PARScoin: A Privacy-preserving, Auditable, and Regulation-friendly Stablecoin

Amirreza Sarencheh, Aggelos Kiayias, and Markulf Kohlweiss *

The University of Edinburgh and IOG, UK
firstname.lastname@ed.ac.uk

Abstract. Stablecoins are digital assets designed to maintain a consistent value relative to a reference point, serving as a vital component in Blockchain and Decentralized Finance (DeFi) ecosystems. Typical implementations of stablecoins via smart contracts come with important downsides such as a questionable level of privacy, potentially high fees, and lack of scalability. We put forth a new design, PARScoin, for a Privacy-preserving, Auditable, and Regulation-friendly Stablecoin that mitigates these issues while enabling high performance both in terms of speed of settlement and scalability. PARScoin enables privacy-preserving transactions where user identities, transaction values, and links to past transactions are hidden.
Our system ensures auditability and regulation-friendliness without compromising privacy. Transfers do not need to be settled on blockchains, allowing for blockchain-independent fees. The system is blockchain-agnostic and interoperable with multiple blockchains via standard smart contract-based withdrawals. Our construction is distributed using threshold cryptography, avoiding single point of trust and failure, and is analyzed in the Universal Composition (UC) framework for secure integration with other systems. PARScoin supports self-custody, avoiding restrictions like address blacklisting or coin destruction, and demonstrates real-world efficiency based on our performance analysis.

Keywords: Stablecoin · Privacy · Regulation · Auditing · Decentralized Finance · Cryptography · Transaction Fee · Scalability · Universal Composition.

1 Introduction

Stablecoins are digital assets engineered to uphold a consistent valuation against a designated reference point. They have surfaced as a foundational element in Decentralized Finance (DeFi) and the cryptocurrency ecosystem in general. They play a crucial role in addressing the volatility risks associated with cryptocurrencies, which can hinder their functionality as global currencies.

* The author order follows the Blockchain Technology Laboratory policy with junior and corresponding author Amirreza Sarencheh and senior authors Aggelos Kiayias and Markulf Kohlweiss.

For an instrument to qualify as a currency three fundamental functions should be fulfilled: serving as (i) a store of value, (ii) a unit of account, and (iii) a medium of exchange [42]. Stability of price is an essential property that relates to all these three fundamental functions. A volatile asset such as a cryptocurrency can be unsuitable as a currency because volatility makes it challenging to preserve wealth, set prices for goods and services, and ensure consistent compensation during transactions. To address this problem, stablecoins were proposed to provide this stability by maintaining a steady price relative to a specified reference point such as a fiat currency or a basket of commodities.

The stablecoin market has witnessed substantial growth, with a total market capitalization of over $127 billion as of July 2023 [17].

Stablecoins can be categorized into three types based on how they maintain their peg:

- fiat-backed (e.g., USDT [53], and USD Coin [55])
- crypto-backed (e.g., MakerDAO's DAI [41], and sUSD [52])
- algorithmic stablecoins (e.g., Ampleforth AMPL [4])

Each type offers unique advantages and challenges in achieving price stability. Fiat-backed stablecoins like Tether (USDT) [53] and USD Coin (USDC) [55] rely on traditional fiat currency reserves (which, in practice, may include assets that supposedly easily translate to fiat). Crypto-backed stablecoins like MakerDAO's DAI [41] and sUSD [52] use cryptocurrencies for backing and smart contracts for stability. Algorithmic stablecoins, like Ampleforth (AMPL) [4], depend on algorithms to regulate supply and demand dynamics. However, their stability remains a subject of scrutiny due to their (in)ability to absorb shocks during adverse macroeconomic conditions, as exemplified recently by the Terra-Luna collapse [47].

In this work, our emphasis is on fiat-backed stablecoins. It is by far the largest class in terms of market capitalization [18] and the one for which it is easiest to argue how it can maintain its peg to its underlying reference point as it is feasible (in principle) to perform an audit of the fiat currency and other assets that are held on custody vis-à-vis the amount of stablecoins issued.

We abstract fiat-backed stablecoins as follows: an *issuer* offers a way to digitize an amount of fiat currency that is deposited by a *user* to a *custodian*. Subsequently, users can exchange any amount of stablecoin they possess for services or other functions. Further, at any time a user can request to withdraw her stablecoin funds by "burning" them and creating a claim (we call this a "proof of (stablecoin) burn") that enables the user to withdraw the funds from the custodian in the form of fiat currency. An *auditor* is capable of monitoring the total amount of stablecoin issued by the issuer so that it can verify with the custodian that a sufficient amount of fiat currency (or equivalent assets) is available to cover the issuer's liabilities to all stablecoin holders. Finally, a *regulator* may impose additional regulatory constraints such as Know Your Customer (KYC), Anti-Money Laundering (AML), and Combating the Financing of Terrorism (CFT) to be performed by the issuer.

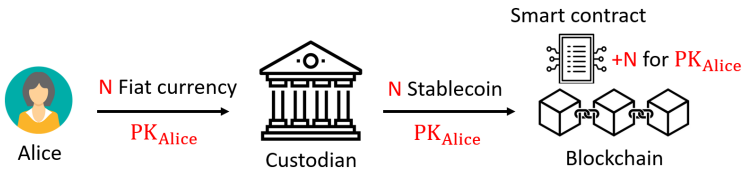

Fig. 1: Canonical fiat-backed stablecoin mechanism (e.g., USDC [55] and USDT [53]).

The canonical way to implement a fiat-backed stablecoin is to have the issuer create a smart contract on a blockchain such as Ethereum [22] and partner with a custodian such as a bank that can accept user deposits and maintain fiat reserves serving as the stablecoin's collateral, see Figure 1. Due to the transparency of the underlying blockchain, the amount of stablecoin issued (and the associated public key) are always available to the auditor who can subsequently investigate the reserves held by the custodian to ensure the matching funds exist. Users can transfer stablecoin to each other by sending transactions to the issuing contract, an operation whose integrity is provided by the underlying blockchain platform.

There are several shortcomings associated with the aforementioned approach for realizing fiat-backed stablecoins:

- There is no **privacy** offered to participating users; indeed, users are typically assigned a pseudonym within the smart contract, and each time they perform a transaction their pseudonym can be associated with other attributes of their identity. Due to the transparency of the underlying blockchain, this lack of privacy seems like an inherent problem.
- There are **transaction fees** due for each stablecoin transfer between users; contrary to standard user-to-user fiat transfers, engaging with the issuing smart contract requires paying a transaction fee, typically in the underlying cryptocurrency of the platform. This has two important downsides; first, it imposes a requirement that each user should have a reserve of such cryptocurrency in addition to the stablecoin in order to be able to use their stablecoin wallet. Second, it puts users to compete for blockchain processing capacity with other applications; in case other blockchain participants (e.g., DeFi users) wish to issue large numbers of transactions they can push the system to increase its transaction fees making it hard to engage with the issuing smart contract for stablecoin operations. This also seems like an inherent problem, since it is infeasible to engage with a blockchain contract without incurring a transaction fee in a "layer 1" blockchain.
- Finally, the **performance and scalability** of the construction is questionable as it directly relies on a L1 for settling all transactions.

1.1 Our results

Motivated by the above issues, we propose a new design for a (fiat-based) stablecoin that, as far as our understanding goes, for the first time mitigates the short-

comings we described above together with other important properties. In detail, our construction **PARScoin** (**P**rivacy-preserving, **A**uditable, and **R**egulation-friendly **S**table**coin**), offers the following.

- **Privacy-preserving operation**. Stablecoins are issued to users who can transfer them to other users without revealing their identities, the transaction value, or even linking these actions to any past transactions.
- **Auditability & regulation-friendliness**. The backing of the stablecoin can be audited without hurting the privacy of users. Furthermore, being issued stablecoin or transferring it between users can be subject to regulatory checks such as KYC, AML, and CFT that can be integrated into the system without hurting the privacy-preserving operation and in an efficient way.
- **Blockchain independent fees**. Stablecoin transfers are executed *user-to-user* via the certification of a quorum of issuers and they do not require to be settled "on chain." This decouples stablecoin transfers from any underlying blockchain settlement mechanism and hence any fees incurred can be decided independently of other applications.
- **Blockchain-agnosticism & interoperability**. The system operates purely independently of any underlying blockchain. Moreover, it can interoperate with any number of them, by enabling the custodian to deploy standard (non-private) smart contract-based withdrawals (e.g. as ERC-20 tokens).
- **UC security**. We formulate a Universal Composition (UC) functionality [16] for (fiat-based) stablecoins and realize it. This facilitates for our protocol secure composition with other systems – an important consideration in the blockchain and DeFi setting.
- **Distributed operation**. The issuance, burning, and transfer operations of the stablecoin together with ensuring regulatory compliance are distributed among a set of independent entities in a threshold cryptographic manner.
- **Efficiency & scalability**. Based on our performance analysis, our design demonstrates practical real-world efficiency through the usage of suitable protocol techniques and efficient building blocks.
- **Self-custody**. Users possess complete control and ownership of their stablecoin, marking a departure from the usual fiat-backed stablecoins where a user's stablecoin balance is documented within a smart contract, subjecting it to potential restrictions such as address blacklisting or coin destruction.

1.2 Related works

Each stablecoin type offers unique advantages and challenges in achieving price stability. Fiat-backed stablecoins like USDT [53], and USD Coin (USDC) [55] rely on traditional fiat currency reserves (note that in practice this includes assets that supposedly easily translate to fiat), while crypto-backed stablecoins like MakerDAO's DAI [41], and sUSD [52] use cryptocurrencies for backing and smart contracts for stability. Algorithmic stablecoins, like Ampleforth (AMPL) [4], depend on algorithms to regulate supply and demand dynamics. However, their stability remains a subject of scrutiny due to their (in)ability to absorb shocks

during adverse macroeconomic conditions as exemplified by the Terra-Luna collapse [47].

Within the realm of fiat-backed stablecoins, two noteworthy ones are USDT [53], and USDC [55]. USDT was introduced in 2014 by Tether, and initially it was a token constructed atop Bitcoin (via the Omni platform), subsequently, it enabled compatibility with other blockchains. USDC is a token integrated within the Stellar blockchain [39].

In the context of the second category, which is crypto-backed stablecoins, DAI [41] and sUSD [52] stand out. DAI is a cryptocurrency issued by the Maker Protocol, which stands as one of Ethereum's prominent DeFi applications. DAI is not underpinned by fiat assets, and its issuance operates without the involvement of a central authority. The creation of DAI transpires when a user locks a designated cryptocurrency asset as collateral within a smart contract. In return for providing collateral, the user receives a stability fee, which is a pivotal component of DAI's stability mechanism. sUSD is issued by Synthetix, a decentralized liquidity provisioning protocol. Synthetic assets are collateralized by stakers via Synthetix Network Token (SNX), which when locked in a staking contract allows the issuance of synthetic assets (synths). The Synthetix protocol facilitates direct conversions between Synths using the smart contract, eliminating the necessity for counterparties.

AMPL [4] and Basis [3] are examples of the third category, algorithmic stablecoins. AMPL represents a purely algorithmic stablecoin introduced by the Ampleforth platform. AMPL's supply is dynamically managed through an algorithmic rebasing mechanism designed to exert countercyclical influence on market fluctuations. In cases where the market price of AMPL surpasses the predefined threshold, the algorithm initiates a corrective measure by expanding the token supply, thereby reducing the price. Conversely, if the market price of AMPL falls below the threshold, the algorithm takes corrective action by contracting the token supply, consequently bolstering the AMPL price. The Basis protocol employed a mechanism of expanding and contracting the supply as an indirect means to stabilize its peg. However, it is worth noting that the Basis project faced challenges in launching due to regulatory concerns.

All three types of stablecoins, as mentioned earlier, have privacy, transaction fee and scalability problems. Furthermore, both USDT and USDC tokens are administered by centralized issuers (stablecoins within the last two classifications offer the advantage of decentralization). Additionally, the regulation-friendliness of all these stablecoins remains unclear.

Zerocash [8], Monero [44], Quisquis [24], Zether [13], Platypus [57], and PEReDi [35] are cryptographic schemes for anonymous payments. In those schemes a sender generates a Zero-Knowledge Proof (ZKP) to demonstrate the well-formedness of transaction information. The transaction conceals the transferred value within a commitment or encryption. The sender in their ZKP provides a proof of knowledge of secret values, including the used randomness in commitments/encryptions. Additionally, the sender proves they possess adequate funds for the transaction (e.g., ensuring the balance remains non-negative post-

transaction) or the sum of input coins equals sum of output coins (no money is generated/destructed) and that the value sent is positive and within a defined range.

Zerocash [8] is a UTxO (Unspent Transaction Output)-based fully anonymous payment system. A Zerocash transaction is linked to the set of all coins (anonymity set), and the coin being spent is from within this set. The sender demonstrates ownership of one of these coins without disclosing the coin itself. Importantly, the transaction size is constant remaining independent of the size of the anonymity set. In Zerocash, each coin is assigned a unique serial number. This number is made public on the blockchain at the time of spending. Consequently, any transaction that generates a previously disclosed serial number is invalidated. In summary, Zerocash offers full anonymity in a UTxO-based system where the anonymity set includes all coins in the system.

Monero [44], similar to Zerocash [8], is also UTxO-based. Monero employs ring signatures, where the anonymity set comprises a collection of public keys utilized in the ring signature created during the signing of a new transaction. This signature proves that the signer knows the secret key of one of the public keys in the group of public keys (anonymity set) without revealing the public key itself. Monero employs one-time signatures for double-spending prevention. The key associated with a UTxO is singularly used to create a valid ring signature for spending that UTxO. This is achievable because an image of the public key is recorded on the blockchain when the UTxO is expended. As a result, any transaction linked to an already existing image will be denied.

Quisquis [24] presents a hybrid approach by integrating UTxOs with accounts. Each user maintains an account, but transactions are processed using UTxO inputs instead of direct account debits. The double-spending prevention is based on the public key. Since a transaction alters all input public keys, each key is guaranteed a unique appearance on the input side of any transaction. Therefore, the recurrence of an input key directly shows a double-spending attempt. Quisquis integrates anonymity sets with updatable keys to maintain anonymity. This approach ensures that the overall balances within the anonymity set remain unchanged (while adjusting the sender's and recipient's balances according to the transaction value). All involved public keys are refreshed. The refreshment of keys provides unlinkability.

Platypus [57], and PEReDi [35] offer privacy-preserving operations combined with various levels of regulation-friendliness. While we share some architectural similarities with PEReDi which is also distributed there are important differences: In Platypus and PEReDi, both the sender and receiver must interact to complete transactions, meaning that offline receivers cannot access funds. This is sharply different from PARScoin, which supports non-interactive payments.

In PEReDi, both the sender and receiver must actively participate in completing transactions, thereby preventing offline receivers from accessing funds. This contrasts with PARScoin, which facilitates payments where the receiver is offline. The transition to such "non-interactive" transactions presents several technical challenges which we describe below.

In PEReDi, account information is concealed, and only the account holder possesses the necessary witness to submit a Zero-Knowledge Proof (ZKP) for advancing the account state in a privacy-preserving manner. This implies that the sender cannot advance the receiver's account state if the receiver is not available during payment. Consequently, the system mandates the receiver's continuous online presence to receive funds. Our work overcomes this challenge. In our approach, the sender advances their account state, and the sender's ZKP leaves a cryptographic element as a footprint on the system's global state maintained by distributed issuers. This allows the offline receiver to later claim the funds once they come online with the help of issuers by proving certain conditions.

The mechanism that solves this problem must be carefully designed to guarantee the funds transfer without requiring maintaining a complete ledger. Additionally, the funds should be claimable only once by the associated receiver to prevent malicious receivers from executing replay attacks. This requires implementing checks within the distributed protocol to efficiently detect and thwart any attempts at reusing transaction elements fraudulently.

Moreover, the construction must ensure the integrity of the transaction, guaranteeing that no additional money is generated and that no funds are destroyed. It must also be designed to safeguard privacy. Specifically, it must obscure the receiver's identity, the transaction value, and its creator's identity (which is the sender). The sender must also efficiently prove the well-formedness of the cryptographic object in relation to their own account state transition. This requirement ensures that system validators (issuers) can verify that the transaction is legitimate, namely, that the value has been appropriately deducted from the sender's account and correspondingly reflected in the cryptographic element.

Whenever the receiver comes online, they must be able to identify the objects that belong to them and subsequently submit an efficient ZKP demonstrating that each cryptographic object belongs to them. This proof must not reveal any information about their identity, but merely confirm their ownership. This ownership confirmation is necessary to transfer the funds embedded within the cryptographic object to the receiver's account.

Furthermore, we should meet the privacy requirements associated with making payments non-interactive (as described above) by relying solely on sigma-protocols for both sender and receiver ZKPs (except range proofs). This makes our approach suitable for real-world applications. We highlight that in privacy-preserving schemes, general-purpose ZKPs can be trivially used to conceal sensitive information. However, these proofs are inefficient in terms of proof generation time (for discrete logarithm-based relations) compared to sigma-protocols. Our construction is based on sigma protocols and we use bulletproofs [14] only for range proofs.

Regarding auditability, the auditability functionality of our work is fundamentally different from PEReDi's. PEReDi's auditability pertains to privacy revocation and tracing users. However, in our work, auditability, as formally defined via our ideal functionality, ensures that the stablecoin in circulation is backed by sufficient off-chain funds, which is important for price stability. This is

not relevant to the setting of [35] where the central bank is fully trusted in terms of the money supply and is not accountable to anyone in terms of its reserves.

Specifically, our stablecoin issuance and stablecoin burn protocols ensure that the amount of stablecoin in circulation is accurately adjusted in a distributed and privacy-preserving manner whenever a new stablecoin is issued or burned. This guarantees that the system always maintains a correct and accountable state.

Moreover, our approach addresses the non-trivial challenge of integrating auditability within a privacy-preserving framework. Our protocol ensures that the issuance and burning of stablecoins occur without revealing the amount of the coin or the identity of the user involved. Additionally, stablecoin burning facilitates interoperability by allowing users to exchange stablecoins for fiat currency or digital assets across different blockchains, ensuring seamless interaction with diverse blockchain networks. To sum it up, privacy-preserving stablecoin issuance and burning, ensuring the stablecoin in circulation is correctly and efficiently updated in both these schemes, and facilitating interoperability is another novelty of our approach compared to [35].

Zether [13] operates on an account-based framework where each public key holder's balance is encrypted using ElGamal encryption under their public key. A valid transaction in Zether updates the receiver's account. Conversely, PARScoin requires receiver's confirmation before crediting their account. This feature is significant in scenarios where accepting an asset involves additional responsibilities (e.g., tax implications). Zether achieves k-anonymity, where in practice k is much smaller than the total anonymity set size n. Zether cannot support a larger anonymity set because the transaction size grows linearly with the size of the anonymity set, i.e., $\mathcal{O}(k)$, and is therefore upper bounded by the block size. Additionally, the sender's zero-knowledge efficiency in Zether is affected by the size of the anonymity set. In Zether, senders are restricted to initiating a single transaction within each epoch, where an epoch consists of n blocks. While Zether's mechanism for preventing double-spending and replay attacks does not result in an expanding tag storage (as it is reset to empty at the end of each epoch), systems like PARScoin, Zerocash [8], and PEReDi [35] experience tag storage growth, with each transaction contributing one group element to the tag storage.

Unlike Zether [13], Monero [44], and Quisquis [24] which support only k-anonymity, PARScoin, PEReDi [35], and Zerocash [8] are designed to achieve full anonymity with transaction sizes independent of the anonymity set size.

PRCash [56] is an anonymous payment system where transactions are verified in a decentralized manner. PRCash aims to ensure privacy and some level of regulatory compliance. Nevertheless, it does not offer complete anonymity because validators can link different transactions.

Aztec [5] is a layer two zk-Rollup on Ethereum that uses succinct arguments and ZKP to offers scalability and privacy for smart contracts. While a step in the right direction with respect to our problem, implementing a stablecoin via an Aztec wrapping of ERC-20 (or similar contract) comes with important downsides

compared to our approach. First, Aztec necessitates resolution in L1, incurring time-related expenditures (and fees for submission to L1). Second, Aztec uses a generalized ZK programming language, Noir, that employs zk-SNARKs, and this proof generation can become too costly for stablecoin transfers (see Section 5 for our ZK implementation details).

2 Stablecoin desiderata and formal modeling

2.1 Stablecoin entities

- The custodian: The custodian plays a critical role in maintaining the stablecoin's value by managing and safeguarding assets like USD and EUR, and controlling the issuance and redemption of stablecoins based on collateral deposits and burns.
- Issuers: Issuers, who are authorized and independent entities, manage user onboarding, stablecoin transactions, and regulatory compliance, allowing the custodian and regulator minimal involvement except in fiat currency transactions, and are accountable for maintaining system integrity[1].
- The auditor: The auditor regularly evaluates the custodian to ensure that the reserve assets are sufficient and well-managed, including the enforcement of strong security measures to protect against theft, cyberattacks, and other risks.
- The regulator: The task of validating transactions for regulatory compliance is assigned to a consortium of issuers to reduce the risk of a single point of failure, allowing the regulatory body to operate minimally or remotely.
- Users: Participants can assume roles as either the sender or the recipient of stablecoins. They can encompass both individuals and organizations.

Security and privacy desiderata. In terms of privacy, we aim to protect privacy by hiding *all transaction metadata* in user-to-user transactions even against an adversary who controls *all entities* in the system who are not counterparties in a transaction. Operations such as issuing stablecoin and burning stablecoin are visible to the auditor (but not to other parties) so it is possible to keep track of the amount of stablecoin circulating in the system (which is necessary for the stability of the price). Finally, in terms of the integrity of transactions, we assume that the adversary controls a number of issuer entities that is below a given a threshold (which is a system parameter).

2.2 Stablecoin requirements

In this section, we informally describe the security requirements of stablecoin systems.

[1] Our focus in this work is on fiat-backed stablecoins. We are already in a scenario where an off-chain, possibly distributed, entity is trusted for stablecoin issuance. Given this, we can rely on a group of issuers for off-chain fast settlement in a privacy-preserving manner.

- **Auditing:** Auditing custodians is crucial for confirming fiat currency backing, increasing transparency, and establishing trust among users, investors, and regulators, thus preventing risks like insolvency and ensuring stablecoin stability through regular, independent reviews. [6, 38].
- **Regulatory compliance:** The term includes facets like KYC, which verifies user identities to prevent fraud and enhance transparency and credibility in digital currencies, and AML/CFT measures, which are essential for preventing money laundering and terrorist financing, thus maintaining regulatory compliance and strengthening stablecoin integrity [43, 54].
- **Full privacy:** This property encompasses four aspects [23, 35]. (i) *account privacy:* Protects the confidentiality of user accounts, preventing exposure of financial balances and other sensitive data within user accounts to network participants. (ii) *identity privacy:* Maintains the anonymity of senders and receivers in transactions, and makes tracing their transaction participation impossible. (iii) *transaction privacy:* Ensures the amount transferred remains hidden. (iv) *unlinkability:* Stops connections between transactions and real-world identities, and prevents tracing the source of funds in subsequent transactions.
- **Avoiding single point of trust/failure:** In decentralized finance (DeFi) and blockchain technology, designing stablecoin systems without a single point of trust or failure is critical. This vulnerability could compromise decentralization, security, and stability. Distributing trust and responsibility across a network enhances system robustness and resilience, reducing risks from attacks and failures.
- **Blockchain agnosticism:** A stablecoin system that is blockchain agnostic has the advantage of broader adoption and enhanced sustainability within the diverse blockchain landscape. This approach allows the system to adapt to new technologies without being tied to the developments of any single blockchain platform, reducing risks and ensuring flexibility.
- **Interoperability:** Blockchain agnosticism facilitates interoperability, allowing different blockchain networks to work together seamlessly. This interoperability is crucial as it promotes inclusivity and liquidity across various blockchain ecosystems.
- **Integrity:** This requirement ensures that user states cannot be modified by unauthorized entities. It includes mechanisms to prevent double-spending, guaranteeing that each transaction updates the user's account balance accurately.

2.3 Notations

In this paper, we employ diverse notations, which are summarized in Table 1 and Table 2.

2.4 Stablecoin formal model

We define a privacy-preserving, auditable, and regulation-friendly stablecoin system in the form of an ideal functionality $\mathcal{F}_{\mathsf{PARS}}$, which formally captures the

Symbol	Description	Symbol	Description
U, v	User, Transacting value	$\mathcal{D}_i$	Local database of the i-th issuer
I, I_i	Issuer, The i-th issuer	$\mathcal{I}, \mathsf{acc}$	Set of all issuers, User's account
N	Total number of issuers	$\mathsf{U_s}$, $\mathsf{U_r}$	Sender, Receiver
$\mathsf{CUS}, \mathsf{AUD}$	Custodian, Auditor	$(\mathsf{pk_A}, \mathsf{sk_A})$	Auditor's key pair
$\tilde{\chi}$	ElGamal encryption	$(\mathsf{pk}, \mathsf{sk}), \mathsf{B}$	User's key pair, its Balance
x	User's transaction counter	$\mathtt{com}$	Pedersen commitment
$\mathcal{A}$, $\mathcal{Z}$	Adversary, Environment	$\mathsf{RB.Sig}$	Randomizable-blind signature
$\mathsf{TRB.Sig}$, Γ	Threshold randomizable-blind signature, its threshold	π	Non-interactive zero knowledge proof
$\mathsf{sn}, \mathsf{tag}$	Serial number, Tag (for double-spending prevention)	$\mathtt{x}, \mathtt{w}$	Statement, Witness (of proof)

Table 1: Table of notations (part I)

relevant security properties. $\mathcal{F}_{\mathsf{PARS}}$ is parameterized by the set of identifiers of issuers $\mathcal{I}$, the custodian CUS, and the auditor AUD. Also it is parameterized by a threshold Γ. In the ideal functionality, we allow the adversary $\mathcal{A}$ to drive communication and potentially block messages.

- **Initialization.** Session identifiers are of the form $\mathsf{sid} = (\{\mathcal{I}, \mathsf{CUS}, \mathsf{AUD}\}, \mathsf{sid}')$. Initially, $\mathsf{init} \leftarrow 0$. At the end of *initialization* init is set to 1.[2]
- **User registration.** Each user U must have their account approved by the system. If issuers have already registered U, it cannot be registered again. Initially, $\mathbb{R}(\mathsf{U}) = \perp$, and once U is registered, $\mathbb{R}(\mathsf{U})$ is set to 0 denoting initial user balance $\mathsf{B} = 0$.
- **Stablecoin issuance.** CUS possesses the authority to confirm the identity U and v by providing (U, v) to $\mathcal{F}_{\mathsf{PARS}}$.[3] Initially, $\mathcal{F}_{\mathsf{PARS}}$ verifies whether the recipient of stablecoin U, is a registered user. Following each stablecoin issuance, the state of the recipient's account is updated. $\mathcal{A}$ has the potential to obstruct the progression. U and v are concealed from $\mathcal{A}$. A successful issuance operation results in an increase in the recipient's balance by v: $\mathbb{R}(\mathsf{U}) \leftarrow (\mathsf{B} + v)$, and an increase in the total value of stablecoin in circulation $\mathsf{TV} \leftarrow (\mathsf{TV} + v)$ (which is not revealed to $\mathcal{A}$). $\mathcal{A}$ is also required to submit a *unique* identifier $\tilde{\chi}$ which is assigned to each updated TV (by

[2] Afterward at the beginning of all parts of the functionality (namely *user registration*, *stablecoin issuance*, *stablecoin transfer*, *stablecoin claim*, *stablecoin burn*, *Proof of burn*, and *reserve audit*) it is checked whether init has been set to 1. If it has not been set to 1, $\mathcal{F}_{\mathsf{PARS}}$ ignores the received message. For the sake of simplicity, we have omitted the inclusion of these particulars within the functionality description.

[3] This signifies that CUS has conducted its own verifications to ensure that U has indeed made a deposit of fiat currency equivalent to v into the account maintained by CUS.

Symbol	Description	Symbol	Description
$\sigma_i^{\mathsf{blind}}$	Blind signature of the i-th issuer	σ_i	Unblinded signature of the i-th issuer
$\sigma_{\mathbb{I}}$	Aggregated signature of issuers	$\sigma_{\mathbb{I}}^{\mathsf{RND}}$	Randomized aggregated signature of issuers
$\sigma_{\mathsf{CUS}}^{\mathsf{blind}}$	Blind signature of the custodian	σ_{CUS}	Unblinded signature of the custodian
$\sigma_{\mathsf{CUS}}^{\mathsf{RND}}$	Randomized signature of the custodian	TV	Total value of stablecoin in circulation
$\mathcal{F}_{\mathsf{KeyReg}}$	Key registration functionality	$\mathcal{F}_{\mathsf{RO}}$	Random oracle functionality
$\mathcal{F}_{\mathsf{Ch}}^{\mathsf{SSA}}$	Secure and sender anonymous channel	$\mathcal{F}_{\mathsf{Ch}}^{\mathsf{SRA}}$	Secure and receiver anonymous channel
$\mathcal{F}_{\mathsf{Ch}}^{\mathsf{ISAS}}$	Insecure, sender anonymous to adversary, and sender known to recipient channel	$\mathcal{F}_{\mathsf{Ch}}^{\mathsf{IRAS}}$	Insecure, receiver anonymous to adversary, and sender known to recipient channel
$\mathcal{F}_{\mathsf{NIZK}}$	Non-interactive zero knowledge functionality	$\mathcal{F}_{\mathsf{B}}^{\mathsf{SA}}$	Sender anonymous broadcast functionality
$\mathcal{F}_{\mathsf{PARS}}$	PARScoin functionality	$\mathcal{F}_{\mathsf{B}}^{\mathsf{S}}$	Broadcast functionality

Table 2: Table of notations (part II)

recording $(\tilde{\chi}, \mathsf{TV})$). $\tilde{\chi}$ is output to $\mathcal{I}$.[4] Public-delayed output means $\mathcal{F}_{\mathsf{PARS}}$ lets $\mathcal{A}$ know the output and decide its delivery.

- **Stablecoin transfer.** The sender $\mathsf{U_s}$ submits $(\mathsf{U_r}, v)$ to $\mathcal{F}_{\mathsf{PARS}}$. After some verification (e.g., $\mathsf{U_s}$ has sufficient funds) $\mathcal{F}_{\mathsf{PARS}}$ informs $\mathcal{A}$. $\mathcal{A}$ provides a *unique* identifier ϕ, which serves as a means to document an entry. This entry takes the shape of $(\phi, \mathsf{U_s}, \mathsf{U_r}, v, b)$, signifying that $\mathsf{U_s}$ has transmitted a quantity of v stablecoins to $\mathsf{U_r}$. The inclusion of $b = 0$ signifies that this payment has not yet been claimed by $\mathsf{U_r}$ (payments are non-interactive) thereby averting double-spending.
- **Stablecoin claim.** Upon the receipt of ϕ from $\mathsf{U_r}$, $\mathcal{F}_{\mathsf{PARS}}$ checks whether the entry $(\phi, \mathsf{U_s}, \mathsf{U_r}, v, b)$ has been previously recorded or not. In the event that it is indeed registered, the $\mathcal{F}_{\mathsf{PARS}}$ proceeds to verify whether it has already been claimed by $\mathsf{U_r}$ (via checking b). Assuming all checks pass successfully, upon receiving ϕ from issuers, and adversary's confirmation on finalization of *stablecoin claim* $\mathcal{F}_{\mathsf{PARS}}$ proceeds to augment the balance of $\mathsf{U_r}$. The transmission of ϕ by issuers to $\mathcal{F}_{\mathsf{PARS}}$ signifies that they have conducted their own verifications pertaining to the identifier ϕ. This method represents a means to conceptualize the process by which for instance, issuers ascertain the presence/absence of the value ϕ within the public blockchain ledger by conducting checks.

[4] In order to make this accessible to $\mathcal{Z}$. Note that more precisely, $\mathcal{F}_{\mathsf{PARS}}$ lets $\mathcal{A}$ decide: message delivery and the order of issuers receiving the message.

Functionality Functionality $\mathcal{F}_{\mathsf{PARS}}$ – Part I

Initialization.

- Upon input $(\mathtt{Int}, \mathsf{sid})$ from party $P \in \{\mathcal{I}, \mathsf{CUS}, \mathsf{AUD}\}$: Ignore if $\mathsf{sid} \neq (\{\mathcal{I}, \mathsf{CUS}, \mathsf{AUD}\}, \mathsf{sid}')$. Else, output $(\mathtt{Int.End}, \mathsf{sid}, P)$ to $\mathcal{A}$. Once all parties have been initialized, set $\mathsf{init} \leftarrow 1$.

User registration.

- Upon receiving a message $(\mathtt{Rgs}, \mathsf{sid})$ from some party U: (i) If $\mathbb{R}(\mathsf{U}) = \bot$, output $(\mathtt{Rgs}, \mathsf{sid}, \mathsf{U})$ to $\mathcal{A}$. (ii) Else, ignore.
- Upon receiving $(\mathtt{Rgs.Ok}, \mathsf{sid}, \mathsf{U})$ from $\mathcal{A}$: (i) Ignore if $\mathbb{R}(\mathsf{U}) \neq \bot$. (ii) Else, set $\mathbb{R}(\mathsf{U}) \leftarrow 0$.

Stablecoin issuance.

- Upon receiving a message $(\mathtt{Iss}, \mathsf{sid}, \mathsf{U}, v)$ from CUS: (i) Ignore if $\mathbb{R}(\mathsf{U}) = \bot$. (ii) Else, generate a new $\mathsf{SI.idn}$ and set $\mathbb{O}(\mathsf{SI.idn}) \leftarrow (\mathsf{U}, v)$. (iii) If U is honest (resp. malicious) output $(\mathtt{Iss}, \mathsf{sid}, \mathsf{SI.idn})$ (resp. $(\mathtt{Iss}, \mathsf{sid}, \mathsf{SI.idn}, (\mathsf{U}, v))$) to $\mathcal{A}$.
- Upon receiving $(\mathtt{Iss.Ok}, \mathsf{sid}, \mathsf{SI.idn}, \tilde{\chi})$ from $\mathcal{A}$: (i) Ignore if $\mathbb{O}(\mathsf{SI.idn}) = \bot$ or there exists $(\tilde{\chi}, \cdot)$. (ii) Else, retrieve $\mathbb{O}(\mathsf{SI.idn}) = (\mathsf{U}, v)$, and $\mathbb{R}(\mathsf{U}) = \mathsf{B}$. (iii) Set $\mathbb{R}(\mathsf{U}) \leftarrow (\mathsf{B} + v)$, $\mathbb{O}(\mathsf{SI.idn}) \leftarrow \bot$, and $\mathsf{TV} \leftarrow (\mathsf{TV} + v)$. (iv) Record $(\tilde{\chi}, \mathsf{TV})$. (v) Output $(\mathtt{Iss.End}, \mathsf{sid}, \tilde{\chi})$ to $\mathcal{I}$ via public-delayed output.

Stablecoin transfer.

- Upon receiving a message $(\mathtt{Gen.ST}, \mathsf{sid}, \mathsf{U_r}, v)$ from some party $\mathsf{U_s}$: (i) Ignore if $\mathbb{R}(\mathsf{U_s}) = \bot$ or $v < 0$. (ii) Else, if $\mathsf{U_r}$ is malicious, set $\mathsf{ST.idn} \leftarrow (\mathsf{U_s}, \mathsf{U_r}, v)$. Else, generate a fresh random value ω and set $\mathsf{ST.idn} \leftarrow \omega$. Set $\mathbb{V}(\mathsf{ST.idn}) \leftarrow (\mathsf{U_s}, \mathsf{U_r}, v)$. (iii) Output $(\mathtt{Gen.ST}, \mathsf{sid}, \mathsf{ST.idn})$ to $\mathcal{A}$.
- Upon receiving $(\mathtt{Gen.ST.Ok}, \mathsf{sid}, \mathsf{ST.idn}, \phi)$ from $\mathcal{A}$: (i) Ignore if $\mathbb{V}(\mathsf{ST.idn}) = \bot$ or there exists $(\phi, \cdot, \cdot, \cdot, \cdot)$. (ii) Else, retrieve $\mathbb{V}(\mathsf{ST.idn}) = (\mathsf{U_s}, \mathsf{U_r}, v)$, and $\mathbb{R}(\mathsf{U_s}) = \mathsf{B_s}$. (iii) Ignore if $\mathsf{B_s} < v$. (iv) Else, set $\mathbb{R}(\mathsf{U_s}) \leftarrow (\mathsf{B_s} - v)$, and $\mathbb{V}(\mathsf{ST.idn}) \leftarrow \bot$ (v) Record $(\phi, \mathsf{U_s}, \mathsf{U_r}, v, 0)$. (vi) Output $(\mathtt{Gen.ST.End}, \mathsf{sid}, \mathsf{U_r}, v, \phi)$ to $\mathsf{U_s}$ via private-delayed output.

Stablecoin claim.

- Upon receiving a message $(\mathtt{Clm.ST}, \mathsf{sid}, \phi)$ from some party $\mathsf{U_r}$: (i) Ignore if $\mathbb{R}(\mathsf{U_r}) = \bot$ or there does not exist $(\phi, \cdot, \mathsf{U_r}, \cdot, 0)$. (ii) Else, retrieve $\mathbb{R}(\mathsf{U_r}) = \mathsf{B_r}$. (iii) Generate a new $\mathsf{SC.idn}$ and set $\mathbb{C}(\mathsf{SC.idn}) \leftarrow (\phi, \mathsf{U_s}, \mathsf{U_r}, v, b)$. (iv) Output $(\mathtt{Clm.ST}, \mathsf{sid}, \mathsf{SC.idn}, \phi)$ to $\mathcal{A}$.
- Upon receiving $(\mathtt{Clm.ST.Ok}, \mathsf{sid}, \mathsf{SC.idn}, \phi)$ from $\mathcal{A}$: (i) Ignore if $\mathbb{C}(\mathsf{SC.idn}) = \bot$. (ii) Else, retrieve $\mathbb{C}(\mathsf{SC.idn}) = (\phi, \mathsf{U_s}, \mathsf{U_r}, v, b)$, and ignore if $b = 1$. (iii) Else, retrieve $\mathbb{R}(\mathsf{U_r}) = \mathsf{B_r}$. Upon receiving $(\mathtt{Clm.ST.Issuer}, \mathsf{sid}, \phi)$ from each of N issuers $\mathsf{I}_i \in \{\mathsf{I}_1, \ldots, \mathsf{I}_N\}$, output $(\mathtt{Clm.ST.Issuer}, \mathsf{sid}, \mathsf{SC.idn}, \mathsf{I}_i, \phi)$ to $\mathcal{A}$. Upon receiving

($\texttt{Clm.ST.Issuer.Ok}, \mathsf{sid}, \mathsf{SC.idn}, \phi$) from $\mathcal{A}$: (i) Retrieve $\mathbb{C}(\mathsf{SC.idn}) = (\phi, \mathsf{U_s}, \mathsf{U_r}, v, b)$, and ignore if $b = 1$. (ii) Else, set $\mathbb{R}(\mathsf{U_r}) \leftarrow (\mathsf{B_r} + v)$, and $b \leftarrow 1$ in $(\phi, \mathsf{U_s}, \mathsf{U_r}, v, b)$. (iii) Output ($\texttt{Clmed.ST}, \mathsf{sid}, \phi, \mathsf{U_s}, v$) to $\mathsf{U_r}$ via private-delayed output.

- **Stablecoin burn.** The user U submits a value v to $\mathcal{F}_{\mathsf{PARS}}$ along with a label ℓ. The purpose of this label, ℓ, is to specify the type of asset against which U intends to initiate the burning process for withdrawal. U has the option to burn their stablecoins in exchange for fiat currency withdrawal or any other digital asset (that can be confirmed by CUS). The latter option allows the user to obtain digital assets on any particular blockchain, the selection of which is determined exclusively by the label ℓ. This operational approach facilitates interoperability within our model. After burning the stablecoin, the total value in circulation TV is updated, and, similar to the *stablecoin issuance* protocol, an $\tilde{\chi}$ value is assigned to the newly updated amount. It's important to note that the amount that has been burned has not yet been withdrawn from the custodian. However, our focus here is not on micromanaging this procedure on the custodian side, as it is beyond our formal modeling scope. The burning process is a private operation to ensure that a potentially malicious custodian does not have visibility into the burned value. Furthermore, users have the option to immediately withdraw from the custodian upon burning. $\mathcal{A}$ provides η that serves as a proof of burn. It is crucial to emphasize that $\mathcal{F}_{\mathsf{PARS}}$ ensures the freshness of η before proceeding further. $\mathcal{F}_{\mathsf{PARS}}$ maintains a record of $(\eta, \mathsf{U}, v, \ell, q)$, which demonstrates that U has executed the burning of stablecoins amounting to v in association with a particular label, ℓ, while utilizing q for the prevention of double-spending.
- **Proof of burn.** U is required to provide evidence to CUS, indicating that they have successfully conducted a burn operation amounting to v for a specific label, ℓ. Upon receipt of CUS's instruction in the form of ($\mathsf{PoB.CUS}, \mathsf{sid}, \eta, \ell$), $\mathcal{F}_{\mathsf{PARS}}$ proceeds to inform CUS about the user's identity U and the burned value, v, if U has already submitted a message to $\mathcal{F}_{\mathsf{PARS}}$ that confirms both η and ℓ.
- **Reserve audit.** As explained above each unique $\tilde{\chi}$ value has been recorded, along with the corresponding total stablecoin supply TV at the time of $\tilde{\chi}$ submission. AUD supplies both $\tilde{\chi}$ and RV to $\mathcal{F}_{\mathsf{PARS}}$ where RV is the total value of reserves held in custody. Consequently, $\mathcal{F}_{\mathsf{PARS}}$ retrieves the corresponding TV value associated with the provided $\tilde{\chi}$ and verifies whether the condition $\mathsf{RV} \geq \mathsf{TV}$ is satisfied. Here, RV represents the reserves held by CUS, provided by $\mathcal{Z}$.[5]

[5] In a real-world implementation, it is imperative for AUD to diligently ensure the validity of RV through their due diligence efforts.

Functionality $\mathcal{F}_{\mathsf{PARS}}$ – Part II

Stablecoin burn.

- Upon receiving a message $(\mathtt{Brn}, \mathsf{sid}, v, \ell)$ from some party U: (i) Ignore if $\mathbb{R}(\mathsf{U}) = \bot$ or $v < 0$. (ii) Else, generate a new $\mathsf{SB.idn}$ and set $\mathbb{B}(\mathsf{SB.idn}) \leftarrow (\mathsf{U}, v, \ell, \mathsf{B})$. (iii) Output $(\mathtt{Brn}, \mathsf{sid}, \ell, \mathsf{SB.idn})$ to $\mathcal{A}$.
- Upon receiving $(\mathtt{Brn.Ok}, \mathsf{sid}, \mathsf{SB.idn}, \eta, \tilde{\chi})$ from $\mathcal{A}$: (i) Ignore if $\mathbb{B}(\mathsf{SB.idn}) = \bot$ or there exists $(\eta, \cdot, \cdot, \cdot, \cdot)$ or there exists $(\tilde{\chi}, \cdot)$. (ii) Else, retrieve $\mathbb{B}(\mathsf{SB.idn}) = (\mathsf{U}, v, \ell, \mathsf{B})$, and $\mathbb{R}(\mathsf{U}) = \mathsf{B}$. (iii) Ignore if $\mathsf{B} < v$. (iv) Else, set $\mathbb{R}(\mathsf{U}) \leftarrow (\mathsf{B} - v)$, $\mathsf{TV} \leftarrow (\mathsf{TV} - v)$, and $\mathbb{B}(\mathsf{SB.idn}) \leftarrow \bot$. (v) Record $(\eta, \mathsf{U}, v, \ell, q)$ where $q = 0$, and $(\tilde{\chi}, \mathsf{TV})$. (vi) Output $(\mathtt{Brn.End}, \mathsf{sid}, \eta, \ell, \tilde{\chi})$ to $\mathcal{I}$ via public-delayed outputs.

Proof of burn.

- Upon receiving a message $(\mathtt{PoB}, \mathsf{sid}, \eta, \ell)$ from some user U: (i) Ignore if $\mathbb{R}(\mathsf{U}) = \bot$ or there does not exist $(\eta, \mathsf{U}, \cdot, \ell, \cdot)$ or there exists $(\eta, \mathsf{U}, \cdot, \ell, 1)$. (ii) Else, retrieve $(\eta, \mathsf{U}, v, \ell, 0)$, generate a new $\mathsf{PB.idn}$ and set $\mathbb{V}(\mathsf{PB.idn}) \leftarrow (\eta, \mathsf{U}, v, \ell, 0)$. (iii) Output $(\mathtt{PoB}, \mathsf{sid}, \ell, \mathsf{PB.idn})$ to $\mathcal{A}$.
- Upon receiving $(\mathtt{PoB.Ok}, \mathsf{sid}, \mathsf{PB.idn})$ from $\mathcal{A}$: (i) Ignore if $\mathbb{V}(\mathsf{PB.idn}) = \bot$. (ii) Else, retrieve $\mathbb{V}(\mathsf{PB.idn}) = (\eta, \mathsf{U}, v, \ell, q)$. Upon receiving $(\mathtt{PoB.CUS}, \mathsf{sid}, \eta, \ell)$ from CUS: (i) Output $(\mathtt{PoB.CUS}, \mathsf{sid}, \mathsf{PB.idn})$ to $\mathcal{A}$. Upon receiving $(\mathtt{PoB.CUS.Ok}, \mathsf{sid}, \mathsf{PB.idn})$ from $\mathcal{A}$: (i) Retrieve $\mathbb{V}(\mathsf{PB.idn}) = (\eta, \mathsf{U}, v, \ell, q)$ and ignore if $q = 1$. (ii) Else, output $(\mathtt{PoB.End}, \mathsf{sid}, \eta, \mathsf{U}, v, \ell)$ to CUS via private-delayed output, and set $q \leftarrow 1$ in $(\eta, \mathsf{U}, v, \ell, q)$.

Reserve audit.

- Upon receiving a message $(\mathtt{Audit}, \mathsf{sid}, \tilde{\chi}, \mathsf{RV})$ from AUD: (i) Ignore if there does not exist $(\tilde{\chi}, \cdot)$. (ii) Else, retrieve $(\tilde{\chi}, \mathsf{TV})$, and if $\mathsf{RV} \geq \mathsf{TV}$, set $b \leftarrow 1$. (iii) Else, set $b \leftarrow 0$. (iv) Output $(\mathtt{Audit.End}, \mathsf{sid}, b)$ to AUD via public delayed output.

3 Our construction PARScoin

Our construction Π_{PARS} employs various cryptographic schemes, including ElGamal encryption scheme (Definition 6), non-threshold randomizable-blind signature (Definition 15), threshold randomizable-blind signature (Definition 16), and Pedersen commitment (Definition 11). Additionally, it uses the following ideal sub-functionalities: key registration functionality $\mathcal{F}_{\mathsf{KeyReg}}$ (Appendix B.1), random oracle functionality $\mathcal{F}_{\mathsf{RO}}$ (Appendix B.2), communication channel functionality $\mathcal{F}_{\mathsf{Ch}}$ (Appendix B.3), broadcast functionality $\mathcal{F}_{\mathsf{B}}$ (Appendix B.4) and non-interactive zero-knowledge functionality $\mathcal{F}_{\mathsf{NIZK}}$ (Appendix B.5). In the following, verifiers, maintainers, and issuers are interchangeably used.

3.1 High-level technical overview

In Π_{PARS}, inspired by PEReDi [35], users enjoy self-sovereignty over their accounts, with the account state being controlled by the user's secret key. User accounts are of the form: $\mathsf{acc} = (\mathsf{B}, \mathsf{sk}, \mathsf{sk}^x, \mathsf{sk}') \in \mathbb{Z}_p^4$ wherein the user balance B and transaction counter x are updated in each transaction. The selection of sk^x is motivated by the necessity of incorporating a transaction counter x for preventing double-spending. This counter assures system's verifiers that users are not utilizing their previous account. However, employing just x is insufficient, as we aim to hide this value[6] while demonstrating, through (efficient, Sigma protocol-based) non-interactive zero knowledge (NIZK) proof π, that in the new account, x has been precisely incremented by 1. Users prove that the third element in their account (sk^x) is accurately updated (sk^{x+1}) without revealing sk and x. Simultaneously, the user discloses a deterministically defined double-spending prevention tag, a function of sk^x. Double-spending is prevented by forcing users to disclose $\mathsf{tag} = g^{(\mathsf{sk}^{x+1})}$ associated with the new account. The disclosed tags are retained by issuers and ensure that only the updated account can be used in the future. This tag aids verifiers in recognizing a valid account update as for each account state update the user has to increment x by 1 (given the current state of the account tag is uniquely defined).

The balance $\mathsf{B} \in \mathsf{acc}$ of the user is updated by themselves (so it is privacy-preserving). As an example, once $\mathsf{U_r}$ receives value v, she updates her account as follows:

$$\mathsf{acc}_\mathsf{r}^{\mathsf{old}} = (\mathsf{B}_\mathsf{r}^{\mathsf{old}}, \mathsf{sk}_\mathsf{r}, \mathsf{sk}_\mathsf{r}^x, \mathsf{sk}_\mathsf{r}') \dashrightarrow \mathsf{acc}_\mathsf{r}^{\mathsf{new}} = (\mathsf{B}_\mathsf{r}^{\mathsf{old}} + v, \mathsf{sk}_\mathsf{r}, \mathsf{sk}_\mathsf{r}^{x+1}, \mathsf{sk}_\mathsf{r}')$$

Each transaction updates the account state. The user provides a NIZK proof π ensuring the integrity of account update by themselves, and well-formedness of tag.

Π_{PARS} achieves four aspects of privacy outlined in Section 2.2 for all transactions between an honest sender and honest receiver. This entails users maintaining the confidentiality of their account information vis-à-vis all network participants (including issuers who are verifying transactions), while efficiently proving via NIZK π, the accuracy of account updates. Through π, users prove the possession of a validly signed account by the issuers $\sigma_{\mathbb{I}}$ (which is a threshold signature of issuers on $\mathsf{acc}^{\mathsf{old}}$), affirming that the proposed account update $\mathsf{acc}^{\mathsf{new}}$ is based on their previously approved account $\mathsf{acc}^{\mathsf{old}}$ while keeping both $\mathsf{acc}^{\mathsf{old}}$, and $\mathsf{acc}^{\mathsf{new}}$ hidden by blinding them. Issuers verify π, which proves that the new blinded account is consistent with the signature of issuers on the old blinded account. If π is verified, issuers sign new blinded account and record the double-spending tag, advancing the user's account state by one step so that the user is ready for the next transaction. Additionally, to achieve efficient unlinkability[7] the user

[6] As x reveals the number of times the user has been a counterparty in a transaction either as a sender or receiver.

[7] In principle, one could prove the full signature in zero-knowledge also achieving unlinkability.

employs signature randomization $\sigma_{\mathbb{I}}^{\mathsf{RND}}$. In addition to transactions among users, Π_{PARS} offers privacy in the *stablecoin issuance*, and *stablecoin burn* protocols, thereby preventing any malicious issuer from learning the identity of the user or the quantity of stablecoin being issued/burned[8].

We emphasize that the system's public parameters are integrated into NIZK statements. For simplicity, we have not explicitly addressed these parameters in statements.

3.2 Details of the construction

In the subsequent protocols, whenever there is a need for user-issuer communication, U engages in such interactions by leveraging the threshold randomizable-blind signature protocol. In order to enhance simplicity within the UC framework, we let $\mathcal{Z}$ provide instructions to entities carrying relevant values. Consequently, we have refrained from addressing a publicly accessible blockchain ledger in our formal modelling. When an environment $\mathcal{Z}$ sends a message to an honest party, if that party is already in a state transition, they will ignore the message.

3.2.1 Initialization. The following public keys are registered calling $\mathcal{F}_{\mathsf{KeyReg}}$.

- **The auditor (AUD):** run KeyGen algorithm of ElGamal encryption and get $(\mathsf{pk}_{\mathsf{A}}, \mathsf{sk}_{\mathsf{A}})$ as output[9].
- **Each issuer (I_i):** engage in the distributed key generation protocol TRB.Sig.KeyGen of threshold randomizable-blind signature and get $\mathsf{sk}_i = (x_i, \{y_{i,\kappa}\}_{\kappa=1}^4)$, and $\mathsf{pk}_i = (\tilde{\alpha}_i, \{\beta_{i,\kappa}, \tilde{\beta}_{i,\kappa}\}_{\kappa=1}^4) = (\tilde{g}^{x_i}, \{g^{y_{i,\kappa}}, \tilde{g}^{y_{i,\kappa}}\}_{\kappa=1}^4)$ as outputs. Signature's public key $\mathsf{Sig.pk} = (\mathsf{par}, \tilde{\alpha}, \{\beta_\kappa, \tilde{\beta}_\kappa\}_{\kappa=1}^4)$ is publicly announced where $\mathsf{par} = (p, \mathbb{G}, \tilde{\mathbb{G}}, \mathbb{G}_t, e, g, \tilde{g}, \{h_\kappa\}_{\kappa=1}^4, g_1, W)$; and $(g_1, W) \in \mathbb{G}$ with unknown discrete logarithm base g.
- **The custodian (CUS):** run RB.Sig.KeyGen algorithm of non-threshold randomizable-blind signature and get $\bar{\mathsf{par}} = (\bar{p}, \bar{\mathbb{G}}, \bar{\tilde{\mathbb{G}}}, \bar{\mathbb{G}}_t, \bar{e}, \bar{g}, \bar{\tilde{g}}, \{\bar{h}_\kappa\}_{\kappa=1}^4)$, $\bar{\mathsf{sk}} = (\bar{x}, \{\bar{y}_\kappa\}_{\kappa=1}^4) \xleftarrow{\$} \mathbb{Z}_p$, and $\bar{\mathsf{pk}} = (\bar{\mathsf{par}}, \bar{\tilde{\alpha}}, \{\bar{\beta}_\kappa, \bar{\tilde{\beta}}_\kappa\}_{\kappa=1}^4) = (\bar{\mathsf{par}}, \bar{\tilde{g}}^x, \{\bar{g}^{\bar{y}_\kappa}, \bar{\tilde{g}}^{\bar{y}_\kappa}\}_{\kappa=1}^4)$ as outputs.
- **The user (U):** choose $\mathsf{sk} \xleftarrow{\$} \mathbb{Z}_p^*$, and $\mathsf{sk} \xleftarrow{\$} \mathbb{Z}_p^*$. Compute $\mathsf{pk} = g^{\mathsf{sk}'}$.

[8] This level of privacy can persist even in scenarios where (malicious) CUS collaborates with malicious issuers. CUS engages with U in a coin flipping protocol to generate sn. Hence issuers will be incapable of establishing any linkage between U's interactions with them and fiat currency deposits via U on CUS's side. It is worth mentioning that the system will be vulnerable to front-running attack by a malicious U if the protocol lets U choose sn as sn is revealed to issuers.

[9] We note that it is possible to "thresholdize" in a straightforward way the key generation for the auditor if so desired.

3.2.2 User registration. Issuers undertake user enrollment process within the system by generating a $\mathsf{TRB.Sig}$ signature for the user's initial account. Subsequently, U utilizes this signature for conducting transactions. Users obtain the signature of issuers for their initial account which is $\mathsf{acc} = (0, \mathsf{sk}, 1, \mathsf{sk}')$ where balance B and transaction counter x are 0. However, users are unwilling to disclose their secret keys to issuers, so that they blind their initial accounts. Subsequently, the user proves, through a NIZK proof π, the validity of the blinded information (e.g., $\mathsf{B} = x = 0$). π proves knowledge of secret keys and proves that the secret key associated with the registered public key $\mathtt{pk}$ (in $\mathcal{F}_{\mathsf{KeyReg}}$) has indeed been blinded. The user U initiates the *user registration* protocol with issuers upon receiving $(\mathtt{Rgs}, \mathsf{sid})$ from $\mathcal{Z}$ as follows:

- ▷ Parse $\mathsf{acc} = (0, \mathsf{sk}, 1, \mathsf{sk}')$.
- ▷ Pick $o \xleftarrow{\$} \mathbb{Z}_p$ and compute $\mathtt{com} = g^o \cdot h_1^0 \cdot h_2^{\mathsf{sk}} \cdot h_3^1 \cdot h_4^{\mathsf{sk}'}$.
- ▷ Call $\mathcal{F}_{\mathrm{RO}}$ with $(\mathtt{Query}, \mathsf{sid}, \mathtt{com})$ and receive $(\mathtt{Query.Re}, \mathsf{sid}, h)$.
- ▷ Commit to each acc element: pick $(o_1, o_2, o_3, o_4) \xleftarrow{\$} \mathbb{Z}_p$ and compute $\mathtt{com}_1 = g^{o_1} \cdot h^0$, $\mathtt{com}_2 = g^{o_2} \cdot h^{\mathsf{sk}}$, $\mathtt{com}_3 = g^{o_3} \cdot h^1$, and $\mathtt{com}_4 = g^{o_4} \cdot h^{\mathsf{sk}'}$.
- ▷ Call $\mathcal{F}_{\mathsf{NIZK}}$ with $(\mathtt{Prove}, \mathsf{sid}, \mathsf{x}, \mathsf{w})$ for the relation:
  - $\mathcal{R}(\mathsf{x}, \mathsf{w}) = \mathsf{NIZK}\{\mathtt{com} = g^o \cdot h_2^{\mathsf{sk}} \cdot h_3 \cdot h_4^{\mathsf{sk}'} \wedge \mathtt{com}_1 = g^{o_1} \wedge \mathtt{com}_2 = g^{o_2} \cdot h^{\mathsf{sk}} \wedge \mathtt{com}_3 = g^{o_3} \cdot h \wedge \mathtt{com}_4 = g^{o_4} \cdot h^{\mathsf{sk}'} \wedge \mathtt{pk} = g^{\mathsf{sk}'}\}$,
  - $\mathsf{x} = (\mathtt{com}, \mathtt{com}_1, \mathtt{com}_2, \mathtt{com}_3, \mathtt{com}_4, h, \mathtt{pk})$,
  - $\mathsf{w} = (\mathsf{sk}, \mathsf{sk}', o, o_1, o_2, o_3, o_4)$.
- ▷ Upon receiving $(\mathtt{Proof}, \mathsf{sid}, \pi)$ from $\mathcal{F}_{\mathsf{NIZK}}$, set $\mathfrak{R} \leftarrow (\mathsf{x}, \pi)$.
- ▷ Call $\mathcal{F}_{\mathsf{B}}^{\mathsf{S}}$ with $(\mathtt{Broadcast}, \mathsf{sid}, \mathfrak{R})$.

Fig. 2: User registration – U

Each issuer checks whether the user has been previously registered. In case U is new and NIZK proof π is valid, the issuer proceeds to record the user's identifier in its local database $\mathcal{D}_i$ as the registered user. Note that $\mathcal{D}_i$ is confidential to the i-th issuer and can be updated only by the i-th issuer. Subsequently, the issuer provides a blind signature $\sigma_i^{\mathsf{blind}}$ for the user's initial blinded account and transmits the blind signature share back to U. The i-th issuer I_i upon receiving $(\mathtt{Broadcasted}, \mathsf{sid}, \mathsf{U}, \mathfrak{R})$ from $\mathcal{F}_{\mathsf{B}}^{\mathsf{S}}$ acts as follows.

- ▷ Ignore $\mathfrak{R}$ if $(\mathsf{U}, \cdot) \in \mathcal{D}_i$. Else, parse $\mathfrak{R} = (\mathsf{x}, \pi)$, and ignore if upon calling $\mathcal{F}_{\mathsf{NIZK}}$ with $(\mathtt{Verify}, \mathsf{sid}, \mathsf{x}, \pi)$, $(\mathtt{Vrfed}, \mathsf{sid}, 0)$ is received.
- ▷ Else, parse $\mathsf{x} = (\mathtt{com}, \mathtt{com}_1, \mathtt{com}_2, \mathtt{com}_3, \mathtt{com}_4, h, \mathtt{pk})$, and save $(\mathsf{U}, \mathtt{pk})$ in $\mathcal{D}_i$.
- ▷ Compute blind signature share $\sigma_i^{\mathsf{blind}}$:
  - Call $\mathcal{F}_{\mathrm{RO}}$ with $(\mathtt{Query}, \mathsf{sid}, \mathtt{com})$, and receive $(\mathtt{Query.Re}, \mathsf{sid}, h')$. Ignore if $h \neq h'$,
  - Else, compute $c_i = h^{x_i} \cdot \mathtt{com}_1^{y_{i,1}} \cdot \mathtt{com}_2^{y_{i,2}} \cdot \mathtt{com}_3^{y_{i,3}} \cdot \mathtt{com}_4^{y_{i,4}}$ and set the blind signature share $\sigma_i^{\mathsf{blind}} = (h, c_i)$.

▷ Call $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{ISAS}}$ with $(\mathtt{Send}, \mathsf{sid}, \mathsf{U}, \sigma_i^{\mathsf{blind}})$.

Fig. 3: User registration – I_i

U performs the unblinding process on the received blind signature share $\sigma_i^{\mathsf{blind}}$ to get σ_i. U disregards it if found to be invalid. Upon accumulating a sufficient number (Γ) of valid signature shares from several issuers, U combines them to generate a single aggregated signature $\sigma_{\mathbb{I}}$ (so that once the user would like to prove the validity of its account instead of providing several signatures it provides one aggregated signature which results in better efficiency).
U upon receiving $(\mathtt{Received}, \mathsf{sid}, \mathsf{I}_i, \sigma_i^{\mathsf{blind}})$ from $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{ISAS}}$ unblinds the received signature share $\sigma_i^{\mathsf{blind}}$ to obtain σ_i. Computes aggregated signature $\sigma_{\mathbb{I}}$ as follows.

▷ Unblind the received signature share $\sigma_i^{\mathsf{blind}}$:
- Parse $\sigma_i^{\mathsf{blind}} = (h', c_i)$. Ignore if $h \neq h'$.
- Else, compute $\sigma_i = (h, s_i) = (h, c_i \cdot \beta_{i,1}^{-o_1} \cdot \beta_{i,2}^{-o_2} \cdot \beta_{i,3}^{-o_3} \cdot \beta_{i,4}^{-o_4})$.
- Ignore if $e(h, \tilde{\alpha}_i \cdot \tilde{\beta}_{i,1}^{0} \cdot \tilde{\beta}_{i,2}^{\mathsf{sk}} \cdot \tilde{\beta}_{i,3}^{1} \cdot \tilde{\beta}_{i,4}^{\mathsf{sk}'}) = e(s_i, \tilde{g})$ does not hold. Else, proceed as follows.

▷ Compute aggregated signature $\sigma_{\mathbb{I}}$:
- Define $S \in [1, N]$ as a set of Γ indices of issuers in the set $\mathbb{I}$. For all $i \in S$, evaluate the Lagrange basis polynomials at 0: $l_i = [\prod_{j \in S, j \neq i}(j)/(j-i)]$.
- For all $i \in S$, take $\sigma_i = (h, s_i)$ and compute the signature $\sigma_{\mathbb{I}} = (h, s) = (h, \prod_{i \in S} s_i^{l_i})$.
- Ignore if $e(h, \tilde{\alpha}) = e(s, \tilde{g})$ does not hold.

Fig. 4: User registration – U

3.2.3 Stablecoin issuance. After obtaining the unique identifier of the user U and the value v from the environment $\mathcal{Z}$, the custodian CUS randomly chooses a serial number sn. This serial number serves the purpose of preventing the occurrence of double-spending, and it is subsequently transmitted to U. Upon receiving $(\mathtt{Iss}, \mathsf{sid}, \mathsf{U}, v)$ from $\mathcal{Z}$, CUS picks $\mathsf{sn} \xleftarrow{\$} \mathbb{Z}_p$ and calls $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{SRA}}$ with $(\mathtt{Send}, \mathsf{sid}, \mathsf{U}, (\mathsf{sn}, v))$.
The user U is required to obtain CUS's signature confirming the value v for the stablecoin that is to be issued. However, due to privacy requirements concerning both v and identity, U cannot simply procure CUS's signature on its identity and v. U needs to present the signature of CUS to the issuers in order to obtain the stablecoins, and it is imperative to conceal both the identity of the user and the value of the stablecoin from all issuers. We use blind version of Pointcheval-Sanders signature as the underlying signature between U and CUS. U binds the

received serial number $\mathtt{sn}$, its secret keys $(\mathsf{sk}, \mathsf{sk}')$, and v to each other to get the signature of CUS.
The user demonstrates in ZK that e.g., sk' is the associated secret key of U's registered public key, and the statement of NIZK reveals v, and $\mathtt{sn}$ so that CUS makes sure that those are correct. Note that while v, and $\mathtt{sn}$ are revealed to CUS, CUS remains blind to $(\mathsf{sk}, \mathsf{sk}')$. U acts as follows upon receiving $(\mathtt{Received}, \mathsf{sid}, \mathsf{CUS}, (\mathsf{sn}, v))$ from $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{SRA}}$:

▷ Pick $\bar{o} \xleftarrow{\$} \mathbb{Z}_p$ and compute $\bar{\mathtt{com}} = \bar{g}^{\bar{o}} \cdot \bar{h}_1^{\mathsf{sn}} \cdot \bar{h}_2^{\mathsf{sk}} \cdot \bar{h}_3^{v} \cdot \bar{h}_4^{\mathsf{sk}'}$.
▷ Submit $(\mathtt{Query}, \mathsf{sid}, \bar{\mathtt{com}})$ to $\mathcal{F}_{\mathrm{RO}}$ and receive $(\mathtt{Query.Re}, \mathsf{sid}, \bar{h})$.
▷ Pick $(\bar{o}_1, \bar{o}_2, \bar{o}_3, \bar{o}_4) \xleftarrow{\$} \mathbb{Z}_p$ and compute $\bar{\mathtt{com}}_1 = \bar{g}^{\bar{o}_1} \cdot \bar{h}^{\mathsf{sn}}$, $\bar{\mathtt{com}}_2 = \bar{g}^{\bar{o}_2} \cdot \bar{h}^{\mathsf{sk}}$, $\bar{\mathtt{com}}_3 = \bar{g}^{\bar{o}_3} \cdot \bar{h}^{v}$, and $\bar{\mathtt{com}}_4 = \bar{g}^{\bar{o}_4} \cdot \bar{h}^{\mathsf{sk}'}$.
▷ Call $\mathcal{F}_{\mathsf{NIZK}}$ with input $(\mathtt{Prove}, \mathsf{sid}, \mathsf{x}, \mathsf{w})$, for the relation:
- $\mathcal{R}(\mathsf{x}, \mathsf{w}) = \mathsf{NIZK}\{\bar{\mathtt{com}} = \bar{g}^{\bar{o}} \cdot \bar{h}_1^{\mathsf{sn}} \cdot \bar{h}_2^{\mathsf{sk}} \cdot \bar{h}_3^{v} \cdot \bar{h}_4^{\mathsf{sk}'} \wedge \bar{\mathtt{com}}_1 = \bar{g}^{\bar{o}_1} \cdot \bar{h}^{\mathsf{sn}} \wedge \bar{\mathtt{com}}_2 = \bar{g}^{\bar{o}_2} \cdot \bar{h}^{\mathsf{sk}} \wedge \bar{\mathtt{com}}_3 = \bar{g}^{\bar{o}_3} \cdot \bar{h}^{v} \wedge \bar{\mathtt{com}}_4 = \bar{g}^{\bar{o}_4} \cdot \bar{h}^{\mathsf{sk}'} \wedge \mathtt{pk} = g^{\mathsf{sk}'}\}$
- $\mathsf{x} = (\bar{\mathtt{com}}, \bar{\mathtt{com}}_1, \bar{\mathtt{com}}_2, \bar{\mathtt{com}}_3, \bar{\mathtt{com}}_4, \bar{h}, \mathtt{pk}, v, \mathsf{sn})$
- $\mathsf{w} = (\mathsf{sn}, \mathsf{sk}, \mathsf{sk}', \bar{o}, \bar{o}_1, \bar{o}_2, \bar{o}_3, \bar{o}_4)$

▷ Upon receiving $(\mathtt{Proof}, \mathsf{sid}, \pi)$ from $\mathcal{F}_{\mathsf{NIZK}}$, set $\bar{\mathfrak{I}} \leftarrow (\mathsf{x}, \pi)$.
▷ Call $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{SSA}}$ with $(\mathtt{Send}, \mathsf{sid}, \mathsf{CUS}, \bar{\mathfrak{I}})$.

Fig. 5: Stablecoin issuance – U

CUS signs the associated cryptographic object (encompassing the user's secret keys $(\mathsf{sk}, \mathsf{sk}')$, serial number $\mathtt{sn}$, and stablecoin value v) to generate $\sigma_{\mathsf{CUS}}^{\mathsf{blind}}$; and transmits $\sigma_{\mathsf{CUS}}^{\mathsf{blind}}$ back to U, so that U can acquire their stablecoins through the collaboration of distributed issuers by using CUS's signature as we will see. CUS acts as follows upon receiving $(\mathtt{Received}, \mathsf{sid}, \mathsf{U}, \bar{\mathfrak{I}})$ from $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{SSA}}$:

▷ Parse $\bar{\mathfrak{I}} = (\mathsf{x}, \pi)$, and $\mathsf{x} = (\bar{\mathtt{com}}, \bar{\mathtt{com}}_1, \bar{\mathtt{com}}_2, \bar{\mathtt{com}}_3, \bar{\mathtt{com}}_4, \bar{h}, \mathtt{pk}, v', \mathsf{sn}')$.
▷ Ignore $\bar{\mathfrak{I}}$ if at least one of the following conditions holds:
- $v' \neq v$ or $\mathsf{sn}' \neq \mathsf{sn}$
- Upon calling $\mathcal{F}_{\mathsf{NIZK}}$ with $(\mathtt{Verify}, \mathsf{sid}, \mathsf{x}, \pi)$, $(\mathtt{Vrfed}, \mathsf{sid}, 0)$ is received.
- Upon calling $\mathcal{F}_{\mathsf{KeyReg}}$ with $(\mathtt{Key.Retrieval}, \mathsf{sid}, \mathsf{U})$, $(\mathtt{Key.Retrieved}, \mathsf{sid}, \mathsf{U}, \mathtt{pk}')$ is received where $\mathtt{pk}' \neq \mathtt{pk}$.
- Call $\mathcal{F}_{\mathrm{RO}}$ with $(\mathtt{Query}, \mathsf{sid}, \bar{\mathtt{com}})$, and $(\mathtt{Query.Re}, \mathsf{sid}, \bar{h}')$ is received where $\bar{h} \neq \bar{h}'$.

▷ Upon receiving a message $(\mathtt{Iss}, \mathsf{sid}, \mathsf{U}', v')$ from $\mathcal{Z}$, proceed if $\mathsf{U}' = \mathsf{U}$ and $v' = v$.
▷ Compute $\bar{c} = \bar{h}^{x} \cdot \bar{\mathtt{com}}_1^{y_1} \cdot \bar{\mathtt{com}}_2^{y_2} \cdot \bar{\mathtt{com}}_3^{y_3} \cdot \bar{\mathtt{com}}_4^{y_4}$, set the blind signature $\sigma_{\mathsf{CUS}}^{\mathsf{blind}} = (\bar{h}, \bar{c})$.
▷ Call $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{IRAS}}$ with $(\mathtt{Send}, \mathsf{sid}, \mathsf{U}, \sigma_{\mathsf{CUS}}^{\mathsf{blind}})$.

Fig. 6: Stablecoin issuance – CUS

U first of all unblinds $\sigma^{\mathsf{blind}}_{\mathsf{CUS}}$, to get σ_{CUS}. Then, for preventing possible linkage randomizes the signature $\sigma^{\mathsf{RND}}_{\mathsf{CUS}}$ (note that the message of the signature which is $(\mathsf{sn}, \mathsf{sk}, v, \mathsf{sk}')$ is kept secret). Note that we use blind Pointcheval-Sanders signature between U and CUS, and threshold blind Pointcheval-Sanders signature between U and issuers. U should update their account in a way that it is consistent with CUS's signature ($\sigma^{\mathsf{RND}}_{\mathsf{CUS}}$). Consequently, U sets $\mathsf{B}^{\mathsf{new}} = \mathsf{B}^{\mathsf{old}} + v$ and increments the transaction counter by 1 (sk^{x+1}). U blinds $\mathsf{acc}^{\mathsf{new}}$ and proves in ZK that the new blinded account is consistent with $\sigma^{\mathsf{RND}}_{\mathsf{CUS}}$ and with old blinded account for which the user has signature of issuers $\sigma_{\mathbb{I}}$.
$\sigma_{\mathbb{I}}$ represents the (threshold) signature of issuers on $\mathsf{acc}^{\mathsf{old}}$, and it undergoes randomization by U to yield $\sigma^{\mathsf{RND}}_{\mathbb{I}}$ (that is verified without revealing the message of the signature itself $\mathsf{acc}^{\mathsf{old}}$). The purpose of randomization is to preclude any potential linkage by a malicious issuer between the moment of signing $\mathsf{acc}^{\mathsf{old}}$ and the subsequent submission of that signature for account update.
U calculates the double-spending tag $g^{(\mathsf{sk}^{x+1})}$, a value deterministically derived from the user's account. It's important to highlight that sn is specifically utilized to prevent double-spending on the custodian's message, while tag serves the purpose of averting double-spending regarding the user's account.
Regarding privacy-enhanced auditing, for stablecoin issuance (and as we will see for stablecoin burn), users encrypt v under the public key pk_{A} of the auditor AUD, generating $\tilde{\chi}_t$. Note that users (efficiently) prove the consistency of v in $\tilde{\chi}_t$ with their account update via π. Notably, we use the homomorphic property of ElGamal encryption in pivotal role here: the i-th issuer multiplies the newly generated ciphertext by the user, the provided $\tilde{\chi}_t$, with the previously-stored ciphertext $\tilde{\chi}_i$ by themselves in their local database $\mathcal{D}_i$. This operation is executed without the issuer knowing v. The result of this operation represents an updated ciphertext, signifying the adjusted total value of stablecoins within the system. Ultimately, this updated ciphertext is then made available within the system (it is output to $\mathcal{Z}$). Hence, $\tilde{\chi}_t$ facilitates private issuance (and burn) by keeping value hidden while providing auditability (note that underlying broadcast functionality guarantees that honest issuers always have the same view). Whenever AUD wants, they can decrypt the ciphertext, thereby starting the auditing protocol in collaboration with CUS as we are about to see. U acts as follows upon receiving $(\texttt{Received}, \mathsf{sid}, \mathsf{CUS}, \sigma^{\mathsf{blind}}_{\mathsf{CUS}})$ from $\mathcal{F}^{\mathsf{IRAS}}_{\mathsf{Ch}}$:

▷ Unblind the received signature:
- Parse $\sigma^{\mathsf{blind}}_{\mathsf{CUS}} = (\bar{h}', \bar{c})$. Ignore if $\bar{h} \neq \bar{h}'$.
- Else, compute $\sigma_{\mathsf{CUS}} = (\bar{h}, \bar{s}) = (\bar{h}, \bar{c} \cdot \bar{\beta}_1^{-\bar{o}_1} \cdot \bar{\beta}_2^{-\bar{o}_2} \cdot \bar{\beta}_3^{-\bar{o}_3} \cdot \bar{\beta}_4^{-\bar{o}_4})$.
- Ignore if $e(\bar{h}, \bar{\tilde{\alpha}} \cdot \bar{\tilde{\beta}}_1^{\mathsf{sn}} \cdot \bar{\tilde{\beta}}_2^{\mathsf{sk}} \cdot \bar{\tilde{\beta}}_3^{v} \cdot \bar{\tilde{\beta}}_4^{\mathsf{sk}'}) = e(\bar{s}, \bar{\tilde{g}})$ does not hold.

▷ Parse $\sigma_{\mathsf{CUS}} = (\bar{h}, \bar{s})$, and pick $(\bar{r}, \bar{r}') \xleftarrow{\$} \mathbb{Z}_p$.
▷ Compute $\sigma^{\mathsf{RND}}_{\mathsf{CUS}} = (\bar{h}', \bar{s}') = (\bar{h}^{\bar{r}'}, \bar{s}^{\bar{r}'} \cdot \bar{h}^{\bar{r}'\bar{r}})$, and $\bar{\varkappa} = \bar{\tilde{\alpha}} \cdot \bar{\tilde{\beta}}_1^{\mathsf{sn}} \cdot \bar{\tilde{\beta}}_2^{\mathsf{sk}} \cdot \bar{\tilde{\beta}}_3^{v} \cdot \bar{\tilde{\beta}}_4^{\mathsf{sk}'} \cdot \bar{\tilde{g}}^{\bar{r}}$.
▷ Parse $\mathsf{acc}^{\mathsf{old}} = (\mathsf{B}^{\mathsf{old}}, \mathsf{sk}, \mathsf{sk}^{x}, \mathsf{sk}')$.
▷ Compute $\mathsf{acc}^{\mathsf{new}} = (\mathsf{B}^{\mathsf{old}} + v, \mathsf{sk}, (\mathsf{sk}^{x+1}), \mathsf{sk}')$.

▷ Pick $o \xleftarrow{\$} \mathbb{Z}_p$ and compute $\mathtt{com} = g^o \cdot h_1^{\mathsf{B}^{\mathsf{old}}+v} \cdot h_2^{\mathsf{sk}} \cdot h_3^{(\mathsf{sk}^{x+1})} \cdot h_4^{\mathsf{sk}'}$.
▷ Submit $(\mathtt{Query}, \mathsf{sid}, \mathtt{com})$ to $\mathcal{F}_{\mathrm{RO}}$, and receive $(\mathtt{Query.Re}, \mathsf{sid}, h)$.
▷ Pick $(o_1, o_2, o_3, o_4) \xleftarrow{\$} \mathbb{Z}_p$ and compute $\mathtt{com}_1 = g^{o_1} \cdot h^{\mathsf{B}^{\mathsf{old}}+v}$, $\mathtt{com}_2 = g^{o_2} \cdot h^{\mathsf{sk}}$, $\mathtt{com}_3 = g^{o_3} \cdot h^{(\mathsf{sk}^{x+1})}$, and $\mathtt{com}_4 = g^{o_4} \cdot h^{\mathsf{sk}'}$
▷ Parse $\sigma_{\mathbb{I}} = (h, s)$, and pick $(r, r') \xleftarrow{\$} \mathbb{Z}_p$.
▷ Compute $\sigma_{\mathbb{I}}^{\mathsf{RND}} = (h', s') = (h^{r'}, s^{r'} \cdot h^{rr'})$.
▷ Compute $\varkappa = \tilde{\alpha} \cdot \tilde{\beta}_1^{\mathsf{B}^{\mathsf{old}}} \cdot \tilde{\beta}_2^{\mathsf{sk}} \cdot \tilde{\beta}_3^{\mathsf{sk}^x} \cdot \tilde{\beta}_4^{\mathsf{sk}'} \cdot \tilde{g}^r$, and $\mathsf{tag} = g^{(\mathsf{sk}^{x+1})}$.
▷ Pick $z \xleftarrow{\$} \mathbb{Z}_p$ and set $\tilde{\chi}_t = (\tilde{C}_1, \tilde{C}_2) = (g^z, \mathtt{pk}_{\mathsf{A}}^z \cdot g_1^v)$.
▷ Call $\mathcal{F}_{\mathsf{NIZK}}$ with input $(\mathtt{Prove}, \mathsf{sid}, \mathsf{x}, \mathsf{w})$, for the relation:
- $\mathcal{R}(\mathsf{x}, \mathsf{w}) = \mathsf{NIZK}\{\bar{\varkappa} = \bar{\tilde{\alpha}} \cdot \bar{\tilde{\beta}}_1^{\mathsf{sn}} \cdot \bar{\tilde{\beta}}_2^{\mathsf{sk}} \cdot \bar{\tilde{\beta}}_3^{v} \cdot \bar{\tilde{\beta}}_4^{\mathsf{sk}'} \cdot \bar{\tilde{g}}^{\bar{r}} \wedge \mathtt{com} = g^o \cdot h_1^{\mathsf{B}^{\mathsf{old}}+v} \cdot h_2^{\mathsf{sk}} \cdot h_3^{(\mathsf{sk}^{x+1})} \cdot h_4^{\mathsf{sk}'} \wedge \mathtt{com}_1 = g^{o_1} \cdot h^{\mathsf{B}^{\mathsf{old}}+v} \wedge \mathtt{com}_2 = g^{o_2} \cdot h^{\mathsf{sk}} \wedge \mathtt{com}_3 = g^{o_3} \cdot h^{(\mathsf{sk}^{x+1})} \wedge \mathtt{com}_4 = g^{o_4} \cdot h^{\mathsf{sk}'} \wedge \varkappa = \tilde{\alpha} \cdot \tilde{\beta}_1^{\mathsf{B}^{\mathsf{old}}} \cdot \tilde{\beta}_2^{\mathsf{sk}} \cdot \tilde{\beta}_3^{\mathsf{sk}^x} \cdot \tilde{\beta}_4^{\mathsf{sk}'} \cdot \tilde{g}^r \wedge \mathsf{tag} = g^{(\mathsf{sk}^{x+1})} \wedge \tilde{C}_1 = g^z \wedge \tilde{C}_2 = \mathtt{pk}_{\mathsf{A}}^z \cdot g_1^v \wedge \mathsf{B}^{\mathsf{old}} + v \in [0, \mathsf{B}_{\mathsf{max}}]\}$
- $\mathsf{x} = (\bar{\varkappa}, \mathsf{sn}, \mathtt{com}, \mathtt{com}_1, \mathtt{com}_2, \mathtt{com}_3, \mathtt{com}_4, h, \varkappa, \mathsf{tag}, \tilde{\chi}_t, \mathtt{pk}_{\mathsf{A}})$
- $\mathsf{w} = (\mathsf{B}^{\mathsf{old}}, \mathsf{sk}, \mathsf{sk}^x, \mathsf{sk}', o, o_1, o_2, o_3, o_4, \bar{r}, \bar{r}', r, r', v, z)$

▷ Upon receiving $(\mathtt{Proof}, \mathsf{sid}, \pi)$ from $\mathcal{F}_{\mathsf{NIZK}}$, set $\mathfrak{I} \leftarrow (\sigma_{\mathsf{CUS}}^{\mathsf{RND}}, \sigma_{\mathbb{I}}^{\mathsf{RND}}, \mathsf{x}, \pi)$.
▷ Call $\mathcal{F}_{\mathsf{B}}^{\mathsf{SA}}$ with $(\mathtt{Broadcast}, \mathsf{sid}, \mathfrak{I})$.

Fig. 7: Stablecoin issuance – U

Each issuer verifies $\sigma_{\mathsf{CUS}}^{\mathsf{RND}}$, $\sigma_{\mathbb{I}}^{\mathsf{RND}}$, and NIZK proof π. Additionally, each issuer examines its local database $\mathcal{D}_i$ to ensure the absence of any double-spending occurrences. If all these checks pass, the encryption of the total value in circulation of the stablecoin is updated by multiplying the received ciphertext $\tilde{\chi}_t$ with the existing one $\tilde{\chi}_i$ recorded locally. This update occurs seamlessly due to the homomorphism property of the underlying ElGamal encryption, allowing for the modification of the total value without the need to reveal the specific value v. The updated ciphertext then is output to the environment $\mathcal{Z}$. Also, the issuer records tag and sn, and signs new blinded account to get $\sigma_i^{\mathsf{new,blind}}$ which is sent back to U. I_i acts as follows upon receiving $(\mathtt{Broadcasted}, \mathsf{sid}, \mathfrak{I}, \mathsf{mid})$ from $\mathcal{F}_{\mathsf{B}}^{\mathsf{SA}}$:

▷ Parse $\mathfrak{I} = (\sigma_{\mathsf{CUS}}^{\mathsf{RND}}, \sigma_{\mathbb{I}}^{\mathsf{RND}}, \mathsf{x}, \pi)$, and $\mathsf{x} = (\bar{\varkappa}, \mathsf{sn}, \mathtt{com}, \mathtt{com}_1, \mathtt{com}_2, \mathtt{com}_3, \mathtt{com}_4, h, \varkappa, \mathsf{tag}, \tilde{\chi}_t, \mathtt{pk}_{\mathsf{A}})$.
▷ Ignore $\mathfrak{I}$ if at least one of the following conditions holds:
- sn or tag already exists in $\mathcal{D}_i$
- Parse $\sigma_{\mathsf{CUS}}^{\mathsf{RND}} = (\bar{h}', \bar{s}')$ and if $\bar{h}' = 1$ or if $e(\bar{h}', \bar{\varkappa}) \neq e(\bar{s}', \bar{\tilde{g}})$.
- Call $\mathcal{F}_{\mathrm{RO}}$ with $(\mathtt{Query}, \mathsf{sid}, \mathtt{com})$, and $(\mathtt{Query.Re}, \mathsf{sid}, h')$ is received where $h \neq h'$.
- Parse $\sigma_{\mathbb{I}}^{\mathsf{RND}} = (h', s')$ and if $h' = 1$ or if $e(h', \varkappa) \neq e(s', \tilde{g})$.
- Upon calling $\mathcal{F}_{\mathsf{NIZK}}$ with $(\mathtt{Verify}, \mathsf{sid}, \mathsf{x}, \pi)$, $(\mathtt{Vrfed}, \mathsf{sid}, 0)$ is received.

▷ Else, save tag and sn in $\mathcal{D}_i$.

▷ Compute $\tilde{\chi}_i \leftarrow \tilde{\chi}_i \cdot \tilde{\chi}_t$. Record $\tilde{\chi}_i$, and output $(\texttt{Iss.End}, \mathsf{sid}, \tilde{\chi}_t)$ to $\mathcal{Z}$.
▷ Compute $\sigma_i^{\mathsf{new,blind}}$ similar to the *user registration* protocol (where $\sigma_i^{\mathsf{blind}}$ was computed).
▷ Call $\mathcal{F}_{\mathsf{B}}^{\mathsf{SA}}$ with $(\texttt{Send.Back}, \mathsf{sid}, \sigma_i^{\mathsf{new,blind}}, \mathsf{mid})$.

Fig. 8: Stablecoin issuance – I_i

U upon receiving $(\texttt{Received}, \mathsf{sid}, \mathsf{I}_i, \sigma_i^{\mathsf{new,blind}})$ from $\mathcal{F}_{\mathsf{B}}^{\mathsf{SA}}$ unblinds the received blind signature share $\sigma_i^{\mathsf{new,blind}}$ to get σ_i^{new}. U disregards it if found to be invalid. Upon accumulating a sufficient number Γ of valid signatures, U combines them to generate a single aggregated signature $\sigma_{\mathbb{I}}^{\mathsf{new}}$.

3.2.4 Stablecoin transfer. The sender $\mathsf{U_s}$ encrypts v under the receiver $\mathsf{U_r}$'s public key $\mathtt{pk_r}$, generating χ_1, while keeping $\mathtt{pk_r}$ secret (for privacy concerns) by generating χ_2 (which is used in NIZK proof); $\mathsf{U_s}$ encrypts $\mathtt{pk_r}$ under public value W, generating χ_2. $\mathsf{U_s}$ encrypts its public key $\mathtt{pk_s}$ under $\mathtt{pk_r}$, generating χ_3 (to help $\mathsf{U_r}$ identify $\mathsf{U_s}$ by decrypting χ_3). Additionally, $\mathsf{U_s}$ encrypts the constant 0 under $\mathtt{pk_r}$, generating χ_4.[10] χ_2 and χ_4 are used in the proofs of the sender and receiver respectively to provide a reference to $\mathtt{pk_r}$. $\mathsf{U_s}$ initiates *stablecoin transfer* protocol with issuers upon receiving $(\texttt{Gen.ST}, \mathsf{sid}, \mathsf{U_r}, v)$ from $\mathcal{Z}$ as follows.

▷ Compute $\mathtt{com}, \mathtt{com}_1, \mathtt{com}_2, \mathtt{com}_3, \mathtt{com}_4$ and h similar to the *stablecoin issuance* protocol with only difference that the new balance is $\mathsf{B}^{\mathsf{old}} - v$.
▷ Compute $\sigma_{\mathbb{I}}^{\mathsf{RND}}$, $\varkappa$, and tag similar to the *stablecoin issuance* protocol.
▷ Call $\mathcal{F}_{\mathsf{KeyReg}}$ with $(\texttt{Key.Retrieval}, \mathsf{sid}, \mathsf{U_r})$. Upon receiving $(\texttt{Key.Retrieved}, \mathsf{sid}, \mathsf{U_r}, \mathtt{pk_r})$, pick $(z, y, q, t) \xleftarrow{\$} \mathbb{Z}_p$; and set:
- $\chi_1 = (C_1, C_2) = (g^z, \mathtt{pk_r}^z \cdot g_1^v)$
- $\chi_2 = (C_1', C_2') = (g^y, W^y \cdot \mathtt{pk_r})$
- $\chi_3 = (C_1'', C_2'') = (g^q, \mathtt{pk_r}^q \cdot \mathtt{pk_s})$
- $\chi_4 = (C_1''', C_2''') = (g^t, \mathtt{pk_r}^t)$

▷ Call $\mathcal{F}_{\mathsf{NIZK}}$ with input $(\texttt{Prove}, \mathsf{sid}, \mathsf{x}, \mathsf{w})$, for the relation:
- $\mathcal{R}(\mathsf{x}, \mathsf{w}) = \mathsf{NIZK}\{\mathtt{com} = g^o \cdot h_1^{\mathsf{B}^{\mathsf{old}}-v} \cdot h_2^{\mathsf{sk_s}} \cdot h_3^{(\mathsf{sk_s}^{x+1})} \cdot h_4^{\mathsf{sk_s'}} \wedge \mathtt{com}_1 = g^{o_1} \cdot h^{\mathsf{B}^{\mathsf{old}}-v} \wedge \mathtt{com}_2 = g^{o_2} \cdot h^{\mathsf{sk_s}} \wedge \mathtt{com}_3 = g^{o_3} \cdot h^{(\mathsf{sk_s}^{x+1})} \wedge \mathtt{com}_4 = g^{o_4} \cdot h^{\mathsf{sk_s'}} \wedge \varkappa = \tilde{\alpha} \cdot \tilde{\beta}_1^{\mathsf{B}^{\mathsf{old}}} \cdot \tilde{\beta}_2^{\mathsf{sk_s}} \cdot \tilde{\beta}_3^{(\mathsf{sk_s}^x)} \cdot \tilde{\beta}_4^{\mathsf{sk_s'}} \cdot \tilde{g}^r \wedge \mathsf{tag} = g^{(\mathsf{sk_s}^{x+1})} \wedge C_1 = g^z \wedge C_2 = (C_2'/W^y)^z \cdot g_1^v \wedge C_1' = g^y \wedge C_1'' = g^q \wedge C_2'' = (C_2'/W^y)^q \cdot g^{\mathsf{sk_s'}} \wedge C_1''' = g^t \wedge C_2''' = (C_2'/W^y)^t \wedge v \in [0, \mathsf{B}^{\mathsf{old}}]\}$
- $\mathsf{x} = (\mathtt{com}, \mathtt{com}_1, \mathtt{com}_2, \mathtt{com}_3, \mathtt{com}_4, h, \varkappa, \mathsf{tag}, \chi_1, \chi_2, \chi_3, \chi_4, W)$
- $\mathsf{w} = (\mathsf{B}^{\mathsf{old}}, \mathsf{sk_s}, \mathsf{sk_s}^x, \mathsf{sk_s'}, v, z, y, q, t, o, o_1, o_2, o_3, o_4, r, r')$

[10] The usage of χ_4 in *stablecoin transfer* (resp. χ_2 in *stablecoin burn*) protocol is to prevent malicious sender and receiver (resp. malicious user) from breaking the integrity of currency transfer (resp. burn) e.g., by generating fake money, without relying on any security assumption e.g., hardness of discrete logarithm.

▷ Upon receiving $(\texttt{Proof}, \mathsf{sid}, \pi)$ from $\mathcal{F}_{\mathsf{NIZK}}$, set $\mathfrak{T} \leftarrow (\sigma_{\mathbb{I}}^{\mathsf{RND}}, \mathsf{x}, \pi)$.
▷ Call $\mathcal{F}_{\mathsf{B}}^{\mathsf{SA}}$ with $(\texttt{Broadcast}, \mathsf{sid}, \mathfrak{T})$.

Fig. 9: Stablecoin transfer – $\mathsf{U_s}$

Similar to *stablecoin issuance* protocol each issuer verifies $\sigma_{\mathbb{I}}^{\mathsf{RND}}$, and π. Additionally, each issuer searches its local database $\mathcal{D}_i$ for double-spending checking. If all these checks pass, the issuer signs new blinded account to get $\sigma_i^{\mathsf{new,blind}}$ which is sent back to U. I_i upon receiving $(\texttt{Broadcasted}, \mathsf{sid}, \mathfrak{T}, \mathsf{mid})$ from $\mathcal{F}_{\mathsf{B}}^{\mathsf{SA}}$ acts as follows.

▷ Parse $\mathfrak{T} = (\sigma_{\mathbb{I}}^{\mathsf{RND}}, \mathsf{x}, \pi)$, and $\mathsf{x} = (\mathtt{com}, \mathtt{com}_1, \mathtt{com}_2, \mathtt{com}_3, \mathtt{com}_4, h, \varkappa, \mathsf{tag}, \chi_1, \chi_2, \chi_3, \chi_4, W)$.
▷ Ignore if at least one of the following conditions holds:
  • tag already exists in $\mathcal{D}_i$.
  • Parse $\sigma_{\mathbb{I}}^{\mathsf{RND}} = (h', s')$ and if $h' = 1$ or if $e(h', \varkappa) \neq e(s', \tilde{g})$.
  • Upon calling $\mathcal{F}_{\mathsf{NIZK}}$ with $(\texttt{Verify}, \mathsf{sid}, \mathsf{x}, \pi)$, $(\texttt{Vrfed}, \mathsf{sid}, 0)$ is received.
▷ Else, save tag, and $\phi = (\chi_1, \chi_3, \chi_4)$ in $\mathcal{D}_i$.
▷ Compute $\sigma_i^{\mathsf{new,blind}}$ similar to the *stablecoin issuance* protocol.
▷ Call $\mathcal{F}_{\mathsf{B}}^{\mathsf{SA}}$ with $(\texttt{Send.Back}, \mathsf{sid}, \sigma_i^{\mathsf{new,blind}}, \mathsf{mid})$.

Fig. 10: Stablecoin transfer – I_i

$\mathsf{U_s}$ upon receiving $(\texttt{Received}, \mathsf{sid}, \mathsf{I}_i, \sigma_i^{\mathsf{new,blind}})$ from $\mathcal{F}_{\mathsf{B}}^{\mathsf{SA}}$ unblinds the received blind signature share $\sigma_i^{\mathsf{new,blind}}$ to get σ_i^{new}, and generates a single aggregated signature $\sigma_{\mathbb{I}}^{\mathsf{new}}$ (as described in earlier protocol). $\mathsf{U_s}$ outputs $(\texttt{Gen.ST.End}, \mathsf{sid}, \mathsf{U_r}, v, \phi)$ to $\mathcal{Z}$ where $\phi = (\chi_1, \chi_3, \chi_4)$.

3.2.5 Stablecoin claim. The receiver $\mathsf{U_r}$ decrypts χ_1, and χ_3 to identify v, and $\mathsf{U_s}$ respectively. $\mathsf{U_r}$ updates their balance with respect to v, and proves ownership of (χ_1, χ_4) (by proving decryption via NIZK – see the relation $\mathcal{R}(\mathsf{x}, \mathsf{w})$) so that with the confirmation of issuers (who receive ϕ from $\mathcal{Z}$) the balance is increased by v. $\mathsf{U_r}$ initiates *stablecoin claim* protocol with issuers upon receiving $(\texttt{Clm.ST}, \mathsf{sid}, \phi)$ from $\mathcal{Z}$ as follows.

▷ Parse $\phi = (\chi_1, \chi_3, \chi_4)$, $\chi_1 = (C_1, C_2)$, $\chi_3 = (C_1'', C_2'')$, and $\chi_4 = (C_1''', C_2''')$.
▷ Compute $g_1^v = C_2/(C_1)^{\mathsf{sk}_\mathsf{r}'}$ and $\mathtt{pk}_\mathsf{s} = C_2''/(C_1'')^{\mathsf{sk}_\mathsf{r}'}$. Extract v from g_1^v.[a]
▷ Compute $\mathtt{com}, \mathtt{com}_1, \mathtt{com}_2, \mathtt{com}_3, \mathtt{com}_4$, h, $\sigma_{\mathbb{I}}^{\mathsf{RND}}$, $\varkappa$, and tag similar to the *stablecoin issuance* protocol.

▷ Call $\mathcal{F}_{\mathsf{NIZK}}$ with input $(\texttt{Prove}, \mathsf{sid}, \mathsf{x}, \mathsf{w})$, for the relation:
- $\mathcal{R}(\mathsf{x}, \mathsf{w}) = \mathsf{NIZK}\{\texttt{com} = g^{o} \cdot h_1^{\mathsf{B}^{\mathsf{old}}+v} \cdot h_2^{\mathsf{sk}_\mathsf{r}} \cdot h_3^{(\mathsf{sk}_\mathsf{r}^{x+1})} \cdot h_4^{\mathsf{sk}'_\mathsf{r}} \wedge \texttt{com}_1 = g^{o_1} \cdot h^{\mathsf{B}^{\mathsf{old}}+v} \wedge \texttt{com}_2 = g^{o_2} \cdot h^{\mathsf{sk}_\mathsf{r}} \wedge \texttt{com}_3 = g^{o_3} \cdot h^{(\mathsf{sk}_\mathsf{r}^{x+1})} \wedge \texttt{com}_4 = g^{o_4} \cdot h^{\mathsf{sk}'_\mathsf{r}} \wedge \varkappa = \tilde{\alpha} \cdot \tilde{\beta}_1^{\mathsf{B}^{\mathsf{old}}} \cdot \tilde{\beta}_2^{\mathsf{sk}_\mathsf{r}} \cdot \tilde{\beta}_3^{(\mathsf{sk}_\mathsf{r}^{x})} \cdot \tilde{\beta}_4^{\mathsf{sk}'_\mathsf{r}} \cdot \tilde{g}^{r} \wedge \mathsf{tag} = g^{(\mathsf{sk}_\mathsf{r}^{x+1})} \wedge C_2 = C_1^{\mathsf{sk}'_\mathsf{r}} \cdot g_1^{v} \wedge C_2''' = (C_1''')^{\mathsf{sk}'_\mathsf{r}} \wedge \mathsf{B}^{\mathsf{old}} + v \in [0, \mathsf{B}_{\mathsf{max}}]\}$
- $\mathsf{x} = (\texttt{com}, \texttt{com}_1, \texttt{com}_2, \texttt{com}_3, \texttt{com}_4, h, \varkappa, \mathsf{tag}, \chi_1, \chi_3, \chi_4)$
- $\mathsf{w} = (\mathsf{B}^{\mathsf{old}}, \mathsf{sk}_\mathsf{r}, \mathsf{sk}_\mathsf{r}^{x}, \mathsf{sk}'_\mathsf{r}, v, o, o_1, o_2, o_3, o_4, r, r')$

▷ Upon receiving $(\texttt{Proof}, \mathsf{sid}, \pi)$ from $\mathcal{F}_{\mathsf{NIZK}}$, set $\mathfrak{D} \leftarrow (\sigma_{\mathbb{I}}^{\mathsf{RND}}, \mathsf{x}, \pi)$.
▷ Call $\mathcal{F}_{\mathsf{B}}^{\mathsf{SA}}$ with $(\texttt{Broadcast}, \mathsf{sid}, \mathfrak{D})$.

[a] see Section 5 on how to efficiently extract v.

Fig. 11: Stablecoin claim – U_r

Each issuing entity operates akin to prior protocols, with the exception that it checks its database for ϕ. Subsequently, it awaits an instruction from the environment $\mathcal{Z}$, which should include the identical ϕ, and then proceeds to sign new blinded account. I_i upon receiving $(\texttt{Broadcasted}, \mathsf{sid}, \mathfrak{D}, \mathsf{mid})$ from $\mathcal{F}_{\mathsf{B}}^{\mathsf{SA}}$ act as follows.

▷ Parse $\mathfrak{D} = (\sigma_{\mathbb{I}}^{\mathsf{RND}}, \mathsf{x}, \pi)$, and $\mathsf{x} = (\texttt{com}, \texttt{com}_1, \texttt{com}_2, \texttt{com}_3, \texttt{com}_4, h, \varkappa, \mathsf{tag}, \chi_1, \chi_3, \chi_4)$.
▷ Ignore if at least one of the following conditions holds:
- tag or χ_1 already exists in $\mathcal{D}_i$.
- Upon calling $\mathcal{F}_{\mathsf{NIZK}}$ with $(\texttt{Verify}, \mathsf{sid}, \mathsf{x}, \pi)$, $(\texttt{Vrfed}, \mathsf{sid}, 0)$ is received.
- Parse $\sigma_{\mathbb{I}}^{\mathsf{RND}} = (h', s')$ and if $h' = 1$ or if $e(h', \varkappa) \neq e(s', \tilde{g})$.
- There does not exist $\phi = (\chi_1, \chi_3, \chi_4)$ recorded in $\mathcal{D}_i$.

▷ Upon receiving $(\texttt{Clm.ST.Issuer}, \mathsf{sid}, \phi^*)$ from $\mathcal{Z}$, parse $\phi^* = (\chi_1^*, \chi_3^*, \chi_4^*)$. Proceed if $\chi_1^* = \chi_1$, $\chi_3^* = \chi_3$, and $\chi_4^* = \chi_4$.
▷ Save tag, and χ_1 in $\mathcal{D}_i$.
▷ Compute $\sigma_i^{\mathsf{new,blind}}$ similar to the *stablecoin issuance* protocol.
▷ Call $\mathcal{F}_{\mathsf{B}}^{\mathsf{SA}}$ with $(\texttt{Send.Back}, \mathsf{sid}, \sigma_i^{\mathsf{new,blind}}, \mathsf{mid})$.

Fig. 12: Stablecoin claim – I_i

U_r upon receiving $(\texttt{Received}, \mathsf{sid}, \mathsf{I}_i, \sigma_i^{\mathsf{new,blind}})$ from $\mathcal{F}_{\mathsf{B}}^{\mathsf{SA}}$ acts similar to the *stablecoin issuance* protocol to unblind the received blind signature share, and generate a single aggregated signature on their new account $\sigma_{\mathbb{I}}^{\mathsf{new}}$. Finally, U_r calls $\mathcal{F}_{\mathsf{KeyReg}}$ with $(\texttt{id.Retrieval}, \mathsf{sid}, \mathsf{pk}_\mathsf{s})$, upon receiving back $(\texttt{id.Retrieved}, \mathsf{sid}, \mathsf{U}_\mathsf{s}, \mathsf{pk}_\mathsf{s})$, outputs $(\texttt{Clmed.ST}, \mathsf{sid}, \phi, \mathsf{U}_\mathsf{s}, v)$ to $\mathcal{Z}$.

3.2.6 Stablecoin burn. The user U receives the value of stablecoin that is burned v and label ℓ from $\mathcal{Z}$. As described in section 2.4, ℓ is to specify the type

of asset against which U intends to initiate the burning process for withdrawal which facilitates interoperability within our model. For instance, user can burn v stablecoin to get another digital asset in a blockchain system (specified via ℓ) with the help of custodian (as we will see in the *proof of burn* protocol).
U employs encryption to hide the value v that is burned, resulting in the creation of a ciphertext χ_1 under their public key. Moreover, U utilizes their public key to encrypt the value 0 (that will be used in the *proof of burn* protocol). Additionally, to facilitate privacy-enhanced auditing, $\tilde{\chi}_t$ is generated as an encryption of v under the public key of AUD ($\mathtt{pk}_\mathsf{A}$), serving the same purpose as outlined in the *stablecoin issuance* protocol.
Subsequently, U proceeds to update their account and provides a NIZK proof, demonstrating the consistency among all ciphertexts, the signature on the old account, and the new blinded account. U initiates *stablecoin burn* protocol with issuers upon receiving $(\mathtt{Brn}, \mathsf{sid}, v, \ell)$ from $\mathcal{Z}$ as follows.

▷ Compute $\mathtt{com}, \mathtt{com}_1, \mathtt{com}_2, \mathtt{com}_3, \mathtt{com}_4$ and h similar to the *stablecoin transfer* protocol.
▷ Compute $\sigma_{\mathbb{I}}^{\mathsf{RND}}$, $\varkappa$, and tag similar to the *stablecoin transfer* protocol.
▷ Pick $(z, q, t) \xleftarrow{\$} \mathbb{Z}_p$, set:
- $\chi_1 = (C_1, C_2) = (g^z, \mathtt{pk}^z \cdot g_1^v)$
- $\chi_2 = (C_1', C_2') = (g^t, \mathtt{pk}^t)$
- $\tilde{\chi}_t = (\tilde{C}_1, \tilde{C}_2) = (g^q, \mathtt{pk}_\mathsf{A}^q \cdot g_1^v)$

▷ Call $\mathcal{F}_{\mathsf{NIZK}}$ with input $(\mathtt{Prove}, \mathsf{sid}, \mathsf{x}, \mathsf{w})$, for the relation:
- $\mathcal{R}(\mathsf{x}, \mathsf{w}) = \mathsf{NIZK}\{\mathtt{com} = g^o \cdot h_1^{\mathsf{B}^{\mathsf{old}} - v} \cdot h_2^{\mathsf{sk}} \cdot h_3^{(\mathsf{sk}^{x+1})} \cdot h_4^{\mathsf{sk}'} \wedge \mathtt{com}_1 = g^{o_1} \cdot h^{\mathsf{B}^{\mathsf{old}} - v} \wedge \mathtt{com}_2 = g^{o_2} \cdot h^{\mathsf{sk}} \wedge \mathtt{com}_3 = g^{o_3} \cdot h^{(\mathsf{sk}^{x+1})} \wedge \mathtt{com}_4 = g^{o_4} \cdot h^{\mathsf{sk}'} \wedge \varkappa = \tilde{\alpha} \cdot \tilde{\beta}_1^{\mathsf{B}^{\mathsf{old}}} \cdot \tilde{\beta}_2^{\mathsf{sk}} \cdot \tilde{\beta}_3^{(\mathsf{sk}^x)} \cdot \tilde{\beta}_4^{\mathsf{sk}'} \cdot \tilde{g}^r \wedge \mathsf{tag} = g^{(\mathsf{sk}^{x+1})} \wedge C_1 = g^z \wedge C_2 = g^{\mathsf{sk}' \cdot z} \cdot g_1^v \wedge \tilde{C}_1 = g^q \wedge \tilde{C}_2 = \mathtt{pk}_\mathsf{A}^q \cdot g_1^v \wedge C_1' = g^t \wedge C_2' = g^{\mathsf{sk}' \cdot t} \wedge v \in [0, \mathsf{B}^{\mathsf{old}}]\}$
- $\mathsf{x} = (\mathtt{com}, \mathtt{com}_1, \mathtt{com}_2, \mathtt{com}_3, \mathtt{com}_4, h, \varkappa, \mathsf{tag}, \chi_1, \chi_2, \tilde{\chi}_t, \mathtt{pk}_\mathsf{A}, \ell)$
- $\mathsf{w} = (\mathsf{B}^{\mathsf{old}}, \mathsf{sk}, \mathsf{sk}^x, \mathsf{sk}', v, o, o_1, o_2, o_3, o_4, r, r', z, q, t)$

▷ Upon receiving $(\mathtt{Proof}, \mathsf{sid}, \pi)$ from $\mathcal{F}_{\mathsf{NIZK}}$, set $\mathfrak{B} \leftarrow (\sigma_{\mathbb{I}}^{\mathsf{RND}}, \mathsf{x}, \pi)$.
▷ Call $\mathcal{F}_{\mathsf{B}}^{\mathsf{SA}}$ with $(\mathtt{Broadcast}, \mathsf{sid}, \mathfrak{B})$.

Fig. 13: Stablecoin burn – U

Similar to previous protocols each issuer verifies signature, NIZK proof, and double-spending tag. If all these checks pass, similar to the *stablecoin issuance* protocol the encryption of the total value in circulation of the stablecoin is updated by dividing the existing ciphertext $\tilde{\chi}_i$ recorded locally by the received ciphertext $\tilde{\chi}_t$. Finally, the issuer outputs $(\eta, \ell, \tilde{\chi}_i)$ to $\mathcal{Z}$ showing that $\eta = (\chi_1, \chi_2)$ is associated to ℓ and updated total value in circulation of stablecoin is handled via $\tilde{\chi}_i$. I_i upon receiving $(\mathtt{Broadcasted}, \mathsf{sid}, \mathfrak{B}, \mathsf{mid})$ from $\mathcal{F}_{\mathsf{B}}^{\mathsf{SA}}$ acts as follows.

▷ Parse $\mathfrak{B} = (\sigma_{\mathbb{I}}^{\mathsf{RND}}, \mathsf{x}, \pi)$, and $\mathsf{x} = (\mathtt{com}, \mathtt{com}_1, \mathtt{com}_2, \mathtt{com}_3, \mathtt{com}_4, h, \varkappa, \mathsf{tag}, \chi_1, \chi_2, \tilde{\chi}_t, \mathsf{pk}_{\mathsf{A}}, \ell)$.
▷ Ignore $\mathfrak{B}$ if at least one of the following conditions holds:
- tag already exists in $\mathcal{D}_i$.
- Upon calling $\mathcal{F}_{\mathsf{NIZK}}$ with $(\mathtt{Verify}, \mathsf{sid}, \mathsf{x}, \pi)$, $(\mathtt{Vrfed}, \mathsf{sid}, 0)$ is received.
- Parse $\sigma_{\mathbb{I}}^{\mathsf{RND}} = (h', s')$ and if $h' = 1$ or if $e(h', \varkappa) \neq e(s', \tilde{g})$.

▷ Compute $\tilde{\chi}_i \leftarrow \tilde{\chi}_i / \tilde{\chi}_t$. Record $\tilde{\chi}_i$, and output $(\mathtt{Brn.End}, \mathsf{sid}, \eta, \ell, \tilde{\chi}_t)$ to $\mathcal{Z}$ where $\eta = (\chi_1, \chi_2)$.
▷ Save tag in $\mathcal{D}_i$, compute $\sigma_i^{\mathsf{new,blind}}$ similar to the *stablecoin transfer* protocol.
▷ Call $\mathcal{F}_{\mathsf{B}}^{\mathsf{SA}}$ with $(\mathtt{Send.Back}, \mathsf{sid}, \sigma_i^{\mathsf{new,blind}}, \mathsf{mid})$.

Fig. 14: Stablecoin burn – I_i

The user U upon receiving $(\mathtt{Received}, \mathsf{sid}, \mathsf{I}_i, \sigma_i^{\mathsf{new,blind}})$ from $\mathcal{F}_{\mathsf{B}}^{\mathsf{SA}}$ unblinds $\sigma_i^{\mathsf{new,blind}}$ to get σ_i^{new}, and generates a single aggregated signature $\sigma_{\mathbb{I}}^{\mathsf{new}}$.

3.2.7 Proof of burn. The user U receives $\eta = (\chi_1, \chi_2)$ and ℓ from $\mathcal{Z}$. U proves to CUS that both ciphertexts belongs to them and also proves the value of encryption is v. U initiates *proof of burn* protocol upon receiving $(\mathtt{PoB}, \mathsf{sid}, \eta, \ell)$ from $\mathcal{Z}$ as follows.

▷ Parse $\eta = (\chi_1, \chi_2)$, $\chi_1 = (C_1, C_2)$, and $\chi_2 = (C_1', C_2')$.
▷ Compute $g_1^v = C_2/(C_1)^{\mathsf{sk}'}$. Extract v from g_1^v.
▷ Call $\mathcal{F}_{\mathsf{NIZK}}$ with input $(\mathtt{Prove}, \mathsf{sid}, \mathsf{x}, \mathsf{w})$, for the relation:
- $\mathcal{R}(\mathsf{x}, \mathsf{w}) = \mathsf{NIZK}\{C_2 = C_1^{\mathsf{sk}'} \cdot g_1^v \wedge C_2' = (C_1')^{\mathsf{sk}'}\}$
- $\mathsf{x} = (\eta, v, \ell)$
- $\mathsf{w} = \mathsf{sk}'$

▷ Upon receiving $(\mathtt{Proof}, \mathsf{sid}, \pi)$ from $\mathcal{F}_{\mathsf{NIZK}}$, set $\mathfrak{V} \leftarrow (\mathsf{x}, \pi)$.
▷ Call $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{SSA}}$ with $(\mathtt{Send}, \mathsf{sid}, \mathsf{CUS}, \mathfrak{V})$.

Fig. 15: Proof of burn – U

CUS verifies the NIZK proof, also checks if η has already been claimed or not. If checks pass, and upon receiving the same values of η, and ℓ from $\mathcal{Z}$, CUS accepts user's proof of burn (of v stablecoins) and outputs $(\eta, \mathsf{U}, v, \ell)$ to $\mathcal{Z}$. CUS upon receiving $(\mathtt{Received}, \mathsf{sid}, \mathsf{U}, \mathfrak{V})$ from $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{SSA}}$ acts as follows.

▷ Parse $\mathfrak{V} = (\mathsf{x}, \pi)$, and $\mathsf{x} = (\eta, v, \ell)$.
▷ Ignore $\mathfrak{V}$ if upon calling $\mathcal{F}_{\mathsf{NIZK}}$ with $(\mathtt{Verify}, \mathsf{sid}, \mathsf{x}, \pi)$, $(\mathtt{Vrfed}, \mathsf{sid}, 0)$ is received or η already exists.
▷ Upon receiving $(\mathtt{PoB.CUS}, \mathsf{sid}, \eta^*, \ell^*)$ from $\mathcal{Z}$, proceed if $\eta^* = \eta$, and $\ell^* = \ell$.

▷ Record η, and output $(\texttt{PoB.End}, \mathsf{sid}, \eta, \mathsf{U}, v, \ell)$ to $\mathcal{Z}$.

Fig. 16: Proof of burn – CUS

3.2.8 Reserve audit. The environment $\mathcal{Z}$ instructs the auditor AUD to audit total value of stablecoin in circulation TV with respect to reserves RV. At any given point in time, AUD can decrypt the most recent encryption of total value of stablecoin in circulation $\tilde{\chi}$ to get TV and compare it with RV given by $\mathcal{Z}$. AUD initiates *reserve audit* upon receiving $(\texttt{Audit}, \mathsf{sid}, \tilde{\chi}, \mathsf{RV})$ from $\mathcal{Z}$.

▷ Parse $\tilde{\chi} = (\tilde{C}_1, \tilde{C}_2)$. Compute $g_1^{\mathsf{TV}} = \tilde{C}_2 / \tilde{C}_1^{\mathsf{sk_A}}$. Extract TV from g_1^{TV}.
▷ If $\mathsf{RV} \geq \mathsf{TV}$, set $b \leftarrow 1$. Else, set $b \leftarrow 0$.
▷ Output $(\texttt{Audit.End}, \mathsf{sid}, b)$ to $\mathcal{Z}$.

Fig. 17: Reserve audit – AUD

4 Discussion

4.1 Regulation-friendliness

In Section 2.1 we explained that no central regulator exists, and any desired regulation can be managed by issuers in a distributed manner. To maintain clarity in our ideal functionality, we intentionally did not incorporate explicit regulatory compliance directly within the model. Our formal modeling and construction encompass several critical aspects necessary for regulatory compliance. Our protocol enables easy and intuitive development of regulatory audit operations on top of the existing framework.
The incorporation of regulatory requirements can be seamlessly accommodated within the framework of PARScoin as follows.

- **KYC, AML, CFT checks.** A PARScoin account can include a user's unique real-world identifier (e.g., social security number in the US) denoted by uid: $\mathsf{acc} = (\mathsf{B}, \mathsf{sk}, \mathsf{sk}^x, \mathsf{sk}', \mathsf{uid})$. If required, users can incorporate additional proofs in their NIZK proof π, such as $\mathcal{R}_{\mathsf{KYC}}(\mathsf{uid}) = 1$, in a privacy-preserving manner without revealing uid, where $\mathcal{R}_{\mathsf{KYC}}$ represents a KYC (Know Your Customer)-related relation dependent on the application.
  For instance, to comply with CFT (Combating Financing of Terrorism) regulations, a recipient may need to verify that they are not identified as a terrorist according to a public list of unique identifiers of terrorists, thereby enabling the recipient to receive funds without disclosing their identity. Recently, anonymous SyRA signatures have been introduced [19] that intrinsically contain the user's real-world identifiers (each signature contains a pseudonym that is a function of uid). SyRA can be seamlessly utilized; allowing the recipient's PARScoin transaction information $\mathfrak{D}$ to be signed

with their SyRA signature secret key. This facilitates the seamless proof of attributes related to uid (e.g., $\mathcal{R}_{\mathsf{CFT}}(\mathsf{uid}) = 1$). Note that, the issuers of PARScoin can play the role of the issuers of SyRA signatures. Similarly, the sender of a transaction can be required to provide a SyRA signature on their PARScoin transaction $\mathfrak{T}$ to prove that she is not associated with any revoked entities listed in AML (Anti Money Laundering) regulations, enabling her to send funds (e.g., $\mathcal{R}_{\mathsf{AML}}(\mathsf{uid}) = 1$).

- **Account limits.** PARScoin can address account-related restrictions, such as those mentioned by the Bank of England [1], the European Central Bank (ECB) [10], and several other central banks [11], concerning financial stability, preventing bank runs, and evasion of tax. These involve imposing an upper bound on the account balance and the sum of all sent and received values. In PARScoin, two integers, snt and rcv, can be added to the user account showing sum of all sent and received values respectively: $\mathsf{acc} = (\mathsf{B}, \mathsf{sk}, \mathsf{sk}^x, \mathsf{sk}', \mathsf{snt}, \mathsf{rcv})$, allowing the user to provide Zero-Knowledge range proofs that B, snt, and rcv are below associated thresholds (e.g., $\mathsf{B} < \mathsf{B}_{\mathsf{max}}$). For instance, the sender should increase snt by v when she sends v to a receiver. She proves that $\mathsf{snt}^{\mathsf{new}} = \mathsf{snt}^{\mathsf{old}} + v \wedge \mathsf{snt}^{\mathsf{new}} < \mathsf{S}_{\mathsf{max}}$.
- **Privacy revocation.** Our design could easily incorporate features for distributed privacy revocation, drawing inspiration from the PEReDi model [35]. This approach enables the integration of privacy revocation mechanisms through the use of threshold encryption. Specifically, it involves encrypting the user's public key $\mathtt{pk}$ and the transaction value v under the key of regulator(s)/auditor(s). This ensures that, in jurisdictions where it is mandatory for transaction information to be accessible to a (centralized/group of) regulator(s), all necessary information can be encrypted with the regulator's public key. Furthermore, the well-formedness of this encryption could be verified efficiently through NIZK proofs (sigma protocol based, see [35]). This process allows authorized authorities to decrypt transactions when required, revealing the public keys of the sender and receiver, as well as the transaction value.
- **Tracing.** Alongside privacy revocation, our system could also facilitate the distributed tracing of malicious users. This is achieved by employing verifiable secret sharing of the user's tracing secret key during the *user registration* protocol, distributed across issuers. In instances where tracing is necessitated, authorities can execute a special purpose efficient Multi-Party Computation (MPC) to generate tags that uniquely identify transactions (no single issuer can trace a user, avoiding signle point of trust and failure). These tags are footprints of transactions, enabling the system to trace a user's transactions through generating her transactions' tags. For more detailed information, we direct readers to [35].

4.2 Account recovery

The account state model necessitates users to possess knowledge of their most recent account status. Account recovery can be accomplished using standard

recovery methodologies. It is plausible to generate values, such as randomness for blinding, in a pseudorandom manner, drawing from the user's secret key. To establish a backup, the user may simply retain this secret key. The user is tasked with generating tag values `tag` up to the point of the latest entry in the blockchain. Armed with the retrieved randomness using a pseudorandom number generator, the user can then unblind the blind signature shares of issuers associated with that particular tag in the blockchain. Regarding the retrieval of their balance, the account balance can either be deduced through a brute-force approach or could be obtained by decrypting (publicly available) ciphertexts generated by the user for each transaction, which encrypts their updated balance with a public key whose secret key can also be retrieved using a pseudorandom number generator.

4.3 Blockchain agnosticism, and interoperability

The system functions independently of any blockchain, relying instead on a group of independent issuers responsible for facilitating stablecoin issuance/burn processes, and transaction verification among users. The successful completion of a transaction is immediately determined by obtaining a sufficient number of signatures of distributed issuers, eliminating the need for reliance on the blockchain layer one (L1) settlement (or calling any function of a smart contract). In other words, unlike canonical ways of implementing a fiat-backed stablecoin via smart contracts, our approach does not require any smart contract call (which requires resolution in L1), avoiding time-related costs and also evading fees for submission to L1.
Furthermore, the system demonstrates interoperability with various blockchains. This is achieved by allowing the custodian to implement standard (non-private) smart contract-based withdrawals, such as ERC-20 tokens. As an example, users can burn their stablecoins with a specified label ℓ indicating the desire to withdraw a specific token rather than the fiat collateral. This withdrawal process is facilitated with the assistance of the custodian.

4.4 Self-custody

In PARScoin, users possess complete control and ownership of their stablecoins, marking a departure from usual fiat-backed stablecoins like USDC [55] and USDT [53]. In those systems, a user's stablecoin balance is recorded within a smart contract, subjecting it to potential restrictions such as address blacklisting (in the cases of USDC and USDT) or coin destruction (USDT). In contrast, PARScoin enables users to gain full control over their digital assets from the moment their accounts are credited with stablecoins by issuers. Note that the regulation friendliness of PARScoin can ensure that no transaction is finalized unless both the sender and receiver have proven their compliance with the rules (hence, removing the need for operations like blacklisting).

5 Implementation details and PARScoin performance

In this section, we detail the computation and communication costs associated with PARScoin's stablecoin transfer and claim processes, for both the user (sender and receiver) and the issuers. Our evaluation uses the Charm cryptographic framework [2] and bilinear pairings implemented over the Barreto-Naehrig curve [34]. Experiments were conducted on a system equipped with an Intel Core i7-9850H CPU operating at 2.60GHz and 16GB of RAM. We define the upper bound on user balance as $\mathsf{B_{max}} = 2^n - 1$ and utilize bulletproofs [14] for range proofs.

Avoiding brute-force in decryption. Similar to the approach of [49], a constant-time decryption method can be added to avoid brute-force attempts in decryption. This scheme operates as follows: The encryptor picks two random values $r, s \xleftarrow{\$} \mathbb{Z}_p$, and computes: $X = g^r, \quad Y = \mathsf{pk}^r \cdot h^s, \quad Z = \mathsf{H}(h^s) \oplus v$, and outputs the ciphertext $C = (X, Y, Z)$. The decryptor parses $C = (X, Y, Z)$ and computes $h^s = \frac{Y}{X^{\mathsf{sk}}}$. Next, recovers the plaintext message as $v = \mathsf{H}(h^s) \oplus Z$. Note that this encryption scheme should be used together with the existing ElGamal encryption scheme. For more details, see [49].

In the following, we consider decryption to be constant.

- In the *stablecoin transfer* protocol, the sender requires $(0.05941 + 0.00712n)$ seconds to generate $\mathfrak{T} = (\sigma_{\mathbb{I}}^{\mathsf{RND}}, \mathsf{x}, \pi)$, with a resulting data size of $2.8711\mathsf{KB} + 2\mathsf{log}_2(n)$. Additionally, $(0.03409\Gamma + 0.02332)$ seconds are needed for the sender to unblind Γ signature shares and aggregate them into a consolidated signature.
- In the *stablecoin claim* protocol, the receiver spends $(0.04784 + 0.00712n)$ seconds to generate $\mathfrak{D} = (\sigma_{\mathbb{I}}^{\mathsf{RND}}, \mathsf{x}, \pi)$, with a data size of $1.793\mathsf{KB} + 2\mathsf{log}_2(n)$. The process of unblinding Γ signature shares and aggregating them into a consolidated signature takes $(0.03409\Gamma + 0.02332)$ seconds.
- For each issuer in both protocols, verification of the sender/receiver proof requires $(0.08816 + 0.00356n)$ seconds, and computing the blind signature share takes 0.00445 seconds.

5.1 Zero-knowledge proof efficiency

Zero-knowledge Proofs (ZKPs) are theoretically applicable to all languages within the complexity class NP, as demonstrated in [29]. However, not all of these proofs can be efficiently implemented in practice. Consequently, a significant body of research has been dedicated to developing and realizing efficient ZKPs for various types of statements. In the context of our paper, we use non-interactive zero-knowledge (NIZK) proofs. The most pragmatic approaches for NIZK proofs include (i) Sigma protocols (that are transformed to a non-interactive mode employing the Fiat-Shamir transformation [25]), (ii) zk-SNARKs (zero-knowledge succinct non-interactive argument of knowledge) [31]. Each of these approaches possesses distinct efficiency characteristics, advantages, and drawbacks.

zk-SNARK proofs are characterized by their short proofs and swift verification. Specifically, the proofs possess a fixed size and can be verified in time linearly proportional to the length of the input, rather than the size of the circuit. In theory, zk-SNARKs could be applied to prove algebraic statements (e.g., by representing the exponentiation circuit as a quadratic arithmetic program (QAPs) [27]). However, the circuit for computing a single exponentiation in a group $\mathbb{G}$ comprises at least thousands of gates. In QAP-based zk-SNARKs, the computational cost for the prover scales linearly with the circuit's size, and the generation of a trusted common reference string is required (that also grows proportionally with the circuit's size). Consequently, zk-SNARKs are highly inefficient for proving algebraic statements. In contrast, Sigma protocols can be used to demonstrate knowledge of a discrete logarithm with a fixed number of exponentiation. The observation above holds particular relevance within the context of our system, where proof generation (for algebraic statements) primarily relies on individuals equipped with relatively low computational resources, such as cell phones.
In our cryptographic construction, we only prove statements that can be efficiently represented as algebraic discrete logarithm equations[11] (except for our range proofs). Sigma protocol-based ZKPs excel in efficiency when applied to algebraic discrete logarithm statements. These protocols yield concise proof sizes, demand a small number of operations, and do not require the generation of a trusted common reference string [33, 50].

5.1.1 Sigma protocol. The proofs showcased within this section, which are initially established as interactive protocols requiring a logarithmic number of rounds, can be transformed into non-interactive protocols that ensure security and zero-knowledge within the random oracle model [7] by employing the Fiat-Shamir transform technique. In this transformation, all random challenges are substituted with hashes generated from the transcript accumulated up to that specific juncture, encompassing the statement itself. A significant concern arises when such a protocol operates within a larger, complex system where multiple protocols may be concurrently executed. The standalone security of a protocol may not guarantee its security within the broader context. To address this issue, the provable security framework of the general universal composition model provides the strongest assurance that the system will function correctly, even when all parties share a common random oracle.
To efficiently establish the instantiation of $\mathcal{F}_{\mathsf{NIZK}}$ two viable approaches can be useful. The first option involves adopting the methodology presented in [40], which relies on Fischlin's transform [36]. Alternatively, one can construct a UC-secure NIZK (a.k.a. simulation-extractable NIZK [30]) by leveraging a simulation-sound NIZK scheme and a (perfectly correct) CPA-secure encryption scheme [37].
In the following, the prover and the verifier are denoted by P and V respectively. The witness and the statement of a relation are denoted by w and x respectively.

[11] Regarding the label ℓ in the user's statement, it can be included in the hash function.

Proof of knowledge of discrete logarithm. ($\mathcal{R}(y, x) \iff y = g^x$, $x = (y, g)$, $w = x$)

1. P**:** The prover samples a random value $\kappa \xleftarrow{\$} \mathbb{Z}_q^*$, computes $a = g^\kappa$, and transmits a to the verifier V.
2. V**:** The verifier selects a random challenge $c \xleftarrow{\$} \mathbb{Z}_q$ and communicates c to the prover P.
3. P**:** Upon receiving c, the prover calculates: $z = \kappa + c \cdot x \pmod q$, and sends z back to the verifier V.
4. V**:** The verifier validates the proof by checking the relation: $a \stackrel{?}{=} g^z y^{-c}$. If the equality holds, the verifier accepts; otherwise, it rejects.

Proof of multiplicative relation for exponents.

- $\mathcal{R}(x, w) : \Gamma = g^{a_1} h^{r_1} \wedge \Delta = g^{a_2} h^{r_2} \wedge \Theta = g^{a_3} h^{r_3} = g^{a_1 a_2} h^{r_3} \wedge \Theta = \Gamma^{a_2} h^r$, where $r + r_1 a_2 = r_3 \pmod q$
- $x = (g, h, \Gamma, \Delta, \Theta)$
- $w = (a_1, a_2, a_3, r_1, r_2, r_3)$

1. P**:** For $i = 1, 2, 3$, the prover selects random values $\kappa_i, R_i \xleftarrow{\$} \mathbb{Z}_q$ and computes:
$$v_i = g^{\kappa_i} h^{R_i}, \quad v = \Gamma^{\kappa_2} h^R, \quad \text{where } R \xleftarrow{\$} \mathbb{Z}_q.$$
The prover sends $(\{v_i\}, v)$ for $i = 1, 2, 3$ to the verifier V.
2. V**:** The verifier selects a random challenge $c \xleftarrow{\$} \mathbb{Z}_{2^k}$ (with $2^k < q$) and sends c to the prover P.
3. P**:** The prover computes:
$$s_i = \kappa_i - c a_i, \quad t_i = R_i - c r_i, \quad t = R - c r, \quad \text{for } i = 1, 2, 3.$$
The prover then sends the tuples (s_i, t_i) for $i = 1, 2, 3$ and t to the verifier V.
4. V**:** The verifier checks the following relations:
$$v_1 \stackrel{?}{=} \Gamma^c g^{s_1} h^{t_1}, v_2 \stackrel{?}{=} \Delta^c g^{s_2} h^{t_2}, v_3 \stackrel{?}{=} \Theta^c g^{s_3} h^{t_3}, \quad \text{and} \quad v \stackrel{?}{=} \Theta^c \Gamma^{s_2} h^t.$$
If all equations hold, the verifier accepts; otherwise, it rejects.

5.1.2 Range proof. In order to prove that users possess adequate funds when sending (resp. burning) stablecoins, it is imperative that they prove the positivity of the value v and that their balance B is equal to or exceeds the amount being sent (resp. burned) $v \in [0, \mathsf{B}]$. To achieve this, it is necessary to employ *range proofs* within our system's NIZK proofs. Range proofs serve as a means to verify that the individual proving their claim is aware of an opening to a commitment and that the value committed to falls within a predefined range. For instance, these range proofs can be utilized for two purposes: firstly,

to substantiate that an integer commitment corresponds to a positive numerical value, and secondly, to affirm that the combination of two homomorphic commitments to elements within a field of prime order does not result in an overflow modulo the prime value. One choice for implementing range proofs is to utilize bulletproof [14] that relies on the discrete logarithm assumption and does not require a trusted setup. The proof size is logarithmic in the number of multiplication gates in the circuit for verifying a witness. The bulletproofs are Pedersen commitment-friendly (which fits are construction's design) and the associated relation is defined as: $\mathcal{R}(\mathsf{x},\mathsf{w}) = \{\mathtt{com} = g^m \cdot h^r \wedge m \in [0, 2^n - 1]\}$ where $\mathsf{x} = (h, g, \mathtt{com}, n)$ and $\mathsf{w} = (m, r)$.

Finally, all NIZK relations employed within the protocols of PARScoin can indeed be implemented using (non-interactive) Sigma protocols together with range proofs.

5.2 Signature and encryption efficiency

The underlying signature scheme is described in Appendix A.3. The signature is short and all the required ZKPs involved in the algorithms of the signature scheme are computationally efficient. Moreover, it has a streamlined algorithm for signature verification. Each component of the signature, whether partial or consolidated, consists of precisely two group elements. The signature's size remains constant, unaffected by the number of elements in the user's account (it is important as it can be required to add more regulatory compliance-related information to the user's account) or issuing authorities. Additionally, the verification requires minimal computational effort and cryptographic material exchange, regardless of the number of authorities involved. For more detailed information regarding efficiency see [51]. Regarding encryption, depending on the sub-protocol, it may require two or three exponentiations and one multiplication in group $\mathbb{G}$ for each ciphertext generation. Decryption involves one exponentiation and one multiplication (in the same group $\mathbb{G}$).

6 PARScoin security proof

In the following, we provide our main theorem.

Theorem 1. *Given two polynomials* max_1 *and* max_2*, in the* $\{\mathcal{F}_{\mathsf{KeyReg}}, \mathcal{F}_{\mathsf{RO}}, \mathcal{F}_{\mathsf{Ch}}, \mathcal{F}_{\mathsf{B}}, \mathcal{F}_{\mathsf{NIZK}}\}$*-hybrid model, under the binding property (§10) of Pedersen commitments (§11), the* IND-CPA *security (§5) of ElGamal encryption (§6), the* EUF-CMA *security (§13) of Pointcheval-Sanders signature (§14) in the random oracle model, and the hardness of the d-strong Diffie-Hellman problem (§2), no PPT environment* $\mathcal{Z}$ *can distinguish the real-world execution* $\mathsf{EXEC}_{\Pi_{\mathsf{PARS}},\mathcal{A},\mathcal{Z}}$ *from the ideal-world execution* $\mathsf{EXEC}_{\mathcal{F}_{\mathsf{PARS}},\mathcal{S},\mathcal{Z}}$ *with static corruptions in the presence of an arbitrary number of malicious users, honest auditor, honest custodian, and up to t malicious issuers that are all colluding, with advantage better than*

$$\mathsf{Adv}^{\mathsf{Bind\text{-}com}}_{\mathcal{A}} + \mathsf{max}_1 \cdot \mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathcal{A}} + \mathsf{max}_2 \cdot \mathsf{Adv}^{\mathsf{d\text{-}SDDH}}_{\mathcal{A}} + \mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathcal{A}}$$

6.1 Simulation

We denote the real-world protocol and adversary as Π_{PARS} and $\mathcal{A}$, respectively. The simulator $\mathcal{S}$ is described in the following sections and is designed to ensure that observations in the real-world execution $\mathsf{EXEC}_{\Pi_{\mathsf{PARS}},\mathcal{A},\mathcal{Z}}$ and the ideal-world execution $\mathsf{EXEC}_{\mathcal{F}_{\mathsf{PARS}},\mathcal{S},\mathcal{Z}}$ remain indistinguishable for any probabilistic polynomial-time environment $\mathcal{Z}$.
At the start of the execution, $\mathcal{Z}$ instructs the adversary to corrupt specific parties by sending a message $(\texttt{Corrupt}, \mathsf{sid}, P)$. The simulator $\mathcal{S}$ processes these corruption messages and subsequently notifies $\mathcal{F}_{\mathsf{PARS}}$ of the corrupted parties by forwarding the message $(\texttt{Corrupt}, \mathsf{sid}, P)$. Additionally, $\mathcal{S}$ maintains a record of the identifiers of the corrupted parties.
Internally, the simulator $\mathcal{S}$ executes a version of Π_{PARS} and makes the view of the dummy adversary $\mathcal{A}$ in the ideal world indistinguishable from that of the actual adversary in the real world. To achieve this, $\mathcal{S}$ emulates the functionalities $\mathcal{F}_{\mathsf{KeyReg}}$, $\mathcal{F}_{\mathsf{RO}}$, $\mathcal{F}_{\mathsf{Ch}}$, $\mathcal{F}_{\mathsf{B}}$, and $\mathcal{F}_{\mathsf{NIZK}}$.
During the ideal-world execution $\mathsf{EXEC}_{\mathcal{F}_{\mathsf{PARS}},\mathcal{S},\mathcal{Z}}$, honest dummy parties forward their inputs from $\mathcal{Z}$ to $\mathcal{F}_{\mathsf{PARS}}$. The session identifier sid is chosen by $\mathcal{Z}$. The adversary $\mathcal{A}$ arbitrarily instructs the corrupted parties. The simulator $\mathcal{S}$ submits messages to $\mathcal{F}_{\mathsf{PARS}}$ on behalf of these corrupted parties.
The simulator $\mathcal{S}$, designed to replicate the view of $\mathcal{A}$ (for example, by simulating the behavior of honest parties) interacts with both the dummy adversary $\mathcal{A}$ and the functionality $\mathcal{F}_{\mathsf{PARS}}$.
For clarity and conciseness, we omit explicit mention of communication channel leakages to the dummy adversary $\mathcal{A}$ in the ideal world. The simulator $\mathcal{S}$, which emulates the communication channel functionality $\mathcal{F}_{\mathsf{Ch}}$, ensures that $\mathcal{A}$ receives the same leakage as the real-world adversary through $\mathcal{F}_{\mathsf{Ch}}$.
Regarding communication between users and issuers, the simulator $\mathcal{S}$, which emulates the channel functionality, has knowledge of the user's identity. When a malicious user initiates a transaction, $\mathcal{S}$ verifies the validity of the associated threshold signature.

User registration.
(i) **Honest user U and no more than t malicious issuers.** The simulator receives $(\texttt{Rgs}, \mathsf{sid}, \mathsf{U})$ from $\mathcal{F}_{\mathsf{PARS}}$. $\mathcal{S}$ simulates dummy values as blind signature shares of honest issuers, $\sigma^{\mathsf{blind}}_{\mathsf{hon}}$. Since $\mathcal{S}$ emulates $\mathcal{F}^{\mathsf{S}}_{\mathsf{B}}$, it knows the channel blockages by $\mathcal{A}$ for each issuer and, based on these observations, submits $(\texttt{Rgs.Ok}, \mathsf{sid}, \mathsf{U})$ to the functionality $\mathcal{F}_{\mathsf{PARS}}$.
(ii) **Malicious user U and no more than t malicious issuers.** The simulator initiates a call to $\mathcal{F}_{\mathsf{PARS}}$ with the message $(\texttt{Rgs}, \mathsf{sid})$ to register U, based on the actions of the internally executed $\mathcal{A}$. Additionally, the simulator generates blind signature shares of honest issuers, $\sigma^{\mathsf{blind}}_{\mathsf{hon}}$, and forwards them to the malicious user U while emulating $\mathcal{F}^{\mathsf{S}}_{\mathsf{B}}$. Finally, $\mathcal{S}$ submits $(\texttt{Rgs.Ok}, \mathsf{sid}, \mathsf{U})$ to $\mathcal{F}_{\mathsf{PARS}}$.

Stablecoin issuance.
(i) **Honest user U, and no more than t malicious issuers.** The simulator $\mathcal{S}$ receives $(\texttt{Iss}, \mathsf{sid}, \mathsf{SI.idn})$ from $\mathcal{F}_{\mathsf{PARS}}$. $\mathcal{S}$, which emulates the honest user U, the honest custodian CUS, and the honest issuers, supplies all the information that

the real-world adversary observes in order to internally run the dummy adversary $\mathcal{A}$. This includes information about channel blockage, which $\mathcal{S}$ sends to $\mathcal{A}$ along with the information the malicious issuers receive. Based on its observations regarding the adversary's choices for message delivery, $\mathcal{S}$ submits the message $(\texttt{Iss.Ok}, \mathsf{sid}, \mathsf{SI.idn}, \tilde{\chi})$ to $\mathcal{F}_{\mathsf{PARS}}$, where $\tilde{\chi}$ is simulated. The value $\tilde{\chi}$ encrypts a random group element under the public key $\mathtt{pk}_{\mathsf{A}}$ of the auditor AUD.
(ii) **Malicious user U, and no more than t malicious issuers.** $\mathcal{S}$ emulates the honest custodian CUS and the honest issuers. Upon receiving $(\texttt{Iss}, \mathsf{sid}, \mathsf{SI.idn}, (\mathsf{U}, v))$ from $\mathcal{F}_{\mathsf{PARS}}$, $\mathcal{S}$ begins emulating the honest custodian CUS. After activating $\mathcal{A}$ by emulating CUS, and upon receiving a message from the malicious user U, the simulator $\mathcal{S}$ parses $\mathfrak{I} = (\sigma^{\mathsf{RND}}_{\mathsf{CUS}}, \sigma^{\mathsf{RND}}_{\mathbb{I}}, \mathsf{x}, \pi)$. While emulating $\mathcal{F}_{\mathsf{NIZK}}$, it sends $(\texttt{Verify}, \mathsf{sid}, \mathsf{x}, \pi)$ to $\mathcal{A}$. Upon receiving $(\texttt{Witness}, \mathsf{sid}, \mathsf{w})$ from $\mathcal{A}$, $\mathcal{S}$ checks whether the NIZK relation holds. If it does, $\mathcal{S}$ generates the blind signature shares of the honest issuers, $\sigma^{\mathsf{blind}}_{\mathsf{hon}}$. It then submits these shares $\sigma^{\mathsf{blind}}_{\mathsf{hon}}$ to $\mathcal{A}$ by emulating $\mathcal{F}^{\mathsf{SA}}_{\mathsf{B}}$. $\mathcal{S}$ then parses $\mathsf{x} = (\bar{\varkappa}, \mathsf{sn}, \mathtt{com}, \mathtt{com}_1, \mathtt{com}_2, \mathtt{com}_3, \mathtt{com}_4, h, \varkappa, \mathsf{tag}, \tilde{\chi}_t, \mathtt{pk}_{\mathsf{A}})$ from $\mathfrak{I}$ and submits the message $(\texttt{Iss.Ok}, \mathsf{sid}, \mathsf{SI.idn}, \tilde{\chi})$ to $\mathcal{F}_{\mathsf{PARS}}$, where $\tilde{\chi} = \tilde{\chi}_t$.

Stablecoin transfer.
(i) **Honest sender $\mathsf{U_s}$, and no more than t malicious issuers.** The simulator receives $(\texttt{Gen.ST}, \mathsf{sid}, \mathsf{ST.idn})$ from $\mathcal{F}_{\mathsf{PARS}}$. While emulating the roles of the honest sender $\mathsf{U_s}$ and the honest issuers, $\mathcal{S}$ provides all the information that the real-world adversary observes in order to internally run the adversary $\mathcal{A}$. Based on its observations concerning the adversary's choices for message delivery on the issuers' side, $\mathcal{S}$ sends the message $(\texttt{Gen.ST.Ok}, \mathsf{sid}, \mathsf{ST.idn}, \phi)$ to $\mathcal{F}_{\mathsf{PARS}}$. Here, ϕ consists of three ciphertexts encrypting dummy values under a dummy public key if the receiver $\mathsf{U_r}$ is honest. However, if $\mathsf{U_r}$ is malicious, the value $\mathsf{ST.idn}$ received from $\mathcal{F}_{\mathsf{PARS}}$ is parsed as $(\mathsf{U_s}, \mathsf{U_r}, v)$. In this case, the simulator encrypts v under $\mathtt{pk}_{\mathsf{r}}$ to simulate χ_1, encrypts $\mathtt{pk}_{\mathsf{s}}$ under $\mathtt{pk}_{\mathsf{r}}$ to simulate χ_3, and encrypts g^0 under $\mathtt{pk}_{\mathsf{r}}$ to simulate χ_4. Note that, since $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{KeyReg}}$, the mapping between $(\mathsf{U_s}, \mathsf{U_r})$ and $(\mathtt{pk}_{\mathsf{s}}, \mathtt{pk}_{\mathsf{r}})$ is known to the simulator. Finally, $\mathcal{S}$ sets $\phi \leftarrow (\chi_1, \chi_3, \chi_4)$ and submits $(\texttt{Gen.ST.Ok}, \mathsf{sid}, \mathsf{ST.idn}, \phi)$ to $\mathcal{F}_{\mathsf{PARS}}$.
(ii) **Malicious sender $\mathsf{U_s}$, and no more than t malicious issuers.** Upon receiving the message from the malicious sender $\mathsf{U_s}$, $\mathcal{S}$, while emulating the roles of the honest issuers, parses $\mathfrak{T} = (\sigma^{\mathsf{RND}}_{\mathbb{I}}, \mathsf{x}, \pi)$. While emulating $\mathcal{F}_{\mathsf{NIZK}}$, it sends $(\texttt{Verify}, \mathsf{sid}, \mathsf{x}, \pi)$ to $\mathcal{A}$. Upon receiving $(\texttt{Witness}, \mathsf{sid}, \mathsf{w})$ from $\mathcal{A}$, $\mathcal{S}$ checks whether the associated relation holds. If it does, $\mathcal{S}$ generates the blind signature shares of the honest issuers, $\sigma^{\mathsf{blind}}_{\mathsf{hon}}$. It then submits these shares to $\mathcal{A}$ by emulating $\mathcal{F}^{\mathsf{SA}}_{\mathsf{B}}$. Next, $\mathcal{S}$ parses $\mathsf{x} = (\mathtt{com}, \mathtt{com}_1, \mathtt{com}_2, \mathtt{com}_3, \mathtt{com}_4, h, \varkappa, \mathsf{tag}, \chi_1, \chi_2, \chi_3, \chi_4, W)$. Knowing the extracted witness (i.e., the identity of the receiver $\mathsf{U_r}$, for example, by decrypting χ_2 using knowledge of the discrete logarithm of W to the base g, as well as the transaction value v), $\mathcal{S}$ sends $(\texttt{Gen.ST}, \mathsf{sid}, \mathsf{U_r}, v)$ to $\mathcal{F}_{\mathsf{PARS}}$ on behalf of the malicious sender $\mathsf{U_s}$. Finally, $\mathcal{S}$ submits the message $(\texttt{Gen.ST.Ok}, \mathsf{sid}, \mathsf{ST.idn}, \phi)$ to $\mathcal{F}_{\mathsf{PARS}}$, where $\phi = (\chi_1, \chi_3, \chi_4)$.

Stablecoin claim.

(i) **Honest receiver $\mathsf{U_r}$, and no more than t malicious issuers.** The simulator receives $(\texttt{Clm.ST}, \mathsf{sid}, \mathsf{SC.idn}, \phi)$ from $\mathcal{F}_{\mathsf{PARS}}$. While emulating the roles of the honest receiver $\mathsf{U_r}$ and the honest issuers, $\mathcal{S}$ provides all the information that the real-world adversary observes in order to internally run the adversary $\mathcal{A}$. The simulator then submits $(\texttt{Clm.ST.Ok}, \mathsf{sid}, \mathsf{SC.idn}, \phi)$ to $\mathcal{F}_{\mathsf{PARS}}$. Upon receiving $(\texttt{Clm.ST.Issuer}, \mathsf{sid}, \mathsf{SC.idn}, \mathsf{I}_i, \phi)$ from $\mathcal{F}_{\mathsf{PARS}}$, and based on its observations concerning the adversary's choices for message delivery on the issuers' side, $\mathcal{S}$ sends the message $(\texttt{Clm.ST.Issuer.Ok}, \mathsf{sid}, \mathsf{SC.idn}, \phi)$ to $\mathcal{F}_{\mathsf{PARS}}$. Note that, regardless of whether the sender is malicious or honest, the simulator receives ϕ from $\mathcal{F}_{\mathsf{PARS}}$.
(ii) **Malicious receiver $\mathsf{U_r}$, and no more than t malicious issuers.** Upon receiving the message from the malicious receiver $\mathsf{U_r}$, the simulator, while emulating the roles of the honest issuers, parses $\mathfrak{D} = (\sigma_{\mathbb{I}}^{\mathsf{RND}}, \mathsf{x}, \pi)$. While emulating $\mathcal{F}_{\mathsf{NIZK}}$, $\mathcal{S}$ submits $(\texttt{Verify}, \mathsf{sid}, \mathsf{x}, \pi)$ to $\mathcal{A}$. Upon receiving $(\texttt{Witness}, \mathsf{sid}, \mathsf{w})$ from $\mathcal{A}$, $\mathcal{S}$ checks the correctness of the associated relation. If the relation holds, $\mathcal{S}$ parses $\mathsf{x} = (\mathtt{com}, \mathtt{com}_1, \mathtt{com}_2, \mathtt{com}_3, \mathtt{com}_4, h, \varkappa, \mathsf{tag}, \chi_1, \chi_3, \chi_4)$ and calls $\mathcal{F}_{\mathsf{PARS}}$ with $(\texttt{Clm.ST}, \mathsf{sid}, \phi)$, where $\phi = (\chi_1, \chi_3, \chi_4)$, on behalf of the malicious $\mathsf{U_r}$. Once $\mathcal{S}$ receives $(\texttt{Clm.ST}, \mathsf{sid}, \mathsf{SC.idn}, \phi)$ from $\mathcal{F}_{\mathsf{PARS}}$, it submits $(\texttt{Clm.ST.Ok}, \mathsf{sid}, \mathsf{SC.idn}, \phi)$ to $\mathcal{F}_{\mathsf{PARS}}$. Upon receiving $(\texttt{Clm.ST.Issuer}, \mathsf{sid}, \mathsf{SC.idn}, \mathsf{I}_i, \phi)$ from $\mathcal{F}_{\mathsf{PARS}}$, $\mathcal{S}$ generates the blind signature shares of the honest issuers. $\mathcal{S}$ then submits these signature shares, $\sigma_{\mathsf{hon}}^{\mathsf{blind}}$, to $\mathcal{A}$ by emulating $\mathcal{F}_{\mathsf{Ch}}$. Finally, based on its observations regarding $\mathcal{A}$'s message-blocking behavior, $\mathcal{S}$ submits the message $(\texttt{Clm.ST.Issuer.Ok}, \mathsf{sid}, \mathsf{SC.idn}, \phi)$ to $\mathcal{F}_{\mathsf{PARS}}$.

Stablecoin burn.
(i) **Honest user U, and no more than t malicious issuers.** The simulator receives $(\texttt{Brn}, \mathsf{sid}, \ell, \mathsf{SB.idn})$ from $\mathcal{F}_{\mathsf{PARS}}$. To avoid repetition, we omit describing steps that are similar to those for honest users, as explained earlier in item (i) of previous protocols. $\mathcal{S}$ sends the message $(\texttt{Brn.Ok}, \mathsf{sid}, \mathsf{SB.idn}, \eta, \tilde{\chi})$ to $\mathcal{F}_{\mathsf{PARS}}$, where η consists of two encryptions (of dummy group elements as plaintexts) under a dummy public key, and $\tilde{\chi}$ is an encryption of a dummy group element under $\mathtt{pk}_{\mathsf{A}}$, generated by $\mathcal{S}$.
(ii) **Malicious user U, and no more than t malicious issuers.** Upon receiving the message from the malicious user, the simulator, while emulating the roles of the honest issuers, parses $\mathfrak{B} = (\sigma_{\mathbb{I}}^{\mathsf{RND}}, \mathsf{x}, \pi)$. While emulating $\mathcal{F}_{\mathsf{NIZK}}$, it sends $(\texttt{Verify}, \mathsf{sid}, \mathsf{x}, \pi)$ to $\mathcal{A}$. Upon receiving $(\texttt{Witness}, \mathsf{sid}, \mathsf{w})$ from $\mathcal{A}$, $\mathcal{S}$ checks the validity of the relation and generates the blind signature shares of the honest issuers. It then submits these shares to $\mathcal{A}$ by emulating $\mathcal{F}_{\mathsf{Ch}}$. Given the extracted witness w (where $v \in \mathsf{w}$), $\mathcal{S}$ sends $(\texttt{Brn}, \mathsf{sid}, v, \ell)$ to $\mathcal{F}_{\mathsf{PARS}}$. Finally, $\mathcal{S}$ parses $\mathsf{x} = (\mathtt{com}, \mathtt{com}_1, \mathtt{com}_2, \mathtt{com}_3, \mathtt{com}_4, h, \varkappa, \mathsf{tag}, \chi_1, \chi_2, \tilde{\chi}_t, \mathtt{pk}_{\mathsf{A}}, \ell)$ and submits the message $(\texttt{Brn.Ok}, \mathsf{sid}, \mathsf{SB.idn}, \eta, \tilde{\chi})$ to $\mathcal{F}_{\mathsf{PARS}}$, where $\tilde{\chi} = \tilde{\chi}_t$ and $\eta = (\chi_1, \chi_2)$.

Proof of burn.
(i) **Honest user U.** The simulator receives $(\mathsf{PoB}, \mathsf{sid}, \ell, \mathsf{PB.idn})$ from $\mathcal{F}_{\mathsf{PARS}}$ and leaks to $\mathcal{A}$ whatever the real-world adversary observes as the leakage of $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{SSA}}$. Based on its observations regarding channel blockage by $\mathcal{A}$, $\mathcal{S}$ sends the message $(\mathsf{PoB.Ok}, \mathsf{sid}, \mathsf{PB.idn})$ to $\mathcal{F}_{\mathsf{PARS}}$. Similarly, upon receiving $(\mathsf{PoB.CUS}, \mathsf{sid}, \mathsf{PB.idn})$ from $\mathcal{F}_{\mathsf{PARS}}$, $\mathcal{S}$ submits $(\mathsf{PoB.CUS.Ok}, \mathsf{sid}, \mathsf{PB.idn})$ back to $\mathcal{F}_{\mathsf{PARS}}$.

(ii) **Malicious user** U**.** Upon receiving the message from the malicious user, the simulator $\mathcal{S}$ begins simulating the honest custodian CUS. $\mathcal{S}$ parses $\mathfrak{V} = (\mathsf{x}, \pi)$, where $\mathsf{x} = (\eta, v, \ell)$. While emulating $\mathcal{F}_{\mathsf{NIZK}}$, $\mathcal{S}$ verifies the correctness of π by extracting the witness. Next, $\mathcal{S}$ calls $\mathcal{F}_{\mathsf{PARS}}$ with $(\mathtt{PoB}, \mathsf{sid}, \eta, \ell)$ and receives $(\mathtt{PoB}, \mathsf{sid}, \ell, \mathsf{PB.idn})$ in response. Upon receiving $(\mathtt{PoB.CUS.}, \mathsf{sid}, \mathsf{PB.idn})$ from $\mathcal{F}_{\mathsf{PARS}}$, $\mathcal{S}$ continues to emulate the honest custodian CUS and submits $(\mathtt{PoB.CUS.Ok}, \mathsf{sid}, \mathsf{PB.idn})$ back to $\mathcal{F}_{\mathsf{PARS}}$.

Reserve audit. Upon receiving $(\mathtt{Audit.End}, \mathsf{sid}, b)$ from $\mathcal{F}_{\mathsf{PARS}}$ via public delayed output, $\mathcal{S}$ confirms the delivery of $(\mathtt{Audit.End}, \mathsf{sid}, b)$ to the dummy auditor AUD by submitting its approval to $\mathcal{F}_{\mathsf{PARS}}$.

We prove the statistical proximity between the random variables $\mathsf{EXEC}_{\Pi_{\mathsf{PARS}}, \mathcal{A}, \mathcal{Z}}$ and $\mathsf{EXEC}_{\mathcal{F}_{\mathsf{PARS}}, \mathcal{S}, \mathcal{Z}}$ through a series of games as follows. Each game, $\mathsf{Game}^{(i)}$, is associated with its own functionality $\mathcal{F}^{(i)}_{\mathsf{PARS}}$ and simulator $\mathcal{S}^{(i)}$. Our progression starts with the most information-leaking functionality $\mathcal{F}^{(0)}_{\mathsf{PARS}}$ and its corresponding simulator $\mathcal{S}^{(0)}$. Gradually, we move towards our primary functionality $\mathcal{F}_{\mathsf{PARS}}$ and the primary simulator $\mathcal{S}$. We represent the probability of the environment $\mathcal{Z}$ outputting 1 in $\mathsf{Game}^{(i)}$ as $\Pr[\mathsf{Game}^{(i)}]$.

6.2 Sequence of games and reductions

We introduce two polynomials, max_1 as the upper limit on the total number of ciphertexts across all honest users, and max_2 as the upper bound on the total number of honest users.

6.2.1 Summary of games. This paper introduces seven distinct games $\mathsf{Game}^{(0)}, \ldots, \mathsf{Game}^{(6)}$ where $\mathsf{Game}^{(0)}$ corresponds to the real-world execution $\mathsf{EXEC}_{\Pi_{\mathsf{PARS}}, \mathcal{A}, \mathcal{Z}}$, and $\mathsf{Game}^{(6)}$ corresponds to the ideal-world execution $\mathsf{EXEC}_{\mathcal{F}_{\mathsf{PARS}}, \mathcal{S}, \mathcal{Z}}$.

- In $\mathsf{Game}^{(1)}$, $\mathcal{F}^{(1)}_{\mathsf{PARS}}$ prohibits $\mathcal{S}^{(1)}$ from submitting any message to $\mathcal{F}^{(1)}_{\mathsf{PARS}}$ on behalf of adversary $\mathcal{A}$ who is providing two different messages with the same associated commitment(s). It is argued that under the binding property (definition 10) of the underlying Pedersen commitment scheme (section 11):

$$|\Pr[\mathsf{Game}^{(1)}] - \Pr[\mathsf{Game}^{(0)}]| \leq \mathsf{Adv}^{\mathsf{Bind\text{-}com}}_{\mathcal{A}}$$

- In $\mathsf{Game}^{(2)}$, all plaintexts[12] generated by honest users are changed to random group elements selected by $\mathcal{S}^{(2)}$. It is argued that under $\mathsf{IND\text{-}CPA}$ property (definition 5) of ElGamal encryption scheme (section 6):

$$|\Pr[\mathsf{Game}^{(2)}] - \Pr[\mathsf{Game}^{(1)}]| \leq \mathsf{max}_1 \cdot \mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathcal{A}}$$

[12] g_1^v in *stablecoin issuance*, $(g_1^v, \mathtt{pk_r}, \mathtt{pk_s}, 1)$ in *stablecoin transfer*, $(g_1^v, 1, g_1^v)$ in *stablecoin burn* protocols of the associated ciphertexts: $(\tilde{\chi}_t, \chi_1, \chi_2, \chi_3, \chi_4, \chi_1^*, \chi_2^*, \tilde{\chi}_t^*)$ respectively (here for distinguishing them we use *, however, in the protocol description $(\chi_1^*, \chi_2^*, \tilde{\chi}_t^*)$ are $(\chi_1, \chi_2, \tilde{\chi}_t)$).

- In $\mathsf{Game}^{(3)}$, all tag values of honest users are changed to random group elements selected by $\mathcal{S}^{(3)}$. It is claimed that under the hardness of d-strong Diffie-Hellman problem (definition 2):

$$|\Pr[\mathsf{Game}^{(3)}] - \Pr[\mathsf{Game}^{(2)}]| \leq \mathsf{max}_2 \cdot \mathsf{Adv}_{\mathcal{A}}^{\mathsf{d\text{-}SDDH}}$$

- In $\mathsf{Game}^{(4)}$, all blind signature shares of honest issuers (on malicious user's account) are simulated by $\mathcal{S}^{(4)}$. It is argued that:

$$|\Pr[\mathsf{Game}^{(4)}] = \Pr[\mathsf{Game}^{(3)}]|$$

- In $\mathsf{Game}^{(5)}$, $\mathcal{F}_{\mathsf{PARS}}^{(5)}$ prohibits $\mathcal{S}^{(5)}$ from submitting any message to $\mathcal{F}_{\mathsf{PARS}}^{(5)}$ on behalf of adversary $\mathcal{A}$ who is generating a valid signature for an honest user. It is argued that under the unforgeability property (definition 13) of Pointcheval-Sanders signature (definition 14):

$$|\Pr[\mathsf{Game}^{(5)}] - \Pr[\mathsf{Game}^{(4)}]| \leq \mathsf{Adv}_{\mathcal{A}}^{\mathsf{EUF\text{-}CMA}}$$

- In $\mathsf{Game}^{(6)}$, the aggregated signature of issuers (on an honest user's account) is randomized by $\mathcal{S}^{(6)}$ for all honest users. It is claimed that:

$$|\Pr[\mathsf{Game}^{(6)}] = \Pr[\mathsf{Game}^{(5)}]|$$

We will conclude the proof by showing that any PPT environment $\mathcal{Z}$ can distinguish $\mathsf{EXEC}_{\Pi_{\mathsf{PARS}},\mathcal{A},\mathcal{Z}}$ from $\mathsf{EXEC}_{\mathcal{F}_{\mathsf{PARS}},\mathcal{S},\mathcal{Z}}$ with a probability which is upper bounded by

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Bind\text{-}com}} + \mathsf{max}_1 \cdot \mathsf{Adv}_{\mathcal{A}}^{\mathsf{IND\text{-}CPA}} + \mathsf{max}_2 \cdot \mathsf{Adv}_{\mathcal{A}}^{\mathsf{d\text{-}SDDH}} + \mathsf{Adv}_{\mathcal{A}}^{\mathsf{EUF\text{-}CMA}}$$

6.2.2 Details of games and reductions. $\mathsf{Game}^{(0)}$: At the outset, $\mathcal{F}_{\mathsf{PARS}}^{(0)}$ relays all communication with $\mathcal{Z}$. And the simulator $\mathcal{S}^{(0)}$ emulates the execution of the real-world protocol $\mathsf{EXEC}_{\Pi_{\mathsf{PARS}},\mathcal{A},\mathcal{Z}}$.
$\mathsf{Game}^{(1)}$: Same as $\mathsf{Game}^{(0)}$ except that $\mathsf{Game}^{(1)}$ checks whether a flag is raised or not. $\mathcal{A}$ provides two commitments $\mathtt{com}$ and $\mathtt{com}'$ where $\mathtt{com} = \mathtt{com}'$ with the associated poof-statements (x, π), and (x', π'). $\mathcal{S}^{(1)}$ who emulates $\mathcal{F}_{\mathsf{NIZK}}$ submits $(\mathtt{Verify}, \mathsf{sid}, \mathsf{x}, \pi)$ and $(\mathtt{Verify}, \mathsf{sid}, \mathsf{x}', \pi')$ to $\mathcal{A}$ and receives $(\mathtt{Witness}, \mathsf{sid}, \mathsf{w})$ and $(\mathtt{Witness}, \mathsf{sid}, \mathsf{w}')$ respectively. The flag is raised when with the given extracted witnesses, the committed values are different. Therefore, any difference between $\mathsf{Game}^{(1)}$ and $\mathsf{Game}^{(0)}$ is due to breaking the binding property of the underlying Pedersen commitment scheme (we refrain from providing a formal proof for this), which enables us to bound the probability that $\mathcal{Z}$ distinguishes $\mathsf{Game}^{(1)}$ from $\mathsf{Game}^{(0)}$ as follows.

$$|\Pr[\mathsf{Game}^{(1)}] - \Pr[\mathsf{Game}^{(0)}]| \leq \mathsf{Adv}_{\mathcal{A}}^{\mathsf{Bind\text{-}com}}$$

$\mathsf{Game}^{(2)}$: Same as $\mathsf{Game}^{(1)}$ except that in $\mathsf{Game}^{(2)}$ we change all plaintexts generated by honest users to random (dummy) group elements. Thus, $\mathsf{Game}^{(2)}$ is

identical to $\mathsf{Game}^{(1)}$ except for the fact that $\mathcal{S}^{(2)}$ selects random group elements as plaintexts for all honest users' ciphertexts. Any disparity between $\mathsf{Game}^{(2)}$ and $\mathsf{Game}^{(1)}$ arises from a violation of the IND-CPA security of encryption used in our construction. This allows us to constrain the probability that $\mathcal{Z}$ distinguishes $\mathsf{Game}^{(2)}$ from $\mathsf{Game}^{(1)}$ as follows. We introduce a sequences of sub-games:

$$(\mathsf{Game}_1^{(1)} = \mathsf{Game}^{(1)}, \ldots, \mathsf{Game}_{i-1}^{(1)}, \mathsf{Game}_i^{(1)}, \ldots, \mathsf{Game}_{\mathsf{max}_1}^{(1)} = \mathsf{Game}^{(2)})$$

Additionally, let's define $\mathsf{Game}_2^{(1)}$ as a game analogous to $\mathsf{Game}_1^{(1)}$ with the exception that in $\mathsf{Game}_2^{(1)}$ we replace the plaintext of the first ciphertext of the first honest user from its real-world value to an ideal-world random group element. The transition from $\mathsf{Game}_{i-1}^{(1)}$ to $\mathsf{Game}_i^{(1)}$ is akin to the reduction described below, ensuring that any difference between $\mathsf{Game}_{i-1}^{(1)}$ and $\mathsf{Game}_i^{(1)}$ is upper bounded by $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{IND\text{-}CPA}}$. Finally, we repeat the same procedure for the last ciphertext of the last honest user, such that in $\mathsf{Game}_{\mathsf{max}_1}^{(1)} = \mathsf{Game}^{(2)}$ all ciphertexts are generated from random group elements by $\mathcal{S}_{\mathsf{max}_1}^{(1)} = \mathcal{S}^{(2)}$.

We establish an IND-CPA reduction between $\mathsf{Game}_{i-1}^{(1)}$ and $\mathsf{Game}_i^{(1)}$ (for $2 \leq i \leq \mathsf{max}_1$) as follows. If $\mathcal{Z}$ can distinguish between $\mathsf{Game}_{i-1}^{(1)}$ and $\mathsf{Game}_i^{(1)}$, we can construct $\mathcal{A}'$ to break the IND-CPA security of ElGamal encryption used in Π_{PARS}.

For $1 \leq j \leq i-2$ all (real-world) plaintexts have already been substituted with (ideal-world) random values.

For $i+1 \leq j \leq \mathsf{max}_1$ all ciphertexts are created using real-world plaintexts.

The ciphertext used by $\mathcal{S}_{i-1}^{(1)}$ has been generated using i-th real-world plaintext value (associated to $b = 0$ in the CPA game); and the ciphertext used by $\mathcal{S}_i^{(1)}$ has been generated using ideal-world random value (associated to $b = 1$ in the CPA game).

It's important to note that regarding the challenge ciphertext c_b provided by the CPA challenger to distinguisher $\mathcal{A}'$, in case $b = 0$ it is associated with real-world value, and for $b = 1$ it is associated with dummy random (ideal-world) value. For $\mathcal{Z}$, $\mathsf{Game}_i^{(1)}$ is essentially equivalent to running the real-world protocol with a real-world value for an honest user, as opposed to a random group element from $\mathbb{G}$. Therefore, if $\mathcal{Z}$ can distinguish between $\mathsf{Game}_{i-1}^{(1)}$ from $\mathsf{Game}_i^{(1)}$, $\mathcal{A}'$ can exploit this distinction to win the encryption scheme's IND-CPA game. Therefore, under IND-CPA property of ElGamal encryption scheme the following inequality holds:

$$|\Pr[\mathsf{Game}^{(2)}] - \Pr[\mathsf{Game}^{(1)}]| \leq \mathsf{max}_1 \cdot \mathsf{Adv}_{\mathcal{A}}^{\mathsf{IND\text{-}CPA}}$$

$\mathsf{Game}^{(3)}$: Same as $\mathsf{Game}^{(2)}$ except that in $\mathsf{Game}^{(3)}$ we change all tag values of all honest users to random group elements selected from $\mathbb{G}$. Thus, $\mathsf{Game}^{(3)}$ is identical to $\mathsf{Game}^{(2)}$ except for the fact that $\mathcal{S}^{(3)}$ selects random group elements as tag values for all honest users. Any disparity between $\mathsf{Game}^{(3)}$ and $\mathsf{Game}^{(2)}$ arises from a violation of the hardness of d-Strong Diffie-Hellman problem used in Π_{PARS}. This allows us to constrain the probability that $\mathcal{Z}$ distinguishes $\mathsf{Game}^{(3)}$ from $\mathsf{Game}^{(2)}$ as follows. We introduce a sequences of sub-games:

$$(\mathsf{Game}_1^{(2)} = \mathsf{Game}^{(2)}, \ldots, \mathsf{Game}_{i-1}^{(2)}, \mathsf{Game}_i^{(2)}, \ldots, \mathsf{Game}_{\mathsf{max}_2}^{(2)} = \mathsf{Game}^{(3)})$$

Additionally, let's define $\mathsf{Game}_2^{(2)}$ as a game analogous to $\mathsf{Game}_1^{(2)}$ with the exception that in $\mathsf{Game}_2^{(2)}$ we replace all the tag values of the first honest user from its real-world values to an ideal-world random group elements. The transition from $\mathsf{Game}_{i-1}^{(2)}$ to $\mathsf{Game}_i^{(2)}$ is akin to the reduction described below, ensuring that any difference between $\mathsf{Game}_{i-1}^{(2)}$ and $\mathsf{Game}_i^{(2)}$ is upper bounded by $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{d\text{-}SDDH}}$. Finally, we repeat the same procedure for the last honest users' tag values, such that in $\mathsf{Game}_{\mathsf{max}_2}^{(2)} = \mathsf{Game}^{(3)}$ all tags are generated from random group elements by $\mathcal{S}_{\mathsf{max}_2}^{(2)} = \mathcal{S}^{(3)}$.

We establish a reduction to d-Strong Diffie-Hellman problem between $\mathsf{Game}_{i-1}^{(2)}$ and $\mathsf{Game}_i^{(2)}$ (for $2 \leq i \leq \mathsf{max}_2$) as follows. If $\mathcal{Z}$ can distinguish between $\mathsf{Game}_{i-1}^{(2)}$ and $\mathsf{Game}_i^{(2)}$, we can construct $\mathcal{A}'$ to break the d-Strong Diffie-Hellman problem used in Π_{PARS}.

For $1 \leq j \leq i-2$ all (real-world) tags have already been substituted with (ideal-world) random values.

For $i+1 \leq j \leq \mathsf{max}_2$ all tags are created using real-world values.

The tags used by $\mathcal{S}_{i-1}^{(2)}$ have been generated using real-world values (associated to g^{x^k} values in the d-SDDH assumption for different values of k based on the number of tags generated by the honest user whose tags are being substituted); and the tag values used by $\mathcal{S}_i^{(2)}$ have been generated using ideal-world random values (associated to g^{x_k} in the d-SDDH assumption). For $\mathcal{Z}$, $\mathsf{Game}_i^{(2)}$ is essentially equivalent to running the real-world protocol with real-world tag values for an honest user, as opposed to a random group element from $\mathbb{G}$. Therefore, if $\mathcal{Z}$ can distinguish between $\mathsf{Game}_{i-1}^{(2)}$ from $\mathsf{Game}_i^{(2)}$, $\mathcal{A}'$ can exploit this distinction to break the hardness of d-SDDH assumption. Therefore, under the hardness of d-SDDH assumption, the following inequality holds:

$$|\Pr[\mathsf{Game}^{(3)}] - \Pr[\mathsf{Game}^{(2)}]| \leq \mathsf{max}_2 \cdot \mathsf{Adv}_{\mathcal{A}}^{\mathsf{d\text{-}SDDH}}$$

$\mathsf{Game}^{(4)}$: This game is identical to $\mathsf{Game}^{(3)}$ with the exception that in $\mathsf{Game}^{(4)}$, the blind signature share of honest issuers are simulated by the simulator $\mathcal{S}^{(4)}$. To achieve this, in this game, $\mathcal{S}^{(4)}$ computes the non-threshold signature using the signing key of the underlying non-threshold Pointcheval-Sanders signature scheme used in the threshold randomizable-blind signature scheme in Π_{PARS}.

Furthermore, by selecting the secret keys of malicious issuers $\mathsf{sk}_{\mathsf{mal}} = (x_{\mathsf{mal}}, \{y_{\mathsf{mal},\kappa}\}_{\kappa=1}^{4})$, $\mathcal{S}^{(4)}$ computes the associated public keys and blind signature shares of malicious issuers that are of the form $\sigma_{\mathsf{mal}}^{\mathsf{blind}} = (h, h^{x_{\mathsf{mal}}} \cdot \mathtt{com}_1^{y_{\mathsf{mal},1}} \cdot \mathtt{com}_2^{y_{\mathsf{mal},2}} \cdot \mathtt{com}_3^{y_{\mathsf{mal},3}})$.

Additionally, $\mathcal{S}^{(4)}$ employs Lagrange interpolation to compute the public keys of honest issuers using the computed public keys for malicious ones and the public key of the non-threshold signature scheme. The simulator $\mathcal{S}^{(4)}$ proceeds to compute honest issuers' blind signature shares using the following procedure.

When a malicious user, initiates the protocol requesting a signature on its account, $\mathcal{S}^{(4)}$ emulating $\mathcal{F}_{\mathsf{NIZK}}$ submits $(\mathtt{Verify}, \mathsf{sid}, \mathsf{x}, \pi)$ to $\mathcal{A}$. Upon receiving $(\mathtt{Witness}, \mathsf{sid}, \mathsf{w})$ from $\mathcal{A}$, $\mathcal{S}^{(4)}$ parses the witness w to know acc, and the associated random values used for blinding like o_1, o_2, and o_3. $\mathcal{S}^{(4)}$ computes the signature shares of malicious issuers as $\sigma_{\mathsf{mal}} = (h, c \cdot \prod_{\kappa=1}^{4} \beta_{\mathsf{mal},\kappa}^{-o_\kappa}) = (h, h^{x_{\mathsf{mal}}} \cdot \prod_{\kappa=1}^{4} h^{m_\kappa y_{\mathsf{mal},\kappa}})$.
Subsequently, for computing the signature shares of honest issuers $\mathcal{S}^{(4)}$ is supposed to compute s_{hon}. $\mathcal{S}^{(4)}$ does so as follows given the set of malicious issuers $\mathcal{MAL}$: $s_{\mathsf{hon}} = s^{\prod_{k \in \mathcal{MAL}}((k-\mathsf{hon})/k)} \cdot \prod_{\mathsf{mal} \in \mathcal{MAL}} s_{\mathsf{mal}}^{\prod_{k \in \mathcal{MAL} \cup \{0\}, k \neq \mathsf{mal}}((\mathsf{hon}-k)/(\mathsf{mal}-k))}$, hence, $\sigma_{\mathsf{hon}} = (h, s_{\mathsf{hon}})$ is computed. With the extracted witness o_1, o_2, and o_3 in hand, the simulator proceeds to compute blind signature shares of honest issuers using the computed signatures of honest issuers as follows: $\sigma_{\mathsf{hon}}^{\mathsf{blind}} = (h, \prod_{\kappa=1}^{4} s_{\mathsf{hon}} \cdot \beta_{\mathsf{hon},\kappa}^{o_\kappa})$.
Consequently, in this game, $\mathcal{S}^{(4)}$ simulated the blind signature share of the honest issuers. Following $\mathsf{TRB.Sig.Unblinding}$ algorithm, which is executed by the malicious user, the signature is computed as $\sigma_{\mathsf{hon}} = (h, \prod_{\kappa=1}^{4} s_{\mathsf{hon}} \cdot \beta_{\mathsf{hon},\kappa}^{o_\kappa} \cdot \prod_{\kappa=1}^{4} \beta_{\mathsf{hon},\kappa}^{-o_\kappa}) = (h, s_{\mathsf{hon}})$ for which the equation: $e(h, \tilde{\alpha}_{\mathsf{hon}} \cdot \prod_{\kappa=1}^{4} \tilde{\beta}_{\mathsf{hon},\kappa}^{m_\kappa}) = e(s_{\mathsf{hon}}, \tilde{g})$ holds. This implies that the following equation holds:

$$\Pr[\mathsf{Game}^{(4)}] = \Pr[\mathsf{Game}^{(3)}]$$

$\mathsf{Game}^{(5)}$: Same as the previous game except that in $\mathsf{Game}^{(5)}$, $\mathcal{F}_{\mathsf{PARS}}^{(5)}$ prohibits $\mathcal{S}^{(5)}$ from submitting any message to $\mathcal{F}_{\mathsf{PARS}}^{(5)}$ on behalf of adversary $\mathcal{A}$ who forges a valid signature for an honest user. Hence, $\mathsf{Game}^{(5)}$ is identical to $\mathsf{Game}^{(4)}$ except for the fact that it checks whether a flag is raised or not. If $\mathcal{A}$, who has not been issued at least Γ valid signature shares, submits a valid signature, the flag is raised. Therefore, any difference between $\mathsf{Game}^{(5)}$ and $\mathsf{Game}^{(4)}$ is due to the forgery for the $\mathsf{TRB.Sig}$ scheme used in Π_{PARS}, which enables us to bound the probability that $\mathcal{Z}$ distinguishes $\mathsf{Game}^{(5)}$ from $\mathsf{Game}^{(4)}$ as follows.
We establish a reduction to existential unforgeability of Pointcheval-Sanders signature. If $\mathcal{A}$ successfully forges $\mathsf{TRB.Sig}$ scheme, it can be leveraged to create another adversary $\mathcal{A}'$ capable of breaking the unforgeability property of the Pointcheval-Sanders signature. The blind signature shares of honest issuers can be reconstructed using the blind signature shares of malicious issuers, and the non-threshold signature obtained from the challenger of the existential unforgeability game. This reconstruction is achieved through the use of Lagrange interpolation for the other shares; the algorithm is analogous to the previous game with the key difference being that $\mathcal{A}'$ obtains the non-threshold signature from the challenger rather than assuming the non-threshold secret signing key.
Therefore, with access to the non-threshold signature, $\mathcal{A}'$ is capable of simulating the entire view of $\mathcal{A}$, including the signature shares contributed by the honest issuers. This implies that $\mathcal{A}$ cannot forge messages in the threshold setting of our construction unless $\mathcal{A}$ can forge them in the non-threshold setting. Therefore, if $\mathcal{A}$ manages to forge in the real world, it will also forge in this threshold setting. $\mathcal{A}'$ can then utilize this forgery as a means to forge in the non-threshold

scheme. Consequently, TRB.Sig scheme is simulatable, and when combined with the unforgeability property of the Pointcheval-Sanders signature, it establishes the unforgeability of the signature scheme used in the context of our construction. Therefore, in accordance with the unforgeability property of the Pointcheval-Sanders signature, the following inequality holds:

$$|\Pr[\mathsf{Game}^{(5)}] - \Pr[\mathsf{Game}^{(4)}]| \leq \mathsf{Adv}_{\mathcal{A}}^{\mathsf{EUF\text{-}CMA}}$$

$\mathsf{Game}^{(6)}$: In $\mathsf{Game}^{(6)}$, everything is similar to $\mathsf{Game}^{(5)}$ except the fact that the aggregated signature of issuers $\sigma_{\mathbb{I}}$ is randomized $\sigma_{\mathbb{I}}^{\mathsf{RND}}$ by the simulator $\mathcal{S}^{(6)}$ for all honest users. $\sigma_{\mathbb{I}}$ is of the form $\sigma_{\mathbb{I}} = (h, s) = (h, h^x \cdot \prod_{\kappa=1}^{q} \mathtt{com}_{\kappa}^{y_\kappa} \cdot \prod_{\kappa=1}^{q} \beta_{\kappa}^{-o_\kappa}) = (h, h^x \cdot \prod_{\kappa=1}^{q} (g^{o_\kappa} \cdot h^{m_\kappa})^{y_\kappa} \cdot \prod_{\kappa=1}^{q} (g^{y_\kappa})^{-o_\kappa}) = (h, h^{(x+\Sigma_{\kappa=1}^{q}(m_\kappa y_\kappa))})$. Given $\sigma_{\mathbb{I}} = (h, s)$ where $s = h^{(x+\Sigma_{\kappa=1}^{q}(m_\kappa y_\kappa))}$, $\sigma_{\mathbb{I}}^{\mathsf{RND}}$ is computed as $\sigma_{\mathbb{I}}^{\mathsf{RND}} = (h^{r'}, h^{r'r} s^{r'})$. Hence, we have $\sigma_{\mathbb{I}}^{\mathsf{RND}} = (h^{r'}, h^{r'(r+x+\Sigma_{\kappa=1}^{q}(m_\kappa y_\kappa))})$. Let define $r'' = (r + x + \Sigma_{\kappa=1}^{q}(m_\kappa y_\kappa))$ to simplify the equation for $\sigma_{\mathbb{I}}^{\mathsf{RND}}$ so that we have $\sigma_{\mathbb{I}}^{\mathsf{RND}} = (h^{r'}, h^{r'r''})$. On the other hand, the equation that is checked on the issuers' sides is $e(h', \tilde{g}^x \cdot \prod_{\kappa=1}^{q} \tilde{g}^{y_\kappa m_\kappa + r}) = e(s', \tilde{g})$; let's simplify it using r'' defined above so that we have $e(h^{r'}, \tilde{g}^{r''}) = e(h^{r'r''}, \tilde{g})$ where $\varkappa = \tilde{g}^{r''}, h' = h^{r'}$, and $s' = h^{r'r''}$. Finally, $\mathcal{S}^{(6)}$ sets $\sigma_{\mathbb{I}}^{\mathsf{RND}} = (h^{r'}, h^{r'r''})$, and $\varkappa = \tilde{g}^{r''}$; and emulating $\mathcal{F}_{\mathsf{NIZK}}$ generates a valid proof (as $\varkappa = \tilde{g}^{r''}$ is not a valid value that an honest user generates, according to the protocol the user includes $\varkappa$ in its NIZK relation) which concludes the fact that

$$\Pr[\mathsf{Game}^{(6))}] = \Pr[\mathsf{Game}^{(5)}]$$

$\mathcal{F}_{\mathsf{PARS}}^{(6)}$ equals to our main functionality $\mathcal{F}_{\mathsf{PARS}}$ and $\mathcal{S}^{(6)}$ equals to our main simulator $\mathcal{S}$ (see below for $\mathcal{S}$'s description). We started from $\mathsf{Game}^{(1)}$ that corresponds to the real-world execution $\mathsf{EXEC}_{\Pi_{\mathsf{PARS}},\mathcal{A},\mathcal{Z}}$, and we ended up with $\mathsf{Game}^{(6)}$ that corresponds to the ideal-world execution $\mathsf{EXEC}_{\mathcal{F}_{\mathsf{PARS}},\mathcal{S},\mathcal{Z}}$. Hence, the probability for any PPT environment $\mathcal{Z}$ to distinguish $\mathsf{EXEC}_{\Pi_{\mathsf{PARS}},\mathcal{A},\mathcal{Z}}$ from $\mathsf{EXEC}_{\mathcal{F}_{\mathsf{PARS}},\mathcal{S},\mathcal{Z}}$ is upper bounded by:

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Bind\text{-}com}} + \mathsf{max}_1 \cdot \mathsf{Adv}_{\mathcal{A}}^{\mathsf{IND\text{-}CPA}} + \mathsf{max}_2 \cdot \mathsf{Adv}_{\mathcal{A}}^{\mathsf{d\text{-}SDDH}} + \mathsf{Adv}_{\mathcal{A}}^{\mathsf{EUF\text{-}CMA}}$$

This together with the simulator description conclude the security proof.

□

Acknowledgements

This work has been supported by Input Output (iohk.io) through their funding of the University of Edinburgh Blockchain Technology Lab.

References

1. Central bank digital currency—opportunities, challenges and design. Bank of England, Discussion Paper, March (2020)

2. Akinyele, J.A., Garman, C., Miers, I., Pagano, M.W., Rushanan, M., Green, M., Rubin, A.D.: Charm: a framework for rapidly prototyping cryptosystems. Journal of Cryptographic Engineering **3**, 111–128 (2013)
3. Al-Naji, N., Chen, J., Diao, L.: Basis: a price-stable cryptocurrency with an algorithmic central bank. Basis. io (2017)
4. Ampleforth: Ampleforth documents (2019), available at: Link (accessed in 2024).
5. Aztec: Aztec (2019), available at: Link (accessed in 2024).
6. Bains, P., Ismail, A., Melo, F., Sugimoto, N.: Regulating the crypto ecosystem: the case of stablecoins and arrangements. International Monetary Fund (2022)
7. Bellare, M., Rogaway, P.: Random oracles are practical: A paradigm for designing efficient protocols. In: Proceedings of the 1st ACM Conference on Computer and Communications Security. pp. 62–73 (1993)
8. Ben-Sasson, E., Chiesa, A., Garman, C., Green, M., Miers, I., Tromer, E., Virza, M.: Zerocash: Decentralized anonymous payments from bitcoin. In: 2014 IEEE Symposium on Security and Privacy, SP 2014, Berkeley, CA, USA, May 18-21, 2014. pp. 459–474. IEEE Computer Society (2014), available at: https://doi.org/10.1109/SP.2014.36
9. Benabbas, S., Gennaro, R., Vahlis, Y.: Verifiable delegation of computation over large datasets. In: Annual Cryptology Conference. pp. 111–131. Springer (2011)
10. Bindseil, U.: Tiered CBDC and the Financial System (2020), available at: https://www.ecb.europa.eu/pub/pdf/scpwps/ecb.wp2351 c8c18bbd60.en.pdf
11. BIS: Bank of Canada, European Central Bank, Bank of Japan, Sveriges Riksbank, Swiss National Bank, Bank of England, Board of Governors of the Federal Reserve, and Bank for International Settlements. central bank digital currencies: foundational principles and core features, (2020), available at: https://www.bis.org/publ/othp33.htm
12. Boyle, E., Cohen, R., Goel, A.: Breaking the $o(\sqrt{n})$-bit barrier: Byzantine agreement with polylog bits per party. In: Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing. pp. 319–330 (2021)
13. Bünz, B., Agrawal, S., Zamani, M., Boneh, D.: Zether: Towards privacy in a smart contract world. In: International Conference on Financial Cryptography and Data Security. pp. 423–443. Springer (2020)
14. Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., Maxwell, G.: Bulletproofs: Short proofs for confidential transactions and more. In: 2018 IEEE symposium on security and privacy (SP). pp. 315–334. IEEE (2018)
15. Camenisch, J., Drijvers, M., Gagliardoni, T., Lehmann, A., Neven, G.: The wonderful world of global random oracles. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 280–312. Springer (2018)
16. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: Proceedings 42nd IEEE Symposium on Foundations of Computer Science. pp. 136–145. IEEE (2001)
17. CCData: Stablecoins & cbdcs report (2023), available at: Link (accessed in 2024).
18. CoinMarketCap: Top stablecoin tokens by market capitalization (2024), available at: Link (accessed in 2024).
19. Crites, E., Kiayias, A., Sarencheh, A.: SyRA: Sybil-resilient anonymous signatures with applications to decentralized identity. Cryptology ePrint Archive (2024)
20. Dingledine, R., Mathewson, N., Syverson, P.F.: Tor: The second-generation onion router. In: Blaze, M. (ed.) USENIX Security 2004. pp. 303–320. USENIX Association (Aug 2004)

21. ElGamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. IEEE transactions on information theory **31**(4), 469–472 (1985)
22. Ethereum: Whitepaper (2022), available at: Link (accessed in 2024).
23. European Central Bank (ECB): Ecb digital euro consultation ends with record level of public feedback (2021)
24. Fauzi, P., Meiklejohn, S., Mercer, R., Orlandi, C.: Quisquis: A new design for anonymous cryptocurrencies. In: Advances in Cryptology–ASIACRYPT 2019: 25th International Conference on the Theory and Application of Cryptology and Information Security, Kobe, Japan, December 8–12, 2019, Proceedings, Part I 25. pp. 649–678. Springer (2019)
25. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: Conference on the theory and application of cryptographic techniques. pp. 186–194. Springer (1986)
26. Garay, J.A., Katz, J., Kumaresan, R., Zhou, H.S.: Adaptively secure broadcast, revisited. In: Proceedings of the 30th annual ACM SIGACT-SIGOPS symposium on Principles of distributed computing. pp. 179–186 (2011)
27. Gennaro, R., Gentry, C., Parno, B., Raykova, M.: Quadratic span programs and succinct nizks without pcps. In: Advances in Cryptology–EUROCRYPT 2013: 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings 32. pp. 626–645. Springer (2013)
28. Gennaro, R., Jarecki, S., Krawczyk, H., Rabin, T.: Secure distributed key generation for discrete-log based cryptosystems. In: Advances in Cryptology—EUROCRYPT'99: International Conference on the Theory and Application of Cryptographic Techniques Prague, Czech Republic, May 2–6, 1999 Proceedings 18. pp. 295–310. Springer (1999)
29. Goldreich, O., Micali, S., Wigderson, A.: Proofs that yield nothing but their validity and a methodology of cryptographic protocol design. In: Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali, pp. 285–306 (2019)
30. Groth, J.: Simulation-sound nizk proofs for a practical language and constant size group signatures. In: Advances in Cryptology–ASIACRYPT 2006: 12th International Conference on the Theory and Application of Cryptology and Information Security, Shanghai, China, December 3-7, 2006. Proceedings 12. pp. 444–459. Springer (2006)
31. Groth, J.: Short pairing-based non-interactive zero-knowledge arguments. In: Advances in Cryptology-ASIACRYPT 2010: 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings 16. pp. 321–340. Springer (2010)
32. Groth, J., Ostrovsky, R., Sahai, A.: Perfect non-interactive zero knowledge for np. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 339–358. Springer (2006)
33. Guillou, L.C., Quisquater, J.J.: A practical zero-knowledge protocol fitted to security microprocessor minimizing both transmission and memory. In: Advances in Cryptology—EUROCRYPT'88: Workshop on the Theory and Application of Cryptographic Techniques Davos, Switzerland, May 25–27, 1988 Proceedings 7. pp. 123–128. Springer (1988)
34. Kasamatsu, K., Kanno, S., Kobayashi, T., Kawahara, Y.: Barreto-naehrig curves. Network Working Group. Internet-Draft. February (2014)

35. Kiayias, A., Kohlweiss, M., Sarencheh, A.: PEReDi: Privacy-enhanced, regulated and distributed central bank digital currencies. In: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security. pp. 1739–1752 (2022)
36. Kondi, Y., Shelat, A.: Improved straight-line extraction in the random oracle model with applications to signature aggregation. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 279–309. Springer (2022)
37. Kosba, A., Zhao, Z., Miller, A., Qian, Y., Chan, H., Papamanthou, C., Pass, R., Shelat, A., Shi, E.: Cϕcϕ: a framework for building composable zero-knowledge proofs. Cryptology ePrint Archive, Report 2015/1093 (2015)
38. Liao, G.Y., Caramichael, J.: Stablecoins: Growth potential and impact on banking (2022)
39. Lokhava, M., Losa, G., Mazières, D., Hoare, G., Barry, N., Gafni, E., Jove, J., Malinowsky, R., McCaleb, J.: Fast and secure global payments with stellar. In: Proceedings of the 27th ACM Symposium on Operating Systems Principles. pp. 80–96 (2019)
40. Lysyanskaya, A., Rosenbloom, L.N.: Universally composable σ-protocols in the global random-oracle model. In: Theory of Cryptography Conference. pp. 203–233. Springer (2022)
41. MakerDAO: The maker protocol: Makerdao's multi-collateral dai (mcd) system (2020), available at: Link (accessed in 2024).
42. Mankiw, N.G.: Principles of economics (29-1 The Meaning of Money). Cengage Learning (2020)
43. Moin, A., Sekniqi, K., Sirer, E.G.: Sok: A classification framework for stablecoin designs. In: Financial Cryptography and Data Security: 24th International Conference, FC 2020, Kota Kinabalu, Malaysia, February 10–14, 2020 Revised Selected Papers 24. pp. 174–197. Springer (2020)
44. Noether, S., Mackenzie, A., et al.: Ring confidential transactions. Ledger **1**, 1–18 (2016)
45. Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. In: Annual international cryptology conference. pp. 129–140. Springer (1991)
46. Pointcheval, D., Sanders, O.: Short randomizable signatures. Cryptology ePrint Archive, Paper 2015/525 (2015), available at: Link.
47. Q.ai: What really happened to luna crypto? (2022), available at: Link (accessed in 2024).
48. Rial, A., Piotrowska, A.M.: Security analysis of coconut, an attribute-based credential scheme with threshold issuance. Cryptology ePrint Archive (2022)
49. Sarencheh, A., Khoshakhlagh, H., Kavousi, A., Kiayias, A.: DART: Decentralized, anonymous, and regulation-friendly tokenization. Cryptology ePrint Archive (2025)
50. Schnorr, C.P.: Efficient signature generation by smart cards. Journal of cryptology **4**, 161–174 (1991)
51. Sonnino, A., Al-Bassam, M., Bano, S., Meiklejohn, S., Danezis, G.: Coconut: Threshold issuance selective disclosure credentials with applications to distributed ledgers. In: NDSS 2019. The Internet Society (Feb 2019)
52. Synthetix: Whitepaper (2022), available at: Link (accessed in 2024).
53. Tether: Whitepaper (2015), available at: Link (accessed in 2024).
54. US Department of the Treasury: Report on stablecoins. Tech. rep. (November 2021), available at: Link

55. USDCoin: Centre whitepaper (2021), available at: Link (accessed in 2024).
56. Wüst, K., Kostiainen, K., Čapkun, V., Čapkun, S.: Prcash: Fast, private and regulated transactions for digital currencies. In: Financial Cryptography and Data Security: 23rd International Conference, FC 2019, Frigate Bay, St. Kitts and Nevis, February 18–22, 2019, Revised Selected Papers 23. pp. 158–178. Springer (2019)
57. Wüst, K., Kostiainen, K., Delius, N., Capkun, S.: Platypus: a central bank digital currency with unlinkable transactions and privacy-preserving regulation. In: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security. pp. 2947–2960 (2022)

A Cryptographic schemes and security definitions

Within this section, we introduce the cryptographic primitives and definitions that constitute components of our construction Π_{PARScoin}.

Definition 1 (Bilinear maps). *Let $\mathbb{G}$, $\tilde{\mathbb{G}}$, and $\mathbb{G}_t$ denote groups of prime order p. A map $e : \mathbb{G} \times \tilde{\mathbb{G}} \to \mathbb{G}_t$ is called a bilinear map if it satisfies the following conditions:*

- ***Bilinearity**: For all $g \in \mathbb{G}$, $\tilde{g} \in \tilde{\mathbb{G}}$, and $x, y \in \mathbb{Z}_p$, the equality $e(g^x, \tilde{g}^y) = e(g, \tilde{g})^{xy}$ holds.*
- ***Non-degeneracy**: For generators $g \in \mathbb{G}$ and $\tilde{g} \in \tilde{\mathbb{G}}$, the element $e(g, \tilde{g})$ is a generator of $\mathbb{G}_t$.*
- ***Efficiency**: There exists an efficient algorithm $\mathcal{G}(1^k)$ that outputs the setup $(p, \mathbb{G}, \tilde{\mathbb{G}}, \mathbb{G}_t, e, g, \tilde{g})$, and the computation of $e(a, b)$ for any $a \in \mathbb{G}$ and $b \in \tilde{\mathbb{G}}$ is efficient.*

In the case of type 3 pairings, $\mathbb{G} \neq \tilde{\mathbb{G}}$, and no efficiently computable homomorphism between $\mathbb{G}$ and $\tilde{\mathbb{G}}$.

Definition 2 (d-SDDH assumption). *Let $\mathbb{G}$ be a cyclic group of prime order p, where p is a prime, and let g be a generator of $\mathbb{G}$. Suppose $(\mathbb{G}, p, g) \leftarrow \mathcal{G}(1^\lambda)$. For $x, x_1, \ldots, x_d \xleftarrow{\$} \mathbb{Z}_p$, the d-Strong Decisional Diffie-Hellman (d-SDDH) assumption asserts that the following holds relative to $\mathcal{G}$. For any probabilistic polynomial-time (PPT) distinguisher $\mathcal{A}$, there exists a negligible function $\mathsf{ngl}(\cdot)$ such that [9]:*

$$\mathsf{Adv}_{\mathcal{A}}^{d\text{-}\mathsf{SDDH}} = \Big| \Pr\left[\mathcal{A}(\mathbb{G}, p, g, g^x, g^{x^2}, \ldots, g^{x^d}) = 1\right] - \Pr\left[\mathcal{A}(\mathbb{G}, p, g, g^{x_1}, g^{x_2}, \ldots, g^{x_d}) = 1\right] \Big| \leq \mathsf{ngl}_{d\text{-}\mathsf{SDDH}}(\lambda).$$

A.1 Public key encryption (PKE) schemes

Definition 3 (Public key encryption (PKE) scheme). *A public key encryption (PKE) scheme is a tuple of algorithms* $\mathsf{PKE} = (\mathsf{PKE.KeyGen}, \mathsf{PKE.Enc}, \mathsf{PKE.Dec})$ *defined as follows:*

- $\mathsf{PKE.KeyGen}$ ***(Key generation):*** *This randomized algorithm takes a security parameter λ as input and outputs a key pair $(\mathsf{pk}, \mathsf{sk})$, where pk is the public key and sk is the secret (private) key. The public key pk implicitly determines the associated message space $\mathsf{MessageSpace}(\mathsf{pk})$, which defines the set of plaintexts that can be encrypted.*
- $\mathsf{PKE.Enc}$ ***(Encryption):*** *This randomized algorithm takes as input the public key pk and a plaintext message $m \in \mathsf{MessageSpace}(\mathsf{pk})$. It outputs a ciphertext C.*
- $\mathsf{PKE.Dec}$ ***(Decryption):*** *The decryption algorithm takes as input the private key sk and a ciphertext C. It outputs the corresponding plaintext m if decryption succeeds or a special failure symbol $\perp$ if decryption fails.*

Definition 4 (Correctness of a PKE scheme). *A public key encryption (PKE) scheme $\mathsf{PKE} = (\mathsf{PKE.KeyGen}, \mathsf{PKE.Enc}, \mathsf{PKE.Dec})$ is said to be correct if the following holds: for all $\lambda \in \mathbb{N}$, for all key pairs $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{PKE.KeyGen}(1^\lambda)$, and for all messages $m \in \mathsf{MessageSpace}(\mathsf{pk})$,*

$$\mathsf{PKE.Dec}(\mathsf{sk}, \mathsf{PKE.Enc}(\mathsf{pk}, m)) = m$$

with overwhelming probability over the randomness of the algorithms $\mathsf{PKE.KeyGen}$ and $\mathsf{PKE.Enc}$.

Definition 5 ($\mathsf{IND\text{-}CPA}$ security of a PKE scheme). *A public key encryption (PKE) scheme $\mathsf{PKE} = (\mathsf{PKE.KeyGen}, \mathsf{PKE.Enc}, \mathsf{PKE.Dec})$ is said to be $\mathsf{IND\text{-}CPA}$-secure if, for all probabilistic polynomial-time (PPT) adversaries $\mathcal{A}$, the advantage $\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathcal{A}}(\lambda)$ in the following experiment is negligible in the security parameter λ:*

$$\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathcal{A}}(\lambda) \leq \mathsf{negl}(\lambda).$$

The security experiment $\mathsf{IND\text{-}CPA}^b_{\mathsf{PKE}}(\mathcal{A}, \lambda)$ is defined between a PPT adversary $\mathcal{A}$ and a challenger as follows:

1. *The challenger generates a key pair $(\mathsf{pk}, \mathsf{sk})$ by executing $\mathsf{PKE.KeyGen}(1^\lambda)$ and sends the public key pk to the adversary $\mathcal{A}$.*
2. *The adversary $\mathcal{A}$ selects two plaintext messages (m_0, m_1), ensuring $|m_0| = |m_1|$ and $m_0, m_1 \in \mathsf{MessageSpace}(\mathsf{pk})$, and submits them to the challenger.*
3. *The challenger chooses a random bit $b \xleftarrow{\$} \{0,1\}$, encrypts the corresponding message m_b to obtain the ciphertext $C_b = \mathsf{PKE.Enc}(\mathsf{pk}, m_b)$, and sends C_b to $\mathcal{A}$.*
4. *The adversary $\mathcal{A}$ outputs a guess b' for b. If no output is produced, the guess b' is set to 0 by default.*

The adversary's advantage in distinguishing which plaintext was encrypted is given by:

$$\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathcal{A}}(\lambda) = \left| \Pr[\mathsf{IND\text{-}CPA}^1_{\mathsf{PKE}}(\mathcal{A}, \lambda) = 1] - \Pr[\mathsf{IND\text{-}CPA}^0_{\mathsf{PKE}}(\mathcal{A}, \lambda) = 1] \right|.$$

Definition 6 (ElGamal encryption scheme). *The ElGamal encryption scheme [21]* EG *is a public key encryption (PKE) scheme defined over a cyclic group* $\mathbb{G}_q$ *of prime order* q*, with generator* g*. It consists of three algorithms* $\mathsf{EG} = (\mathsf{EG.KeyGen}, \mathsf{EG.Enc}, \mathsf{EG.Dec})$*, described as follows:*

- $\mathsf{EG.KeyGen}$ ***(Key generation):*** *Using a group generation algorithm* $\mathcal{G}$ *that outputs* (q, g)*, where* q *is a prime and* g *is a generator of* $\mathbb{G}_q$*, the key generation is performed as follows:*
  - *Input the security parameter* 1^λ*.*
  - *Sample a secret key* $x \xleftarrow{\$} \mathbb{Z}_q$*, chosen uniformly at random.*
  - *Compute the public key component* $\mathsf{P} = g^x \in \mathbb{G}_q$*.*
  - *Output the public key* $\mathsf{pk} = (q, g, \mathsf{P})$ *and the secret key* $\mathsf{sk} = x$*.*
- $\mathsf{EG.Enc}$ ***(Encryption):*** *To encrypt a message* $M \in \mathbb{G}_q$ *under the public key* $\mathsf{pk} = (q, g, \mathsf{P})$*:*
  - *Sample a random value* $r \xleftarrow{\$} \mathbb{Z}_q$*.*
  - *Compute:* $\mathsf{C}_1 = g^r, \quad \mathsf{B} = \mathsf{P}^r$*.*
  - *Compute the ciphertext as a pair:* $\psi = (\mathsf{C}_1, \mathsf{C}_2), \quad$ *where* $\mathsf{C}_2 = \mathsf{B} \cdot M$*.*
- $\mathsf{EG.Dec}$ ***(Decryption):*** *To decrypt a ciphertext* $\psi = (\mathsf{C}_1, \mathsf{C}_2)$ *using the secret key* $\mathsf{sk} = x$*:*
  - *Compute:* $\mathsf{B} = \mathsf{C}_1^x$*.*
  - *Recover the plaintext* M *as:* $M = \mathsf{C}_2 \cdot \mathsf{B}^{-1}$*.*

The ElGamal encryption scheme achieves chosen-plaintext attack (CPA) security based on the hardness of the Decisional Diffie-Hellman (DDH) problem in $\mathbb{G}_q$*.*

A.2 Commitment schemes

Definition 7 (Commitment scheme). *A commitment scheme* $\mathsf{COM} = (\mathsf{COM.KeyGen}, \mathsf{COM.Commit}, \mathsf{COM.Verify})$ *consists of the following algorithms:*

- $\mathsf{COM.KeyGen}$ ***(Key generation):*** *This probabilistic algorithm takes a security parameter* λ *as input and generates a pair of keys* $(\mathsf{ck}, \mathsf{vk})$*, where* ck *is the commitment key used by the committer, and* vk *is the verification key used by the verifier. In publicly verifiable schemes, the keys are identical, i.e.,* $\mathsf{ck} = \mathsf{vk}$*.*
- $\mathsf{COM.Commit}$ ***(Commitment):*** *The commitment algorithm takes as input the commitment key* ck *and a message* $m \in \mathsf{MessageSpace}(\mathsf{ck})$*. It outputs a commitment* c *and an opening value* d*:* $(c, d) \leftarrow \mathsf{COM.Commit}(\mathsf{ck}, m)$*. The pair* (c, d) *allows the committer to prove the committed message* m *during the verification phase.*
- $\mathsf{COM.Verify}$ ***(Verification):*** *The verification algorithm takes as input the verification key* vk*, a message* m*, a commitment* c*, and an opening value* d*. It outputs* accept *if* (c, d) *corresponds to a valid commitment to* m*, and* reject *otherwise. Formally:*

$$\mathsf{COM.Verify}(\mathsf{vk}, c, m, d) = \begin{cases} \mathsf{accept}, & \text{if } (c, d) \text{ is a valid opening for } m, \\ \mathsf{reject}, & \text{otherwise.} \end{cases}$$

Definition 8 (Commitment correctness). *A commitment scheme* $\mathsf{COM} = (\mathsf{COM.KeyGen}, \mathsf{COM.Commit}, \mathsf{COM.Verify})$ *satisfies correctness if, for all security parameters* $\lambda \in \mathbb{N}$*, for all key pairs* $(\mathsf{ck}, \mathsf{vk}) \leftarrow \mathsf{COM.KeyGen}(1^\lambda)$*, and for all valid messages* $m \in \mathsf{MessageSpace}(\mathsf{ck})$*, the following holds with probability 1 over the randomness of* COM.Commit*:*

$$\mathsf{COM.Verify}(\mathsf{vk}, c, m, d) = \mathsf{accept},$$

where $(c, d) \leftarrow \mathsf{COM.Commit}(\mathsf{ck}, m)$.

Definition 9 (Commitment hiding). *Let* $\mathsf{COM} = (\mathsf{COM.KeyGen}, \mathsf{COM.Commit}, \mathsf{COM.Verify})$ *denote a (publicly verifiable) commitment scheme. The* ***hiding*** *property ensures that no probabilistic polynomial-time (PPT) adversary* $\mathcal{A}$ *can distinguish between commitments to two messages of the same length. The hiding security is formally defined as follows.*
The experiment $\mathsf{Hide}^b_{\mathsf{COM}}(\mathcal{A}, \lambda)$ *between a PPT adversary* $\mathcal{A}$ *and a challenger proceeds as follows:*

1. *The challenger generates a commitment key* $\mathsf{ck} = \mathsf{vk}$ *by running* $\mathsf{COM.KeyGen}(1^\lambda)$ *and sends* ck *to the adversary* $\mathcal{A}$.
2. *The adversary* $\mathcal{A}$ *submits two messages* (m_0, m_1) *such that* $|m_0| = |m_1|$ *and* $m_0, m_1 \in \mathsf{MessageSpace}(\mathsf{ck})$.
3. *The challenger selects a random bit* $b \xleftarrow{\$} \{0, 1\}$*, computes the commitment* $(c, d) \leftarrow \mathsf{COM.Commit}(\mathsf{ck}, m_b)$*, and sends the commitment* c *to the adversary* $\mathcal{A}$.
4. *The adversary* $\mathcal{A}$ *outputs a guess* b' *for* b.

The advantage of $\mathcal{A}$ *in this experiment is defined as:*

$$\mathsf{Adv}^{\mathsf{Hide}}_{\mathcal{A}}(\lambda) = \left| \Pr[\mathsf{Hide}^1_{\mathsf{COM}}(\mathcal{A}, \lambda) = 1] - \Pr[\mathsf{Hide}^0_{\mathsf{COM}}(\mathcal{A}, \lambda) = 1] \right|.$$

A commitment scheme satisfies the hiding property under the following conditions:

- ***Perfect hiding:*** *For all adversaries* $\mathcal{A}$*,* $\mathsf{Adv}^{\mathsf{Hide}}_{\mathcal{A}}(\lambda) = 0$.
- ***Computational hiding:*** *For all PPT adversaries* $\mathcal{A}$*,* $\mathsf{Adv}^{\mathsf{Hide}}_{\mathcal{A}}(\lambda) \leq \mathsf{negl}(\lambda)$*, where* $\mathsf{negl}(\lambda)$ *is a negligible function in* λ.

Hiding ensures that no adversary can distinguish between commitments to m_0 *and* m_1 *with probability significantly better than random guessing. This guarantees that the commitment conceals the underlying message.*

Definition 10 (Commitment binding). *Let* $\mathsf{COM} = (\mathsf{COM.KeyGen}, \mathsf{COM.Commit}, \mathsf{COM.Verify})$ *denote a (publicly verifiable) commitment scheme. The* ***binding*** *property ensures that no probabilistic polynomial-time (PPT) adversary* $\mathcal{A}$ *can produce a commitment* c *that can be opened to two distinct messages. The binding security is formally defined as follows.*
The experiment $\mathsf{Bind}_{\mathsf{COM}}(\mathcal{A}, \lambda)$ *between a PPT adversary* $\mathcal{A}$ *and a challenger proceeds as follows:*

1. *The challenger generates a commitment key* $\mathsf{ck} = \mathsf{vk}$ *by running* $\mathsf{COM.KeyGen}(1^\lambda)$ *and sends* ck *to the adversary* $\mathcal{A}$.
2. *The adversary* $\mathcal{A}$ *outputs a tuple* (c, m, d, m', d'), *where* $m, m' \in \mathsf{MessageSpace}(\mathsf{ck})$.
3. *The challenger checks the following conditions:*

$$\mathsf{COM.Verify}(\mathsf{vk}, c, m, d) = \mathsf{accept} \quad \textit{and} \quad \mathsf{COM.Verify}(\mathsf{vk}, c, m', d') = \mathsf{accept}.$$

4. *If both verifications succeed and* $m \neq m'$, *the adversary* $\mathcal{A}$ *wins the binding game. Otherwise, the adversary fails.*

The adversary's advantage in the binding experiment is defined as:

$$\mathsf{Adv}^{\mathsf{Bind}}_{\mathcal{A}}(\lambda) = \mathsf{Pr}[\mathsf{Bind}_{\mathsf{COM}}(\mathcal{A}, \lambda) = 1].$$

A commitment scheme satisfies the binding property under the following conditions:

- ***Perfect binding:*** *For all adversaries* $\mathcal{A}$, $\mathsf{Adv}^{\mathsf{Bind}}_{\mathcal{A}}(\lambda) = 0$.
- ***Computational binding:*** *For all PPT adversaries* $\mathcal{A}$, $\mathsf{Adv}^{\mathsf{Bind}}_{\mathcal{A}}(\lambda) \leq \mathsf{negl}(\lambda)$, *where* $\mathsf{negl}(\lambda)$ *is a negligible function in* λ.

Binding ensures that no adversary can produce a valid commitment c *that can be opened to two distinct messages* $m \neq m'$ *with non-negligible probability. This guarantees that the commitment binds the committer to a single value.*

Definition 11 (Pedersen commitment scheme). *The Pedersen commitment scheme [45]* $\mathsf{PCOM} = (\mathsf{PCOM.KeyGen}, \mathsf{PCOM.Commit}, \mathsf{PCOM.Verify})$ *is a cryptographic commitment scheme with the following components:*

- $\mathsf{PCOM.KeyGen}$ ***(Key generation):*** *This probabilistic algorithm takes a security parameter* λ *as input and generates cyclic group parameters* (G, q, g), *where* G *is a cyclic group of prime order* q, *and* g *is a generator of* G. *It also selects a random group element* $h \xleftarrow{\$} G$. *The commitment key and verification key are set as* $\mathsf{ck} = \mathsf{vk} = (G, q, g, h)$.
- $\mathsf{PCOM.Commit}$ ***(Commitment):*** *The commitment algorithm takes the commitment key* $\mathsf{ck} = (G, q, g, h)$ *and a message* $m \in \mathbb{Z}_q$ *as input. It generates a random opening value* $d \xleftarrow{\$} \mathbb{Z}_q$ *and computes the commitment:*

$$c = g^d \cdot h^m$$

  The output of the algorithm is the pair (c, d), *denoted as:*

$$(c, d) \leftarrow \mathsf{PCOM.Commit}(\mathsf{ck}, m)$$

- $\mathsf{PCOM.Verify}$ ***(Verification):*** *The verification algorithm takes as input the verification key* $\mathsf{vk} = (G, q, g, h)$, *a message* $m \in \mathbb{Z}_q$, *an opening value* $d \in \mathbb{Z}_q$,

and a commitment $c \in G$. It checks whether the commitment was correctly generated by verifying the equation:

$$g^d \cdot h^m = c.$$

The algorithm outputs accept *if the equality holds, and* reject *otherwise:*

$$\mathsf{PCOM.Verify}(\mathsf{vk}, c, m, d) = \begin{cases} \mathsf{accept}, & \text{if } g^d \cdot h^m = c, \\ \mathsf{reject}, & \text{otherwise.} \end{cases}$$

The Pedersen commitment scheme PCOM *satisfies the following security properties:*

- ***Perfect hiding:*** *The commitment value c reveals no information about the message m, even with unbounded computational resources.*
- ***Computational binding:*** *It is computationally infeasible for any adversary to find two distinct messages $m, m' \in \mathbb{Z}_q$ and corresponding openings $d, d' \in \mathbb{Z}_q$ such that:*

$$g^d \cdot h^m = g^{d'} \cdot h^{m'}$$

A.3 Digital signatures

Definition 12 (Digital signature schemes). *A digital signature scheme is a tuple $\Gamma = (\mathcal{K}, \mathcal{S}, \mathcal{V})$ consisting of the following components:*

- *$\mathcal{K}$* ***(Key generation):*** *A probabilistic algorithm that takes a security parameter λ as input and generates a key pair $(\mathsf{sk}, \mathsf{vk})$, where:*
  - *sk is the private signing key.*
  - *vk is the public verification key.*
- *$\mathcal{S}$* ***(Signature generation):*** *A (potentially probabilistic) algorithm that, given the private key sk and a message μ from the message space $\mathcal{M}$, produces a signature $\sigma \in \Xi$. This process is denoted as:*

$$\sigma \leftarrow \mathcal{S}_{\mathsf{sk}}(\mu).$$

- *$\mathcal{V}$* ***(Signature verification):*** *A deterministic algorithm that, given the public verification key vk, a message $\mu \in \mathcal{M}$, and a signature $\sigma' \in \Xi$, determines whether σ' is valid for μ under vk. It outputs a binary decision $\beta \in \{0, 1\}$, where:*
  - *$\beta = 1$ indicates that σ' is a valid signature for μ.*
  - *$\beta = 0$ indicates that σ' is invalid.*

  This is denoted as:

$$\beta \leftarrow \mathcal{V}_{\mathsf{vk}}(\mu, \sigma').$$

Correctness: *A digital signature scheme is correct if, for all key pairs $(\mathsf{sk}, \mathsf{vk})$ generated by $\mathcal{K}$ and all messages $\mu \in \mathcal{M}$, it holds that:*

$$\mathcal{V}_{\mathsf{vk}}(\mu, \mathcal{S}_{\mathsf{sk}}(\mu)) = 1.$$

This ensures that any validly signed message μ will always be accepted by the verification algorithm under the corresponding public key vk.

Definition 13 (EU-CMA security). *A digital signature scheme $\Gamma = (\mathcal{K}, \mathcal{S}, \mathcal{V})$ is said to be existentially unforgeable under a chosen message attack (*EU-CMA*) if, for every probabilistic polynomial-time (PPT) adversary $\mathcal{A}$, the success probability $\mathrm{Adv}_{\mathcal{A}}^{\text{EU-CMA}}(\lambda)$ in the following experiment is negligible in the security parameter λ:*

$$\mathrm{Adv}_{\mathcal{A}}^{\text{EU-CMA}}(\lambda) \leq \mathrm{negl}(\lambda).$$

The experiment $\mathsf{Exp}_{\Gamma}^{\text{EU-CMA}}(\mathcal{A}, \lambda)$ is defined as follows:

1. *The challenger runs the key generation algorithm $\mathcal{K}(1^\lambda)$ to produce a key pair $(\mathsf{vk}, \mathsf{sk})$ and sends the public key vk to the adversary $\mathcal{A}$.*
2. *The adversary $\mathcal{A}$ can query a signing oracle $\mathcal{S}_{\mathsf{sk}}(\cdot)$ on any message $\mu \in \mathcal{M}$. For each query, the oracle computes the signature $\sigma = \mathcal{S}_{\mathsf{sk}}(\mu)$ and returns it to $\mathcal{A}$. All queried messages are recorded in a set $\mathcal{Q}$, referred to as the query history.*
3. *Eventually, the adversary $\mathcal{A}$ outputs a forgery attempt, consisting of a message-signature pair (μ^*, σ^*), where $\mu^* \in \mathcal{M}$ and $\sigma^* \in \Xi$.*
4. *The adversary $\mathcal{A}$ wins the experiment if both of the following conditions hold:*
   - *$\mu^* \notin \mathcal{Q}$ (i.e., the forged message μ^* was not previously queried to the signing oracle), and*
   - *$\mathcal{V}_{\mathsf{vk}}(\mu^*, \sigma^*) = 1$ (i.e., σ^* is a valid signature for μ^* under the public key vk).*

The advantage of $\mathcal{A}$ in the EU-CMA *experiment is defined as:*

$$\mathrm{Adv}_{\mathcal{A}}^{\text{EU-CMA}}(\lambda) = \Pr[\mathsf{Exp}_{\Gamma}^{\text{EU-CMA}}(\mathcal{A}, \lambda) = 1]$$

A.3.1 Randomizable-blind signature The Coconut framework, as described in [51], introduces an optional declaration credential construction supporting distributed threshold issuance. This framework relies on the Pointcheval-Sanders (PS) signature scheme [46], which is existentially unforgeable and randomizable. The PS signature scheme is defined as follows:

Definition 14 (Pointcheval-Sanders signature). *The Pointcheval-Sanders signature scheme [46] consists of the following algorithms:*

- $\mathsf{KeyGen}(1^\lambda, q)$*: Execute the pairing group setup algorithm $\mathcal{G}(1^\lambda)$, which outputs the group parameters:*

  $$\mathsf{par} = (p, \mathbb{G}, \tilde{\mathbb{G}}, \mathbb{G}_t, e, g, \tilde{g}),$$

  where $e : \mathbb{G} \times \tilde{\mathbb{G}} \to \mathbb{G}_t$ is a bilinear map, p is a prime order, and $g \in \mathbb{G}$, $\tilde{g} \in \tilde{\mathbb{G}}$ are generators. The secret key is selected as:

  $$\mathsf{sk} = (x, \{y_i\}_{i=1}^q) \xleftarrow{\$} \mathbb{Z}_p^{q+1}.$$

  The public key is:

  $$\mathsf{pk} = (\mathsf{par}, \alpha, \{\beta_i\}_{i=1}^q) \leftarrow (\mathsf{par}, \tilde{g}^x, \{\tilde{g}^{y_i}\}_{i=1}^q).$$

- $\mathsf{Sign}(\mathsf{sk}, \{m_i\}_{i=1}^q)$*: To sign a message vector* $\{m_i\}_{i=1}^q \in \mathbb{Z}_p^q$*:*
  - *Select a random value* $r \xleftarrow{\$} \mathbb{Z}_p$*.*
  - *Compute* $h \leftarrow g^r$*.*
  - *Compute the signature:*

$$\sigma = (h, s) \leftarrow \left(h, h^{(x+\sum_{i=1}^q y_i m_i)}\right).$$

  The output is the signature $\sigma = (h, s)$*.*
- $\mathsf{VerifySig}(\mathsf{pk}, \sigma, \{m_i\}_{i=1}^q)$*: To verify a signature* $\sigma = (h, s)$ *on message vector* $\{m_i\}_{i=1}^q$*, check the following:*
  - *Ensure* $h \neq 1$*.*
  - *Verify:*

$$e(h, \alpha \cdot \prod_{i=1}^{q} \beta_i^{m_i}) = e(s, \tilde{g}).$$

  If both conditions hold, output 1 *(accept); otherwise, output* 0 *(reject).*

Within the Coconut framework, it is possible to achieve unlinkable optional attribute disclosures and the management of both public and private attributes, even in scenarios where some issuing authorities may exhibit malicious behavior or be offline.

A recent analysis of Coconut's security properties was conducted by Rial et al. [48], where they introduced an ideal functionality capturing the essential security characteristics of a threshold randomizable-blind signature scheme. Rial et al. [48] proposed a modified construction based on Coconut, designed to realize the ideal functionality.

In our construction, similar to PEReDi [35], we leverage an enhanced iteration of the Coconut framework [51], incorporating several modifications. Notably, we introduce a communication model representing the interaction between the user and the signing party. Additionally, we modify the usage of non-interactive zero-knowledge (NIZK) proofs (see Section 3 for more details).

The randomizable-blind signature scheme is characterized by its adherence to three fundamental security properties, that are informally as follows:

- **Unforgeability:** This property ensures that it is computationally infeasible for a malicious user, to convince an honest verifier that they possess a valid signature when they do not.
- **Blindness:** Blindness guarantees that a corrupted signer cannot gain any information about the content of the message m during the signature issuance process, beyond the fact that m satisfies a specific predicate.
- **Unlinkability:** It ensures that it is computationally infeasible for a compromised signer or verifier to learn any information about the message m, other than its satisfaction of a predefined predicate. Moreover, unlinkability guarantees that no link can be established between the execution of showing signature and either another show execution or the execution of issuance.

These security properties collectively define the robustness and privacy-preserving characteristics of the randomizable-blind signature scheme.

Definition 15 (Non-threshold randomizable-blind signature). *The scheme* $\mathsf{RB.Sig} = (\mathsf{RB.Sig.KeyGen}, \mathsf{Issue.Sig}, \mathsf{Rand.Sig}, \mathsf{Verify.Sig})$ *consists of the following algorithms and protocols:*

- ***Key generation.*** $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{RB.Sig.KeyGen}(1^\lambda)$
  - *Run* $(p, \mathbb{G}, \tilde{\mathbb{G}}, \mathbb{G}_t, e, g, \tilde{g}) \leftarrow \mathcal{G}(1^\lambda)$*, where* $e : \mathbb{G} \times \tilde{\mathbb{G}} \to \mathbb{G}_t$ *is a bilinear map,* p *is a prime order, and* $g, \tilde{g}$ *are generators.*
  - *Pick* q *random generators* $\{h_\kappa\}_{\kappa=1}^q \leftarrow \mathbb{G}$ *and set the parameters:*

$$\mathsf{par} \leftarrow (p, \mathbb{G}, \tilde{\mathbb{G}}, \mathbb{G}_t, e, g, \tilde{g}, \{h_\kappa\}_{\kappa=1}^q).$$

  - *Set the secret key:*

$$\mathsf{sk} = (x, \{y_\kappa\}_{\kappa=1}^q) \xleftarrow{\$} \mathbb{Z}_p.$$

  - *Set the public key:*

$$\mathsf{pk} = (\mathsf{par}, \tilde{\alpha}, \{\beta_\kappa, \tilde{\beta}_\kappa\}_{\kappa=1}^q) \leftarrow (\mathsf{par}, \tilde{g}^x, \{g^{y_\kappa}, \tilde{g}^{y_\kappa}\}_{\kappa=1}^q).$$

- ***Signature issuance.*** $\mathsf{Issue.Sig}$ *consists of three algorithms:* $\mathsf{RB.Sig.Blinding}$*,* $\mathsf{RB.Sig.Signing}$*, and* $\mathsf{RB.Sig.Unblinding}$*.*
  1. $(\mathtt{x}_1, \pi_1, \{o_\kappa\}_{\kappa=1}^q) \leftarrow \mathsf{RB.Sig.Blinding}(\mathsf{message}, \phi)$*:*
     - *Parse* $\mathsf{message} = \{m_\kappa\}_{\kappa=1}^q \in \mathbb{Z}_p$*. Pick* $o \xleftarrow{\$} \mathbb{Z}_p$ *and compute:*

$$\mathtt{com} = g^o \cdot \prod_{\kappa=1}^{q} h_\kappa^{m_\kappa}.$$

       Send $\mathtt{com}$ *to* $\mathcal{F}_{\mathrm{RO}}$ *and receive* h*.*
     - *For each* $\kappa \in \{1, \ldots, q\}$*, pick* $o_\kappa \xleftarrow{\$} \mathbb{Z}_p$ *and compute:*

$$\mathtt{com}_\kappa = g^{o_\kappa} \cdot h^{m_\kappa}.$$

     - *Generate a* NIZK *proof* π_1 *for the relation:*

$$\mathcal{R}(\mathtt{x}_1, \mathtt{w}_1) = \mathsf{NIZK}\left\{\begin{array}{l} \mathtt{com} = g^o \cdot \prod_{\kappa=1}^{q} h_\kappa^{m_\kappa} \wedge \\ \{\mathtt{com}_\kappa = g^{o_\kappa} \cdot h^{m_\kappa}\}_{\kappa=1}^q \wedge \\ \phi(\{m_\kappa\}_{\kappa=1}^q) = 1 \end{array}\right\}.$$

       where:
       - $\mathtt{x}_1 = (\mathtt{com}, \{\mathtt{com}_\kappa\}_{\kappa=1}^q, h, \phi)$
       - $\mathtt{w}_1 = (\{m_\kappa\}_{\kappa=1}^q, o, \{o_\kappa\}_{\kappa=1}^q)$
  2. $\sigma^{\mathsf{blind}} \leftarrow \mathsf{RB.Sig.Signing}(\mathsf{sk}, \pi_1, \mathtt{x}_1)$*:*
     - *Send* $\mathtt{com}$ *to* $\mathcal{F}_{\mathrm{RO}}$ *and receive* h'*. Abort if* $h \neq h'$ *or* π_1 *is incorrect.*

– *Compute:*

$$c = h^x \cdot \prod_{\kappa=1}^{q} \mathtt{com}_\kappa^{y_\kappa},$$

and set the blind signature:

$$\sigma^{\mathsf{blind}} = (h, c).$$

3. $\sigma \leftarrow \mathsf{RB.Sig.Unblinding}(\{o_\kappa\}_{\kappa=1}^q, \sigma^{\mathsf{blind}})$*:*
   – *Parse* σ^{blind} *as* (h', c)*. Abort if* $h \neq h'$*.*
   – *Compute:*

$$\sigma = (h, s) \leftarrow \left(h, c \cdot \prod_{\kappa=1}^{q} \beta_\kappa^{-o_\kappa}\right).$$

   – *Verify that:*

$$e(h, \tilde{\alpha} \cdot \prod_{\kappa=1}^{q} \tilde{\beta}_\kappa^{m_\kappa}) = e(s, \tilde{g}).$$

- ***Signature randomization.*** $(\sigma^{\mathsf{RND}}, \pi_2, \mathtt{x}_2) \leftarrow \mathsf{Rand.Sig}(\varphi, \sigma, \{m_\kappa\}_{\kappa=1}^q, \mathsf{pk})$*:*
  – *Parse* $\sigma = (h, s)$*. Pick* $r, r' \xleftarrow{\$} \mathbb{Z}_p$*.*
  – *Compute:*

$$\sigma^{\mathsf{RND}} = (h', s') \leftarrow (h^{r'}, s^{r'} \cdot h^{rr'}).$$

  – *Compute:*

$$\varkappa \leftarrow \tilde{\alpha} \cdot \prod_{\kappa=1}^{q} \tilde{\beta}_\kappa^{m_\kappa} \cdot \tilde{g}^r.$$

  – *Generate a* NIZK *proof* π_2 *for:*

$$\mathcal{R}(\mathtt{x}_2, \mathtt{w}_2) = \mathsf{NIZK}\left\{\varkappa = \tilde{\alpha} \cdot \prod_{\kappa=1}^{q} \tilde{\beta}_\kappa^{m_\kappa} \cdot \tilde{g}^r \wedge \varphi(\{m_\kappa\}_{\kappa=1}^q) = 1\right\}.$$

  where:
  * $\mathtt{x}_2 = (\varkappa, \varphi)$
  * $\mathtt{w}_2 = (\{m_\kappa\}_{\kappa=1}^q, r)$
- ***Signature verification.*** $(1, 0) \leftarrow \mathsf{Verify.Sig}(\sigma^{\mathsf{RND}}, \pi_2, \mathtt{x}_2, \mathsf{pk})$*:*
  – *Parse* $\mathtt{x}_2 = (\varkappa, \varphi)$*, and* $\sigma^{\mathsf{RND}} = (h', s')$*.*
  – *Check that* $h' \neq 1$ *and:*

$$e(h', \varkappa) = e(s', \tilde{g}).$$

  – *Verify* π_2*. If incorrect, output* 0*; otherwise, output* 1*.*

Definition 16 (Threshold randomizable-blind signature (modified Coconut)). *The scheme* $\mathsf{TRB.Sig} = (\mathsf{TRB.Sig.KeyGen}, \mathsf{Issue.Sig}, \mathsf{TRB.Sig.Agg}, \mathsf{Rand.Sig}, \mathsf{Verify.Sig})$ *consists of the following algorithms and protocols.*
$\mathsf{TRB.Sig.KeyGen}$ *algorithm can be replaced by a distributed key generation protocol using Gennaro et al.'s distributed key generation protocol [28], which is*

fully simulatable, making it suitable for use with threshold constructions in the Universal Composability (UC) setting. This approach is recognized for its effectiveness, especially when compared to well-known alternatives like the Pedersen DKG [45], which has been shown not to generate a uniformly random joint public key.

- $(\{(\mathsf{pk}_j, \mathsf{sk}_j)\}_{j=1}^{N}, \mathsf{pk}) \leftarrow \mathsf{TRB.Sig.KeyGen}(1^\lambda, N, \Gamma)$
  - *Run* $(p, \mathbb{G}, \tilde{\mathbb{G}}, \mathbb{G}_t, e, g, \tilde{g}) \leftarrow \mathcal{G}(1^\lambda)$*, where:*
    - p *is the order of the groups* $\mathbb{G}, \tilde{\mathbb{G}}, \mathbb{G}_t$*,*
    - $e : \mathbb{G} \times \tilde{\mathbb{G}} \to \mathbb{G}_t$ *is a bilinear map, and*
    - $g \in \mathbb{G}, \tilde{g} \in \tilde{\mathbb{G}}$ *are generators.*

    Pick q *random generators* $\{h_\kappa\}_{\kappa=1}^{q} \leftarrow \mathbb{G}$*, and set the parameters:*

$$\mathsf{par} \leftarrow (p, \mathbb{G}, \tilde{\mathbb{G}}, \mathbb{G}_t, e, g, \tilde{g}, \{h_\kappa\}_{\kappa=1}^{q}).$$

  - *Choose* $(q+1)$ *polynomials* $(v, \{w_\kappa\}_{\kappa=1}^{q})$ *of degree* $(\Gamma - 1)$ *with random coefficients in* $\mathbb{Z}_p$*. Compute:*

$$(x, \{y_\kappa\}_{\kappa=1}^{q}) \leftarrow (v(0), \{w_\kappa(0)\}_{\kappa=1}^{q}).$$

  - *For* $j = 1$ *to* N*, set the secret key* sk_j *of each signer* I_j *as:*

$$\mathsf{sk}_j = (x_j, \{y_{j,\kappa}\}_{\kappa=1}^{q}) \leftarrow (v(j), \{w_\kappa(j)\}_{\kappa=1}^{q}),$$

    and set the verification key pk_j *of each signer* I_j *as:*

$$\mathsf{pk}_j = (\tilde{\alpha}_j, \{\beta_{j,\kappa}, \tilde{\beta}_{j,\kappa}\}_{\kappa=1}^{q}) \leftarrow (\tilde{g}^{x_j}, \{g^{y_{j,\kappa}}, \tilde{g}^{y_{j,\kappa}}\}_{\kappa=1}^{q}).$$

  - *Set the public key for the scheme as:*

$$\mathsf{pk} = (\mathsf{par}, \tilde{\alpha}, \{\beta_\kappa, \tilde{\beta}_\kappa\}_{\kappa=1}^{q}) \leftarrow (\mathsf{par}, \tilde{g}^{x}, \{g^{y_\kappa}, \tilde{g}^{y_\kappa}\}_{\kappa=1}^{q}).$$

- Issue.Sig *consists of the following three algorithms:* TRB.Sig.Blinding*,* TRB.Sig.Signing*, and* TRB.Sig.Unblinding*.*
  1. $(\mathsf{x}_1, \pi_1, \{o_\kappa\}_{\kappa=1}^{q}) \leftarrow \mathsf{TRB.Sig.Blinding}(\mathsf{message}, \phi)$*:*
     - *Parse* $\mathsf{message} = \{m_\kappa\}_{\kappa=1}^{q} \in \mathbb{Z}_p$*. Pick* $o \xleftarrow{\$} \mathbb{Z}_p$ *and compute:*

$$\mathtt{com} = g^o \cdot \prod_{\kappa=1}^{q} h_\kappa^{m_\kappa}.$$

       Send com *to* $\mathcal{F}_{\mathrm{RO}}$[13] *and receive* h *from* $\mathcal{F}_{\mathrm{RO}}$*.*
     - *For each* $\kappa \in \{1, \ldots, q\}$*, pick* $o_\kappa \xleftarrow{\$} \mathbb{Z}_p$ *and compute commitments:*

$$\mathtt{com}_\kappa = g^{o_\kappa} \cdot h^{m_\kappa}.$$

[13] $\mathcal{F}_{\mathrm{RO}}$ denotes the functionality of a random oracle, which provides a truly random response from the output domain for every unique input.

– *Compute a* NIZK *proof* π_1 *for the following relation:*

$$\mathcal{R}(\mathtt{x}_1, \mathtt{w}_1) = \mathsf{NIZK}\left\{\begin{array}{l} \mathtt{com} = g^{o} \cdot \prod_{\kappa=1}^{q} h_{\kappa}^{m_\kappa} \wedge \\ \{\mathtt{com}_\kappa = g^{o_\kappa} \cdot h^{m_\kappa}\}_{\kappa=1}^{q} \wedge \\ \phi(\{m_\kappa\}_{\kappa=1}^{q}) = 1 \end{array}\right\}.$$

where
  * $\mathtt{x}_1 = (\mathtt{com}, \{\mathtt{com}_\kappa\}_{\kappa=1}^{q}, h, \phi)$
  * $\mathtt{w}_1 = (\{m_\kappa\}_{\kappa=1}^{q}, o, \{o_\kappa\}_{\kappa=1}^{q})$

2. $\sigma_j^{\mathsf{blind}} \leftarrow \mathsf{TRB.Sig.Signing}(\mathsf{sk}_j, \pi_1, \mathtt{x}_1)$*:*
   – *Send* com *to* $\mathcal{F}_{\mathrm{RO}}$ *and receive* h' *from* $\mathcal{F}_{\mathrm{RO}}$*. Abort if* $h \neq h'$ *or if* π_1 *is invalid.*
   – *Compute:*

$$c_j = h^{x_j} \cdot \prod_{\kappa=1}^{q} \mathtt{com}_{\kappa}^{y_{j,\kappa}}.$$

   Set the blind signature share:

$$\sigma_j^{\mathsf{blind}} = (h, c_j).$$

3. $\sigma_j \leftarrow \mathsf{TRB.Sig.Unblinding}(\{o_\kappa\}_{\kappa=1}^{q}, \sigma_j^{\mathsf{blind}})$*:*
   – *Parse* $\sigma_j^{\mathsf{blind}} = (h', c_j)$*. Abort if* $h \neq h'$*.*
   – *Compute the unblinded signature:*

$$\sigma_j = (h, s_j) \leftarrow \left(h, c_j \cdot \prod_{\kappa=1}^{q} \beta_{j,\kappa}^{-o_\kappa}\right).$$

   – *Verify the unblinded signature by checking that:*

$$e(h, \tilde{\alpha}_j \cdot \prod_{\kappa=1}^{q} \tilde{\beta}_{j,\kappa}^{m_\kappa}) = e(s_j, \tilde{g}).$$

   Abort if this condition does not hold.

- $\sigma_{\mathbb{I}} \leftarrow \mathsf{TRB.Sig.Agg}(\{\sigma_j\}_{j=1}^{\Gamma}, \mathsf{pk})$
  – *Define* $S \subseteq [1, N]$ *as a set of* Γ *indices corresponding to the signers in* $\mathcal{I}$*.*
  – *For each* $j \in S$*, compute the Lagrange basis polynomial evaluated at 0:*

$$l_j = \left[\prod_{i \in S, i \neq j} \frac{i}{i-j}\right] \bmod p.$$

  – *For each* $j \in S$*, parse* $\sigma_j = (h, s_j)$ *and compute the aggregated signature:*

$$\sigma_{\mathbb{I}} = (h, s) \leftarrow \left(h, \prod_{j \in S} s_j^{l_j}\right).$$

– *Verify the aggregated signature by checking:*

$$e(h, \tilde{\alpha} \cdot \prod_{\kappa=1}^{q} \tilde{\beta}_{\kappa}^{m_\kappa}) = e(s, \tilde{g}).$$

Abort if this condition does not hold.

- $(\sigma_{\mathbb{I}}^{\mathsf{RND}}, \pi_2, \mathtt{x}_2) \leftarrow \mathsf{Rand.Sig}(\varphi, \sigma_{\mathbb{I}}, \{m_\kappa\}_{\kappa=1}^{q}, \mathsf{pk})$
  – *Parse the aggregated signature* $\sigma_{\mathbb{I}} = (h, s)$ *and pick random values* $(r, r') \xleftarrow{\$} \mathbb{Z}_p$.
  – *Compute the randomized signature:*

$$\sigma_{\mathbb{I}}^{\mathsf{RND}} = (h', s') \leftarrow (h^{r'}, s^{r'} \cdot h^{rr'}).$$

  – *Parse the message* $\mathsf{message} = \{m_\kappa\}_{\kappa=1}^{q}$ *and compute:*

$$\varkappa \leftarrow \tilde{\alpha} \cdot \prod_{\kappa=1}^{q} \tilde{\beta}_{\kappa}^{m_\kappa} \cdot \tilde{g}^{r}.$$

  – *Generate the* NIZK *proof* π_2 *for the following relation:*

$$\mathcal{R}(\mathtt{x}_2, \mathtt{w}_2) = \mathsf{NIZK}\left\{ \varkappa = \tilde{\alpha} \cdot \prod_{\kappa=1}^{q} \tilde{\beta}_{\kappa}^{m_\kappa} \cdot \tilde{g}^{r} \wedge \varphi(\{m_\kappa\}_{\kappa=1}^{q}) = 1 \right\},$$

  where:
    * $\mathtt{x}_2 = (\varkappa, \varphi)$
    * $\mathtt{w}_2 = (\{m_\kappa\}_{\kappa=1}^{q}, r)$
- $(1, 0) \leftarrow \mathsf{Verify.Sig}(\sigma_{\mathbb{I}}^{\mathsf{RND}}, \pi_2, \mathtt{x}_2, \mathsf{pk})$
  – *Parse* $\mathtt{x}_2 = (\varkappa, \varphi)$ *and* $\sigma_{\mathbb{I}}^{\mathsf{RND}} = (h', s')$.
  – *Check the following conditions:*
    1. $h' \neq 1$, *and*
    2. *The pairing equation holds:*

$$e(h', \varkappa) = e(s', \tilde{g}).$$

  If either condition fails, output 0.
  – *Verify the* NIZK *proof* π_2. *If* π_2 *is not valid, output* 0.
  – *If all checks pass, output* 1.

B Ideal functionalities

B.1 Key registration functionality

The functionality $\mathcal{F}_{\mathsf{KeyReg}}$ is parameterized by a secret key–public key relation $\mathcal{R}(\mathtt{sk}, \mathtt{pk})$ and models key registration within the context of our protocol. A party invokes this functionality with the command $(\mathtt{Reg.key}, \mathsf{sid}, \mathtt{sk}, \mathtt{pk})$, thereby registering a cryptographic key $\mathtt{pk}$ under the unique identifier U associated with

the party. This registration is successful if the caller knows the associated secret key $\mathtt{sk}$, the key $\mathtt{pk}$ has not been previously registered for any other party, and if the caller has not registered any key before. Subsequently, all participants have the ability to invoke the functionality using either $(\mathtt{Key.Retrieval}, \mathsf{sid}, \mathsf{U})$ to obtain the registered key $\mathtt{pk}$ belonging to party U, or $(\mathtt{id.Retrieval}, \mathsf{sid}, \mathtt{pk})$ to ascertain the identifier of the owner of the key $\mathtt{pk}$.

Functionality $\mathcal{F}_{\mathsf{KeyReg}}$

- **Register key.** Upon input $(\mathtt{Reg.Key}, \mathsf{sid}, \mathtt{sk}, \mathtt{pk})$ from some party U, ignore if $\mathcal{R}(\mathtt{sk}, \mathtt{pk})$ does not hold, or there exists $(\cdot, \mathtt{pk})$ or $(\mathsf{U}, \cdot)$. Else, output $(\mathtt{Register}, \mathsf{sid}, \mathsf{U}, \mathtt{pk})$ to $\mathcal{A}$. Upon receiving $(\mathtt{Ok}, \mathsf{sid}, \mathsf{U}, \mathtt{pk})$ from $\mathcal{A}$, record the pair $(\mathsf{U}, \mathtt{pk})$, and output $(\mathtt{Key.Registered}, \mathsf{sid})$ to U.
- **Retrieve key.** Upon input $(\mathtt{Key.Retrieval}, \mathsf{sid}, \mathsf{U})$ from some party U^*, generate a fresh α. Record $(\alpha, \mathsf{U}, \mathsf{U}^*)$. Output $(\mathtt{Key.Retrieval}, \mathsf{sid}, \alpha)$ to $\mathcal{A}$. Upon receiving $(\mathtt{Ok}, \mathsf{sid}, \alpha)$ from $\mathcal{A}$, if there exists a recorded entry $(\alpha, \mathsf{U}, \mathsf{U}^*)$, and $(\mathsf{U}, \mathtt{pk})$, output $(\mathtt{Key.Retrieved}, \mathsf{sid}, \mathsf{U}, \mathtt{pk})$ to U^*. Else, output $(\mathtt{Key.Retrieved}, \mathsf{sid}, \mathsf{U}, \perp)$ to U^*.
- **Retrieve ID.** Upon input $(\mathtt{id.Retrieval}, \mathsf{sid}, \mathtt{pk})$ from some party U^*, generate a fresh β. Record $(\beta, \mathtt{pk}, \mathsf{U}^*)$. Output $(\mathtt{id.Retrieval}, \mathsf{sid}, \beta)$ to $\mathcal{A}$. Upon receiving $(\mathtt{Ok}, \mathsf{sid}, \beta)$ from $\mathcal{A}$, if there exists a recorded entry $(\beta, \mathtt{pk}, \mathsf{U}^*)$, and $(\mathsf{U}, \mathtt{pk})$, output $(\mathtt{id.Retrieved}, \mathsf{sid}, \mathsf{U}, \mathtt{pk})$ to U^*. Else, output $(\mathtt{id.Retrieved}, \mathsf{sid}, \mathtt{pk}, \perp)$ to U^*.

B.2 Random oracle functionality

$\mathcal{F}_{\mathsf{RO}}$ defined in the following models an idealized hash function [15]. The functionality is parameterized by M and H message space and output space respectively.

Functionality $\mathcal{F}_{\mathsf{RO}}$

The functionality is parameterized by M and H message space and output space respectively that acts as follows: Upon receiving $(\mathtt{Query}, \mathsf{sid}, m)$ from a party U:

- Ignore if $m \notin M$. If there exists a tuple (sid, m', h') where $m' = m$, set $h \leftarrow h'$. Else, select $\bar{h} \xleftarrow{\$} H$ such that there is no stored tuple (sid, m^*, h') where $h' = \bar{h}$.[a] Set $h \leftarrow \bar{h}$. Store (sid, m, h). Output $(\mathtt{Query.Re}, \mathsf{sid}, h)$ to party U.

[a] This check is incorporated to prevent output collisions for simplicity, although it is omitted in realizations. This method is adopted to facilitate a more straightforward formal analysis, as seen in approaches like [48].

B.3 Communication channel functionality

To protect the privacy of users, our ideal functionality $\mathcal{F}_{\mathsf{PARS}}$ does not disclose user identities. To achieve this, we use various types of communication channels to send and receive messages, and ensure anonymity at the network level. Even with strong cryptographic measures in place at the protocol level, there is still a risk of "network leakage" that could reveal the identities of the people involved in a communication. Therefore, in our framework, it is crucial to include a certain level of anonymity for both senders and receivers to uphold Universal Composability (UC) anonymity. Beyond the UC formalization, we assume that communications of entities can be routed through anonymous network infrastructures, like Tor [20], to hide their IP addresses. Using such anonymous networks is vital for any system that depends on anonymity to safeguard privacy. Let define the sender and receiver of a message m by S and R respectively.

Functionality $\mathcal{F}_{\mathsf{Ch}}$

Upon input $(\mathtt{Send}, \mathsf{sid}, R, m)$ from S, record a mapping $\mathsf{P}(\mathsf{mid}) \leftarrow (S, R, m)$ where mid is chosen freshly. Output $(\mathtt{Send}, \mathsf{sid}, \alpha, \beta, \mathsf{mid})$ to $\mathcal{A}$. Upon receiving $(\mathtt{Ok}, \mathsf{sid}, \mathsf{mid})$ from $\mathcal{A}$, retrieve $\mathsf{P}(\mathsf{mid}) = (S, R, m)$ and send $(\mathtt{Received}, \mathsf{sid}, \gamma, m)$ to R.

- Once $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{SSA}}$ is called, set $\alpha \leftarrow R, \beta \leftarrow |m|, \gamma \leftarrow S$ (secure and sender anonymous channel).
- Once $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{SRA}}$ is called, set $\alpha \leftarrow S, \beta \leftarrow |m|, \gamma \leftarrow S$ (secure and receiver anonymous channel).
- Once $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{ISAS}}$ is called, set $\alpha \leftarrow R, \beta \leftarrow m, \gamma \leftarrow S$ (insecure, sender anonymous to adversary, and sender known to recipient channel).
- Once $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{IRAS}}$ is called, set $\alpha \leftarrow S, \beta \leftarrow m, \gamma \leftarrow S$ (insecure, receiver anonymous to adversary, and sender known to recipient channel).

B.4 Broadcast functionality

In the following, we define the standard Broadcast functionality $\mathcal{F}_{\mathsf{B}}^{\mathsf{S}}$ from [26] where it does not guarantee secrecy for the message m, and sender anonymous Broadcast functionality $\mathcal{F}_{\mathsf{B}}^{\mathsf{SA}}$. Regarding the realization of $\mathcal{F}_{\mathsf{B}}^{\mathsf{SA}}$, one can employ a point-to-point sender anonymous channel between the user and (receiving) servers, followed by the execution of a Byzantine agreement protocol [12] among servers. Broadcast functionality $\mathcal{F}_{\mathsf{B}}$ parameterized by the set $\mathbb{I} = \{\mathsf{I}_1, ..., \mathsf{I}_D\}$ proceeds as follows:

Functionality $\mathcal{F}_{\mathsf{B}}$

- Once standard broadcast functionality $\mathcal{F}_{\mathsf{B}}^{\mathsf{S}}$ is called act as follows. Upon receiving $(\texttt{Broadcast}, \mathsf{sid}, m)$ from a user U, send $(\texttt{Broadcasted}, \mathsf{sid}, \mathsf{U}, m)$ to all entities in the set $\mathbb{I}$ and to $\mathcal{A}$.
- Once sender anonymous broadcast functionality $\mathcal{F}_{\mathsf{B}}^{\mathsf{SA}}$ is called act as follows:
  - Upon receiving $(\texttt{Broadcast}, \mathsf{sid}, m)$ from a user U, record $(\mathsf{mid}, \mathsf{U})$ where mid is chosen freshly. Send $(\texttt{Broadcasted}, \mathsf{sid}, m, \mathsf{mid})$ to all entities in the set $\mathbb{I}$ and to $\mathcal{A}$.
  - Upon receiving $(\texttt{Send.Back}, \mathsf{sid}, m', \mathsf{mid})$ from I_j, retrieve $(\mathsf{mid}, \mathsf{U})$, and record $(\mathsf{I}_j, m', \mathsf{mid}, \mathsf{U})$. Output $(\texttt{Send}, \mathsf{sid}, \mathsf{I}_j, m', \mathsf{mid})$ to $\mathcal{A}$.
  - Upon receiving $(\texttt{Send.Back.Ok}, \mathsf{sid}, \mathsf{mid})$ from $\mathcal{A}$, retrieve $(\mathsf{I}_j, m', \mathsf{mid}, \mathsf{U})$, and send $(\texttt{Received}, \mathsf{sid}, \mathsf{I}_j, m)$ to U.

B.5 Non-interactive zero-knowledge functionality

Groth et al. [32] established a formal representation of Non-Interactive Zero-Knowledge (NIZK) via an ideal functionality $\mathcal{F}_{\mathsf{NIZK}}$. In contrast to interactive Zero-knowledge Proofs, NIZK does not require the prior specification of the verifier. Consequently, the resulting proof can undergo verification by any party.

Functionality $\mathcal{F}_{\mathsf{NIZK}}$

$\mathcal{F}_{\mathsf{NIZK}}$ is parameterized by a relation $\mathcal{R}$.

- **Proof generation:** On receiving $(\texttt{Prove}, \mathsf{sid}, \mathtt{x}, \mathtt{w})$ from some party U, ignore if $\mathcal{R}(\mathtt{x}, \mathtt{w}) = 0$. Else, send $(\texttt{Prove}, \mathsf{sid}, \mathtt{x})$ to $\mathcal{A}$. Upon receiving $(\texttt{Proof}, \mathsf{sid}, \pi)$ from $\mathcal{A}$, store $(\mathtt{x}, \pi)$ and send $(\texttt{Proof}, \mathsf{sid}, \pi)$ to U.
- **Proof verification:** Upon receiving $(\texttt{Verify}, \mathsf{sid}, \mathtt{x}, \pi)$ from some party U, check whether $(\mathtt{x}, \pi)$ is stored. If not send $(\texttt{Verify}, \mathsf{sid}, \mathtt{x}, \pi)$ to $\mathcal{A}$. Upon receiving the answer $(\texttt{Witness}, \mathsf{sid}, \mathtt{w})$ from $\mathcal{A}$, check $\mathcal{R}(\mathtt{x}, \mathtt{w}) = 1$ and if so, store $(\mathtt{x}, \pi)$. If $(\mathtt{x}, \pi)$ has been stored, output $(\texttt{Vrfed}, \mathsf{sid}, 1)$ to U, else output $(\texttt{Vrfed}, \mathsf{sid}, 0)$.