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

Amirreza Sarencheh, Aggelos Kiayias, and Markulf Kohlweiss*

The University of Edinburgh & IOG
Edinburgh, UK
Amirreza.Sarencheh@ed.ac.uk, Aggelos.Kiayias@ed.ac.uk,
Markulf.Kohlweiss@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) ecosystem. 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 **P**rivacy-preserving, **A**uditable, and **R**egulation-friendly **S**table**coin** that mitigates these issues while enabling high performance both in terms of speed of settlement and for scaling to large numbers of users. Our construction is blockchain-agnostic and is analyzed in the Universal Composition (UC) framework, offering a secure and modular approach for its integration into the broader blockchain ecosystem.

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 stablecoin market has witnessed substantial growth, with a total market capitalization of over $127 billion as of July 2023 [14].

Stablecoins can be categorized into three types based on how they maintain their peg: (i) fiat-backed (e.g., USD Coin [46]), (ii) crypto-backed (e.g., MakerDAO's DAI [35]), and (iii) algorithmic stablecoins (e.g., AMPL [3]). In this work, our emphasis is on fiat-backed stablecoins. It is by far the largest class in terms of market capitalization [15] and the one for which it is easiest to argue

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

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.

The canonical way to implement a fiat-backed stablecoin is to have the issuer create a smart contract on a blockchain such as Ethereum [18] and partner with a custodian such as a bank that can accept user deposits and maintain fiat reserves serving as the stablecoin's collateral. Due to the transparency of the underlying blockchain, the amount of stablecoin issued is 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:

1. 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.
2. 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.

3. Finally, the **performance and scalability** of the construction is questionable as it directly relies on a L1 for settling all transactions.

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 shortcomings 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.

1. **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.
2. **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.
3. **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.
4. **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).
5. **UC security**. We formulate a Universal Composition (UC) functionality [13] 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.
6. **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.
7. **Efficiency & Scalability**. Our design offers practical real-world efficiency through the usage of suitable protocol techniques and efficient building blocks.

1.1 Related Works

Each stablecoin type offers unique advantages and challenges in achieving price stability. Fiat-backed stablecoins like USDT [44], and USD Coin (USDC) [46] 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 [35], and sUSD [43] use cryptocurrencies for backing and smart contracts for stability. Algorithmic stablecoins, like Ampleforth (AMPL) [3], 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 [39].

Within the realm of fiat-backed stablecoins, two noteworthy ones are USDT [44], and USDC [46]. 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 [33].

In the context of the second category, which is crypto-backed stablecoins, DAI [35] and sUSD [43] 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 [3] and Basis [2] 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 of bonds 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 fees 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.

Aztec is a layer two zk-Rollup on Ethereum [1] that uses succinct arguments and Zero-Knowledge (ZK) 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 (see Section 5 for our ZK implementation details) for stablecoin transfers.

In the following, we review some contributions within the digital currency privacy-preserving literature. It is important to note, however, that none of these works are specifically desgined for fiat-backed stablecoins.

Garman et al. [21] investigated the enforcement of regulatory rules within blockchain-ledger-based anonymous payment systems such as Zerocash [8]. Payment systems resembling the Zerocash approach suffer from a drawback in terms of efficiency, rendering them unsuitable for users with limited resources.

Androulaki et al. [4] presented a privacy-preserving auditable token management system, utilizing a UTxO model within a permissioned blockchain. In divergence from our account-based framework, their focus is on business-to-business scenarios, and their scheme lacks a comprehensive approach to regulation-friendliness, a feature incorporated in our approach.

Central Bank Digital Currency (CBDC) systems are another related line of work. Notably, Platypus [47] and PEReDi [29] offer privacy-preserving operations combined with various levels of regulation-friendliness. Our system design is inspired by their account-based model. While we share some architectural similarities especially 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. Furthermore, PARScoin offers an auditability functionality as well as withdrawals (via proofs of burn) that seamlessly interoperate with other blockchains via our proof of burn subprotocol; — these elements are not applicable to the CBDC setting and thus are novel in our approach.

2 Desiderata and Formal Modeling

2.1 Stablecoin Entities

1. **The Custodian:** The custodial assumes a pivotal role in preserving the stability of the stablecoin's value. This responsibility encompasses the custody and proficient management of the assets underpinning the stablecoin, with these assets potentially encompassing conventional currencies such as the USD and EUR. The custodian is tasked with the dual objectives of safeguarding these assets and ensuring their liquidity. When users wish to generate stablecoins, they engage in a process of depositing collateral with the custodian. Conversely, when users seek to exchange their stablecoins for the underlying collateral, they provide proof of (stablecoin) burn, which subsequently prompts the custodian to release the corresponding collateral.
2. **Issuers:** The responsibility for onboarding users, verifying transactions between them, issuing stablecoins, burning stablecoins, and ensuring compliance with regulatory standards is delegated to a group of officially authorized, and independent entities referred to as *issuers* to avoid a single point of trust and failure. Consequently, the custodian, and regulator do not need

to actively engage in the operations of the system, except the custodian for handling fiat currency deposits and withdrawals. The fundamental characteristics of the system, such as accurate transaction verification, are a result of the actions taken by these issuers. They are identified and may potentially be held accountable for any lapses in their responsibilities.

3. **The Auditor:** The auditor is expected to conduct regular audits of the custodian to assess the adequacy of both the reserve assets (relative to the stablecoin in circulation) and their management practices (e.g., this could include overseeing the implementation of robust security protocols on the custodian's end to protect reserve assets from theft, cyberattacks, and other potential risks).
4. **The Regulator:** The responsibility for validating transactions from a regulatory compliance standpoint has been entrusted to a consortium of issuers to mitigate the risks of a singular point of trust or failure. Thus, the regulatory body can be entirely *offline* (or depending on an application, the regulator can run some of the issuer's servers).
5. **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. **Auditing:** Auditing custodians is vital to verify the fiat currency backing, enhance transparency, and build trust among stakeholders, including users, investors, and regulators. This practice mitigates risks associated with insolvency or mismanagement, ensuring the stablecoin's stability. Audits, typically conducted by independent third-party firms, should align with the ecosystem's dynamics and occur regularly [32,5].
2. **Regulatory Compliance:** This term could encapsulate the following facets [45,36].
   (a) *KYC:* KYC procedures are paramount for preventing illicit activities, and fraud by verifying user identities. This promotes essential transparency and accountability in digital currency markets and enhances stablecoin credibility.

   (b) *AML and CFT:* AML measures are pivotal in the stablecoin ecosystem to prevent money laundering and maintain compliance with regulations. CFT measures are crucial as well to prevent the illicit use of stablecoins for funding terrorism. AML and CFT, bolster stablecoin credibility and integrity while preventing illicit financial activities.
3. **Full Privacy:** This property encompasses four aspects [16,29].
   (a) *Account Privacy:* The safeguarding of user account confidentiality is imperative, signifying the necessity to shield the financial balance and other pertinent information within user accounts from the scrutiny of all network participants.
   (b) *Identity Privacy:* Ensuring that for any given transaction, the real-world identities of the sender and receiver remain undisclosed. Moreover, given the identity of a specific user, it should be impossible to trace their involvement in transactions as a sender or receiver.
   (c) *Transaction Privacy:* Guaranteeing that the transferred value from sender to recipient remains concealed.
   (d) *Unlinkability:* Preventing the linkage of an anonymous payment's real-world identities of the sender or receiver to other transactions. Additionally, it ensures that given a transaction, it is impossible to link it to any prior transaction that resulted in the possession of funds used in the current transaction.
4. **Avoiding Single Point of Trust/Failure:** The design and modeling of stablecoin systems represent a pivotal area of concern within the broader realm of Decentralized Finance (DeFi) and blockchain technology. In this context, it is paramount to underscore the significance of avoiding single point of trust and failure that within stablecoin systems can have far-reaching consequences. They potentially can undermine the very principles of decentralization, security, and stability that are central to their functioning. By distributing trust and responsibility across a network of participants, stablecoin systems can enhance their robustness and resilience, reducing vulnerability to malicious attacks, operational failures, and systemic risks.
5. **Blockchain Agnosticism, and Interoperability:** The importance of a stablecoin system being blockchain agnostic lies in its potential to foster broader adoption and long-term sustainability within the rapidly evolving landscape of blockchain technologies. By not being specific to a single blockchain platform, a stablecoin system can mitigate the risks associated with the inherent uncertainties of blockchain development, ensuring adaptability to emerging and superior technological paradigms. Furthermore, blockchain agnosticism promotes interoperability and inclusivity across diverse blockchain ecosystems, thereby enhancing liquidity and reducing systemic vulnerabilities. Such versatility not only supports the resilience of stablecoin systems but also underscores their capacity to fulfill their fundamental role as a reliable store of value and medium of exchange in an ever-expanding digital financial ecosystem.
6. **Integrity:** This requirement mandates that no entity can alter another user's state/account. It also covers double-spending prevention, ensuring that a

user's account state (which for instance records their balance) is updated correctly with each transaction.

2.3 Notations

In this paper, we employ diverse notations, which are summarized in concise tables (refer to Table 1 and Table 2).

Symbol	Description	Symbol	Description
U	User	v	Transacting value
I, I_i	Issuer, The i-th issuer	$\mathcal{I}$	Set of all issuers
N	Total number of issuers	$\mathsf{U_s}$, $\mathsf{U_r}$	Sender, Receiver
CUS	Custodian	AUD	Auditor
$\tilde{\chi}$	ElGamal Encryption	$(\mathsf{pk}, \mathsf{sk})$	User's public-secret key pair
$(\mathsf{pk_A}, \mathsf{sk_A})$	Auditor's public-secret key pair	B	Balance of the user
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
$\mathsf{account}$	User's account	$\mathcal{D}_i$	Local database of the i-th issuer
sn	Serial number for double-spending prevention	$\mathtt{tag}$	Tag for double-spending prevention
$\mathtt{x}$	Statement of proof	$\mathtt{w}$	Witness of proof

Table 1. Table of Notations

2.4 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 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 Universal Composition setting, we allow the adversary $\mathcal{A}$ to drive communication and potentially block messages.

1. **Initialization.** Session identifiers are of the form $\mathtt{sid} = (\{\mathcal{I}, \mathsf{CUS}, \mathsf{AUD}\}, \mathtt{sid}')$. Initially, $\mathsf{init} \leftarrow 0$. At the end of *Initialization* init is set to 1.[1]

[1] 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.

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	$\mathcal{F}_{\mathsf{B}}^{\mathsf{S}}$	Standard broadcast functionality

Table 2. Table of Notations

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 $\mathsf{B} = 0$.
3. **Stablecoin Issuance.** CUS possesses the authority to confirm the identity U and v by providing (U, v) to $\mathcal{F}_{\mathsf{PARS}}$.[2] 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 recording $(\tilde{\chi}, \mathsf{TV})$). $\tilde{\chi}$ is output to $\mathcal{I}$.[3]
   Public-delayed output means $\mathcal{F}_{\mathsf{PARS}}$ lets $\mathcal{A}$ know the output and decide its delivery.
4. **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

[2] 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.

[3] 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.

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.

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

Initialization.

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

User Registration.

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

Stablecoin Issuance.

(a) Upon receiving a message $(\mathtt{Iss}, \mathtt{sid}, \mathsf{U}, v)$ 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 $(\mathtt{Iss}, \mathtt{sid}, \mathsf{SI.idn})$ (resp. $(\mathtt{Iss}, \mathtt{sid}, \mathsf{SI.idn}, (\mathsf{U}, v))$) to $\mathcal{A}$.
(b) Upon receiving $(\mathtt{Iss.Ok}, \mathtt{sid}, \mathsf{SI.idn}, \tilde{\chi})$ 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 $(\mathtt{Iss.End}, \mathtt{sid}, \tilde{\chi})$ to $\mathcal{I}$ via public-delayed output.

Stablecoin Transfer.

(a) Upon receiving a message $(\mathtt{Gen.ST}, \mathtt{sid}, \mathsf{U_r}, v)$ from some party $\mathsf{U_s}$:
  i. Ignore if $\mathbb{R}(\mathsf{U_s}) = \bot$ or $v < 0$.
  ii. Else, generate a new $\mathsf{ST.idn}$ and set $\mathbb{V}(\mathsf{ST.idn}) \leftarrow (\mathsf{U_s}, \mathsf{U_r}, v)$.
  iii. Output $(\mathtt{Gen.ST}, \mathtt{sid}, \mathsf{ST.idn})$ to $\mathcal{A}$.
(b) Upon receiving $(\mathtt{Gen.ST.Ok}, \mathtt{sid}, \mathsf{ST.idn}, \phi)$ 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 $(\mathtt{Gen.ST.End}, \mathtt{sid}, \mathsf{U_r}, v, \phi)$ to $\mathsf{U_s}$ via private-delayed output.

5. **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 honest issuers, $\mathcal{F}_{\mathsf{PARS}}$ proceeds to augment the balance of $\mathsf{U_r}$. The condition $|f_h| < \Gamma$ signifies that fewer than Γ honest issuers are actively engaged in the protocol, thus the protocol remains incomplete (the function f_h gets as input a set of issuer identifiers and outputs the same set where malicious issuer identifiers are removed). 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.
6. **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.
7. **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 $(\mathtt{PoB}.\mathsf{CUS}, \mathtt{sid}, \eta, \ell)$, $\mathcal{F}_{\mathsf{PARS}}$ 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.
8. **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 $\mathcal{Z}$.[4]

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

Stablecoin Claim.

(a) Upon receiving a message $(\texttt{Clm.ST}, \texttt{sid}, \phi)$ 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 $(\texttt{Clm.ST}, \texttt{sid}, \mathsf{SC.idn})$ to $\mathcal{A}$.

(b) Upon receiving $(\texttt{Clm.ST.Ok}, \texttt{sid}, \mathsf{SC.idn})$ 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 $\mathbb{R}(\mathsf{U_r}) = \mathsf{B_r}$.

(c) Upon receiving $(\texttt{Clm.ST.Issuer.Ok}, \texttt{sid}, \phi)$ from $f_h(\{\mathsf{I}_1, \ldots, \mathsf{I}_N\})$:
  i. Ignore if $|f_h(\{\mathsf{I}_1, \ldots, \mathsf{I}_N\})| < \Gamma$.
  ii. Else, output $(\texttt{Clm.ST.Issuer.Ok}, \texttt{sid}, \mathsf{SC.idn})$ to $\mathcal{A}$.

(d) Upon receiving $(\texttt{Clm.ST.Issuer.Ok}, \texttt{sid}, \mathsf{SC.idn})$ 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 $(\phi, \mathsf{U_s}, \mathsf{U_r}, v, b)$.

Stablecoin Burn.

(a) Upon receiving a message $(\texttt{Brn}, \texttt{sid}, v, \ell)$ 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 $(\texttt{Brn}, \texttt{sid}, \ell, \mathsf{SB.idn})$ to $\mathcal{A}$.

(b) Upon receiving $(\texttt{Brn.Ok}, \texttt{sid}, \mathsf{SB.idn}, \eta, \tilde{\chi})$ 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 $(\texttt{Brn.End}, \texttt{sid}, \eta, \ell, \tilde{\chi})$ to $\mathcal{I}$ via public-delayed outputs.

Proof of Burn.

(a) Upon receiving a message $(\texttt{PoB}, \texttt{sid}, \eta, \ell)$ 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 $(\eta, \mathsf{U}, \cdot, \ell, 1)$.

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

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 $(\mathtt{PoB}, \mathtt{sid}, \ell, \mathsf{PB.idn})$ to $\mathcal{A}$.

(b) Upon receiving $(\mathtt{PoB.Ok}, \mathtt{sid}, \mathsf{PB.idn})$ from $\mathcal{A}$:
i. Ignore if $\mathbb{V}(\mathsf{PB.idn}) = \perp$.
ii. Else, retrieve $\mathbb{V}(\mathsf{PB.idn}) = (\eta, \mathsf{U}, v, \ell, q)$.

(c) Upon receiving $(\mathtt{PoB.CUS}, \mathtt{sid}, \eta, \ell)$ from CUS:
i. Output $(\mathtt{PoB.CUS.Ok}, \mathtt{sid}, \mathsf{PB.idn})$ to $\mathcal{A}$.

(d) Upon receiving $(\mathtt{PoB.CUS.Ok}, \mathtt{sid}, \mathsf{PB.idn})$ from $\mathcal{A}$:
i. Retrieve $\mathbb{V}(\mathsf{PB.idn}) = (\eta, \mathsf{U}, v, \ell, q)$ and ignore if $q = 1$.
ii. Else, output $(\mathtt{PoB.End}, \mathtt{sid}, \eta, \mathsf{U}, v, \ell)$ to CUS via private-delayed output, and set $q \leftarrow 1$ in $(\eta, \mathsf{U}, v, \ell, q)$.

Reserve Audit.

(a) Upon receiving a message $(\mathtt{Audit}, \mathtt{sid}, \tilde{\chi}, \mathsf{RV})$ 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 $(\mathtt{Audit.End}, \mathtt{sid}, b)$ to AUD via public delayed output.

3 Our Construction

Our construction Π_{PARS} employs various cryptographic primitives, including ElGamal encryption, randomizable-blind signature, and Pedersen commitment (see Appendix A for formal definitions and associated security properties). Additionally, it uses the following ideal sub-functionalities: $\mathcal{F}_{\mathsf{KeyReg}}, \mathcal{F}_{\mathsf{RO}}, \mathcal{F}_{\mathsf{Ch}}, \mathcal{F}_{\mathsf{B}}$, and $\mathcal{F}_{\mathsf{NIZK}}$ (see Table.2 for their full names and Appendix B for their formal definitions). In the following, verifiers, maintainers, and issuers are interchangeably used.

3.1 High-Level Technical Summary

In Π_{PARS}, inspired by PEReDi [29], users enjoy self-sovereignty over their accounts, with the account state being securely controlled by the user's secret key (together with other values as we will see).

User accounts are of the form: $\mathsf{account} = (\mathsf{B}, \mathsf{sk}, \mathsf{sk}^x, \mathsf{sk}') \in \mathbb{Z}_p^4$ wherein the user balance B and transaction counter x are updated in each transaction. sk, and sk' are randomly chosen by the user from $\mathbb{Z}_p$. The selection of sk^x is motivated by the necessity of incorporating a transaction counter x for preventing double-spending. This counter assures system verifiers (a.k.a. maintainers) that users are not utilizing their previous account. However, employing just x is insufficient,

as we aim to hide this value[5] while demonstrating, through (efficient)[6] 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 $\mathfrak{tag}$, a function of sk^x. Double-spending is prevented by forcing users to disclose $\mathfrak{tag} = g^{(\mathsf{sk}^{x+1})}$ 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 system maintainers (issuers) 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 $\mathfrak{tag}$ is uniquely defined).

The balance B of the user (that is the first element of $\mathsf{account}$) is updated by themselves (so it is privacy-preserving). As an example, once $\mathsf{U_r}$ receives value v, the new balance $\mathsf{B}^{\mathsf{new}}$ is updated by $\mathsf{U_r}$ as $\mathsf{B}^{\mathsf{old}} + v$. Also, the new value for sk^x will be sk^{x+1} so that we have $\mathsf{account}^{\mathsf{old}} = (\mathsf{B}^{\mathsf{old}}, \mathsf{sk}, \mathsf{sk}^x, \mathsf{sk}') \dashrightarrow \mathsf{account}^{\mathsf{new}} = (\mathsf{B}^{\mathsf{old}} + v, \mathsf{sk}, \mathsf{sk}^{x+1}, \mathsf{sk}')$. 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 $\mathfrak{tag}$.

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 $\mathsf{account}^{\mathsf{old}}$), affirming that the proposed account update $\mathsf{account}^{\mathsf{new}}$ is based on their previously approved account while keeping $\mathsf{account}^{\mathsf{old}}$, and $\mathsf{account}^{\mathsf{new}}$ secret. Issuers verify π, which proves that the new blinded account is consistent with the signature of issuers on the old account. If π is verified, they sign the new blinded account and record a 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 employs signature randomization $\sigma_{\mathbb{I}}^{\mathsf{RND}}$ (see Appendix A.4). 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].

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

[6] Non-interactive version of Sigma-protocols

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

[8] This level of privacy can persist even in scenarios where (malicious) CUS collaborates with malicious issuers by using coin flipping protocol between U and CUS 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.

It is helpful to mention that in the construction's protocols $\mathtt{com}, \mathtt{com}_1, \mathtt{com}_2, \mathtt{com}_3, \mathtt{com}_4$, and h values form the user's blinded account where $\mathtt{com}$ is a Pedersen commitment to all values of the account and $\mathtt{com}_i$ is a Pedersen commitment to the i-th element of the account. h is the hash of $\mathtt{com}$.

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 Our PARScoin Protocol

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 (which is the modification of Coconut [42], see Appesectiondix A.4). 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.

Initialization.

1. **The Auditor.** AUD runs KeyGen algorithm of ElGamal encryption and gets $(\mathtt{pk}_{\mathsf{A}}, \mathsf{sk}_{\mathsf{A}})$ as output[9].
2. **Each Issuer.** I_i engages in the distributed key generation protocol $\mathsf{TRB.Sig.KeyGen}$ of threshold randomizable-blind signature and gets $\mathsf{sk}_i = (x_i, \{y_{i,\kappa}\}_{\kappa=1}^4)$, and $\mathtt{pk}_i = (\tilde{\alpha}_i, \{\beta_{i,\kappa}, \tilde{\beta}_{i,\kappa}\}_{\kappa=1}^4) \leftarrow (\tilde{g}^{x_i}, \{g^{y_{i,\kappa}}, \tilde{g}^{y_{i,\kappa}}\}_{\kappa=1}^4)$ 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 $\mathsf{par} \leftarrow (p, \mathbb{G}, \tilde{\mathbb{G}}, \mathbb{G}_t, e, g, \tilde{g}, \{h_\kappa\}_{\kappa=1}^4, g_1, W)$; and $(g_1, W) \in \mathbb{G}$ with unknown discrete logarithm base g.
3. **The Custodian.** CUS runs $\mathsf{RB.Sig.KeyGen}$ algorithm of Non-Threshold Randomizable-Blind Signature (section A.4) and gets $\bar{\mathsf{par}} \leftarrow (\bar{p}, \bar{\mathbb{G}}, \bar{\tilde{\mathbb{G}}}, \bar{\mathbb{G}}_t, \bar{e}, \bar{g}, \bar{\tilde{g}}, \{\bar{h}_\kappa\}_{\kappa=1}^4)$, $\bar{\mathsf{sk}} = (\bar{x}, \{\bar{y}_\kappa\}_{\kappa=1}^4) \xleftarrow{\$} \mathbb{Z}_p$, and $\bar{\mathtt{pk}} = (\bar{\mathsf{par}}, \bar{\tilde{\alpha}}, \{\bar{\beta}_\kappa, \bar{\tilde{\beta}}_\kappa\}_{\kappa=1}^4) \leftarrow (\bar{\mathsf{par}}, \bar{\tilde{g}}^x, \{\bar{g}^{\bar{y}_\kappa}, \bar{\tilde{g}}^{\bar{y}_\kappa}\}_{\kappa=1}^4)$ as outputs.
4. **The User.** U chooses $\mathsf{sk}' \xleftarrow{\$} \mathbb{Z}_p^*$ and computes $\mathtt{pk} = g^{\mathsf{sk}'}$.

Public keys are registered calling $\mathcal{F}_{\mathsf{KeyReg}}$ and are presumed to be readily accessible upon request. Issuers undertake user enrollment process within the system by generating a $\mathsf{TRB.Sig}$ signature for the user's inaugural account. Subsequently, U utilizes this signature for conducting transactions.

User Registration.

Users obtain the signature of issuers for their initial account which is $\mathsf{account} = (0, \mathsf{sk}, 1, \mathsf{sk}')$ where balance and transaction counter are 0. However, users are unwilling to disclose their secret keys to issuers, so that they blind their initial accounts by computing Pedersen commitments. Subsequently, the user proves, through a NIZK proof π, the validity of the blinded information (commitments).

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

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 $(\mathtt{Rgs}, \mathtt{sid})$ from $\mathcal{Z}$ as follows:

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

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. 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 $(\mathtt{Broadcasted}, \mathtt{sid}, \mathsf{U}, \mathfrak{R})$ from $\mathcal{F}_{\mathsf{B}}^{\mathsf{S}}$ acts as follows.

1. 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 $(\mathtt{Verify}, \mathtt{sid}, \mathsf{x}, \pi)$, $(\mathtt{Verification}, \mathtt{sid}, 0)$ is received.
2. 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 $\mathcal{D}_i$.
3. Compute blind signature share $\sigma_i^{\mathsf{blind}}$:
   - Call $\mathcal{F}_{\mathrm{RO}}$ with $(\mathtt{Query}, \mathtt{sid}, \mathtt{com})$, and receive $(\mathtt{Query.Re}, \mathtt{sid}, h')$. 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 $\sigma_i^{\mathsf{blind}} = (h, c_i)$.
4. Call $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{ISAS}}$ with $(\mathtt{Send}, \mathtt{sid}, \mathsf{U}, \sigma_i^{\mathsf{blind}})$.

U performs the unblinding process on the received blind signature share to get σ_i. U disregards it if found to be invalid. Upon accumulating a sufficient number (Γ) of valid signatures 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 $(\mathtt{Received}, \mathtt{sid}, \mathsf{I}_i, \sigma_i^{\mathsf{blind}})$ from $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{ISAS}}$ acts as follows.

1. Unblind the received signature share:
   - 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.
2. 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.

Stablecoin Issuance.

After obtaining the distinctive identifier of the user U and the value v from the environment $\mathcal{Z}$, the custodian CUS randomly chooses a serial number $\mathtt{sn}$. This serial number serves the purpose of preventing the occurrence of double-spending, and it is subsequently transmitted to U.

Upon receiving $(\mathtt{Iss}, \mathtt{sid}, \mathsf{U}, v)$ from $\mathcal{Z}$, CUS picks $\mathsf{sn} \xleftarrow{\$} \mathbb{Z}_p$ and calls $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{SRA}}$ with $(\mathtt{Send}, \mathtt{sid}, \mathsf{U}, (\mathsf{sn}, v))$.

The user U is required to obtain the custodian's signature confirming the value v for the stablecoin that is to be issued. However, due to privacy requirements concerning both v and identity, the user 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 (see Appendix A.4) 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 generating a blind message to get the signature of CUS. The user U demonstrates in ZK the well-formedness of the blinded message (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 $(\mathtt{Received}, \mathtt{sid}, \mathsf{CUS}, (\mathsf{sn}, v))$ from $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{SRA}}$:

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

- $\mathsf{x} = (\bar{\mathtt{com}}, \bar{\mathtt{com}}_1, \bar{\mathtt{com}}_2, \bar{\mathtt{com}}_3, \bar{\mathtt{com}}_4, \bar{h}, \mathtt{pk}, v, \mathsf{sn})$
- $\mathsf{w} = (\mathsf{sn}, \mathsf{sk}, \mathsf{sk}', \bar{o}, \bar{o}_1, \bar{o}_2, \bar{o}_3, \bar{o}_4)$

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

CUS signs the blinded message (encompassing the user's secret keys $(\mathsf{sk}, \mathsf{sk}')$, serial number $\mathtt{sn}$, and stablecoin value v) to generate $\sigma_{\mathsf{CUS}}^{\mathsf{blind}}$. Subsequently, CUS transmits the signed message $\sigma_{\mathsf{CUS}}^{\mathsf{blind}}$ 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 $(\mathtt{Received}, \mathtt{sid}, \mathsf{U}, \bar{\mathfrak{I}})$ from $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{SSA}}$:

1. Parse $\bar{\mathfrak{I}} = (\mathsf{x}, \pi)$, and $\mathsf{x} = (\bar{\mathtt{com}}, \bar{\mathtt{com}}_1, \bar{\mathtt{com}}_2, \bar{\mathtt{com}}_3, \bar{\mathtt{com}}_4, \bar{h}, \mathtt{pk}, v', \mathsf{sn}')$.
2. 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 $(\mathtt{Verify}, \mathtt{sid}, \mathsf{x}, \pi)$, $(\mathtt{Verification}, \mathtt{sid}, 0)$ is received.
   - Upon calling $\mathcal{F}_{\mathsf{KeyReg}}$ with $(\mathtt{Key.Retrieval}, \mathtt{sid}, \mathsf{U})$, $(\mathtt{Key.Retrieved}, \mathtt{sid}, \mathsf{U}, \mathtt{pk}')$ is received where $\mathtt{pk}' \neq \mathtt{pk}$.
   - Call $\mathcal{F}_{\mathrm{RO}}$ with $(\mathtt{Query}, \mathtt{sid}, \bar{\mathtt{com}})$, and $(\mathtt{Query.Re}, \mathtt{sid}, \bar{h}')$ is received where $\bar{h} \neq \bar{h}'$.
3. Upon receiving a message $(\mathtt{Iss}, \mathtt{sid}, \mathsf{U}', v')$ from $\mathcal{Z}$, proceed if $\mathsf{U}' = \mathsf{U}$ and $v' = v$.
4. 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 $\sigma_{\mathsf{CUS}}^{\mathsf{blind}} = (\bar{h}, \bar{c})$.
5. Call $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{IRAS}}$ with $(\mathtt{Send}, \mathtt{sid}, \mathsf{U}, \sigma_{\mathsf{CUS}}^{\mathsf{blind}})$.

U first of all unblinds the signature of CUS, $\sigma_{\mathsf{CUS}}^{\mathsf{blind}}$, to get σ_{CUS}. Then, for preventing possible linkage randomizes the signature $\sigma_{\mathsf{CUS}}^{\mathsf{RND}}$ and generates $\bar{\varkappa}$ to help the verifier of the signature $\sigma_{\mathsf{CUS}}^{\mathsf{RND}}$ to be able to verify it using $\bar{\varkappa}$ (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 for communications between U and CUS, and threshold blind Pointcheval-Sanders signature for communications between U and issuers (see Appendix A.4 for more details).

U is required to update their account $\mathsf{account}^{\mathsf{old}}$ to get $\mathsf{account}^{\mathsf{new}}$ in a way it is consistent with CUS's signature on value v which is supposed to be hidden. Consequently, the user U augments their balance by v ($\mathsf{B}^{\mathsf{new}} = \mathsf{B}^{\mathsf{old}} + v$) and increments the transaction counter by 1 (sk^{x+1}). U blinds $\mathsf{account}^{\mathsf{new}}$ and proves in ZK that the blinded $\mathsf{account}^{\mathsf{new}}$ is consistent with $\sigma_{\mathsf{CUS}}^{\mathsf{RND}}$ and $\mathsf{account}^{\mathsf{old}}$ for which the user has signature of issuers $\sigma_{\mathbb{I}}$.

$\sigma_{\mathbb{I}}$ represents the (threshold) signature of issuers on $\mathsf{account}^{\mathsf{old}}$, and it undergoes randomization by U to yield $\sigma_{\mathbb{I}}^{\mathsf{RND}}$. The purpose of randomization is to preclude any potential linkage by a malicious issuer between the moment of signing $\mathsf{account}^{\mathsf{old}}$ and the subsequent submission of that signature for account update. The computation of the value $\varkappa$ is because of the underlying threshold

blind signature and serves to assist verifiers in validating $\sigma_{\mathbb{I}}^{\mathsf{RND}}$ without revealing the message of the signature which is $\mathsf{account}^{\mathsf{old}}$.

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 $\mathtt{sn}$ is specifically utilized to prevent double-spending on the custodian's message, while $\mathtt{tag}$ serves the purpose of averting double-spending in updating the user's account.

Regarding privacy-enhanced auditing, for stablecoin issuance (and as we will see for stablecoin burn), users encrypt the value v under the public key $\mathtt{pk}_{\mathsf{A}}$ of the auditor AUD, generating $\tilde{\chi}_t$. Note that users 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 as we will see) 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 $(\mathtt{Received}, \mathtt{sid}, \mathsf{CUS}, \sigma_{\mathsf{CUS}}^{\mathsf{blind}})$ from $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{IRAS}}$:

1. Unblind the received signature:
   - Parse $\sigma_{\mathsf{CUS}}^{\mathsf{blind}} = (\bar{h}', \bar{c})$. 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.
2. Parse $\sigma_{\mathsf{CUS}} = (\bar{h}, \bar{s})$, and pick $(\bar{r}, \bar{r}') \xleftarrow{\$} \mathbb{Z}_p$. Compute $\sigma_{\mathsf{CUS}}^{\mathsf{RND}} = (\bar{h}', \bar{s}') = (\bar{h}^{\bar{r}'}, \bar{s}^{\bar{r}'} \cdot \bar{h}^{\bar{r}'\bar{r}})$, and $\bar{\varkappa} = \bar{\tilde{\alpha}} \cdot \bar{\tilde{\beta}}_1^{\mathsf{sn}} \cdot \bar{\tilde{\beta}}_2^{\mathsf{sk}} \cdot \bar{\tilde{\beta}}_3^{v} \cdot \bar{\tilde{\beta}}_4^{\mathsf{sk}'} \cdot \bar{\tilde{g}}^{\bar{r}}$.
3. Parse $\mathsf{account}^{\mathsf{old}} = (\mathsf{B}^{\mathsf{old}}, \mathsf{sk}, \mathsf{sk}^x, \mathsf{sk}')$.
4. Compute $\mathsf{account}^{\mathsf{new}} = (\mathsf{B}^{\mathsf{old}} + v, \mathsf{sk}, (\mathsf{sk}^{x+1}), \mathsf{sk}')$. Pick $o \xleftarrow{\$} \mathbb{Z}_p$ and compute $\mathtt{com} = g^o \cdot h_1^{\mathsf{B}^{\mathsf{old}}+v} \cdot h_2^{\mathsf{sk}} \cdot h_3^{(\mathsf{sk}^{x+1})} \cdot h_4^{\mathsf{sk}'}$.
5. Submit $(\mathtt{Query}, \mathtt{sid}, \mathtt{com})$ to $\mathcal{F}_{\mathrm{RO}}$, and receive $(\mathtt{Query.Re}, \mathtt{sid}, h)$.
6. 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 $\mathtt{com}_4 = g^{o_4} \cdot h^{\mathsf{sk}'}$
7. Parse $\sigma_{\mathbb{I}} = (h, s)$, and pick $(r, r') \xleftarrow{\$} \mathbb{Z}_p$. Compute $\sigma_{\mathbb{I}}^{\mathsf{RND}} = (h', s') = (h^{r'}, s^{r'} \cdot h^{rr'})$.
8. 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 $\mathtt{tag} = g^{(\mathsf{sk}^{x+1})}$.
9. Pick $z \xleftarrow{\$} \mathbb{Z}_p$ and set $\tilde{\chi}_t = (\tilde{C}_1, \tilde{C}_2) = (g^z, \mathtt{pk}_{\mathsf{A}}^z \cdot g_1^v)$.
10. Call $\mathcal{F}_{\mathsf{NIZK}}$ with input $(\mathtt{Prove}, \mathtt{sid}, \mathsf{x}, \mathsf{w})$, for the relation:
    - $\mathsf{R}(\mathsf{x}, \mathsf{w}) = \mathsf{NIZK}\{\bar{\varkappa} = \bar{\tilde{\alpha}} \cdot \bar{\tilde{\beta}}_1^{\mathsf{sn}} \cdot \bar{\tilde{\beta}}_2^{\mathsf{sk}} \cdot \bar{\tilde{\beta}}_3^{v} \cdot \bar{\tilde{\beta}}_4^{\mathsf{sk}'} \cdot \bar{\tilde{g}}^{\bar{r}} \wedge \mathtt{com} = g^o \cdot h_1^{\mathsf{B}^{\mathsf{old}}+v} \cdot h_2^{\mathsf{sk}} \cdot h_3^{(\mathsf{sk}^{x+1})} \cdot h_4^{\mathsf{sk}'} \wedge \mathtt{com}_1 = g^{o_1} \cdot h^{\mathsf{B}^{\mathsf{old}}+v} \wedge \mathtt{com}_2 = g^{o_2} \cdot h^{\mathsf{sk}} \wedge \mathtt{com}_3 =$

$g^{o_3} \cdot h^{(\mathsf{sk}^{x+1})} \wedge \mathtt{com}_4 = g^{o_4} \cdot h^{\mathsf{sk}'} \wedge \varkappa = \tilde{\alpha} \cdot \tilde{\beta}_1^{\mathsf{B}^{\mathsf{old}}} \cdot \tilde{\beta}_2^{\mathsf{sk}} \cdot \tilde{\beta}_3^{\mathsf{sk}^x} \cdot \tilde{\beta}_4^{\mathsf{sk}'} \cdot \tilde{g}^r \wedge \mathfrak{tag} = g^{(\mathsf{sk}^{x+1})} \wedge \tilde{C}_1 = g^z \wedge \tilde{C}_2 = \mathtt{pk}_\mathsf{A}^z \cdot g_1^v\}$

- $\mathsf{x} = (\bar{\varkappa}, \mathsf{sn}, \mathtt{com}, \mathtt{com}_1, \mathtt{com}_2, \mathtt{com}_3, \mathtt{com}_4, h, \varkappa, \mathfrak{tag}, \tilde{\chi}_t, \mathtt{pk}_\mathsf{A})$
- $\mathsf{w} = (\mathsf{B}^{\mathsf{old}}, \mathsf{sk}, \mathsf{sk}^x, \mathsf{sk}', o, o_1, o_2, o_3, o_4, r, \bar{r}, v, z)$

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

Each issuer conducts verification processes, including checking the randomized signature of CUS ($\sigma_{\mathsf{CUS}}^{\mathsf{RND}}$), the randomized signature on the user's old account ($\sigma_{\mathbb{I}}^{\mathsf{RND}}$), and NIZK proof ($\pi$). 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 $\mathfrak{tag}$ and sn, and signs the new blinded account of the user (that is part of the statement x of NIZK) to get $\sigma_i^{\mathsf{new,blind}}$ which is sent back to U.

I_i acts as follows upon receiving $(\mathtt{Broadcasted}, \mathtt{sid}, \mathfrak{I}, \mathsf{mid})$ from $\mathcal{F}_\mathsf{B}^{\mathsf{SA}}$:

1. Parse $\mathfrak{I} = (\sigma_{\mathsf{CUS}}^{\mathsf{RND}}, \sigma_{\mathbb{I}}^{\mathsf{RND}}, \mathsf{x}, \pi)$, and $\mathsf{x} = (\bar{\varkappa}, \mathsf{sn}, \mathtt{com}, \mathtt{com}_1, \mathtt{com}_2, \mathtt{com}_3, \mathtt{com}_4, h, \varkappa, \mathfrak{tag}, \tilde{\chi}_t, \mathtt{pk}_\mathsf{A})$.
2. Ignore $\mathfrak{I}$ if at least one of the following conditions holds:
   - sn or $\mathfrak{tag}$ 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 $(\mathtt{Query}, \mathtt{sid}, \mathtt{com})$, and $(\mathtt{Query.Re}, \mathtt{sid}, h')$ 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 $(\mathtt{Verify}, \mathtt{sid}, \mathsf{x}, \pi)$, $(\mathtt{Verification}, \mathtt{sid}, 0)$ is received.
3. Else, save $\mathfrak{tag}$ and sn in $\mathcal{D}_i$.
4. Compute $\tilde{\chi}_i \leftarrow \tilde{\chi}_i \cdot \tilde{\chi}_t$. Record $\tilde{\chi}_i$, and output $(\mathtt{Iss.End}, \mathtt{sid}, \tilde{\chi}_i)$ to $\mathcal{Z}$.
5. Compute $\sigma_i^{\mathsf{new,blind}}$ similar to the *User Registration* protocol (where $\sigma_i^{\mathsf{blind}}$ was computed).
6. Call $\mathcal{F}_\mathsf{B}^{\mathsf{SA}}$ with $(\mathtt{Send.Back}, \mathtt{sid}, \sigma_i^{\mathsf{new,blind}}, \mathsf{mid})$.

U upon receiving $(\mathtt{Received}, \mathtt{sid}, \mathsf{I}_i, \sigma_i^{\mathsf{new,blind}})$ 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 $\sigma_{\mathbb{I}}^{\mathsf{new}}$.

Stablecoin Transfer.

Our system supports non-interactive payments: $\mathsf{U_s}$ encrypts v under $\mathsf{U_r}$'s public key $\mathtt{pk}_\mathsf{r} = g^{\mathsf{sk}'_\mathsf{r}}$, generating χ_1, while keeping $\mathtt{pk}_\mathsf{r}$ secret (for privacy concerns) by generating χ_2 ($\mathsf{U_s}$ encrypts $\mathtt{pk}_\mathsf{r}$ under useless public value W to commit to

$\mathtt{pk}_\mathsf{r}$, generating χ_2). $\mathsf{U_s}$ encrypts its public key $\mathtt{pk}_\mathsf{s} = g^{\mathsf{sk}'_\mathsf{s}}$ under $\mathtt{pk}_\mathsf{r}$, generating χ_3 (to help $\mathsf{U_r}$ identify $\mathsf{U_s}$). Additionally, $\mathsf{U_s}$ encrypts the constant 0 under $\mathtt{pk}_\mathsf{r}$, generating χ_4.

χ_2 and χ_4 are used in the proofs of the sender and receiver respectively to provide a reference to $\mathtt{pk}_\mathsf{r}$.

The sender $\mathsf{U_s}$ initiates *Stablecoin Transfer* protocol with issuers upon receiving $(\mathtt{Gen.ST}, \mathtt{sid}, \mathsf{U_r}, v)$ from $\mathcal{Z}$ as follows.

1. 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 $\mathsf{B}^{\mathsf{old}} - v$.
2. Compute $\sigma_{\mathbb{I}}^{\mathsf{RND}}$, $\varkappa$, and $\mathtt{tag}$ similar to the *Stablecoin Issuance* protocol.
3. Call $\mathcal{F}_{\mathsf{KeyReg}}$ with $(\mathtt{Key.Retrieval}, \mathtt{sid}, \mathsf{U_r})$. Upon receiving $(\mathtt{Key.Retrieved}, \mathtt{sid}, \mathsf{U_r}, \mathtt{pk}_\mathsf{r})$, pick $(z, y, q, t) \xleftarrow{\$} \mathbb{Z}_p$; and set
   - $\chi_1 = (C_1, C_2) = (g^z, \mathtt{pk}_\mathsf{r}^z \cdot g_1^v)$
   - $\chi_2 = (C'_1, C'_2) = (g^y, W^y \cdot \mathtt{pk}_\mathsf{r})$
   - $\chi_3 = (C''_1, C''_2) = (g^q, \mathtt{pk}_\mathsf{r}^q \cdot \mathtt{pk}_\mathsf{s})$
   - $\chi_4 = (C'''_1, C'''_2) = (g^t, \mathtt{pk}_\mathsf{r}^t)$.
4. Call $\mathcal{F}_{\mathsf{NIZK}}$ with input $(\mathtt{Prove}, \mathtt{sid}, \mathsf{x}, \mathsf{w})$, for the relation:
   - $\mathsf{R}(\mathsf{x}, \mathsf{w}) = \mathsf{NIZK}\{\mathtt{com} = g^o \cdot h_1^{\mathsf{B}^{\mathsf{old}}-v} \cdot h_2^{\mathsf{sk_s}} \cdot h_3^{(\mathsf{sk_s}^{x+1})} \cdot h_4^{\mathsf{sk}'_\mathsf{s}} \wedge \mathtt{com}_1 = g^{o_1} \cdot h^{\mathsf{B}^{\mathsf{old}}-v} \wedge \mathtt{com}_2 = g^{o_2} \cdot h^{\mathsf{sk_s}} \wedge \mathtt{com}_3 = g^{o_3} \cdot h^{(\mathsf{sk_s}^{x+1})} \wedge \mathtt{com}_4 = g^{o_4} \cdot h^{\mathsf{sk}'_\mathsf{s}} \wedge \varkappa = \tilde{\alpha} \cdot \tilde{\beta}_1^{\mathsf{B}^{\mathsf{old}}} \cdot \tilde{\beta}_2^{\mathsf{sk_s}} \cdot \tilde{\beta}_3^{(\mathsf{sk_s}^x)} \cdot \tilde{\beta}_4^{\mathsf{sk}'_\mathsf{s}} \cdot \tilde{g}^r \wedge \mathtt{tag} = g^{(\mathsf{sk_s}^{x+1})} \wedge C_1 = g^z \wedge C_2 = (C'_2/W^y)^z \cdot g_1^v \wedge C'_1 = g^y \wedge C'_2 \wedge C''_1 = g^q \wedge C''_2 = (C'_2/W^y)^q \cdot g^{\mathsf{sk}'_\mathsf{s}} \wedge C'''_1 = g^t \wedge C'''_2 = (C'_2/W^y)^t \wedge v \in [0, \mathsf{B}^{\mathsf{old}}]\}$
   - $\mathsf{x} = (\mathtt{com}, \mathtt{com}_1, \mathtt{com}_2, \mathtt{com}_3, \mathtt{com}_4, h, \varkappa, \mathtt{tag}, \chi_1, \chi_2, \chi_3, \chi_4, W)$
   - $\mathsf{w} = (\mathsf{B}^{\mathsf{old}}, \mathsf{sk_s}, (\mathsf{sk_s}^x), \mathsf{sk}'_\mathsf{s}, v, z, y, q, t, o, o_1, o_2, o_3, o_4, r)$.
5. Upon receiving $(\mathtt{Proof}, \mathtt{sid}, \pi)$ from $\mathcal{F}_{\mathsf{NIZK}}$, set $\mathfrak{T} \leftarrow (\sigma_{\mathbb{I}}^{\mathsf{RND}}, \mathsf{x}, \pi)$.
6. Call $\mathcal{F}_{\mathsf{B}}^{\mathsf{SA}}$ with $(\mathtt{Broadcast}, \mathtt{sid}, \mathfrak{T})$.

Similar to *Stablecoin Issuance* protocol each issuer verifies the randomized signature on the user's old account ($\sigma_{\mathbb{I}}^{\mathsf{RND}}$), and NIZK proof ($\pi$). Additionally, each issuer searches its local database $\mathcal{D}_i$ for double-spending checking. If all these checks pass, the issuer signs the new blinded account of the user $\sigma_i^{\mathsf{new,blind}}$ which is sent back to U.

I_i upon receiving $(\mathtt{Broadcasted}, \mathtt{sid}, \mathfrak{T}, \mathsf{mid})$ from $\mathcal{F}_{\mathsf{B}}^{\mathsf{SA}}$ acts as follows.

1. Parse $\mathfrak{T} = (\sigma_{\mathbb{I}}^{\mathsf{RND}}, \mathsf{x}, \pi)$, and $\mathsf{x} = (\mathtt{com}, \mathtt{com}_1, \mathtt{com}_2, \mathtt{com}_3, \mathtt{com}_4, h, \varkappa, \mathtt{tag}, \chi_1, \chi_2, \chi_3, \chi_4, W)$.
2. Ignore if at least one of the following conditions holds:
   - $\mathtt{tag}$ 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 $(\mathtt{Verify}, \mathtt{sid}, \mathsf{x}, \pi)$, $(\mathtt{Verification}, \mathtt{sid}, 0)$ is received.
3. Else, save $\mathtt{tag}$, and $\phi = (\chi_1, \chi_3, \chi_4)$ in $\mathcal{D}_i$.
4. Compute $\sigma_i^{\mathsf{new,blind}}$ similar to the *Stablecoin Issuance* protocol.

5. Call $\mathcal{F}_{\mathsf{B}}^{\mathsf{SA}}$ with $(\texttt{Send.Back}, \texttt{sid}, \sigma_i^{\mathsf{new,blind}}, \mathsf{mid})$.

$\mathsf{U_s}$ upon receiving $(\texttt{Received}, \texttt{sid}, \mathsf{I}_i, \sigma_i^{\mathsf{new,blind}})$ 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 $(\texttt{Gen.ST.End}, \texttt{sid}, \mathsf{U_r}, v, \phi)$ to $\mathcal{Z}$ where $\phi = (\chi_1, \chi_3, \chi_4)$.

Stablecoin Claim.

$\mathsf{U_r}$ 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 $\mathsf{R}(\mathsf{x}, \mathsf{w})$) so that with the confirmation of issuers (who receive ϕ from $\mathcal{Z}$) the balance is increased by v.

The receiver $\mathsf{U_r}$ initiates *Stablecoin Claim* protocol with issuers upon receiving $(\texttt{Clm.ST}, \texttt{sid}, \phi)$ from $\mathcal{Z}$ as follows.

1. Parse $\phi = (\chi_1, \chi_3, \chi_4)$, $\chi_1 = (C_1, C_2)$, $\chi_3 = (C_1'', C_2'')$, and $\chi_4 = (C_1''', C_2''')$.
2. Compute $g_1^v = C_2/(C_1)^{\mathsf{sk}_\mathsf{r}'}$ and $\mathtt{pk_s} = C_2''/(C_1'')^{\mathsf{sk}_\mathsf{r}'}$. Extract v from g_1^v.
3. Compute $\sigma_{\mathbb{I}}^{\mathsf{RND}}$, $\varkappa$, and $\mathtt{tag}$ similar to the *Stablecoin Issuance* protocol.
4. Call $\mathcal{F}_{\mathsf{NIZK}}$ with input $(\texttt{Prove}, \texttt{sid}, \mathsf{x}, \mathsf{w})$, for the relation:
   - $\mathsf{R}(\mathsf{x}, \mathsf{w}) = \mathsf{NIZK}\{\mathtt{com} = g^o \cdot h_1^{\mathsf{B}^{\mathsf{old}}+v} \cdot h_2^{\mathsf{sk}_\mathsf{r}} \cdot h_3^{(\mathsf{sk}_\mathsf{r}{}^{x+1})} \cdot h_4^{\mathsf{sk}_\mathsf{r}'} \wedge \mathtt{com}_1 = g^{o_1} \cdot h^{\mathsf{B}^{\mathsf{old}}+v} \wedge \mathtt{com}_2 = g^{o_2} \cdot h^{\mathsf{sk}_\mathsf{r}} \wedge \mathtt{com}_3 = g^{o_3} \cdot h^{(\mathsf{sk}_\mathsf{r}{}^{x+1})} \wedge \mathtt{com}_4 = g^{o_4} \cdot h^{\mathsf{sk}_\mathsf{r}'} \wedge \varkappa = \tilde{\alpha} \cdot \tilde{\beta}_1^{\mathsf{B}^{\mathsf{old}}} \cdot \tilde{\beta}_2^{\mathsf{sk}_\mathsf{r}} \cdot \tilde{\beta}_3^{(\mathsf{sk}_\mathsf{r}{}^{x})} \cdot \tilde{\beta}_4^{\mathsf{sk}_\mathsf{r}'} \cdot \tilde{g}^r \wedge \mathtt{tag} = g^{(\mathsf{sk}_\mathsf{r}{}^{x+1})} \wedge C_2 = C_1^{\mathsf{sk}_\mathsf{r}'} \cdot g_1^v \wedge C_2''' = (C_1''')^{\mathsf{sk}_\mathsf{r}'}\}$
   - $\mathsf{x} = (\mathtt{com}, \mathtt{com}_1, \mathtt{com}_2, \mathtt{com}_3, \mathtt{com}_4, h, \varkappa, \mathtt{tag}, \chi_1, \chi_4)$
   - $\mathsf{w} = (\mathsf{B}^{\mathsf{old}}, \mathsf{sk}_\mathsf{r}, (\mathsf{sk}_\mathsf{r}{}^{x}), \mathsf{sk}_\mathsf{r}', v, o, o_1, o_2, o_3, o_4, r)$
5. Upon receiving $(\texttt{Proof}, \texttt{sid}, \pi)$ from $\mathcal{F}_{\mathsf{NIZK}}$, set $\mathfrak{D} \leftarrow (\sigma_{\mathbb{I}}^{\mathsf{RND}}, \mathsf{x}, \pi, \chi_3)$.
6. Call $\mathcal{F}_{\mathsf{B}}^{\mathsf{SA}}$ with $(\texttt{Broadcast}, \texttt{sid}, \mathfrak{D})$.

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 the user's new blinded account.

I_i upon receiving $(\texttt{Broadcasted}, \texttt{sid}, \mathfrak{D}, \mathsf{mid})$ from $\mathcal{F}_{\mathsf{B}}^{\mathsf{SA}}$ act as follows.

1. Parse $\mathfrak{D} = (\sigma_{\mathbb{I}}^{\mathsf{RND}}, \mathsf{x}, \pi, \chi_3)$, and $\mathsf{x} = (\mathtt{com}, \mathtt{com}_1, \mathtt{com}_2, \mathtt{com}_3, \mathtt{com}_4, h, \varkappa, \mathtt{tag}, \chi_1, \chi_4)$.
2. Ignore $\mathfrak{D}$ if at least one of the following conditions holds:
   - $\mathtt{tag}$ or χ_1 already exists in $\mathcal{D}_i$.
   - Upon calling $\mathcal{F}_{\mathsf{NIZK}}$ with $(\texttt{Verify}, \texttt{sid}, \mathsf{x}, \pi)$, $(\texttt{Verification}, \texttt{sid}, 0)$ 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 $\mathcal{D}_i$
3. Upon receiving $(\texttt{Clm.ST.Ok}, \texttt{sid}, \phi^*)$ from $\mathcal{Z}$, parse $\phi^* = (\chi_1^*, \chi_3^*, \chi_4^*)$. Proceed if $\chi_1^* = \chi_1$, $\chi_3^* = \chi_3$, and $\chi_4^* = \chi_4$.

4. Save $\mathtt{tag}$, and χ_1 in $\mathcal{D}_i$.
5. Compute $\sigma_i^{\mathsf{new,blind}}$ similar to the *Stablecoin Issuance* protocol.
6. Call $\mathcal{F}_{\mathsf{B}}^{\mathsf{SA}}$ with $(\mathtt{Send.Back}, \mathtt{sid}, \sigma_i^{\mathsf{new,blind}}, \mathsf{mid})$.

$\mathsf{U_r}$ upon receiving $(\mathtt{Received}, \mathtt{sid}, \mathsf{I}_i, \sigma_i^{\mathsf{new,blind}})$ 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 $\sigma_{\mathbb{I}}^{\mathsf{new}}$.

Stablecoin Burn.

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.

The user U initiates *Stablecoin Burn* protocol with issuers upon receiving $(\mathtt{Brn}, \mathtt{sid}, v, \ell)$ from $\mathcal{Z}$ as follows.

1. Compute $\mathtt{com}, \mathtt{com}_1, \mathtt{com}_2, \mathtt{com}_3, \mathtt{com}_4$ and h similar to the *Stablecoin Transfer* protocol.
2. Compute $\sigma_{\mathbb{I}}^{\mathsf{RND}}$, $\varkappa$, and $\mathtt{tag}$ similar to the *Stablecoin Transfer* protocol.
3. Pick $(z, q, t) \xleftarrow{\$} \mathbb{Z}_p$, set
   - $\chi_1 = (C_1, C_2) = (g^z, \mathtt{pk}^z \cdot g_1^v)$
   - $\chi_2 = (C_1', C_2') = (g^t, \mathtt{pk}^t)$
   - $\tilde{\chi}_t = (\tilde{C}_1, \tilde{C}_2) = (g^q, \mathtt{pk}_{\mathsf{A}}^q \cdot g_1^v)$.
4. Call $\mathcal{F}_{\mathsf{NIZK}}$ with input $(\mathtt{Prove}, \mathtt{sid}, \mathsf{x}, \mathsf{w})$, for the relation:
   - $\mathsf{R}(\mathsf{x}, \mathsf{w}) = \mathsf{NIZK}\{\mathtt{com} = g^o \cdot h_1^{\mathsf{B}^{\mathsf{old}}-v} \cdot h_2^{\mathsf{sk}} \cdot h_3^{(\mathsf{sk}^{x+1})} \cdot h_4^{\mathsf{sk}'} \wedge \mathtt{com}_1 = g^{o_1} \cdot h^{\mathsf{B}^{\mathsf{old}}-v} \wedge \mathtt{com}_2 = g^{o_2} \cdot h^{\mathsf{sk}} \wedge \mathtt{com}_3 = g^{o_3} \cdot h^{(\mathsf{sk}^{x+1})} \wedge \mathtt{com}_4 = g^{o_4} \cdot h^{\mathsf{sk}'} \wedge \varkappa = \tilde{\alpha} \cdot \tilde{\beta}_1^{\mathsf{B}^{\mathsf{old}}} \cdot \tilde{\beta}_2^{\mathsf{sk}} \cdot \tilde{\beta}_3^{(\mathsf{sk}^x)} \cdot \tilde{\beta}_4^{\mathsf{sk}'} \cdot \tilde{g}^r \wedge \mathtt{tag} = g^{(\mathsf{sk}^{x+1})} \wedge C_1 = g^z \wedge C_2 = g^{\mathsf{sk}' \cdot z} \cdot g_1^v \wedge \tilde{C}_1 = g^q \wedge \tilde{C}_2 = \mathtt{pk}_{\mathsf{A}}^q \cdot g_1^v \wedge C_1' = g^t \wedge C_2' = g^{\mathsf{sk}' \cdot t} \wedge v \in [0, \mathsf{B}^{\mathsf{old}}]\}$
   - $\mathsf{x} = (\mathtt{com}, \mathtt{com}_1, \mathtt{com}_2, \mathtt{com}_3, \mathtt{com}_4, h, \varkappa, \mathtt{tag}, \chi_1, \chi_2, \tilde{\chi}_t, \mathtt{pk}_{\mathsf{A}}, \ell)$
   - $\mathsf{w} = (\mathsf{B}^{\mathsf{old}}, \mathsf{sk}, (\mathsf{sk}^x), \mathsf{sk}', v, o, o_1, o_2, o_3, o_4, r, z, q, t)$.
5. Upon receiving $(\mathtt{Proof}, \mathtt{sid}, \pi)$ from $\mathcal{F}_{\mathsf{NIZK}}$, set $\mathfrak{B} \leftarrow (\sigma_{\mathbb{I}}^{\mathsf{RND}}, \mathsf{x}, \pi)$.
6. Call $\mathcal{F}_{\mathsf{B}}^{\mathsf{SA}}$ with $(\mathtt{Broadcast}, \mathtt{sid}, \mathfrak{B})$.

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 $(\texttt{Broadcasted}, \texttt{sid}, \mathfrak{B}, \mathsf{mid})$ from $\mathcal{F}_{\mathsf{B}}^{\mathsf{SA}}$ acts as follows.

1. Parse $\mathfrak{B} = (\sigma_{\mathbb{I}}^{\mathsf{RND}}, \mathsf{x}, \pi)$, and $\mathsf{x} = (\texttt{com}, \texttt{com}_1, \texttt{com}_2, \texttt{com}_3, \texttt{com}_4, h, \varkappa, \mathfrak{tag}, \chi_1, \chi_2, \tilde{\chi}_t, \mathtt{pk}_{\mathsf{A}}, \ell)$.
2. Ignore $\mathfrak{B}$ if at least one of the following conditions holds:
   - $\mathfrak{tag}$ already exists in $\mathcal{D}_i$.
   - Upon calling $\mathcal{F}_{\mathsf{NIZK}}$ with $(\texttt{Verify}, \texttt{sid}, \mathsf{x}, \pi)$, $(\texttt{Verification}, \texttt{sid}, 0)$ is received.
   - Parse $\sigma_{\mathbb{I}}^{\mathsf{RND}} = (h', s')$ and if $h' = 1$ or if $e(h', \varkappa) \neq e(s', \tilde{g})$.
3. Compute $\tilde{\chi}_i \leftarrow \tilde{\chi}_i / \tilde{\chi}_t$. Record $\tilde{\chi}_i$, and output $(\texttt{Brn.End}, \texttt{sid}, \eta, \ell, \tilde{\chi}_i)$ to $\mathcal{Z}$ where $\eta = (\chi_1, \chi_2)$.
4. Save $\mathfrak{tag}$ in $\mathcal{D}_i$, compute $\sigma_i^{\mathsf{new,blind}}$ similar to the *Stablecoin Transfer* protocol.
5. Call $\mathcal{F}_{\mathsf{B}}^{\mathsf{SA}}$ with $(\texttt{Send.Back}, \texttt{sid}, \sigma_i^{\mathsf{new,blind}}, \mathsf{mid})$.

The user U upon receiving $(\texttt{Received}, \texttt{sid}, \mathsf{I}_i, \sigma_i^{\mathsf{new,blind}})$ from $\mathcal{F}_{\mathsf{B}}^{\mathsf{SA}}$ acts similar to the *Stablecoin Transfer* protocol to compute $\sigma_{\mathbb{I}}^{\mathsf{new}}$.

Proof of Burn.

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 (without revealing their secret key).

The user U initiates *Proof of Burn* protocol upon receiving $(\texttt{PoB}, \texttt{sid}, \eta, \ell)$ from $\mathcal{Z}$ as follows.

1. Parse $\eta = (\chi_1, \chi_2)$, $\chi_1 = (C_1, C_2)$, and $\chi_2 = (C_1', C_2')$.
2. Compute $g_1^v = C_2 / (C_1)^{\mathsf{sk}'}$. Extract v from g_1^v.
3. Call $\mathcal{F}_{\mathsf{NIZK}}$ with input $(\texttt{Prove}, \texttt{sid}, \mathsf{x}, \mathsf{w})$, for the relation:
   - $\mathsf{R}(\mathsf{x}, \mathsf{w}) = \mathsf{NIZK}\{C_2 = C_1^{\mathsf{sk}'} \cdot g_1^v \wedge C_2' = (C_1')^{\mathsf{sk}'}\}$
   - $\mathsf{x} = (\eta, v, \ell)$
   - $\mathsf{w} = \mathsf{sk}'$
4. Upon receiving $(\texttt{Proof}, \texttt{sid}, \pi)$ from $\mathcal{F}_{\mathsf{NIZK}}$, set $\mathfrak{V} \leftarrow (\mathsf{x}, \pi)$.
5. Call $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{SSA}}$ with $(\texttt{Send}, \texttt{sid}, \mathsf{CUS}, \mathfrak{V})$.

CUS verifies the NIZK proof of U, also checks if η has alrady 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 $\mathcal{Z}$.

The custodian CUS upon receiving $(\texttt{Received}, \texttt{sid}, \mathsf{U}, \mathfrak{V})$ from $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{SSA}}$ acts as follows.

1. Parse $\mathfrak{V} = (\mathsf{x}, \pi)$, and $\mathsf{x} = (\eta, v, \ell)$.
2. Ignore $\mathfrak{V}$ if upon calling $\mathcal{F}_{\mathsf{NIZK}}$ with $(\texttt{Verify}, \texttt{sid}, \mathsf{x}, \pi)$, $(\texttt{Verification}, \texttt{sid}, 0)$ is received or η already exists.
3. Upon receiving $(\texttt{PoB}.\mathsf{CUS}, \texttt{sid}, \eta^*, \ell^*)$ from $\mathcal{Z}$, proceed if $\eta^* = \eta$, and $\ell^* = \ell$.
4. Record η, and output $(\texttt{PoB}.\texttt{End}, \texttt{sid}, \eta, \mathsf{U}, v, \ell)$ to $\mathcal{Z}$.

Reserve Audit.

As described in section 2.4, $\mathcal{Z}$ 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 $(\texttt{Audit}, \texttt{sid}, \tilde{\chi}, \mathsf{RV})$ from $\mathcal{Z}$.

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

[a] Considering realistic values for TV, we emphasize that the auditor possesses the necessary capabilities to effectively calculate TV.

Regulation-Friendliness. The incorporation of regulatory requirements can be seamlessly accommodated within the framework of PARScoin. Regarding KYC verification, in the *User Registering* protocol, the issuers can check KYC guidelines against a user in a distributed fashion before on-boarding them (if below the threshold of issuers approve a user, user registration will not be approved). Subsequent to the registration process, the expenditure for continuous KYC can be low, as users possess the capacity to efficiently prove things about themselves such as confirming their non-inclusion in any revocation list. As another example, in the context of the *Stablecoin Transfer* protocol, the sender encrypts the public key of the receiver $\mathtt{pk}_\mathsf{r}$, generating an ElGamal encryption χ. The sender can incorporate within their NIZK proof the assertion that $\mathtt{pk}_\mathsf{r}$ does not feature in the list of entities associated with terrorism (which is publicly available), thereby ensuring compliance with CFT regulations while keeping $\mathtt{pk}_\mathsf{r}$ hidden for privacy concerns.

Moreover, the account-based structure of PARScoin permits the storage of supplementary information within an account state thereby facilitating *efficient* (see Appendix 5) NIZK proofs for demonstrating compliance with specific regulations. For instance, the European Central Bank [6], has mentioned imposing limits on the total amount of digital currency that can be sent in a specific period of time (sending limit). In PARScoin, it is straightforward to augment user accounts with a value s that increases with the value of the transaction when the user acts as a sender and mandate the incorporation of statement $s < \mathsf{S.max}$ in NIZK proofs via a range proof while preserving privacy.

Additionally, incorporating features akin to those outlined in PEReDi [29], such as *distributed privacy revocation* and *distributed tracing malicious users*,

is easily achievable. As previously stated, our design draws inspiration from PEReDi's account-based model. Therefore, we can seamlessly integrate support for privacy revocation (by introducing threshold encryption[10] of user's public key and transaction value) and tracing (by verifiable secret sharing of user's tracing secret key at the *User Registration* protocol) through distributed servers.

Account Recovery. The account state model necessitates users to possess an understanding 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 $\mathtt{tag}$ up to the point of the latest entry in the blockchain (refer to 3.2 for discussions around modelling public ledger in this paper). 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.

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 PARScoin Security

In the following, we provide our main *Theorem 1.*.

[10] in some jurisdictions, it is mandatory that all transaction information should be visible to a centralized regulator. In such cases, one can encrypt all of this data under the regulator's public key and embed the well-formedness of the encryption in the NIZK.

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 (§A.6), the* IND-CPA *security (§3) of ElGamal (§A.2) encryption, the* EUF-CMA *security (§7) of Pointcheval-Sanders signatures (§4) in the random oracle model, and the hardness of the d-strong Diffie-Hellman problem (§11), 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 advantage better than* $\mathsf{Adv}^{\mathsf{Bind\text{-}com}}_{\mathcal{A}} + \mathsf{max}_1 \cdot \mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathcal{A}} + \mathsf{max}_2 \cdot \mathsf{Adv}^{\mathsf{d\text{-}SDDH}}_{\mathcal{A}} + \mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathcal{A}}$ *with static corruptions in the presence of an arbitrary number of malicious users, and up to t malicious issuers that are all colluding.*

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, $\mathfrak{game}^{(i)}$, 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 $\mathfrak{game}^{(i)}$ as $\Pr[\mathfrak{game}^{(i)}]$.

4.1 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.

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

1. In $\mathfrak{game}^{(1)}$, $\mathcal{F}^{(1)}_{\mathsf{PARS}}$ 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 A.6):

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

2. In $\mathfrak{game}^{(2)}$, all plaintexts[11] generated by honest users are changed to random group elements selected by $\mathcal{S}^{(2)}$. It is argued that under IND-CPA property (definition 3) of ElGamal encryption scheme (section A.2):

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

[11] g_1^v in *Stablecoin Issuance*, $(g_1^v, \mathtt{pk}_\mathsf{r}, \mathtt{pk}_\mathsf{s}, 1)$ 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 $(\chi_1, \chi_2, \tilde{\chi}_t)$).

3. In $\mathfrak{game}^{(3)}$, all $\mathtt{tag}$ 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 11):

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

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

$$|\Pr[\mathfrak{game}^{(4)}] = \Pr[\mathfrak{game}^{(3)}]|$$

5. In $\mathfrak{game}^{(5)}$, $\mathcal{F}^{(5)}_{\mathsf{PARS}}$ prohibits $\mathcal{S}^{(5)}$ from submitting any message to $\mathcal{F}^{(5)}_{\mathsf{PARS}}$ 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 7) of Pointcheval-Sanders signature (definition 4):

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

6. In $\mathfrak{game}^{(6)}$, 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:

$$|\Pr[\mathfrak{game}^{(6)}] = \Pr[\mathfrak{game}^{(5)}]|$$

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

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

Details of Games and Reductions. $\mathfrak{game}^{(0)}$: At the outset, $\mathcal{F}^{(0)}_{\mathsf{PARS}}$ relays all communication with $\mathcal{Z}$. And the simulator $\mathcal{S}^{(0)}$ emulates the execution of the real-world protocol $\mathsf{EXEC}_{\Pi_{\mathsf{PARS}},\mathcal{A},\mathcal{Z}}$.

$\mathfrak{game}^{(1)}$: Same as $\mathfrak{game}^{(0)}$ except that $\mathfrak{game}^{(1)}$ 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 $(\mathtt{Verify}, \mathtt{sid}, \mathsf{x}, \pi)$ and $(\mathtt{Verify}, \mathtt{sid}, \mathsf{x}', \pi')$ to $\mathcal{A}$ and receives $(\mathtt{Witness}, \mathtt{sid}, \mathsf{w})$ and $(\mathtt{Witness}, \mathtt{sid}, \mathsf{w}')$ respectively. The flag is raised when with the given extracted witnesses, the committed values are different. Therefore, any difference between $\mathfrak{game}^{(1)}$ and $\mathfrak{game}^{(0)}$ 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 $\mathfrak{game}^{(1)}$ from $\mathfrak{game}^{(0)}$ as follows.

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

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

identical to $\mathfrak{game}^{(1)}$ except for the fact that $\mathcal{S}^{(2)}$ selects random group elements as plaintexts for all honest users' ciphertexts. Any disparity between $\mathfrak{game}^{(2)}$ and $\mathfrak{game}^{(1)}$ 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 $\mathfrak{game}^{(2)}$ from $\mathfrak{game}^{(1)}$ as follows. We introduce a sequences of sub-games:

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

Additionally, let's define $\mathfrak{game}_2^{(1)}$ as a game analogous to $\mathfrak{game}_1^{(1)}$ with the exception that in $\mathfrak{game}_2^{(1)}$ 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 $\mathfrak{game}_{i-1}^{(1)}$ to $\mathfrak{game}_i^{(1)}$ is akin to the reduction described below, ensuring that any difference between $\mathfrak{game}_{i-1}^{(1)}$ and $\mathfrak{game}_i^{(1)}$ 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 $\mathfrak{game}_{\mathsf{max}_1}^{(1)} = \mathfrak{game}^{(2)}$ 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 $\mathfrak{game}_{i-1}^{(1)}$ and $\mathfrak{game}_i^{(1)}$ (for $2 \leq i \leq \mathsf{max}_1$) as follows. If $\mathcal{Z}$ can distinguish between $\mathfrak{game}_{i-1}^{(1)}$ and $\mathfrak{game}_i^{(1)}$, 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 $\mathcal{Z}$, $\mathfrak{game}_i^{(1)}$ 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 $\mathfrak{game}_{i-1}^{(1)}$ from $\mathfrak{game}_i^{(1)}$, $\mathcal{A}'$ 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:

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

$\mathfrak{game}^{(3)}$: Same as $\mathfrak{game}^{(2)}$ except that in $\mathfrak{game}^{(3)}$ we change all tag values of all honest users to random group elements selected from $\mathbb{G}$. Thus, $\mathfrak{game}^{(3)}$ is identical to $\mathfrak{game}^{(2)}$ except for the fact that $\mathcal{S}^{(3)}$ selects random group elements as tag values for all honest users. Any disparity between $\mathfrak{game}^{(3)}$ and $\mathfrak{game}^{(2)}$ 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 $\mathfrak{game}^{(3)}$ from $\mathfrak{game}^{(2)}$ as follows. We introduce a sequences of sub-games:

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

Additionally, let's define $\mathfrak{game}_2^{(2)}$ as a game analogous to $\mathfrak{game}_1^{(2)}$ with the exception that in $\mathfrak{game}_2^{(2)}$ 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 $\mathfrak{game}_{i-1}^{(2)}$ to $\mathfrak{game}_i^{(2)}$ is akin to the reduction described below, ensuring that any difference between $\mathfrak{game}_{i-1}^{(2)}$ and $\mathfrak{game}_i^{(2)}$ 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 $\mathfrak{game}_{\mathsf{max}_2}^{(2)} = \mathfrak{game}^{(3)}$ 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 $\mathfrak{game}_{i-1}^{(2)}$ and $\mathfrak{game}_i^{(2)}$ (for $2 \leq i \leq \mathsf{max}_2$) as follows. If $\mathcal{Z}$ can distinguish between $\mathfrak{game}_{i-1}^{(2)}$ and $\mathfrak{game}_i^{(2)}$, 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 $\mathcal{Z}$, $\mathfrak{game}_i^{(2)}$ 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 $\mathfrak{game}_{i-1}^{(2)}$ from $\mathfrak{game}_i^{(2)}$, $\mathcal{A}'$ can exploit this distinction to break the hardness of d-SDDH assumption. Therefore, under the hardness of d-SDDH assumption, the following inequality holds:

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

$\mathfrak{game}^{(4)}$: This game is identical to $\mathfrak{game}^{(3)}$ with the exception that in $\mathfrak{game}^{(4)}$, 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 $(\mathtt{Verify}, \mathtt{sid}, \mathsf{x}, \pi)$ to $\mathcal{A}$. Upon receiving

$(\texttt{Witness}, \texttt{sid}, \mathsf{w})$ from $\mathcal{A}$, $\mathcal{S}^{(4)}$ parses the witness w to know $\mathsf{account}$, 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 $\mathcal{MAL}$: $s_{\mathsf{hon}} = s^{\prod_{k \in \mathcal{MAL}}((k-\mathsf{hon})/k)} \cdot \prod_{\mathsf{mal} \in \mathcal{MAL}} s_{\mathsf{mal}}^{\prod_{k \in \mathcal{MAL}, k \neq \mathsf{mal}}((\mathsf{hon}-k)/(\mathsf{mal}-k))}$, 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:

$$\Pr[\mathfrak{game}^{(4)}] = \Pr[\mathfrak{game}^{(3)}]$$

$\mathfrak{game}^{(5)}$: Same as the previous game except that in $\mathfrak{game}^{(5)}$, $\mathcal{F}_{\mathsf{PARS}}^{(5)}$ 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, $\mathfrak{game}^{(5)}$ is identical to $\mathfrak{game}^{(4)}$ 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 $\mathfrak{game}^{(5)}$ and $\mathfrak{game}^{(4)}$ 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 $\mathfrak{game}^{(5)}$ from $\mathfrak{game}^{(4)}$ 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, $\mathsf{TRB.Sig}$ 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:

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

$\mathfrak{game}^{(6)}$: In $\mathfrak{game}^{(6)}$, everything is similar to $\mathfrak{game}^{(5)}$ 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

$$\Pr[\mathfrak{game}^{(6))}] = \Pr[\mathfrak{game}^{(5)}]$$

$\mathcal{F}_{\mathsf{PARS}}^{(6)}$ 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 $\mathfrak{game}^{(1)}$ that corresponds to the real-world execution $\mathsf{EXEC}_{\Pi_{\mathsf{PARS}}, \mathcal{A}, \mathcal{Z}}$, and we ended up with $\mathfrak{game}^{(6)}$ 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 in the following concludes the security proof.

Double-spending $\mathtt{tag}$ *is deterministically defined based on the account's current state. This implies that when a user initiates a transaction, utilizing their corresponding tag, and the transaction remains unconfirmed, the exposure of this tag compromises the user's ability to maintain transaction unlinkability in subsequent transactions, as the tag will match the previously revealed one. It's worth noting that a transaction may remain unconfirmed due to factors such as deliberate channel blockage by an adversary. In our formal modelling,* $\mathcal{F}_{\mathsf{PARS}}$*, we intentionally do not disclose this information to the adversary* $\mathcal{A}$*, simplifying the overall analysis. Nevertheless, it is straightforward to achieve unlinkability since the user is able to submit a transaction to the issuers, during which they update only their transaction counter* x *while leaving other account details (*B *and* sk*) unchanged. By doing so, they effectively modify their* x *value, ensuring that the future tag differs from the one that was previously exposed.*

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.*

4.2 Simulation

We denote the real-world protocol and adversary as Π_{PARS} and $\mathcal{A}$, respectively. The simulator $\mathcal{S}$ is described in the subsequent sections, and it is designed to ensure that the observations during real-world execution $\mathsf{EXEC}_{\Pi_{\mathsf{PARS}},\mathcal{A},\mathcal{Z}}$ and ideal-world execution $\mathsf{EXEC}_{\mathcal{F}_{\mathsf{PARS}},\mathcal{S},\mathcal{Z}}$ remain indistinguishable for any probabilistic polynomial-time (PPT) environment $\mathcal{Z}$. At the outset of the execution, $\mathcal{Z}$ initiates the adversary to corrupt certain parties by transmitting a message $(\texttt{Corrupt}, \texttt{sid}, P)$. Subsequently, $\mathcal{S}$ intercepts and processes these corruption messages. It then informs $\mathcal{F}_{\mathsf{PARS}}$ about the parties that have been corrupted by dispatching the message $(\texttt{Corrupt}, \texttt{sid}, P)$. Additionally, the simulator $\mathcal{S}$ maintains a record of the identifiers of the corrupted parties. Internally, the simulator $\mathcal{S}$ operates a version of Π_{PARS} and aims to render the view of the dummy adversary $\mathcal{A}$ in the ideal world indistinguishable from the view of the actual adversary in the real world. $\mathcal{S}$ internally 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 $\mathsf{EXEC}_{\mathcal{F}_{\mathsf{PARS}},\mathcal{S},\mathcal{Z}}$, the honest (dummy) parties transmit their inputs from $\mathcal{Z}$ to $\mathcal{F}_{\mathsf{PARS}}$. The session identifier $\texttt{sid}$ is chosen by $\mathcal{Z}$. The adversary $\mathcal{A}$ issues arbitrary instructions to the corrupted parties. $\mathcal{S}$ submits messages to $\mathcal{F}_{\mathsf{PARS}}$ on behalf of the corrupted parties. The simulator $\mathcal{S}$, which is designed to simulate the view of $\mathcal{A}$ (e.g., by simulating the behavior of honest parties), engages in interactions with both the dummy adversary $\mathcal{A}$ and the functionality $\mathcal{F}_{\mathsf{PARS}}$.

For the sake of clarity and simplicity, we will exclude the 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 $\mathcal{F}_{\mathsf{Ch}}$, leaks to the dummy adversary $\mathcal{A}$ whatever $\mathcal{F}_{\mathsf{Ch}}$ leaks to the real-world adversary. Additionally, for the same reasons, we omit to address the details concerning the fact that all ciphertexts generated by honest users in all the protocols of Π_{PARS} are simulated by the simulator $\mathcal{S}$, following the explanations in section 4.1.

In order to prevent redundancy, we address this matter once here for all the subsidiary protocols of our framework. Concerning the communication between users and issuers, the simulator $\mathcal{S}$, which emulates channel functionality, possesses 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 $(\texttt{Rgs}, \texttt{sid}, \mathsf{U})$ from $\mathcal{F}_{\mathsf{PARS}}$. $\mathcal{S}$ simulates dummy values as blind signature shares of honest issuers $\sigma^{\mathsf{blind}}_{\mathsf{hon}}$. As $\mathcal{S}$ emulates $\mathcal{F}^{\mathsf{S}}_{\mathsf{B}}$, based on its observations concerning channel blockage by $\mathcal{A}$ with respect to each issuer, it submits $(\texttt{Rgs.Ok}, \texttt{sid}, \mathsf{U})$ 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 $(\mathtt{Rgs}, \mathtt{sid})$ to register U (based on internally run $\mathcal{A}$'s actions). Additionally, as previously detailed in section 4.1, the simulator simulates blind signature shares of honest issuers as $\sigma_{\mathsf{hon}}^{\mathsf{blind}}$ and, while emulating $\mathcal{F}_{\mathsf{B}}^{\mathsf{S}}$, forwards them to the malicious user U. $\mathcal{S}$ also submits $(\mathtt{Rgs.Ok}, \mathtt{sid}, \mathsf{U})$ to $\mathcal{F}_{\mathsf{PARS}}$.

Stablecoin Issuance.

(i) **Honest user U, honest custodian CUS, and no more than t malicious issuers.** The simulator $\mathcal{S}$ gets $(\mathtt{Iss}, \mathtt{sid}, \mathsf{SI.idn})$ from $\mathcal{F}_{\mathsf{PARS}}$. $\mathcal{S}$ which emulates the honest user U, honest custodian CUS, and honest issuers, supplies all the information that the real-world adversary observes to internally run the dummy adversary $\mathcal{A}$ as channel blockage (and information malicious issuers see). $\mathcal{S}$ submits the message $(\mathtt{Iss.Ok}, \mathtt{sid}, \mathsf{SI.idn}, \tilde{\chi})$ to $\mathcal{F}_{\mathsf{PARS}}$ based on its observations regarding the adversary's choices for message delivery. Here, $\tilde{\chi}$ is a dummy value.

(ii) **Malicious user U, honest custodian CUS, and no more than t malicious issuers.** $\mathcal{S}$ emulates the honest custodian CUS, and honest issuers. Upon receiving $(\mathtt{Iss}, \mathtt{sid}, \mathsf{SI.idn}, (\mathsf{U}, v))$ from $\mathcal{F}_{\mathsf{PARS}}$, $\mathcal{S}$ starts emulating honest CUS. After activating $\mathcal{A}$ by emulating CUS, and upon receiving the message from the malicious user U, the simulator $\mathcal{S}$ parses $\mathfrak{I} = (\sigma_{\mathsf{CUS}}^{\mathsf{RND}}, \sigma_{\mathbb{I}}^{\mathsf{RND}}, \mathsf{x}, \pi)$. While emulating $\mathcal{F}_{\mathsf{NIZK}}$, it sends $(\mathtt{Verify}, \mathtt{sid}, \mathsf{x}, \pi)$ to $\mathcal{A}$. Upon receiving $(\mathtt{Witness}, \mathtt{sid}, \mathsf{w})$ from $\mathcal{A}$ checks if the NIZK relation holds. If it holds, $\mathcal{S}$ follows the previously described procedure in section 4.1 to simulate the blind signature shares of honest issuers $\sigma_{\mathsf{hon}}^{\mathsf{blind}}$. $\mathcal{S}$ then submits these shares $\sigma_{\mathsf{hon}}^{\mathsf{blind}}$ to $\mathcal{A}$ by emulating $\mathcal{F}_{\mathsf{B}}^{\mathsf{SA}}$. $\mathcal{S}$ parses $\mathsf{x} = (\bar{\varkappa}, \mathtt{sn}, \mathtt{com}, \mathtt{com}_1, \mathtt{com}_2, \mathtt{com}_3, \mathtt{com}_4, h, \varkappa, \mathtt{tag}, \tilde{\chi}_t, \mathsf{pk}_{\mathsf{A}})$ from $\mathfrak{I}$, and submits the message $(\mathtt{Iss.Ok}, \mathtt{sid}, \mathsf{SI.idn}, \tilde{\chi})$ to $\mathcal{F}_{\mathsf{PARS}}$ where $\tilde{\chi} \leftarrow \tilde{\chi}_t$.

Stablecoin Transfer.

(i) **Honest sender $\mathsf{U_s}$, and no more than t malicious issuers.** The simulator receives $(\mathtt{Gen.ST}, \mathtt{sid}, \mathsf{ST.idn})$ from $\mathcal{F}_{\mathsf{PARS}}$. While emulating the roles of the honest sender $\mathsf{U_s}$ and honest issuers, $\mathcal{S}$ provides all the information that the real-world adversary observes to internally run adversary $\mathcal{A}$. Based on its observations concerning the adversary's choices for message delivery on the issuers' side, $\mathcal{S}$ sends the message $(\mathtt{Gen.ST.Ok}, \mathtt{sid}, \mathsf{ST.idn}, \phi)$ to $\mathcal{F}_{\mathsf{PARS}}$ (ϕ serves as two ciphertexts that are encrypting dummy values as previously described in section 4.1).

(ii) **Malicious sender $\mathsf{U_s}$, and no more than t malicious issuers.** Upon receiving the message from the malicious $\mathsf{U_s}$, $\mathcal{S}$, while emulating the roles of honest issuers, parses $\mathfrak{T} = (\sigma_{\mathbb{I}}^{\mathsf{RND}}, \mathsf{x}, \pi)$. While emulating $\mathcal{F}_{\mathsf{NIZK}}$, it sends $(\mathtt{Verify}, \mathtt{sid}, \mathsf{x}, \pi)$ to $\mathcal{A}$. Upon receiving $(\mathtt{Witness}, \mathtt{sid}, \mathsf{w})$ from $\mathcal{A}$, $\mathcal{S}$ checks if the associated relation holds or not. If it holds, $\mathcal{S}$ follows the previously described procedure in section 4.1 to simulate the blind signature shares of honest issuers $\sigma_{\mathsf{hon}}^{\mathsf{blind}}$. It then submits these shares to $\mathcal{A}$ by emulating $\mathcal{F}_{\mathsf{B}}^{\mathsf{SA}}$. Additionally, knowing the extracted witness (the identity of the receiver $\mathsf{U_r}$, and transaction value v), $\mathcal{S}$ sends $(\mathtt{Gen.ST}, \mathtt{sid}, \mathsf{U_r}, v)$ to $\mathcal{F}_{\mathsf{PARS}}$ on behalf of the malicious sender $\mathsf{U_s}$. $\mathcal{S}$ parses $\mathsf{x} = (\mathtt{com}, \mathtt{com}_1, \mathtt{com}_2, \mathtt{com}_3, \mathtt{com}_4, h, \varkappa, \mathtt{tag}, \chi_1, \chi_2, \chi_3, \chi_4, W)$, and finally, submits the message $(\mathtt{Gen.ST.Ok}, \mathtt{sid}, \mathsf{ST.idn}, \phi)$ 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 $(\texttt{Clm.ST}, \texttt{sid}, \mathsf{SC.idn})$ from $\mathcal{F}_{\mathsf{PARS}}$. While emulating the roles of the honest $\mathsf{U_r}$ and 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 $(\texttt{Clm.ST.Ok}, \texttt{sid}, \mathsf{SC.idn})$ to $\mathcal{F}_{\mathsf{PARS}}$.

(ii) **Malicious receiver $\mathsf{U_r}$, and no more than t malicious issuers.** Upon receiving the message from the malicious $\mathsf{U_r}$, the simulator, while emulating the roles of honest issuers, parses $\mathfrak{D} = (\sigma_{\mathbb{I}}^{\mathsf{RND}}, \mathsf{x}, \pi, \chi_3)$. While emulating $\mathcal{F}_{\mathsf{NIZK}}$, $\mathcal{S}$ submits $(\texttt{Verify}, \texttt{sid}, \mathsf{x}, \pi)$ to $\mathcal{A}$. Upon receiving $(\texttt{Witness}, \texttt{sid}, \mathsf{w})$ from $\mathcal{A}$, it checks the correctness of the associated relation. $\mathcal{S}$ follows the previously described procedure in section 4.1 to simulate the blind signature shares of honest issuers once it receives $(\texttt{Clm.ST.Issuers.Ok}, \texttt{sid}, \mathsf{SC.idn})$ from $\mathcal{F}_{\mathsf{PARS}}$ (we highlight that $\mathcal{S}$ parses $\mathsf{x} = (\texttt{com}, \texttt{com}_1, \texttt{com}_2, \texttt{com}_3, \texttt{com}_4, h, \varkappa, \mathfrak{tag}, \chi_1, \chi_4)$, and calls $\mathcal{F}_{\mathsf{PARS}}$ with $(\texttt{Clm.ST}, \texttt{sid}, \phi)$ where $\phi = (\chi_1, \chi_3, \chi_4)$, $^{-}\chi_3$ is parsed using $\mathfrak{D}^{-}$ on behalf of the malicious $\mathsf{U_r}$; $\mathcal{S}$ receives $(\texttt{Clm.ST}, \texttt{sid}, \mathsf{SC.idn})$ from $\mathcal{F}_{\mathsf{PARS}}$). $\mathcal{S}$ submits simulated signature shares $\sigma_{\mathsf{hon}}^{\mathsf{blind}}$ to $\mathcal{A}$ by emulating $\mathcal{F}_{\mathsf{Ch}}$. Finally, $\mathcal{S}$ submits the message $(\texttt{Clm.ST.Issuer.Ok}, \texttt{sid}, \mathsf{SC.idn})$ to $\mathcal{F}_{\mathsf{PARS}}$ (based on its observation regarding $\mathcal{A}$ blockage).

Stablecoin Burn.

(i) **Honest user U, and no more than t malicious issuers.** The simulator receives $(\texttt{Brn}, \texttt{sid}, \ell, \mathsf{SB.idn})$ from $\mathcal{F}_{\mathsf{PARS}}$.To avoid repetition, we skip describing steps similar to those for honest users explained earlier in item (i) of previous protocols. $\mathcal{S}$ sends the message $(\texttt{Brn.Ok}, \texttt{sid}, \mathsf{SB.idn}, \eta, \tilde{\chi})$ to $\mathcal{F}_{\mathsf{PARS}}$ where η, and $\tilde{\chi}$ are encryptions of dummy values generated by $\mathcal{S}$ (section 4.1).

(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 honest issuers, parses $\mathfrak{B} = (\sigma_{\mathbb{I}}^{\mathsf{RND}}, \mathsf{x}, \pi)$. While emulating $\mathcal{F}_{\mathsf{NIZK}}$, it sends $(\texttt{Verify}, \texttt{sid}, \mathsf{x}, \pi)$ to $\mathcal{A}$. $\mathcal{S}$ checks the validity of relation upon receiving $(\texttt{Witness}, \texttt{sid}, \mathsf{w})$ from $\mathcal{A}$, and it follows the procedure in section 4.1 to simulate the blind signature shares of 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 $(\texttt{Brn}, \texttt{sid}, v, \ell)$ to $\mathcal{F}_{\mathsf{PARS}}$. Finally, $\mathcal{S}$ parses $\mathsf{x} = (\texttt{com}, \texttt{com}_1, \texttt{com}_2, \texttt{com}_3, \texttt{com}_4, h, \varkappa, \mathfrak{tag}, \chi_1, \chi_2, \tilde{\chi}_t, \mathsf{pk_A}, \ell)$., and submits the message $(\texttt{Brn.Ok}, \texttt{sid}, \mathsf{SB.idn}, \eta, \tilde{\chi})$ to $\mathcal{F}_{\mathsf{PARS}}$ where $\tilde{\chi} \leftarrow \tilde{\chi}_t$, and $\eta = (\chi_1, \chi_2)$.

Proof of Burn.

(i) **Honest user U, and honest custodian CUS.** The simulator receives $(\texttt{PoB}, \texttt{sid}, \ell, \mathsf{PB.idn})$ from $\mathcal{F}_{\mathsf{PARS}}$ and leaks to $\mathcal{A}$ whatever real-world adversary sees as the leakage of $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{SSA}}$. $\mathcal{S}$ sends the message $(\texttt{PoB.Ok}, \texttt{sid}, \mathsf{PB.idn})$ to $\mathcal{F}_{\mathsf{PARS}}$ based on its observation regarding channel blockage by $\mathcal{A}$.

(ii) **Malicious user U, and honest custodian CUS.** Upon receiving the message from the malicious user, the simulator $\mathcal{S}$ starts simulating the honest CUS. $\mathcal{S}$ parses $\mathfrak{V} = (\mathsf{x}, \pi)$, and $\mathsf{x} = (\eta, v, \ell)$. $\mathcal{S}$, emulating $\mathcal{F}_{\mathsf{NIZK}}$ checks the correctness of π (by extracting witness). $\mathcal{S}$ also calls $\mathcal{F}_{\mathsf{PARS}}$ with $(\texttt{PoB}, \texttt{sid}, \eta, \ell)$ and

gets $(\texttt{PoB}, \texttt{sid}, \ell, \mathsf{PB.idn})$. Upon receiving $(\texttt{PoB.CUS.Ok}, \texttt{sid}, \mathsf{PB.idn})$ from $\mathcal{F}_{\mathsf{PARS}}$, $\mathcal{S}$ starts emulating the honest custodian CUS as Π_{PARS}.

Reserve Audit.

Honest auditor AUD. Upon receiving $(\texttt{Audit.End}, \texttt{sid}, b)$ from $\mathcal{F}_{\mathsf{PARS}}$, $\mathcal{S}$ confirms the delivery of $(\texttt{Audit.End}, \texttt{sid}, b)$ to dummy AUD by submitting its approval to $\mathcal{F}_{\mathsf{PARS}}$ (based on its observation regarding internally run $\mathcal{A}$).

5 PARScoin Performance

5.1 Zero-knowledge Proof Efficiency

Zero-knowledge Proofs (ZKPs) are theoretically applicable to all languages within the complexity class NP, as demonstrated in [24]. 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 [19]), (ii) zk-SNARKs (Zero-Knowledge Succinct Non-Interactive Argument of Knowledge) [26]. 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) [22]). 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.

We follow the approach of PEReDi [29] and in our cryptographic construction only prove statements that can be efficiently represented as algebraic discrete logarithm equations[12]. They also provide benchmarks that also should be good approximations for our construction in terms of both **time complexity** and **communication cost**. 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 [28,41].

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

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 by employing the Fiat-Shamir transform technique [7]. 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 [34], which relies on Fischlin's transform [30]. Alternatively, one can construct a UC-secure NIZK (a.k.a. simulation-extractable NIZK [25]) by leveraging a simulation-sound NIZK scheme and a (perfectly correct) CPA-secure encryption scheme [31].

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.

Proof of Knowledge of Discrete Log. $\mathsf{R}(\mathsf{x},\mathsf{w}) = \{y = g^x\}$, where $\mathsf{x} = (y, g)$, and $\mathsf{w} = x$.

1. P computes $a \leftarrow g^\theta$ for $\theta \xleftarrow{\$} \mathbb{Z}_q^*$. Sends a to V.
2. V selects $c \xleftarrow{\$} \mathbb{Z}_q$. Sends c to P.
3. P computes $z = \theta + cx (\mathrm{mod} q)$. Sends z to V.
4. V checks if $a = g^z y^{-c}$ holds. If holds, the verifier accepts.

Proof of Committed Values' Multiplicative Relation. In our scheme, users' accounts is of the form $\mathsf{account} = (\mathsf{B}, \mathsf{sk}, \mathsf{sk}^x)$ and for double-spending prevention they need to provide $\mathtt{tag}$ values that are of the form $g^{(\mathsf{sk}^x)\cdot\mathsf{sk}}$. Users are supposed to prove $\mathtt{tag}$ is well-formed. To do so, we introduce the following relation.

$\mathsf{R}(\mathsf{x},\mathsf{w}) = \{\mathtt{com}_1 = g^{a_1}h^{r_1}, \mathtt{com}_2 = g^{a_2}h^{r_2}, \mathtt{com}_3 = g^{a_3}h^{r_3} = g^{a_1 a_2}h^{r_3}\}$, where $\mathsf{x} = (g, h, \mathtt{com}_1, \mathtt{com}_2, \mathtt{com}_3)$, and $\mathsf{w} = (a_1, a_2, a_3, r_1, r_2, r_3)$. We have to also prove that $\mathtt{com}_3 = \mathtt{com}_1^{a_2} h^r$ where $r = r_3 - r_1 a_2 \bmod q$.

1. P computes $v_i = g^{\theta_i} h^{R_i}$ for $i = 1, 2, 3, v = \mathtt{com}_1^{\theta_2} h^R$ for $(\theta_i, R_i, \theta, R) \xleftarrow{\$} \mathbb{Z}_q$. Sends $(\{v_i\}, v)$ for $i = 1, 2, 3$ to V.
2. V chooses a challenge $c \xleftarrow{\$} \mathbb{Z}_{2^k}$ (k is fixed where $2^k < q$). Sends c to P.
3. P computes $s_i = \theta_i - ca_i, t_i = R_i - cr_i, t = R - cr$. Sends the tuple (s_i, t_i) for $i = 1, 2, 3$ and t to V.
4. V Checks if $v_i = (\mathtt{com}_i)^c g^{s_i} h^{t_i}$ for $i = 1, 2, 3$ and $v = \mathtt{com}_3^c \mathtt{com}_1^{s_2} h^t$ hold. V accepts if all four equations hold.

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 [11] 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: $\mathsf{R}(\mathsf{x}, \mathsf{w}) = \{\mathtt{com} = g^m \cdot h^r \wedge m \in [0, 2^n - 1]\}$ 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 effectively implemented using (non-interactive) Sigma protocols together with range proofs.

5.2 Signature and Encryption Efficiency

The underlying signature scheme is described in section A.4. 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 [42].

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 $\mathbb{G}$).

Acknowledgements

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

References

1. Aztec, https://aztec.network/
2. Al-Naji, N., Chen, J., Diao, L.: Basis: a price-stable cryptocurrency with an algorithmic central bank. Basis. io (2017)
3. Ampleforth: Ampleforth documents, https://docs.ampleforth.org/
4. Androulaki, E., Camenisch, J., De Caro, A., Dubovitskaya, M., Elkhiyaoui, K., Tackmann, B.: Privacy-preserving auditable token payments in a permissioned blockchain system. Cryptology ePrint Archive, Report 2019/1058 (2019), https://eprint.iacr.org/2019/1058
5. Bains, P., Ismail, A., Melo, F., Sugimoto, N.: Regulating the crypto ecosystem: the case of stablecoins and arrangements. International Monetary Fund (2022)
6. Bank, E.C.: Exploring anonymity in central bank digital currencies (December 2019), https://www.ecb.europa.eu/paym/intro/publications/pdf/ecb.mipinfocus191217.en.pdf
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. pp. 459–474. IEEE Computer Society Press (May 2014). 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. 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)
11. 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)
12. 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)
13. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: 42nd FOCS. pp. 136–145. IEEE Computer Society Press (Oct 2001). https://doi.org/10.1109/SFCS.2001.959888
14. CCData: Stablecoins & cbdcs report (JULY 20, 2023), https://ccdata.io/reports/stablecoins-cbdcs-report-july-2023
15. CoinMarketCap: Top stablecoin tokens by market capitalization, https://coinmarketcap.com/view/stablecoin/
16. (ECB), E.C.B.: Ecb digital euro consultation ends with record level of public feedback (13 January 2021), https://www.ecb.europa.eu/press/pr/date/2021/html/ecb.pr210113~ec9929f446.en.html
17. ElGamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. In: Blakley, G.R., Chaum, D. (eds.) CRYPTO'84. LNCS, vol. 196, pp. 10–18. Springer, Heidelberg (Aug 1984)
18. Ethereum: Whitepaper (2022), https://ethereum.org/en/whitepaper/
19. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) CRYPTO'86. LNCS, vol. 263, pp. 186–194. Springer, Heidelberg (Aug 1987). https://doi.org/10.1007/3-540-47721-7_12

20. 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)
21. Garman, C., Green, M., Miers, I.: Accountable privacy for decentralized anonymous payments. In: Grossklags, J., Preneel, B. (eds.) FC 2016. LNCS, vol. 9603, pp. 81–98. Springer, Heidelberg (Feb 2016)
22. Gennaro, R., Gentry, C., Parno, B., Raykova, M.: Quadratic span programs and succinct NIZKs without PCPs. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 626–645. Springer, Heidelberg (May 2013). https://doi.org/10.1007/978-3-642-38348-9_37
23. Gennaro, R., Jarecki, S., Krawczyk, H., Rabin, T.: Secure distributed key generation for discrete-log based cryptosystems. In: Stern, J. (ed.) EUROCRYPT'99. LNCS, vol. 1592, pp. 295–310. Springer, Heidelberg (May 1999). https://doi.org/10.1007/3-540-48910-X_21
24. 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)
25. Groth, J.: Simulation-sound NIZK proofs for a practical language and constant size group signatures. In: Lai, X., Chen, K. (eds.) ASIACRYPT 2006. LNCS, vol. 4284, pp. 444–459. Springer, Heidelberg (Dec 2006). https://doi.org/10.1007/11935230_29
26. Groth, J.: Short pairing-based non-interactive zero-knowledge arguments. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 321–340. Springer, Heidelberg (Dec 2010). https://doi.org/10.1007/978-3-642-17373-8_19
27. 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)
28. Guillou, L.C., Quisquater, J.J.: A practical zero-knowledge protocol fitted to security microprocessor minimizing both trasmission and memory. In: Günther, C.G. (ed.) EUROCRYPT'88. LNCS, vol. 330, pp. 123–128. Springer, Heidelberg (May 1988). https://doi.org/10.1007/3-540-45961-8_11
29. 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)
30. Kondi, Y., abhi shelat: Improved straight-line extraction in the random oracle model with applications to signature aggregation. Cryptology ePrint Archive, Paper 2022/393 (2022), https://eprint.iacr.org/2022/393, https://eprint.iacr.org/2022/393
31. 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)
32. Liao, G.Y., Caramichael, J.: Stablecoins: Growth potential and impact on banking (2022)
33. 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)

34. Lysyanskaya, A., Rosenbloom, L.N.: Universally composable sigma-protocols in the global random-oracle model. Cryptology ePrint Archive, Paper 2022/290 (2022), https://eprint.iacr.org/2022/290, https://eprint.iacr.org/2022/290
35. MakerDAO: The maker protocol: Makerdao's multi-collateral dai (mcd) system, https://makerdao.com/en/whitepaper/#abstract
36. 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)
37. Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. In: Annual international cryptology conference. pp. 129–140. Springer (1991)
38. Pointcheval, D., Sanders, O.: Short randomizable signatures. Cryptology ePrint Archive, Paper 2015/525 (2015), https://eprint.iacr.org/2015/525
39. Q.ai: What really happened to luna crypto? (2022), https://www.forbes.com/sites/qai/2022/09/20/what-really-happened-to-luna-crypto/
40. Rial, A., Piotrowska, A.M.: Security analysis of coconut, an attribute-based credential scheme with threshold issuance. Cryptology ePrint Archive (2022)
41. Schnorr, C.P.: Efficient signature generation by smart cards. Journal of Cryptology **4**(3), 161–174 (Jan 1991). https://doi.org/10.1007/BF00196725
42. 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)
43. Synthetix: Whitepaper (2022), https://docs.synthetix.io/synthetix-protocol/readme
44. Tether: Whitepaper, https://tether.to/en/whitepaper
45. of the Treasury (gov), U.D.: Report on stablecoins. Tech. rep. (November 2021), https://home.treasury.gov/system/files/136/StableCoinReport_Nov1_508.pdf
46. USDCoin: Centre whitepaper (03/04/2021), https://whitepaper.io/coin/usd-coin
47. Wüst, K., Kostiainen, K., Delius, N., Capkun, S.: Platypus: A central bank digital currency with unlinkable transactions and privacy-preserving regulation. In: Yin, H., Stavrou, A., Cremers, C., Shi, E. (eds.) Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022. pp. 2947–2960. ACM (2022). https://doi.org/10.1145/3548606.3560617, https://doi.org/10.1145/3548606.3560617

A Cryptographic Primitives and Security Definitions

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

A.1 Bilinear Maps

Definition 1. *To define a bilinear map, consider three multiplicative cyclic groups* $(\mathbb{G}, \tilde{\mathbb{G}}, \mathbb{G}_t)$*, all of prime order* p*.* $\mathbb{G}$*, and* $\tilde{\mathbb{G}}$ *are called base groups and* $\mathbb{G}_t$ *is called target group. Now, we introduce a map denoted as* $e : \mathbb{G} \times \tilde{\mathbb{G}} \to \mathbb{G}_t$ *with the following distinctive properties:*

1. ***Bilinearity***: *For any $g \in \mathbb{G}$, and $\tilde{g} \in \tilde{\mathbb{G}}$, and integers x and y drawn from the finite field $\mathbb{F}_p$, the map satisfies the equation $e(g^x, \tilde{g}^y) = e(g, \tilde{g})^{xy}$.*
2. ***Non-degeneracy***: *For any generators $g \in \mathbb{G}$ and $\tilde{g} \in \tilde{\mathbb{G}}$, the pairing $e(g, \tilde{g})$ has the property of generating the entire group $\mathbb{G}_t$.*
3. ***Efficiency***: *Bilinear maps must be accompanied by efficient algorithms. Specifically, there exists an efficient algorithm denoted as $\mathcal{G}(1^\lambda)$, which outputs the pairing group setup $(p, \mathbb{G}, \tilde{\mathbb{G}}, \mathbb{G}_t, e, g, \tilde{g})$. Additionally, efficient algorithms should be available to compute $e(g, \tilde{g})$ for any given elements $g \in \mathbb{G}$ and $\tilde{g} \in \tilde{\mathbb{G}}$. It is noteworthy that in the case of type 3 pairings, the groups $\mathbb{G}$ and $\tilde{\mathbb{G}}$ are distinct, and there exists no efficiently computable homomorphism $f : \tilde{\mathbb{G}} \to \mathbb{G}$.*

A.2 ElGamal Encryption Scheme

Definition 2. *The ElGamal encryption scheme, as introduced by ElGamal [17], is a public key cryptosystem that relies on the computational intractability of the discrete logarithm problem. It consists of three fundamental algorithms computations are* $\mod p$*:*

1. *Key Generation. Assuming that p is a large prime and g is a generator of group $\mathbb{Z}_p^*$, the key generation algorithm is performed as follows.*
   - *Randomly select a secret key $\mathsf{sk} \xleftarrow{\$} \mathbb{Z}_p^*$.*
   - *Compute the corresponding public key as $\mathtt{pk} = g^{\mathtt{sk}}$.*
   - *Make the parameters $(g, p, \mathtt{pk})$ publicly available.*
2. *Encryption. To encrypt a message $m \in \mathbb{Z}_p$, the encryption algorithm is performed as follows.*
   - *Randomly select $k \xleftarrow{\$} \mathbb{Z}_p^*$.*
   - *The ciphertext $\chi = (C_1, C_2) \mod p$ is computed as $C_1 = g^k$, and $C_2 = \mathtt{pk}^k \cdot m$.*
3. *Decryption. Given a ciphertext $\chi = (C_1, C_2)$, the decryption algorithm is performed as follows.*
   - *Compute the message m as: $m = C_2 / C_1^{\mathsf{sk}}$.*

A.3 CPA Security of Public Key Encryption Scheme

Definition 3. *Let $\mathsf{PE} = (\mathsf{PE.Key.Gen}, \mathsf{Enc}, \mathsf{Dec})$ be a public key encryption scheme. The following security experiment is conducted between a Probabilistic Polynomial-Time (PPT) adversary $\mathcal{A}$ and a challenger. This security experiment $\mathsf{IND\text{-}CPA}^b_{\mathsf{PE}}(\mathcal{A}, \lambda)$ is parameterized by a bit $b \in \{0, 1\}$ and a security parameter λ.*

1. *The challenger randomly selects a key pair $(\mathtt{pk}, \mathsf{sk})$ by executing $\mathsf{PE.Key.Gen}(1^\lambda)$ and provides the public key $\mathtt{pk}$ to the adversary $\mathcal{A}$.*
2. *The adversary $\mathcal{A}$ submits two plaintext messages, denoted as (m_0, m_1), where $|m_0| = |m_1|$.*

3. *The challenger encrypts one of the plaintexts based on the bit b to create a ciphertext $c_b = \mathsf{Enc}_{\mathsf{pk}}(m_b)$ and provides this ciphertext c_b to the adversary $\mathcal{A}$.*
4. *The adversary $\mathcal{A}$ outputs a bit b', indicating its guess of the value of b. If $\mathcal{A}$ decides to abort without producing an output, the output bit b' is set to 0.*

The adversary's advantage is defined as:

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

The public key encryption scheme PE *is considered* IND-CPA *secure if for any PPT adversary $\mathcal{A}$, the advantage $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{IND\text{-}CPA}}$ is negligible in the security parameter λ:*

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

A.4 Randomizable-Blind Signature

The Coconut framework, as presented in [42], constitutes an optional declaration credential construction that supports distributed threshold issuance. It relies on the Pointcheval-Sanders signature scheme [38] as its foundation which is as follows.

Definition 4. *Pointcheval-Sanders Signature. Pointcheval-Sanders signature scheme [38] is existentially unforgeable and randomizable which consists of the following algorithms:*

1. $\mathsf{KeyGen}(1^\lambda, q)$*:*
   - *Execute $\mathcal{G}(1^\lambda)$ that outputs the pairing group setup $\mathsf{par} = (p, \mathbb{G}, \tilde{\mathbb{G}}, \mathbb{G}_t, e, g, \tilde{g})$.*
   - *Select secret key $\mathsf{sk} = (x, \{y_i\}_{i=1}^q) \xleftarrow{\$} \mathbb{Z}_p^{q+1}$.*
   - *Set the public key $\mathsf{pk} = (\mathsf{par}, \alpha, \{\beta_i\}_{i=1}^q) \leftarrow (\mathsf{par}, \tilde{g}^x, \{\tilde{g}^{y_i}\}_{i=1}^q)$.*
2. $\mathsf{Sign}(\mathsf{sk}, \{m_i\}_{i=1}^q)$*:*
   - *Select $r \xleftarrow{\$} \mathbb{Z}_p$, and set $h \leftarrow g^r$.*
   - *Output the signature $\sigma = (h, s) \leftarrow (h, h^{(x+\Sigma_{i=1}^q (y_i m_i))})$*
3. $\mathsf{VerifySig}(\mathsf{pk}, \sigma, \{m_i\}_{i=1}^q)$*:*
   - *Output 1 if $h \neq 1$ and $e(h, \alpha \cdot \prod_{i=1}^q \beta_i^{m_i}) = e(s, \tilde{g})$. Else, output 0.*

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 examination of Coconut's security properties was conducted by Rial et al. [40]. In their analysis, they introduced an ideal functionality that captures all the essential security characteristics of a threshold randomizable-blind signature scheme. Subsequently, Rial et al. [40] proposed a new construction that is based on Coconut, albeit with a few modifications aimed at realizing the ideal functionality. In our construction, we employ an enhanced iteration of the Coconut system [42]. This improved iteration incorporates several modifications to the original framework. We have enhanced the construction

by introducing a model to represent the communication between the user and the signing party. Additionally, we have seamlessly integrated the necessary non-interactive zero-knowledge (NIZK) proofs, which are a fundamental component of the Threshold Randomizable-Blind Signature (TRB.Sig) scheme, into our construction. Further details regarding these modifications are expounded upon in section 3. The randomizable-blind signature scheme can be characterized by its adherence to three fundamental security properties:

1. Unforgeability: This property ensures that it is computationally infeasible for a malicious user, who may be compromised or corrupted, to convincingly persuade an honest verifier that they possess a valid signature when, in reality, they do not.
2. Blindness: The concept of blindness within the scheme guarantees that a corrupted signer, even when attempting to manipulate the protocol during the signature issuance process (Issue.Sig), cannot gain any knowledge regarding the content of the message m other than the fact that it satisfies a particular predicate.
3. Unlinkability: Within the TRB.Sig scheme, the notion of unlinkability asserts that it is computationally infeasible for a compromised signer or verifier to acquire any information about the message m except for its adherence to a predefined predicate. Furthermore, it ensures that no linkage can be established between the execution of Rand.Sig and either another Rand.Sig execution or the execution of Issue.Sig.

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

Non-Threshold Randomizable-Blind Signature.

Definition 5. *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.*

1. $(\mathtt{pk}, \mathsf{sk}) \leftarrow \mathsf{RB.Sig.KeyGen}(1^\lambda)$
   (a) *Run* $(p, \mathbb{G}, \tilde{\mathbb{G}}, \mathbb{G}_t, e, g, \tilde{g}) \leftarrow \mathcal{G}(1^\lambda)$ *and pick* q *random generators* $\{h_\kappa\}_{\kappa=1}^q \leftarrow \mathbb{G}$ *and set the parameters* $\mathsf{par} \leftarrow (p, \mathbb{G}, \tilde{\mathbb{G}}, \mathbb{G}_t, e, g, \tilde{g}, \{h_\kappa\}_{\kappa=1}^q)$
   (b) *Set the secret key as* $\mathsf{sk} = (x, \{y_\kappa\}_{\kappa=1}^q) \xleftarrow{\$} \mathbb{Z}_p$.
   (c) *Set* $\mathtt{pk} = (\mathsf{par}, \tilde{\alpha}, \{\beta_\kappa, \tilde{\beta}_\kappa\}_{\kappa=1}^q) \leftarrow (\mathsf{par}, \tilde{g}^x, \{g^{y_\kappa}, \tilde{g}^{y_\kappa}\}_{\kappa=1}^q)$.
2. $\mathsf{Issue.Sig}$ *protocol consists three following algorithms* ($\mathsf{TRB.Sig.Blinding}$, $\mathsf{TRB.Sig.Blinding}$, $\mathsf{TRB.Sig.Unblinding}$).
   2.1. $(\mathtt{x}_1, \pi_1, \{o_\kappa\}_{\kappa=1}^q) \leftarrow \mathsf{TRB.Sig.Blinding}(\mathsf{message}, \phi)$ *algorithm is run by user* U *which is as follows.*
   (a) *Parse* $\mathsf{message} = \{m_\kappa\}_{\kappa=1}^q \in \mathbb{Z}_p$. *Pick* $o \xleftarrow{\$} \mathbb{Z}_p$ *and compute* $\mathtt{com} = g^o \cdot \prod_{\kappa=1}^q h_\kappa^{m_\kappa}$ *and send* $\mathtt{com}$ *to* $\mathcal{F}_{\mathrm{RO}}$ *and receive* h *from* $\mathcal{F}_{\mathrm{RO}}$.
   (b) *Compute commitments to each of the messages. For* $\{\kappa\}_{\kappa=1}^q$, *pick* $o_\kappa \xleftarrow{\$} \mathbb{Z}_p$ *and set* $\mathtt{com}_\kappa = g^{o_\kappa} \cdot h^{m_\kappa}$.

(c) *Compute a* NIZK *proof* π_1 *for the following relation:* $\mathsf{R}(\mathtt{x}_1, \mathtt{w}_1) = \mathsf{NIZK}\{\mathtt{com} = g^o \cdot \prod_{\kappa=1}^{q} h_\kappa^{m_\kappa} \wedge \{\mathtt{com}_\kappa = g^{o_\kappa} \cdot h^{m_\kappa}\}_{\kappa=1}^{q} \wedge \phi(\{m_\kappa\}_{\kappa=1}^{q} = 1)\}$ *where* ϕ *is a statement that* message *satisfies, the statement is* $\mathtt{x}_1 = (\mathtt{com}, \{\mathtt{com}_\kappa\}_{\kappa=1}^{q}, h, \phi)$, *and the witness is* $\mathtt{w}_1 = (\{m_\kappa\}_{\kappa=1}^{q}, o, \{o_\kappa\}_{\kappa=1}^{q})$.

2.2. $\sigma^{\mathsf{blind}} \leftarrow \mathsf{TRB.Sig.Signing}(\mathsf{sk}, \pi_1, \mathtt{x}_1)$ *algorithm is run by the signer which is as follows.*

(a) *Send* com *to* $\mathcal{F}_{\mathrm{RO}}$ *and receive* h' *from* $\mathcal{F}_{\mathrm{RO}}$. *Abort if* $h \neq h'$ *or* π_1 *is not correct.*

(b) *Compute* $c = h^x \cdot \prod_{\kappa=1}^{q} \mathtt{com}_\kappa^{y_\kappa}$ *and set the blind signature* $\sigma^{\mathsf{blind}} = (h, h^x \cdot \prod_{\kappa=1}^{q} \mathtt{com}_\kappa^{y_\kappa})$

2.3. $\sigma \leftarrow \mathsf{TRB.Sig.Unblinding}(\{o_\kappa\}_{\kappa=1}^{q}, \sigma^{\mathsf{blind}})$ *algorithm is run by user* U *which is as follows.*

(a) *Parse* σ^{blind} *as* (h', c). *Abort if* $h \neq h'$.

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

(c) *Abort if* $e(h, \tilde{\alpha} \cdot \prod_{\kappa=1}^{q} \tilde{\beta}_\kappa^{m_\kappa}) = e(s, \tilde{g})$ *does not hold.*

3. $(\sigma^{\mathsf{RND}}, \pi_2, \mathtt{x}_2) \leftarrow \mathsf{Rand.Sig}(\varphi, \sigma, \{m_\kappa\}_{\kappa=1}^{q}, \mathtt{pk})$

User U *does the following:*

(a) *Parse* $\sigma = (h, s)$, *pick* $r \xleftarrow{\$} \mathbb{Z}_p$ *and* $r' \xleftarrow{\$} \mathbb{Z}_p$.

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

(c) *Parse* $\mathsf{message} = \{m_\kappa\}_{\kappa=1}^{q}$. *Compute* $\varkappa \leftarrow \tilde{\alpha} \cdot \prod_{\kappa=1}^{q} \tilde{\beta}_\kappa^{m_\kappa} \cdot \tilde{g}^r$.

(d) *Compute the* NIZK *proof* π_2 *for the relation:* $\mathsf{R}(\mathtt{x}_2, \mathtt{w}_2) = \mathsf{NIZK}\{\varkappa = \tilde{\alpha} \cdot \prod_{\kappa=1}^{q} \tilde{\beta}_\kappa^{m_\kappa} \cdot \tilde{g}^r \wedge \varphi(\{m_\kappa\}_{\kappa=1}^{q}) = 1\}$ *where* φ *is a statement that* message *satisfies. The statement is* $\mathtt{x}_2 = (\varkappa, \varphi)$, *and the witness is* $\mathtt{w}_2 = (\{m_\kappa\}_{\kappa=1}^{q}, r)$.

4. $(1, 0) \leftarrow \mathsf{Verify.Sig}(\sigma^{\mathsf{RND}}, \pi_2, \mathtt{x}_2, \mathtt{pk})$

The verifier does the following:

(a) *Parse* $\mathtt{x}_2 = (\varkappa, \varphi)$, *and* $\sigma^{\mathsf{RND}} = (h', s')$. *Output 0 if* $h' = 1$ *or if* $e(h', \varkappa) = e(s', \tilde{g})$ *does not hold.*

(b) *Verify* π_2 *and output 0 if the proof is not correct. Else, output 1.*

Threshold Randomizable-Blind Signature (Modified Coconut).

Definition 6. *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.*

TRB.Sig.KeyGen *algorithm can be replaced by a distributed key generation protocol using Gennaro et al.'s distributed key generation protocol [23].*

1. $(\{(\mathtt{pk}_j, \mathsf{sk}_j)\}_{j=1}^{N}, \mathtt{pk}) \leftarrow \mathsf{TRB.Sig.KeyGen}(1^\lambda, N, \Gamma)$

(a) *Run* $(p, \mathbb{G}, \tilde{\mathbb{G}}, \mathbb{G}_t, e, g, \tilde{g}) \leftarrow \mathcal{G}(1^\lambda)$ *and pick* q *random generators* $\{h_\kappa\}_{\kappa=1}^{q} \leftarrow \mathbb{G}$ *and set the parameters* $\mathsf{par} \leftarrow (p, \mathbb{G}, \tilde{\mathbb{G}}, \mathbb{G}_t, e, g, \tilde{g}, \{h_\kappa\}_{\kappa=1}^{q})$.

(b) *Choose* $(q+1)$ *polynomials* $(v, \{w_\kappa\}_{\kappa=1}^{q})$ *of degree* $(\Gamma - 1)$ *with random coefficients in* $\mathbb{Z}_p$ *and set* $(x, \{y_\kappa\}_{\kappa=1}^{q}) \leftarrow (v(0), \{w_\kappa(0)\}_{\kappa=1}^{q})$.

(c) *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* $\mathtt{pk}_j$ *of each signer* I_j *as* $\mathtt{pk}_j = (\tilde{\alpha}_j, \{\beta_{j,\kappa}, \tilde{\beta}_{j,\kappa}\}_{\kappa=1}^{q}) \leftarrow (\tilde{g}^{x_j}, \{g^{y_{j,\kappa}}, \tilde{g}^{y_{j,\kappa}}\}_{\kappa=1}^{q})$

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

2. Issue.Sig *protocol consists three following algorithms* (TRB.Sig.Blinding, TRB.Sig.Signing, TRB.Sig.Unblinding).

2.1. $(\mathtt{x}_1, \pi_1, \{o_\kappa\}_{\kappa=1}^{q}) \leftarrow$ TRB.Sig.Blinding$(\mathsf{message}, \phi)$ *algorithm is run by user* U *which is as follows:*

(a) Parse $\mathsf{message} = \{m_\kappa\}_{\kappa=1}^{q} \in \mathbb{Z}_p$. *Pick* $o \xleftarrow{\$} \mathbb{Z}_p$ *and compute* $\mathtt{com} = g^o \cdot \prod_{\kappa=1}^{q} h_\kappa^{m_\kappa}$ *and send* com *to* $\mathcal{F}_{\mathrm{RO}}$[13] *and receive* h *from* $\mathcal{F}_{\mathrm{RO}}$.

(b) Compute commitments to message. *For* $\{\kappa\}_{\kappa=1}^{q}$, *pick* $o_\kappa \xleftarrow{\$} \mathbb{Z}_p$ *and set* $\mathtt{com}_\kappa = g^{o_\kappa} \cdot h^{m_\kappa}$.

(c) Compute a NIZK *proof* π_1 *for the following relation:* $\mathsf{R}(\mathtt{x}_1, \mathtt{w}_1) = \mathsf{NIZK}\{\mathtt{com} = g^o \cdot \prod_{\kappa=1}^{q} h_\kappa^{m_\kappa} \wedge \{\mathtt{com}_\kappa = g^{o_\kappa} \cdot h^{m_\kappa}\}_{\kappa=1}^{q} \wedge \phi(\{m_\kappa\}_{\kappa=1}^{q} = 1)\}$ *where* ϕ *is a statement that* message *satisfies, the statement is* $\mathtt{x}_1 = (\mathtt{com}, \{\mathtt{com}_\kappa\}_{\kappa=1}^{q}, h, \phi)$, *and the witness is* $\mathtt{w}_1 = (\{m_\kappa\}_{\kappa=1}^{q}, o, \{o_\kappa\}_{\kappa=1}^{q})$.

2.2. $\sigma_j^{\mathsf{blind}} \leftarrow$ TRB.Sig.Signing$(\mathsf{sk}_j, \pi_1, \mathtt{x}_1)$ *algorithm is run by the signer* I_j *which is as follows:*

(a) Send com *to* $\mathcal{F}_{\mathrm{RO}}$ *and receive* h' *from* $\mathcal{F}_{\mathrm{RO}}$. *Abort if* $h \neq h'$ *or* π_1 *is not correct.*

(b) Compute $c_j = h^{x_j} \cdot \prod_{\kappa=1}^{q} \mathtt{com}_\kappa^{y_{j,\kappa}}$ *and set the blind signature share* $\sigma_j^{\mathsf{blind}} = (h, c_j)$.

2.3. $\sigma_j \leftarrow$ TRB.Sig.Unblinding$(\{o_\kappa\}_{\kappa=1}^{q}, \sigma_j^{\mathsf{blind}})$ *algorithm is run by user* U *which is as follows:*

(a) Parse $\sigma_j^{\mathsf{blind}}$ *as* (h', c_j). *Abort if* $h \neq h'$.

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

(c) Abort if $e(h, \tilde{\alpha}_j \prod_{\kappa=1}^{q} \tilde{\beta}_{j,\kappa}^{m_\kappa}) = e(s_j, \tilde{g})$ *does not hold.*

3. $\sigma_{\mathbb{I}} \leftarrow$ TRB.Sig.Agg$(\{\sigma_j\}_{j=1}^{\Gamma}, \mathtt{pk})$.
User U *does the following:*

(a) Define $S \in [1, N]$ *as a set of* Γ *indices of signers in* $\mathcal{I}$.

(b) For all $j \in S$, *evaluate the Lagrange basis polynomials at 0:* $l_j = [\prod_{i \in S, i \neq j}(i)/(i - j)] \bmod p$.

(c) For all $j \in S$, *take* $\sigma_j = (h, s_j)$ *and compute the signature* $\sigma_{\mathbb{I}} = (h, s) \leftarrow (h, \prod_{j \in S} s_j^{l_j})$.

(d) Abort if $e(h, \tilde{\alpha} \cdot \prod_{\kappa=1}^{q} \tilde{\beta}_\kappa^{m_\kappa}) = e(s, \tilde{g})$ *does not hold.*

4. $(\sigma_{\mathbb{I}}^{\mathsf{RND}}, \pi_2, \mathtt{x}_2) \leftarrow$ Rand.Sig$(\varphi, \sigma_{\mathbb{I}}, \{m_\kappa\}_{\kappa=1}^{q}, \mathtt{pk})$.
User U *does the following:*

(a) Parse $\sigma_{\mathbb{I}} = (h, s)$, *pick* $(r, r') \xleftarrow{\$} \mathbb{Z}_p$.

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

(c) Parse $\mathsf{message} = \{m_\kappa\}_{\kappa=1}^{q}$. *Compute* $\varkappa \leftarrow \tilde{\alpha} \cdot \prod_{\kappa=1}^{q} \tilde{\beta}_\kappa^{m_\kappa} \cdot \tilde{g}^r$

(d) Compute the NIZK *proof* π_2 *for the relation:* $\mathsf{R}(\mathtt{x}_2, \mathtt{w}_2) = \mathsf{NIZK}\{\varkappa = \tilde{\alpha} \cdot \prod_{\kappa=1}^{q} \tilde{\beta}_\kappa^{m_\kappa} \cdot \tilde{g}^r \wedge \varphi(\{m_\kappa\}_{\kappa=1}^{q}) = 1\}$ *where* φ *is a statement that* message *satisfies. The statement is* $\mathtt{x}_2 = (\varkappa, \varphi)$, *and the witness is* $\mathtt{w}_2 = (\{m_\kappa\}_{\kappa=1}^{q}, r)$.

[13] $\mathcal{F}_{\mathrm{RO}}$ denotes functionality of random oracle which is a black box that provides a truly random response from an output domain for every unique request.

5. $(1,0) \leftarrow \mathsf{Verify.Sig}(\sigma_{\mathbb{I}}^{\mathsf{RND}}, \pi_2, \mathtt{x}_2, \mathtt{pk})$
   The verifier does the following:
   (a) *Parse* $\mathtt{x}_2 = (\varkappa, \varphi)$, *and* $\sigma_{\mathbb{I}}^{\mathsf{RND}} = (h', s')$. *Output 0 if* $h' = 1$ *or if* $e(h', \varkappa) = e(s', \tilde{g})$ *does not hold.*
   (b) *Verify* π_2 *and output 0 if the proof is not correct. Else, output 1.*

A.5 Existential Unforgeability of Digital Signature Scheme

Definition 7. *Given a digital signature scheme* $\mathsf{Sig} = (\mathsf{Sig.Gen}, \mathsf{Sig.Sign}, \mathsf{Sig.Verify})$, *the Existential Unforgeability under Chosen-Message Attack (EUF-CMA) is formally characterized through a structured interactive scenario involving a Probabilistic Polynomial-Time (PPT) adversary* $\mathcal{A}$ *and a designated challenger. This scenario* $\mathsf{EUF\text{-}CMA}_{\mathsf{Sig}}(\mathcal{A}, \lambda)$ *can be succinctly described as follows:*

1. *The challenger initiates the process by randomly selecting a key pair* $(\mathsf{vk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{Sig.Gen}(1^\lambda)$. *Subsequently, the challenger securely conveys the verification key* vk *to the adversary* $\mathcal{A}$, *while retaining the secret key* sk *confidential and undisclosed.*
2. *The adversary* $\mathcal{A}$ *proceeds by presenting signature queries* $\{m_\kappa\}_{\kappa=1}^{q}$. *In response to each query* m_κ *the challenger responds by running* Sig.Sign *generating a signature* σ_κ *of* m_κ *which is then transmitted to the adversary* $\mathcal{A}$.
3. *The adversary* $\mathcal{A}$ *produces a pair* (m, σ) *and wins if* σ *is a valid signature of* m *according to* Sig.Verify. *Additionally, the pair* (m, σ) *must not correspond to any of the pairs* $(m_\kappa, \sigma_\kappa)$ *generated during the query phase.*

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{EUF\text{-}CMA}} = \Pr[\mathsf{EUF\text{-}CMA}_{\mathsf{Sig}}(\mathcal{A}, \lambda)] \leq \mathsf{ngl}_{\mathsf{Sig}}(\lambda)$$

A.6 Pedersen Commitment

Definition 8. *Let* g *and* h *represent elements within the group* $\mathbb{G}_q$ *(where the logarithm of* h *to the base* g *is unknown to anyone). The committer binds themselves to a message* $m \in \mathbb{Z}_q$ *by selecting* $r \xleftarrow{\$} \mathbb{Z}_q$ *and performing the computation:* $\mathtt{com} = g^m \cdot h^r$. *This form of commitment can subsequently be revealed by disclosing* v *and* r. *An easily demonstrable theorem in [37] shows that the scheme has binding and hiding defined in the next section.*

More formally, let $\mathtt{com} = (\mathtt{com.Setup}, \mathtt{com.Commit}, \mathtt{com.Verify})$ *be a commitment scheme.*

The Pedersen commitment protocol operates between two parties: a committer, who possesses a confidential message $m \in \mathbb{Z}_q$ *that they wish to commit to, and a receiver.*

1. com.Setup*: Both the committer and receiver have agreed upon a specific group* $(\mathbb{G}, q, g, h)$ *for their communication, where* q *represents the order of the group* $\mathbb{G}$, g *serves as its generator, and* $h \in \mathbb{G}$.
2. com.Commit*: The committer chooses* $r \xleftarrow{\$} \mathbb{Z}_q$, *and computes* $\mathtt{com} = g^m \cdot h^r$.
3. com.Verify*: Given* $(m, r, \mathtt{com})$, *the receiver checks if* $\mathtt{com} = g^m \cdot h^r$ *holds or not.*

A.7 Security Properties of Commitment Scheme

Definition 9. *(As defined above) let* $\mathtt{com} = (\mathtt{com.Setup}, \mathtt{com.Commit}, \mathtt{com.Verify})$ *be a commitment scheme. For any PPT adversary* $\mathcal{A}$*, the following security experiment* $\mathsf{Hid\text{-}com}(\mathcal{A}, \lambda)$ *defines* ***hiding*** *where* $b \in \{0,1\}$*:*

1. *The challenger runs* $\mathsf{PubPar} \xleftarrow{\$} \mathtt{com.Setup}(1^\lambda)$ *and outputs* PubPar *to* $\mathcal{A}$*.*
2. $\mathcal{A}$ *gives two messages* (m_0, m_1) *such that* $m_0 \wedge m_1 \in \mathcal{M}$ *to the challenger.*
3. *The challenger computes* $(\mathtt{com}_b; r) = \mathtt{com.Commit}(m_b)$ *and outputs* $\mathtt{com}_b$ *to* $\mathcal{A}$*.*
4. $\mathcal{A}$ *outputs a bit* b' *to the challenger.*

$$\mathsf{Adv}^{\mathsf{Hid\text{-}com}}_{\mathcal{A}} = |\frac{1}{2} - \Pr[\mathsf{Hid\text{-}com}(\mathcal{A}, \lambda) \ s.t. \ b' = b]| \leq \mathsf{ngl}_{\mathtt{com}}(\lambda)$$

Definition 10. *For any PPT adversary* $\mathcal{A}$*, the following security experiment* $\mathsf{Bind\text{-}com}(\mathcal{A}, \lambda)$ *defines* ***binding*** *where* $b \in \{0,1\}$*:*

1. *The challenger runs* $\mathsf{PubPar} \xleftarrow{\$} \mathtt{com.Setup}(1^\lambda)$ *and outputs* PubPar *to* $\mathcal{A}$*.*
2. $\mathcal{A}$ *outputs* $(\mathtt{com}, m_0, m_1, r_0, r_1)$*.*

$\mathsf{Adv}^{\mathsf{Bind\text{-}com}}_{\mathcal{A}} = \Pr[\mathsf{Bind\text{-}com}(\mathcal{A}, \lambda) \ s.t. \ \mathtt{com.Verify}(\mathtt{com}, m_0, r_0) = 1 \wedge \mathtt{com.Verify}(\mathtt{com}, m_1, r_1) = 1 \ \wedge \ m_0 \neq m_1] \leq \mathsf{ngl}_{\mathtt{com}}(\lambda)$

A.8 d-SDDH Assumption

Definition 11. *Assuming that* g *is a generator of a group* $\mathbb{G}$ *of order* p *where* p *is a prime:* $(\mathbb{G}, p, g) \leftarrow \mathcal{G}(1^\lambda)$*. Given* $(x, x_1, \ldots, x_d) \xleftarrow{\$} \mathbb{Z}_p$*, we say that the decisional arrangement of the d-strong DH assumption (d-Strong Decisional Diffie-Hellman) is hard relative to* $\mathcal{G}$ *if for any PPT distinguisher* $\mathcal{A}$ *there exists a negligible function* $\mathsf{ngl}(\cdot)$ *such that [9]:*

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

B Auxiliary Ideal Functionalities

The elucidation of the functionalities employed within our protocol is articulated as follows.

The functionality denoted as $\mathcal{F}_{\mathsf{KeyReg}}$ serves as a model for key registration within the context of our protocol. A party calls this functionality with the command $(\mathtt{Reg.key}, \mathtt{sid}, \mathtt{pk})$, thereby registering a cryptographic key denoted as $\mathtt{pk}$ under the unique identifier U associated with the party. Subsequently, all participants have the ability to invoke the functionality using either $(\mathtt{Key.Retrieval}, \mathtt{sid}, \mathsf{U})$ to obtain the registered key $\mathtt{pk}$ belonging to party U, or $(\mathtt{id.Retrieval}, \mathtt{sid}, \mathtt{pk})$ to ascertain the identifier of the owner of the key $\mathtt{pk}$.

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

1. **Register Key.**
   (a) Upon input $(\texttt{Reg.Key}, \texttt{sid}, \texttt{pk})$ from some party U, ignore if there exists $(\mathsf{U}', \texttt{pk}')$ where $\mathsf{U}' = \mathsf{U}$. Else, output $(\texttt{Register}, \texttt{sid}, \mathsf{U}, \texttt{pk})$ to $\mathcal{A}$.
   (b) Upon receiving $(\texttt{Ok}, \texttt{sid}, \mathsf{U})$ from $\mathcal{A}$, record the pair $(\mathsf{U}, \texttt{pk})$, and output $(\texttt{Key.Registered}, \texttt{sid})$ to U.
2. **Retrieve Key.**
   (a) Upon input $(\texttt{Key.Retrieval}, \texttt{sid}, \mathsf{U})$ from some party U_j, output $(\texttt{Key.Retrieval}, \texttt{sid}, \mathsf{U}, \mathsf{U}_\mathsf{j})$ to $\mathcal{A}$.
   (b) Upon receiving $(\texttt{Ok}, \texttt{sid}, \mathsf{U}, \mathsf{U}_\mathsf{j})$ from $\mathcal{A}$, if there exists a recorded pair $(\mathsf{U}, \texttt{pk})$, output $(\texttt{Key.Retrieved}, \texttt{sid}, \mathsf{U}, \texttt{pk})$ to U_j. Else, output $(\texttt{Key.Retrieved}, \texttt{sid}, \mathsf{U}, \perp)$ to U_j.
3. **Retrieve ID.**
   (a) Upon input $(\texttt{id.Retrieval}, \texttt{sid}, \texttt{pk})$ from some party U_j, output $(\texttt{id.Retrieval}, \texttt{sid}, \texttt{pk}, \mathsf{U}_\mathsf{j})$ to $\mathcal{A}$.
   (b) Upon receiving $(\texttt{Ok}, \texttt{sid}, \texttt{pk}, \mathsf{U}_\mathsf{j})$ from $\mathcal{A}$, if there exists a recorded pair $(\mathsf{U}, \texttt{pk})$, output $(\texttt{id.Retrieved}, \texttt{sid}, \mathsf{U}, \texttt{pk})$ to U_j. Else, output $(\texttt{id.Retrieved}, \texttt{sid}, \texttt{pk}, \perp)$ to U_j.

$\mathcal{F}_{\mathsf{RO}}$ defined in the following models an idealized hash function [12].

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

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

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

For privacy-preserving requirements, $\mathcal{F}_{\mathsf{PARS}}$ does not reveal the identities of users. To realize this functionality, our protocol employs different kinds of communication channels $\mathcal{F}_{\mathsf{Ch}}$ to deliver messages and to fulfill network-level anonymity (e.g., stemming traffic analysis attacks and extracting identities).

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

Let define the sender and receiver of a message m by S and R respectively.

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

1. Once $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{SSA}}$ is called, set $\alpha \leftarrow R, \beta \leftarrow |m|, \gamma \leftarrow S$ (secure and sender anonymous channel).
2. Once $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{SRA}}$ is called, set $\alpha \leftarrow S, \beta \leftarrow |m|, \gamma \leftarrow S$ (secure and receiver anonymous channel).
3. 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).
4. 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).

In the following, we define the standard Broadcast functionality $\mathcal{F}_{\mathsf{B}}^{\mathsf{S}}$ from [20] 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 [10] among servers.

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

Broadcast functionality $\mathcal{F}_{\mathsf{B}}$ parameterized by the set $\mathbb{I} = \{\mathsf{I}_1, ..., \mathsf{I}_D\}$ proceeds as follows:

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

Groth et al. [27] 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.

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

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

Proof Generation:

1. On receiving $(\texttt{Prove}, \texttt{sid}, \texttt{x}, \texttt{w})$ from some party U, ignore if $\mathsf{R}(\texttt{x}, \texttt{w}) = 0$. Else, send $(\texttt{Prove}, \texttt{sid}, \texttt{x})$ to $\mathcal{A}$.
2. Upon receiving $(\texttt{Proof}, \texttt{sid}, \pi)$ from $\mathcal{A}$, store $(\texttt{x}, \pi)$ and send $(\texttt{Proof}, \texttt{sid}, \pi)$ to U.

Proof Verification:

1. Upon receiving $(\texttt{Verify}, \texttt{sid}, \texttt{x}, \pi)$ from some party U, check whether $(\texttt{x}, \pi)$ is stored. If not send $(\texttt{Verify}, \texttt{sid}, \texttt{x}, \pi)$ to $\mathcal{A}$.
2. Upon receiving the answer $(\texttt{Witness}, \texttt{sid}, \texttt{w})$ from $\mathcal{A}$, check $\mathsf{R}(\texttt{x}, \texttt{w}) = 1$ and if so, store $(\texttt{x}, \pi)$. If $(\texttt{x}, \pi)$ has been stored, output $(\texttt{Verification}, \texttt{sid}, 1)$ to U, else output $(\texttt{Verification}, \texttt{sid}, 0)$.