PRCash: Centrally-Issued Digital Currency with Privacy and Regulation

Karl Wüst
ETH Zurich, Switzerland
karl.wuest@inf.ethz.ch

Kari Kostiainen
ETH Zurich, Switzerland
kari.kostiainen@inf.ethz.ch

Vedran Čapkun
HEC Paris, France
capkun@hec.fr

Srdjan Čapkun
ETH Zurich, Switzerland
srdjan.capkun@inf.ethz.ch

ABSTRACT

Decentralized cryptocurrencies based on blockchains provide attractive features, including user privacy and system transparency, but lack active control of money supply and capabilities for regulatory oversight, both existing features of modern monetary systems. These limitations are critical, especially if the cryptocurrency is to replace, or complement, existing fiat currencies. Centralized cryptocurrencies, on the other hand, provide controlled supply of money, but lack transparency and transferability. Finally, they provide only limited privacy guarantees, as they do not offer recipient anonymity or payment value secrecy.

We propose a novel digital currency, called PRCash, where the control of money supply is centralized, money is represented as value-hiding transactions for transferability and improved privacy, and transactions are verified in a distributed manner and published to a public ledger for verifiability and transparency. Strong privacy and regulation are seemingly conflicting features, but we overcome this technical problem with a new regulation mechanism based on zero-knowledge proofs. Our implementation and evaluation shows that payments are fast and large-scale deployments practical. PRCash is the first digital currency to provide control of money supply, transparency, regulation, and privacy at the same time, and thus make its adoption as a fiat currency feasible.

1 INTRODUCTION

Over the last ten years, decentralized cryptocurrencies based on blockchains have gained significant attention. Currencies like Bitcoin [36] provide attractive new features compared to the currently widely-used payment methods, including improved *privacy*, as such currencies use pseudonyms instead of real identities. Decentralized cryptocurrencies also provide improvements in terms of *transparency*, because money is represented as transactions that are published on a ledger, and anyone can verify transaction correctness and creation of new money from the ledger.

However, such designs also lack important features. One limitation of currencies like Bitcoin is that they do not support *issuance control*. Instead, creation of new money is based on fixed rules and schedule. This makes it difficult for a central bank – whose main tasks include managing the currency and controlling money supply – to use such a cryptocurrency as fiat money. Another limitation of decentralized currencies is that they do not support *regulation*. Without regulatory oversight, money laundering and other criminal activities are difficult to prevent. These limitations present a major obstacle for the adoption of a cryptocurrency as fiat money.

Centralized digital currencies, like Chaum's original e-cash [19], represent money as coins. Such currencies provide their own benefits, including payer anonymity and control over money supply through a central issuer. However, they provide no transparency for the creation of money and no public verifiability for the correctness of transactions. Another drawback of centralized currencies is that money is typically not transferable and received coins need to be deposited immediately to the bank to prevent double spending. This imposes privacy limitations, in particular value secrecy and recipient anonymity are hard to achieve without transferability.

Our goals and solution. Our goal is to design a novel digital currency that combines the above discussed desirable properties. Our currency should (i) provide *issuance control* that enables adaptive monetary policy, (ii) support *transparency* over issuance of new money and *verifiability* of transaction correctness, (iii) enable *regulation* so that authorities can track the flow of larger sums of money, and (iv) guarantee *privacy* such that payments below a certain amount have strong anonymity.

In this paper, we design a novel digital currency called PRCash that can be seen as a hybrid of a centralized and decentralized system. In PRCash, the issuance of money is centralized, similar to traditional e-cash schemes, but money is represented as transactions that are verified in a distributed manner by a pre-defined set of validators and maintained on a public ledger as a permissioned blockchain. Such a design enables controlled issuance, provides transparency and verifiability, and money is transferable, which allows improved privacy. Our high-level design is similar to a previous solution, called RSCoin [23]. However, RSCoin lacks important features (namely, privacy and regulation) that we discuss next.

Strong anonymity and regulatory oversight are conflicting properties and known transaction techniques provide only one or the other. For example, transactions that use plaintext identities and amounts enable regulation but no privacy; usage of pseudonyms improves privacy, but makes regulation ineffective; novel transaction techniques like Confidential Transactions [34], Mimblewimble [30] and ZeroCash [40] provide strong privacy protection, but no regulation. In addition, systems like ZeroCash [40] (that provides the highest level of anonymity) are not efficient enough for a cash-like system in which transactions should be completed within seconds.

We address this conflict between privacy and regulation with a novel transaction creation and verification technique that leverages

zero-knowledge proofs. Using verifiable pseudorandom identifiers and range proofs, we limit the total amount of money that any user can receive anonymously within an epoch. We choose to control *receiving* of money, to mimic existing laws in many countries (e.g., in the US, received cash transactions exceeding $10,000 must be reported to the IRS), but our solution can be easily modified to limit *spending* as well. The user can choose for each payment if it should be made anonymous as long as he stays within the allowed limit, chosen by a regulatory authority. Anonymous transactions have strong privacy protection: payer anonymity, recipient anonymity and value secrecy. Regulation based on zero-knowledge proofs has been previously proposed for coin-based currencies by Camenisch et al. [14]. Our technique is novel in the sense that it enables regulation for (value-hiding) transactions.

We implemented a prototype of PRCash and evaluated its performance. Our currency can handle high transaction loads (e.g., 1000 tps) with modest computing infrastructure (e.g., 4×25 quad-core servers). Due to distributed validation, transaction verification has no single point of failure, and payment confirmation is fast (e.g., less than a second).

Usages. The primary use case that we consider is one where a central bank uses PRCash to issue digital money to complement, or replace, cash as the fiat currency. To the best of our knowledge, PRCash is the first digital currency to provide privacy, regulation, transparency and issuance control that are all important features for the usage as fiat money. The performance of our solution makes such usage feasible. This is in contrast to ZeroCash [40] style transactions that require minutes of computation to be created or decentralized currencies like Bitcoin that cannot support real-time payments or high transaction loads.

Adoption of PRCash as fiat currency could benefit the society in multiple ways. Individuals gain convenience, as they no longer have to carry cash, as well as improved privacy, especially for online payments, and new flexibility, as they can choose which payments are kept private. Businesses save on costs of handling cash and on card payment fees. Regulators would have more effective means of tracking money and expensive manual auditing processes could be automated. Finally, increased transparency could benefit all involved parties and increase trust in the system. Another interesting use case is privately-issued money, where the importance of transparency in even greater.

Contributions. To summarize, in this paper we make the following contributions:

- *Novel digital currency.* We propose PRCash, a novel digital currency that is the first to provide privacy, transparency, issuance control and regulation at the same time.
- *New techniques.* As part of the currency, we develop a new cryptographic technique for regulation of anonymous transactions based on zero-knowledge proofs.
- *Implementation and evaluation.* We show that fast, fault-tolerant, large-scale deployments are possible.

The rest of this paper is organized as follows. Section 2 gives an overview of our solution. Section 3 describes our currency in detail. We analyze the security in Section 4 and explain our implementation and evaluation in Section 5. Section 6 discusses deployment issues, Section 7 reviews related work, and Section 8 concludes the paper.

2 PRCASH OVERVIEW

Our goal in this paper is to design a new digital currency that provides a novel combination of features: issuance control, transparency, regulation, and privacy. In this section we give an overview of our solution, PRCash.

2.1 Motivation for Central Issuance

Research on digital currencies has recently focused on decentralized systems. In this paper, we deviate from this trend and design a currency where money is issued centrally. Central issuance for cryptocurrencies has previously been explored by Danezis and Meiklejohn with RSCoin [23], though with somewhat different focus. While we mainly focus on privacy and regulation, RSCoin focuses on scalability.

Centrally-issued currencies can have significant advantages, especially if used as fiat money. Control over issuance provides an important tool for monetary policy, as the issuer can, e.g., increase supply of money to stimulate the economy when needed. Similar monetary policy is hard to realize if issuance is distributed and based on predefined rules and schedule, as in the case of Bitcoin mining. Another advantage is that a well-established issuer (e.g., a central bank) can provide reassurance to the public to start using a digital currency, in contrast to a decentralized solution with no institutional backing (it seems that the public prefers money that is reliable and boring rather than new and exciting [42]). Also, features like regulation are difficult to achieve in a fully decentralized setting.

We note that central issuance does not necessarily imply a fully-centralized solution. The operation of the currency (e.g., transaction verification, consensus and double-spending protection) can be distributed for increased security and fault tolerance, as is the case in our solution.

2.2 System Model

Figure 1 shows the system model of PRCash. Here, we describe the involved entities:

Issuer. In our currency new money is created by a central entity called the *issuer*. This role can be taken, e.g., by a central bank.

Users. Users in our system can act in two roles: as *payers* and as payment *recipients*. Users of the currency can be private individuals or organizations.

Validators. Our system leverages a pre-defined (permissioned) set of *validators*. Validators have two tasks: they verify correctness of transactions and publish transactions to a public data structure, called the *ledger*, that they also maintain in distributed manner. The role of the validators could be taken, e.g., by commercial banks or other institutions appointed by the central bank.

Regulator. The flow of money is regulated by a central entity called the *regulator*. If a user exceeds his allowed limit of anonymous transactions, his identity is revealed to the regulator. The role of the regulator could be taken, e.g., by a public authority like the IRS.

If our system is used for a privately-issued currency, these roles can be assigned differently (cf. Section 6).

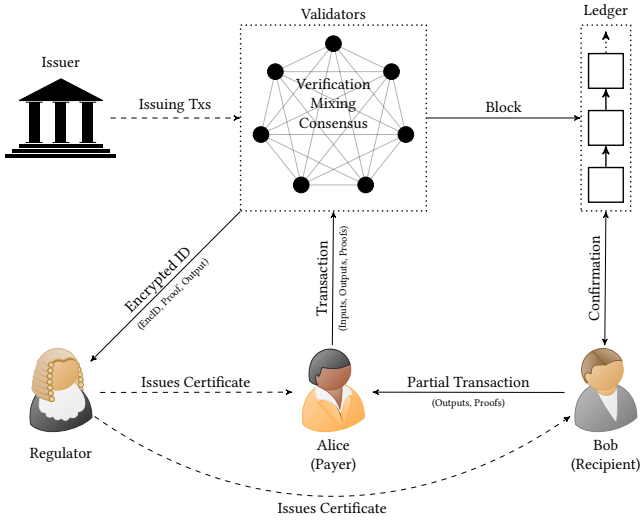


Figure 1: System model and operation. In PRCash, new money is created centrally by the issuer. Users enroll in the system by obtaining certificates from the regulator. In each payment, the payer (Alice) and the recipient (Bob) prepare a transaction that is sent to permissioned validators who verify its correctness and add it to the next block in the public ledger. If the transaction exceeds the allowed amount of anonymous payments for Alice or Bob, the identity of the user is revealed to the regulator.

2.3 High-Level Operation

Figure 1 illustrates the high-level operation of PRCash. To supply new money, the issuer creates signed issuance transactions that it sends to the validators, who verify them and publish them to the ledger. Similarly, the issuer can remove money (that is in the issuers possession) from the currency using deletion transactions.

Each user enrolls in the system by obtaining a payment credential (certificate) from the regulator. Payments involve two parties: the payer (Alice) and the recipient (Bob). To initiate a payment, Alice and Bob first agree on the transaction value. Each payment transaction consists of inputs and outputs, where the inputs are outputs from previous transactions (cf. Figure 2), as well as associated proofs, and for each transaction Alice and Bob can choose if their identity should remain anonymous. Bob prepares his part of the transaction (that includes value outputs and proofs) and sends it to Alice, who completes the transaction (by adding inputs, change outputs, proofs, and an encrypted identifier in case of a non-anonymous transaction). Alice sends the complete transaction to the validators.

The validators work in rounds. In each round, the validators collect incoming transactions, verify their correctness, mix the order of transaction inputs and outputs for increased privacy (cf. Section 4), and agree on the set of transaction that should be published. Consensus among validators is achieved through standard (Byzantine fault tolerant) protocols. Optionally, the privacy and robustness of the system can be increased by running the functionality of the validators in trusted execution environments (TEEs), such as Intel SGX

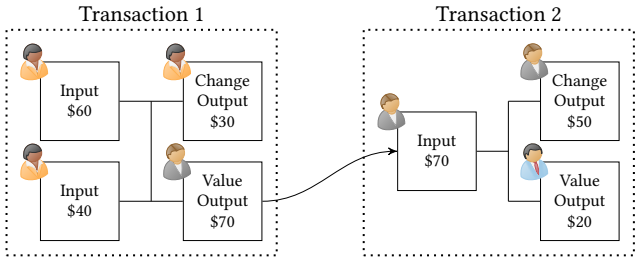


Figure 2: Transactions overview. Alice uses two inputs for Transaction 1 to transfer a value of $70 to Bob and creates a $30 change output in the process. Bob then uses the received output as an input in later Transaction 2 to Dave.

enclaves [28] (cf. Section 4). At the end of the round, the validators publish a set of verified transactions as a new block on the ledger. Once the recipient (Bob) verifies the presence of the transaction in the ledger, he considers the payment confirmed. Bob can then use the value outputs from this transaction as inputs in the next payment (cf. Figure 2).

If a transaction does not pass the verification (e.g., Alice or Bob attempts to create a transaction that exceeds the allowed anonymity limit, transaction inputs and outputs do not match, or one of the attached proofs is invalid), the transaction is rejected by the validators and not included in the next block. If the transaction contains any non-anonymous outputs, the validators first verify its correctness, and then forward the encrypted identifier to the regulator, who can recover the identity of Alice or Bob, depending on which transaction output was made non-anonymous or exceeded the limit.

2.4 Main Properties

PRCash provides different properties with respect to each involved entity. We summarize them below:

Issuer. The issuer can choose when and how much new money is created (or deleted). Every issuance event is recorded on the ledger. In Section 6, we discuss the option of the keeping issuance amounts secret.

Users. For users (and third parties), PRCash provides *transaction verifiability*. All accepted transactions are recorded on the ledger and anyone can verify that for each transaction, inputs match to outputs, and thus transactions can only transfer existing money from one user to another, and not create new money. Users also get *issuance transparency*. This means that users see every money issuance event and its amount on the ledger. For double-spending protection, users need to trust validator consensus. Users also have flexibility in terms of privacy protection. For each transaction, they can choose if their identity should be protected or not, and they can make unlimited non-anonymous transactions. If both the payer and the recipient choose anonymity protection, our system provides *strong privacy* guarantees: payer anonymity, recipient anonymity and value secrecy. Bob (e.g., merchant) cannot link multiple payments received from Alice (e.g., customer).

Validators. Validators can see all transactions, but they do not learn user identities or transaction amounts. Validators have

limited ability to link together multiple transactions from the same user (cf. Section 4).

Regulator. For the regulator, PRCash provides flexibility. By setting the limit to a suitable value, the regulator can enforce different regulatory rules. The regulator learns the identity of each user that exceeds the allowed limit. The regulator also learns the user identity of every transaction (output) that is non-anonymous.

2.5 Attacker Models

We consider an adversary that controls all networking between users and from users to validators. The validators and the regulator are connected with secure links. Regarding the validators, we consider two different attacker models:

AM1: Compromised validators. In this model, we assume that $f < N/3$ validators can be fully compromised, where N is the total number of validators. The adversary can read any secrets stored by the compromised validators and modify their execution control flow.

AM2: Trusted Execution Environments. During system operation, the attacker may compromise the OS on any validator, but all validator TEEs (e.g., SGX enclaves) remain secure, i.e., the adversary cannot read any data from the TEE or modify its execution. During system initialization, at most $f < N/3$ of the validator operating systems may be compromised.

3 PRCASH DETAILS

In this section, we describe PRCash in detail. We first introduce the main idea of our regulation technique, and then we explain our system operations. Our solution uses a number of cryptographic techniques as building blocks. We provide background on them in Appendix A.

3.1 Regulation Idea

In many countries, it is required by law to report large financial transactions. For example, the legislation in the US mandates companies and individuals to report any received cash transaction that exceeds $10,000 [1]. To enable enforcement of such laws, we design a regulation mechanism that limits the *total amount* of anonymous payments any user can *receive* within a time period (epoch). By adjusting the amount and the period, authorities can control the flow of anonymous money, e.g., reception of anonymous payments up to $10,000 could be allowed within a month. With small changes, similar limits are also possible for spending instead of receiving (cf. Section 6).

To realize regulation, for each transaction the user either proves without disclosing his identity that he does not exceed the limit v_a in the current epoch e or he connects his identity encrypted with the regulators public key to the transaction. For anonymous transactions within the limit, each user computes a pseudorandom ID per epoch (PID_e) that he attaches to his transaction outputs. He additionally attaches a zero-knowledge proof that the ID was computed correctly and a range proof over the sum of all transaction *outputs* from this PID. These values are sent together with the transaction outputs to the validators. The proofs are checked by the validators and after verifying their correctness, the PIDs and the corresponding proofs are not published with the transactions to preserve anonymity towards third parties.

Since anonymous change outputs are indistinguishable from anonymous value transferring outputs, they count towards the receiving limit of a user. However, since users are in control of the size of the outputs they receive, they can mitigate this issue by using smaller received outputs, by splitting larger outputs in non-anonymous transactions, or by creating large change outputs non-anonymously (cf. Appendix 6).

During enrollment, the user receives a *certificate* that allows him to create correctness proofs required for regulation. As the user may lose his certificate, or the corresponding private key, we limit their validity to I_Δ epochs.

A similar idea for regulation was proposed previously for centralized coin-based digital cash by Camenisch et al. [14]. Our solution uses blockchain based transactions and thus needs to consider not just the number of transactions (which is equal to the amount in coin based systems), but also the transaction amounts. Our solution therefore uses different zero-knowledge proofs and provides some additional desired properties (cf. Section 7).

3.2 System Initialization

Our system uses two groups $G = \langle g \rangle$ and $\mathbf{G} = \langle \mathbf{g_1} \rangle = \langle \mathbf{g_2} \rangle = \langle \mathbf{h} \rangle$ of the same order, where the discrete logarithms of $\mathbf{g_1}$, $\mathbf{g_2}$, and $\mathbf{h}$ with respect to each other are unknown. The involved entities perform the following initialization steps:

Regulator. The regulator generates a keypair $(pk_{R,S}, sk_{R,S})$ for randomizable signatures (cf. Appendix A.4), an encryption keypair $(pk_{R,E}, sk_{R,E})$ for Elgamal encryption, and publishes the public keys as part of the system setup.

Validators. Each validator installs the same code (in its TEE) that creates a keypair, exports the public key, and seals the private key for local storage. The public keys are published as part of the system setup. Validators use the private keys for signing new blocks. Users use the validator public keys to send transactions securely to the validators.

Issuer. The issuer also creates a keypair that he uses for transactions that create and delete money. The issuer publishes his public key as part of the system setup.

3.3 User Enrollment

Every new user obtains the system setup that includes the public keys of the regulator, issuer, and validators. To enroll in the system, the user generates a keypair $(pk_U, sk_U) = (\mathbf{g_1}^{sk_U}, sk_U)$ for regulation proofs and sends the public key to the regulator while proving knowledge of the secret key (cf. Appendix A.2). To ensure that a user cannot enroll multiple identities, and thus circumvent the regulation, the regulator has to verify the identity of the user. If a PKI is already in place, this can be used for identification, otherwise users could, e.g., be required to visit a registration office in person.

The regulator then creates a certificate consisting of a randomizable signature σ on (sk_U, I_V) based on the user's public key pk_U and I_V, the index of the first epoch in which the certificate is valid, and sends the signature σ to the user. Recall that a randomizable signature is a signature on a list of committed values (cf. Appendix A.4). Using values pk_U and I_V, the regulator creates and signs

the commitment $pk_U \cdot \mathbf{g_2}^{I_V}\mathbf{h}^r = \mathbf{g_1}^{sk_U}\mathbf{g_2}^{I_V}\mathbf{h}^r$ where r is chosen at random.

3.4 Regulation Proof Creation

In each epoch e, the user computes a pseudorandom ID as $\mathsf{PID}_e = f_{sk_U}(e)$ (cf. Appendix A.1) and initializes the value of anonymously spent transaction outputs to $v_e = 0$. Regulation proofs are created either when Bob creates value outputs during transaction preparation or when Alice creates change outputs during transaction completion. For each output, the user can choose if it should be made anonymous or non-anonymous. For each new output, the user creates a *regulation proof*. Depending on whether the output should be anonymous or not, he does one of the following to construct the proof:

Anonymous Output. If the user wants to create an output anonymously and the value v_o of the transaction output plus v_e is below the limit v_a, the user adds PID_e and a zero-knowledge proof of knowledge (cf. Appendix A.2) of (sk_U, I_V, σ) to the transaction such that:

(i) The certificate is valid in the current epoch, i.e., a range proof that $I_{current} - I_\Delta < I_V \leq I_{current}$.
(ii) The value PID_e is equal to the output of the pseudorandom function based on the secret key sk_U on input e, i.e., $\mathsf{PID}_e = f_{sk_U}(e)$.
(iii) The certificate is valid, i.e.,

$$\mathsf{verify}(pk_{R,S}, (sk_U, I_V), \sigma) = \mathsf{true}$$

In detail, the regulation proof consists of the following steps:

(i) The user creates two commitments $\mathbf{A} = \mathbf{g_1}^{sk_u}\mathbf{h}^{r_1}$ and $\mathbf{B} = \mathbf{g_2}^{I_V}\mathbf{h}^{r_2}$ with two fresh random values r_1 and r_2 and proves knowledge of a signature on the openings of these commitments.
(ii) Prove that $\mathbf{B}$ is a commitment to an integer in the range $[I_{current} - I_\Delta + 1, I_{current}]$.
(iii) Given the commitment $\mathbf{A}$ to the value sk_U, prove that

$$\mathsf{PID}_e = f_{sk_U}(e) = g^{1/(e+sk_U)}$$

i.e., this is the following proof of knowledge:

$$PK\{(\alpha, \gamma) : \mathbf{A} = \mathbf{g_1}^{\alpha}\mathbf{h}^{\gamma} \wedge g \cdot \mathsf{PID}_e^{-e} = \mathsf{PID}_e^{\alpha}\}$$

We use the common notation where greek letters correspond to values of which knowledge is being proven (cf. Appendix A.2). In the proof above, α corresponds to sk_U and γ corresponds to the blinding value of the commitment. The second term proves that the ID was computed correctly since

$$\begin{aligned} & g \cdot \mathsf{PID}_e^{-e} = \mathsf{PID}_e^{\alpha} \\ \Rightarrow \quad & g = \mathsf{PID}_e^{e+\alpha} \\ \Rightarrow \quad & g^{\frac{1}{e+sk_U}} = \left(\mathsf{PID}_e^{e+\alpha}\right)^{\frac{1}{e+\alpha}} = \mathsf{PID}_e \end{aligned}$$

The interactive protocol can be easily converted to a non-interactive signature on the message $M = H(o)$ using the Fiat-Shamir heuristic [27], where o is the transaction output. Including this message in the zero-knowledge proof binds the proof to the transaction output.

(iv) The user additionally creates a range proof over the product of all anonymous outputs that share the same identifier PID_e, proving that their combined value is below the allowed limit v_a.

The user then updates $v_e := v_e + v_o$ after completing the transaction.

Non-anonymous Output. If the user does not want to create the output anonymously or the value v_o of the output plus v_e is above the transaction amount limit v_a, the user adds his public key encrypted with the public key of the regulator to the transaction, together with a proof that the encryption was created correctly. The user completes the following steps to create the regulation proof:

(i) The user creates two commitments $\mathbf{A} = \mathbf{g_1}^{sk_u}\mathbf{h}^{r_1}$ and $\mathbf{B} = \mathbf{g_2}^{I_V}\mathbf{h}^{r_2}$ with two fresh random values r_1 and r_2 and proves knowledge of a signature on the openings of these commitments.
(ii) Prove that $\mathbf{B}$ is a commitment to an integer in the range $[I_{current} - I_\Delta + 1, I_{current}]$.
(iii) Compute $C = \mathsf{ENC}(pk_U, pk_{R,E}) = \left(g^{y_1}, pk_{R,E}^{y_1} \cdot pk_U\right)$
(iv) Given the commitment $\mathbf{A}$ to the value sk_U, prove that

$$C = \mathsf{ENC}(pk_U, pk_{R,E}) = \left(g^{y_1}, pk_{R,E}^{y_1} \cdot pk_U\right)$$

i.e., this is the following proof of knowledge:

$$\begin{aligned} PK\{(\alpha, \gamma_1, \gamma_2) : & \mathbf{A} = \mathbf{g_1}^{\alpha}\mathbf{h}^{\gamma_1} \\ & \wedge C[0] = g^{\gamma_2} \wedge C[1] = pk_{R,E}^{\gamma_2} g^{\alpha}\} \end{aligned}$$

Here, α again corresponds to sk_U and γ_1 corresponds to the blinding value of the commitment, while γ_2 corresponds to the random value used for the Elgamal encryption of the users public key. The interactive protocol can again be converted to a non-interactive signature on the message $M = H(o)$ using the Fiat-Shamir heuristic [27], where o is the transaction output, to bind the proof to the transaction output.

3.5 Transaction Creation

Our transactions should provide strong privacy and public verifiability at the same time. Previous value-hiding transaction schemes such as Confidential Transactions [34] and MimbleWimble [30] (cf. Appendix A.6) are publicly verifiable for correctness, but have the undesirable property that the payment recipient necessarily sees the change outputs created by the payer. This means that, e.g., a merchant can link two independent sales if a client uses a change output from a previous transaction with the same merchant.

We adopt the high-level transaction approach from [30], but enhance our transaction processing for improved privacy. Similar to [30, 34], our transactions are based on a group G in which the discrete logarithm problem is hard, with generators g and h for which the discrete logarithm to each others base is unknown. These generators are used to represent transaction inputs and outputs as homomorphic commitments to the associated value, thereby hiding their values from other parties. The homomorphic commitments have the property that one can easily add and subtract committed values without opening the commitments, e.g. for two commitments

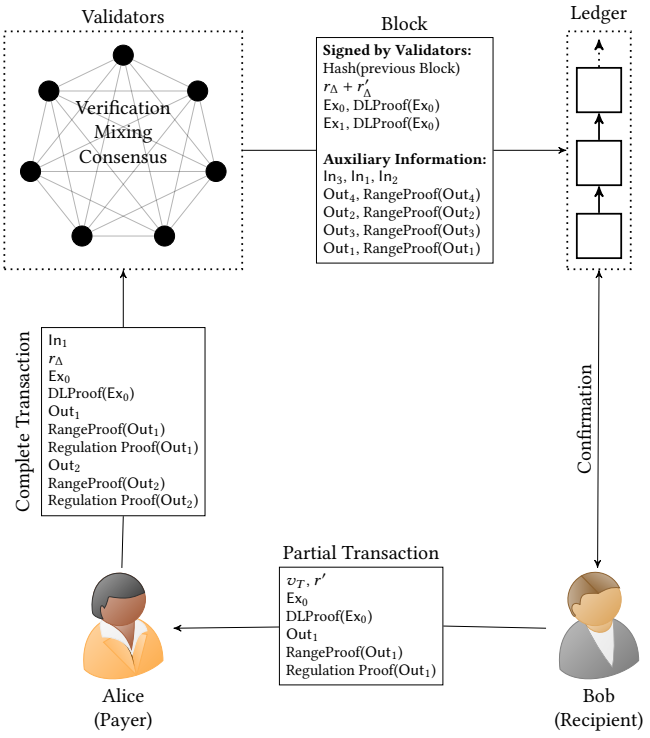


Figure 3: Transaction and block creation. In this example transaction, Alice pays an amount v_T to Bob. First, Bob creates a partial transaction that he sends to Alice, who completes it by adding her inputs, outputs and proofs. Alice then sends the complete transaction over a secure connection to a validator. The validators verify and mix the transactions and reach consensus on a block that they then sign and publish as part of the ledger. The block in this example consists of the two transactions shown in Figure 4.

$g^{r_1}h^{v_1}$ and $g^{r_2}h^{v_2}$ to the values v_1 and v_2, one can easily compute a commitment to their sum $v_1 + v_2$ by multiplying the commitments: $g^{r_1}h^{v_1} \cdot g^{r_2}h^{v_2} = g^{r_1+r_2}h^{v_1+v_2}$. If the blinding factors are chosen carefully, this property can be used to check that the sum of the input values of a transaction is equal to the sum of the output values, and the knowledge of the blinding factors can be used to authenticate and authorize payments [30]. We show in Appendix B that the knowledge of the blinding factor of an output is a secure method for payment authorization.

To prevent the above mentioned transaction tracking, we modify the transaction creation such that the payer finalizes the transaction. To increase payment anonymity further, we also include another output (r_Δ) that does not have a value attached. This additional output is submitted to the validators as a scalar such that multiple transactions can be merged. Inclusion of such additional output makes it impossible to later match transaction inputs to corresponding outputs.[1]

[1]Matching transaction inputs to outputs after reordering is in general already an NP-complete problem (subset sum). However, most transactions will only have few inputs and outputs, which can make linking feasible in practice without this additional measure.

Our transaction creation protocol, that includes the regulation proofs explained above, is shown in Figure 3. The protocol proceeds as follows:

(i) The recipient, Bob, creates k *value outputs* $\mathsf{Out}_i = g^{r'_i}h^{v'_i}$ $(1 \le i \le k)$, for the payment value $v_T = \sum_{i=1}^{k} v'_i$. For each of the value outputs, he also creates a range proof to prove that the value is in a valid range (i.e., that no overflow occurs where money is created out of nothing). He additionally attaches a regulation proof to each output as described above in Section 3.4. He then creates an *excess output* $\mathsf{Ex}_0 = g^{r'_0}$ that has no value attached, proves knowledge of r'_0 by proving knowledge of the discrete log of Ex_0 to base g (DLProof(Ex_0)) and sends his outputs (including range proofs, proof of knowledge of r'_0 and regulation proofs), v_T and $r' = r'_0 + \sum_{i=1}^{k} r'_i$ to Alice. The additional excess output Ex_0 is required to ensure that only Bob can spend his newly created outputs. Otherwise Alice would know the sum of the blinding factors of his outputs and could thus spend them. An example for such a *partial transaction* is shown in Figure 3, where Bob creates one value output (Out_1).

(ii) If Alice agrees with the transaction value v_T, with her inputs $\mathsf{In}_i = g^{r_i}h^{v_i}$ $(1 \le i \le n)$, s.t. $v = \sum_{i=1}^{n} v_i$ and $r = \sum_{i=1}^{n} r_i$, she creates m *change outputs* $\mathsf{Out}_i = g^{r'_i}h^{v'_i}$ $(k < i \le k + m)$, s.t. $v - \sum_{i=k+1}^{k+m} v'_i = v_T$ and she creates range proofs and regulation proofs for these outputs. She then computes a *delta output* $r_\Delta = r - \sum_{i=k+1}^{k+m} r'_i - r'$ and combines all of her inputs, Bob's and her outputs (including all proofs) and r_Δ into a *complete transaction*. Alice' inputs are outputs of a previous transaction that can be a money issuing transaction as described in Section 3.9. In the example in Figure 3, Alice uses one input (In_1) and one change output (Out_2) in the transaction.

(iii) Finally, Alice sends the complete transaction to one or more validators, as shown in Figure 3 encrypted under their public keys. The number of validators depends on the used transaction validation strategy (see Section 5).

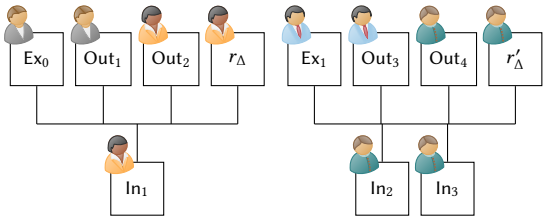
3.6 Transaction Verification

The validators work in rounds and verify every received transaction. A transaction is correct, if

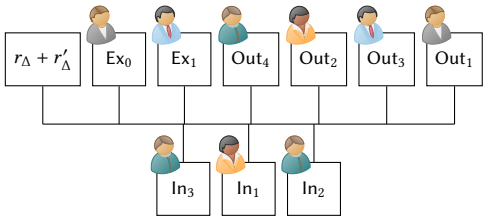
(i) all inputs are unspent outputs of previous transactions,
(ii) the range proofs for all outputs are correct,
(iii) the zero-knowledge proof for excess outputs is correct, and
(iv) the total amount of transaction inputs matches the outputs: $\Pi_{i=1}^{n}\mathsf{In}_i = g^{r_\Delta} \cdot \mathsf{Ex}_0 \cdot \Pi_{i=1}^{k+m}\mathsf{Out}_i$

Two example transactions are shown in Figure 4a. For the transaction from Alice to Bob, the validators check if $\mathsf{In}_1 = g^{r_\Delta}\mathsf{Ex}_0\mathsf{Out}_1\mathsf{Out}_2$ and if the proof of knowledge of the discrete logarithm of Ex_0 (DLProof(Ex_0)), as well as the range proofs for Out_1 and Out_2 are correct.

In addition to verifying the correctness of the transaction itself, the validators verify the regulation proofs. First, the validators verify the randomized certificate, i.e., they verify the signature on the provided commitments and check if the range proof for I_V is correct. If the verification fails, the transaction is discarded.



(a) A transaction from Alice to Bob (on the left) and a transaction from Charlie to Dave (on the right). The transactions fulfill the conditions $\mathsf{In}_1 = g^{r_\Delta}\mathsf{Ex}_0\mathsf{Out}_1\mathsf{Out}_2$ and $\mathsf{In}_2\mathsf{In}_3 = g^{r'_\Delta}\mathsf{Ex}_1\mathsf{Out}_3\mathsf{Out}_4$, respectively.



(b) The two transactions from above can be merged as shown here. The merged transactions fulfills the condition $\Pi_{i=1}^{3}\mathsf{In}_i = g^{r_\Delta + r'_\Delta}\mathsf{Ex}_0\mathsf{Ex}_1\Pi_{i=1}^{4}\mathsf{Out}_i$ and is thus still a valid transaction. Since the order of inputs and outputs is irrelevant for the validity condition, inputs and outputs can be reordered arbitrarily.

Figure 4: Transaction combining. Shown above are two transactions before and after combining. The first transaction (on the left) goes from Alice to Bob, the second transaction (on the right) from Charlie to Dave. Outputs created by Alice are marked, outputs created by Bob, outputs created by Charlie and outputs created by Dave.

Otherwise, for anonymous transaction outputs, the validators verify that PID_e has been computed correctly and that the proof is bound to the associated output. If this check succeeds, they compute the product of all outputs from epoch e that share the pseudorandom identifier PID_e and check if the provided range proof holds for this product. If this is the case, the total associated value is below the allowed limit and the transaction can be included in the next block. Otherwise, the transaction is discarded.

For non-anonymous transaction outputs, the validators verify the corresponding regulation proof, i.e., that the public key of the user has been encrypted correctly with the public encryption key of the regulator and that this proof is bound to the associated transaction output. If these verifications are successful, the validators include the transaction in the next block and forward the output and the proof to the regulator, otherwise the transaction is discarded.

When the regulator receives transaction outputs with their corresponding proofs, he can decrypt the encrypted public key which serves as identifier for the user. The regulator also checks the proofs to ensure that the output was indeed created by the owner of the corresponding public key. Since the regulator knows the real-world identities associated with each public key, he can then take action as required.

3.7 Mixing and Consensus

The validators collect a set of verified transactions and in the end of the round mix them by using two merging properties of our transactions. The first merging option is to *combine* two valid transactions together which creates another valid transaction. Combining several transactions into one large transaction breaks the direct correlation between inputs and outputs in the original transactions. The more transactions are combined in one round, the harder it is for third parties to link inputs and outputs based on published, combined transactions. An example for this process is shown in Figure 4 where two transactions are combined into one and the inputs and outputs are reordered. Since the order of inputs and outputs is irrelevant for the correctness of a transaction, they can be reordered arbitrarily. Additionally, by only publishing the sum of the delta outputs instead of the individual values, deciding which set of transaction outputs belong to which set of inputs becomes impossible.

The second merging option is *compacting*. If an output of one transaction appears as an input in another transaction, the matching input-output pair can be simply be removed, resulting in a smaller but still valid transaction. Compacting makes transaction linking more difficult and improves storage efficiency. Once the validator has verified and merged (mixed) all received transactions in the current round, the remaining inputs and outputs can be simply sorted as a list for publishing.

The validators need to achieve consensus over the content of the next block depending on the assumed attacker model and used block validation model. The validators run a Byzantine fault tolerant consensus protocol to protect against double spending. If $f < N/3$ fully compromised validators are assumed (**AM1**), the consensus protocol also needs to cover transaction verification. The list of published inputs and outputs needs be signed by $f + 1$ validators. If secure trusted execution environments are assumed (**AM2**), transactions only need to be verified by a single validator and consensus is only required to ensure that no double-spending has happened.

Validators can cache unspent transaction outputs from all previous blocks to speed up verification of new transactions (needed for double-spending protection). After achieving consensus over a block, validators can remove all inputs of the block from their cached set and add all new outputs to it.

3.8 Block Structure

Each block consists of a first part signed by the validators and a second part containing auxiliary information. The first signed part contains the sum of all delta outputs, all excess outputs including the zero-knowledge proofs of their exponents, and the hash of the previous block. Additionally, if the block contains an issuance or a deletion transaction, the signed part also contains the explicit amounts of money that are added or removed. As auxiliary information, the block contains a list of inputs and a list of outputs including their range proofs.

An example block that consists of two transactions is shown in Figure 3. The signed part of the block only contains the excess outputs and the sum of the delta outputs of all transactions (Ex_0, Ex_1 and $r_\Delta + r'_\Delta$ in the example). The transaction inputs and transaction

outputs with a value do not need to be included in the signed part, but they still need to be published including the range proofs of the outputs, so that other parties can verify the correctness of the blockchain.

This block structure allows compression of the blockchain by compacting transactions across blocks. Outputs of previous transactions that are used as inputs in the new block can be removed from storage without losing the ability to verify the complete chain. All that is required for the verification is the set of unspent transaction outputs, excess and delta outputs of all blocks, and the values of issuance and deletion transactions. All of this combined can be interpreted as one large transaction that, if valid, implies the validity of the whole blockchain. This makes the storage required to verify the full chain very small and slowly growing for third parties that do not want to store all transactions. An example for this is shown in Figure 5 (Appendix A).

3.9 Issuance and Deletion

Our currency provides an explicit mechanism for the issuer to increase, or decrease, the amount of currency in circulation. This can be done with a special transaction type that requires a signature from the issuer.

Specifically, the issuer can publish an *issuance transaction* with an explicitly stated amount v. The issuer creates k transaction outputs $\mathsf{Out}_i = g^{r'_i} h^{v'_i}$ $(1 \leq i \leq k)$, such that $v = \sum_{i=1}^{k} v'_i$, and which all have a range proof attached. The issuer then additionally creates an excess output $\mathsf{Ex}_0 = g^{r'_0}$, s.t. $r'_0 + \sum_{i=1}^{k} r'_i = 0$ and proves knowledge of r'_0. The transaction is valid, if h^v is equal to the sum of the outputs. The outputs created by such an issuing transaction could, e.g., be transferred to commercial banks who can then further distribute the newly created money. The issued amount v is published in plaintext to the next block with the issuance transaction. In Appendix 6 we discuss the option of keeping the amount secret.

The issuer can also remove money in their possession from the system by creating a special *deletion transaction* that destroys an explicitly specified amount v of money. Inputs to the transaction are n (previously unspent) transaction outputs $\mathsf{In}_i = g^{r_i} h^{v_i}$ $(1 \leq i \leq n)$. As output the transaction has k change outputs $\mathsf{Out}_i = g^{r'_i} h^{v'_i}$ $(1 \leq i \leq k)$, and an excess output $\mathsf{Ex}_0 = g^{r'_0}$, s.t. $r'_0 + \sum_{i=1}^{k} r'_i = \sum_{i=1}^{n} r_i$ and $v + \sum_{i=0}^{k} v'_i = \sum_{i=1}^{n} v_i$. The transaction is valid if $\Pi_{i=1}^{n} \mathsf{In}_i = h^v \cdot \mathsf{Ex}_0 \cdot \Pi_{i=1}^{k} \mathsf{Out}_i$

4 SECURITY ANALYSIS

In this section, we provide an informal security analysis of PRCash.

Payment authorization. We first consider an attacker that tries to spend an output belonging to another user without the knowledge of the corresponding blinding factor. We show in Appendix B that if an adversary capable of such an attack exists, our assumptions are violated, namely either the discrete logarithm problem can be solved efficiently in the used group or the adversary knows the discrete logarithm of h to base g, where g and h are the generators used for the commitments. The intuition behind this is that, to create a valid transaction, the outputs require range proofs for which knowledge of the blinding factor is needed and the outputs have to be chosen such that their product is equal to that of the inputs.

Double-spending protection. We first consider cases with $f < N/3$ fully compromised validator (**AM1**). During each round, each non-compromised validator discards transactions with previously used or otherwise invalid inputs (cf. Section 3.6), and then all validators run a standard Byzantine fault tolerant consensus protocol. As a result, all non-compromised validators agree on the same set of transactions (i.e., the next block). Users consider the next block confirmed when it has been signed by $f + 1$ validators. Thus, compromised validators cannot produce a block that would contain conflicting transactions and the required number of signatures. When TEEs are considered secure (**AM2**), the consensus protocol guarantees that all validators agree on the next block. A block signed by the TEE of one validator cannot contain conflicting transactions that allow double spending.

Creation of money. Only the issuer can create new money. Creation of money using normal transactions is prevented as the validators verify (i) the range proofs of all outputs for overflow and (ii) that the sum of inputs values matches the sum of output values, and only include compliant transactions in the next block. Assuming f fully compromised validators, the consensus protocol guarantees that each block (signed by $f + 1$ validators) contains only compliant transactions.

Privacy towards third parties. Transaction values are completely hidden and can therefore not leak any information about a transaction. Additionally, all transactions are mixed by the validators (in TEEs), and since the delta outputs of all transactions are summed up (cf. Section 3.5) and not published individually, it becomes impossible for third parties examining the ledger to determine which outputs belong to which inputs, even for a merchant receiving a transaction. PRCash therefore provides k-anonymity [39] against third parties, where k is the number of transactions in a block. For example, even if an adversary knows that Alice payed Bob in a transaction with output Out_1 contained in a block with 500 transactions, he can only guess Alice' input with probability of at most $\frac{1}{500}$. If more privacy is desired, blocks can be made larger and validators could even add dummy transactions (with a tradeoff in efficiency).

Privacy between users. As the payer finalizes the transaction, the recipient only sees his own outputs, i.e. he is in the same position as the third party entity with partial information as described above. The payer additionally sees output commitments from the recipient which allows him to see when the output is spent. However, once the output has been used, no more information is leaked to the user.

Privacy towards validators. Assuming secure trusted execution environments (**AM2**), a compromised validator OS can withhold messages from its TEE, which could be used for a Sybil attack on the mixing of transactions. If the validator only forwards a single real transaction to the TEE, he will be able to determine which outputs belong to which inputs for that transaction.

Assuming f fully compromised validators (**AM1**), the adversary can link transaction inputs and outputs for all transactions that the compromised validator receives. In addition, the validator is able to link multiple outputs from the same epoch if they share the same pseudorandom ID. However, in both cases, the transaction value is still hidden, no addresses are reused as no explicit adresses exist in our system, and only a small amount of partial information about

the transaction graph is leaked to the adversary which makes it unlikely that an analysis similar to deanonymization attacks on Bitcoin (e.g. [6, 35]) would succeed, as the most powerful heuristics for such attacks are tracing transactions by amounts and clustering by addresses.

Regulation enforcement. The security of our regulation system relies on the security of the underlying zero-knowledge proofs and the pseudorandom function. The pseudorandom function (cf. Appendix A.1) is secure under the decisional Diffie-Hellman inversion assumption (DDHI). The zero-knowledge proofs rely on the hardness of the discrete logarithm problem (which is implied by DDHI) and they are secure as non-interactive proofs in the random oracle model using the Fiat-Shamir heuristic [27, 38].

To bypass regulation, Alice could send the blinding factor of one of her outputs to Bob. This would allow Bob to spend that output in the future and additionally Bob's epoch specific receiving count would stay unaffected. While this could be seen as a violation against regulation, we do not consider it a practical attack, because Bob has no guarantee that Alice will not use the same blinding factor herself, and thus such direct transfer of blinding factor cannot be used as a payment (at least not in most payment scenarios). A payment of this form would be equivalent to Alice disclosing the private key of one of her addresses to Bob for a payment in Bitcoin to transfer all of the money in that address to Bob.

5 EVALUATION

We implemented a prototype of PRCash to evaluate its performance. In this section, we describe our implementation, transaction verification models, verification overhead, and overall performance in terms of throughput and latency.

5.1 Implementation

We implemented a prototype that covers the generation and verification of transactions, including the regulation proofs. Our implementation uses the randomizable signature from Pointcheval and Sanders [37] for the generation of certificates. Other signatures with efficient protocols, such as CL-Signatures [15, 16], could be used as well. We use the RELIC toolkit [7] for the elliptic curve and bilinear map operations. Our implementation makes use of the 256-bit elliptic curve BN-P256 as the base curve of a type-3 pairing that we use for the randomizable signatures. Our range proofs use commitments to digits in base 4 (cf. Appendix A.3) as this is in practice the most efficient base for the size and computation of bit-commitment based proofs. The size and computation required for the proofs could be optimized by using bulletproofs from Bünz et al. [11].

5.2 Verification Models

The throughput and latency of PRCash depends on the used transaction verification model that in turn is dependent on the assumed attacker model. For our evaluation, we consider the following three verification models, to give examples of performance under different assumptions and requirements.

VM1: Full replication. In this model, all validators verify all transactions, including the regulation proofs, and consensus is needed on the validity of all transactions and proofs. This model guarantees transaction correctness, double-spending protection, and enforcement regulation at all times, given f compromised validators (**AM1**).

VM2: Partitioned regulation, replicated verification. In this model, all validators verify correctness of all transactions including their range proofs, but excluding the regulation proofs. Verification of regulation proofs is instead partitioned evenly among the validators. If one validator attests to the validity of a regulation proof, it is accepted by the other validators. If a validator gets compromised (**AM1**), users can transact anonymously above the regulatory limit. This model may be used, if validator compromise is considered unlikely and temporary, and it is acceptable to lose the ability to enforce regulation momentarily. Transaction correctness (i.e., no new money is created and no double-spending occurs) is guaranteed regardless of the compromise. This model may also be suitable, if e.g. regulation is delegated to commercial banks that act as validators and check the proofs for their customers (cf. Section 6).

VM3: Full partitioning. In this model, only one validator TEE verifies all proofs for each transaction. If one validator TEE attests to the validity of a transaction and associated proofs, it is accepted by the other validators. Full partitioning can be used when validator TEEs are assumed to be secure (**AM2**). Consensus is only required for the set of published transactions (i.e., no conflicting transactions and double-spending).

Proof Type	Time [s]	Size [bytes]
ZKPoK of discrete log (DLProof)	0.00038	64
PIDProof (epoch range = 2^6)	0.01067	1033
PIDProof (epoch range = 2^8)	0.01235	1226
PIDProof (epoch range = 2^{10})	0.01404	1419
EncIDProof (epoch range = 2^6)	0.01115	968
EncIDProof (epoch range = 2^8)	0.01284	1161
EncIDProof (epoch range = 2^{10})	0.01452	1354
RangeProof (range = 2^8)	0.00665	722
RangeProof (range = 2^{16})	0.01345	1544
RangeProof (range = 2^{20})	0.01678	1930
RangeProof (range = 2^{32})	0.02722	3088

Table 1: The average time for proof verification for different proof types and their sizes.

5.3 Transaction Verification Overhead

We measured the verification overhead (shown in Table 1), averaged over 1000 runs on a single core of an Intel Core i7-4770 CPU, for the following proof types:

ZKPoK of discrete log. This is a zero-knowledge proof of knowledge (ZKPoK) of the discrete logarithm and is required to verify that an excess output has no value attached.

PIDProof. This is the proof that the pseudo-random ID was constructed correctly, i.e., the user who created the proof is in possession of a valid certificate on his key and that the PID was derived correctly from this key. Depending on the number of epochs for which the signature is valid, the computation time differs, due to the included range proof. In

Table 1, the measurements for epoch ranges between 2^6 and 2^{10} are shown.

EncIDProof. This is the proof that the user who created the proof is in possession of a valid certificate on his key and that his corresponding public key was correctly encrypted with the public key of the regulator. Again, the verification time differs depending on the number of epochs for which the certificate is valid.

RangeProof. The range proof by itself is used to show that an output is in the correct range, which is necessary to show that no overflow occurs, and to prove that the sum of anonymous outputs with the same PID are below the allowed threshold. The size of the range proof and its verification time depend on the size of the range. For example, with a granularity of cents, a range of 2^{32} would allow transaction outputs of up to 43 million dollars.

Most commonly, transactions will have one value-transferring output, one change output, one or more inputs, plus an excess and a delta output. Since inputs do not require range proofs, and the time required to compute the commitment to the sum of their values is negligible compared to the proof verification time, we can estimate the time required to validate a standard transaction independently of the number of inputs.

In the case of a transaction with two anonymous outputs (different PIDs each), a full verification of the transaction requires verifying one ZKPoK of a discrete logarithm, two PID proofs, and four range proofs (one for each individual output and one per PID).

Since the maximum amount for anonymous transactions is limited, one can use a smaller range proof than for non-anonymous transactions. For example, the US requires reporting for transactions above $10,000 [1]. An equivalent regulatory rule with a granularity of cents would approximately correspond to a range of 2^{20}. Assuming a certificate validity of 2^{10} epochs, this leads to a total verification time of 0.096 seconds.

For transactions with non-anonymous outputs, we can allow a much larger range (e.g., 2^{32}), since in this case the goal is not to limit transaction size but to prevent overflows. Such a transaction requires two range proofs, giving, in the same setting as before, a verification time of 0.084 seconds. Combinations, where one output is anonymous and one is not, are, of course, also possible.

Given this transaction verification overhead, within one second, roughly ten transactions can be fully verified on a single core. From this value we can in turn estimate the required computing resources to handle the expected transaction load (e.g., 1000 transactions per second).

In verification model **VM1**, each validator checks all transactions and proofs. To verify 1000 tps, each validator would require approximately 25 quad-core servers. In **VM2**, transactions and range proofs are verified by all validators to protect against overflows in outputs, but verification of regulation proofs can be partitioned across the validators. Assuming 16 validators, each of them would require 15 quad-core servers to process 1000 tps. In **VM3**, the load for verification can be fully spread across the validators. With 16 validators each of them would require two quad-core servers to verify 1000 tps.

# Validators	Batch	Latency	VM1	VM2	VM3
4 (1 region)	6.2MB	0.288s	2000 tx/s	3400 tx/s	72000 tx/s
8	1.6MB	0.58s	250 tx/s	420 tx/s	8800 tx/s
8	6.2MB	1.48s	390 tx/s	670 tx/s	14000 tx/s
16	1.6MB	0.69s	210 tx/s	360 tx/s	7500 tx/s
16	3.1MB	1.04s	280 tx/s	480 tx/s	10000 tx/s
32	0.4MB	0.48s	80 tx/s	130 tx/s	2700 tx/s
32	1.6MB	0.925s	160 tx/s	270 tx/s	5600 tx/s
64	0.4MB	0.824s	40 tx/s	70 tx/s	1500 tx/s
64	1.6MB	1.79s	80 tx/s	140 tx/s	2900 tx/s

Table 2: The latency given the number of validators and batch size (from [22]). Estimated values for the throughput given the verification model.

5.4 Consensus Performance and Liveness

The validators also need to run a consensus protocol to agree on the set of transactions that are published to the ledger. We use the measurements from Croman et al. [22] to estimate the performance of a standard consensus protocol (PBFT [18]). As their measurements become bandwidth bound with larger batches, we can use their numbers for latency and throughput to estimate the throughput in our system if the batch size in bytes remains the same.

The sizes of our proofs are summarized in Table 1. Note that the transaction outputs can be reconstructed from their range proofs, i.e., the size for range proofs includes the transaction output. Since a normal transaction, consisting of multiple inputs and two outputs with an attached value, has a size of approximately 10.8kB (anonymous outputs) or 9.0kB (non-anonymous outputs) including all proofs, and the numbers from [22] use 190 byte transactions, the throughput in terms of transactions per second (tx/s) has to be adjusted conservatively by a factor of 0.018 if we require consensus on all proofs (**VM1**). In a deployment where no consensus on the validity of the regulation proofs is required (**VM2**), the transaction size reduces to approx. 4.0kB (anonymous) or 6.3kB (non-anonymous), i.e., we conservatively adjust by a factor of 0.030. In verification model **VM3**, the verification of all proofs is partitioned and consensus is only required to protect against double spending and not on the validity of transactions. The resulting size for the consensus relevant part of a standard transaction (two inputs, two outputs with attached value, and an unspendable output and the zero-knowledge proof of its discrete logarithm) is 229 bytes. Since some transactions can be larger, we assume an average size of 300 bytes which would correspond to a scaling factor of 0.63.

The achieved values for throughput and latency for the three verification models are shown in Table 2. For example, with 16 validators and a batch size of 3.1MB, the latency of consensus is 1.04s and the throughput ranges from 10,000 tps (**VM3**) to 280 tps (**VM1**). Note that the measurements on which our estimations are based, were conducted with nodes that were globally distributed in 8 regions (except for the 4 node experiment). Croman et al. [22] used t2.medium Amazon EC2 instances which have limited bandwidth. If the validators are geographically close and have a higher bandwidth or dedicated lines between them (which would be reasonable for a digital fiat currency), the throughput could be increased and the latency could be reduced further.

Liveness in our system is provided by the underlying consensus protocol. In the case of PBFT [18] or similar BFT protocols, this requires that less than 1/3 of the validator nodes are faulty (crash or behave arbitrarily). It can be easily seen for **VM1** that this requirement ensures liveness, but might not be obvious for **VM2/3**. In **VM2/3**, if the verification of transaction outputs and proofs are partitioned among the validators based on some fixed property (e.g., associated PID), interrupting the communication of one validator could remove liveness for the transaction outputs assigned to that validator. However, the partitioning can instead be done arbitrarily (e.g. a user chooses a validator at random) and any validator can verify any transaction outputs and associated proofs. For that, it is only required that a validator informs all other validators of the PID associated with an anonymous output that he has verified. This allows other validators to verify range proofs for that same PID later in the epoch. It follows that **VM2/3** offer the same liveness guarantees as **VM1**. Since all validator TEEs are trusted in **VM3**, faults can no longer be completely arbitrary. Thus, optimized protocols (e.g., FastBFT [31]) could be used to further increase throughput, decrease latency, or to scale to a larger number of validators.

6 DISCUSSION

Regulation limits. PRCash controls the total amount of money that a user can receive anonymously. All anonymous transaction outputs, including *change outputs*, count towards this limit. Change outputs must be counted, because they are not distinguishable from value transferring outputs. While new techniques might allow making such a distinction, keeping this indistinguishability can be also explicitly desired: If change outputs and other outputs can be distinguished, analyzing the transaction graph becomes significantly easier. Research on Bitcoin shows that tracing change outputs is one of the most powerful heuristics for deanonymization [6, 35].

We argue that in most deployments, the regulation limit would be much higher than the typical (anonymous) transaction volume of users (e.g., $10,000 per month) and they would not be adversely affected by the inclusion of (small) change inputs. The issue can also be mitigated by using smaller transaction outputs and, if necessary, splitting up large outputs into smaller ones using non-anonymous transaction. Nevertheless, it remains an interesting open question for future research whether this issue can be solved elegantly without sacrificing privacy by making change outputs distinguishable.

If desired, the regulation limit can alternatively be put on *spending*. To limit spending, instead of creating regulation proofs for the outputs, the payer has to create regulation proofs for the inputs of a transaction.Systems that limit both spending and receiving are also possible. In such a case, each user would require one key pair for spending transactions, another for receiving transactions and two corresponding certificates.

Regulation enforcement. In the traditional banking system it is usually the responsibility of commercial banks to report high value transaction. For our system, this would mean that instead of having one regulator, we could delegate this role to multiple commercial banks, where each bank would be responsible for the regulation for their customers. In such a system, a user would receive a certificate from his bank when, e.g., opening a bank account. The commercial banks could operate as validators and each bank would check the regulation proofs that use a certificate signed by them. Such a deployment would be fitting for partitioned regulation (**VM2/3**), where each bank, also acting as validator, is responsible for checking the regulation proofs of a subset of transaction outputs.

Issuance. Instead of having a central bank issue new currency, a digital currency, that is still subject to regulatory legislation, could also be issued by a private company or a consortium. In such a scenario, the importance of issuance transparency is increased, as (i) private entities may not be as cautious in their monetary policy and (ii) the trust in private entities may be lower than the trust in a central bank. Excessive issuance can have drastic consequences such as hyperinflation [9] that can even lead to the collapse of the currency.

The issuance of new currency could be the responsibility of a single entity, or of a consortium, or could be predefined similar to systems such as Bitcoin [36]. Alternatively, instead of just providing issuance transparency, issuance control could be delegated to the users of the system which poses interesting open questions of how to realize distributed issuance control on a technical level.

Secrecy of issuance could be a desirable feature, e.g. because a central bank does not want to publicly release fine-grained issuance information, but rather periodic summaries. In such cases, the explicit issuance values could simply be removed (and privately shared with banks that distributed the money further). The central bank could still prove later the amount they issued, e.g., when releasing a yearly report.

7 RELATED WORK

In this section we compare PRCash to payments in the current monetary system and to previous digital currency systems. Table 3 summarizes our comparison.

Current monetary system. The current monetary systems enable two types of payments. The first is cash that provides good privacy and makes large payments that bypass regulatory oversight difficult (although not impossible). The second type is digital payments that transfer money with the help of a trusted authority, e.g., credit card payments, bank transfers, as well as systems like PayPal [5], Apple Pay [2], and Google Wallet [3]. Such payments provide no privacy, as all transaction details can be seen by the trusted authority. As shown in Table 3, PRCash provides the combined benefits of cash and digital payments, with added issuance transparency.

Coin-based systems. Chaum's original e-cash system [19] uses a centralized bank and anonymous coins and provides payer anonymity, but, as received money needs to be deposited immediately to prevent double spending, it cannot hide payment values or ensure recipient anonymity. Payments are also expensive in the sense that payments of large amounts of money require the transfer and processing of many coins.

Various improvements to Chaum's design have been proposed. Schemes such as [13, 14, 20] allow offline payments, where payment recipients can deposit coins later, allowing several payments to be mixed together, which hides transaction values from the bank (to certain extent). This has the drawback that it only guarantees

		Privacy			Functionality				Performance	Security
		Payer Anonymity	Recipient Anonymity	Value Hiding	Regulatory Control	Issuance Transparency	Transaction Verifiability	Transferable	Constant Overhead Payments	Double-Spending Prevention
Current monetary system	Cash	●	●	●	◐	-	-	●	●	●
	Digital payments (credit card, PayPal)	-	-	-	●	-	-	◐	●	●
Coin-based systems (with trusted authority)	Traditional e-cash (Chaum [19])	●	-	-	●	-	-	-	-	●
	Offline e-cash (Chaum et al. [20])	●	-	◐	◐	-	-	-	-	◐
	Regulated e-cash (Camenisch et al. [14])	●	-	◐	●	-	-	-	-	◐
	Regulated e-cash (online variant of [14])	●	-	-	●	-	-	-	-	●
	Transferable e-cash (Balmitsi et al. [8])	●	●	●	-	-	-	●	-	◐
Transaction-based systems (with distributed ledger)	Pseudonymous (Bitcoin [36])	◐	◐	-	-	●	●	●	●	●
	Pseudonymous Transactions (RSCoin [23])	◐	◐	-	-	●	●	●	●	●
	Plaintext IDs (Hyperledger [4])	-	-	-	●	●	●	●	●	●
	Value-hiding (Confidential Txs. [34])	◐	◐	●	-	●	●	●	●	●
	Value-hiding (MimbleWimble [30])	◐	●	●	-	●	●	●	●	●
	Fully anonymous (ZeroCash [40])	●	●	●	-	●	●	●	●	●
PRCash		●	●	●	●	●	●	●	●	●

Table 3: Comparison of payments in the current monetary system and proposed digital currencies (● = provided property, ◐ = partially provided property, - = not provided property).

double-spending detection in contrast to double-spending prevention.

The variant that is closest to ours, is a regulated e-cash scheme by Camenisch et al. [14]. This design allows a trusted authority to control the total amount of anonymously spent money. We use similar zero-knowledge proof techniques for PRCash. However, in their scheme, it suffices to limit the number of transactions, since the system is coin-based, i.e., the number of spent coins is equal to the amount. In our solution, we also need to take into account the values of the transactions, while keeping them secret. In a coin-based scheme, the size of the transaction and the computation required to verify the proofs grows with the transaction value. Additionally, such a system cannot provide recipient anonymity. Partial value secrecy is possible when offline payments are allowed, but this option ensures only double-spending detection (no prevention). In comparison, PRCash provides better privacy, constant payment overhead, and more transparency.

Transferable e-cash schemes [8] can provide anonymity to both the payer and the recipient, and also hides transaction values, since the coins do not need to be deposited immediately. However, such schemes only detect but do not prevent double-spending, and lack regulatory control and transparency.

As shown in Table 3, none of the e-cash schemes provides strong privacy and regulation at the same time, in contrast to PRCash. Additionally, PRCash provides better transparency and payment efficiency compared to all e-cash schemes. While compact e-cash schemes like [13] improve the efficiency of withdrawing and storing coins, all coin based schemes still require spending each coin individually and thus the size of the payment and verification time grows with the payment amount.

Transaction-based systems. Bitcoin [36] uses a consensus mechanism based on mining incentives, which removes the necessity for a central authority and makes the digital money transferable by representing it as transactions on a public ledger. While Bitcoin is a permissionless design (i.e. anyone can participate in the consensus protocol), permissioned blockchain deployments (with a pre-defined set of validators) are also possible. In either model, transactions can be represented in different ways.

Transactions in Bitcoin, and related cryptocurrencies, are pseudonymous which provides limited privacy as transaction amounts are publicly visible and many transactions can be deanonymized [6, 35]. Additionally, usage of pseudonyms does not allow regulation, as any user can create an unlimited number of identities for receiving and spending money. In fact, avoiding any oversight by authorities is a desired feature in currencies like Bitcoin.

Blockchain transactions can also be based on real identities. Hyperledger [4] provides frameworks for permissioned blockchains, where the trusted authorities that maintain the ledger could verify the identity of each user and encode this to each transaction. Such systems support easy regulation, but provide no privacy.

Similar to our paper, RSCoin [23] explores the idea of a centrally issued cryptocurrency, although with a different goal. While we focus on privacy and regulation, RSCoin focuses on the scalability of the consensus.

Recently proposed value-hiding blockchain transactions provide improved privacy guarantees, but no regulation. Confidential transactions [34] make transaction values private, but do not efficiently protect payer or recipient anonymity. MimbleWimble [30] provides recipient anonymity, but lacks strong payer anonymity. ZeroCash [40] provides the strongest level of anonymity, i.e., value, payer and recipient are completely hidden. However, while verification is efficient, creating an anonymous transaction requires minutes of computation which makes it impractical for cash-like payments, where transaction should be finalized within seconds.

In summary, none of the transaction-based systems provides simultaneously similar privacy and regulation properties as PRCash (see Table 3).

8 CONCLUSION

Despite more than three decades of research on digital currencies, their adoption as fiat money issued by a central bank has not become a reality. While the reasons for this may be numerous, and not always purely technical, a major obstacle for the adoption of previous solutions has been missing features, such as controlled issuance, regulation, privacy and transparency. In this paper, we have presented PRCash, a novel digital currency that enables all of these features at the same time, and also provides the required performance to make replacement of cash with a digital currency practical. We believe that the new currency, and the new techniques, put forth in this paper can be a significant step towards the adoption of digital currencies as legal tender.

REFERENCES

[1] [n. d.]. 31 CFR 1010.330 - Reports relating to currency in excess of $10,000 received in a trade or business. https://www.law.cornell.edu/cfr/text/31/1010.330. ([n. d.]).

[2] [n. d.]. Apple Pay. ([n. d.]). https://www.apple.com/apple-pay.

[3] [n. d.]. Google Wallet. ([n. d.]). https://www.google.com/wallet.

[4] [n. d.]. Hyperledger. ([n. d.]). https://www.hyperledger.org.

[5] [n. d.]. PayPal. ([n. d.]). https://www.paypal.com.

[6] Elli Androulaki, Ghassan O. Karame, Marc Roeschlin, Tobias Scherer, and Srdjan Capkun. 2013. Evaluating User Privacy in Bitcoin. In *Financial Cryptography and Data Security: 17th International Conference, FC 2013, Okinawa, Japan, April 1-5, 2013, Revised Selected Papers*. Springer Berlin Heidelberg, Berlin, Heidelberg, 34–51. https://doi.org/10.1007/978-3-642-39884-1_4

[7] D. F. Aranha and C. P. L. Gouvêa. [n. d.]. RELIC is an Efficient LIbrary for Cryptography. https://github.com/relic-toolkit/relic. ([n. d.]).

[8] Foteini Baldimtsi, Melissa Chase, Georg Fuchsbauer, and Markulf Kohlweiss. 2015. Anonymous Transferable E-Cash.. In *Public Key Cryptography*. 101–124.

[9] Peter Bernholz. 2003. *Monetary Regimes and Inflation*.

[10] Fabrice Boudot. 2000. Efficient proofs that a committed number lies in an interval. In *Advances in Cryptology—EUROCRYPT 2000*. Springer, 431–444.

[11] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. 2017. *Bulletproofs: Efficient range proofs for confidential transactions*. Technical Report. Cryptology ePrint Archive, Report 2017/1066, 2017. https://eprint. iacr. org/2017/1066.

[12] Jan Camenisch, Rafik Chaabouni, et al. 2008. Efficient protocols for set membership and range proofs. In *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 234–252.

[13] Jan Camenisch, Susan Hohenberger, and Anna Lysyanskaya. 2005. Compact E-Cash. In *Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings (Lecture Notes in Computer Science)*, Vol. 3494. Springer, 302–321. https://doi.org/10.1007/11426639_18

[14] Jan Camenisch, Susan Hohenberger, and Anna Lysyanskaya. 2006. Balancing accountability and privacy using e-cash. In *International Conference on Security and Cryptography for Networks*. Springer, 141–155.

[15] Jan Camenisch and Anna Lysyanskaya. 2003. *A Signature Scheme with Efficient Protocols*. Springer Berlin Heidelberg, Berlin, Heidelberg, 268–289. https://doi.org/10.1007/3-540-36413-7_20

[16] Jan Camenisch and Anna Lysyanskaya. 2004. *Signature Schemes and Anonymous Credentials from Bilinear Maps*. Springer Berlin Heidelberg, Berlin, Heidelberg, 56–72. https://doi.org/10.1007/978-3-540-28628-8_4

[17] Jan Camenisch and Markus Stadler. 1997. *Efficient group signature schemes for large groups*. Springer Berlin Heidelberg, Berlin, Heidelberg, 410–424. https://doi.org/10.1007/BFb0052252

[18] Miguel Castro and Barbara Liskov. 1999. Practical Byzantine Fault Tolerance. In *Proceedings of the Third Symposium on Operating Systems Design and Implementation (OSDI '99)*. USENIX Association, Berkeley, CA, USA, 173–186. http://dl.acm.org/citation.cfm?id=296806.296824

[19] David Chaum. 1983. Blind Signatures for Untraceable Payments. In *Advances in Cryptology: Proceedings of Crypto 82*. Springer US, Boston, MA, 199–203. https://doi.org/10.1007/978-1-4757-0602-4_18

[20] D. Chaum, A. Fiat, and M. Naor. 1990. Untraceable Electronic Cash. In *Proceedings on Advances in Cryptology (CRYPTO '88)*. Springer-Verlag New York, Inc., New York, NY, USA, 319–327. http://dl.acm.org/citation.cfm?id=88314.88969

[21] Victor Costan and Srinivas Devadas. 2016. *Intel SGX explained*. Technical Report. Cryptology ePrint Archive, Report 2016/086.

[22] Kyle Croman, Christian Decker, Ittay Eyal, Adem Efe Gencer, Ari Juels, Ahmed Kosba, Andrew Miller, Prateek Saxena, Elaine Shi, Emin Gün Sirer, et al. 2016. On scaling decentralized blockchains. In *International Conference on Financial Cryptography and Data Security*. Springer, 106–125.

[23] George Danezis and Sarah Meiklejohn. 2016. Centrally Banked Cryptocurrencies. In *23nd Annual Network and Distributed System Security Symposium, NDSS 2016, San Diego, California, USA, February 21-24, 2016*. http://www.internetsociety.org/sites/default/files/blogs-media/centrally-banked-cryptocurrencies.pdf

[24] Yevgeniy Dodis and Aleksandr Yampolskiy. 2005. *A Verifiable Random Function with Short Proofs and Keys*. Springer Berlin Heidelberg, Berlin, Heidelberg, 416–431. https://doi.org/10.1007/978-3-540-30580-4_28

[25] T. Elgamal. 1985. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory* 31, 4 (Jul 1985), 469–472. https://doi.org/10.1109/TIT.1985.1057074

[26] Uriel Feige, Amos Fiat, and Adi Shamir. 1988. Zero-knowledge proofs of identity. *Journal of cryptology* 1, 2 (1988), 77–94.

[27] Amos Fiat and Adi Shamir. 1987. *How To Prove Yourself: Practical Solutions to Identification and Signature Problems*. Springer Berlin Heidelberg, Berlin, Heidelberg, 186–194. https://doi.org/10.1007/3-540-47721-7_12

[28] Intel. [n. d.]. Intel Software Guard Extensions. ([n. d.]). https://software.intel.com/en-us/sgx.

[29] Intel. 2016. Software Guard Extensions Tutorial Series. (2016). Available at: https://software.intel.com/en-us/articles/introducing-the-intel-software-guard-extensions-tutorial-series.

[30] Tom Elvis Jedusor. [n. d.]. Mimblewimble. http://mimblewimble.org/mimblewimble.txt. ([n. d.]).

[31] Jian Liu, Wenting Li, Ghassan O Karame, and N Asokan. 2016. Scalable Byzantine Consensus via Hardware-assisted Secret Sharing. *arXiv preprint arXiv:1612.04997* (2016).

[32] Wenbo Mao. 1998. Guaranteed correct sharing of integer factorization with off-line shareholders. In *Public Key Cryptography*. Springer, 60–71.

[33] Ueli M Maurer. 2009. Unifying Zero-Knowledge Proofs of Knowledge. *AFRICACRYPT* 9 (2009), 272–286.

[34] Gregory Maxwell. 2015. Confidential Transactions. https://people.xiph.org/~greg/confidential_values.txt. (2015).

[35] Sarah Meiklejohn, Marjori Pomarole, Grant Jordan, Kirill Levchenko, Damon McCoy, Geoffrey M Voelker, and Stefan Savage. 2013. A fistful of bitcoins: characterizing payments among men with no names. In *Proceedings of the 2013 conference on Internet measurement conference*. ACM, 127–140.

[36] Satoshi Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system. (2008).

[37] David Pointcheval and Olivier Sanders. 2016. Short randomizable signatures. In *Cryptographers' Track at the RSA Conference*. Springer, 111–126.

[38] David Pointcheval and Jacques Stern. 1996. Security proofs for signature schemes. In *Eurocrypt*, Vol. 96. Springer, 387–398.

[39] Pierangela Samarati and Latanya Sweeney. 1998. *Protecting privacy when disclosing information: k-anonymity and its enforcement through generalization and suppression*. Technical Report. Technical report, SRI International.

[40] 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*. IEEE, 459–474.

[41] C. P. Schnorr. 1991. Efficient signature generation by smart cards. *Journal of Cryptology* 4, 3 (1991), 161–174. https://doi.org/10.1007/BF00196725

[42] Tim Sullivan. 2015. Transparency, Trust, and Bitcoin. *Harvard Business Review* (June 2015).

A BACKGROUND

In this appendix, we provide background information on the cryptographic primitives that we use as building blocks for our currency. We also summarize Intel's SGX architecture [28].

A.1 Dodis-Yampolskiy Pseudorandom Functions

Dodis and Yampolskiy introduced a pseudorandom function [24] which, for a secret key sk and a generator g of a group G is defined as

$$f_{sk}(x) = g^{1/(x+sk)} \tag{1}$$

This construction is secure if the Decisional Diffie-Hellman Inversion assumption holds in group G.

A.2 Zero-Knowledge Proofs of Knowledge

A zero-knowledge proof of knowledge (ZKPOK) [26] allows one party (the prover) to prove to another party (the verifier) that a certain statement is true, without revealing any other information. Well-known techniques for proving knowledge of a discrete logarithm in zero-knowledge exist, such as [33, 41]. We use the notation introduced in [17] for proofs of knowledge. For example

$$PK\{(\alpha, \beta) : x = g_1^{\alpha} h^{\beta} \wedge y = g_2^{\alpha}\} \tag{2}$$

is a zero-knowledge proof of integers α and β, s.t. $x = g_1^{\alpha} h^{\beta}$ and $y = g_2^{\alpha}$ where g_1, h, x are elements of a group $G_1 = \langle g_1 \rangle = \langle h \rangle$ and g_2, y are elements of a group $G_2 = \langle g_2 \rangle$. In this notation, the values of which knowledge is proven are denoted by greek letters and all other values are known to the verifier. Using the Fiat-Shamir heuristic [27], such proofs can be converted to non-interactive proofs of knowledge.

A.3 Range Proofs

A range proof [10, 12] is a specific type of zero knowledge proof that allows to prove to a verifier that a committed value lies within a given range. A simple range proof can consist of proving that the committed value is one of the values in the interval, e.g., to prove that a commitment C is a commitment $g^r h^x$ where $x \in [0, 7]$ and r is a random blinding factor, one can prove this with the following proof of knowledge:

$$PK\{(\alpha) : C = g^{\alpha} \vee C \cdot h^{-1} = g^{\alpha} \vee C \cdot h^{-2} = g^{\alpha} \vee \cdots \vee C \cdot h^{-7} = g^{\alpha}\}$$

Clearly, this becomes inefficient with larger ranges. Instead one can decompose the value into multiple commitments to the powers of two (bit commitments), for example, such that the product of these commitments is equal to the original commitment. It then suffices to prove for every commitment that it either commits to zero or the correct power of two [32]. The proof above then becomes the following proof by splitting the commitment into three commitments C_0, C_1, C_2:

$$\begin{aligned} PK\{(\alpha, \beta, \gamma) :& (C_0 = g^{\alpha} \vee C_0 \cdot h^{-1} = g^{\alpha}) \\ &\wedge (C_1 = g^{\beta} \vee C_1 \cdot h^{-2} = g^{\beta}) \\ &\wedge (C_2 = g^{\gamma} \vee C_2 \cdot h^{-4} = g^{\gamma})\} \end{aligned}$$

This approach can be generalized to proofs in any base [12].

A.4 Randomizable Signatures

A randomizable signature scheme provides the ability to prove the possession of a signature on committed values. The signature schemes by Camenisch and Lysyanskaya [15, 16] or the scheme by Pointcheval and Sanders [37] allow to obtain a signature on a list of committed values without disclosing the values to the signer. The recipient of the signature can then prove efficiently that he is in possession of a signature on these values using fresh commitments on the same values.

A.5 Elgamal Encryption

Elgamal encryption [25] is an encryption scheme based on the difficulty of the discrete logarithm and thus compatible with standard techniques for zero-knowledge proofs. Elgamal encryption is secure in a group G if the Decisional Diffie-Hellman assumption holds in that group. To encrypt a message $m \in G$ with the public key $pk \in G$, a value r is chosen at random and the ciphertext is then computed as $(g^r, m \cdot pk^r)$. The recipient of a ciphertext (c_1, c_2) can then decrypt the message using the secret key sk (corresponding to the public key pk) as $m = c_2 \cdot (c_1^{sk})^{-1}$.

A.6 Confidential Transactions and MimbleWimble

In a transaction-based digital currency, Confidential Transactions [34] represent the transaction input and output values as homomorphic commitments. As the commitments are homomorphic, one can choose the blinding factors for the outputs such that the sum of the input commitments is equal to the sum of the output commitments, if the sum of the input values is equal to the sum of the output values. This allows verifying the correctness of a transaction without knowledge of the transferred values.

MimbleWimble [30] uses the same approch, but adds an additional non-spendable output to which no value is attached. This allows the recipient of a transaction to create output commitments without the payer knowing the blinding factor, i.e., the blinding factor of the commitment is only known to the recipient of a payment, and can thus be used to authenticate a following payment.

MimbleWimble has the property that the blockchain can be easily compressed. Our solution inherits this property from MimbleWimble. The signed part of each block contains a hash of the previous block, all excess outputs of the block, and the sum of the delta outputs. The auxiliary part of the block contains all transaction inputs and all transaction outputs. Outputs of previous transactions that are used as inputs in the new block can be removed from the set of unspent transaction outputs (UTXOs) while new outputs are added. All spent outputs and inputs can be completely removed from storage once the UTXO set has been updated. Using the UTXOs, excess and delta outputs of all blocks, and the values of issuance and deletion transactions, the full chain can still be verified. All of this combined can be interpreted as one large transaction that, if valid, implies the validity of the whole blockchain. An example for how this compression works in PRCash is shown in Figure 5.

A.7 Intel SGX

Intel's Software Guard Extension (SGX) [28] is a set of CPU instructions for creating and managing isolated software components, called *enclaves*. Enclaves are isolated from all software running on the system, including privileged OS. Enclave data is handled in plain-text only inside the CPU (i.e., in caches and registers) and when moved out of the CPU, e.g., into the memory (DRAM), encrypted and integrity protected.

The OS, although untrusted, is responsible for creating and managing enclaves. However, all initialization actions of the OS are recorded securely by SGX inside the CPU. The initialization process creates a *measurement* that captures the enclave's code configuration and can be used for later verification by an external party using *remote attestation*. The *sealing* capability of SGX enables persistent secure storage of enclave data such that the data is only available

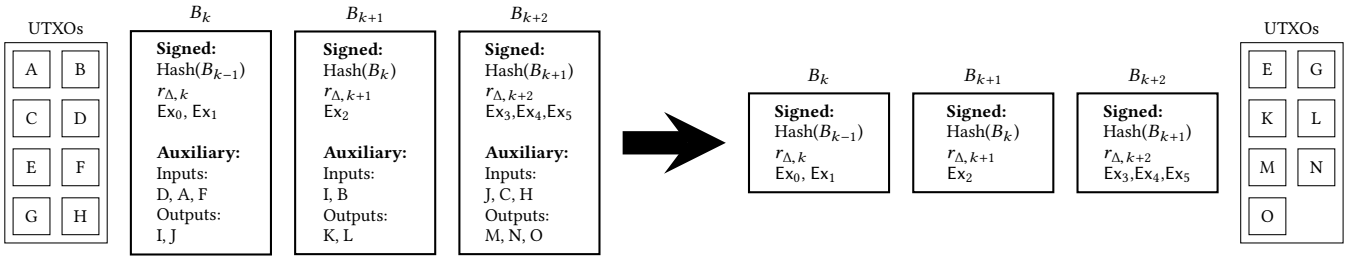


Figure 5: Blockchain compression. Given a set of unspent transaction outputs (UTXOs) the blockchain can be easily compressed when adding new blocks (here B_k,B_{k+1},B_{k+2}). All inputs of the new blocks are removed from and new outputs are added the UTXO set. The inputs and outputs can then be removed from the individual blocks, thus compressing the chain.

to correctly created instances of the same enclave that originally saved it.

Enclaves cannot execute system calls, and therefore developers must divide their applications into two parts: protected enclave and an unprotected part that run as normal user-level process and handles operations such as file system access and networking using the untrusted OS. For more detailed explanation of SGX, we refer the reader to [21, 29].

B PAYMENT AUTHORIZATION PROOF

In this Appendix, we show that, given a group $G = \langle g \rangle = \langle h \rangle$, a transaction output $\mathsf{Out} = g^r h^v$ can only be spent by an authorized entity, i.e. an entity who knows the secret blinding factor r.

Theorem B.1. *If the discrete logarithm problem is hard in G and the discrete logarithm of h to base g is unknown, the probability that an output* $\mathsf{Out} = g^r h^v$ *can be spent by an adversary without knowing r is negligible.*

Proof. For our proof, we distinguish the following two cases:

Case 1: W.l.g. assume that an attacker is able to create three outputs $\mathsf{Out}' = g^{r'} h^v$, $\mathsf{Ex}' = g^{r''}$ and r_Δ s.t. $\mathsf{Out} = g^{r_\Delta} \mathsf{Ex}' \mathsf{Out}'$, i.e. $g^r h^v = g^{r_\Delta} g^{r''} g^{r'} h^v$ with non-negligible probability. For the transaction to be valid, the adversary needs to attach a proof of knowledge of r'' and a range proof for v, i.e. a proof of knowledge of r', v s.t. v is in the allowed range. As the zero knowledge proofs are sound if the discrete logarithm problem is hard, the adversary knows r', r'', v (and of course r_Δ). Consider a game where the adversary wins, if on input $(g^r h^v, v)$ the adversary outputs values r', r'', r_Δ s.t. $g^r h^v = g^{r_\Delta} g^{r''} g^{r'} h^v$. Clearly, the adversary considered above can win this game with non-negligible advantage.

We now show how to construct a solver for the discrete logarithm problem given this adversary. On input $X = g^x$, the solver creates a randomized instance $(X \cdot g^{r_1} h^{r_2}, r_2) = (g^{x+r_1} h^{r_2}, r_2)$ with r_1, r_2 chosen uniformly at random. On output (y, z, w) from the adversary, the solver computes and outputs $x' = y + z + w - r_1$. If the adversary wins the game, clearly $x = x'$, i.e. the solver correctly computes the discrete logarithm of the input X and thus, if the adversary has a non-negligible probability to be able to create a valid transaction, the discrete logarithm problem can be solved efficiently in group G. As this violates our assumption, it follows that an attacker cannot create such a valid transaction with non-negligible probability.

Case 2: W.l.g. assume that an attacker is able to create three outputs $\mathsf{Out}' = g^{r'} h^{v'}$, $\mathsf{Ex}' = g^{r''}$ and r_Δ s.t. $v \neq v'$ and $\mathsf{Out} = g^{r_\Delta} \mathsf{Ex}' \mathsf{Out}'$, i.e. $g^r h^v = g^{r_\Delta} g^{r''} g^{r'} h^{v'}$, with non-negligible probability. For the transaction to be valid, the adversary needs to attach a proof of knowledge of r'' and a range proof for v', i.e. a proof of knowledge of r', v' s.t. v' is in the allowed range. As the zero knowledge proofs are sound if the discrete logarithm problem is hard, the adversary knows r', r'', v' (and of course r_Δ). We consider the game where the adversary wins, if on input $(g^r h^v, v)$ the adversary outputs values (r', r'', r_Δ, v') s.t. $g^r h^v = g^{r_\Delta} g^{r''} g^{r'} h^{v'}$ and $v \neq v'$.

We now show how this adversary can be used to compute $\mathsf{DL}_g(h)$. The solver creates a randomized instance $(g^{r_1} h^{r_2}, r_2)$ with r_1, r_2 chosen uniformly at random and gives this as input to the adversary. On output (y, z, w, u) from the adversary, the solver computes and outputs $x = (y + z + w - r_1)(r_2 - u)^{-1}$. If the adversary wins the game, this corresponds to $\mathsf{DL}_g(h)$, which violates our assumption, i.e. an attacker cannot create such a valid transaction.

□