


Fast, private and regulated payments in asynchronous networks

Maxence Brugeres ✉ 

LTCI, Telecom Paris, Institut Polytechnique de Paris, France
Forvis Mazars, France

Victor Languille ✉ 

LTCI, Telecom Paris, Institut Polytechnique de Paris, France
EDF R&D, France

Petr Kuznetsov ✉ 

LTCI, Telecom Paris, Institut Polytechnique de Paris, France

Hamza Zarfaoui 

LTCI, Telecom Paris, Institut Polytechnique de Paris, France

Abstract

We propose a decentralized asset-transfer system that enjoys *full privacy*: no party can learn the details of a transaction, except for its issuer and its recipient. Furthermore, the recipient is only aware of the amount of the transaction. Our system does not rely on consensus or synchrony assumptions, and therefore, it is *responsive*, since it runs at the actual network speed. Under the hood, every transaction creates a consumable *coin* equipped with a non-interactive zero-knowledge proof (NIZK) that confirms that the issuer has sufficient funds without revealing any information about her identity, the recipient's identity, or the payment amount. Moreover, we equip our system with a *regulatory enforcement* mechanism that can be used to regulate transfer limits or restrict specific addresses from sending or receiving funds, while preserving the system's privacy guarantees.

Finally, we report on *PaxPay*, our implementation of Fully Private Asset Transfer (FPAT) that uses the Gnark library for the NIZKs. In our benchmark, PaxPay exhibits better performance than earlier proposals that either ensure only partial privacy, require some kind of network synchrony or do not implement regulation features. Our system thus reconciles privacy, responsiveness, regulation enforcement and performance.

2012 ACM Subject Classification Security and privacy; Computer systems organization → Dependable and fault-tolerant systems and networks; Computing methodologies → Distributed computing methodologies

Keywords and phrases Anonymous, Asset Transfer, Asynchronous System, BFT, CBDC, NIZK, Payment System, Privacy, Regulation, Scalability, zk-SNARK

Related Version The Full Version is accessible at: <https://eprint.iacr.org/2025/098.pdf>

Funding *Maxence Brugeres*: Forvis Mazars Group.

Victor Languille: EDF R&D.

Petr Kuznetsov: CHIST-ERA-22-SPiDDS-05 (REDONDA project) and Agence Nationale de la Recherche (ANR-23-CHR4-0009).

1 Introduction

Payments, with and without consensus. Bitcoin [34] revolutionized the world of finance by proposing a fully decentralized asset-transfer system that allows an open set of users to consistently exchange assets without any mutual trust. Instead of relying on a central authority, users repeatedly engage in a form of *consensus* [21, 17] to maintain a replicated ledger—an ever-growing record of all transactions. It was later observed [26, 25] that,

strictly speaking, global consensus, a notoriously hard task [15, 16] that requires partial synchrony [21, 11], is not necessary in payments. In most common cases, when every account is maintained by a dedicated user, asset transfer can be implemented in an asynchronous, *responsive* way on top of the *reliable-broadcast* primitive [10] (instead of consensus).

Reliable broadcast is weaker than consensus as it allows the correct users to eventually and consistently agree on the *set* of issued transactions but not on their order. However, it turned out to be strong enough to prevent *double spending*, a major issue in asset-transfer systems. This observation gave rise to a series of purely asynchronous, *consensus-free* asset-transfer systems [13, 5, 32].

Privacy and regulations in payment systems. However, a decentralized asset-transfer system, as any replicated service, still faces a vital challenge of preserving privacy of its users. Indeed, in all the mentioned payment protocols, all transactions are inherently *public* and every user’s activities are traceable. Hiding the amount of a payment offers *confidentiality* and hiding the identities of the sender and receiver offers *anonymity*. Fully *private* payment systems ought to offer both confidentiality and anonymity. Furthermore, many important payment applications, such as Central Bank Digital Currency (CBDC) [4], require mechanisms for *regulatory enforcement*, such as limiting transaction amounts.

Existing distributed asset-transfer systems ([6, 7, 38, 39, 42, 45, 46], to name a few) arbitrate a trade-off between the four following aspects: **(a) Privacy guarantees**, **(b) Model assumptions**, **(c) Regulation enforcement**, and **(d) Performance**. Reconciling these four aspects is a onerous challenge, as strengthening one aspect often weakens another. Table 1 provides a detailed comparison of several payment systems across these dimensions. This table and its content are detailed and discussed in Section 8.

FPAT and PaxPay. In this paper, we address this challenge. We introduce the abstraction of *Fully Private Asset Transfer* (FPAT) that maintains conventional safety and liveness properties of a payment system (informally, no asset is spent twice and every transaction takes effect), while at the same time making sure that no transaction’s detail is leaked to a non-involved party. We then describe the *PaxPay* protocol, a FPAT implementation over an *asynchronous* network. PaxPay has been implemented in Golang using Gnark library [8].

As in UTXO transactions [34], to execute a transfer, the sender *spends* a set of *old coins* it has received earlier and *generates* a set of *new coins* of the same total value. Every coin is uniquely identified by its *serial number*. Every coin should be signed by a sufficiently large set (a $2/3$ *quorum*) of *validators*, i.e., dedicated parties that maintain lists of spent coins and make sure that no user spends a same coin twice.

The protocol operates in an asynchronous network, where any number of users and less than one third of validators can be *Byzantine* (deviating arbitrarily from its algorithm). The rest of the validators are considered *semi-honest* (following their algorithm but possibly sharing their states with the adversary). To ensure that the transfer is legal, several assertions have to be checked: the total value of the spent coins equals the total value of the new coins, the user that calls the transfer is the owner of the spent coins, all the coins are properly signed, etc.

To provide *full privacy* (confidentiality and anonymity), PaxPay leverages several cryptographic primitives. The legality of transactions is verified within a non-interactive zero-knowledge proofs (NIZK). A *blind* signature scheme is used at the validators’ side and these signatures are verified within the NIZK, allowing not to reveal the coins or their signatures when spending them. To verify these signatures efficiently, we implemented the NIZK using the Groth16 [24] scheme and employed a pairing-based signature scheme [36]. We then selec-

ted a pair of elliptic curves that form a chain to instantiate these cryptographic primitives, allowing efficient verification of the signature within the NIZK (we refer to Section 7 for more details).

Regulation. We also describe and implement a regulation enforcement mechanism that can be built on top of our system. Similar to PEReDi and PARScoin [38, 39], our approach allows setting limits on individual transfer amounts or the cumulative amount transferred by a user since they joined the system. However, regulations for CBDCs mandate stringent measures for Anti-Money Laundering (AML) and Countering the Financing of Terrorism (CFT). These measures go beyond simply limiting transaction amounts.

PEReDi and PARScoin [38, 39] address these requirements and enable validators to reveal the details of transfers if needed. This could put at risk the privacy of the users. Their approach relies on additional trust assumptions, requiring non-Byzantine validators to be *honest* (i.e., not to share any information with the adversary) to maintain privacy. We avoid such extra assumptions by equipping the users with a mechanism to generate cryptography proofs for arbitrary statements about the histories of their transactions. It enables extracting a requested information without disclosing the full content of the transaction histories.

Additionally, we incorporate a novel mechanism, empowering the regulator to impose *sanctions*, precluding specific addresses from interactions with other users. This ensures compliance with traditional financial sanctions emitted by the central banks or intergovernmental organizations (e.g., the UN¹) and explicitly required by the European Central Bank for the digital Euro [4].

The regulatory mechanism is implemented by introducing an additional type of coin, referred to as the *compliance coin*. At a time, each user may possess several regular coins, but only one unique compliance coin. In each transfer, the compliance coin of the sender must be spent together with the regular coins. Then a new compliance coin containing the information of the transfer is generated (i.e., signed by a quorum of validators). The compliance coin thus allows us to keep track records of all the interactions of a user with the system, which can be seen as a commitment to the transaction history and some aggregated values. To generate an initially valid compliance coin, before joining the system, the user must be registered with a *regulatory authority*.

Our regulation framework incurs very low impact on the transaction throughput. It mainly affects the transaction proving time (by doubling it compared to the non-regulated case), which we do not consider a primary performance factor. In contrast, such a mechanism could be difficult and performance-intensive to implement in other protocols that rely on Sigma protocols [14], such as UTT [42], Zef [6], Peredi [38], and Parscoin [39], which use these protocols to construct their NIZKs.

Performance. Finally, we evaluated the performance of PaxPay, in comparison with state-of-the-art payment systems that provide privacy-preserving and/or regulatory-enforcement features (cf Table 1). In our benchmark (detailed in Section 7.2), we witness that PaxPay can process at least 4 times more transactions per second than any of these systems. The benchmark on AWS EC2 instances demonstrates a throughput of 925 transactions per second. The performance gains can be explained by the use of Groth16 as a succinct NIZK with low verification cost. In return, this comes at the cost of heavy computations required to generate a proof. The time required to create a transfer request with limited computational resources in PaxPay on a one-core CPU is up to 7s for example. However, given the throughput

¹ <https://main.un.org/securitycouncil/en/sanctions/information>

gains, this appears to be a good trade-off. Moreover, PaxPay’s design allows for pushing the throughput even further by parallelizing the validator’s load on several machines.

Summary. To sum up our contributions:

- We formalize the problem by introducing the *Fully Private Asset Transfer* abstraction (FPAT).
- We propose PaxPay, a protocol that implements a FPAT object, and prove its correctness.
- We implement PaxPay in Golang, mainly leveraging on the Gnark [8] library.
- PaxPay achieves a true reconciliation of **(a) Privacy**, **(b) Model Assumptions** (network synchrony and trust assumptions for validators), **(c) Regulation Enforcement**, and **(d) Performance**. Unlike earlier works that propose trade-offs, PaxPay excels in each of these aspects simultaneously, bringing together the best of all these dimensions:
 - Regarding **Privacy**, transactions in PaxPay are *confidential*, *fully anonymous*, and *unlinkable*, ensuring the most robust privacy guarantees.
 - Regarding **Model Assumptions**, the system tolerates up to one-third Byzantine validators, with the remaining validators only required to be *semi-honest*, and does not rely on the network synchrony.
 - Regarding **Regulation**, PaxPay provides a comprehensive regulation framework by enabling users to generate cryptographic proofs about their transaction history, setting spending limits, and incorporating a sanction mechanism preventing sanctioned addresses to receive or spend funds.
 - Finally, our implementation outperforms state-of-the-art protocols. Also, we show that the transaction throughput of PaxPay scales with the computational power of validators. The system can further increase its supported throughput by distributing validators across additional machines.
- PaxPay can find a variety of use cases such as a standalone payment system, a *layer 2* solution on top of a blockchain (brings scalability, compliance, or privacy) or a CBDC.

2 Related work

We overview payment systems related to our work, the summary of our comparative analysis is presented in Table 1 compares this work with the related works (additional details are delegated to Table 3 and Appendix D).

Consensus-free payment systems. Gupta [26] and Guerraoui *et al.* [25] proposed a payment system that replaces the conventional consensus-based synchronization with the secure broadcast primitive [10, 33]. Systems like Astro [13], FastPay [5], Zef [6], PARScoin [39] and UTT [42] extend this approach, operating without consensus in asynchronous networks.

Private payment systems without regulation. Zerocash [7], Zexe [9], Monero [35], Quisqus [20] and Lelantus [28] are payment systems built on top of blockchains and consensus. They provide *privacy* via NIZKs or ring signatures. Zerocash and Zexe [7, 9] use *commitments* stored in Merkle trees and nullifiers to prevent double spending. Each transaction is accompanied by an NIZK to provide privacy but these systems offer inherently low transaction throughput. Monero [35] and Quisqus [20] relies on ring signatures to provide anonymity

² Fully anonymous among all the users (Full) or anonymous among an anonymity set (AS).

³ Semi-Honest (SH) or Honest (H).

■ **Table 1** Comparison of PaxPay vs. Zcash [18], Lelantus [28], Quisquis [20], Zef [6], UTT [42], PRCash [45], PEReDi [38] and PARScoin [39].

	Zcash	Lelantus	Quisquis	Zef	UTT	PRCash	PEReDi	PARScoin	PaxPay
PRIVACY PROPERTIES									
Confidential transfers	●	●	●	●	●	●	●	●	●
Sender-anonymous transfers	●	●	●	○	●	●	●	●	●
Receiver-anonymous transfers	●	●	●	●	●	●	●	●	●
Unlinkable transfers	●	●	●	○	●	●	●	●	●
Anonymity strategy ²	Full	AS	AS	Full	Full	Full	Full	Full	Full
MODEL ASSUMPTIONS									
Asynchronous network	○	○	○	●	●	○	○	●	●
Correct validators ³	SH	SH	SH	SH	SH	SH	H	H	SH
REGULATION FEATURES									
Limited held amount per user	○	○	○	○	○	○	●	○	○
Limited spendable amount per tx	○	○	○	○	○	●	●	●	●
Limited spendable amount in total	○	○	○	○	●	○	●	●	●
Full asset tracing	○	○	○	○	○	○	●	●	○
Sanctions	○	○	○	○	○	○	○	○	●
Provable transaction history	○	○	○	○	○	○	○	○	●
PERFORMANCE									
Transaction throughput (tx/s)	25	0	0	88	235	0	0	0	925
Transaction latency (s)	1000	0	0	< 1	< 1	0	0	0	< 1
NIZK Proving time (ms)	21K	2378	2110	438	60	100	3100	392	6959
NIZK Verification time (ms)	9	40	251	142	49	96	518	159	5

but only limits the sender and receiver’s identity to a small subset of the users, known as the *anonymity set*. Compared to Monero, Quisquis offers stronger privacy guarantees in certain scenarios preventing possible de-anonymization.

Lelantus [28, 29] offers privacy via NIZK requires less advanced cryptographic assumptions than Zerocash or Zexe, for instance it does not require any trusted setup. It also proposes batch verification for validators.

All payment systems mentioned so far are built on a blockchain and, thus, require consensus. Baudet et al. [6] describe Zef, a private payment system, that assumes an asynchronous network but only provides *partial* anonymity and confidentiality. To improve performance, the transaction data in Zef is distributed over a set of *authorities*. Each authority can be sharded — i.e., distributed over several machines, as in Astro [13] and in this work.

Private payment systems with regulation. PRCash [45], Platypus [46], PEReDi [38], PARScoin [39] and UTT [42] incorporate certain regulatory features into private payment systems. A common element in their designs is the reliance on NIZK.

Garman et al. [22] were among the first to address regulation by enforcing *policies* such as fixing a spending limit on a system with a similar construction as Zerocash. Their system has not been implemented and, as it is built on top of Zerocash, it shares the same major

drawbacks.

PRCash [45], as Zerocash, is blockchain-based and operates with commitments but offers only partial privacy. Regulation features proposed by PRCash are limited: users cannot spend more than a predefined amount set by the regulator within a certain time window.

UTT [42], a concurrent work to our paper, proposes an asynchronous private payment system with privacy guarantees, a low transaction latency and a high transaction throughput. Both UTT and PaxPay protocols follow a similar approach, using coin commitments and consistent broadcast to create and spend coins in a fully private manner. However, the regulation features offered by UTT are restricted to the *anonymity budget* that limits the amount of anonymous coins that a user can transfer. Compared to UTT, PaxPay offers a more comprehensive set of regulation features and exhibit better performance, due to improved NIZK construction.

Platypus [46] offers regulation features of holding limits, receiving limits, and spending limits. It uses NIZK with low verification costs to increase transaction throughput. However, unlike other payment systems described in this section, Platypus is a centralized payment system rather than a distributed one.

PEReDi [38] is an account-based system. The regulation framework enforces spending and receiving limits for each transaction and imposes restrictions on the maximum amount a user can hold. Additionally, the system allows validators to trace funds and reveal details of certain payments. Users encrypt their transaction details with a threshold encryption scheme so that a quorum of validators can decrypt them. PEReDi needs a synchronous network and *correct* (i.e., non-Byzantine) validators must be *honest* (i.e., strictly follow the protocol and avoid seeking additional information). In contrast, most systems discussed here (as well as ours) are designed to tolerate *semi-honest* correct validators.

Concurrently with this work, the same authors have recently proposed PARScoin [39]. PARScoin enhances PEReDi by allowing asynchronous transactions yet still requires honest validators and faces limited throughput. PARScoin's protocol is similar to the one employed by UTT and our work, relying on Byzantine consistent broadcast. Our work differs from PARScoin in the NIZK construction which offers better performance, fewer assumptions on the validators and additional regulations features. Also, neither PRCash, PARScoin nor PEReDi have conducted latency tests to evaluate the maximum transaction throughput of their systems, they only provide benchmarks of their primitives (such as the time to prove or verify a transaction).

Other decentralised privacy-preserving payment systems adopt this auditability approach such as [3],[37],[12].

Specifications for a responsive and private payment system. In this paper, we introduce a new specification (inspired by the formalism recently proposed by Albouy et al. [1]), as existing ones do not address our particular problem. They describe either private payment systems that cannot be implemented in asynchronous networks [18], asynchronous payment systems that lack anonymity [25, 13, 32], or non-sequential specifications that do not align with how specifications are typically defined in the distributed systems literature [38, 39, 42].

3 Model

3.1 Participants and adversary

The system is composed of two types of participants:

- A set \mathcal{U} of U users of the payment system.

- A set \mathcal{V} of $N = 3f + 1$ *validators*.

Here f denotes the maximum number of validators that can go *Byzantine*, i.e., deviate from the protocol. A non-Byzantine (faithful following the protocol) is called *correct*. Correct validators may, however, be *semi-honest* [19]: a correct party may try to learn as much as possible from the messages they receive from other parties, which may involve colluding and pooling their views together. In contrast, *honest* validators that are not trying to learn any information. The correct participants are thus following the protocol but they may exchange information with any other participant (Byzantine or not) to learn as much as possible. As we shall see, Byzantine participants pose challenges to all correctness aspects of our system (safety, liveness and privacy), while semi-honest participants affect only privacy.

The adversary \mathcal{A} thus controls any number of users and all validators, but at most f validators can deviate from the protocol. \mathcal{A} can be seen as a hybrid adversary between honest and semi-honest. We assume that \mathcal{A} is *static* and *network-ignorant*: the set of Byzantine participants controlled by \mathcal{A} is chosen a priori, and \mathcal{A} has no information about message delay between the validators and the other participants. It can delay the messages but must eventually deliver them. Conventionally, the adversary \mathcal{A} is computationally bounded. More precisely, \mathcal{A} is *probabilistic polynomial-time* (PPT).

Every participant is provided with a pair of distinct public and secret addresses denoted apk and ask . These addresses are used to identify users and validators. The public addresses of the validators are known to every participant. Otherwise, a user only needs to know the identifiers of the users she engages in transfers with.

3.2 Network and communications

Concerning the network, we assume *asynchronous* but *reliable* communication. The participants can communicate via *anonymous*, *secure*, and *asynchronous network channels*. The channels do not modify or create messages. If a correct participants sends a message to a correct one, the message is eventually received, though we assume no bounds on the communication delays. The sender's identity is not known to the receiver, but the receiver can still respond to the sender. No other participant can tell who is the sender or the receiver, or tamper with the message content. We could use a network based on Syverson et al. [41] work. We consider that the latency distribution is the same for all the channels.

3.3 Cryptographic tools

Our protocol makes use of several cryptographic tools that we list below. Due to the space constraints, we only give informal definitions and refer to [23] for more details. Also, in the algorithms mentioned below, we omit the security parameters taken as inputs. Some primitives require a setup phase, which will be handled by a trusted third party \mathcal{T} . Note that these setups could be carried out via MPC [19] between the validators.

(k, N) -Threshold Blind Signature Scheme . Tuple of algorithms $\Pi_{\text{sig}} = (\text{SETUP}_{\text{Sig}}, \text{BLIND}, \text{SIGN}, \text{UNBLIND}, \text{AGGREGATE}, \text{VERIFY}_{\text{Sig}})$ allowing to divide a secret key sk in n fragments $[sk_i]_{i=1}^N$ between n signers, such that valid signatures from any subset of k signers can be aggregated into a valid signature on behalf of the corresponding public key pk_{agg} . Each fragment sk_i has a corresponding public key pk_i so that a partial signature σ_i generated with sk_i can be verified with respect to pk_i . Moreover, the signers sign a blinding version \tilde{m} of a message m such that no information on m can be derived from \tilde{m} . A valid signature with respect to m can then be computed. Signature are unforgeable,

which means that no PPT adversary can forge a valid partial signature σ_i of a message m that correctly verifies against pk_i without the knowledge of sk_i . As a blind signature, this should not also be possible for a signer to eventually make the link between the signature is has issued, and the final signature (after UNBLIND and AGGREGATE).

- **SETUP_{Sig}**(k, N) $\rightarrow ([sk_i]_{i=1}^N, [pk_i]_{i=1}^N, pk_{agg})$: Randomised algorithm run by a trusted party. Takes as input threshold parameters (k, N) and returns a list of N signing keys $[sk_i]_{i=1}^N$ and one corresponding verification key pk_{agg} .
- **BLIND**(m, b) $\rightarrow (\tilde{m})$: Takes as input a message m and a blinding factor b , returns a blinded message \tilde{m} .
- **SIGN**(\tilde{m}, sk_i) $\rightarrow \tilde{\sigma}_i$: Takes as input a blinded message \tilde{m} and a secret key sk_i , and returns a partial blinded signature $\tilde{\sigma}_i$.
- **UNBLIND**($\tilde{m}, \tilde{\sigma}_i, b$) $\rightarrow \sigma_i$: Takes as input a blinded message \tilde{m} , the blinding factor b used to blind m and the corresponding partial blinded signature $\tilde{\sigma}_i$, and returns a valid partial signature σ_i for the original message m .
- **AGGREGATE**($[\sigma_i]_{i=1}^k$) $\rightarrow \sigma$: Takes as input a list of k signatures $[\sigma_i]_{i=1}^k$ and produce a signature σ such that σ is a valid signature on behalf of the public key pk_{agg} if and only if all σ_i are valid signatures for distinct public keys pk_i .
- **VERIFY_{Sig}**(m, σ, pk) $\rightarrow b$: Takes as input a partial or aggregated signature σ of a message m and returns a bit b of value 1 if σ is valid with respect to pk and 0 otherwise. The signature can be a partial or aggregated signature.

Collision-Resistant and Preimage-Resistant Pseudorandom Function Family [23].

A family of functions $\text{PRF} = \{\text{PRF}_s : \{0, 1\}^* \rightarrow \{0, 1\}^{O(|s|)}\}_s$, where s denotes a seed, computationally indistinguishable from a random function family. *Collision-Resistance* means here that it is computationally infeasible to find couples $(s, x) \neq (s', x')$ such that $\text{PRF}_s(x) = \text{PRF}_{s'}(x')$. *Preimage-Resistance* means that it is computationally infeasible, given y , to find (s, x) such that $\text{PRF}_s(x) = y$.

Non-Interactive Zero-Knowledge Proof (NIZK) [23]. A tuple of algorithms $\Pi_{\text{proof}} = (\text{SETUP}_{\text{NIZK}}, \text{PROVE}, \text{VERIFY}_{\text{NIZK}})$ allowing a *prover* to prove to a *verifier* that, given a statement defined by an NP relation $\mathcal{R}(a, b)$ and an instance **public_input**, she knows a witness **private_input** such that $\mathcal{R}(\text{public_input}, \text{private_input})$.

- **SETUP_{NIZK}**(\mathcal{R}) $\rightarrow \text{pp}_{\text{NIZK}}$: Randomised algorithm run by a trusted party. Takes as input a relation \mathcal{R} and outputs public parameters pp_{NIZK} (also known as common reference string). These public parameters are taken as input by the two following algorithms **PROVE** and **VERIFY**, but we omit it to lighten the notation.
- **PROVE**(**public_input**, **private_input**, pp_{NIZK}) $\rightarrow \pi$: Randomised algorithm. Takes as inputs instance and witness. Outputs a proof π such that : $\mathcal{R}(\text{public_input}, \text{private_input})$.
- **VERIFY_{NIZK}**(**public_input**, pp_{NIZK} , π) $\rightarrow b$: Takes as input an instance **public_input** and a proof π . Outputs 1 if the proof is valid and 0 otherwise.

Incremental commitment scheme: A tuple of algorithms $\Pi_{\text{com}} = (\text{COM}, \text{INCR})$ that allows us, given a sequence of messages $[m_i]_{i=1}^n$ and a sequence of randomness $[r_i]_{i=1}^n$, to produce a *commitment* c . The commitment is *hiding*, i.e., no information on $[m_i]_{i=1}^n$ can be derived from c without prior knowledge on $[r_i]_{i=1}^n$. The commitment is also *binding*, meaning that given a sequence of couples $[(m_i, r_i)]_{i=1}^n$ that commits to a value c , it should be computationally infeasible for a PPT adversary to compute $[(m'_i, r'_i)]_{i=1}^n$ such that $[(m'_i, r'_i)]_{i=1}^n \neq [(m_i, r_i)]_{i=1}^n$ also commits to c . The commitment scheme is also *incremental*: let us consider a sequence $[(m_i, r_i)]_{i=1}^n$, its commitment c and a couple

(m', r') . Given the knowledge of c, m' and r' , it is possible to compute c' that commits to $[(m_1, r_1), (m_2, r_2), \dots, (m_n, r_n), (m', r')]$:

- $\text{COM}([(m_i, r_i)]_{i=1}^n) \rightarrow c$ takes as input a sequence of messages and randomness and returns the corresponding commitment. COM is hiding and binding.
- $\text{INCR}(c, m', r') \rightarrow c'$ takes as input a commitment c of a sequence $[(m_i, r_i)]_{i=1}^n$, a new message m' and a randomness r' . It returns c' , the commitment of the sequence $[(m_1, r_1), (m_2, r_2), \dots, (m_n, r_n), (m', r')]$.

4 Fully private asset transfer (FPAT)

State and interface. At the abstract level, the state of the FPAT object is represented as an array of U positive integer values $[v_k]_{k=1}^U$, one for each user, interpreted as the current balances of the users' accounts. Let $[v_k^{\text{init}}]_{k=1}^U$ be the *initial state* of the object. The object exports two operations: *transfer* and *balance*. Assuming that a user u invokes an operation, they are defined as follows:

- $\text{transfer}_u(v, w)/r$ takes as inputs a value v and a user identifier w . It transfers the amount v from u to w : updates a state $[v_k]_{k=1}^U$ to the state $[v'_k]_{k=1}^U$ where $v'_k = v_k + v$ if $k = w \wedge u \neq w$, $v'_k = v_k - v$ if $k = u \wedge u \neq w$ and $v'_k = v_k$ otherwise. It returns a binary response $r = \text{confirm}$ if it succeeds and $r = \text{fail}$ otherwise.
- $\text{balance}_u()/v_u$ takes no inputs and returns the value v_u stored at location u of the state $[v_k]_{k=1}^U$.

Let \mathcal{O} be a set of FPAT *operations*—invocations of *transfer* and *balances* provided with matching responses, each associated with a distinct user. Let $\text{transfer}_u(*, *)/C$ (resp., $\text{transfer}_u(*, *)/F$) denotes a transfer operation invoked by a user u that returns *confirm* (resp. *fail*). For each user u , we define a function $\text{total}_u(\mathcal{O})$ as follows:

$$\text{total}_u(\mathcal{O}) = v_u^{\text{init}} + \sum_{\text{transfer}_*(v, u)/C \in \mathcal{O}} v - \sum_{\text{transfer}_u(v, *)/C \in \mathcal{O}} v$$

$\text{total}_u(\mathcal{O})$ is thus defined as the current *balance* of u after all successful transfers in \mathcal{O} complete: the initial amount owned by u , minus all the funds sent by u plus all the funds received by u in the set of operation \mathcal{O} .

A *sequential history* S (of FPAT) is a totally ordered set of FPAT operations, let \prec_S be this order. Let $S|_u$ denotes the sub-sequence of S consisting of the events of user u . We say that S is *legal* if:

1. $\forall o = \text{transfer}_u(v, w)/C \in S \quad v \leq \text{total}_u(\{o' \in S : o' \prec_S o\})$
2. $\forall o = \text{balance}_u()/v \in S \quad v \leq \text{total}_u(\{o' \in S : o' \prec_S o\})$
3. If an operation $\text{balance}_u()$ returning v_1 directly precedes a $\text{transfer}_u(v_2, w)$ operation in $S|_u$, and $v_2 \leq v_1$, then the transfer operation cannot return *fail*.

Histories and serializations. Consider an *execution* of a FPAT algorithm: a sequence of *events* produced by the algorithm, such as invocations and responses of FPAT operations, sending and receiving messages, etc. A *local history* L_i of a user i is the sequence of *operations* invoked by i in that execution. We assume that correct users are *well-formed*—they never invoke a new operation before the previous one returns, and thus if u is correct, then L_u is sequential. In case the last operation of L_u is *incomplete* (not followed by a response), we can add any matching response and get a *completion* of L_u . Now a *history* H is a vector $(L_1, L_2, \dots, L_{|U|})$ of local histories, one for each user. Notice that if the history is produced

by an execution of a FPAT algorithm, only the local histories of correct users are of interest for us: a Byzantine user is not obliged to follow the protocol.

A sequential (FPAT) history S is a *serialization* of a history $H = (L_1, L_2, \dots, L_{|\mathcal{U}|})$ if for each *correct* user u , there exists \hat{L}_u , a completion of L_u , such that $S|_u = \hat{L}_u$ (\prec_S respects the local order $\prec_{\hat{L}_u}$). S can be seen as a *global interpretation* of the local histories of correct users. Notice that we allow any operations to be executed by Byzantine users in S , as long as it “makes sense” to the correct ones.

FPAT-Safety. An implementation of a FPAT-object is *FPAT-Safe* if and only if, for any finite history of execution H it produces, there exists a serialisation S of H which is legal.

FPAT-Liveness. Liveness ensures that (1) every operation invoked by a correct user eventually completes and (2) considering two correct users u and w , if w transfers money to u , then u eventually receives this money. Let us consider the existence of a global time during the execution of an implemented FPAT object. An implementation of a FPAT-object is *FPAT-Live* if:

1. All *transfer* and *balance* operations terminate for correct users.
2. Consider a correct user u . For each operation $transfer_*(*, u)$ completed during the execution at time t , there exists a time $t' \geq t$ such that any operation *balance* inserted at time $t'' \geq t'$ will return a value v such that:

$$v \geq \sum_{\substack{transfer_*(v_+, u) \\ \text{invoked by correct users} \\ \text{completed before time } t}} v_+ - \sum_{\substack{transfer_u(v_-, *) \\ \text{completed before time } t''}} v_-$$

FPAT-privacy. We capture the privacy guarantees of FPAT through a *distinguishing game* \mathcal{G}^{priv} described in details in Appendix C.3. *FPAT-privacy* has important implications that can be expressed as a collection of properties we use here. Let ϵ be a negligible function and λ the security parameter. Consider a protocol execution, let H be its history and \mathcal{U}_{Byz} the set of users controlled by the adversary, among the U users of the system. Let $o = transfer_u(v, w) \in H$ be any *transfer* operation such that u is honest. Then for each guess (u', v', w') made by an adversary with no prior knowledge, where u' is the payment sender, v' is the value, and w' is the payment recipient, the following properties hold:

1. **Sender-anonymity:** $\mathbb{P}(u' = u) \leq \frac{1}{U - |\mathcal{U}_{Byz}|} + \epsilon(\lambda)$, i.e., the adversary only knows that the sender is not itself.
2. **Receiver-anonymity:** If w is honest, then $\mathbb{P}(w' = w) \leq \frac{1}{U - |\mathcal{U}_{Byz}|} + \epsilon(\lambda)$, i.e., the adversary only knows that the receiver is not itself.
3. **Confidentiality:** If w is honest, then $\mathbb{P}(v' = v) \leq \epsilon(\lambda)$, i.e., the adversary cannot guess the amount of the transaction.

These three properties together constitute **full privacy**. Additionally, *FPAT-privacy*, as captured by the *distinguishing game* \mathcal{G}^{priv} , implies **unlinkability**: given two transfers, no adversary can determine whether the sender, receiver, or amount is the same in both transfers.

5 PaxPay Protocol

We overview our FPAT implementation below and delegate detailed algorithms and proofs of correctness to Appendices A and C.

Setup. Every user is identified by her public address apk that is derived from a secret address ask as follows: $apk = \text{PRF}_{ask}(0)$ ⁴

The protocol uses a $(2f + 1, N)$ -threshold blind signature scheme, as defined in Section 3.3. The secret signing keys $[sk_i]_{i=1}^N$ are held by each of the validators. The aggregated public key is denoted by pk_{agg} . The protocol will also make use of NIZK, as defined in Section 3.3. The setup for these primitives is described in Algorithm 1 in Appendix A.

Coin structure A *coin* is a tuple $\mathbf{c} = (v, apk, \rho)$ with:

- v the (integer) value of the coin.
- apk the public address of the owner of the coin.
- ρ the seed of the coin, from which the serial number is derived.

Coins are similar to unspent transactions (UTXO) in Bitcoin: to make a payment, a user “spends” *old* coins and creates *new* coins whose owners are the payment recipients.

Coin validity Using a quorum of validators signatures (inspired by Byzantine Consistent Broadcast), we decide whether a coin is valid or not. In concrete terms, a coin is valid if it has been signed by $2f + 1$ validators. During a transfer, several old coins $[\mathbf{c}_i^{old}]_{i \in [1, n]}$ are spent to create new coins $[\mathbf{c}_j^{new}]_{j \in [1, m]}$. To make sure that a coin is not spent twice, a unique *serial number* is derived from the coin’s seed ρ as $sn = \text{PRF}_{ask}(\rho)$. Each validator maintains a list `snList` of the old coins’ serial numbers. The coins $[\mathbf{c}_i^{old}]_{i \in [1, n]}$ are considered spent when a quorum of $2f + 1$ validators have appended the corresponding serial numbers $[sn_i^{old}]_{i \in [1, n]}$ to their `snList`.

Intuitively, as the computation of sn_i^{old} is using the PRF and the secret address of the payer, no party can link sn_i^{old} to ρ_i^{old} or apk_i^{old} , and thus the sender-anonymity property is preserved. Algorithm 3 initializes the balances for the initial set of users by creating coins for each of them.⁵

Local variables. User storage consists of:

- apk/ask : its public/secret address pair
- `coinList` = $[(\mathbf{c}_i, \sigma_i)]_i$: the set of coins owned by the user associated with the aggregated signature of each coin.
- `rhoList`: the list of all seed ρ corresponding to coins received by the user

Validator storage consists of:

- sk : its secret signing key
- `snList`: the list of serial numbers identifying spent coins.
- `signedCoinList`: the list of all blinded coins signed by the replica.

Transfer. Assuming a user has enough funds, she can issue payments to several recipients $[apk_j^{new}]_{j=1}^m$ in one transfer by sending v_j^{new} to apk_j^{new} . For more details, see Algorithm 6 in Appendix A. The user first chooses n (old) coins denoted $[\mathbf{c}_i^{old}]_{i=1}^n$ and their signatures $[\sigma_i^{old}]_{i=1}^n$ from her `coinList`, where $\mathbf{c}_i^{old} = (v_i^{old}, apk_i^{old}, \rho_i^{old})$, such that the total value of the old coins does not fall below the total value payed in the transfer: $\sum_{i=1}^n v_i^{old} \geq \sum_{j=1}^m v_j^{new}$. (Otherwise, the transfer operation returns fail.) The user then computes the serial number $[sn_i^{old}]_{i=1}^n$ of every old coin as follows: $sn_i^{old} = \text{PRF}_{ask_i^{old}}(\rho_i^{old})$.

⁴ Algorithm 2 in Appendix A shows the address generation process by sampling a random ask and deriving the apk .

⁵ As discussed in Section 8, depending on the use cases of the system, this algorithm might not be executed. Instead, users might freely join the system and call a `Mint` algorithm that provides them with new coins.

The j new coins are then produced by the user according to the values and addresses to pay: $\forall j \in [1, m]$, $\mathbf{c}_j^{\text{new}} = (v_j^{\text{new}}, \text{apk}_j^{\text{new}}, \rho_j^{\text{new}})$. ρ_j^{new} is derived as $\rho_j^{\text{new}} = \text{PRF}_{\rho_{\text{seed}}}(\text{sn}_1^{\text{old}} || \dots || \text{sn}_n^{\text{old}} || j)$, where ρ_{seed} is sampled randomly. This binds each new coin to the old coins spent for its creation. This way, we make sure that the validators signing \mathbf{c}^{new} mark the same old coins as spent. Since the value of the chosen old coins might exceed the value of the new coins, the user also produces a *redeem coin* $\mathbf{c}_{m+1}^{\text{new}} = (v_{m+1}^{\text{new}}, \text{apk}_{m+1}^{\text{new}}, \rho_{m+1}^{\text{new}})$, with $\text{apk}_{m+1}^{\text{new}} = \text{apk}$ and v_{m+1}^{new} the exceeding value. The coins $[\mathbf{c}_j^{\text{new}}]_{j=1}^{m+1}$ are then *blinded*. The blinding of $\mathbf{c}_j^{\text{new}}$ is denoted $\tilde{\mathbf{c}}_j^{\text{new}} = \text{BLIND}(\mathbf{c}_j^{\text{new}}, b_j)$, where b_j is a randomly sampled blinding factor.

The sender of the payment then computes an NIZK with:

- **public_input** : $([\text{sn}_i^{\text{old}}]_{i=1}^n, [\tilde{\mathbf{c}}_j^{\text{new}}]_{j=1}^{m+1})$
- **private_input** : $([\mathbf{c}_i^{\text{old}}]_{i=1}^n, [\sigma_i^{\text{old}}]_{i=1}^n, [\text{ask}_i^{\text{old}}]_{i=1}^n, [\mathbf{c}_j^{\text{new}}]_{j=1}^{m+1}, [b_j]_{j=1}^{m+1}, \rho_{\text{seed}})$

This NIZK π_{transfer} proves the following relations:

1. Serial numbers $[\text{sn}_i^{\text{old}}]_{i=1}^n$ are correctly derived from the old coins $[\mathbf{c}_i^{\text{old}}]_{i=1}^n$.
2. New coins $[\mathbf{c}_j^{\text{new}}]_{j=1}^{m+1}$ are correctly derived from the old coins $[\mathbf{c}_i^{\text{old}}]_{i=1}^n$.
3. Blinded coins $[\tilde{\mathbf{c}}_j^{\text{new}}]_{j=1}^{m+1}$ are correctly derived from $[\mathbf{c}_j^{\text{new}}]_{j=1}^{m+1}$ and $[b_j]_{j=1}^{m+1}$.
4. Signatures $[\sigma_i^{\text{old}}]_{i=1}^n$ of the coin $[\mathbf{c}_i^{\text{old}}]_{i=1}^n$ are correct.
5. Private addresses $[\text{ask}_i^{\text{old}}]_{i=1}^n$ match the public address $[\text{apk}_i^{\text{old}}]_{i=1}^n$.
6. The sum of the values of the old coins $[\mathbf{c}_i^{\text{old}}]_{i=1}^n$ equals the sum of the values of the new coins $[\mathbf{c}_j^{\text{new}}]_{j=1}^{m+1}$.

The user can now send the NIZK π_{transfer} along with the public inputs to all the validators. Algorithm 12 describes the algorithm run by the validators. Once a validator receives the proof:

1. It checks that none of the $[\text{sn}_i^{\text{old}}]_{i=1}^n$ appears in its `snList`;
2. It checks that the proof π is correct;
3. If the last two conditions are fulfilled, it adds all $[\text{sn}_i^{\text{old}}]_{i=1}^n$ to `snList`, add all $[\tilde{\mathbf{c}}_j^{\text{new}}]_{j=1}^{m+1}$ to `signedCoinList`, it signs all the coins $[\tilde{\mathbf{c}}_j^{\text{new}}]_{j=1}^{m+1}$ and returns the blinded partial signatures.
4. If any of the previous conditions is not fulfilled, the validator checks if the blinded coins $[\tilde{\mathbf{c}}_j^{\text{new}}]_{j=1}^{m+1}$ appear in `signedCoinList`. If they all do, then the validator still sends back the signatures. This is done because a Byzantine validator p might receive a transfer request from a user u and send it to other validators using a private channel. As a result, other validators might answer p before answering u , preventing u from receiving the signatures while the old coins are actually spent (their serial numbers would have been added to `snList` already).

Once the user receives $2f + 1$ valid partial signatures for the coins $[\tilde{\mathbf{c}}_j^{\text{new}}]_{j=1}^{m+1}$ from $2f + 1$ validators, she can unblind and aggregate them to form $m + 1$ signatures $[\sigma_j]_{j=1}^{m+1}$ that are valid signatures for $[\mathbf{c}_j^{\text{new}}]_{j=1}^{m+1}$. She can now send each couple $(\mathbf{c}_j^{\text{new}}, \sigma_j^{\text{new}})$ to $\text{apk}_j^{\text{new}}$. The user $\text{apk}_j^{\text{new}}$ only accepts the payment if $\rho_j^{\text{new}} \notin \text{rhoList}_{\text{apk}_j^{\text{new}}}$ and $\text{VERIFY}_{\text{Sig}}(\mathbf{c}_j^{\text{new}}, \sigma_j^{\text{new}}, \text{pk}_{\text{agg}}) = 1$.⁶

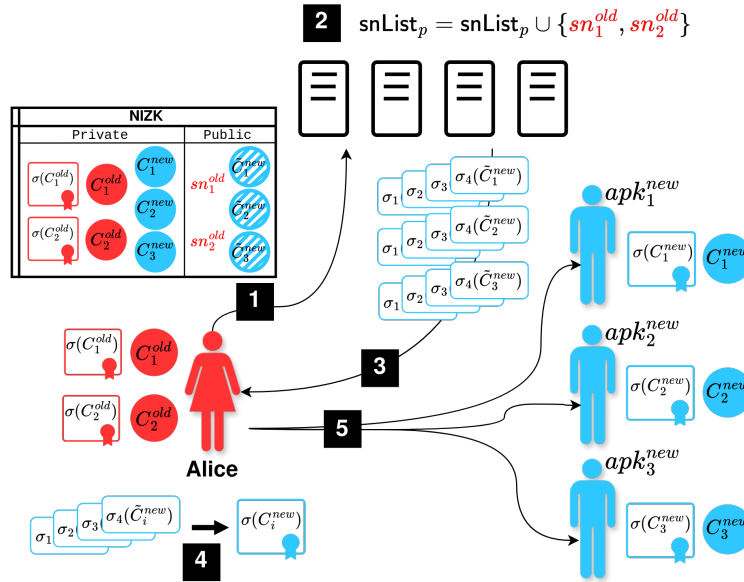
Double spending. Consider a coin \mathbf{c}^{old} that has been spent. Hence, a set \mathcal{V}_1 of $2f + 1$ validators have claimed to have added sn^{old} to their `snList`, and at least $f + 1$ out of them

⁶ For convenience, we allow the `TRANSFER` algorithm to send money to several recipients in a single call, in contrast with the specification, which allows only one recipient at a time. As explained in the proof in Appendix C, this more generic transfer can be seen as a batch of successive transfer calls, each destined to a single user.

are correct. If a Byzantine user tries to spend c^{old} again, it should collect confirmations from a set \mathcal{V}_2 of $2f + 1$ validators that will have to add sn^{old} in their $snList$ again. Since there are $3f + 1$ validators in total, \mathcal{V}_1 and \mathcal{V}_2 have at least one correct validator in common. It must refuse to sign the second transaction because sn^{old} is already in its $snList$, so double spending is prevented and FPAT-Safety guaranteed (see Appendix C for the proof).

Example. Figure 1 depicts a transfer:

1. Alice wants to pay 3 recipients with 2 coins. She generates 3 blinded coins and a matching NIZK, and sends them to the validators.
2. Each validator checks the validity of the NIZK and that the old coins are not in its $snList$, and, if so, sign the blinded coins and add the serial numbers to their $snList$.
3. Alice receives the signatures.
4. For each coin: once $2f + 1$ blinded partial signatures are received, Alice unblinds and then aggregates them into one aggregated signature.
5. Alice sends the coin tuples and their signatures to their recipients.



■ **Figure 1** PaxPay example: Alice pays 3 recipients with 2 old coins

6 Regulatory enforcement

PaxPay is designed as a private payment system. As in PEReDi [38], PRCash [45] or Platypus [46], the protocol can be enhanced to be *regulation-compliant*. The use of succinct proofs limits the impact of regulatory enforcement on the system's performance. This section describes how to build such a regulatory enforcement on top of the protocol described above.

Our regulatory enforcement is achieved via *compliance coins*. Each user owns its unique compliance coin. The coin commits to some data related to the user and all his transactions. The user must attach the compliance coin to each of its transactions as an input and the validators will sign the updated compliance coin. Well-formedness of the updated compliance coin is proved through NIZK. As for a classical coin, the old compliance coin is spent by revealing its serial number. As a result, a user has one valid compliance coin at a

time and this coin acts as an append-only tracing mechanism. We introduce the following regulation features, shown in Table 1:

- *Limited held amount per user*: Users cannot hold more than V_{held}^{max} .
- *Full asset tracing*: A trusted authority (centralized or decentralized) can reveal the content of transaction to trace back assets and users activities.
- *Limited spendable amount per tx*: Users cannot spend more than V_{sent}^{max} in one transfer.
- *Limited spendable amount in total*: Users cannot spend more than V_{total}^{max} in total.
- *Sanctions*: Users in the sanction list cannot send or receive funds.
- *Provable transaction history*: A user can prove arbitrary statements about her transactions. For instance, reveal all of her transactions and prove that the revealed set is complete (no transactions are missing).

Our regulatory construction enforces *Limited spendable amount per tx*, *Limited spendable amount in total*, *Sanction list* and *Provable transaction history*. V_{sent}^{max} , V_{total}^{max} and *SancList* are public parameters chosen and potentially updated by the regulator. The sanction list *SancList* is represented as a sorted Merkle tree, which allows efficient proof of non-membership.

Compliance coin The compliance coin of a user u is defined as a tuple $cc_u = (apk, \rho, v, com)$ where:

- apk is u 's sole public address
- ρ the coin seed, as for a normal coin
- v is the total amount of money sent so far by u
- com is the commitment of the list of all the transfer done by u so far. For each coin created, it commits the tuple (apk, v) with apk the public key of the receiver and v the value sent. It uses the incremental commitment introduced in Section 3.3

com allows to trace u 's activities and allows u to prove additional statements on her transaction history in case of an update of the regulation.

Registration Adding a user registration process is essential for regulatory enforcement, to comply with the *know-your-customer* (KYC) and *customer-due-diligence* (CDD) checks. Additionally, it ensures the uniqueness of the compliance coin for each physical user.

To this end, the validator storage is provided with a new list `registeredList`, the list of the enrolled users associated with their public key. `registeredList` is used to make sure that a physical user can only register once on the system. The registration process is as follows for the user u :

1. Generate a random ρ and computes $cc_u = (apk_u, \rho, 0, 0)$ and its blinding \tilde{cc}_u (with blinding factor b).
2. Computes an NIZK $\pi_{register}$ that takes (\tilde{cc}_u, apk_u) as public input and (ask_u, cc_u, b) as private input. It proves that:
 - a. $\tilde{cc}_u = \text{BLIND}((apk_u, \rho, 0, 0), b)$.
 - b. ask_u and apk_u form a correct secret/public address pair.
3. Send to each validator $\pi_{register}$ and (\tilde{cc}_u, apk_u) .
4. Upon correct KYC and CDD proving u 's identity, each validator check if $\pi_{register}$ is correct and that: $(u, *) \notin \text{registeredList}$ (u has registered no public address yet). If so, they sign \tilde{cc}_u , send it back to u , and add (u, apk_u) to `registeredList`.
5. Once u has received $2f + 1$ partial signature, she can unblind and aggregates the partial signatures to form a valid signature for cc_u .

The detailed algorithm is described in Appendix A as Algorithm 11.

Transfer Section 5 described how a transfer is handled by a user u , by spending n old coins $[c_i^{old}]_{i=1}^n$ and creating $m + 1$ new coins $[c_j^{new}]_{j=1}^{m+1}$ to pay m recipients (m coins are created to pay the recipient and a redeem coin is also created with index $m + 1$ to get the excess of money back to u). Now the user also has to update a compliance coin at each payment and prove that their transfer is compliant. The following additional steps are thus required. She generates m random values: $[r_j]_{j=1}^m$. The user provides an additional proof π_{comply} . This proof takes the following public inputs:

- \tilde{c}_u^{new} the blinding of the new compliance coin
- sn_{cc}^{old} the serial number of the old compliance
- V_{sent}^{max} the maximum amount that can be transferred in one transfer
- V_{total}^{max} the total amount of money that u can transfer in her use of the system
- $SancList$ the list of addresses under sanctions

π_{comply} takes the following as private input:

- $cc^{old} = (apk_{cc}^{old}, \rho_{cc}^{old}, v_{cc}^{old}, com^{old})$ the old compliance coin
- $cc^{new} = (apk_{cc}^{new}, \rho_{cc}^{new}, v_{cc}^{new}, com^{new})$ the new compliance coin
- ask_{cc}^{old} the secret address corresponding to apk_{cc}^{old}

π_{comply} should verify the following clauses:

1. $\forall i \in [1, n], apk_i^{old} = apk_{cc}^{old}$
2. $apk_{cc}^{old} = \text{PRF}_{ask_{cc}^{old}}(0)$
3. $apk_{m+1}^{new} = apk_{cc}^{old}$
4. $apk_{cc}^{new} = apk_{cc}^{old}$
5. $\rho_{cc}^{new} = \text{PRF}_{\rho_{seed}}(sn_1^{old} || \dots || sn_n^{old} || m + 2)$
6. $v_{cc}^{new} = v_{cc}^{old} + \sum_{1 \leq j \leq m} v_j^{new}$
7. $com_0 = com^{old} \wedge \forall j \in [1, m], com_j = \text{INCR}(com_{j-1}, apk_j^{new}, v_j^{new}, r_j) \wedge com^{new} = com_m$
8. $\sum_{1 \leq j \leq m} v_j^{new} \leq V_{sent}^{max}$
9. $v_{cc}^{new} \leq V_{total}^{max}$
10. $apk_{cc}^{old} \notin SancList$
11. $\forall j \in [1, m], apk_j^{new} \notin SancList$

Conditions (1) ensures that all the input coins belong to the owner of the input compliance coin. Condition (2) ensures that u , the user that invokes *transfer*, is the owner of the compliance coin. Condition (3) ensures that the redeem coin will belong to u . Conditions (4), (5), (6) and (7) ensure that the new compliance coin is well-formed. Condition (8), (9) and (10) ensure that the transfer is compliant. (8) verifies that the amount of the transfer does not exceed that maximum allowed for a single transfer. (9) verifies that the total amount of money sent by u does not exceed that maximum amount. (10) verifies that the sender is not a user under sanctions. (11) verifies that none of the receivers of the transfer is under sanction.

The user storage is also augmented with a list *comList*, in which she stores the tuples (apk, v, r) that she has used at each transfer. At the end of the transfer described above, she thus append to *comList* the following: $\forall j, j \in [1, m], (apk_j^{new}, v_j^{new}, r_j)$. She can later use *comList* and her compliance coin to prove any statement about her transaction.

Upon receiving correct proofs $(\pi_{comply}, \pi_{transfer})$ and their public parameters, the validators execute the same algorithm as for a non regulated transfer, except that they

verify both π_{comply} and $\pi_{transfer}$. They can then sign the news coins and then send the signature back. The detailed algorithm is described in Appendix A as Algorithm 9.

7 PaxPay implementation and performance

PaxPay implementation in Golang is available on Github. We developed with two implementations, with and without regulation support. Below we overview the cryptographic tools we employed and report on the performance of our protocols.

7.1 Cryptographic building blocks

NIZK. NIZK proofs are implemented using the Groth16 [24] protocol. Groth16 allows for constructing succinct non-interactive arguments of knowledge (SNARKs) with constant verification complexity, regardless of the complexity of the relation \mathcal{R} it attests to. The relation \mathcal{R} is represented by an arithmetic circuit. We instantiate this scheme with the BW6-761 [27] curve, designed to cycle with the BL12-377 [27] curve, allowing for efficient pairing verification over BL12-377 within the SNARK.

In the regulated version, each transfer requires two NIZK. For performance reasons, these two proofs $\pi_{transfer}$ and π_{comply} are merged into a single proof with a unique circuit. The proof of non-membership in the merkle tree is implemented as follows: the sanction list has the following form: $SancList = [s_1, s_2, \dots, s_\xi]$. $SancList$ is supposed to be published sorted by the regulator. To verify the non-inclusion of an address apk in the list, we prove that there exist s_i and s_j such that:

- s_i and s_j are elements of $SancList$.
- $j = i + 1$ (s_i and s_j are consecutive elements of $SancList$).
- $s_i < apk < s_j$ (s_1 and s_ξ are set to $0x00..00$ and $0xFF..FF$ to make sure this constraint can always be fulfilled).

Hash functions and PRF. To optimize the computational cost of the proving algorithm in SNARK, we rely on MiMC [2]. MiMC is an arithmetization-oriented, collision-resistant and preimage-resistant hash function that is efficiently represented as arithmetic circuits. We use it to derive the seeds ρ of created coins from the serial numbers sn of spent coins. The pseudorandom function family $PRF_x()$ is derived from a MiMC function H as $PRF_x(m) := H(m||x)$.⁷

Incremental commitment scheme. The incremental commitment scheme is also derived from the MiMC function H . Given a commitment com^{old} and a message m to be added to the commitment with a randomness r , $INCR(com^{old}, m, r)$ is defined as follows: $INCR(com^{old}, m, r) = H(com^{old}||m||r)$. The commitment function COM is then an iteration of increments $INCR$ over the sequence of messages and randomnesses provided.

(k, N)-threshold blind signature. Our signature scheme is based on a slightly modified version of the Coconut scheme [40], which itself is a variant of the Pointcheval and Sanders signature scheme [36]. Indeed, certain checks originally performed using a sigma protocol in Coconut are instead handled in the Groth16 proof in our implementation, reducing the overall verification complexity. We instantiate this scheme with the BLS12-377 curve so that

⁷ The natural way to implement a PRF from a hash function is to consider $PRF_x := H(m||x)$, however, replacing concatenation by an addition in the field does not compromise security in our case while reducing the computational cost.

the verification algorithm, which involves pairings, is efficient in the Groth16 proof. (See Appendix B for more details).

7.2 Performance analysis

We now report on our comparative performance analysis.

NIZK Benchmark. We benchmarked our NIZK, implemented using a Groth16 SNARK, on an Intel i7 @ 2.6 GHz CPU running Ubuntu 22.04. The results are summarized in Table 2. Each value is based on the average of 10 proving and verification runs. A comparison with related protocols is provided in Table 1. The data in this table was gathered from the respective research papers, as the source codes of these protocols were not always publicly available. However, the CPU used for our benchmark has comparable performance to the machines reported in those works: they all use Intel Core i7 CPUs except for Zef and UTT. For Zef and UTT, AWS EC2 instances were used for the benchmark. These two types of instance have similar CPUs (Intel Xeon Platinum 8175 and 8124). To ensure a fair comparison, we also ran our benchmark on the same AWS instances, with detailed results provided in Table 3 and Table 4 in Appendix E.

PaxPay has two implementations: with and without the regulatory feature. The benchmark was conducted on both. The regulated version, running on a single core, serves as a reference point. The results indicate that our SNARK verification time is notably short, taking only 5.6 ms. As shown in Table 1, this verification time is between 8 and 100 times shorter than the NIZK verification times reported in other works (except for Zcash, which takes approximately the same time but suffers from other major issues discussed in Section 8). The verification time is a key metric since slow verification limits the transaction throughput of the system.

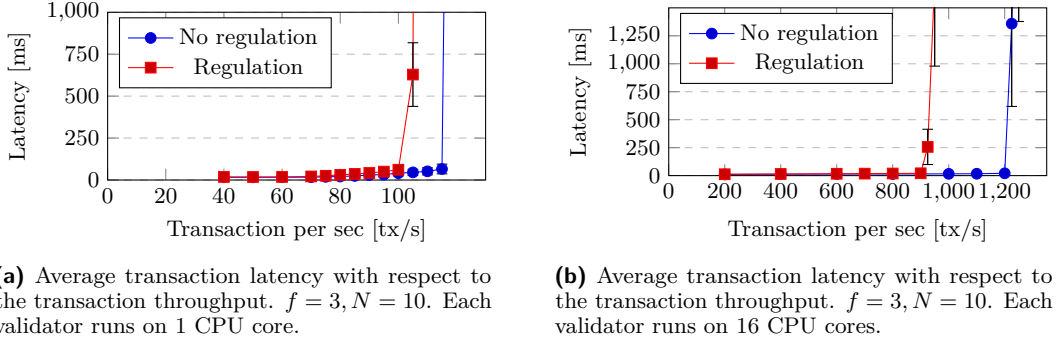
This fast verification comes at the cost of relatively slow proving, requiring nearly 7 s to generate a proof for a transfer on a single core. However, with 6 cores, the proving time drops to 2 s.

■ **Table 2** SNARK Proving and Verification Times on an Intel i7 2.6 GHz CPU, running Ubuntu 22.04.

	Regulated		Not Regulated	
	1 Core	6 Cores	1 Core	6 Cores
Proving time (ms)	6959 (± 13)	2001 (± 13)	3080 (± 6)	882 (± 5)
Verification time (ms)	5.61 (± 0.20)	5.38 (± 0.28)	5.22 (± 0.15)	4.71 (± 0.85)

Latency test and transaction throughput Apart from Zcash, which is already deployed in real-world use, the only payment systems in the related work to report on their latency and throughput are Zef [6] and UTT [42]. The performance analyses of UTT and Zef were performed on different types of AWS EC2 instances (c5.4xlarge for UTT and m5.8xlarge for Zef). Since the micro-benchmarks show better performance for PaxPay on UTT’s EC2 instance than on the one used by Zef, we chose to use the same instance type as in the Zef paper for consistency.⁸

⁸ We were unable to build and test the UTT code due to dependencies that broke over time. Since the



■ **Figure 2** Latency tests on AWS machines within the same region.

We thus run our PaxPay implementation on multiple m5.8xlarge instances, assigning one validator per instance. For each version of PaxPay (regulated and non-regulated), we conducted two tests: one with each validator running on a single CPU core, and another with each validator using 16 CPU cores. The results are presented in Figures 2a and 2b. We consider that the system support a given throughput if the corresponding transaction latency is smaller than 500 ms. Our regulated implementation achieves the throughput of 100 **tx/s** (transactions per second) in the single-core setup and 925 **tx/s** in the 16-core setup. Under the same conditions, Zef processes 5 **tx/s** and 88 **tx/s**, respectively. UTT processes 235 **tx/s** using 16 cores.⁹ For the non-regulated PaxPay implementation, the throughput reaches 115 **tx/s** in the single-core setup and 1200 **tx/s** in the 16-core setup.

These promising results can be attributed to the low verification time of the SNARK presented in Table 2, Table 3 and Table 4. In the reference setting (regulated implementation with 1-core validators), the verification of the SNARK takes 9 **ms** in our implementation while it takes 142 **ms** in Zef with the same setting (Table 3). The difference in performance between the regulated and non-regulated versions is due to the additional public parameters in the regulated SNARK. This requires validators to perform more point multiplications on the elliptic curve used in the Groth16 construction.

To improve scalability, validators can be *sharded* (run on multiple parallel machines) as in Zef and UTT. The same approach could be applied to our system, though it has not been implemented. A load balancer could efficiently distribute transfer requests among the validators by assigning a specific range of serial numbers to each shard of a validator. Upon receiving a request, the load balancer would determine the appropriate shard to handle it based on these predefined ranges. Our experiments thus show that PaxPay is scalable. The throughput can be improved even further by allocating more computational power to the validators, either by using more CPU cores per validator or by distributing the workload across additional machines.

UTT authors chose an EC2 instance type on which PaxPay performs better compared to the m5.8xlarge instance, we consider the results reported in the UTT paper to be suitable for direct comparison.

⁹ As stated in Zef [6], each authority is distributed over multiple shards, each running in a single core. The 16-core result is extrapolated from the linear relationship between throughput and the number of shards per validator, as shown in their paper.

8 Discussion and comparison with related protocols

Comparative analysis. Table 1 provides a comparison between PaxPay and the related protocols. Lelantus appears to outperform Monero across all metrics in the table and offers larger anonymity sets. We therefore decided to omit Monero from the table. Similarly, Platypus was not included in the table, as we focus exclusively on decentralized payment systems. Details and explanations regarding the data presented in the table can be found in Appendix E (Table 3).

Privacy. As shown in Table 1, PaxPay provides the strongest privacy guarantees, comparable to those provided by Zcash, whereas some protocols offer reduced privacy guarantees. This is especially the case for Zef and PRCash.

To ensure full privacy, it is important to hide coin details when creating them with blind signatures. Yet, when the signatures are verified by validators, some users may still learn information on the payment.¹⁰ UTT circumvents this problem by using a re-randomizable signature scheme (where both the signature and the signed message can be randomized), but this solution increases the workload for verifying transactions. Our solution is to verify the signature within the NIZK, which also decrease the validators workload.

Model assumptions PaxPay is responsive, meaning it can be implemented in an asynchronous network. Among the related protocols only Zef, UTT, and PARScoin also provide responsiveness. All the four protocols are built on some form of Byzantine consistent broadcast. However, PARScoin requires correct validators (those who follow the protocol) to be honest, ensuring they do not share any information with the adversary. In contrast, PaxPay tolerates correct validators being semi-honest, meaning they can share any information they receive or transmit during the protocol's execution without compromising the system's privacy guarantees.

Regulation As displayed in Table 1, PaxPay does not support the *Limited held amount per user* feature. PEReDi [38] implements this feature, requiring synchrony between sender and receiver. PARScoin [39] claims a different approach: the sender initiates a transaction to reduce their balance, and the receiver can asynchronously claim funds later, provided their balance remains within the imposed limit. However, this method merely limits the amount a user can hold at a given time, as receivers can bypass it by spending excess funds before claiming the pending funds they have received. In practice, this mechanism has the same effect as setting a limit on the amount that can be spent in a single transaction. We believe that implementing a *Limited held amount per user* feature is particularly challenging in a private and asynchronous payment system. Such a feature would require that transfers atomically change the balances of both the sender and the receiver, to prove that receiver balance does not exceed a certain amount after executing the transfer. However, to affect the balance of a user, privacy typically requires this user to interact with another user or the validators, in order to provide some secret data known only to him (such as coin data or account commitments). Yet, asynchrony precludes live protocols from relying on interaction between senders and receivers. Indeed, in an asynchronous setting, the sender who gets no

¹⁰ Suppose that, as in Zef [6], we resort to only using blind signatures. Let a user A creates a coin c for a user B during a transfer o_1 . A requests the validators to sign the blinded coin \tilde{c} , and then sends the aggregated signature to B . B now spends c during a transfer o_2 , and in process reveals c and the (randomized) signature. Even though validators cannot reveal that the coin spent in o_2 was created in o_1 , A knows c and, thus, can deduce the sender of o_2 and a lower bound on the amount spent during o_2 .

response from the receiver cannot distinguish whether this absence of response is caused by network delays or by the receiver refusing to respond. Implementing *Limited Held Amount per User* seems hard when ensuring both asynchrony and privacy.

PaxPay also avoids *Full Asset Tracing* due to privacy risks. Instead, it allows users to generate privacy-preserving proofs about their transactions. Let us give a real life example. A user can prove that her account received funds from multiple addresses, each transfer being under a certain threshold, demonstrating legitimate income patterns. This proof could demonstrate that the income pattern of a business is legitimate and consistent with regular business operations, precluding money laundering (which might involve receiving a large sum from a single address).

Performance. As mentioned in Section 7.2, PaxPay outperforms all other protocols in terms of transaction throughput. This performance advantage is directly attributed to its fast NIZK verification, which surpasses all existing protocols. The only exception is Zcash, which also benefits from a fast NIZK verification but is constrained by the limitations of its underlying consensus protocol.

Additionally, the system’s scalability, that is achieved by increasing the computational power allocated to each validator, further strengthens its practical applicability. The Visa payment system processed an average of 7,388 transactions per second in 2024 [44], which could be supported by PaxPay by distributing each validator across 8 m5.8xlarge EC2 instances each¹¹.

As mentioned in Section 7.2, this high throughput comes at the cost of a relatively slow transfer proving, requiring between 2 and 7 seconds depending on the user computational power to prove a transaction. We believe that this delay is acceptable from the user’s perspective, especially since multiple payments can be aggregated into a single transfer. As a result, even if a user needs to make numerous payments, she can prove them all within a single transaction, requiring only one proof. It is worth noting that the statement being proven in such a case would be larger than for a proof of a single payment, which would still increase the proving time, though less significantly compared to generating a separate proof for each payment.

Use cases. Our system is versatile and adaptable to various use cases. It can function as a standalone payment system, a CBDC, or a scaling solution for an existing blockchain (a so-called Layer 2). It can be enriched with a *Mint* and *Redeem* operations that user would call to put or retrieve money in/from the system. Depending on the use case, *Mint* can be equipped with a proof showing that the user has the right to mint a new coin. For example, if PaxPay were used as a scaling solution for Bitcoin, the *Mint* operation would take some SPV (Simplified payment verification) [34] as input to prove to the validators that some value has been locked on Bitcoin. The implementation of the *Mint* and *Redeem* operations would vary depending on whether the system is deployed as a CBDC or as a Layer 2 solution for a blockchain. The performance of PaxPay (Section 7.2) and its scaling capacity are key factors that enable our system to effectively address these use cases.

Bad user behavior. If a user tries to pass two transactions spending the same input coin, her account might get blocked because she will not get enough signatures for either of the transactions. In this case, consensus is required to unlock the user’s account [25, 43].

¹¹ This estimation assumes a direct linear relationship between throughput and the computational power of the validators, as shown in Zef [6].

Future work. Our protocol can be reused with some slight modifications to implement the same functionalities as in Zexe [9], hence not only offering a payment system but *private computation*. We can also considerably improve performance of our system by introducing sharding in the implementation, or improve the usability by moving to a different type of NIZK with a universal trusted setup.

9 Conclusion

In this paper, we consider the problem of fully-private asset transfer (FPAT). We propose PaxPay, an asynchronous FPAT protocol, prove its correctness, detail its implementation in Golang, and analyze its performance. PaxPay leverages succinct non-interactive zero-knowledge proofs and threshold blind signatures. Our system demonstrates significant improvements over existing systems, processing transactions at a high rate, while maintaining full anonymity for the users and responsiveness. PaxPay also ensures regulatory enforcement, including some features that no other private payment system has managed to provide so far. The flexibility offered by succinct NIZKs opens avenues for incorporating other regulatory compliance features with minimal impact on the system's performance. These elements make PaxPay suitable for a wide range of applications, from a scaling solution of blockchain to a standalone private payment system that can be used as a technical layer for CBDCs.

References

- 1 Timoth   Albouy, Emmanuelle Anceaume, Davide Frey, Mathieu Gestin, Arthur Rauch, Michel Raynal, and Fran  ois Taiani. Asynchronous BFT asset transfer: Quasi-anonymous, light, and consensus-free, 2024. [arXiv:2405.18072](https://arxiv.org/abs/2405.18072).
- 2 Martin R. Albrecht, Lorenzo Grassi, Christian Rechberger, Arnab Roy, and Tyge Tiessen. Mimc: Efficient encryption and cryptographic hashing with minimal multiplicative complexity. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I*, volume 10031 of *Lecture Notes in Computer Science*, pages 191–219, 2016.
- 3 Elli Androulaki, Jan Camenisch, Angelo De Caro, Maria Dubovitskaya, Kaoutar Elkhiyaoui, and Bj  rn Tackmann. Privacy-preserving auditable token payments in a permissioned blockchain system. In *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies, AFT '20*, page 255–267, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3419614.3423259.
- 4 European Central Bank. Report on a digital euro, 2020. URL: https://www.ecb.europa.eu/pub/pdf/other/Report_on_a_digital_euro~4d7268b458.en.pdf.
- 5 Mathieu Baudet, George Danezis, and Alberto Sonnino. Fastpay: High-performance byzantine fault tolerant settlement. In *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies*, pages 163–177, 2020.
- 6 Mathieu Baudet, Alberto Sonnino, Mahimna Kelkar, and George Danezis. Zef: Low-latency, scalable, private payments. In Bart P. Knijnenburg and Panos Papadimitratos, editors, *Proceedings of the 22nd Workshop on Privacy in the Electronic Society, WPES 2023, Copenhagen, Denmark, 26 November 2023*, pages 1–16. ACM, 2023.
- 7 Eli Ben Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy*, pages 459–474, 2014. doi:10.1109/SP.2014.36.
- 8 Gautam Botrel, Thomas Piellard, Youssef El Housni, Ivo Kubjas, and Arya Tabaie. Consensus/gnark: v0.11.0, September 2024.

- 9 Sean Bowe, Alessandro Chiesa, Matthew Green, Ian Miers, Pratyush Mishra, and Howard Wu. Zexe: Enabling decentralized private computation. *IACR Cryptol. ePrint Arch.*, page 962, 2018.
- 10 Gabriel Bracha. Asynchronous Byzantine agreement protocols. *Information and Computation*, 75(2):130–143, 1987.
- 11 Tushar Deepak Chandra, Vassos Hadzilacos, and Sam Toueg. The weakest failure detector for solving consensus. *Journal of the ACM (JACM)*, 43(4):685–722, 1996.
- 12 Panagiotis Chatzigiannis and Foteini Baldimtsi. Miniledger: Compact-sized anonymous and auditable distributed payments. In *European Symposium on Research in Computer Security*, pages 407–429. Springer, 2021.
- 13 Daniel Collins, Rachid Guerraoui, Jovan Komatovic, Petr Kuznetsov, Matteo Monti, Matej Pavlovic, Yvonne-Anne Pignolet, Dragos-Adrian Seredinschi, Andrei Tonkikh, and Athanasios Xygkis. Online payments by merely broadcasting messages. In *50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2020, Valencia, Spain, June 29 - July 2, 2020*, pages 26–38. IEEE, 2020.
- 14 Ronald Cramer. *Modular design of secure yet practical cryptographic protocols*. PhD thesis, U. of Amsterdam, 1996.
- 15 Danny Dolev and Rüdiger Reischuk. Bounds on information exchange for byzantine agreement. *Journal of the ACM (JACM)*, 32(1):191–204, 1985.
- 16 Danny Dolev and H. Raymond Strong. Authenticated algorithms for byzantine agreement. *SIAM Journal on Computing*, 12(4):656–666, 1983.
- 17 Cynthia Dwork, Nancy A. Lynch, and Larry J. Stockmeyer. Consensus in the presence of partial synchrony (preliminary version). In Tiko Kameda, Jayadev Misra, Joseph G. Peters, and Nicola Santoro, editors, *Proceedings of the Third Annual ACM Symposium on Principles of Distributed Computing, Vancouver, B. C., Canada, August 27-29, 1984*, pages 103–118. ACM, 1984.
- 18 Electric Coin Company and Zcash Contributors. Zcash: Protocol and reference implementation. <https://github.com/zcash/zcash>, 2023. Accessed: 01/09/2024.
- 19 David Evans, Vladimir Kolesnikov, and Mike Rosulek. *A Pragmatic Introduction to Secure Multi-Party Computation*. Now Publishers Inc, 2018. doi:10.1561/33000000019.
- 20 Prastudy Fauzi, Sarah Meiklejohn, Rebekah Mercer, and Claudio Orlandi. Quisquis: A new design for anonymous cryptocurrencies. Cryptology ePrint Archive, Paper 2018/990, 2018. URL: <https://eprint.iacr.org/2018/990>.
- 21 Michael J. Fischer, Nancy A. Lynch, and Mike Paterson. Impossibility of distributed consensus with one faulty process. In Ronald Fagin and Philip A. Bernstein, editors, *Proceedings of the Second ACM SIGACT-SIGMOD Symposium on Principles of Database Systems, March 21-23, 1983, Colony Square Hotel, Atlanta, Georgia, USA*, pages 1–7. ACM, 1983.
- 22 Christina Garman, Matthew Green, and Ian Miers. Accountable privacy for decentralized anonymous payments. In Jens Grossklags and Bart Preneel, editors, *Financial Cryptography and Data Security - 20th International Conference, FC 2016, Christ Church, Barbados, February 22-26, 2016, Revised Selected Papers*, volume 9603 of *Lecture Notes in Computer Science*, pages 81–98. Springer, 2016.
- 23 Oded Goldreich. *Foundations of Cryptography*, volume 1. Cambridge University Press, 2001. doi:10.1017/CB09780511546891.
- 24 Jens Groth. On the size of pairing-based non-interactive arguments. In *Advances in Cryptology—EUROCRYPT 2016: 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II 35*, pages 305–326. Springer, 2016.
- 25 Rachid Guerraoui, Petr Kuznetsov, Matteo Monti, Matej Pavlovič, and Dragos-Adrian Seredinschi. The consensus number of a cryptocurrency. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC '19*. ACM, July 2019.

- 26 Saurabh Gupta. A Non-Consensus Based Decentralized Financial Transaction Processing Model with Support for Efficient Auditing. Master's thesis, Arizona State University, USA, 2016.
- 27 Youssef El Housni and Aurore Guillevic. Optimized and secure pairing-friendly elliptic curves suitable for one layer proof composition. Cryptology ePrint Archive, Paper 2020/351, 2020. URL: <https://eprint.iacr.org/2020/351>.
- 28 Aram Jivanyan. Lelantus: A new design for anonymous and confidential cryptocurrencies. Cryptology ePrint Archive, Paper 2019/373, 2019. URL: <https://eprint.iacr.org/2019/373>.
- 29 Aram Jivanyan and Aaron Feickert. Lelantus spark: Secure and flexible private transactions. Cryptology ePrint Archive, Paper 2021/1173, 2021. URL: <https://eprint.iacr.org/2021/1173>, doi:10.1007/978-3-031-32415-4_28.
- 30 A. B. Kahn. Topological sorting of large networks. *Commun. ACM*, 5(11):558–562, November 1962.
- 31 Chelsea Komlo and Ian Goldberg. FROST: flexible round-optimized schnorr threshold signatures. In Orr Dunkelman, Michael J. Jacobson Jr., and Colin O'Flynn, editors, *Selected Areas in Cryptography - SAC 2020 - 27th International Conference, Halifax, NS, Canada (Virtual Event), October 21-23, 2020, Revised Selected Papers*, volume 12804 of *Lecture Notes in Computer Science*, pages 34–65. Springer, 2020.
- 32 Petr Kuznetsov, Yvonne-Anne Pignolet, Pavel Ponomarev, and Andrei Tonkikh. Permissionless and asynchronous asset transfer. *Distributed Comput.*, 36(3):349–371, 2023.
- 33 Dahlia Malkhi and Michael K. Reiter. A high-throughput secure reliable multicast protocol. *Journal of Computer Security*, 5(2):113–128, 1997.
- 34 Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008. URL: <http://bitcoin.org/bitcoin.pdf>.
- 35 Shen Noether, Adam Mackenzie, and The Monero Research Lab. Ring Confidential Transactions. *Ledger*, 1:1–18, December 2016. URL: <http://ledger.pitt.edu/ojs/ledger/article/view/34>, doi:10.5195/ledger.2016.34.
- 36 David Pointcheval and Olivier Sanders. Short randomizable signatures. Cryptology ePrint Archive, Paper 2015/525, 2015. <https://eprint.iacr.org/2015/525>. URL: <https://eprint.iacr.org/2015/525>.
- 37 Guillaume Quispe, Pierre Jouvelot, and Gérard Memmi. Nickpay, an auditable, privacy-preserving, nickname-based payment system. *CoRR*, abs/2503.19872, 2025. URL: <https://doi.org/10.48550/arXiv.2503.19872>, arXiv:2503.19872, doi:10.48550/ARXIV.2503.19872.
- 38 Amirreza Sarencheh, Aggelos Kiayias, and Markulf Kohlweiss. PEReDi: Privacy-enhanced, regulated and distributed central bank digital currencies. Cryptology ePrint Archive, Paper 2022/974, 2022. URL: <https://eprint.iacr.org/2022/974>.
- 39 Amirreza Sarencheh, Aggelos Kiayias, and Markulf Kohlweiss. Parscoin: A privacy-preserving, auditable, and regulation-friendly stablecoin. In *Cryptology and Network Security: 23rd International Conference, CANS 2024, Cambridge, UK, September 24–27, 2024, Proceedings, Part I*, page 289–313, Berlin, Heidelberg, 2024. Springer-Verlag.
- 40 Alberto Sonnino, Mustafa Al-Bassam, Shehar Bano, Sarah Meiklejohn, and George Danezis. Coconut: Threshold issuance selective disclosure credentials with applications to distributed ledgers. *arXiv preprint arXiv:1802.07344*, 2018.
- 41 Paul F. Syverson, David M. Goldschlag, and Michael G. Reed. Anonymous connections and onion routing. In *1997 IEEE Symposium on Security and Privacy, May 4-7, 1997, Oakland, CA, USA*, pages 44–54. IEEE Computer Society, 1997.
- 42 Alin Tomescu, Adithya Bhat, Benny Applebaum, Ittai Abraham, Guy Gueta, Benny Pinkas, and Avishay Yanai. UTT: decentralized ecash with accountable privacy. *IACR Cryptol. ePrint Arch.*, 2022. URL: <https://eprint.iacr.org/2022/452>.

- 43 Andrei Tonkikh, Pavel Ponomarev, Petr Kuznetsov, and Yvonne-Anne Pignolet. Cryptoconcurrency: (almost) consensusless asset transfer with shared accounts. In *CCS*, pages 1556–1570. ACM, 2023.
- 44 Visa. Visa annual report 2024, 2024. URL: https://s29.q4cdn.com/385744025/files/doc_downloads/2024/Visa-Fiscal-2024-Annual-Report.pdf.
- 45 Karl Wüst, Kari Kostiainen, Vedran Capkun, and Srdjan Capkun. PRCash: Fast, private and regulated transactions for digital currencies. Cryptology ePrint Archive, Paper 2018/412, 2018. URL: <https://eprint.iacr.org/2018/412>.
- 46 Karl Wüst, Kari Kostiainen, Noah Delius, and Srdjan Capkun. Platypus: A central bank digital currency with unlinkable transactions and privacy preserving regulation. Cryptology ePrint Archive, Paper 2021/1443, 2021. URL: <https://eprint.iacr.org/2021/1443>, doi: 10.1145/3548606.3560617.

A Protocol

Network channel As explained in the model description in Section 3, participants can use private channels. These private channels are formalized as follows. **SEND**(type, data, apk_{dst}) allows to send data without revealing the sender identity. **SEND** triggers a callback **RECEIVE**(type, data, apk_{src}) on the receiver side. The receiver can call **SEND**(type, data, apk_{src}) to send data back. To show that the receiver can respond but without knowing the identity of the sender, we denote by $\widetilde{apk_{src}}$ an anonymous version of apk_{src} received by the receiver, that allows him to answer without knowing apk_{src} . **data** is the payload of the message and **type** its type. These notations will apply in the protocol below.

Setup Phase: During the setup phase, a trusted third party \mathcal{T} first runs the algorithm **SETUP** (Algorithm 1), generating public parameters over the NIZK and keys for the threshold signature scheme. Depending on whether the protocol is run with or without regulation, the public parameters are either $pp_{NIZK}^{tx_no_regul}$ (for the transfer operation) in the non-regulated case or $pp_{NIZK}^{tx_regul}$ and $pp_{NIZK}^{register}$ (for the transfer and register operations) in the regulated case.

The user $u \in \mathcal{U}$ generates her public/secret address pair running **ADDRGEN** algorithm (Algorithm 2) and stores it. She also initializes her list **coinList** (valid unused coins) and **rhoList** (seeds of the received coins).

The trusted third party distributes initial values to each user via the **INITIALIZATION** algorithm (Algorithm 3).

Running Phase: The *transfer* operation for users is implemented either as **TRANSFER_{no_regul}** (Algorithm 6) or **TRANSFER_{regul}** (Algorithm 9), depending on whether regulation enforcement is applied. Algorithms 8 and 7 are used by **TRANSFER_{no_regul}**. Algorithms 10 and 7 are used by **TRANSFER_{regul}**. Algorithms 8 and 10 generate the NIZK in the non-regulated and regulated case. The proof in the regulated case is the same as the one in the regulated case with some additional requirements explained in Section 6 as π_{comply} . π_{regul} in Algorithms 10 is thus the implemented merge of $\pi_{transfer}$ introduced in Section 5 and π_{comply} . If regulation enforcement is applied, user register via the **REGISTER_{regul}** algorithm (Algorithm 11).

The implementation of the *balance* operation is described in Algorithm 4. When a user receives a transfer, Algorithm 5 is called. If regulation enforcement is not applied, validators handle transfer requests with Algorithm 12. If regulation enforcement is applied, validators handle transfer and register requests with Algorithm 13. Note that the algorithm for handling transfer operations is exactly the same, regardless of whether regulation is enforced.

Algorithm 1 **SETUP**()

```

1   $([sk_i]_{i=1}^N, pk_{agg}) \leftarrow \text{SetupSig}(2f + 1, N)$ 
2   $pp_{NIZK} \leftarrow \text{SetupNIZK}(\mathcal{R})$ 
3  for  $p \in (\mathcal{V} \cup \mathcal{U})$  do
4    SEND(type : setupnizk,  $(pp_{NIZK}^{tx\_no\_regul} \mid pp_{NIZK}^{tx\_regul} \ \& \ pp_{NIZK}^{register}), p$ )
5    SEND(type : setupsig,  $pk_{agg}, p$ )
6  end
7  for  $p \in \mathcal{V}$  do
8    SEND(type : sk,  $sk_p, p$ )
9  end

```

Algorithm 2 ADDRGEN()

```

1  $ask \xleftarrow{\$} \{0,1\}^\lambda$  // Sample  $ask$  randomly
2  $apk \leftarrow \text{PRF}_{ask}(0)$ 
3 return  $(ask, apk)$ 

```

Algorithm 3 INITIALIZATION($[apk_u]_{u \in \mathcal{U}}$)

```

1 for  $u \in \mathcal{U}$  do
2   Choose an initial value  $v_u^{init}$ 
3    $\rho_u^{init} \xleftarrow{\$} \{0,1\}^\lambda$  // Sample  $\rho_u$  randomly
4    $\mathbf{c}_u^{init} \leftarrow (v_u^{init}, apk_u, \rho_u^{init})$ 
5   for  $i \in [1, 2f + 1]$  do
6      $\sigma_{u,i} \leftarrow \text{SIGN}(\mathbf{c}_u^{init}, sk_i)$ 
7   end
8    $\sigma_u \leftarrow \text{AGGREGATE}([\sigma_{u,i}]_{i=1}^n)$ 
9   SEND(type : coin,  $(\mathbf{c}_u^{init}, \sigma_u), u$ )
10 end

```

Algorithm 4 BALANCE()

```

1 Parse coinList as  $[\mathbf{c}_i]_{i=1}^l$  //  $l$  is the total number of coins in coinList
2 Parse  $[\mathbf{c}_i]_{i=1}^l$  as  $[(v_i, apk_i, \rho_i)]_{i=1}^l$ 
3 return  $\sum_{i=1}^l v_i$ 

```

Algorithm 5 RECEIVE()

```

1 Upon RECEIVE(type : coin,  $(\mathbf{c}^{new}, \sigma), \widetilde{apk}_{src})$ :
2   Parse  $\mathbf{c}^{new}$  as  $(v^{new}, apk^{new}, \rho^{new})$ 
3   if  $(apk^{new} = \text{PRF}_{ask}(0)) \wedge (\text{VERIFY}_{\text{Sig}}(\sigma, cm, pk_{agg}) == 1) \wedge (\rho^{new} \notin \text{rhoList})$  then
4     Append  $\mathbf{c}$  to coinList

```

■ **Algorithm 6** $\text{TRANSFER}_{\text{no_regul}}([v_j^{\text{new}}]_{j=1}^m, [apk_j^{\text{new}}]_{j=1}^m)$

```

1 Function  $\text{TRANSFER}([v_j^{\text{new}}]_{j=1}^m, [apk_j^{\text{new}}]_{j=1}^m)$ :
   Data: List of new coin owner  $[apk_j^{\text{new}}]_{j=1}^m$  and value  $[v_j^{\text{new}}]_{j=1}^m$ .
   Result: Values  $v_j^{\text{new}}$  are transferred to the users of address  $apk_j^{\text{new}}$ .
2 if  $\exists [c_i]_{i=1}^n \subseteq \text{coinList}, \sum_{i=1}^n c_i.v \geq x$  then
3    $[c_i^{\text{old}}]_{i=1}^n \leftarrow [c_i]_{i=1}^n$ 
4 else
5   return fail
6  $v_{m+1}^{\text{new}} \leftarrow \sum_{i=1}^n v_i^{\text{old}} - \sum_{j=1}^m v_j^{\text{new}}$  // Redeemed to u
7  $apk_{m+1}^{\text{new}} \leftarrow apk$ 
8 Remove  $[c_i^{\text{old}}]_{i=1}^n$  from coinList
9  $([sn_i^{\text{old}}]_{i=1}^n, [\tilde{c}_j^{\text{new}}]_{j=1}^{m+1}, [c_j^{\text{new}}]_{j=1}^{m+1}, [b_j]_{j=1}^{m+1}, \rho_{\text{seed}}) \leftarrow$ 
    $\text{Tx}([c_i^{\text{old}}, sk_i^{\text{old}}]_{i=1}^n, [apk_j^{\text{new}}, v_j^{\text{new}}]_{j=1}^{m+1})$ 
10  $\text{public\_input} \leftarrow ([sn_i^{\text{old}}]_{i=1}^n, [\tilde{c}_j^{\text{new}}]_{j=1}^{m+1})$ 
11  $\text{private\_input} \leftarrow ([c_i^{\text{old}}]_{i=1}^n, [\sigma_i^{\text{old}}]_{i=1}^n, [sk_i^{\text{old}}]_{i=1}^n, [c_j^{\text{new}}]_{j=1}^{m+1}, [b_j]_{j=1}^{m+1}, \rho_{\text{seed}})$ 
12  $\pi \leftarrow \text{Prove}_{\text{no\_regul}}(\text{public\_input}, \text{private\_input}, \text{pp}_{\text{NIZK}}^{\text{tx\_no\_regul}})$ 
13 Create  $m+1$  new empty lists:  $[\text{sigList}_j]_{j=1}^{m+1} \leftarrow [[]]_{j=1}^{m+1}$ 
14 for  $p \in \mathcal{V}$  do
15   // Send the request to each validator in  $\mathcal{V}$ 
16    $\text{SEND}(\text{type} : \text{tx}, ([sn_i^{\text{old}}]_{i=1}^n, [\tilde{c}_j^{\text{new}}]_{j=1}^{m+1}), \pi, p)$ 
17 // Receive  $\tilde{\sigma}_j^p$  sent by validator  $p$ :
18 Upon  $\text{RECEIVE}(\text{type} : \text{sig}, [\tilde{\sigma}_j^p]_{j=1}^{m+1}, p)$ : for  $j = 1, 2, \dots, m+1$  do
19    $\sigma_j^p \leftarrow \text{UNBLIND}(\tilde{\sigma}_j^p, b_j)$ 
20   if  $(\sigma_j^p \notin \text{sigList}_j) \wedge (\text{VERIFY}_{\text{Sig}}(\sigma_j^p, c_j^{\text{new}}, pk_p) == 1)$  then
21     Append  $\sigma_j^p$  to  $\text{sigList}_j$ 
22   if  $\text{length}(\text{sigList}_j) == 2f+1$  then
23      $\sigma_j \leftarrow \text{AGGREGATE}(\text{sigList}_j)$ 
24      $\text{SEND}(\text{type} : \text{coin}, (c_j^{\text{new}}, \sigma_j), apk_j^{\text{new}})$ 
25   return confirm

```

■ **Algorithm 7** $\text{Tx}([c_i^{\text{old}}, ask_i^{\text{old}}]_{i=1}^n, [(apk_j^{\text{new}}, v_j^{\text{new}})]_{j=1}^m)$

```

1 Function  $\text{Tx}([c_i^{\text{old}}, ask_i^{\text{old}}]_{i=1}^n, [(apk_j^{\text{new}}, v_j^{\text{new}})]_{j=1}^m)$ :
   Data: List of old coins to spend with the corresponding secret address. Public address
   and values for the new coins
   Result: Compute the needed data to sign the new coins
2 Parse  $[c_i^{\text{old}}]_{i=1}^n$  as  $[(v_i^{\text{old}}, apk_i^{\text{old}}, \rho_i^{\text{old}})]_{i=1}^n$ 
3  $[sn_i^{\text{old}}]_{i=1}^n \leftarrow [\text{PRF}_{ask_i^{\text{old}}}(\rho_i^{\text{old}})]_{i=1}^n$ 
4  $\rho_{\text{seed}} \xleftarrow{\$} \{0, 1\}^\lambda$ 
5  $[\rho_j^{\text{new}}]_{j=1}^{m+1} \leftarrow [\text{PRF}_{\rho_{\text{seed}}}(sn_1^{\text{old}} || \dots || sn_n^{\text{old}} || j)]_{j=1}^{m+1}$ 
6  $[c_j^{\text{new}}]_{j=1}^{m+1} \leftarrow [(v_j^{\text{new}}, apk_j^{\text{new}}, \rho_j^{\text{new}})]_{j=1}^{m+1}$ 
7  $[b_j]_{j=1}^{m+1} \xleftarrow{\$} \{0, 1\}^\lambda$ 
8  $[\tilde{c}_j^{\text{new}}]_{j=1}^{m+1} \leftarrow [\text{BLIND}(c_j^{\text{new}}, b_j)]_{j=1}^{m+1}$ 
9 return  $([sn_i^{\text{old}}]_{i=1}^n, [\tilde{c}_j^{\text{new}}]_{j=1}^{m+1}, [c_j^{\text{new}}]_{j=1}^{m+1}, [b_j]_{j=1}^{m+1})$ 

```

■ **Algorithm 8** $\text{PROVE}_{no_regul}(\text{public_input}, \text{private_input}, \text{pp}_{\text{NIZK}})$

```

1 Function  $\text{PROVE}_{no\_regul}(\text{public\_input}, \text{private\_input}, \text{pp}_{\text{NIZK}})$ :
   Result:  $\pi_{no\_regul}$  the proof for the transfer without regulation
2   Compute the proof  $\pi_{no\_regul}$  using the NIZK function PROVE with inputs
   ( $\text{public\_input}, \text{private\_input}$ ) and public parameters  $\text{pp}_{\text{NIZK}}$  representing the following
   relation:
   // Created notes are well-formed:
3    $(([sn_i^{old}]_{i=1}^n, [\tilde{c}_j^{new}]_{j=1}^{m+1}), [c_j^{new}]_{j=1}^{m+1}) == \text{Tx}([(\mathbf{c}_i^{old}, sk_i^{old})]_{i=1}^n, (pk_j^{new}, v_j^{new}))$ 
   // Balance is conserved:
4    $\sum_{i=1}^n v_i^{old} == \sum_{j=1}^{m+1} v_j^{new}$ 
   // Secret addresses corresponding to each consumed coin are known:
5    $[apk_i]_{i=1}^n == [\text{PRF}_{ask_i}(0)]_{i=1}^n$ 
   // Spent coins are signed by 2f+1 validators:
6   for  $i \in [1, n]$  do
7   |    $\text{VERIFY}_{\text{Sig}}(\sigma_i, \mathbf{c}_i^{old}, pk) == 1$ 
   // The blinding is correct
8    $[\tilde{c}_j^{new}]_{j=1}^{m+1} == [\text{BLIND}(\mathbf{c}_j^{new}, b_j)]_{j=1}^{m+1}$ 
9   return  $\pi_{no\_regul}$ 

```

■ **Algorithm 9** $\text{TRANSFER}_{\text{regul}}([v_j^{\text{new}}]_{j=1}^m, [apk_j^{\text{new}}]_{j=1}^m)$

```

1 Function  $\text{TRANSFER}([v_j^{\text{new}}]_{j=1}^m, [apk_j^{\text{new}}]_{j=1}^m)$ :
   Data: List of new coin owner  $[apk_j^{\text{new}}]_{j=1}^m$  and value  $[v_j^{\text{new}}]_{j=1}^m$ .
   Result: Values  $v_j^{\text{new}}$  are transferred to the users of address  $apk_j^{\text{new}}$ .
2 if  $\exists [c_i]_{i=1}^n \subseteq \text{coinList}, \sum_{i=1}^n c_i.v \geq x$  then
3   |  $[c_i^{\text{old}}]_{i=1}^n \leftarrow [c_i]_{i=1}^n$ 
4 else
5   | return fail
6  $v_{m+1}^{\text{new}} \leftarrow \sum_{i=1}^n v_i^{\text{old}} - \sum_{j=1}^m v_j^{\text{new}}$  // Redeemed to u
7  $apk_{m+1}^{\text{new}} \leftarrow apk$ 
8 Remove  $[c_i^{\text{old}}]_{i=1}^n$  from coinList
9  $([sn_i^{\text{old}}]_{i=1}^n, [\tilde{c}_j^{\text{new}}]_{j=1}^{m+1}, [c_j^{\text{new}}]_{j=1}^{m+1}, [b_j]_{j=1}^{m+1}, \rho_{\text{seed}}) \leftarrow$ 
   Tx( $[(c_i^{\text{old}}, sk_i^{\text{old}})]_{i=1}^n, [(apk_j^{\text{new}}, v_j^{\text{new}})]_{j=1}^{m+1}$ )
10 Parse  $cc^{\text{old}}$  as  $(apk_{cc}^{\text{old}}, \rho_{cc}^{\text{old}}, v_{cc}^{\text{old}}, com^{\text{old}})$ 
11  $apk_{cc}^{\text{new}} \leftarrow apk_{cc}^{\text{old}}$ 
12  $\rho \leftarrow \text{PRF}_{\rho_{\text{seed}}}(sn_1^{\text{old}} || \dots || sn_n^{\text{old}} || m + 2)$ 
13  $v_{cc}^{\text{new}} \leftarrow v_{cc}^{\text{old}} + \sum_{j=1}^m v_j^{\text{new}}$ 
14  $[r_j] \xleftarrow{\$} \{0, 1\}^\lambda$ 
15  $com_0 \leftarrow com^{\text{old}}$ 
16  $\forall j \in [1, m], com_j = \text{INCR}(com_{j-1}, apk_j^{\text{new}}, v_j^{\text{new}}, r_j)$ 
17  $com^{\text{new}} \leftarrow com^m$ 
18  $b_{m+2} \xleftarrow{\$} \{0, 1\}^\lambda$ 
19  $cc^{\text{new}} \leftarrow (apk_{cc}^{\text{new}}, \rho_{cc}^{\text{new}}, v_{cc}^{\text{new}}, com^{\text{new}})$ 
20  $\tilde{cc} \leftarrow \text{BLIND}(cc^{\text{new}}, b_{m+2})$ 
21  $sn_{n+1}^{\text{old}} = \text{PRF}_{ask_{cc}^{\text{old}}}(\rho_{cc}^{\text{old}})$ 
22  $\text{public\_input} \leftarrow ([sn_i^{\text{old}}]_{i=1}^n, [\tilde{c}_j^{\text{new}}]_{j=1}^{m+1}, V_{\text{sent}}^{\text{max}}, V_{\text{total}}^{\text{max}}, \text{SanctList}, cc^{\text{new}})$ 
23  $\text{private\_input} \leftarrow ([c_i^{\text{old}}]_{i=1}^n, [\sigma_i^{\text{old}}]_{i=1}^n, sk, [c_j^{\text{new}}]_{j=1}^{m+1}, [b_j]_{j=1}^{m+2}, \rho_{\text{seed}}, \sigma_{cc}^{\text{old}}, cc^{\text{old}}, cc^{\text{new}})$ 
24  $\pi \leftarrow \text{Prove}_{\text{regul}}(\text{public\_input}, \text{private\_input}, pp_{\text{NIZK}}^{\text{tx\_regul}})$ 
25 Append  $[(apk_j^{\text{new}}, v_j^{\text{new}}, r_j)]_{j=1}^m$  to comList
26 Create  $m + 1$  new empty lists:  $[\text{sigList}_j]_{j=1}^{m+1} \leftarrow [[]]_{j=1}^{m+1}$ 
27 for  $p \in \mathcal{V}$  do
   | // Send the request to each validator in  $\mathcal{V}$ 
   | SEND(type : tx,  $(([sn_i^{\text{old}}]_{i=1}^n, [\tilde{c}_j^{\text{new}}]_{j=1}^{m+1}), \pi), p$ )
   // Receive  $\tilde{\sigma}_j^p$  sent by validator  $p$ :
29 Upon RECEIVE(type : sig,  $[\tilde{\sigma}_j^p]_{j=1}^{m+1}, p$ ): for  $j = 1, 2, \dots, m + 1$  do
30    $\sigma_j^p \leftarrow \text{UNBLIND}(\tilde{\sigma}_j^p, b_j)$ 
31   if  $(\sigma_j^p \notin \text{sigList}_j) \wedge (\text{VERIFY}_{\text{sig}}(\sigma_j^p, c_j^{\text{new}}, pk_p) == 1)$  then
32     Append  $\sigma_j^p$  to sigListj
33     if length(sigListj) ==  $2f + 1$  then
34        $\sigma_j \leftarrow \text{AGGREGATE}(\text{sigList}_j)$ 
35       SEND(type : coin,  $(c_j^{\text{new}}, \sigma_j), apk_j^{\text{new}})$ 
36     return confirm

```

■ **Algorithm 10** $\text{PROVE}_{\text{regul}}(\text{public_input}, \text{private_input}, \text{pp}_{\text{NIZK}})$

```

1 Function  $\text{PROVE}_{\text{regul}}(\text{public\_input}, \text{private\_input}, \text{pp}_{\text{NIZK}})$ :
   Result:  $\pi_{\text{regul}}$  the proof for the transfer without regulation
2   Compute the proof  $\pi_{\text{regul}}$  using the NIZK function PROVE with inputs
   ( $\text{public\_input}, \text{private\_input}$ ) and public parameters  $\text{pp}_{\text{NIZK}}$  representing the following
   relation:
   // Created notes are well-formed:
3    $(([sn_i^{\text{old}}]_{i=1}^n, [\tilde{c}_j^{\text{new}}]_{j=1}^{m+1}), [c_j^{\text{new}}]_{j=1}^{m+1}) == \text{Tx}([c_i^{\text{old}}, sk_i^{\text{old}}]_{i=1}^n, (pk_j^{\text{new}}, v_j^{\text{new}}))$ 
   // Balance is conserved:
4    $\sum_{i=1}^n v_i^{\text{old}} == \sum_{j=1}^{m+1} v_j^{\text{new}}$ 
   // Secret addresses corresponding to each consumed coin are known:
5    $[apk_i]_{i=1}^n == [\text{PRF}_{ask_i}(0)]_{i=1}^n$ 
   // Spent coins are signed by 2f+1 validators:
6   for  $i \in [1, n]$  do
7   |  $\text{VERIFY}_{\text{Sig}}(\sigma_i, c_i^{\text{old}}, pk) == 1$ 
   // The blinding is correct
8    $[\tilde{c}_j^{\text{new}}]_{j=1}^{m+1} == [\text{BLIND}(c_j^{\text{new}}, b_j)]_{j=1}^{m+1}$ 
   // Regulation checks
9   for  $i \in [1, n]$  do
10  |  $apk_i^{\text{old}} == apk_{cc}^{\text{old}}$ 
11  |  $apk_{cc}^{\text{old}} == \text{PRF}_{ask_{cc}^{\text{old}}}(0)$ 
12  |  $apk_{cc}^{\text{new}} == apk_{cc}^{\text{old}}$ 
13  |  $\rho_{cc}^{\text{new}} == \text{PRF}_{\rho_{seed}}(sn_1^{\text{old}} || \dots || sn_n^{\text{old}} || m + 2)$ 
14  |  $v_{cc}^{\text{new}} == v_{cc}^{\text{old}} + \sum_{j=1}^m v_j^{\text{new}}$ 
15  |  $com_0 \leftarrow com^{\text{old}}$ 
16  |  $\forall j \in [1, m], com_j \leftarrow \text{INCR}(com_{j-1}, apk_j^{\text{new}}, v_j^{\text{new}}, r_j)$ 
17  |  $com^{\text{new}} == com_m$ 
18  |  $\sum_{j=1}^m v_j^{\text{new}} \leq V_{total}^{\text{max}}$ 
19  |  $v_{cc}^{\text{new}} \leq V_{total}^{\text{max}}$ 
20  |  $apk_{cc}^{\text{old}} \notin \text{SancList}$ 
21  |  $\forall j \in [1, m], apk_j^{\text{new}} \notin \text{SancList}$ 
22  return  $\pi_{\text{regul}}$ 

```

■ **Algorithm 11** REGISTER_{regul}()

```

1 Function REGISTER():
  Result:  $cc_u$  an initial compliance coin
2  if  $\exists [c_i]_{i=1}^n \subseteq \text{coinList}, \sum_{i=1}^n c_i.v \geq x$  then
3     $[c_i^{old}]_{i=1}^n \leftarrow [c_i]_{i=1}^n$ 
4  else
5    return fail
6   $\rho \xleftarrow{\$} \{0, 1\}^\lambda$ 
7   $cc_u \leftarrow (0, \rho, 0, 0)$ 
8   $b \xleftarrow{\$} \{0, 1\}^\lambda$ 
9   $\tilde{c}_u \leftarrow \text{BLIND}((apk_u, \rho, 0, 0), b)$ 
10  $\text{public\_input} \leftarrow (apk_u, \tilde{c}_u)$ 
11  $\text{private\_input} \leftarrow (ask_u, cc_u, b)$ 
12 Compute  $\pi_{register}$  by calling PROVENIZK with inputs (public_input, private_input) and
   public parameters  $pp_{NIZK}^{register}$  that represent the following relations:
13    $\tilde{c}_u == \text{BLIND}((apk_u, \rho, 0, 0), b)$ 
14    $apk_u == \text{PRF}_{ask_u}(0)$ 
15 Create a new empty list: sigList  $\leftarrow []$ 
16 for  $p \in \mathcal{V}$  do
17   // Send the request to each validator in  $\mathcal{V}$ 
   SEND(type : register, ( $\tilde{c}_u, apk_u, u, \pi_{register}$ ),  $p$ )
   // Receive  $\tilde{\sigma}_{cc}^p$  sent by validator  $p$ :
18   Upon RECEIVE(type : sig,  $\tilde{\sigma}_{cc}^p, p$ ):
19      $\sigma_{cc}^p \leftarrow \text{UNBLIND}(\tilde{\sigma}_{cc}^p, b)$ 
20     if VERIFYSig( $\sigma_{cc}^p, cc_u, pk_p$ ) == 1 then
21       if length(sigList) ==  $2f + 1$  then
22          $\sigma_{cc} \leftarrow \text{AGGREGATE}(\text{sigList})$ 
23         Store  $\sigma_{cc}, cc_u$ 
24       return confirm

```

■ **Algorithm 12** VALIDATOR_{no_regul}()

```

1 Storage:
2   snList // Stores all the serial numbers sn seen in valid transactions. Used
   to avoid double spending.
3   signedCoinList // Stores all the signed blinded coins  $\tilde{c}$  seen in valid
   transactions. Used to avoid malicious replica attack.
4 Upon RECEIVE(type : tx, ( $[sn_i^{old}]_{i=1}^n, [\tilde{c}_j^{new}]_{j=1}^{m+1}, \pi$ ),  $\widehat{apk}_{src}$ ) :
5   if (VERIFYNIZK( $[sn_i^{old}]_{i=1}^n, [\tilde{c}_j^{new}]_{j=1}^{m+1}, pp_{NIZK}^{tx\_no\_regul}, \pi$ )  $\wedge (\forall i \in [1, n], sn_i^{old} \notin$ 
   snList)  $\vee ([\tilde{c}_j^{new}]_{j=1}^{m+1} \subseteq \text{signedCoinList})$  then
6      $[\tilde{\sigma}_i] \leftarrow \text{SIGN}(\tilde{c}_i^{new}, sk)_{i=1}^n$  // Computes the blinded coin's signatures
7     for  $sn \in [sn_i^{old}]_{i=1}^n$  do
8       Append  $sn$  to snList
9     Append  $[\tilde{c}_j^{new}]_{j=1}^{m+1}$  to signedCoinList
10    SEND(type : sig,  $[\tilde{\sigma}_i]_{i=1}^{m+1}, \widehat{apk}_{src}$ )
11  else
12    Abort

```

■ **Algorithm 13** $\text{VALIDATOR}_{\text{regul}}()$

```

1 Storage:
2   snList // Stores all the serial numbers sn seen in valid transactions. Used
   to avoid double spending.
3   signedCoinList // Stores all the signed blinded coins  $\tilde{c}$  seen in valid
   transactions. Used to avoid malicious replica attack.
4   registeredList // Stores a pair (u,apk) of user ID associated with a public
   key for all the registered users

5 Upon RECEIVE(type : tx,  $(([sn_i^{old}]_{i=1}^n, [\tilde{c}_j^{new}]_{j=1}^{m+1}), \pi), \widetilde{apk}_{src})$  :
6   if ( $\text{VERIFY}_{\text{NIZK}}([sn_i^{old}]_{i=1}^n, [\tilde{c}_j^{new}]_{j=1}^{m+1}), pp_{\text{NIZK}}^{\text{tx-regul}}, \pi) \wedge (\forall i \in [1, n], sn_i^{old} \notin$ 
   snList)  $\vee ([\tilde{c}_j^{new}]_{j=1}^{m+1} \subseteq \text{signedCoinList})$  then
7      $[\tilde{\sigma}_i \leftarrow \text{SIGN}(\tilde{c}_i^{new}, sk)]_{i=1}^n$  // Computes the blinded coin's signatures
8     for  $sn \in [sn_i^{old}]_{i=1}^n$  do
9       Append sn to snList
10    Append  $[\tilde{c}_j^{new}]_{j=1}^{m+1}$  to signedCoinList
11    SEND(type : sig,  $[\tilde{\sigma}_i]_{i=1}^{m+1}, \widetilde{apk}_{src})$ 
12  else
13    Abort

14 Upon RECEIVE(type : register,  $(\tilde{c}_u, apk_u, u, \pi), \widetilde{apk}_{src})$  :
15  if
   ( $\text{VERIFY}_{\text{NIZK}}((\tilde{c}_u, apk_u), pp_{\text{NIZK}}^{\text{register}}, \pi) \wedge ((u, *) \notin \text{registeredList}) \vee (\tilde{c}_u \in \text{signedCoinList}))$ 
   then
16     $\tilde{\sigma}_{cc} \leftarrow \text{SIGN}(\tilde{c}_u^{new}, sk)$  // Computes the blinded coin's signature
17    Append  $\tilde{c}_u$  to signedCoinList
18    SEND(type : sig,  $\tilde{\sigma}_{cc}, \widetilde{apk}_{src})$ 
19  else
20    Abort

```

B Implementation Details

B.1 Cryptographic building blocks

Pointcheval-Sanders. As explained in Section 7, the signature scheme is a slightly modified version of the Coconut scheme [40], which itself is a variant of the Pointcheval-Sanders signature scheme [36]. It is based on type-3 bilinear pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ over secure groups $\mathbb{G}_1, \mathbb{G}_2$ and \mathbb{G}_T . We concretely instantiated it with BLS12-377. We denote by \mathbb{F}_q its scalar field. First, let:

- \mathcal{H} be a cryptographic collision and pre-image resistant hash function with image in \mathbb{G}_1 , so called hash-to-curve¹².
- H be a cryptographic collision and pre-image resistant hash function¹³ with image in \mathbb{F}_q . Since the message m will be a coin in the implementation, we allow H to take a tuple (m_1, m_2, \dots, m_n) as input and return $H(m_1 || m_2 || \dots || m_n)$. m is thus always manipulated as $H(m)$ in the signature scheme.

The signature scheme is as follows:

- **SETUP_{sig}** $() \rightarrow ([sk_i]_{i=1}^N, [pk_i]_{i=1}^N, pp, pk_{agg})$: Operate two Shamir secret sharing choosing two random polynomials $P_1, P_2 \in \mathbb{F}_q[X]$ of degree $k-1$ evaluating it in the points $0, 1, 2, \dots, N \in \mathbb{F}_q$. Let the partial secret keys be the couples $[sk_i = (x_i, y_i) = (P_1(i), P_2(i))]_{i=1}^N$. Sample a generator $g_1 \in \mathbb{G}_1$ and a generator $g_2 \in \mathbb{G}_2$ and let the global verification key be $pk_{agg} = (X, Y) = (g_2^{P_1(0)}, g_2^{P_2(0)})$. Compute and publish the list $[\gamma_i]_{i=1}^N = [g_1^{y_i}]_{i=1}^N$ altogether with g_1 and g_2 .
- **BLIND** $(m, b) \rightarrow \tilde{m}$: Sample a random $s \in \mathbb{F}_q$, compute $c_m = \text{PRF}_s(m)$ and $h = \mathcal{H}(c_m)$. Let $\tilde{m} = (c_m, h, h^{H(m)} g_1^b)$ and store b . Compute a proof π_{aux} that \tilde{m} is well computed¹⁴ and send it altogether with \tilde{m} to the signer.
- **SIGN** $(\tilde{m}, sk_i) \rightarrow \tilde{\sigma}_i$: Parse \tilde{m} as $(\tilde{m}_1, \tilde{m}_2, \tilde{m}_3)$. Parse sk_i as (x_i, y_i) . Check that $\mathcal{H}(\tilde{m}_2) = \tilde{m}_1$ and π_{aux} is valid. Compute $\tilde{\sigma}_i = (\tilde{\sigma}_{i,1}, \tilde{\sigma}_{i,2}) = (\tilde{m}_2, \tilde{m}_2^{x_i} \tilde{m}_3^{y_i})$.
- **UNBLIND** $(\tilde{m}, \tilde{\sigma}_i, b) \rightarrow \sigma_i$: Parse $\tilde{\sigma}_i$ as $(\tilde{\sigma}_{i,1}, \tilde{\sigma}_{i,2})$. Compute $\sigma_{i,2} = \tilde{\sigma}_{i,2} \gamma_i^{-b}$. Compute $\sigma_i = (\tilde{\sigma}_{i,1}, \sigma_{i,2})$.
- **AGGREGATE** $([\sigma_i]_{i=1}^n) \rightarrow \sigma$: Compute $\sigma_2 = \prod_{i=1}^n \sigma_{i,2}^{\lambda_i}$ where the $\lambda_i = \prod_{j \in (0,k) \setminus \{i\}} \frac{j}{j-i}$ are the Lagrange coefficients. Sample a random t to randomize the signature. Let $\sigma = (h^t, \sigma_2^t)$.
- **VERIFY_{sig}** $(m, \sigma, pk) \rightarrow b$: Parse σ as (σ_1, σ_2) . Parse pk as (X, Y) . Check that $e(\sigma_1, XY^{H(m)}) = e(\sigma_2, g_2)$ ¹⁵.

¹² <https://pkg.go.dev/github.com/consensys/gnark-crypto@v0.13.0/ecc/bls12-377#HashToG1>

¹³ <https://pkg.go.dev/github.com/consensys/gnark-crypto/ecc/bls12-377/fr/mime>

¹⁴ i.e. π_{aux} has public inputs $\tilde{m} = (\tilde{m}_1, \tilde{m}_2, \tilde{m}_3) = (c_m, h, h^{H(m)} g_1^b)$, private inputs (m, s, b) and statement $(\tilde{m}_1 == \text{PRF}_s(m) \wedge \tilde{m}_3 == \tilde{m}_2^m g_1^b)$. Computing first $c_m = \text{PRF}_s(m)$ and then \mathcal{H} instead of directly computing $\mathcal{H}(m + k)$ saves us from computing the costly hash-to-curve \mathcal{H} inside of the NIZK. In our protocol, this condition is verified within the proof π in **TRANSFER** algorithm. The fact that this modification doesn't impact blinding property of the scheme is given proof of *FPAT-privacy*. *Preimage-Resistance* of the PRF implies the security is neither impacted, because it prevents the requester from reusing several times the group element h .

¹⁵ Let $sk_{agg} = (x, y) = (P_1(0), P_2(0))$ and (σ_1, σ_2) a valid signature. For left-hand side of the equation we have $e(\sigma_1, XY^{H(m)}) = e(h^t, g_2^x g_2^{yH(m)}) = e(h^t, g_2^{x+yH(m)}) = e(h, g_2)^{t(x+yH(m))}$, and for the right-hand side we have $e(\sigma_2, g_2) = e((h^{x+yH(m)})^t, g_2) = e(h^{t(x+yH(m))}, g_2) = e(h, g_2)^{t(x+yH(m))}$, so equality holds.

Note that a great advantage of this signature scheme over concurrent threshold signature schemes (as the popular FROST [31]) is that aggregation is completely non-interactive while working on an unlimited number of messages, which makes it suitable for usage in our asynchronous protocol.

C Proofs

For convenience, our implementations of **TRANSFER** and **VALIDATOR** (Algorithms 6 and 12) allow performing transfers to multiple receivers in one **TRANSFER** call as explained in Section 5. This can be seen as a batch of successive transfers, with only one receiver each. The proofs below follow the specification introduced in Section 4 and therefore only consider transfers with one receiver ($m = 1$). Hence, in this section, **TRANSFER** creates two coins: a coin \mathbf{c}_1^{new} for the recipient of the payment and a redeem coin \mathbf{c}_2^{new} . The proofs can be easily extended to transfers with multiple receivers.

To simplify notations, we also assume that each user has exactly one public/secret address pair. Thus, apk_2^{new} must equal the sole public address of the sender.

The proofs concern the non-regulated protocol first, but they can be extended to the regulated protocol as explained in Appendix C.4.

C.1 FPAT-Safety

Let H be the history of an execution of PaxPay (composed of invocations and responses of FPAT operations). Let L_p denote the local history of a participant p . Notice that, as the Byzantine participants are not expected to follow the protocol, the FPAT-safety specification allows us to assign them arbitrary local histories, as long as the safety property for the correct participants is met. There are only three ways for a correct participant to receive a message:

- line 1 of Algorithm 5 - **Receive** (the participant is a user);
- line 16 of Algorithm 6 - **Transfer** (the participant is a user);
- line 4 of Algorithm 12 - **Validator** (the participant is a validator).

In each of these cases, the message content is checked (lines 3 in Algorithm 5, 18 in Algorithm 6 and 5 in Algorithm 12). The messages coming from Byzantine participants that do not pass these checks are ignored, we thus do not consider them in our analysis.

Consider an execution E of our algorithm. Let $H = (L_1, \dots, L_U)$ be the history of E (in fact only local histories of correct users are of interest for us). Our goal is to construct a legal serialisation S of H —a sequential history that agrees with L_u for every correct user u ($S|u = L_u$) that respects the FPAT specification (Section 4). Considering *transfer* and *balance* operations *instantiated* in E (to be defined later), we build a directed graph G_H , which accounts for local histories L_p for correct participants, as well as causality relations across the operations. We then show that G_H is acyclic and construct from it a *legal serialization* of H .

Let us first define the set of operations that constitute the vertices of G_H . We say that a *transfer* operation o invoked by a correct user u *consumes* a coin \mathbf{c}^{old} if the execution of line 2 of **TRANSFER** (Algorithm 6) selects the coin \mathbf{c}^{old} . Let c be a coin consumed by a correct user. By the algorithm, if c is sent by a correct user u , it has been *created* within a *transfer* operation o by u in which c has been appended to **coinList** (line 4 of **RECEIVE** Algorithm 5). We then include o to the vertices of G_H . If c is sent by a Byzantine user u' , we add to G_H an (artificial) operation *transfer* o' by u' that creates c .

We say that a complete *balance* operation o performed by correct user *reads* a coin \mathbf{c} , and we add o to G_H , if the execution of line 1 of **BALANCE** (Algorithm 4) parses \mathbf{c} .

► **Lemma 1.** *No more than one transfer operation can consume a given coin.*

Proof. Suppose that a *transfer* operation $o_1 \in H$ consumes a coin \mathbf{c}^{old} . We show that for any operation $o_2 \in H, o_2 \neq o_1$, o_2 does not consume \mathbf{c}^{old} .

The operation o_1 received valid responses from a set \mathcal{V}_1 of at least $2f + 1$ validators (line 20 of Algorithm 6). At least $f + 1$ of them are correct and thus have added sn^{old} to their `snList` (line 8 of Algorithm 12). Consider another complete operation o_2 and let it consume coins $[\mathbf{c}_i]_{i=1}^n$. Again, a set \mathcal{V}_2 of $2f + 1$ validators have added $[sn_i^{old}]_{i=1}^n$ to their `snList`. Since there are $N = 3f + 1$ validators in total, there are at least $f + 1$ validators in the intersection of \mathcal{V}_1 and \mathcal{V}_2 . Hence, at least one correct validator has added sn^{old} and $[sn_i^{old}]_{i=1}^n$ to its `snList`. Because of line 5 of Algorithm 12, sn^{old} is different from any $[sn_i^{old}]_{i=1}^n$. ◀

Lemma 1 implies that a coin can only be consumed by a single operation executed by a correct user. Thus, a coin can be consumed by at most one vertex of G_H .

► **Lemma 2.** *If a transfer operation o consumes a coin \mathbf{c}^{old} , either there exists another transfer o' that creates \mathbf{c}^{old} , or \mathbf{c}^{old} has been created during the initialization.*

Proof. Suppose that a *transfer* o consumes a coin \mathbf{c}^{old} . By the Algorithm 6 (line 2), this coin has been previously appended to `coinList`. The only way to append a coin in `coinList` is either at initialization, or through the algorithm **RECEIVE** (Algorithm 5) line 4. The second and third clause of the check in line 3 of **RECEIVE** (Algorithm 5), combined with the unforgeability of the signature scheme, imply that this coin has been signed by $2f + 1$ validators. Indeed, the unforgeability of the signature scheme directly comes from the one of the variant of the Pointcheval-Sanders signature scheme which is randomizable and aggregatable. This holds under the SXDH assumption, as stated in [36]. There are at $f + 1$ correct validators amongst them. Those correct validators only produce these signatures if this coin has been created in the algorithm **TRANSFER** or in the algorithm **INITIALIZATION**. ◀

The directed edges of G_H consist of (o_1, o_2) such that, for some correct user u , $o_1 \prec_u o_2$ (u performs o_1 before o_2) or there exists a coin \mathbf{c} such that one of the following conditions holds:

- o_1 and o_2 are both *transfer*, o_1 creates \mathbf{c} and o_2 consumes \mathbf{c}
- o_1 is a *transfer*, o_2 is a *balance*, o_1 creates \mathbf{c} and o_2 reads \mathbf{c}
- o_1 is a *balance*, o_2 is a *transfer*, o_1 reads \mathbf{c} and o_2 consumes \mathbf{c}

► **Lemma 3.** *G_H is acyclic.*

Proof. By Lemma 1, a coin can be consumed at most once. The PRF used to derive the seeds ρ^{new} of the created coins from the seed ρ^{old} of the consumed coins is *preimage-resistant*. Thus, no coin can be created more than once, no coin can be consumed or read before it has been created, and no coin can be read after it has been consumed. As a *transfer* operation creates a coin *after* it consumes one. Therefore, the edges of G_H defined by the coin relations above preclude cycles. Also for every such edge (o_1, o_2) it cannot happen that o_2 completes before o_1 begins. Suppose, by contradiction, that there is a cycle in G_H . The arguments above imply that the cycle must have at least one edge (o_1, o_2) such that $o_1 \prec_u o_2$ for some correct u . As o_1 completes before o_2 begins, a simple inductive argument implies that there cannot be a path from o_2 to o_1 in G_H . ◀

Now we can use Lemma 3 and any algorithm for topological sorting (e.g., Kahn's algorithm [30]) to create a *serialization* S that, for each correct user u , includes all complete operations of u and respects L_u .

It remains to prove that:

► **Lemma 4.** S is legal.

Proof. We will show that S is legal by induction on its prefixes. At initialization, S_0 is void so trivially legal. Now suppose, by induction, that S_i , the prefix of S of length i is legal and let o be the $(i + 1)$ -th operation in S .

- o is a *balance operation*: By the construction of S , for each coin c read by o , the matching *transfer* operation that creates c have already been appended to the sequence S_{i+1} . Also, these coins have not been spent yet by any *transfer* operation since the transfers that spend them have o as input. The NIZK proofs ensure that the output of o cannot exceed the sum of the values of the consumed coins, so S_{i+1} is legal.
- o is a *transfer operation*: Consider the coins consumed by o . By Lemma 2, it exists operations in S_i that created them. By Lemma 1 and the balance between the values of consumed and created coins (line 4 of Algorithm 8), o is the only *transfer* operation that consumes these coins. Again, o cannot spend more than the current balance of u , and S_{i+1} is legal.

◀

► **Lemma 5.** If a correct user u invokes an operation $o = \text{balance}_u()$ returning value v_1 then, if the next operation $\text{transfer}_u(v_2, w)$ invoked is such that $v_2 \leq v_1$, it does not return fail.

Proof. This condition is immediately verified because the only event that can make a *transfer* operation invoked by a correct user fail is non-existence of a list of coins $[c_i^{old}]_{i=1}^n \subseteq \text{coinList}$ such that $\sum_{i=1}^n c_i^{old} \cdot v \geq x$ (line 2 of **TRANSFER**). But no elements are removed from **coinList** during protocol execution except in a later step of **Transfer** algorithm. This guarantees that all coins read (line 1) in the **Balance** algorithm called by the preceding *balance* operation are still in **coinList**. By hypothesis they pass the check, so the *transfer* does not return fail. ◀

Lemmas 4 and 5 imply:

► **Theorem 6.** *PaxPay* ensures *FPAT-Safety*.

As said in Section 5, our **TRANSFER** algorithm allows to transfer money to several users in one call. Each call to **TRANSFER** can be translated as successive *transfer* operations to a single user, thus the Theorem 6 still holds for our algorithm **TRANSFER**.

C.2 FPAT-Liveness

Recall that *FPAT-Liveness* guarantees that (1) every operation invoked by a correct user eventually terminates and (2) every transfer operation tx performed by a correct user eventually takes effect, i.e., there is a time after which *balance* operations at correct users (sender or receiver) return values accounting for tx .

► **Theorem 7.** *PaxPay* ensures *FPAT-Liveness*.

Proof. (1) Operation *balance* is performed locally and so trivially terminates. Concerning the *transfer* operations, the only blocking instruction in the **TRANSFER** algorithm is the request of coins' signatures (line 16). The first clause of the check performed line 3 of the **RECEIVE** algorithm (Algorithm 5) ensures that the calling user u knows the secret key corresponding to all the coins of her `coinList`. The second clause guarantees that all these coins have valid signatures. Then, the *completeness* property of the NIZK guarantees that the proof π computed at line 12 of Algorithm 6 is valid. The removal of spent coins from `coinList` at line 8 of the **TRANSFER** algorithm, along with the last clause of the check line 3 of the **RECEIVE** algorithm guarantees that u will not try to spend the same coin twice. Moreover, the *preimage-resistance* of the PRF guarantees that secret addresses ask of u have not been computed by any Byzantine participant. The same property of the PRF implies that no Byzantine participant have computed the serial numbers sn computed at line 3, so it does not already appears in the `snList` of validators. Thus, since the serial number do not already appear in their `snList` and the proof π is valid, the correct validators that receive the transfer request will sign the new coins and answer. Then termination of *transfer* operation directly follows from the network assumption (communication channels are reliable) and from the model assumption (at least $2f + 1$ validators are correct).

(2) There are only a finite number of $transfer_*(v+, u)$ operation completed before time t and the channels are reliable. Thus, there exists a time t' after which all the coins that were sent to a correct user u before time t are received by u . Because of the line 4 of the **RECEIVE** algorithm, these coins have been appended to u 's `coinList`. Let o be a *balance* operation invoked by u at time $t'' \geq t'$. The only coins removed from her `coinList` have been removed during completed *transfers* operation she has invoked before t'' . The coins in her `coinList` at time t'' are thus the coins that have been received before t' but not removed by the transfer operation she has called before t'' . Hence:

$$\sum_{\substack{transfer_*(v+, u) \\ \text{invoked by correct users} \\ \text{completed before time } t}} v_+ - \sum_{\substack{transfer_u(v-, *) \\ \text{completed before time } t''}} v_-$$

Thus, FPAT-Liveness holds. ◀

C.3 FPAT-privacy

In this section we describe more precisely the game \mathcal{G}^{priv} defining *FPAT-privacy*.

Let $Pr(\mathcal{G}^{priv})$ be the probability that the adversary wins the game \mathcal{G}^{priv} . By definition, *FPAT-privacy* means that it exists a negligible function Adv such that, for a security parameter λ :

$$|Pr(\mathcal{G}^{priv}) - \frac{1}{2}| \leq \text{Adv}(\lambda)$$

First we give more details about the functioning of the PaxPay oracle \mathcal{O}^{PaxPay} . It provides the three following interfaces:

- An interface allowing the caller to create addresses. Upon receiving such a *createAddress* query, it runs Algorithm 2, stores the generated private address ask and returns to the caller the corresponding public address apk . It also initializes an empty `coinList` attached to this address.
- An interface allowing the caller to append coins to the `coinList` of the different users.

- An interface allowing the caller to submit abstract transactions. Abstract transactions are tuples $tx = (apk_S, [c_i]_{i=1}^n, v, apk_R)$ specifying a transaction sender's public key apk_S , a list consumed coins $[c_i]_{i=1}^n$ belonging to the `coinList` attached to apk_S , a transferred value v , and a receiver public address apk_R . Upon receiving such a query tx , \mathcal{O}^{PaxPay} runs Algorithm 6 with inputs (v, apk_R) (with knowledge of the local storage of user apk_S , including her `coinList` and apk_S) as well as Algorithm 12, internally simulating the interaction between the sender and validators. Finally, it outputs the execution traces of all parties.

Remark: Algorithm 4, corresponding to *balance* operation, as Algorithm 5, by which users receive funds, are performed completely locally. So they trivially do not impact the privacy guarantees of our protocol and we exclude them from our analysis.

We suppose the NIZK scheme is perfect zero-knowledge, with a simulator taking as input a trapdoor td_{NIZK} generated with the common reference string pp_{NIZK} during execution of $\text{SETUP}_{NIZK}^{\text{trapdoor}}$, as it is the case for Groth16.

The distinguishing game \mathcal{G}^{priv} is defined by following interactions between the adversary \mathcal{A} , and a challenger \mathcal{C} :

Initialization Phase:

- \mathcal{C} chooses the security parameter λ . He generates public parameters pp_{NIZK} of the NIZK scheme running $\text{SETUP}_{NIZK}^{\text{trapdoor}}$, obtaining at the same time the corresponding trapdoor td_{NIZK} taken as input by the NIZK simulator. \mathcal{C} also runs $\text{SETUP}_{\text{sig}}$ obtaining the public parameters pp_{sig} of the signature scheme. \mathcal{C} stores the secret key $sk_{agg} = (P_1(0), P_2(0))$ corresponding to the threshold public key $pk_{agg} = (g_2^{P_1(0)}, g_2^{P_2(0)})$.
 \mathcal{C} sends those public parameters $(pp_{NIZK}, pp_{\text{sig}})$ to the adversary \mathcal{A} and uses them to initialize two oracles $\mathcal{O}_1^{\text{PaxPay}}$ and $\mathcal{O}_2^{\text{PaxPay}}$.
- For each user, \mathcal{A} sends a *createAddress* queries to \mathcal{C} who transmit it to the oracles. They outputs public keys to \mathcal{C} , who transmits them to \mathcal{A} .
- \mathcal{A} initializes the balances of the users generating coins appended to the `coinList` of each participant in both oracles $\mathcal{O}_1^{\text{PaxPay}}$ and $\mathcal{O}_2^{\text{PaxPay}}$.
- \mathcal{C} samples a random bit b .

Challenge Phase: The challenge phase consists in several iterations of the following steps:

- \mathcal{A} submits pairs of consistent abstract transactions (tx_0, tx_1) to the challenger \mathcal{C} . Two transactions are consistent if and only if the number n of coins they consume are identical, and, if the receiver apk_R is controlled by \mathcal{A} in one of the two transactions, then the receiver of the other transaction is the same apk_R in the other transaction and the transferred value v is also equal in both transactions.
- Receiving couple of abstract transactions (tx_0, tx_1) , the challenger \mathcal{C} checks its consistency. If the two abstract transactions are consistent, he provides the oracles $\mathcal{O}_1^{\text{PaxPay}}$ and $\mathcal{O}_2^{\text{PaxPay}}$ respectively with tx_0 and tx_1 , receiving as output execution traces Tr_0 and Tr_1 .
- Finally, \mathcal{C} provides \mathcal{A} with the couple $(Tr_b^{\mathcal{A}}, Tr_{1-b}^{\mathcal{A}})$ of the restrictions of execution traces to the parties controlled by \mathcal{A} , in a permuted order depending on the bit b sampled in initialization phase.

Decision Phase: At the end of the interactions with the challenger \mathcal{C} , the adversary \mathcal{A} outputs a guess b' about the bit b sampled by \mathcal{C} during the initialization. \mathcal{A} wins the game if $b' = b$.

Proof: We give a sketch of proof of *FPAT-privacy* by an hybrid argument. We describe a sequence $[\mathcal{G}_i]_{i=0}^5$ of games such that \mathcal{G}_0 is the game \mathcal{G}^{priv} defining our security property, and \mathcal{G}_5 is a game where the challenger's responses to the adversary queries does not depend on the bit b . We prove that for all i in $\{0, \dots, 5\}$ the adversary's advantage in distinguishing game \mathcal{G}_i from game \mathcal{G}_{i+1} is negligible. Because the winning advantage of \mathcal{A} in \mathcal{G}_5 is trivially null, this proof the adversary's advantage is negligible in \mathcal{G}^{priv} by triangular inequality.

In PaxPay protocol, correct users only interact with the validators by broadcasting a same message to all of them (message of type tx line 15 of **TRANSFER**). Thus, the views of each validator are identical up to the order of received requests. Moreover, network assumptions states that, for each user u , communication channels between u and any validator have similar latency distributions. It is thus sufficient to restrict our analysis to the view of a generic validator. Indeed, the combined knowledge of all the Byzantine validators leaks no more information to \mathcal{A} than this generic validator's knowledge.

We note Adv^{PRF} the advantage the adversary has in distinguishing a random function from an element of the PRF family sampled with an uniformly random seed. For the threshold blind signature scheme, rather than with an abstraction, we reason directly on the modified Coconut described in Appendix B.1. We note Adv^{DDH} the advantage in winning the Decisional Diffie-Hellman (DDH) game in \mathbb{G}_1 .

Game \mathcal{G}_1 is the same game as \mathcal{G}_0 excepts the NIZK in validator's trace is replaced by a simulated proof π_{sim} generated by the challenger using the trapdoor td_{NIZK} corresponding to public parameters pp_{NIZK} computed at initialization of the game. We have:

$$Pr(\mathcal{G}_0) = Pr(\mathcal{G}_1)$$

Proof. Straightforwardly follows from definition of perfect zero-knowledge. ◀

Game \mathcal{G}_2 is the same as \mathcal{G}_1 with the blinded coins $[\tilde{c}_j]_{j=1}^m$ in validator's trace replaced by random values of corresponding form. More precisely, each blinded coin $\tilde{c}_j = (c_{c_j}, h, h^{H(c_j)}g^b)$ with $c_{c_j} = \text{PRF}_s(c_j)$ and $h = \mathcal{H}(c_{c_j})$, is replaced by a tuple $(r_j, \mathcal{H}(r_j), g_j)$ where r_j is a random element of the field \mathbb{F}_q , and g_j a random element of \mathbb{G}_1 . \mathcal{C} also replaces the proof π_{aux} of the algorithm BLIND by a simulated one. We have:

$$|Pr(\mathcal{G}_1) - Pr(\mathcal{G}_2)| \leq 1 - (1 - \text{Adv}^{\text{PRF}})^2$$

Proof. The seed s used in the PRF to compute c_{c_j} being sampled at random in Algorithm BLIND, it is indistinguishable from the random element r_j . Moreover, element b being chosen at random in algorithm BLIND, g^b is perfectly indistinguishable from a random element of \mathbb{G}_1 and so is $h^{H(c_j)}g^b$. Finally, π_{aux} is also perfectly indistinguishable from a simulated proof by hypothesis and thus, the above upper bound follows. The right-hand side is quadratic in Adv^{PRF} since two coins are created (the payment coin and the redeem coin). ◀

Game \mathcal{G}_3 is the same game as \mathcal{G}_2 with the serial numbers $[sn_i]_{i=1}^n$ in validator's trace replaced by truly random values of corresponding size (concretely, an elements uniformly sampled in \mathbb{F}_q). We have:

$$|Pr(\mathcal{G}_2) - Pr(\mathcal{G}_3)| \leq 1 - (1 - \text{Adv}^{\text{PRF}})^n$$

Proof. The seed ask is chosen at random by the oracles (line 1 of Algorithm 2), and evaluated on the values ρ_i which are unique with overwhelming probability, so the upper bound directly follows from the pseudorandomness property of the used PRF. ◀

Game \mathcal{G}_4 is the same as \mathcal{G}_3 except that the challenger \mathcal{C} modifies not only the generic validator's trace, but also the receiver's trace in case it is controlled by \mathcal{A} . \mathcal{C} replaces the threshold signature σ received from the oracle by a signature σ_{sim} generated by \mathcal{C} with $sk_{agg} = (x, y)$. \mathcal{C} generates σ_{sim} sampling a random $\sigma_{sim,1} \in \mathbb{G}_1$, computing $\sigma_{sim,2} = \sigma_{sim,1}^{x+y \cdot H(m)}$ and letting $\sigma_{sim} = (\sigma_{sim,1}, \sigma_{sim,2})$.

$$|Pr(\mathcal{G}_3) - Pr(\mathcal{G}_4)| \leq \text{Adv}^{\text{DDH}}$$

Proof. The randomisation step in algorithm AGGREGATE, takes (σ_1, σ_2) and compute (σ_1^t, σ_2^t) with t uniformly sampled in $\{0, |\mathbb{G}_1| - 1\}$. The not randomized signature is thus $(h, h^{x+y \cdot H(m)})$, the randomized signature is $(h^t, h^{(x+y \cdot H(m)) \cdot t})$ and the simulated signature is $(z, z^{x+y \cdot H(m)})$, with z a random element of \mathbb{G}_1 .

With the knowledge of h obtained from the blinded coin in the validator's trace, distinguish the randomized and the simulated signature is equivalent to distinguish $(h, h^t, h^{t(x+y \cdot H(m))})$ from $(h, z, z^{x+y \cdot H(m)})$. The DDH problem reduces to it, as it is argued in [36]. ◀

Game \mathcal{G}_5 is the same as \mathcal{G}_4 with a modification of the coin sent to the receiver in case it is controlled by \mathcal{A} . The coin $\mathbf{c}_{sim}^{new} = (v, apk_A, \rho_{sim}^{new})$ has the same value v and public key apk_A as in the game \mathcal{G}_4 (\mathcal{C} extract it directly from the adversary's queries without calling any oracle), but ρ_{sim}^{new} is a field element chosen at random in \mathcal{G}_5 . We have:

$$|Pr(\mathcal{G}_4) - Pr(\mathcal{G}_5)| \leq \text{Adv}^{\text{PRF}}$$

Proof. ρ^{new} is computed from a PRF with a seed ρ_{seed} chosen uniformly at random (line 4 Algorithm 6) and evaluated on an input $sn_1 || sn_2 || \dots || sn_n || j$ which is unique with overwhelming probability, so upper bound directly follows from definition of a PRF. ◀

C.4 Regulation impact

The various proofs provided above still apply to the regulated version, as the *transfer* operation is handled in exactly the same way by the validators. The main problem that could arise would concern the privacy. While the NIZK proof differs between the regulated and non-regulated versions, its public inputs remain the same. The compliance coin is processed like any other coin from an external perspective. Therefore, the revealed data includes the blinding of the new compliance coin and the serial number of the old compliance coin, just as it would for a standard coin in the non-regulated NIZK.

D Detailed related works

Table 3 Detailed comparison of PaxPay vs. Zcash [18], Lelantus [28], Quisquis [20], Zef [6], UTT [42], PRCash [45], PEReDi [38] and PARScoin [39].

	Zcash	Lelantus	Quisquis	Zef	UTT	PRCash	PEReDi	PARScoin	PaxPay
PRIVACY PROPERTIES									
Confidential transfers: Yes No Partial	●	●	●	● ¹⁶	●	● ¹⁷	●	●	●
Sender-anonymous transfers: Yes No Partial	●	●	●	○	●	● ¹⁸	● ¹⁹	● ²⁰	●
Receiver-anonymous transfers: Yes No	●	●	●	●	●	●	●	●	●
Unlinkable transfers: Yes No	●	●	●	○	●	● ²¹	●	●	●
Anonymity strategy: Full AS (Anonymity Set)	Full	AS	AS	Full	Full	Full	Full	Full	Full
MODEL ASSUMPTIONS									
Asynchronous network: Yes No	○	○	○	●	●	○	○	●	●
Correct validators model: H (Honest) SH (Semi-Honest)	SH	SH	SH	SH	SH	SH	H	H	SH
REGULATION FEATURES									
Limited held amount per user	○	○	○	○	○	○	●	○ ²²	○
Limited spendable amount per tx	○	○	○	○	○	● ²³	●	●	●
Limited spendable amount in total	○	○	○	○	● ²⁴	○	●	●	●
Full asset tracing	○	○	○	○	○	○	●	●	○
Sanction list	○	○	○	○	○	○	○	○	●
Provable transaction history	○	○	○	○	○	○	○	○	●
PERFORMANCE									
Transaction throughput (tx/s)	25	∅	∅	88 ²⁵	235 ²⁶	∅ ²⁷	∅	∅	925 ²⁸
Transaction latency (s)	1000 ²⁹	∅	∅	< 1	< 1	∅	∅	∅	< 1
NIZK Proving time (ms)	21K	2378	2110	438	60	100	3100	392	6959
NIZK Verification time (ms)	9 ³⁰	40 ³¹	251 ³²	142 ³³	49 ³⁴	96 ³⁵	518 ³⁶	159 ³⁷	5 ³⁸

Consensus-free payment systems. Guerraoui *et al.* [25] proposed a payment system that relies on the secure broadcast primitive [10, 33] instead of consensus. The key property exported by secure broadcast is *source ordering*: if two correct validators deliver two messages from the same sender, then they should deliver them in the same order. To prevent double spending, the transactions are broadcast with sequence numbers, incremented each time the user issues a new transaction. Astro [13], FastPay [5], Zef [6] and UTT [42] use the same idea to build a payment system without consensus, on top of an asynchronous network.

Private payment systems without regulation. Zerocash [7], Zexe [9], Monero [35], Quisqus [20] and Lelantus [28] are examples of payment systems that provide *privacy* on top of blockchains (and thus require consensus and synchronous network). They rely on NIZK or ring signatures to ensure anonymity of transactions, hiding the sender, the receiver and the

¹⁶The sender of the output coin will know a lower bound (or the exact value) on the amount of the future payment during which the receiver will spend the coin.

¹⁷The sender of the output coin will know a lower bound (or the exact value) on the amount of the future payment during which the receiver will spend the coin.

¹⁸The receiver knows the sender.

¹⁹The receiver knows the sender.

²⁰The receiver of the transaction is the only one to know the sender's identity. The protocol could be sender-anonymous but it would weaken the regulatory enforcement. In our construction, we circumvent this problem with the *SanctionList*, proving that the sender is legit within the NIZK.

²¹Validators can link transactions emitted during the same epoch by the same user.

²²See Section 8, in the **Regulation** paragraph

²³PRCash as a limit defined per epoch, not per transaction.

²⁴UTT implements an *anonymity budget*, that limits the amount a user can spend anonymously. When the limit is reached, the user can still spend money but has to reveal the content of the transaction to an external actor called Auditor.

²⁵Estimate from Zef [6] paper on the throughput for each validator running on 16 one-core shard, each shard running on a m5.8xlarge EC2 (AWS) machine equipped with an Intel Xeon Platinum 8175 CPU.

²⁶Tests running on 16 CPU cores of a c5.4xlarge EC2 instance equipped with an Intel Xeon Platinum 8124 CPU

²⁷PRCash paper provide a theoretical throughput that is only computed based on the NIZK verification time, and provide no more information this metric.

²⁸Tests running on 16 CPU core of a m5.8xlarge EC2 instance equipped with an Intel Xeon Platinum 8175 CPU, equivalent to Zef.

²⁹Considering immediate inclusion in a block and 15 block validation with 1min15 average block production time

³⁰Benchmark done on a computer equipped with an Intel Core i7 3.5GHz CPU, 32 GB of RAM and Linux.

³¹Over an anonymity set of 2^{16} and a proof batch size of 50. Computed on an Intel I7-4870HQ CPU.

³²Anonymity set of size 64. Computed on an Intel Core i7 2.8GHz CPU.

³³Tests running on one CPU core of a m5.8xlarge EC2 instance equipped with an Intel Xeon Platinum 8175 CPU. The verification time has only a relative impact on the scalability of the system since a validator can be efficiently distributed over several machines, increasing the transaction throughput.

³⁴Tests running on a c5.4xlarge EC2 instance equipped with an Intel Xeon Platinum 8124 CPU, supposedly on one CPU core

³⁵On one core of an Intel Core i7-4770 CPU. Range proof on 2^{20}

³⁶The benchmark was made on a computer with an Intel Core i7-9850H CPU at 2.60 GHz with 16 GB of RAM using Ubuntu 20.04.2 LTS. This value is computed with the formula provided in the paper for 8 byzantine validators and all compliance parameters set to $2^{20} \approx 1,000,000$ (Considering a maximum value of 1000 coins with precision of 10^{-3}).

³⁷The benchmark was made on a computer with an Intel Core i7-9850H CPU at 2.60 GHz with 16 GB of RAM using Ubuntu 20.04.2 LTS. This value is computed with the formula provided in the paper for 8 byzantine validators and the maximum receivable amount set to $2^{20} \approx 1,000,000$ (Considering a maximum value of 1000 coins with precision of 10^{-3}).

³⁸Benchmark on one CPU core of an Intel i7 2.6 GHz CPU. The verification time has a relative impact on the scalability of the system since a validator can be efficiently distributed over several machines, increasing the transaction throughput.

amount of payments.

In Zerocash and Zexe [7, 9], users can exchange coins that are represented as secret *commitments*. These coins are equivalent to the unspent transactions (UTXO) of Bitcoin. These commitments are organized in a shared Merkle tree. If a user wants to spend a coin, she provides an NIZK that confirms the inclusion of her commitment in the Merkle tree. Then the user reveals the *nullifier* used in the commitment, which is added to the list of spent nullifiers. This list ensures that double spending cannot take place. The users engage in consensus to agree both on the list of valid commitments and the list of nullifiers.

Monero [35] describes a payment system designed for private transactions. Like Zerocash, Monero operates on a blockchain where users manage UTXOs. However, while Zerocash conceals the sender or receiver's identity among all users of the system, Monero limits this obfuscation to a smaller group, known as the *anonymity set*. Monero achieves this through the use of ring signatures. These signatures enable any member of the anonymity set (referred to as the *ring*) to sign a transaction, confirming that she belongs to the set without disclosing her identity. To prevent double spending, the ring signature scheme incorporates a feature called *linkability*, which ensures that if the same member of the ring signs multiple transactions originating from the same ring, it can be detected.

Quisquis [20] presents another private payment system built on a blockchain. Similar to Monero, Quisquis leverages anonymity sets to hide user identities during transactions. A distinguishing feature of Quisquis is its use of *updatable keys*, which enable users to update public keys without altering the associated secret keys, so that the updated public keys are indistinguishable from ones that are freshly generated with new private keys. To initiate a payment, the sender updates the input keys and provides an NIZK that verifies, among others, the correctness of the update. This mechanism ensures that a public key cannot be used as an input more than once, effectively preventing double spending. Compared to Monero, Quisquis offers stronger privacy guarantees in certain scenarios. Indeed, there are specific transaction configurations in Monero where an attacker could potentially de-anonymize the transaction, by comparing intersections of anonymity sets. In contrast, such de-anonymizations are precluded in Quisquis thanks to the updatable public keys, although it also relies on anonymity sets.

Lelantus [28, 29] proposes a private payment system built on a blockchain, and uses anonymity sets to provide privacy. Lelantus uses NIZK to provide privacy but requires less advanced cryptographic assumptions than Zerocash. For instance, it does not require any trusted setup. In the protocol, validators can verify *batches* of proofs, which help maintain a good overall performance by limiting the average verification time of a single proof.

All payment systems mentioned so far are built on a blockchain and, thus, require consensus. Baudet et al. [6] describe Zef, a private payment system, that assumes an asynchronous network but only provides *partial* anonymity and confidentiality: when a payment is made, the receiver and the amount are hidden but the sender is known (thus, no sender-anonymity). Also, Zef does not guarantee that payments amount are entirely confidential. Indeed, when a user makes a payment, she reveals coin commitments that were previously sent by other users, and thus known by other users. These previous senders can, therefore, deduce some information about the amount of the payments being made if they recognize a spent coin that they created. To improve performance, the transaction data in Zef is distributed over a set of *authorities*. Each authority can be sharded — *i.e.* distributed over several machines as in Astro [13]. The throughput of Zef grows linearly with the number of shards for each authority, which allows it to be “arbitrarily” scalable.

Private payment systems with regulation. PRCash [45], Platypus [46], PEReDi [38],

and PARScoin [39] incorporate certain regulatory features into private payment systems. A common element in their designs is the reliance on NIZK.

Garman et al. [22] was one of the first work to address regulation by enforcing *policies* such as fixing a spending limit on a system with a similar construction as Zerocash. Their system has not been implemented and as it is build on Zerocash it suffers from the same major drawbacks.

PRCash [45], similar to some of the systems mentioned above, operates with commitments that are consumed and created during each transaction. PRCash only offers partial privacy. Like Zef, it provides only partial confidentiality, as some details about the payment amount can leak to the users who created the input commitments. Also, PRCash only ensures partial anonymity, as the sender of a payment is known to the receiver, making anonymous donations impossible. Validators can also link certain transactions, violating transaction unlinkability. Since PRCash is blockchain-based, it incorporates a concept of time defined by the number of blocks. Regulation features proposed by PRCash are limited to one: users cannot spend more than a predefined amount set by the regulator within a certain time window.

UTT [42] is an asynchronous private payment system with privacy guarantees, a low transaction latency and a high transaction throughput. Similar to PRCash, the regulation features offered by UTT are restricted to the *anonymity budget* that limits the amount of anonymous coins that a user can transfer. Compared to UTT, PaxPay offers a more comprehensive set of regulation features and exhibit better performance than UTT, most likely due to the NIZK construction.

Platypus [46] introduces regulations that includes features such as holding limits, receiving limits, and spending limits. It utilizes a type of NIZK with low verification costs. However, unlike other payment systems described in this section, Platypus is a centralized payment system rather than a distributed one.

Sarencheh et al. introduced PEReDi [38] which provides a more comprehensive regulatory framework compared to PRCash. It operates on the account basis: users have blinded accounts that are signed by a quorum of validators. Transfers are executed by creating a joint transaction between the sender and the receiver. They compute the states of their accounts after the transfer, and NIZK proofs that the accounts are updated correctly. The regulation framework enforces spending and receiving limits for each transaction and imposes restrictions on the maximum amount a user can hold. Additionally, the system allows validators to trace funds and share details of certain payments using the threshold encryption scheme. This approach has, however, several drawbacks: it assumes a synchronous network and provides only partial anonymity, as the receiver of a payment knows the sender. Moreover, the system only tolerates fewer than $1/5$ validators to be Byzantine, compared to $1/3$ in most of other systems we discussed. Also, due to the tracing capability granted to validators, $4/5$ of the *correct* (i.e., non-Byzantine) validators must also be *honest* (i.e., strictly follow the protocol and avoid seeking additional information). In contrast, most systems discussed here (as well as ours) are designed to tolerate *semi-honest* correct validators. Another noticeable drawback is the high verification time of a transaction, which limits the overall transaction throughput.

Concurrently with this submission, the same authors have proposed PARScoin [39], a payment system designed to handle stable coins exchange in a private and regulated manner. Its construction is close to PEReDi's but mitigates some of its drawbacks. PARScoin allows to issue payments without the help of the receiver to form a transaction, which turns the interaction between the sender and the receiver asynchronous. However, PARScoin still requires the non-Byzantine validators to be strictly honest to preserve user's privacy. The verification time of the NIZK is also reduced. Also, neither PRCash, PARScoin nor PEReDi

have conducted latency tests to evaluate the system’s maximum transaction throughput or assess its scalability.

E Benchmark

■ **Table 3** SNARK Proving and Verification Time on a m5.8xlarge EC2 instance (AWS) with Ubuntu 22.

	Regulated		Not Regulated	
	1 Core	16 Cores	1 Core	16 Cores
Proving time (ms)	12079 (± 19)	1624 (± 40)	5471 (± 15)	783 (± 22)
Verification time (ms)	9.60 (± 0.01)	5.98 (± 0.04)	8.97 (± 0.02)	5.69 (± 0.04)

■ **Table 4** SNARK Proving and Verification Time on a c5.9xlarge EC2 instance (AWS) with Ubuntu 22.

	Regulated		Not Regulated	
	1 Core	16 Cores	1 Core	16 Cores
Proving time (ms)	10251 (± 28)	1380 (± 29)	4613 (± 12)	698 (± 20)
Verification time (ms)	8.32 (± 0.05)	5.53 (± 0.09)	7.81 (± 0.01)	5.32 (± 0.15)

Table 3 details PaxPay NIZK benchmark on an AWS m5.8xlarge EC2 instance running Ubuntu 22. In Table 3 (repeated from the introduction), we give additional details on the comparative analysis of PaxPay with competitors. Note that the NIZK proving and verification time have been tested in the same conditions for all the protocols.