRCash is based on a transaction design called mimblewimble [27] that does not provide full unlinkability for transactions and the regulation mechanism requires linking several transactions within an epoch. Platypus therefore provides better privacy and at the same time improves scalability through its centralized design. In addition, Platypus simplifies regulation compared to the designs of [23] and [42] due to its account-based design, since it does not require the inclusion of multiple UTXOs in proofs.

Parallel work by Androulaki et al. [6] proposed an auditable anonymous token management system for use in a permissioned blockchain targeted towards enterprise networks. In contrast to Platypus, which is account-based, their design uses a UTXO model, in which the UTXOs are represented as Pedersen commitments [35]. They then use a combination of a permissioned blockchain and a potentially distributed *certifier* to authorize payments. Transactions are committed to the blockchain after proving that the spender has a signature on each spent UTXO and later the newly created UTXOs get signed by the *certifier* using randomizable signatures [36]. In addition, the scheme allows for a set of *auditors*, each of which is responsible for auditing a different set of participants and which can access all information of their assigned participants. Platypus instead allows for fully anonymous transactions as long as specified conditions are not violated and is extendible with different regulatory rules. The regulation mechanisms enabled by Platypus make it more suitable for the use as central bank digital currency in which most transactions should be equivalent to cash with respect to their privacy properties [8]. In contrast, the auditability provided by the design from Androulaki et al. [6] is targeted at business-to-business usecases in which each business has their own auditor who should be able to access all of the transaction information of the business.

Another parallel design by Tomescu et al. [40] called UTT uses a similar base design to that of Androulaki et al. [6]. Namely, UTXOs (called coins) resulting from a transaction are represented as homomorphic commitments and signed with randomizable signatures [36] by a bank (that can be distributed using threshold cryptography). To later use these commitments as inputs, they are rerandomized, a nullifier (used to prevent double-spending) is deterministically computed and revealed by the sender and then the sender proves that the sum of the output coins is the same as the sum of the inputs. UTT also provides a monthly anonymity budget that limits how much money can be sent anonymously. Platypus, in contrast, enables more expressive regulation policies and can also enforce them on the side of the recipient. Similar to Zerocash [38], to receive a UTT payment, the recipient also has to scan all transactions on a ledger and perform a trial decryption for each. In addition, UTT transactions are 16× larger than ours and transaction verification time is significantly slower than in Platypus.

Finally, parallel work by Gross et al. [25] proposes the use of a modified Zerocash [38] for a “privacy pool” of a CBDC. Similar to Platypus, it replaces UTXOs with accounts, but in contrast to the e-cash style transaction execution of Platypus, it requires proofs of inclusion in a Merkle tree (like Zcash). In addition, regulation in [25] only allows hard limits (per transaction or for the account balance) and is thus less expressive and versatile than in our system.

8 CONCLUSION

Despite the prominence of blockchain-based digital currencies, they may not be the best technology choice for issuing a CBDC. Given the trust model of CBDCs (central authority) and the desirable features of a CBDC (privacy, performance, scalability, regulation), we argue that a traditional e-cash scheme can be a more suitable starting point for designing CBDCs. With our solution Platypus we have shown that an e-cash like system can provide all these features at the same time.

We have also proposed a new style of building digital currencies that combines e-cash style transaction processing with the account-model that is common in blockchain systems like Ethereum [41] and with privacy techniques inspired by Zerocash [38]. We hope that our work can inspire other researchers to design new e-cash solutions that leverage the design pattern proposed in this paper, extend our work, and ultimately provide better CBDC designs.

REFERENCES

[1] [n.d.]. gnark Library. https://github.com/ConsenSys/gnark. (Cited on p. 10)

[2] [n.d.]. SQLite. https://www.sqlite.org/. (Cited on p. 11)

[3] [n.d.]. Tor Browser. https://www.torproject.org/. (Cited on p. 3, 15)

[4] Martin Albrecht, Lorenzo Grassi, Christian Rechberger, Arnab Roy, and Tyge Tiessen. 2016. MiMC: Efficient encryption and cryptographic hashing with minimal multiplicative complexity. In *International Conference on the Theory and Application of Cryptology and Information Security.* (Cited on p. 10)

[5] Sarah Allen, Srdjan Čapkun, Ittay Eyal, Giulia Fanti, Bryan A Ford, James Grimmelmann, Ari Juels, Kari Kostiainen, Sarah Meiklejohn, Andrew Miller, Eswar Prasad, Karl Wüst, and Fan Zhang. 2020. *Design Choices for Central Bank Digital Currency: Policy and Technical Considerations.* Technical Report. The Brookings Institution. (Cited on p. 1, 2, 3, 6, 14)

[6] Elli Androulaki, Jan Camenisch, Angelo De Caro, Maria Dubovitskaya, Kaoutar Elkhiyaoui, and Björn Tackmann. 2020. Privacy-preserving auditable token payments in a permissioned blockchain system. In *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies.* (Cited on p. 12)

[7] Foteini Baldimtsi, Melissa Chase, Georg Fuchsbauer, and Markulf Kohlweiss. 2015. Anonymous Transferable E-Cash.. In *Public Key Cryptography.* (Cited on p. 12, 16)

[8] Bank of Canada, European Central Bank, Bank of Japan, Sveriges Riksbank, Swiss National Bank, Bank of England, Board of Governors of the Federal Reserve, and Bank for International Settlements. 2020. Central bank digital currencies: foundational principles and core features. https://www.bis.org/publ/othp33.htm. (Cited on p. 1, 2, 3, 6, 7, 11, 12, 14, 16)

[9] Bank of England. 2020. Central Bank Digital Currency: Opportunities, challenges and design. https://www.bankofengland.co.uk/-/media/boe/files/paper/2020/central-bank-digital-currency-opportunities-challenges-and-design.pdf. (Cited on p. 1, 2, 14)

[10] Daniel J Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. 2012. High-speed high-security signatures. *Journal of cryptographic engineering* 2, 2 (2012). (Cited on p. 10)

[11] Jan Camenisch, Susan Hohenberger, and Anna Lysyanskaya. 2005. Compact E-Cash. In *Advances in Cryptology - EUROCRYPT 2005 (Lecture Notes in Computer Science, Vol. 3494).* (Cited on p. 2, 12, 16)

[12] Jan Camenisch, Susan Hohenberger, and Anna Lysyanskaya. 2006. Balancing accountability and privacy using e-cash. In *International Conference on Security and Cryptography for Networks.* (Cited on p. 2, 12, 16)

[13] Sébastien Canard, David Pointcheval, Olivier Sanders, and Jacques Traoré. 2015. Divisible e-cash made practical. In *IACR International Workshop on Public Key Cryptography.* Springer, 77–100. (Cited on p. 2, 12)

[14] David Chaum. 1983. Blind Signatures for Untraceable Payments. In *Advances in Cryptology: Proceedings of Crypto 82.* (Cited on p. 1, 2, 9, 11, 16)

[15] D. Chaum, A. Fiat, and M. Naor. 1990. Untraceable Electronic Cash. In *Proceedings on Advances in Cryptology (CRYPTO '88).* (Cited on p. 16)

[16] David Chaum, Christian Grothoff, and Thomas Moser. 2021. How to issue a central bank digital currency. *SNB Working Papers* (2021). (Cited on p. 1, 3, 16)

[17] George Danezis and Sarah Meiklejohn. 2016. Centrally Banked Cryptocurrencies. In *23nd Annual Network and Distributed System Security Symposium, NDSS.* (Cited on p. 1)

[18] T. Elgamal. 1985. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory* 31, 4 (1985). (Cited on p. 10)

[19] Henk Esselink and Lola Hernández. 2017. The use of cash by households in the euro area. *ECB Occasional Paper* 201 (2017). (Cited on p. 11)

[20] European Central Bank. 2019. Exploring anonymity in central bank digital currencies. https://www.ecb.europa.eu/paym/intro/publications/pdf/ecb.mipinfocus191217.en.pdf. (Cited on p. 3, 7, 14)

[21] European Central Bank. 2021. Eurosystem report on the public consultation on a digital euro. https://www.ecb.europa.eu/pub/pdf/other/Eurosystem_report_on_the_public_consultation_on_a_digital_euro~539fa8cd8d.en.pdf. (Cited on p. 3)

[22] eurostat. [n.d.]. Population Development and Projections. https://ec.europa.eu/eurostat/web/population-demography-migration-projections/visualisations (retrieved 2021-04-10). (Cited on p. 11)

[23] Christina Garman, Matthew Green, and Ian Miers. 2016. Accountable privacy for decentralized anonymous payments. In *International Conference on Financial Cryptography and Data Security.* (Cited on p. 1, 2, 3, 6, 7, 12, 14)

[24] Arthur Gervais, Srdjan Capkun, Ghassan O Karame, and Damian Gruber. 2014. On the privacy provisions of bloom filters in lightweight bitcoin clients. In *Proceedings of the 30th Annual Computer Security Applications Conference.* (Cited on p. 1)

[25] Jonas Gross, Johannes Sedlmeir, Matthias Babel, Alexander Bechtel, and Benjamin Schellinger. 2021. Designing a central bank digital currency with support for cash-like privacy. *Available at SSRN 3891121* (2021). (Cited on p. 12)

[26] Jens Groth. 2016. On the size of pairing-based non-interactive arguments. In *Annual international conference on the theory and applications of cryptographic techniques.* (Cited on p. 4, 5, 10)

[27] Tom Elvis Jedusor. [n.d.]. Mimblewimble. http://mimblewimble.org/mimblewimble.txt. (Cited on p. 12)

[28] Butler Lampson and Howard E Sturgis. 1979. Crash recovery in a distributed data storage system. (1979). (Cited on p. 2, 15)

[29] Zeyu Liu and Eran Tromer. 2021. Oblivious Message Retrieval. *Cryptology ePrint Archive* (2021). (Cited on p. 1, 16)

[30] Sinisa Matetic, Karl Wüst, Moritz Schneider, Kari Kostiainen, Ghassan Karame, and Srdjan Capkun. 2019. BITE: Bitcoin Lightweight Client Privacy using Trusted Execution. In *28th USENIX Security Symposium (USENIX Security 19).* 783–800. (Cited on p. 1)

[31] Ian Miers, Christina Garman, Matthew Green, and Aviel D Rubin. 2013. Zerocoin: Anonymous distributed e-cash from bitcoin. In *2013 IEEE Symposium on Security and Privacy.* (Cited on p. 10)

[32] Satoshi Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system. (2008). (Cited on p. 1)

[33] Jakob Nielsen. 1994. *Usability engineering.* Morgan Kaufmann. (Cited on p. 11)

[34] Morgen Peck. 2016. The Crazy Security Behind the Birth of Zcash, the Inside Story. *IEEE Spectrum* (2016). https://spectrum.ieee.org/tech-talk/computing/networks/the-crazy-security-behind-the-birth-of-zcash (Cited on p. 4)

[35] Torben Pryds Pedersen. 1992. Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing. In *Advances in Cryptology — CRYPTO '91.* (Cited on p. 12)

[36] David Pointcheval and Olivier Sanders. 2016. Short randomizable signatures. In *Cryptographers' Track at the RSA Conference.* (Cited on p. 12)

[37] Tim Ruffing, Sri Aravinda Thyagarajan, Viktoria Ronge, and Dominique Schroder. 2018. (Short Paper) Burning Zerocoins for Fun and for Profit-A Cryptographic Denial-of-Spending Attack on the Zerocoin Protocol. In *2018 Crypto Valley Conference on Blockchain Technology (CVCBT).* (Cited on p. 5, 10)

[38] Eli Ben Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. 2014. Zerocash: Decentralized anonymous payments from bitcoin. In *Security and Privacy (SP), 2014 IEEE Symposium on.* (Cited on p. 1, 2, 3, 9, 12, 13)

[39] Sveriges Riksbank. 2020. The Riksbank's e-krona pilot. https://www.riksbank.se/globalassets/media/rapporter/e-krona/2019/the-riksbanks-e-krona-pilot.pdf. (Cited on p. 1)

[40] Alin Tomescu, Adithya Bhat, Benny Applebaum, Ittai Abraham, Guy Gueta, Benny Pinkas, and Avishay Yanai. 2022. UTT: Decentralized Ecash with Accountable Privacy. *Cryptology ePrint Archive* (2022). (Cited on p. 12)

[41] Gavin Wood. 2014. Ethereum: A secure decentralised generalised transaction ledger. (2014). (Cited on p. 2, 13)

[42] Karl Wüst, Kari Kostiainen, Vedran Čapkun, and Srdjan Čapkun. 2019. PRCash: Fast, private and regulated transactions for digital currencies. In *International Conference on Financial Cryptography and Data Security.* (Cited on p. 1, 2, 3, 6, 7, 12, 14, 16)

[43] Karl Wüst, Sinisa Matetic, Moritz Schneider, Ian Miers, Kari Kostiainen, and Srdjan Čapkun. 2019. Zlite: Lightweight clients for shielded zcash transactions using trusted execution. In *International Conference on Financial Cryptography and Data Security.* (Cited on p. 1, 12, 16)

[44] Wolfie Zhao. [n.d.]. Chinese State-Owned Bank Offers Test Interface for PBoC Central Bank Digital Currency. *Coindesk.* 2020. https://www.coindesk.com/chinese-state-owned-bank-offers-test-interface-for-pboc-central-bank-digital-currency (retrieved 2021-04-14). (Cited on p. 1)

A REGULATION DETAILS

In this appendix, we provide descriptions of two example regulation policies, namely limits on how much money can be held in one account and how much money can be received within a given time period. We also show that our regulation mechanism ensures compliance with the policies that are in place.

A.1 Holding Limits

One compliance rule that is of particular interest for financial stability in an economic system, specifically to prevent bank runs, consists of limiting the amount of money that can be held in a CBDC [8, 9]. In addition, such a *holding limit* can be useful to authorities to prevent evasion of wealth tax.

A holding limit can be designed in different ways. The simplest way is to enforce a hard global limit on the amount that can be held by a single account. The only regulation mechanism required to enforce this is the establishment of real identities and proving the possession of a certificate. In addition to this, the value $\mathsf{bal}_{\mathsf{max}}$ that is used in the base transaction (see Section 3) and used to prevent overflows is set to the holding limit required by the regulatory rule which will prevent any balance from exceeding this limit.

A more flexible option could allow different holding limits for different users, for example to allow business accounts to hold more digital currency than private accounts. To do this, this individual holding limit lim_{hold} is included as part of the parameters $params$ in the user's certificate (see Section 4.1). In each transaction, the user then proves in zero-knowledge (i.e. without revealing the limit) that their new balance does not exceed this limit, i.e. the predicate $\mathsf{checkOther}$ checks that the user's new balance bal_{i+1}^U is less than the holding limit lim_{hold}. With such hard limits, there is no need to provide a ciphertext E_{i+1}^U with encrypted information for the regulator and the corresponding.

Lastly, it is possible to have soft limits instead of hard limits that allow holding a larger amount of currency with the requirement of revealing this information to the regulator. To enable this, the user's certificate again includes an individual holding limit as before, but the proof in the transaction changes. Instead of proving that they have not exceeded the limit in the transaction, the user encrypts their public identity and their account balance with the regulator's public key if they have exceeded the limit, or fixed dummy values otherwise. That is, the function $\mathsf{RegInfo}$ (see Section 4.1) returns the public identity of the user and their balance if the balance is above the limit lim_{hold} and the dummy value otherwise. This ciphertext E_{i+1}^U is added to the transaction and the user proves in zero-knowledge that they have either not exceeded the holding limit and encrypted the dummy value or that they have exceeded the limit and encrypted their public identity and their account balance.

Creating the proof in this way leaks no information to third parties, only to the regulator. The regulator can decrypt the encrypted information and disregard it if it contains the dummy values or keep it otherwise. However, to third parties all transactions are indistinguishable and they do not learn whether a transaction contains real information or dummy values.

A.2 Receiving Limits

Another example for a compliance rule that is commonly suggested for CBDCs is a limit on how much money can be received or spent by a party within a given amount of time [5, 8, 20, 23, 42]. Such a limit serves to emulate reporting requirements for cash transactions that are required for compliance with anti-money-laundering legislation or to prevent tax evasion. Since it is easy to quickly create a large number of digital transactions, these limits should cover a certain amount of time instead of only applying to a single transaction to ensure that they cannot be circumvented by simply splitting a large transaction into multiple smaller transactions.

In the following, we describe how such limits can be added for receiving currency, but the same techniques could also be directly applied for sending currency. Similar to the "anonymity vouchers" proposed by the european central bank [20] and proposed limits in previous work on blockchain-based digital currencies [23, 42], we focus on soft limits that allow for fully anonymous transactions if the total received value for each user is below a given threshold within a fixed time interval, but require reporting if the threshold is exceeded.

Similar to the previous example, the user enrolls in the system where they receive a certificate that includes a receiving limit lim_{rec} as part of the parameters $params$. The system additionally defines *epochs*, the time intervals for which the limits are defined. The length of these epochs is a parameter of the deployed system and can be arbitrary, e.g. a day, a week, or even a year, without affecting the linkability of transactions (in contrast to PRCash [42]). The current epoch number is part of the auxiliary public information aux_{pub}.

For each user, the account state includes two additional pieces of information in its auxiliary information aux_i^U, namely, the last epoch in which the user's account was updated and the cumulative sum of all funds that the user received within that epoch. With each transaction, the function $\mathsf{updateAux}$ updates this information accordingly.

Similar to the holding limits above, each transaction includes an encryption E_{i+1}^U (with the regulator's public key) of either the total received value in the current epoch and the recipient's identity – i.e., the function $\mathsf{RegInfo}$ (see Section 4.1) returns the public identity of the user and the cumulative epoch total – if the balance is above the limit lim_{rec} or dummy values otherwise. The user then proves (in zero-knowledge) that they performed this correctly, i.e. that the total value that they've received in the current epoch is below the limit or that their correct identity and the correct value were encrypted. Similar to the previous example, to third parties (including the central bank), all transactions remain indistinguishable.

A.3 Regulation Integrity

Since Platypus includes regulation mechanisms, we also need to consider the integrity of this mechanism. In particular, we make the following claim:

Claim A.1. *No client can create a transaction that is non-compliant with a regulation mechanism.*

Proof. This follows directly from the soundness of the zero-knowledge proof system. A transaction will only be valid if the transacting parties prove compliance with the regulatory rules that are in place. For example, if a receiving limit is in place, the recipient proves that either the received amount is within the limit or that the encrypted values attached to the transaction are correct encryptions of their identity and the received value with the public key of the regulator. Since the central bank will only sign updated account state commitments if the corresponding transaction is valid, and by our assumptions, the regulator trusts the central bank to verify this, no client can create a transaction that is non-compliant with the regulation mechanisms that are in place. □

A.4 Privacy towards the Regulator

For any transaction in which the client is not required by the regulation mechanism to include additional encrypted information, the regulator only receives dummy values from decrypting the fields storing regulatory information. Since the dummy values are fixed, the regulator does not gain any additional information from them and thus, these transaction are indistinguishable for the regulator (analogous to Section 5.2).

Of course, since this is the explicit goal of the regulation mechanism, the regulator can decrypt encrypted regulatory information included in a transaction and can thus distinguish them from other transactions and learn additional information about the client, their account and their account history, depending on what information the regulation mechanism requires.

B DISCUSSION

Network level attacks on privacy. As mentioned in Section 2.2, full protection against network level deanonymization attacks is out of scope for this paper. Nevertheless, we designed Platypus to provide some resilience against such attacks. In particular, the sender of the transaction communicates with the central bank through the recipient in the standard case, such that the central bank cannot link the sender and recipient based on the network connections. In exceptional cases, in which the recipient stops cooperating with the sender and does not return the signature on the sender's new account state, the sender can access the public transaction log, or a mirror of this log, to retrieve recent transactions.

While these two mechanisms provide some protection against simple deanonymization attempts, they do not fully protect against all adversaries, in particular if the adversary can see other traffic in the network. If a client is worried about such network level attacks, they can mitigate the risk by using anonymous communication networks such as Tor [3].

Backups and Account Recovery. The account state model that Platypus uses, requires users to have knowledge of their current account state. To enable efficient backups and account recovery, values such as the blinding value of the account state commitment or the serial number are pseudorandomly generated from the user's secret key. To create a backup, the user can simply store this secret key as well as their key and certificate used for regulation.

To recover the account from a backed up secret key, the user needs to retrieve their most recent account state. There are two possibilities to do this. As first option, the user can estimate a time interval in which their most recent transaction took place and retrieve all transactions from that time from the public transaction log. They can then use their secret key and generate serial numbers from it using the pseudorandom generator until they find one that matches a serial number from the log. The second option is that the user generates a list of potential serial numbers (pseudorandomly derived from their secret key), which they then use to query the public transaction log in binary search until they find the latest matching transaction.

The main drawback of the first option is that the user potentially needs to download a large amount of data, if they are unsure in which time interval their latest transaction took place. The drawback of the second option is that some of their transactions can potentially be linked if the central bank is monitoring and correlating queries to the transaction log. Since the number of transactions that could be linked with this approach is only logarithmic in the number of total transactions from the user and no other information about these transactions is revealed, it is unlikely that this presents an issue for most users in practice.

Once the user has retrieved their account state, they need to find their account balance and other values that their account state commits to (i.e. the values used for the regulation mechanism). Without these values they cannot create new transactions. The account balance is much smaller than the serial number and the blinding value and could therefore in principle be brute forced. However, this is inconvenient and can become infeasible if a significant amount of additional information (for the regulation mechanism) is also part of the account state. An easier solution is to add a *memo* field (similar to Zcash) in addition to each account state commitment as part of every transaction, which stores this information encrypted with a long-term key known to the user. The user can then simply include this key in their backup and use it to retrieve all relevant values when performing account recovery after retrieving it together with the account state commitment.

Sharding in Platypus. As mentioned in Section 2, the centralized and account-based design of Platypus simplifies sharding, as it enables the use of standard database sharding techniques.

Figure 3 shows an example of how transaction validation can be sharded. The verification of the zero-knowledge proofs can be performed in separate compute nodes independently from checking and updating the serial numbers of the used account states. While we show each shard here as one database node, each shard can, of course, also be replicated individually. Each database shard is assigned a specified subset of all serial numbers. For example, when using 4 shards, each shard could be assigned a quarter of all possible serial numbers based on the two most significant bits of the serial number. The compute nodes are independent of the transactions. When submitting a transaction, a client can connect to any compute node of the central bank (e.g. through a load balancer), which verifies the zero-knowledge proofs. If the proofs verify, the compute node checks in the database shards if the account states with the provided serial numbers have been invalidated already. Since the serial numbers are pseudorandom, most transactions will be cross-shard transactions if there are at least two database shards. However, since Platypus uses an account-based design, each transaction will never require more than two shards, one for checking the serial number of the sender and one for checking the serial number of the recipient. This is in contrast to UTXO-based systems in which an arbitrary number of shards could be involved in each transaction.

To check the serial numbers in the database shards, the compute node acts as a coordinator in a two-phase commit protocol [28] between the database shards. Each database shard checks if the serial number already exists in the database. If this is the case in one of the shards, the coordinator sends an abort to both shards. Otherwise, they both add the respective serial number to the set of used serial numbers and return a success to the compute node. Finally, the compute node signs the new account states, returns the signatures to the client and publishes the transaction on the public transaction log. Since the transaction log does not require

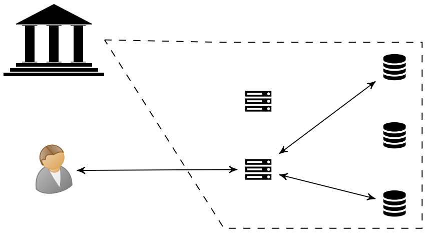

Figure 3: Sharding Potential in Platypus. The central bank can shard both computation and the storage of serial numbers internally. A client can connect to an arbitrary compute node (e.g. through a load balancer) which validates transactions independently from other compute nodes. The compute node then uses a two-phase commit to check and update serial numbers in the database shards corresponding to the serial numbers of the sender and recipient.

ordering, this step can be done concurrently by separate compute nodes without requiring any consensus protocol between them.

Offline Recipient. Most designs of blockchain-based cryptocurrencies allow a recipient to be offline when receiving funds. The sender only needs the recipient's public key to create a full transaction. One limitation of Platypus is that creating a transaction requires interaction between both participants, i.e. the recipient needs to be online to receive funds. This is similar to other e-cash schemes [7, 11, 12, 14, 15], in which the sender and recipient always need to interact. Involving the recipient in the transaction creation is necessary for two main reasons. First, it enables an account-based design with full anonymity. In an account-based system, without involvement of the recipient, some other party would need to be able to update the recipient's account and thus the account would be linkable to a public key of the recipient by that party.

Anonymous UTXO-based systems with offline receiving are possible, but require retrieving this information later, e.g. through downloading all transactions. This puts a heavy load on clients, who need to download and process all of this information. To reduce this load, clients currently either need to rely on trusted execution environments [43], private information retrieval protocols that are very expensive for the server [29], or accept reduced privacy guarantees by trusting a server to filter transactions for them. By directly communicating all necessary transaction information between the sender and recipient, this issue is side-stepped in the e-cash style transaction processing used by Platypus. This guarantees that privacy can be provided efficiently even with a large transaction volume.

Second, and most importantly, including the recipient in the transaction creation is a requirement for enabling regulatory rules affecting the recipient (similar to [42]) that allow full anonymity, even with respect to the regulator, as long as the transaction conforms to some constraints.

As an example, consider a simple holding limit (as described in Section 4) that puts a fixed limit $\mathsf{bal}_{\mathsf{max}}$ on the amount that each party can hold. Let us now assume that there is some mechanism that allows the central bank to check compliance with such a rule without violating any of the privacy properties and without interaction with the recipient. If a sender Alice now creates a transaction of value v_{Tx} with Bob as the recipient, there are two options, which both leak information about Bob's funds to Alice: Either the transaction is accepted, or it is rejected. In the first case, Alice knows that previous to the transaction, Bob owned less than $\mathsf{bal}_{\mathsf{max}} - v_{\mathsf{Tx}}$, in the second case, Alice now knows that Bob owned more than $\mathsf{bal}_{\mathsf{max}} - v_{\mathsf{Tx}}$.

We argue that not enabling offline receiving is a small drawback compared to the advantages of Platypus for our use cases. Recall that Platypus is intended as a "cash-like" CBDC, which is the main goal for many central banks [8, 16]. In such a setting, interaction between the recipient and the sender is the standard case, e.g. for credit card payments or for actual cash payments. Most payments are for retail payments, in which the device of the user interacts with a payment terminal or the user interacts with an online shop, or for peer-to-peer payments between friends, in which their devices can interact. Nevertheless, online receiving does not necessarily require the user to be active, but only their device. For example, if Alice wants to send some funds to Bob and Bob's device is not online, Alice can already initiate the transfer on her device. The device can then, without initiating the actual transaction at that point, contact Bob's device in the background until it becomes available. At that point, the device can initiate the actual transaction and then the payment can complete since both are online.