NovaTEE: Private Clearing and Settlement on Trusted Execution Hardware

Ahmet Ramazan Ağırtaş[1], James Ball[1], Michael Belegris[1] and Gustave Charles-Saigne[a,2]

[1] Nethermind Research, United Kingdom
[2] EPFL, Switzerland

Abstract. NovaTEE is a novel private multilateral settlement network designed to address critical inefficiencies in both traditional financial markets and cryptocurrency trading. The current clearing landscape suffers from fragmented capital allocation, restrictive prime brokerage relationships, and prolonged settlement timeframes in traditional finance, while cryptocurrency markets face challenges with over-collateralization, siloed lending pools, and security risks from centralized exchanges.
We introduce a settlement system that leverages Trusted Execution Environments (TEEs) and threshold cryptography to enable secure, private, and efficient settlement of obligations between multiple parties. The system utilizes a distributed key generation model and novel clearing mechanisms to optimize capital efficiency through multilateral netting, while maintaining strong privacy guarantees and regulatory compliance capabilities. By combining TEE-based security with advanced cryptographic protocols, including zero-knowledge proofs and sparse Merkle trees for data verification, our solution enables efficient cross-venue and cross-chain settlement while protecting sensitive trading information. This approach significantly reduces capital requirements for market participants, optimizes transaction costs, and provides institutional-grade clearing infrastructure without compromising on security or privacy. The system's architecture ensures that no single party has complete access to transaction details while maintaining auditability through a distributed backup network, offering a practical solution for institutional adoption of on-chain settlement.

Keywords: Trusted Execution Environments · Threshold Cryptography · Privacy · Clearing · Settlement

E-mail: ahmet@nethermind.io (Ahmet Ramazan Ağırtaş), james@nethermind.io (James Ball), michael.belegris@nethermind.io (Michael Belegris), gustave.charles--saigne@epfl.ch (Gustave Charles-Saigne)

[a] Work undertaken while interning at Nethermind.

Contents

1 Introduction **3**

2 Protocol Overview **4**

3 Our Approach **5**
3.1 Joining the Network . . . 6
3.2 Account Balance Management . . . 6
3.3 Execution Venue Connections . . . 7
3.4 Execution and Settlement . . . 8
3.5 Novation and Capital Efficiency . . . 9
3.6 Receipt Handling . . . 11
3.7 Privacy . . . 12

4 Concerns **13**
4.1 Compliance . . . 13
4.2 Trusted Execution Environments . . . 14
4.3 Security Guarantees . . . 14
4.4 Scaling . . . 14

5 Preliminaries **17**
5.1 Distributed Key Generation . . . 17
5.2 Reconstruction of shares . . . 18
5.3 Threshold version of ElGamal encryption scheme (Hashed variant) . . . 19
5.4 Sparse Merkle Tree (SMT) . . . 19

6 TEE-based Private Settlement Network Construction **21**
6.1 Setup and Terminology . . . 22

7 Protocol **24**
7.1 Deposit Phase . . . 24
7.2 Clearing Phase . . . 25
7.3 Sub-processes . . . 25
7.4 Partial Obligations . . . 26
7.5 Withdrawal Phase . . . 27

8 Compliance Details (Backup Network $\mathcal{B}$) **28**
8.1 SMT for Trader Data . . . 28
8.2 SMT for TEE Key Backups . . . 29
8.3 SMT for Obligation List Backups . . . 29
8.4 Network Operations . . . 29

9 Algorithms **31**

References **39**

1 Introduction

The traditional financial system relies heavily on prime brokers to facilitate trading and manage risk between market participants. Although this model has served traditional markets for decades, it has created significant inefficiencies in capital allocation and market access. Traditional finance suffers from restrictive prime brokerage relationships, where services are limited to large institutional clients who can meet substantial capital requirements and navigate extensive due diligence processes. These arrangements lead to fragmented credit lines and redundant capital locked across multiple prime brokers. Standard settlement timeframes, typically T+1 or T+2, further compound capital inefficiency by necessitating prolonged capital immobilization, particularly in international transactions where various intermediaries maintain separate capital reserves.

Compared to traditional capital markets, cryptocurrency markets face their own set of unique challenges. Capital is frequently fragmented across centralized exchanges, forcing traders to maintain multiple margin accounts with significant over-collateralization requirements, while on-chain, the trustless nature of the technology means that it is difficult for markets to breach collateralization thresholds without putting their markets at risk. The inability to net positions across venues and chains leads to substantial capital inefficiency. DeFi protocols, while innovative, often require extreme over-collateralization, sometimes exceeding 150%, and operate in siloed lending pools that cannot efficiently share liquidity. Recent failures of centralized exchanges such as FTX have highlighted the risks inherent in traditional prime brokerage models applied to cryptocurrency markets, while also emphasizing the lack of institutional-grade clearing and settlement options in the space.

Clearing houses have emerged as a crucial component of financial infrastructure to address similar challenges in traditional markets. They enable multiple parties to efficiently transfer risk between each other while minimizing capital requirements through optimized debt settlement between counterparties. A modern clearing system can provide settlers with optimal access to credit facilities while maintaining the trustless properties of blockchain systems. When multiple venues are introduced to a clearing system, the efficiency gains compound. Through cross-venue and cross-chain position netting, we can significantly reduce capital requirements for market makers and traders, optimize transaction costs, and enable more efficient settlement mechanisms.

Finally, there is the issue of privacy. While trading can occur off-chain for speed and efficiency, settlement must ultimately happen on-chain through blockchain networks. Current market solutions often force an unnecessary coupling of these processes, where off-chain trading leads to delayed settlement cycles that lock up capital and impede trading strategies. This has pushed many participants toward custodial prime brokerage solutions, services where third parties hold and instruct the movement of assets on behalf of clients.

These custodial arrangements introduce operational friction and create security risks due to third-party key management. Custodial solutions are exposed to hacking and exploit risks; in 2023 alone, $1. 7 billion were lost to exchanges and other custodial platforms [Cha24]. While the underlying assets are tokens represented on-chain, their settlement could happen in seconds or minutes after trading, yet participants often wait hours before being able to move their assets due to current market infrastructure moving trading activity off-chain in ways that make markets more opaque and inefficient. On-chain, the transparency of distributed ledgers like Ethereum—a public blockchain network—exposes the historical, current and future state of a user account:

- All user trade intentions, both past and present, are visible to all participants, as well as the record of future transaction states after trade. The consequence of this transparency is that users' transactions are exposed to quote fading, where the price or terms of a trade worsen before execution due to market participants reacting

to the visible information. This is in association with other externalities of public trading such as front-running, sandwiching, and other transaction order manipulation strategies.

- Block builders and sequencers, who are responsible for assembling and ordering transactions into blocks, can see the pending state of new blocks. This visibility allows them to reorder transactions in the mempool, the pool of unconfirmed transactions awaiting inclusion in a block, and extract the Miner extractable value (MEV) from the transactions of users. MEV refers to the profit that miners (or validators) can extract from a transaction by manipulating the order of the transactions in the block.

Managing the issuance of tokenized assets on public chains becomes increasingly difficult on fully transparent ledgers, where traders might seek to preserve information about their strategies or the participants involved in the issuance and redemption of their assets. In the long term, our vision is that financial activity will increasingly move on-chain. Asset tokenization - the process of representing ownership rights to real-world assets as digital tokens on a blockchain - represents a fundamental shift in market structure, and this will be mirrored in the adaptation of the existing market infrastructure on the chain. We believe that one of the impediments for institutions to fully embrace Distributed Ledger Technology (DLT) is that they need privacy-preserving solutions to secure their on-chain interactions while still desiring to retain the security guarantees that transparent, decentralized, and credibly neutral public blockchains provide; a TEE based solution offers a compromise between these two. Our goal in building unlit execution layers, private trading mechanisms that operate on-chain, is to return trading and execution back to the asset layer and allow institutions to take full advantage of native on-chain settlement, where trades are settled directly on the blockchain without intermediaries.

2 Protocol Overview

In the following, we introduce a settlement network designed from first principles to be **secure**, **private**, **mutual** and **credibly neutral**. To achieve this we utilize a novel key encumbrance model powered by threshold cryptography and a Trusted Execution Environments (TEEs) sidecar as the primary security mechanism. Together these two technologies form part of a clearing and settlement layer that, through a model of 'abstraction' allows users to post-obligations (agreements to settle) and "net-off" owed balances multilaterally and efficiently through novation of credit. Other recent work has explored similar approaches to multilateral clearing and settlement. For example, Cycles[BDA+24] proposes a graph-based clearing protocol that optimizes debt reduction through cyclic settlement flows. While Cycles focuses on obligation networks and permissionless credit issuance, our approach emphasizes institutional-grade security through TEEs and threshold cryptography. This work also builds upon earlier work into mutual credit and obligation clearing by Fleishman, Dini and Littera. [FD21] TEEs in this protocol serve a dual purpose: they guarantee the secure execution of settlement obligations between counterparties while protecting user privacy by minimizing exposed information. The system uses distributed key generation (DKG) and a threshold encryption scheme for key management, complemented by Sparse Merkle Trees for efficient data verification. Withdrawals from the clearing network are securely managed through a threshold signature scheme. Using TEEs has the added advantage of being able to provide signed attestations and logic guarantees for intents performed inside of the network that are auditable by third parties such as regulators.

3 Our Approach

The settlement network is setup with the following participants:

- **Traders (T_i):** Individual participants who want to trade assets, and settle balances on the network.
- **Settlement Network ($\mathcal{S}$):** Network of N nodes running in Trusted Execution Environments (TEEs).
- **Backup Network ($\mathcal{B}$):** Network of M nodes running in TEEs for redundancy and data backup storage.
- **Execution Venue (EV):** The execution venue or platform which instructs settlement between users on the network.

Fundamentally, when a user has agreed to sell an asset - they owe an outstanding commitment to settle, an obligation. Each obligation defines the essential parameters of the transfer, including the sender, recipient, and asset quantity. Commitments are unidirectional, representing a single movement of assets from one party to another.

Such a system could also be extended to provide flexibility in settlement by allowing participants to define acceptance Preferences. These preferences specify which asset types are acceptable for settlement - enabling users to define their preferences to receive alternative assets for settlement provided they fall within their predefined parameters. For instance, a participant might specify they will accept either USDC or USDT for a particular commitment. Such a system would give way to more advanced chains of settlement between counterparties and their respective Execution venues (for example Electronic Communication Networks or ECNs), where third party liquidity providers ("Creditors") could provide execution services to 'step in' efficiently to close novation cycles, or creditors could extend credit lines to counterparties allowing users to borrow capital to fulfill outstanding obligations to counterparties in the clearing network. In traditional financial systems, this is typically the role of a Prime Broker - who plays the role of a centralized credit party by offering credit lines to their respective customers.

For settlement, the network relies upon two types of balances; free balances and encumbered balances:

- **Free Balances** - Free balances are funds that are immediately available for settlement or withdrawal, and have not been committed to pending trades or obligations. In addition, these funds are available for settlement with Execution Venues. Free balances should be thought of as available balances for spending.
- **Encumbered Balances** - Encumbered balances are funds that are locked or reserved for outstanding settlement with an Execution Venue. Encumbered balances are, for example, committed to an outstanding position on an Execution Venue or some other counterparty obligation - an example being a new order that hasn't settled yet, trade that's being processed, in-progress withdrawal request, Credit agreements. Encumbered funds cannot be used for new trading until they have released (become 'free').

A distinction between Free and Encumbered balances is made to ensure there's always sufficient collateral for pending trades and to provide clear visibility into what funds are actually available for Execution Venues. Balances are held in EOA's controlled by a distributed network of TEEs. The private keys associated with a user's account are never shared with the user until withdrawal where a fresh key pair is generated and shared with the withdrawing party. Although the solution is non-custodial, a user is unable to

withdraw encumbered balances from the platform without mutual consent that the balance is free (unencumbered).

Additionally, because the settlement network operates on top of underlying blockchains where assets are held, users can natively settle transactions trustlessly across chains. In practice this means that a user holding USDC could settle a balance against a user holding BTC on the Bitcoin network - and still retain the settlement finality and security guarantees that they should expect from a 'delivery versus payment' settlement.

The clearing network design is centered around three main sets of actions; **Deposits** - where traders add funds to the settlement system, **Clearing** - which specifically handles obligations/novation cycles, and **Withdrawals** - where traders remove funds from the settlement system.

3.1 Joining the Network

Users can onboard to the network without needing a KYC credential. However, to participate in a clearing cycle, each Execution Venue sets its own KYC requirements. Users complete onboarding either directly with the venue or through a compliance service provider, which issues a credential linked to the user's account for all future interactions.

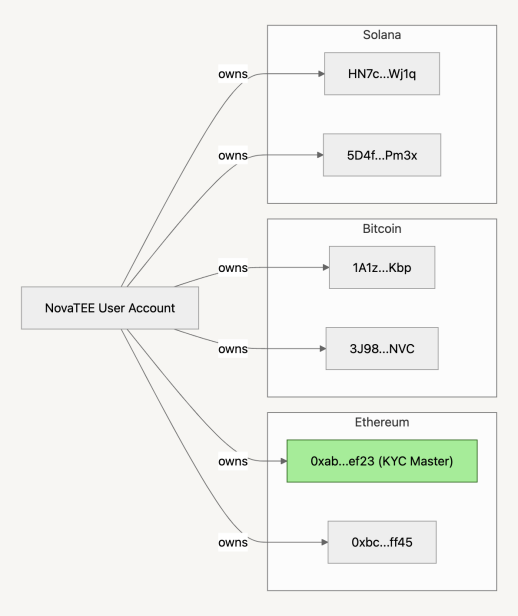


Figure 1: Each account on the settlement network can be represented by an almost unlimited number of EOA's from across many supported chains.

When users onboard, an account is created for them and linked to a set of wallets across the chains they interact with. There is no limit to the number of EOA wallets a user can associate with their account. Additionally, information about a user's account remains private from other users on the system.

3.2 Account Balance Management

After the trader has deposited assets into a wallet associated with their account, the transferred balance available for settlement will be reflected in the clearing network.

1. The Trader requests a fresh deposit address from the settlement network
2. The Network generates an address, creating a new public key (PK_i) for depositing funds and sends it back to the Trader confidentially
3. Trader transfers funds to the provided address (PK_i)
4. The Network confirms receipt and updates trader's free balance reflected in their account on the settlement network
5. The network state steps up in the backup network

If users want to make additional deposits, they can request new deposit addresses as needed. The system also allows users to deposit funds into an aggregated wallet, reducing the number of network transactions and lowering withdrawal fees by consolidating withdrawals into a single transaction per asset and network.

Withdrawals are processed only after the next novation and settlement cycle. When a user requests a withdrawal, funds are transferred to a newly generated EOA created by the settlement network (note that this is optional, and the network can also operate on a single 'aggregated wallet per network' model). Creating a new wallet for each transaction preserves user anonymity by eliminating any direct link between the initial deposit and subsequent withdrawals. This privacy feature is particularly valuable for traders, as it abstracts their settlement activity across off-chain execution venues while keeping their on-chain asset movements discreet.

3.3 Execution Venue Connections

A user is not required to connect to an execution venue to post an obligation and settle a balance, user can unilaterally create a settlement obligation. Users are required to permission an execution venue to generate obligations on their behalf. Users may connect to multiple execution venues, but in the case of a simple bilateral settlement users are able to generate obligations independently. It is important to note that the Execution Venue can only create obligations between account with the consent of both counterparties.

1. The trader (T) connect their account to an Execution Venue (EV)
   (a) T reviews the EV's KYC requirements and trading policies
   (b) T completes the EV's onboarding process and KYC verification
   (c) T receives credentials specific to that EV
2. The EV completes the connection:
   (a) Validates T's credentials and permissions
   (b) Links T's settlement network account to their venue account
   (c) Configures trading limits based on T's free balance
   (d) Sends confirmation to T that trading can begin
3. The Settlement Network ($\mathcal{S}$) processes the connection request:
   (a) Verifies T's signature and credentials
   (b) Validates the EV's registration status
   (c) Records the permission grant in the system
   (d) Updates T's account settings to allow EV access

(e) Notifies the EV of the successful connection

(f) Sends confirmation to T

4. T initiates the venue connection request:

(a) T generates a connection request containing:

- Their account credentials (Cred_i)
- The EV's identifier
- Permission scope specifications

(b) T signs the connection request with their private key

(c) T submits the signed request to the settlement network

By granting an execution venue permissions, a user grants the execution venue access to their account balance information (including free and encumbered balances), in addition to the ability to view KYC information, records of their net obligations to the venue and permissions to generate new net obligations between the user's account and any counterparties they owe outstanding balances to throughout 'the course of trading on the execution platform.

3.4 Execution and Settlement

Execution Venues pass settlement instructions for users on their platform on a periodic basis. These settlement instructions reflect one way balance settlements between users on their platform, or outstanding obligations to intermediate parties (for example, any fees which may be payable to the Execution Venue).

1. Trading Phase:

(a) T places orders on the EV platform

(b) When a trade matches:

- EV monitors T's global free balance
- EV tracks current outstanding obligations
- Trading continues as long as sufficient free balance exists (as determined by venue)
- When trades match, free balance is automatically encumbered

(c) EV records all matched trades and resulting obligations

2. Settlement Instruction Generation:

(a) EV periodically (e.g., every hour) compiles settlement instructions:

- Calculates net positions for each trader
- Includes any venue fees or charges
- Creates one-way settlement obligations

(b) EV signs the settlement instruction batch

(c) EV submits signed instructions to the settlement network

3. Settlement Network Processing:

(a) $\mathcal{S}$ receives settlement instructions and verifies EV signature

(b) For each trader involved:

- Validates sufficient encumbered balance exists
- Checks for any conflicting obligations

(c) $\mathcal{S}$ initiates the novation cycle:

- Combines all pending obligations across all venues
- Identifies traders with insufficient balance
- Removes ineligible obligations
- Computes optimal settlement paths

4. Settlement Execution:

(a) $\mathcal{S}$ executes the novated obligations:

- Updates all trader balances
- Releases encumbered balances no longer needed
- Records all settlements

(b) $\mathcal{S}$ notifies all parties:

- Sends settlement confirmations to EVs
- Updates trader balance statements
- Flags any failed settlements

5. Post-Settlement:

(a) EVs receive settlement confirmation:

- Update their internal records
- Release any trading holds
- Notify traders of completed settlements

(b) Traders can view:

- Updated free and encumbered balances
- Settlement confirmations
- Any failed settlements requiring action (where obligations have been excluded from a settlement cycle)

Importantly on this system, because the Execution Venues have insight into the global balance of the user, as well as the users current outstanding obligations to the venue, users can continue trading as long as they have sufficient free balance (as deemed on a per counterparty or settlement venue). Throughout the course of trading as the user accrues profit and losses on the execution venue, the venue passes net obligations to add or remove balances from the user's account (through the creation of one way obligations between the user and counterparties registered on their system. Free balance is automatically encumbered when trades are matched and obligations are posted by the Execution Venue.

Settlement is enacted globally across all user accounts on the clearing and settlement system in novation cycles. If a user has insufficient free capital to settle their outstanding obligations, they are excluded from the cycle, and the novation is recomputed with the parties who cannot pay outstanding obligations excluded.

3.5 Novation and Capital Efficiency

Novation is the act of 'netting off' owed balances between a set of counterparties down to their net-owed balances (or obligations) - see the example in Figure 2. In our proposed system, Novation is used to efficiently compress netting cycles to compute the shortest

and most efficient pathway of owed liquidity in order to efficiently clear a set of owed debt positions.

Novation mechanisms provide significant capital efficiency improvements over traditional bilateral settlements, where in systems of increasingly counterparties with complex credit obligations, it can be used to compress the owed balance transfers between counterparties into as few transfer operations as possible.

In a simple example we have three participants (Alice, Bob, and Charlie) engaged in multiple transfers:

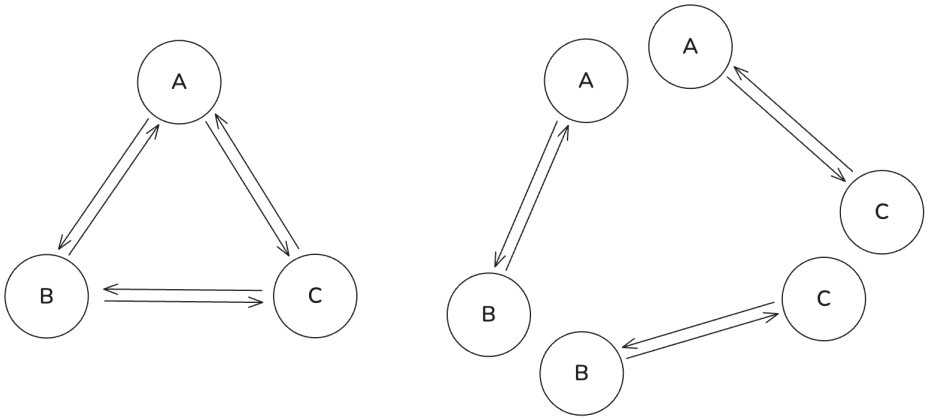


Figure 2: Coupled (Left) versus an uncoupled (Right) settlement run.

In a standard settlement run, each transfer would require individual, isolated movement of capital. For example:

- A must reserve capital for transfers to both B and C
- B must reserve capital for transfers to A and C
- C must reserve capital for its transfers to A and B

Through multilateral netting and novation, the system can:

1. Identify where circular dependencies exist
2. Collapse multiple bilateral commitments into a single netted position
3. Reduce the total capital required by eliminating redundant transfers

In theory, participants only need to maintain capital for their net position rather than the gross bilateral commitments. However, in reality the participants need enough capital to satisfy their gross bilateral commitments between one net settlement and the next - because they do not know which of their trading counterparties will be in, and which will be out, of the net settlement run.

For instance, if Alice owes Bob 100 USDC, Bob owes Charlie 100 USDC, and Charlie owes Alice 100 USDC, the system can settle all commitments with zero net capital movement through novation. Without novation, this would require 300 units of locked capital across the participants.

The capital efficiency gains of the settlement system become more pronounced as the number of participants increases, the volume of bilateral commitments grows, the complexity of transfer relationships increases or the frequency of settlement cycles increases in cadence.

In a scenario where multiple traders may have hundreds of open positions across multiple execution venues, novation allows the net obligations between each users for their outstanding positions to be compressed down to a single set of settlement obligations with a shared margin balance between venues.

3.5.1 Credit

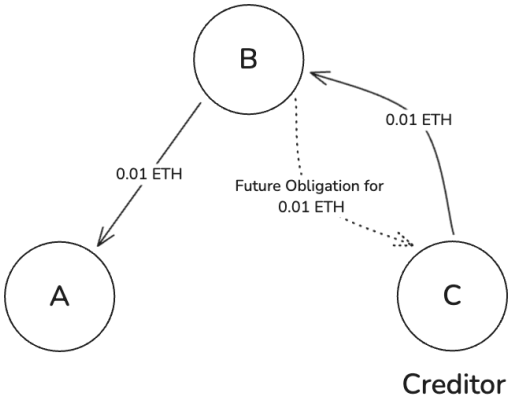


Figure 3: Example of a credit cycle where B repays their net obligation to A with credit. The supply of credit create a new future repayment obligation back to the Creditor.

During a novation cycle, participants can offer to extend credit to other users to facilitate settlement. This creates a new future commitment between the credit provider and the recipient. For example, if Bob requires additional liquidity to settle with Charlie, Alice might extend credit to Bob. This generates a new commitment where Bob must settle with Alice at a future date (see Figure 3). Users might choose to borrow from another counterparty - in the example below; Bob borrows from Charlie in order to clear outstanding debts to Alice, but in doing so creates a new future obligation to Charlie to repay his debts.

The extension of a credit line can therefore be thought of as the generation of a future obligation, by a third party creditor to pay a present obligation. The addition of credit facilities to the novation cycle enables more efficient settlement by providing temporary liquidity support within the system's framework.

3.6 Receipt Handling

The receipt of incoming transfers is processed automatically without requiring recipient confirmation, leveraging the cryptographic security of the TEE-based settlement system. Once a commitment is initiated and confirmed by the sending party through valid signatures and sufficient balance verification, the receiving party cannot reject or modify the incoming transfer - this design choice ensures settlement finality and prevents gridlock scenarios in the netting process.

A core principle of the system is that both the sender and recipient receive cryptographic proof that their commitment has been included in a novation and has been settled. For each novation cycle, the system generates proofs through multiple mechanisms; First, the initial obligation list is signed by the Execution Venue and verified by the settlement network. Then, during the netting process, zero-knowledge proofs are generated to demonstrate the equivalence between initial and final obligation lists. The final settled state is backed up to the backup network and periodically anchored on-chain through Merkle root commitments.

This creates a chain of cryptographic evidence from obligation creation through netting to final settlement. These proofs can be verified independently, can verify that a transaction has completed without revealing details of the settlement, create an audit trail linking obligations to settlements and allow for a reconstruction of the full history of the settlement system.

The proof data is retained in a backup network using Sparse Merkle Trees, allowing for efficient storage and verification while maintaining privacy through the same threshold encryption mechanisms used for other sensitive data.

3.7 Privacy

The settlement system has been setup in such a way that:

- No single party has access to complete information about processes running inside the settlement system.
- Data is compartmentalized and encrypted.
- Access requires multiple parties to cooperate.

On-chain, privacy is achieved by requiring each user to deposit to a fresh unlinked address generated on-chain. There is no visible link between the addresses and the user visible on-chain. This is described in Figure 4.

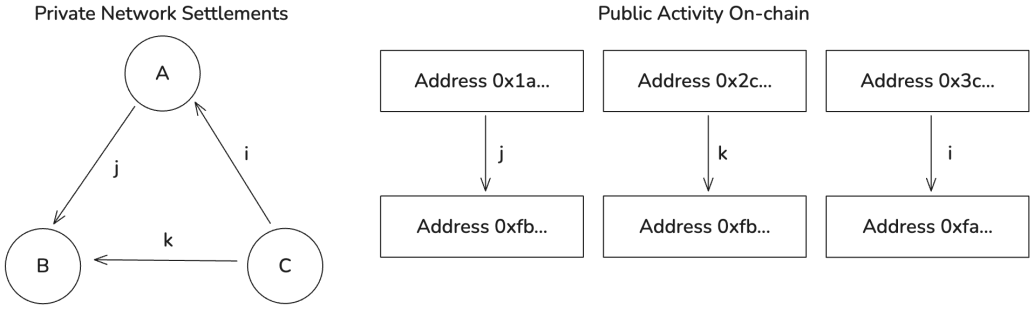


Figure 4: A very simple hypothetical example, where a single set of three participants each owe 100 units to each other. The net settlement between accounts is expressed as a set of unlinkable movements on-chain.

Once the funds are deposited - the Traders account balance is updated, a Traders account information is maintained privately within the trusted execution environment - which upon updating the balances and key lists, the settlement network reaches a consensus on the balances and key lists. Balance updates remain encrypted to the network when they are backed up and shared with the backup network.

Only execution venues authorized by the user can request balance information, and this requires the Execution Venues to share valid credentials, proper signatures and a threshold decryption process. Additionally, obligations are also handled securely inside of the TEE environment, as well as the clearing process - only the final netted results are shared with the end user, and the original obligation mappings are kept confidential to prevent information about users' settlement activity from leaking from the system. While the TEE can see and process the complete settlement graph, the public blockchain only shows disconnected transactions between unlinked addresses.

Access to data is separated over multiple layers within the settlement system:

- Traders can only access their own data and cannot view the data of other traders.

- Execution venues can only request balance information from Traders (or the backup network), and can only view outstanding obligations between Traders who have permissioned the venue.
- Settlement network nodes have controlled access to necessary data - TEEs can only access data they're authorized to handle and do not have access to complete secret keys, all key backups are double encrypted. Although nodes have access to other key backups, the Traders private keys are kept secure.
- Backup network nodes store encrypted data but cannot decrypt without threshold cooperation from other nodes on the network.

For public observers looking from the outside in, they only see unrelated addresses, information about the relationships between the parties is not leaked and final settlements appear as independent transactions to new addresses. In totality, this makes it extremely difficult to ascertain the flow of funds or relationships between parties on the network. The flow of obligations, relationships between parties, and purpose of users' transactions become obscured.

3.7.1 Deposit and Withdrawal Transaction Privacy

For deposits to or from the settlement system, an optional secure system of encumbered private keys is utilized to additionally obfuscate the movement of users funds to or from the clearing system. The withdrawal transaction are generated within the TEEs - key pairs are secured by the TEE, and for each withdrawal, a fresh address is generated by the network. Transaction details for the withdrawal (e.g. TxHash) are only shared with the party who initiated the transaction.

4 Concerns

4.1 Compliance

In some scenarios, information about the private clearing system might need to be revealed for the purposes of regulatory compliance. Rather than a single privileged key, the system employs a distributed 'security council' whose members collectively hold shares of a threshold signing key, and only cooperatively utilize this function when presented with valid legal authority (e.g., a court order).

To make a request to reveal information, the security council must jointly sign a specific data request to the backup network. If the requested data exists, multiple nodes in the backup network must cooperatively participate in threshold decryption, each sending partial decryptions back to the requester alongside Merkle proofs that verify the data's existence and authenticity. An important feature of this system is that data requests must be specific - they cannot perform broad queries, and each request only reveals the exact data specified - with Merkle proofs revealing the minimum necessary information about a user's transaction. When a valid request is received, multiple nodes in the backup network must cooperatively participate in threshold decryption. Each node sends partial decryptions back to the requester alongside Merkle proofs that verify the data's existence and authenticity. These proofs ensure that only the minimum necessary information about a user's activity is revealed.

Because all data requests must be jointly signed by the security council and Merkle roots of data access are periodically recorded on-chain, this creates a verifiable record of what was accessed in the network - acting as a 'canary' informing users if their private information was deanonymized and shared with a third party. This multi-party threshold

approach ensures that no single entity can unilaterally access private information while still enabling legitimate regulatory compliance.

4.2 Trusted Execution Environments

A Trusted Execution Environment (TEE) is a secure, isolated area within a processor that guarantees the confidentiality and integrity of code and data loaded inside it. TEEs provide hardware-enforced isolation from the main operating system and other applications, creating a protected space where sensitive computations can be performed with a high degree of trust. They enable secure key storage, cryptographic operations, and confidential computing while protecting against both software and hardware-based attacks, even from privileged system software or the host operating system.

Modern TEE implementations typically offer three core security guarantees: isolation, ensuring that protected code and data cannot be viewed or modified from outside the TEE; attestation, allowing verification that the correct code is running in an authentic TEE; and sealing, enabling secure persistent storage of sensitive data. Leading commercial TEE solutions include Intel's Software Guard Extensions (SGX) and Trust Domain Extensions (TDX), AMD's Secure Encrypted Virtualization (SEV), Arm's TrustZone, and NVIDIA's Confidential Computing platform. These implementations vary in their specific architecture and security properties, but all aim to provide a hardware root of trust for sensitive operations shielding information from external operators who may seek to obtain this information ("side channel attack"). While TEE's can be treated as a secure environment, a parallel universe for programs to execute in a physically secure and un-observable universe - it is important to acknowledged that current implementations of TEEs are still prone to exploits such as side-channel attacks. It is important to knowledge that crypto-physically secure commodity hardware that can be deployed to untrusted operators does not yet exist.[Bel24]

4.3 Security Guarantees

The network's security architecture is fundamentally built on Trusted Execution Environments (TEEs). The system employs a distributed security model where no single TEE or small group of TEEs can compromise overall system security. This is particularly critical because the TEEs collectively maintain custody of users' private keys for settlement operations, with these keys remaining secure even from the TEE operators themselves. Through remote attestation, it is also possible for participant nodes or third parties to verify that a node is operating the network inside of a secure environment and it is running a canonical version of the settlement network software. Any pitfalls or concerns of possible vulnerabilities which could be exploited by malicious hardware operators are countered by a 'defence-in-depth' through the addition of distributed key generation and signing.

The long-term vision for the network is to achieve credible neutrality through a diverse set of independent hardware operators. However, we cannot conclusively trust that the hardware is completely secure against side channel attacks - and therefore a model where operators of the TEE hardware are untrusted may expose the users of the network to the risk of compromised private keys.

4.4 Scaling

Rather than relying on a single Trusted Execution Environment, the network utilizes a network of nodes which participate in distributed key generation and a threshold signature scheme. In this approach, no single TEE has access to the private keys of traders o the system, and the system requires t-out-of-n TEEs to cooperate for any key operations. This

approach however creates some practical limitations to the maximum number of nodes who can participate in the network.

This approach has some drawbacks, because of the $O(n^2)$ communication complexity in DKG operations, the consensus overhead increases logarithmically with the number of nodes due to networking overheads between nodes in the system. The addition of more TEEs to the system means an increase in system security, however the trade off is an impact in key-signing performance. Likewise, the independent backup network is also required to participate in DKG, however this is only required once.

Since the settlement network does every operation in a distributed fashion, some operations may cause overhead in communication and computation. In the flow of the settlement network, the main operations are:

- DKG in deposit and withdrawal phases
- Threshold TX generation (e.g., Threshold ECDSA for Ethereum) in withdrawal phase
- ZKP generation in clearing phase

The concerns about the communication cost of DKG - Since the communication complexity of the DKG is $O(n^2)$, where n is the number of nodes, adding more nodes to the settlement network makes the communication infeasible after a certain number of nodes.

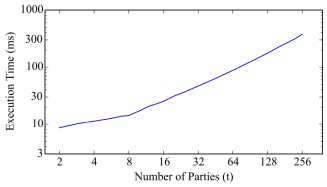


Figure 2: **Wall Clock Times for t-Party Signing over LAN**. Note that all parties reside on individual machines in the same datacenter, and latency is on the order of a few tenths of a millisecond.

(a)

Protocol	Signing $t=2$	Signing $t=20$	Setup $n=2$	Setup $n=20$
This Work	8.6	31.5	48.5	242
[GG18]	77	509	–	–
[LNR18]	304	5194	~11000	~28000
[BGG17]	~650	~1500	–	–
[GGN16]	205	1136	–	–
[Lin17]	36.8	–	2435	–
[DKLs18]	3.8	–	43.4	177

Table 4: **Wall-clock Time Comparison to Other Works**. Note that benchmarking environments are non-identical, but all benchmarks are networkless or over LAN.

(b)

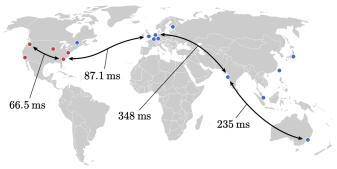


Figure 3: **Map of Datacenter Locations used for WAN Benchmarks**, with latency figures along a few of the longer routes. The subgroup of five zones inside the US are highlighted in red.

(c)

Parties/Zones	Signing Rounds	Signing Time	Setup Time
5/1	9	12.2	67.9
5/5	9	288	330
16/1	10	25.2	181
16/16	10	3018	1655
80/1	12	109	539
80/5	12	388	1521
256/1	13	376	2300
256/16	13	4408	7283

Table 5: **Wall-clock Times in Milliseconds over WAN**. The benchmark configurations used are described in Section 7.3. For signing we varied t according to these parameters, and for setup we varied n, fixing $t = \lfloor (n+1)/2 \rfloor$. Benchmarks involving only a single zone are LAN benchmarks, for comparison.

(d)

Figure 5: Performance benchmarks for DKL implementations from Doerner, J., Kondi, Y., Lee, E., & Shelat, A. (2019). Threshold ECDSA from ECDSA assumptions: The multiparty case.[DKLS19]

Looking at the table in figure 5, (d) we see that:

- 80 nodes from 5 -geographically distributed- zones can perform setup (including DKG) under 2 seconds
- 256 nodes from 16 -geographically distributed- zones can perform setup (including DKG) under 8 seconds

Regarding the threshold transaction generation, the communication complexity is $O(t)$ where $t < n$ is the threshold value. The state-of-the-art threshold ECDSA implementations like [GG18, DKLS19, GS22, DKLS23] have quite good performance as seen in Figure 5. The first two figures correspond to the benchmark over a LAN, and the last two are the results for a WAN benchmark.

> *We benchmarked our implementation using a set of Google Cloud Platform n1-highcpu-8 nodes, each running Ubuntu 18.04 with kernel 4.15.0. Each node of this type has four physical cores clocked at 2.0 GHz, and is capable of executing eight threads simultaneously.*[DKLS19]

In the previous example, we see that:

- 80 nodes from 5 -geographically distributed- zones can generate a threshold ECDSA signature under 1 second
- 256 nodes from 16 -geographically distributed- zones can generate a threshold ECDSA signature under 5 seconds

Regarding the ZKP in the clearing phase, in our clearing engine (which we call "solver") where obligations are netted, user balances are updated, and a zero-knowledge proof is generated for correctness, the choice of proof system and potential use of folding schemes significantly impacts efficiency.

Table 1: Prover comparison. m = number of wires, n = number of multiplication gates, a = number of addition gates

	size $\leq d$ SRS	size $= n$ CRS/SRS	prover work	proof length	succinct	universal
Groth'16	-	$3n + m$ $\mathbb{G}_1$	$3n + m - \ell$ $\mathbb{G}_1$ exp, n $\mathbb{G}_2$ exp	2 $\mathbb{G}_1$, 1 $\mathbb{G}_2$	✓	✗
Sonic (helped)	$12d$ $\mathbb{G}_1$, $12d$ $\mathbb{G}_2$	$12n$ $\mathbb{G}_1$	$18n$ $\mathbb{G}_1$ exp	4 $\mathbb{G}_1$, 2 $\mathbb{F}$	✗	✓
Sonic (succinct)	$4d$ $\mathbb{G}_1$, $4d$ $\mathbb{G}_2$	$36n$ $\mathbb{G}_1$	$273n$ $\mathbb{G}_1$ exp	20 $\mathbb{G}_1$, 16 $\mathbb{F}$	✓	✓
Auroralight	$2d$ $\mathbb{G}_1$, $2d$ $\mathbb{G}_2$	$2n$ $\mathbb{G}_1$	$8n$ $\mathbb{G}_1$ exp	6 $\mathbb{G}_1$, 4 $\mathbb{F}$	✗	✓
This work (small)	$3d$ $\mathbb{G}_1$, 1 $\mathbb{G}_2$	$3n + 3a$ $\mathbb{G}_1$, 1 $\mathbb{G}_2$	$11n + 11a$ $\mathbb{G}_1$ exp, $\approx 54(n+a)\log(n+a)$ $\mathbb{F}$ mul	7 $\mathbb{G}_1$, 6 $\mathbb{F}$	✓	✓
This work (fast prover)	d $\mathbb{G}_1$, 1 $\mathbb{G}_2$	$n + a$ $\mathbb{G}_1$, 1 $\mathbb{G}_2$	$9n + 9a$ $\mathbb{G}_1$ exp, $\approx 54(n+a)\log(n+a)$ $\mathbb{F}$ mul	9 $\mathbb{G}_1$, 6 $\mathbb{F}$	✓	✓

(a)

Table 2: Verifier comparison per proof, P=pairing, ℓ=num of pub inputs. For non-succinct protocols, additional helper work is specified

	verifier work	elem. from helper	extra verifier work in helper mode
Groth'16	$3P$, ℓ $\mathbb{G}_1$ exp	-	-
Sonic (helped)	$10P$	3 $\mathbb{G}_1$, 2 $\mathbb{F}$	$4P$
Sonic (succinct)	$13P$	-	-
Auroralight	$5P$, 6 $\mathbb{G}_1$ exp	8 $\mathbb{G}_1$, 10 $\mathbb{F}$	$12P$
This work (small)	$2P$, 16 $\mathbb{G}_1$ exp	-	-
This work (fast prover)	$2P$, 18 $\mathbb{G}_1$ exp	-	-

(b)

Figure 6: Prover (a) and Verifier(b) scheme performance comparison. [GWC19]

Groth16 [Gro16] and PLONK [GWC19] are both widely used zk-SNARK proof systems, but they differ significantly in terms of efficiency for the prover and verifier, as well as their scalability with public inputs. On the verifier's side, Groth16 requires $(3P + \ell)$ $\mathbb{G}_1$ operations, where P denotes pairing computations, and ℓ is the number of public inputs. This means the cost grows linearly with the number of public inputs, making Groth16 efficient for circuits with few public inputs but less favorable for scenarios with a large number of inputs. In contrast, PLONK has a fixed verifier cost of $(2P + 16)$ $\mathbb{G}_1$ operations for the small-proof version, or $(2P + 18)$ $\mathbb{G}_1$ operations for the fast-prover version. Since PLONK's verifier complexity is independent of the number of public inputs, it is better suited for applications where scalability with public inputs is a concern.

Note that, to avoid the public input issue, one can use the following trick:

1. Put all public inputs in private inputs
2. Commit the private inputs
3. Put the commitment to the private inputs as a single public input

This makes Groth16 verification cheaper. However, to note that on the other hand, regarding the proof size and proving time, Groth16 is better than PLONK.

Consequently, Groth16 is often favored for scenarios requiring high prover efficiency and small circuits, while PLONK is preferred for applications needing scalability, universality,

and reduced verifier costs. Thus, the trade-off between faster proving but slower verifying and slower proving but faster verifying is to be considered within the scheme. The scheme performs the following: one solver proves the equivalence of the initial obligation list and the netted off final obligation list, this proof is sent to the Settlement network to be verified for consensus. Based on the explanation above, and the requirements of the network, Groth16 should be used to provide faster proofs. Due to the number of proofs to be constructed it is important to generate them as fast as possible. Nodes can take longer to verify since it is existing in a consensus network whereby even if the verifier does not have time to verify it in time, the rest of the network can be relied on to do so fast enough and can be finalized.

Next, we present the preliminary information we utilize in the TEE-based private settlement network construction.

5 Preliminaries

5.1 Distributed Key Generation

We present FROST Key Generation [KG21], which is a 2-round algorithm. **In the first round**, everyone proves (against rogue-key attacks) that they know the secret value they share. **In the second round,** users participate in Pedersen's DKG with the secret values they chose in the first round.

Let $\mathbb{G}$ be a cyclic group of order q, and G be its generator. Assume we have n players, i.e., $P_1, \ldots, P_n$, and t is the threshold.

Round 1

1. Every player P_i samples t random values $(a_{i0}, \ldots, a_{i(t-1)}) \xleftarrow{\$} \mathbb{Z}_q$, and uses these values as coefficients to define a degree $t-1$ polynomial

$$f_i(x) = \sum_{j=0}^{t-1} a_{ij} x^j.$$

2. Every P_i computes a proof of knowledge for the corresponding secret a_{i0} by calculating $\sigma_i = (R_i, \mu_i)$, such that:

$$k \xleftarrow{\$} \mathbb{Z}_q, \quad R_i = k \cdot G, \quad c_i = H(i, \Phi, g^{a_{i0}}, R_i), \quad \mu_i = k + a_{i0} \cdot c_i,$$

with Φ being a context string to prevent replay attacks. (This may be the hash of requests)

3. Every player P_i computes a public commitment $\mathcal{C}_i = (C_{i0}, \ldots, C_{i(t-1)})$, where: $C_{ij} = a_{ij} \cdot G, \quad 0 \leq j \leq t-1$.

4. Every P_i broadcasts $(\mathcal{C}_i, \sigma_i)$ to all other players P_j.

5. Upon receiving $(\mathcal{C}_j, \sigma_j)$ from participants $1 \leq j \leq n$ $(j \neq i)$, participant P_i verifies $\sigma_j = (R_j, \mu_j)$, aborting on failure, by checking:

$$R_j \stackrel{?}{=} \mu_j \cdot G - c_j \cdot C_{j0},$$

where $c_j = H(j, \Phi, \phi_{j0}, R_j)$.

6. Upon success, players delete $\{\sigma_j : 1 \leq j \leq n\}$.

Round 2

1. Each P_i securely sends to each other participant P_j a secret share $(j, f_i(j))$, deleting f_i and each share afterward except for $(i, f_i(i))$, which they keep for themselves.

2. Each P_i verifies their shares by calculating:

$$f_i(i) \cdot G \stackrel{?}{=} \sum_{k=0}^{t-1} i^k \cdot C_{jk} \mod q,$$

   aborting if the check fails.

3. Each P_i calculates their long-lived private signing share by computing: $sk_i = \sum_{j=1}^{n} f_j(i)$, stores sk_i securely, and deletes each $f_j(i)$. (Note that sk_i is a secret share of sk, which is not known to anyone)

4. Each P_i calculates their public verification share $pk_i = sk_i \cdot G$, and the group's public key $pk = \sum_{j=1}^{n} C_{j0}$.

5. Any participant can compute the public verification share of any other participant by calculating: $pk_i = \sum_{j=1}^{n} \sum_{k=0}^{t-1} i^k \cdot C_{jk} \mod q$.

5.2 Reconstruction of shares

Definition 1 (Lagrange Interpolation)**.** Given t shares (x_i, y_i) for distinct x_i's and $i = 1, \ldots, t$, the unique polynomial of degree $t-1$ which satisfies all the points is the linear combination of Lagrange basis polynomials, and given by the equation

$$P(x) = \sum_{i=1}^{t} y_i \cdot \ell_i(x),$$

where the Lagrange basis polynomial $\ell_i(x)$ is given by the equation

$$\ell_i(x) = \prod_{\substack{1 \le j \le t \\ i \ne j}} \frac{x - x_j}{x_i - x_j}.$$

Now, we define another notion called the *Lagrange coefficient*, which we will use to recover the secret keys from the backups.

Definition 2 (Lagrange Coefficient)**.** Given a set of t points (x_i, y_i) for distinct x_i's and $i = 1, \ldots, t$, the Lagrange coefficient λ_i is the evaluation of the Lagrange basis polynomial $\ell_i(x)$ at 0, i.e.,

$$\lambda_i = \ell_i(0) = \prod_{\substack{1 \le j \le t \\ i \ne j}} \frac{-x_j}{x_i - x_j}.$$

Reconstruction of the shared secret

Using the definitions above, the reconstruction of the shared secret key can be computed as follows:

$$sk = \sum_{j \in \mathcal{Q}} \lambda_i sk_i,$$

where $\mathcal{Q}$ is the set of at least t (threshold in the above DKG protocol) distinct indices, and λ_i is the corresponding Lagrange coefficient.

5.3 Threshold version of ElGamal encryption scheme (Hashed variant)

Let $\mathbb{G}$ be a cyclic group of order q, and G be its generator. Assume we have n players, i.e., $P_1, \ldots, P_n$, and t is the threshold. Let $H : \{0,1\}^* \to \mathbb{F}_q$ be a hash function that maps arbitrary-length binary inputs to $\mathbb{F}_q$.

The threshold variant of hash-based ElGamal encryption scheme [ElG85, Gai23] is composed of three algorithms, i.e., *key generation, encryption, threshold decryption.*

Key generation

Each P_i participates in a Distributed Key Generation (DKG) and obtains:

- its partial (secret) decryption key $x_i \leftarrow_R \mathbb{F}_q$ (shares of group's decryption key $x \in \mathbb{F}_q$)
- a public encryption key, i.e., $Y \leftarrow x \cdot G \in \mathbb{G}$

Encryption

Let $m \in \mathbb{F}_q$ be the message to be encrypted. To encrypt m, one proceeds as follows:

- pick a randomness $r \leftarrow_R \mathbb{F}_q$
- compute the ciphertext, i.e., $C = (A, B) \in \mathbb{G}^2$, where $A = r \cdot G \in \mathbb{G}$ and $B = m + H(r \cdot Y) \in \mathbb{G}$

Notice that the plaintext is limited by q. Therefore, if one has $m > q$, then one should split m into chunks smaller than q, and perform encryption per chunk.

Threshold decryption

To decrypt a ciphertext $C = (A, B) \in \mathbb{G}^2$, each player P_i proceeds as follows:

- compute partial decryption, i.e., $A_i = x_i \cdot A$
- exchange A_i with other players P_j
- Upon receiving at least t partial decryptions, compute $x \cdot A = \sum_{j \in \mathcal{Q}} \lambda_j \cdot A_j$, where $\mathcal{Q}$ is the set of at least t partial decryptions, and $\lambda_j \in \mathbb{F}_q$ is the appropriate Lagrange coefficient with respect to P_j
- compute $m = B - H(x \cdot A)$

Next, we describe the data structure that we need in backup network to store the backup and compliance data.

5.4 Sparse Merkle Tree (SMT)

A **sparse Merkle tree (SMT)** [BB, But18, DPP16] is a binary Merkle tree where each leaf corresponds to a unique position derived from the hash of the entry. The tree can accommodate a vast number of potential entries, but only a sparse subset is populated. The path to store an entry is determined by hashing the data and interpreting the resulting bit array. If a conflict arises at a node (due to a different entry with a similar prefix), the paths are traversed until the first differing bit, at which point the entries are stored in separate branches. This structure enables efficient proof generation and verification for data inclusion **or non-inclusion.**

Example

To understand the SMT better, we present the example given in [BB]. Consider an entry Ent = e4cef53e, the path to the leaf is computed as H_{path} = Hash(Ent). Consider Hash(Ent) = 0111111 . . . and the tree below where in each leaf there is a value:

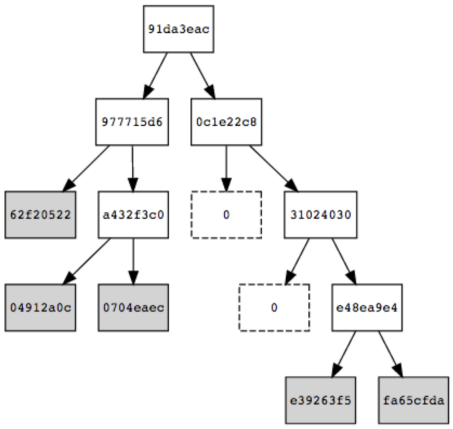


Figure 7: Creating a Sparse Merkle Tree [BB]

If we go down the tree following the sequence 01111111 . . . (for 0 go to the left, for 1 go to the right) we get to the leaf containing the value 0704eaec of some Ent′ with H'_{path} = Hash(Ent) = 0111110 . . .

Comparing H_{path} and H'_{path}, the 7-th bit is the first different bit. This means that we should go down further to the 7-th level (again, for 0 go to the left, for 1 go to the right) and store there the entries as shown in figure 8:

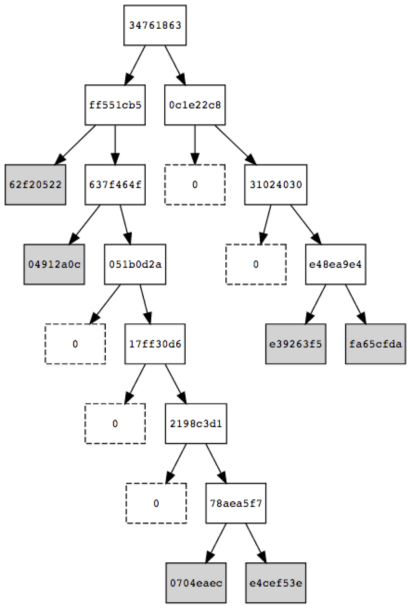


Figure 8: Adding new data to a Sparse Merkle Tree [BB]

Notice that Ent is stored in the right place (as the 7-th bit is a 1) and Ent′ is stored in

the left place (as the 7-th bit is 0). Also notice that the rest of siblings are empty nodes.

6 TEE-based Private Settlement Network Construction

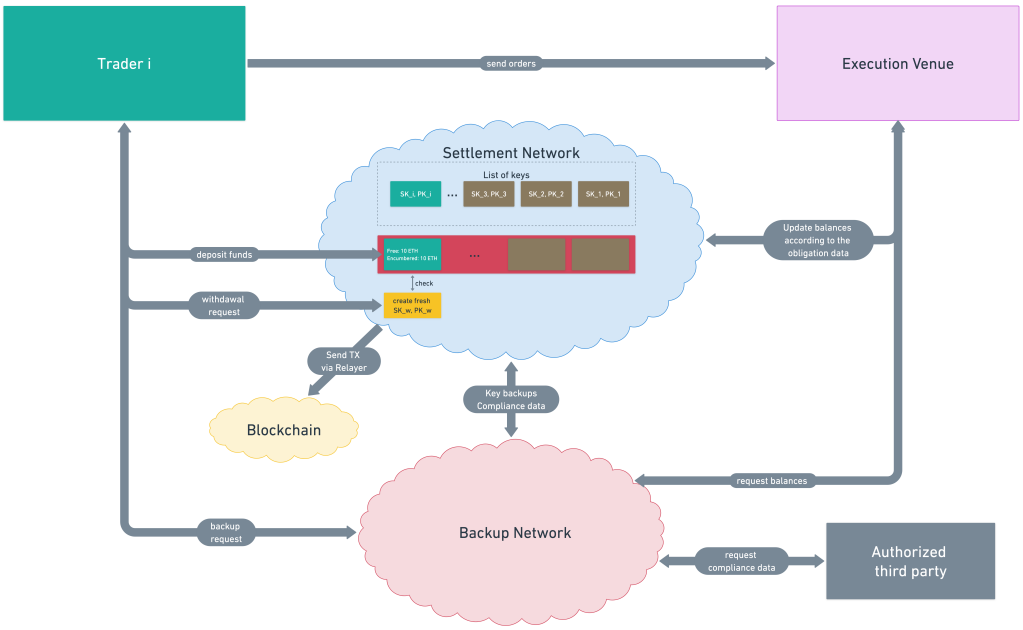


Figure 9: The architecture of proposed TEE-based private settlement network

In our construction, we have four main actors, i.e., the trader T_i, settlement network $\mathcal{S}$ run by N nodes (TEEs of different brands), backup network $\mathcal{B}$ run by M nodes (TEEs of different brands), and an Execution Venue (EV). These actors communicate as follows. Each trader, T_i, starts by requesting a fresh address from settlement network $\mathcal{S}$. In response, settlement network $\mathcal{S}$ jointly generates a new key pair (SK_i, PK_i) for the trader T_i. The secret key SK_i is not known to any TEEs, the k^{th} TEE in $\mathcal{S}$ only has a secret share of it, i.e., $SK_{i,k}$. The partial secret key $SK_{i,k}$ remains securely within the k^{th} TEE, while the public key PK_i is shared with the trader T_i. The trader T_i transfers its funds to the newly provided address associated with PK_i. After confirming that the funds are moved to the address of PK_i, each TEE in the settlement network $\mathcal{S}$ updates the free balance of the T_i, and appends the key pair $(SK_{i,k}, PK_i)$ to its chain-specific lists of keys, locally.

For each novation cycle, the EV will send a signed obligation list $O_{\text{init}} = \{o_1, \ldots, o_n\}$ with n obligations to the settlement network $\mathcal{S}$ to be processed. The network will have assigned a solver that will net off the obligations and produce a more efficient obligation list $O_{\text{final}} = \{o_1, \ldots, o_{n'}\}$ such that $|O_{\text{final}}| \leq |O_{\text{init}}|$. By transforming the initial obligation list into a directed graph $G = (V, E, o)$, it will find Strongly Connected Components, known as cycles, and net them off. Moreover, any parallel lines are merged to further improve efficiency. Finally, the graph is converted back into a list of obligations, and the balances of the traders is altered accordingly. For consensus, a proof is generated ensuring equivalence between O_{init} and O_{final}. This proof along with O_{init} (and EV's signature) and O_{final} are gossiped around $\mathcal{S}$ to ensure consensus on the new state of the balances of the traders.

When a trader wants to withdraw its funds, it sends a withdrawal request to the settlement network $\mathcal{S}$. Each k^{th} TEE in settlement network $\mathcal{S}$ generates a fresh key pair $(SK_{w,k}, PK_w)$ by participating in a DKG. Then, each node in $\mathcal{S}$ picks the first public key pair from its chain-specific lists of key pairs. The settlement network $\mathcal{S}$ jointly generates TXs for sending the funds from the addresses of the picked public keys to the address of PK_w. These TXs are sent to the relayer. The relayer submits the TXs to the corresponding blockchain. After confirming that the TXs are executed, each k^{th} TEE sends the secret key $SK_{w,k}$ to T_i confidentially, and updates the free balance of T_i. Upon receiving enough number of partial secret keys, the trader T_i can reconstruct SK_w.

The backup network $\mathcal{B}$ ensures the security and recoverability of secret keys, balances, and obligations by distributing encrypted key shares across multiple TEEs. It uses Sparse Merkle Trees (SMTs) to efficiently store and verify trader data, providing proofs of inclusion or non-inclusion when needed. Key backups are encrypted through threshold encryption, requiring collaboration from at least t TEEs for decryption, preventing single points of failure.

In the following, we describe our TEE-based construction in detail. First, we introduce our setup and terminology, then we present our protocol.

6.1 Setup and Terminology

We assume that a KYC process has occurred and traders have received credentials, which are stored in both $\mathcal{S}$ and $\mathcal{B}$.

Note that in the below definitions, we use dictionary structure of Python syntax. For example, assume we have the following JSON object ExampleObject:

```
{
    "key1": value1,
    "key2": value2,
}
```

We identify the value1 as ExampleObject[key1], and value2 as ExampleObject[key2]. Below, we give terminology used in our protocol:

- T_i denotes the i-th trader. Similarly, for the clearing section we define T_{i_o} as the trader i from obligation o_x and T_{i_e} defines the trader i from edge e.
- $\text{TEE}_k^{\mathcal{S}}$ denotes the k^{th} TEE in the settlement network $\mathcal{S}$.
- $\text{TEE}_j^{\mathcal{B}}$ denotes the j^{th} TEE in the backup network $\mathcal{B}$.
- $(SK_{i,k}, PK_i)$ denotes the secret and public key pairs the $\text{TEE}_k^{\mathcal{S}}$ generates for T_i's deposit.
- $\mathcal{L}^{sk}_{\text{ChainID}_i,k}$ is the list for $SK_{i,k}$
- $\mathcal{L}^{pk}_{\text{ChainID}_i}$ is the list for PK_i (these two lists are of the same order)
- (sk_i, pk_i) denotes the secret and public key pairs of the trader T_i, which are used for encryption/decryption process in secure communication with $\mathcal{S}$ and $\mathcal{B}$.
- $(ek_{\mathcal{B}}, dk_{\mathcal{B},j})$ denotes the encryption key and partial decryption key of the $\text{TEE}_j^{\mathcal{B}}$, which are used in for encryption and threshold decryption.
- (ek_k, dk_k) denotes the encryption and decryption key pairs of the $\text{TEE}_k^{\mathcal{S}}$, which is used for encrypting its partial key backups.
- $(SK_{\text{EV}_k}, PK_{\text{EV}_k})$ denotes the secret and public key pair of the k^{th} execution venue.

- Cred_i denotes the credential given to the trader T_i via a KYC process.
- ChainID_i denotes the chain ID, a unique identifier per blockchain.
- Token_i denotes the type of the token
- Amount_i denotes the amount of the fund

Balance_i denotes the balance of the trader T_i, whose record is maintained by each TEE as in the below example. (Balance of a trader includes free and encumbered balances, for all tokens and all chains. In the below example, we assume that TEEs support c different chains each having at most d token types.)

```
{
    "Cred": Cred_i,
    "Free Balance": {
        "ChainID_1": {
            "Token_1": 0,
            "Token_2": 0,
            ...
            "Token_d": 0
        },
        "ChainID_2": {
            "Token_1": 100,
            "Token_2": 55,
            ...
            "Token_d": 245
        },
        ...
        "ChainID_c": {
            "Token_1": 0,
            "Token_2": 2000,
            ...
            "Token_d": 1200
        },
    },
    "Encumbered Balance": {
        ... same ...
    }
}
```

- Balance_{PK_i} denotes the balance of the address that corresponds to PK_i
- $\text{Sig}_{\text{sk}}(m) \rightarrow \sigma, \text{VerifySig}_{\text{pk}}(m, \sigma) \rightarrow 0/1$ denotes the signature scheme used in the clearing mechanism. Whereby $\text{Sig}_{\text{sk}}(m)$ denotes the signing algorithm that returns a signature σ given a message m and a secret key sk. $\text{VerifySig}_{\text{pk}}(m, \sigma)$ denotes the verification algorithm used to verify if a signature has message m belongs to a particular public key.
- $o()$ denotes the obligation returning function which returns the amount of an obligation for a given edge in the graph $G = (V, E, o)$, given $e \in E$, $o(e) =$ Amount. Although similar this is a distinction with an obligation denoted as $o_x = \langle(\text{ChainID}, \text{Token}), \text{T}_i, \text{T}_j, \text{Amount}\rangle$ where $x \in \mathbb{Z}$ and o_x is one of the obligations in the list of obligations provided to the clearing engine by the EV.

- $G = (V, E, o)$ defines an obligation graph with m vertices $V = \{T_1, \ldots, T_m\}$, n edges $E = \{E_1, \ldots, E_n\}$ and an obligation function o where $o(e) =$ Amount where the amount is defined by the original obligation o_x. Furthermore, we define a merged edge e_{merged} which is the vector sum of parallel lines between two vertices $T_i, T_j \in V$.
- $\text{Gen}_{\text{ZKP}}(\text{PubInputs}, \text{PrivInputs}) \rightarrow \pi$, $\text{Verify}_{\text{ZKP}}(\pi, \text{PubInputs}) \rightarrow 0/1$ define the algorithms to generate and verify a zero knowledge proof.

It is important to not that all p2p communication is TLS-enabled.[1]

7 Protocol

We describe our protocol in three phases, i.e., *deposit*, *clearing*, and *withdrawal* phases.

7.1 Deposit Phase

The **deposit phase** begins when a trader T_i wants to add funds to the system. The trader sends a request with the details of the deposit, and the TEEs verify the request's validity. Then, participating in DKG, they jointly create a (SK_i, PK_i), and sends the PK_i to the trader T_i. When the trader deposits its funds to the address of PK_i, settlement network updates the balance of the user and they reach consensus on updated balances and deposit addresses. This ends the deposit phase. The complete data flow of the deposit phase is illustrated in Figure 10.

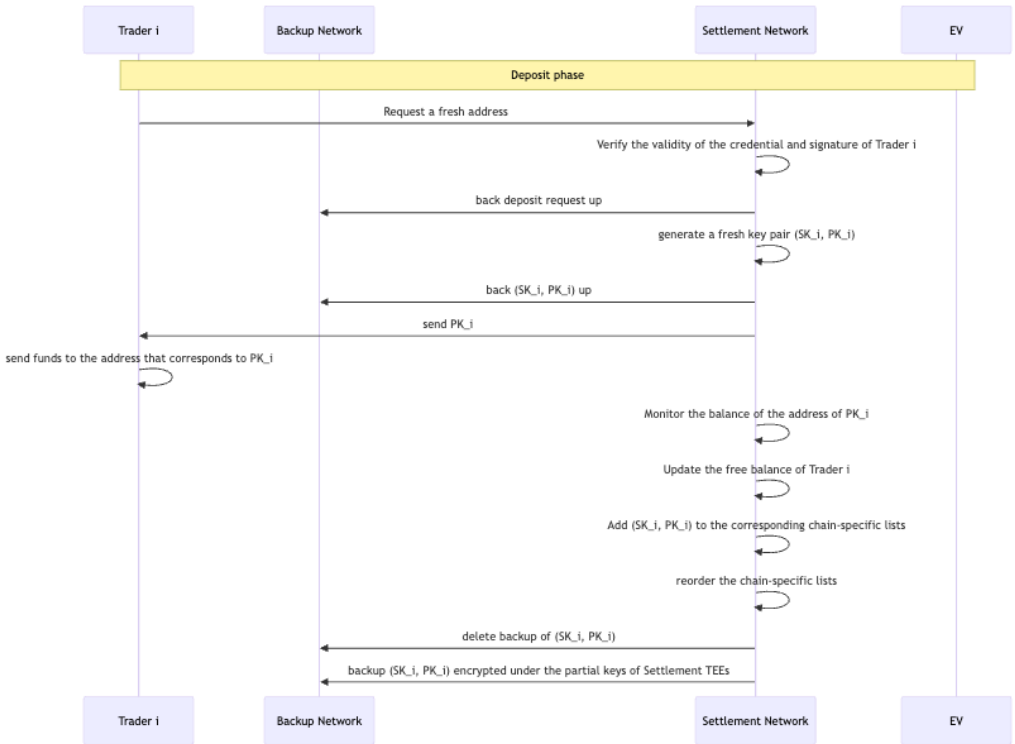


Figure 10: Deposit Sequence Diagram.

When a trader T_i wants to add funds to its balance, it proceeds as follows:

[1] See The libp2p Foundation libp2p - A modular network stack and tls/tls.md for further details.

7.2 Clearing Phase

In the **clearing phase**, each $\text{TEE}_k^{\mathcal{S}}$ receives obligation list per novation cycle from the EV, and gossips it though $\mathcal{S}$. Then, a solver (one of the nodes in $\mathcal{S}$) performs clearing operation and gossips the result. Finally, all TEEs update the traders' balances accordingly. We assume n obligations between m unique traders.The sequence of operations in the clearing phase is shown in Figure 11.

In this construction we restrict the obligations to only describe a 1:1 transfer between users, as it is assumed the EV providing the list of obligations will have already broken down the transactions between parties.

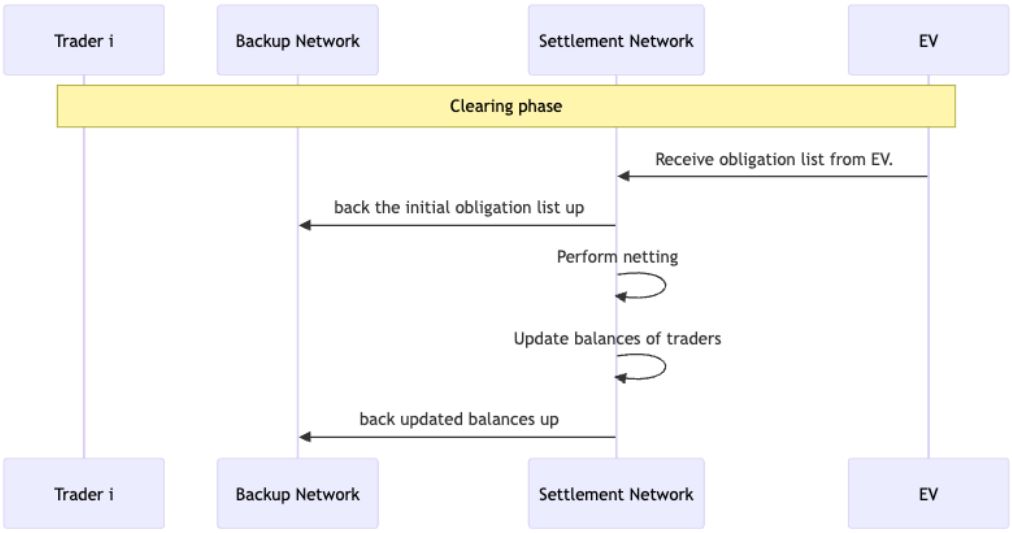


Figure 11: Clearing Sequence Diagram.

An obligation is defined as $o_x = \langle(\text{ChainID}, \text{Token}), \text{T}_i, \text{T}_j, \text{Amount}, \text{Note}\rangle$ for $x = 1, \ldots, n$ where:

- (ChainID, Token) specifies the chain this obligation is for and the token being transferred.
- T_i, T_j describe the users involved in the transfer, specifically $\text{T}_i \to \text{T}_j$. Each user is verified by a KYC credential but for simplicity we refer to this identifier as T. These are also the vertices on the obligation network s.t. $i, j \in V$.
- Amount specifies the amount of the coin/token to be transferred between the parties. This is also the value of the edge between T_i, T_j, s.t. $e_{(i,j)} \in E$.
- Note: A fixed-length character array enabling the EV to attach arbitrary message to their obligations.

The process of clearing is split into novation cycles which define a predetermined period of time to send the obligation list, process it, store the updated state and finally send the results to the EV. The process is described in Algorithm 2.

7.3 Sub-processes

7.3.1 Graph Construction

The constructGraph$(O) \to G$ function converts an obligation list $O = \{o_1, \cdots, o_n\}$ to a direct flow graph $G = (V, E, o)$ [FD21]. The function (described in Algorithm 3) defines

vertices as unique trader identities, edges as connections between traders with associated amounts, and an obligation function mapping edges to their amounts.

7.3.2 Graph Cycle Removal

The digraphToDAG(G) $\rightarrow \{G', C\}$ function converts a directed graph G into an acyclic graph G' by identifying and removing cycles. A cycle is a strongly connected component where there exists a path of obligations connecting any pair of traders in both directions.The cycle removal process is detailed in Algorithm 4.

Definition 3 (Tarjan's Algorithm)**.** Tarjan(G) $\rightarrow C$ identifies strongly connected components with complexity $O(|V| + |E|)$.

- **Input:** A digraph $G = (V, E, o)$
- **Output:** Set of unique cycles $C = \{\{e_x, e_y, e_z\}, \{e_a, e_b, e_c, e_d\}, \dots\}$

7.3.3 Parallel Line Removal

The removeParallelLines(G) $\rightarrow \{G', E_{\text{multi}}\}$ function simplifies a multigraph by combining parallel edges between the same vertices (see Algorithm 5).

7.3.4 Zero-Knowledge Proof Generation

For verification in the consensus network, it is necessary that a zkp of this process is generated and later verified. This is depicted in 6.

- $\text{Gen}_{\text{ZKP}}(\text{PubInputs}, \text{PrivInputs}) \rightarrow \pi$: Proves the relation described below using the public and private inputs outlined.
- $\text{Verify}_{\text{ZKP}}(\pi, \text{PubInputs}) \rightarrow 0/1$: Verifies the zero knowledge proof generated and outputs 1 if valid and 0 otherwise.

7.4 Partial Obligations

In parallel with the clearing phase the settlement network $\mathcal{S}$ can also accept partial obligations which an EV can send to signal that a particular trader is involved in a trade but is not yet ready to settle. Thus, we update the encumbered balance to reflect this.

The following flow outlined in Algorithm 7 is defined to verifiably take into account yet-to-be-completed trades. To do so, an EV will send a partial obligation to $\mathcal{S}$ which will update the encumbered balance of the trader(s) involved. Then, once the EV and trader(s) are ready to settle the partial trade, a full obligation is sent to the $\mathcal{S}$ and the clearing process described above is performed. The EV can then send a partial order complete message to $\mathcal{S}$ to revert the encumbered balance back to its original value.

A partial obligation is defined as:

$$o_{\text{partial}} = \big\langle(\text{ChainID}, \text{Token}, T_i, \text{Amount}, \text{Note}, \text{PartialFlag} = 1)\big\rangle \tag{1}$$

which is similar to a full obligation but contains a flag called PartialFlag which, if set to 1, describes a partial obligation not to be included in the clearing process. There is no defined receiver.

7.5 Withdrawal Phase

The **withdrawal phase** happens at the following novation cycle after the trader's withdrawal request has been received by the settlement network, i.e., if a withdrawal request is received at the ℓ-th novation cycle, it is processed at the $(\ell + 1)$-th novation cycle. Upon receiving a withdrawal request from a trader, the settlement network waits for the next novation cycle to ensure that the future free balance of the trader is enough to perform the withdrawal. After confirming that the free balance is enough for the withdrawal, each $\text{TEE}_k^{\mathcal{S}}$ checks the withdrawal request and verifies the signature, then picks the enough number of addresses from the chain-specific key list to gather the necessary amount. Once enough key pairs are collected, each $\text{TEE}_k^{\mathcal{S}}$ generates transactions to transfer the funds to the trader's specified destination, which are also freshly and jointly generated by the TEEs in $\mathcal{S}$. The Withdrawal phase is described in the Algorithm outlined in 8 and the withdrawal process flow is depicted in Figure 12.

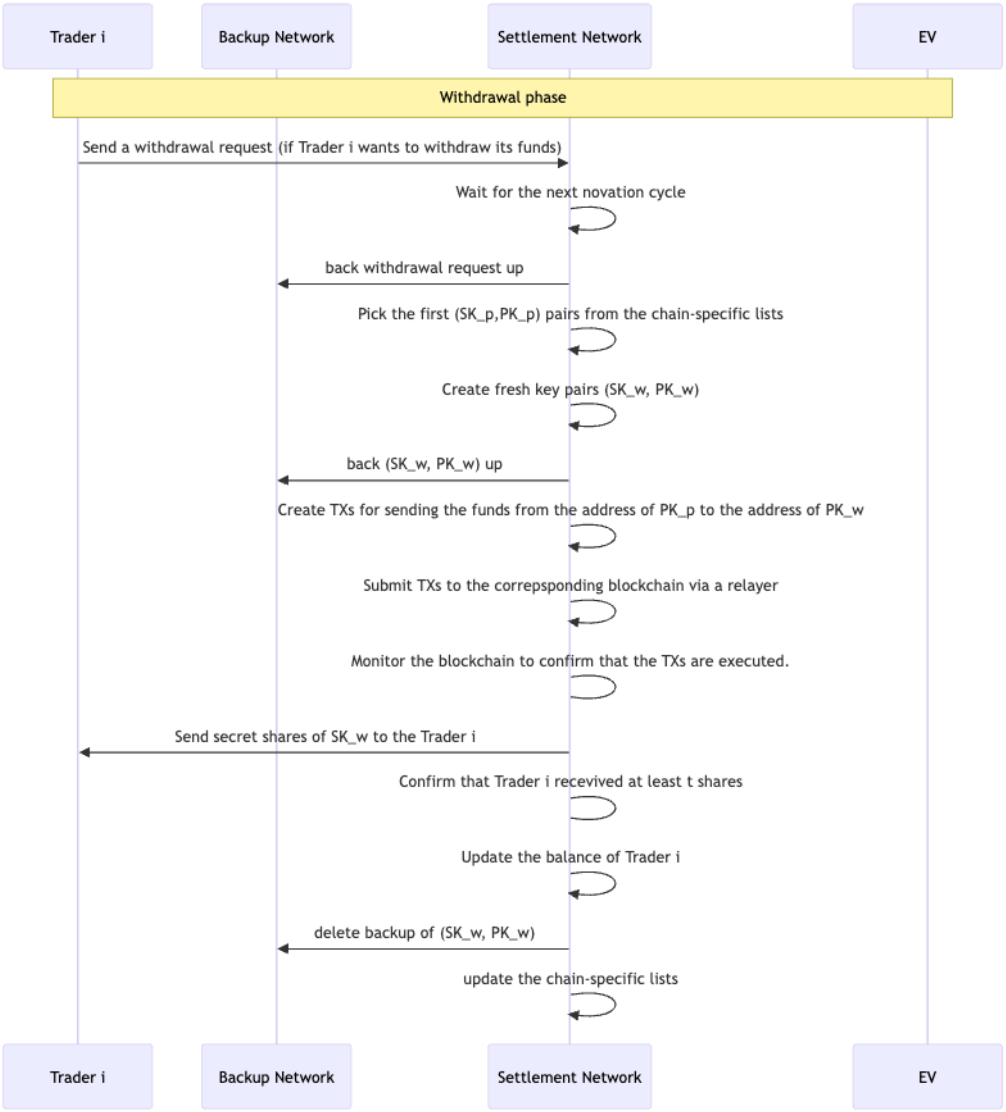


Figure 12: Withdrawal Sequence Diagram.

8 Compliance Details (Backup Network $\mathcal{B}$)

The backup network $\mathcal{B}$ is a consensus network run by M nodes (TEEs of different brands). Each node in the backup network $\mathcal{B}$ participates in a DKG for threshold encryption scheme described in preliminaries section above. After participating the DKG protocol, each j^{th} TEE in the backup network, i.e., $\text{TEE}_j^{\mathcal{B}}$, obtains an encryption key and a partial decryption key, i.e., $(ek_{\mathcal{B}}, dk_{\mathcal{B},j})$.

Each $\text{TEE}_j^{\mathcal{B}}$ maintains the key backups, the balances and obligations lists as leaves of SMTs. These data can only be sent by the nodes of settlement network $\mathcal{S}$. Note that EVs, KYC-ed traders, and the nodes of the settlement network $\mathcal{S}$ have access to the backup network $\mathcal{B}$. Traders can only request their key backups, while the nodes of the settlement network $\mathcal{S}$ are authorized to read and write data to the backup network $\mathcal{B}$. And the EVs can only request trader's current balances.

In the backup network $\mathcal{B}$, we use three SMTs to store the trader data, the TEE key backups and the obligation lists:

8.1 SMT for Trader Data

The leaf A that corresponds to a trader T_i's data is placed according to the path $H_{path} = \text{Hash}(\text{Cred}_i)$. This leaf A includes the root of another sub-SMT, which stores all trader related data, i.e., Cred_i, Balance_i, DR_i, WR_i, and $\text{KeyBackup}_{i,k}$ for $k = 1, \ldots, N$ as leafs.

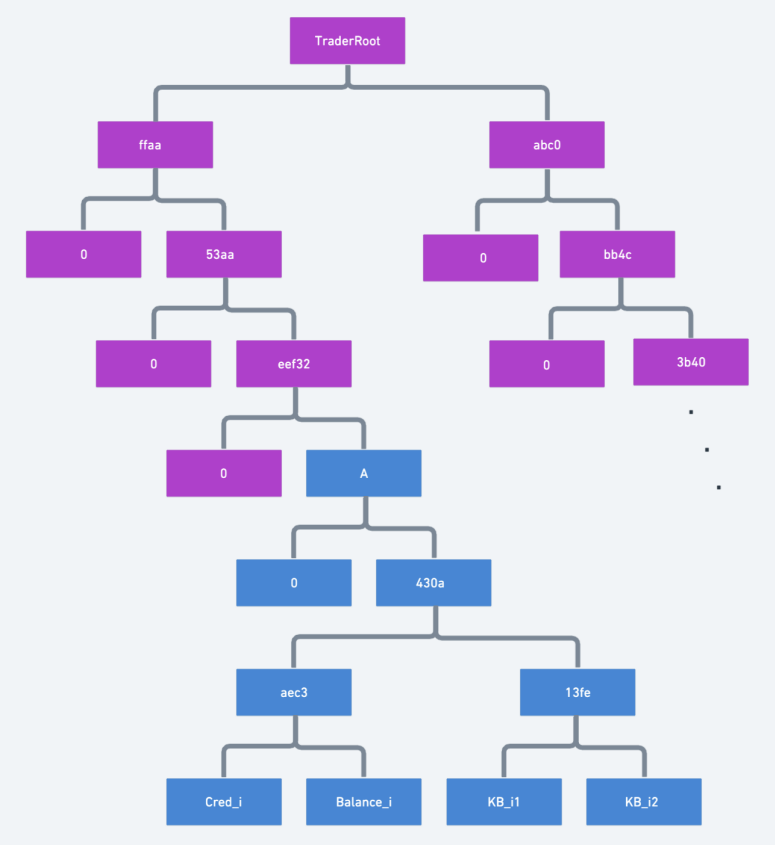


Figure 13: SMT Structure for Trader Data

In 13, the blue sub-SMT is for an example trader's data, which includes Cred_i, Balance_i, $\text{KeyBackup}_{i,1}$, and $\text{KeyBackup}_{i,2}$. Similarly, these data are also placed using the same

approach, e.g, to place the Balance_i compute the $\text{Hash}(\text{Cred}_i, \text{Balance}_i)$.

For n traders, the depth of the pink SMT would be $\log n$. For N TEEs in the settlement network, the depth of the blue sub-SMT would be $\sim \log N$. In total, the overall depth would be $\sim (\log n + \log N)$, which means a proof of inclusion (or non-inclusion) would include $\sim (\log n + \log N)$ hash values (possibly 32 bytes each), and computation would require computing $\sim (\log n + \log N)$ hashes.

8.2 SMT for TEE Key Backups

For TEE key backups of the form:

$$\text{KeyBackup}^{\mathcal{S}}_{w,k} = \left\langle \text{“KeyBackup"}, \text{TEE}^{\mathcal{S}}_k, PK_w, \text{Enc}_{ek_{\mathcal{B}}} \left(\text{Enc}_{ek_{\text{TEE}_k}} \left(SK_{w,k} \right) \right) \right\rangle$$

The path to node A would be $H^{\text{pink}}_{path} = \text{Hash}(\text{TEE}^{\mathcal{S}}_k)$. The blue sub-SMT tree places leafs according to path $H^{\text{blue}}_{path} = \text{Hash}(\text{PK}_w)$.

For N TEEs in the settlement network, the depth of the pink SMT would be $\log N$. For ℓ key backups, the depth of the blue sub-SMT would be $\log \ell$. The total depth would be $(\log N + \log \ell)$.

8.3 SMT for Obligation List Backups

For obligation list backups of the form:

$$(\langle \text{“OLBackup"}, EV_k, \text{NovationCycleId}, O = \{o_1, \cdots, o_n\} \rangle, \sigma, PK_{\text{EV}})$$

The SMT places leaves according to path $H^{\text{pink}}_{path} = \text{Hash}(EV_k, \text{NovationCycleId})$. Assuming maximum c novation cycles, the SMT depth would be $\log c$.

For compliance reasons, the backup network $\mathcal{B}$ stores and periodically (e.g., per day) updates its state root, i.e., root of the roots of SMTs, on a smart contract. Using the state root, one can prove any inclusion and non-inclusion of any data.

8.4 Network Operations

8.4.1 Key Backups

Upon receiving a key backup, each $\text{TEE}^{\mathcal{B}}_j$:

1. Gossips it through the backup network $\mathcal{B}$
2. Stores the backup to the corresponding SMT
3. Computes the state root
4. Reaches consensus on the state root

8.4.2 Backup Requests

See Algorithm 9.

8.4.3 Balance and Obligation Updates

Upon receiving a balance update, each $\text{TEE}^{\mathcal{B}}_j$:

1. Gossips it through the backup network $\mathcal{B}$
2. Stores the balance to the corresponding SMT

3. Computes the state root

4. Reaches consensus on the state root

Similarly for obligation updates, each $\text{TEE}_j^{\mathcal{B}}$ follows the same process but stores the obligation list in its corresponding SMT.

8.4.4 Balance Requests from EVs

EVs can request trader balances by sending a balance request with the trader's credential, see Algorithm 10.

8.4.5 Authorized Party Data Requests

An authorized entity must first obtain approval from a security council. The council members perform DKG to generate a signing key pair, with each member i receiving partial signing key $sk_{auth,i}$ and public key pk_{auth}. The council verifies real-world data (e.g., court order), translates it to specific data requests, and jointly signs the request - Algorithm 11.

9 Algorithms

Algorithm 1 Deposit Protocol

T_i generates a deposit request $\mathrm{DR}_i = \langle \text{“DepositRequest"}, \mathrm{Cred}_i, \mathrm{ChainID}_i \rangle$ and a signature σ_i on DR_i
T_i sends $(\mathrm{DR}_i, \sigma_i)$ to $\mathcal{S}$
Upon receiving $(\mathrm{DR}_i, \sigma_i)$, each $\mathrm{TEE}_k^{\mathcal{S}}$ performs:
if Cred_i is not valid **then**
  sends notification to trader T_i
  aborts the protocol
end if
if σ_i is not valid **then**
  sends notification to trader T_i
  aborts the protocol
end if
Gossips DR_i and σ_i to network $\mathcal{S}$
Creates (SK_i, PK_i) according to blockchain spec with id $\mathrm{ChainID}_i$
Keeps $SK_{i,k}$ secret
Computes $\mathrm{KeyBackup}_{i,k} = \big\langle \text{“KeyBackup"}, \mathrm{Cred}_i, PK_i, \mathrm{Enc}_{ek_{\mathcal{B}}}(\mathrm{Enc}_{pk_i}(SK_{i,k})) \big\rangle$
Sends $\mathrm{KeyBackup}_{i,k}$ to backup network $\mathcal{B}$
Sends PK_i to trader T_i confidentially
Trader T_i deposits funds to address of PK_i
$\mathrm{TEE}_k^{\mathcal{S}}$ monitors $\mathrm{Balance}_{PK_i}$ via blockchain client
if $\mathrm{Balance}_{PK_i} \neq 0$ **then**
  Update: $\mathrm{Balance}_i[\mathrm{FreeBalance}_i][\mathrm{ChainID}_i][\mathrm{Token}_i] += \mathrm{Balance}_{PK_i}$
  Add PK_i to $\mathcal{L}^{pk}_{\mathrm{ChainID}_i}$ and $SK_{i,k}$ to $\mathcal{L}^{sk}_{\mathrm{ChainID}_i,k}$
  Order key pairs by balance in descending order
  Reach consensus on $\mathrm{Balance}_i$ and $\mathcal{L}^{pk}_{\mathrm{ChainID}_i}$
  Send $(\text{“BalanceUpdate"}, \mathrm{Cred}_i, \mathrm{Enc}_{ek_{\mathcal{B}}}(\mathrm{Balance}_i))$ to $\mathcal{B}$
  Delete $\mathrm{KeyBackup}_{i,k}$ from backup network $\mathcal{B}$
  Compute new backup
  $\mathrm{KeyBackup}^{\mathcal{S}}_{i,k} = \big\langle \text{“KeyBackup"}, \mathrm{TEE}_k^{\mathcal{S}}, PK_i,$
  $\mathrm{Enc}_{ek_{\mathcal{B}}}(\mathrm{Enc}_{ek_k}(SK_{i,k})) \big\rangle$
  Send $\mathrm{KeyBackup}^{\mathcal{S}}_{i,k}$ to backup network $\mathcal{B}$
  Send $(\text{“DRBackup"}, \mathrm{Cred}_i, \mathrm{Enc}_{pk_{\mathcal{B}}}(\mathrm{DR}_i, \sigma_i, PK_i))$ to network $\mathcal{B}$
end if

Algorithm 2 Novation Cycle Protocol

for each novation cycle **do**
  NovationCycleID ← NovationCycleID + 1
  Each EV sends $O_{\text{init}} = \{o_1, \ldots, o_n\}$ with signature $\sigma = \text{Sig}_{SK_{\text{EV}}}(O_{\text{init}})$
  Wait for novation cycle end
  $b \leftarrow \text{VerifySig}_{PK_{\text{EV}}}(\sigma, O_{\text{init}})$
  if $b = 0$ **then**
    Send “invalid signature" to EV
    abort
  end if
  Send $\langle$“OLBackup", EV, NovationCycleId, $O_{\text{init}}, \sigma, \text{PK}_{\text{EV}}\rangle$ to $\mathcal{B}$
  Aggregate all O_{init} into one list
  for each (ChainID, Token) tuple **do**
    $G_{\text{init}} \leftarrow \text{constructGraph}(O_{\text{init}})$
    $\{G_{\text{inter}}, C\} \leftarrow \text{digraphToDAG}(G_{\text{init}})$
    $\{G_{\text{final}}, E_{\text{multi}}\} \leftarrow \text{removeParallelLines}(G_{\text{inter}})$
    for each $e_x \in E_{\text{final}}$ **do**
      $o_{\text{new}} \leftarrow ((\text{ChainID}, \text{Token}), T_i, T_j, \text{Amount})$
      Append o_{new} to $O_{\text{final}}^{\text{ChainID, Token}}$
    end for
    $\pi^{\text{ChainID, Token}} \leftarrow \text{Gen}_{\text{ZKP}}(\text{PubInputs}, \text{PrivInputs})$
    Append $\pi^{\text{ChainID, Token}}$ to Π
    Append obligations to O_{final}
  end for
  for each $o_x \in O_{\text{final}}$ **do**
    ChainID ← o_x[ChainID], Token ← o_x[Token]
    Amount ← o_x[Amount]
    $\text{Balance}_i[\text{FreeBalance}_i][\text{ChainID}][\text{Token}]$
    $\leftarrow \text{Balance}_i[\text{FreeBalance}_i][\text{ChainID}][\text{Token}] - \text{Amount}$
    $\text{Balance}_j[\text{FreeBalance}_j][\text{ChainID}][\text{Token}]$
    $\leftarrow \text{Balance}_j[\text{FreeBalance}_j][\text{ChainID}][\text{Token}] + \text{Amount}$
  end for
  Gossip new state and proofs through $\mathcal{S}$
  if $|\Pi| \neq$ Number of ChainID, Token's in O_{init} **then**
    Reject state
  end if
  if $\text{Verify}_{PK_{\text{EV}}}(\sigma, O_{\text{init}}) = 0$ **then**
    Reject state
  end if
  for each ChainID, Token proof **do**
    if $O_{\text{init}}^{\text{ChainID, Token}} \not\subseteq O_{\text{init}}$ **then**
      Reject state
    end if
    if $O_{\text{final}}^{\text{ChainID, Token}} \not\subseteq O_{\text{final}}$ **then**
      Reject state
    end if
    if $\text{Verify}_{\text{ZKP}}(\pi^{\text{ChainID, Token}}) = 0$ **then**
      Reject state
    end if
  end for
  Reach consensus on balances
  for each updated trader i **do**
    Send (“BalanceUpdate", $\text{Cred}_i, \text{Enc}_{ek_{\mathcal{B}}}(\text{Balance}_i)$) to $\mathcal{B}$
  end for
  Send new balances to EV
end for

Algorithm 3 constructGraph

function CONSTRUCTGRAPH(O)
  $V \leftarrow \emptyset$ ▷ Initialize empty vertex set
  $E \leftarrow \emptyset$ ▷ Initialize empty edge set
  for each obligation $o_x \in O$ **do**
    $V \leftarrow V \cup \{T_i, T_j\}$ ▷ Add unique traders
    $e \leftarrow (T_i, T_j, \text{Amount})$
    $E \leftarrow E \cup \{e\}$ ▷ Add edge
  end for
  Define $o(e_x) = \text{Amount}_x$ ▷ Obligation function
  return $G = (V, E, o)$
end function

Algorithm 4 digraphToDAG

function DIGRAPHTODAG(G)
  $C \leftarrow \text{Tarjan}(G)$ ▷ Get strongly connected components
  for each cycle $c \in C$ **do**
    $e_{\min} \leftarrow \arg\min_{e \in c} o(e)$ ▷ Find minimum obligation
    if $e_{\min} \in G$ **then**
      Remove $e_{\min}$ from G
      for each remaining edge $e \in c$ **do**
        $\text{Amount}_e \leftarrow \text{Amount}_e - o(e_{\min})$
        Update G with new amount
      end for
    end if
  end for
  return $\{G, C\}$
end function

Algorithm 5 removeParallelLines

function REMOVEPARALLELLINES(G)
  $E_{\text{multi}} \leftarrow \emptyset$ ▷ Store merged edges
  for each pair $(e_{i,j}, e_{j,i}) \in E$ **do**
    $\text{Amount}_{\text{new}} \leftarrow o(e_{i,j}) - o(e_{j,i})$
    if $\text{Amount}_{\text{new}} > 0$ **then**
      $e_{\text{new}} \leftarrow (T_i, T_j, \text{Amount}_{\text{new}})$
    else if $\text{Amount}_{\text{new}} = 0$ **then**
      Remove $e_{i,j}$ and $e_{j,i}$ from E
      continue
    else
      $e_{\text{new}} \leftarrow (T_j, T_i, |\text{Amount}_{\text{new}}|)$
    end if
    Remove $e_{i,j}$ and $e_{j,i}$ from E
    Add e_{new} to E
    $E_{\text{multi}} \leftarrow E_{\text{multi}} \cup \{\{e_{i,j}, e_{j,i}, e_{\text{new}}\}\}$
  end for
  return $\{G, E_{\text{multi}}\}$
end function

Algorithm 6 Zero-Knowledge Proof

Public Inputs:

Initial obligation list $O = \{o_1, \dots o_n\}$

Final obligation list $O_{\text{final}} = \{o_{1'}, \dots, o_{n'}\}$

Private Inputs:

Set of unique cycles $C = \{\{e_x, e_y, e_z\}, \{e_a, e_b, e_c, e_d\} \cdots\}$

Set of multigraph edges $E_{\text{multi}} = \{\{e_{i,j}, e_{j,i}, e_{i,j\ \text{new}}\}, \cdots\}$

Initial graph $G_{\text{init}} = (V, E, o)$

Intermediate graph $G_{\text{inter}} = (V, E, o)$

Final graph $G_{\text{final}} = (V, E, o)$

Relations to Prove:

1. Size validity:

$|O_{\text{final}}| \leq |O|$

2. Graph correspondence:

$|E| = |O|$

$\forall o_x \in O : \mathrm{T}_{i_x}, \mathrm{T}_{j_x} \in V$

$\forall e \in G_{\text{init}}, o \in O_{\text{init}} : \mathrm{T}_{i_o} = \mathrm{T}_{i_e} \wedge \mathrm{T}_{j_o} = \mathrm{T}_{j_e} \wedge o(e) = \text{Amount}$

3. Intermediate graph correctness:

$\forall \{e_1, e_2, \cdots\} \in C : e_1, e_2, \cdots \in G_{\text{init}}$

For minimum edge $e_1 \in G_{\text{init}}$ in cycle: $e_1 \notin G_{\text{inter}}$

$\forall e_2 \in G_{\text{init}} \cap G_{\text{inter}}, c \in C : e_{2\ \text{inter}} = e_{2\ \text{init}} - \min(c)$

4. Final graph correctness:

$\forall e \in E_{\text{multi}} : e \in G_{\text{inter}}$

$\forall \{e_1, e_2, \cdots\} \in E_{\text{multi}}, \exists e_{\text{merged}} \in G_{\text{final}} : o(e_1) + o(e_2) + \cdots = o(e_{\text{merged}})$

5. Final obligation list correctness:

$|E_{\text{final}}| = |O_{\text{final}}|$

$\forall T \in V_{\text{final}} : T$ appears in obligation list

$\forall e \in G_{\text{final}}, o \in O_{\text{final}} : \mathrm{T}_{i_o} = \mathrm{T}_{i_e} \wedge \mathrm{T}_{j_o} = \mathrm{T}_{j_e} \wedge o(e) = \text{Amount}$

Algorithm 7 Partial Obligation Processing

Step 1: Inform $\mathcal{S}$ of encumbrance
EV sends o_{partial} and $\sigma = \text{Sign}_{\text{sk}_{\text{EV}}}(o_{\text{partial}})$ to $\mathcal{S}$
if PartialFlag $= 0$ **then**
  Abort process
end if
$b \leftarrow \text{Verify}_{PK_{\text{EV}}}(\sigma, o_{\text{partial}})$
if $b = 1$ **then**
  Send message to EV (invalid signature)
  Abort process
end if
Update: $\text{Balance}_i[\text{EncumberedBalance}_i][\text{ChainID}][\text{Token}] -= \text{Amount}$
Send $(\text{“BalanceUpdate"}, \text{Cred}_i, \text{Enc}_{ek_{\mathcal{B}}}(\text{Balance}_i))$ to $\mathcal{B}$
Step 2: Perform clearing
EV constructs $o = \langle(\text{ChainID, Token}), T_i, T_j, \text{Amount}, \text{PartialFlag} = 0\rangle$
Include o in obligation list O sent to $\mathcal{S}$
Perform clearing process
Step 3: Remove encumbrance
EV sends reverse o_{partial} and $\sigma = \text{Sign}_{\text{sk}_{\text{EV}}}(o_{\text{partial}})$
$b \leftarrow \text{Verify}_{PK_{\text{EV}}}(\sigma, o_{\text{partial}})$
if $b = 1$ **then**
  Send message to EV (invalid signature)
  Abort process
end if
Update: $\text{Balance}_i[\text{EncumberedBalance}_i][\text{ChainID}][\text{Token}] += \text{Amount}$
Send $(\text{“BalanceUpdate"}, \text{Cred}_i, \text{Enc}_{ek_{\mathcal{B}}}(\text{Balance}_i))$ to $\mathcal{B}$

Algorithm 8 Withdrawal Protocol

T_i sends withdrawal request:
  $\mathrm{WR}_i = \langle$"WithdrawRequest", Cred_i,
  $(\mathrm{ChainID}_w, \mathrm{Token}_w, \mathrm{Amount}_w)\rangle$,
  and signature δ_i
Each $\mathrm{TEE}_k^{\mathcal{S}}$ waits for next novation cycle
if Cred_i is not valid **or** δ_i is not valid **or then**
  $\mathrm{Amount}_w > \mathrm{Balance}_i[\mathrm{FreeBalance}_i][\mathrm{ChainID}_w][\mathrm{Token}_w]$
  $+\mathrm{Balance}_i[\mathrm{EncumberedBalance}_i][\mathrm{ChainID}_w][\mathrm{Token}_w]$
  Send notification to trader T_i
  Abort protocol
end if
Gossip $(\mathrm{WR}_i, \delta_i)$ to network $\mathcal{S}$
if $\mathcal{L}^{pk}_{\mathrm{ChainID}_w}$ has entry j with balance $= \mathrm{Amount}_w$ **then**
  Update $\mathrm{Balance}_i[\mathrm{FreeBalance}_i][\mathrm{ChainID}_w][\mathrm{Token}_w] -= \mathrm{Amount}_w$
  Delete entries from $\mathcal{L}^{sk}_{\mathrm{ChainID}_w,k}[j]$ and $\mathcal{L}^{pk}_{\mathrm{ChainID}_w}[j]$
  Reorder key pairs by balance (descending) and public key
  Reach consensus on $\mathrm{Balance}_i$ and $\mathcal{L}^{pk}_{\mathrm{ChainID}_i}$
  Send $\mathcal{L}^{sk}_{\mathrm{ChainID}_i,k}[j]$ to T_i confidentially
  Delete backups from $\mathcal{B}$
  Send backup tuple to $\mathcal{B}$
  return
end if
$ctr \leftarrow 0$
$\mathrm{Sum} \leftarrow \mathrm{Balance}_{\mathcal{L}^{pk}_{\mathrm{ChainID}_w}[0]}$
while $\mathrm{Sum} < \mathrm{Amount}_w$ **do**
  $ctr \leftarrow ctr + 1$
  Get $\mathcal{L}_{\mathrm{ChainID}^{pk}_w}[ctr]$
  $\mathrm{Sum} \leftarrow \mathrm{Sum} + \mathrm{Balance}_{\mathcal{L}^{pk}_{\mathrm{ChainID}_w}[ctr]}$
end while
Create $(SK_{w,k}, PK_w)$ via DKG
Compute backup $\mathrm{KeyBackup}^{\mathcal{S}}_{w,k}$
Send backup to $\mathcal{B}$
$acc \leftarrow 0$
for x in range(ctr) **do**
  Generate TX_x to send $\mathrm{Balance}_{\mathcal{L}^{pk}_{\mathrm{ChainID}_w}[x]}$ to PK_w
  Append TX_x to TXlist
  $acc \leftarrow acc + \mathrm{Balance}_{\mathcal{L}^{pk}_{\mathrm{ChainID}_w}[x]}$
end for
Generate final TX_{ctr} to send $(\mathrm{Amount}_w - acc)$ to PK_w
Append TX_{ctr} to TXlist
Send TXlist to relayer
Wait for transactions to execute
Update $\mathrm{Balance}_i[\mathrm{FreeBalance}_i][\mathrm{ChainID}_w][\mathrm{Token}_w] -= \mathrm{Amount}_w$
Delete used keys from lists
Reorder remaining keys
Reach consensus on balances and lists
Send $SK_{w,k}$ to T_i confidentially
Clean up backups in $\mathcal{B}$
Send final backup tuple to $\mathcal{B}$
T_i receives $\geq t$ partial keys and reconstructs SK_w

Algorithm 9 Backup Request Processing

function ProcessBackupRequest(BR_i, ς_i)
  $BR_i \leftarrow \langle$"BackupRequest", $\text{Cred}_i, PK_i\rangle$
  if ς_i is not valid **then**
    Send notification to T_i
    abort
  end if
  if Cred_i is not valid **then**
    abort
  end if
  if Cred_i is valid **then**
    Gossip BR_i through network $\mathcal{B}$
    if no backup exists for (Cred_i, PK_i) **then**
      Send non-inclusion proof to T_i
    else
      for $k = 1$ to N **do**
        $A_{k,j} \leftarrow dk_{\mathcal{B},j} \cdot A_k$
        Send $(A_{k,j}, B_k)$ to T_i
      end for
    end if
  end if
end function
function ProcessPartialDecryptions($\{A_{k,j}\}_{j\in\mathcal{Q}}$)
  for $k = 1$ to N **do**
    $dk_{\mathcal{B}} \cdot A_k \leftarrow \sum_{j\in\mathcal{Q}} \lambda_j \cdot A_{k,j}$
    $m_k \leftarrow B_k - H(dk_{\mathcal{B}} \cdot A_k)$
    $SK_{i,k} \leftarrow \text{Dec}_{sk_i}(m_k)$
  end for
  $SK_i \leftarrow \sum_{k\in[N]} \lambda_k SK_{i,k}$
end function

Algorithm 10 Balance Request Processing

function ProcessBalanceRequest(BalanceRequest$_k$, ϖ_k)
  BalanceRequest$_k \leftarrow \langle$"BalanceRequest", PK_{EV_k}, Cred$_i\rangle$
  if ϖ_k not valid OR Cred$_i$ not valid **then**
    Send notification to EV_k
    abort
  end if
  Gossip BalanceRequest$_k$ through network $\mathcal{B}$
  if no balance exists for Cred$_i$ **then**
    Send non-inclusion proof to EV_k
    abort
  end if
  if balance exists **then**
    $A_{i,j} \leftarrow dk_{\mathcal{B},j} \cdot A_i$
    Send $(A_{i,j}, B_i)$ to EV_k
    Send inclusion proof against state root
  end if
end function
function ProcessBalanceResponse($\{A_{i,j}\}_{j\in\mathcal{Q}}$, proof)
  Verify Merkle proof
  $dk_{\mathcal{B}} \cdot A_i \leftarrow \sum_{j\in\mathcal{Q}} \lambda_j \cdot A_{i,j}$
  $m_i \leftarrow B_i - H(dk_{\mathcal{B}} \cdot A_i)$
end function

Algorithm 11 Authorized Data Request Processing

function ProcessAuthRequest(DataRequest, Π)
  DataRequest $\leftarrow$ (Cred$_i$, RequestedData, Π)
  if Verify$_{pk_{auth}}$(DataRequest, Π) $= 0$ **then**
    abort
  end if
  if RequestedData not in storage **then**
    Send non-inclusion proof against state root
    abort
  end if
  Participate in threshold decryption
  Send partial decryptions to authorized party
  Send Merkle proof of inclusion against state root
end function
function ProcessAuthResponse(proofs, decryptions)
  Verify Merkle proofs
  Perform Lagrange interpolation over partial decryptions
  Obtain RequestedData
end function

References

[BB] J. Baylina and M. Bellés. Sparse merkle trees. https://docs.iden3.io/publications/pdfs/Merkle-Tree.pdf.

[BDA+24] E. Buchman, P. Dini, S. Ahmed, T. Fleischman, and A. Miller. Cycles protocol: A peer-to-peer electronic clearing system. https://cycles.money/whitepaper.pdf, November 2024.

[Bel24] S. Bellemare. Research directions for verifiable crypto-physically secure tees. *arXiv preprint arXiv:2410.03183*, 2024. doi:10.48550/arXiv.2410.03183.

[But18] V. Buterin. Optimizing sparse merkle trees. https://ethresear.ch/t/optimizing-sparse-merkle-trees/3751, 2018.

[Cha24] Chainalysis. The 2024 crypto crime report, 2024. Page 35.

[DKLS19] J. Doerner, Y. Kondi, E. Lee, and A. Shelat. Threshold ecdsa from ecdsa assumptions: The multiparty case. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 1051–1066. IEEE, 2019. doi:10.1109/SP.2019.00024.

[DPP16] R. Dahlberg, T. Pulls, and R. Peeters. Efficient sparse merkle trees: Caching strategies and secure (non-)membership proofs. https://eprint.iacr.org/2016/683, 2016. Cryptology ePrint Archive: Report 2016/683.

[ElG85] Taher ElGamal. A public-key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985. CiteSeerX 10.1.1.476.4791; S2CID 2973271; (conference version appeared in CRYPTO'84, pp. 10–18). doi:10.1109/TIT.1985.1057074.

[FD21] T. Fleischman and P. Dini. Mathematical foundations for balancing the payment system in the trade credit market. *Journal of Risk and Financial Management*, 14(452), 2021. doi:10.3390/jrfm14090452.

[Gai23] N. Gailly. Efficient threshold bls & encryption. https://hackmd.io/@nikkolasg/SJb2mIhNj, 2023.

[Gro16] J. Groth. On the size of pairing-based non-interactive arguments. In *Advances in Cryptology - EUROCRYPT 2016*, pages 305–326, 2016. doi:10.1007/978-3-662-49896-5_11.

[GWC19] A. Gabizon, Z. J. Williamson, and O. Ciobotaru. Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. https://eprint.iacr.org/2019/953, 2019. Cryptology ePrint Archive.

[KG21] C. Komlo and I. Goldberg. Frost: Flexible round-optimized schnorr threshold signatures. In *Selected Areas in Cryptography (SAC 2020)*, volume 12804 of *Lecture Notes in Computer Science*. Springer, 2021. doi:10.1007/978-3-030-81652-0_2.