Decentralized Access Control Infrastructure for Enterprise Digital Asset Management

Chirag Madaan
chirag@cypherock.com

Rohan Agarwal
rohan@cypherock.com

Vipul Saini
vipul@cypherock.com

Ujjwal Kumar
ujjwal@cypherock.com

March 2024

Abstract

With the rapidly evolving landscape of cryptography, blockchain technology has advanced to cater to diverse user requirements, leading to the emergence of a multi-chain ecosystem featuring various use cases characterized by distinct transaction speed and decentralization trade-offs. At the heart of this evolution lies digital signature schemes, responsible for safeguarding blockchain-based assets such as ECDSA, Schnorr, and EdDSA, among others.

However, a critical gap exists in the current landscape — there is no solution empowering a consortium of entities to collectively manage or generate digital signatures for diverse digital assets in a distributed manner with dynamic threshold settings, all while mitigating counter-party risks. Existing threshold signature schemes impose a fixed threshold during the key generation phase, limiting the adaptability of threshold settings for the subsequent signature phase. Attempts to address this challenge often involve relinquishing signature generation control either partially or entirely from the participating parties, introducing vulnerabilities that could jeopardize digital assets in the event of network disruptions.

Addressing this gap, our work introduces an innovative infrastructure that allows a group of users to programmatically define and manage access control policies, supported by a blockchain network dedicated to policy enforcement. This network is uniquely designed to prevent any entity, including itself, from autonomously generating digital signatures, thereby mitigating counter-party risks and enhancing asset security. This system is particularly suited for enterprise contexts, where collaborative asset oversight and policy adherence are essential. Our solution marks a significant stride in the realm of blockchain technology, paving the way for more

sophisticated and secure digital asset management in a rapidly evolving digital landscape.

Keywords: Linearly homomorphic encryption, Class groups, Threshold signature schemes, Blockchain.

1 Introduction

With Bitcoin and Ethereum serving as prominent examples, the proliferation of blockchain technologies has not only ushered in a new era of financial innovation but has also given rise to continuous research and development in underlying cryptographic schemes. These ongoing efforts are aimed at further enhancing the axes of the scalability trilemma, striving to strike an optimal balance between decentralization, security, and transaction speed. This relentless pursuit of cryptographic innovation has catalyzed the emergence of a diverse and dynamic blockchain ecosystem, where various chains coexist to provide versatile solutions for decentralized applications. As we look to the future, it becomes increasingly evident that the coexistence of multiple blockchains will not only be the norm but also the catalyst for groundbreaking advancements in decentralized technologies, providing fertile ground for the creation of innovative, powerful, and highly specialized decentralized applications.

In the multichain landscape, the need for non-custodial wallets has become imperative for managing assets across various blockchains, catering to both individuals and institutions. Institutional scenarios often involve a collective of individuals jointly overseeing assets while adhering to predefined policy settings. These policy settings serve as the governing rules dictating when and how digital assets possessed by the group on the blockchain, can be spent, factoring in considerations such as time frames, transaction amounts, and more. Remarkably, the existing wallets in the market have yet to offer a comprehensive solution that combines full self-custody for groups of individuals managing digital assets with a flexible and customizable policy layer that is chain-agnostic. In essence, there exists an unmet demand for a solution that provides robust security for digital assets while allowing users to tailor policy settings to their specific needs.

2 Requirements

We have identified the following fundamental requirements that the non-custodial solution should fulfill for an institution or a group of members to be able to manage their digital assets:

- The group or the institution should be able to create/update the policy according to which the digital assets possessed by them are spent.
- No external entity or network should possess all the critical cryptographic key materials or the capability to generate valid signatures by itself. Only

the group or the institution should own the their digital assets ensuring full self-custody.

- The policy-enforcement layer should be decentralised in order to prevent a single point of failure and make the compromisation of the network near impossible for a malicious actor.
- To policy-enforcement decentralised network should be large and should support joining and leaving of nodes at any time. There should be minimal or zero P2P communication between the nodes to make the speed of the protocol more practical.
- The availability of digital assets held by the group should not depend on the availability of any external factors including the policy-enforcement layer if all the members of the group are present.
- The solution should support spending of digital assets on almost all blockchains to open up the possibilities of various applications and hence improve the usefulness of the solution.
- The solution should allow the generation of group receive/change addresses to enhance privacy and security for each transaction.

Keeping these requirements in mind, we have worked on building a fast decentralized policy-enforcement blockchain network for institutions or a group of members to jointly manage their digital assets on multiple chains with the ability to securely update the policy settings whenever required. Our solution ensures that the digital assets are cryptographically possessed only by the group and are available even when our policy-enforcement network is down, assuming the group members are available. We have utilized the latest research and development of various secure cryptographic primitives such as Hierarchically Threshold Linearly Homomorphic Encryption based on class groups and Threshold Signature Generation schemes, to realize our architecture. The security of handling individual user shares is greatly improved by integrating the hardware security provided by the Cypherock devices. In this paper, we describe our approach to generate ECDSA signatures in a group setting as it is the most popular digital signature algorithm currently used on the blockchains, but the same architecture can be realized for other digital signature algorithms like Schnorr and EdDSA as well.

3 Related Work

The recent advancements in threshold signature schemes, particularly for Elliptic Curve Digital Signature Algorithm (ECDSA), have been significant, with multiple efficient protocols being proposed in a relatively short span. [10] [9] [6] [5] [7] [4]. These protocols vary in their assumptions, adversarial models, building blocks, and performance metrics, providing a broad spectrum of choices depending on the specific requirements of the use case at hand.

Paper	Rounds	Space	Time	Channels	IA	Assumptions
LN18 [10]	7	$O(n^2)$	$O(n^2)$	P2P	No	DCR or OT, DDH
GG19 [9]	8	$O(n^2)$	$O(n^2)$	P2P & BC	No	sRSA, DCR, DDH
DKLS19 [6]	$\lceil \log t \rceil + 6$	$O(n^2)$	$O(n^2)$	P2P & BC	No	DH
DJN20 [5]	4	$O(n^2)$	$O(n^2)$	P2P	No	-
DKLS23 [7]	3	$O(n^2)$	$O(n^2)$	P2P	No	DH

Table 1: Comparison of some of the popular TSS protocols.

One major requirement for implementing true institutional self-custody with an access control layer is dynamic threshold, which is not possible via any of the above TSS protocols. There is very limited work done on securely enforcing dynamic threshold property. Moreover, we require the protocol to have identifiable abort to prevent malicious users in the network from performing DoS attacks. As can be seen in Table 1, most of the popular TSS protocols require high rounds of communication, high bandwidth and do not support identifiable abort.

There has been some work that focus on establishing a decentralised access control layer for general purpose use case [11] but the management of assets held be a group of users is not defined and there is no protection of funds availability in case of network outage.

3.1 Our Work

To address the limitations of previous works and fulfill the requirements described above where signing thresholds can be defined based on specific conditions within the distributed message signing phase, a cryptographic foundation is essential. This necessitates that the message signing phase should support different thresholds as defined in the policy, and should be enforced by the network of validators.

This further necessitates that during the distributed key generation phase, participants should not hold the private shares generated over the group polynomials, otherwise, participants will always be able to generate signatures with the lowest threshold established. In this paper, we propose an a secure group setup which results in encrypted group private key which is then used to generate encrypted signatures (See Figure 4). This process also generates additive shares of the group private key which are stored on individual Cypherock devices, protected against the loss and theft of the device, by further sharding of shares into the X1 cards. The additive shares are generated to help protect/recover the funds in the very rare case the policy enforcement network becomes unavailable.

The encryption scheme should have threshold linear homomorphic properties in the message space of the ECDSA which is now possible due to the recent advancements in this research field [3]. Using the properties of such a homomorphic encryption scheme, the participants will be able to generate an encrypted

signature following a suitable distributed threshold signature scheme from the encrypted shares with the help of the policy-enforcement blockchain network. During signature generation, the network only participates in providing randomness to the ephemeral key of the signature to prevent the calculation of private key from a signature by the malicious group members. The decryption of the signature will be done in a hierarchical threshold manner by the network only when the policy compliance checks are passed and the threshold number of parties in a group. This is done to prevent the generation of malicious signatures if somehow the network gets corrupted. The validator network itself is incapable of decrypting an encrypted signature due to the hierarchical distribution of decryption key among enterprise users and the network (See Figure 5). The final decrypted signature can then be utilized to transfer the digital assets on the respective blockchains. This approach helps enterprise achieve full self custody with a decentralized layer to enforce policies and regulations on the funds and the transactions.

We are currently working on researching efficient ZKP algorithms and implementing ZKP generation on Cypherock X1 devices to create an overall integrated system for digital asset management.

4 Cryptography Primitives

4.1 Hierarchical Threshold Linearly Homomorphic Encryption

In our approach, we extend the Threshold Linearly Homomorphic Scheme [2] [3] to be used hierarchically. The paper defines the following framework to setup the class groups:

- $G = \langle g \rangle$: cyclic group of order $s \cdot 2^k$, with $\gcd(2^k, s) = 1$
- $F = \langle f \rangle$: subgroup of 2^k-roots of unity of G, of order 2^k
- Efficient algorithm for computing discrete logarithms in F (x given f^x)
- $H = \langle h \rangle$: subgroup of 2^k-th powers of G of unknown odd order s
- $G \cong F \times H$
- $\tilde{s}$ is a known upper bound for s

HSM_{2k} assumption: Given $f, h, \tilde{s}$ and z_b, where $b \leftarrow \mathcal{U}\{0,1\}$,
$z_0 = h^x \cdot f^u$, and $z_1 = h^x$ for $x \leftarrow \mathcal{D}_H$ and $u \leftarrow \mathcal{U}((\mathbb{Z}/2^k\mathbb{Z})^\times)$
no PPT algorithm can decide b with probability significantly greater than $1/2$.

After setting up the required class groups, the paper describes the following mathematics of essential functions: `KeyGen`, `Encrypt`, `Decrypt`, `EvalAdd` and `EvalScal`.

4.1.1 KeyGen

- This function generates Threshold Additive Homomorphic Encryption (tAHE) key pairs. It either takes a private key sk as input or generates it randomly.
- A random private key sk is chosen in the private key bounds of the cryptosystem.
- The corresponding public key is computed as follows:

$$pk = h^{sk}$$

- The function then returns the key pairs.

4.1.2 Encrypt

- In our paper, we have referred to the encryption function as $\text{enc}(m, pk)$ where m is the message to be encrypted and pk is the encryption public key.
- A random number r is chosen in the private key bounds of the cryptosystem.
- The first part of the encrypted message is computed as: $c1 = h^r$.
- The second part of the encrypted message is computed as: $c2 = f^m \times pk^r$.
- The function returns $(c1, c2)$ as the encrypted message.

4.1.3 Decrypt

- In our paper, we have referred to the decryption function as $\text{dec}(e, sk)$ where e is the encrypted message i.e $e = (c1, c2)$ and sk is the decryption private key.
- Then the function computes $M = c2 \times c1^{-sk}$.
- If $M \notin F$, then the function returns $\perp$.
- Else, the function returns $\log_f(M)$.

4.1.4 EvalAdd

- This function adds two encrypted messages $e1$ and $e2$ and returns their encrypted sum.
- This function takes pk, $e_1 = (c1, c2)$ and $e2 = (c'_1, c'_2)$ as input.
- It then computes $c''_1 = c1 \times c'_1$ and $c''_2 = c2 \times c'_2$.

- A random number r is chosen in the private key bounds of the cryptosystem.
- The function returns $e = e1 + e2 = (c_1'' \times h^r, c_2'' \times pk^r)$.

4.1.5 EvalScal

- This function multiplies a non-encrypted scalar value with an encrypted message and returns the encrypted product.
- This function takes pk, $e = (c1, c2)$ and α as input.
- It then computes $c_1' = c1^{\alpha}$ and $c_2' = c2^{\alpha}$.
- A random number r is chosen in the private key bounds of the cryptosystem.
- The function returns $\alpha \times e = (c_1' \times h^r, c_2' \times pk^r)$.

4.1.6 Threshold Decryption

Let's say a value s is encrypted using public key pk to get $e = \text{enc}(s, pk) = (c1, c2)$ and the corresponding decryption private key sk such that $pk = h^{sk}$ is distributed using the Shamir secret sharing scheme with threshold value t, i.e., the i-th group member has the share $S_{sk}(i)$ such that any t number of shares can be interpolated to generate the value sk as:

$$sk = \sum_{i=1}^{t} (\lambda_i \times S_{sk}(i))$$

where,

$$\lambda_i = \prod_{1 \leq m \leq n, m \neq j} \frac{m}{m - j}$$

Instead of computing the decryption key sk using interpolation and then decrypting e value, each party can compute a partial decryption of e using their share as detailed below and finally, t number of partial decryptions of e can be multiplied together to compute the value of s.

The i-th group member can compute the partial decryption of e as: $d_i = c1^{-\lambda_i \times S_{sk}(i)}$ and broadcast it. In our paper, we refer to this process of computing partial decryption of e as $\text{partDec}(e, \lambda_i, S_{sk}(i))$. Finally, s can be computed as: $s = \log_f (c2 \times (\prod_{i=1}^{t} d_i))$ if $c2 \times (\prod_{i=1}^{t} d_i) \in F$, we refer to this process of computing final decrypted value from partial decryptions as $\text{aggPartDecs}(e, \{d_1, d_2, ..., d_i\})$ This function returns $\perp$ if $c2 \times (\prod_{i=1}^{t} d_i) \notin F$.

In our paper, the private decryption key sk is distributed hierarchically among User Group members and validators without the help of any trusted

dealer, but with the Distributed Key Generation (DKG) process as visualized in Figure 5. Even in this case, our functions partDec and aggPartDecs will work correctly to compute the final decrypted value if the encryption was performed using the global public key, i.e., $pk = pk_g^{t_p} \times pk_v = h^{sk_g^{t_p}} \times h^{sk_v}$ (View Figure 5).

4.2 Distributed Key Generation

This section describes the approach for a group of total n members to securely generate shares of group threshold additive homomorphic (tAHE) decryption key with the threshold value t, and make the group tAHE encryption key public. This implies that any t number of group members can re-generate the group tAHE decryption key, but they'll never be required to do so in the protocol explained later. Hence, the group tAHE decryption key is never computed and stored in its original form.

For the DKG process, parties engage in the following process:

1. Each group member, represented by the i-th member, creates its own random individual polynomial of degree $t-1$. The individual polynomial of the i-th member is represented as:

$$S_i(x) = r_{t-1} * x^{t-1} + r_{t-2} * x^{t-2} + ... + r_0$$

   The coefficients of this polynomial are randomly chosen in the secret key bounds of the tAHE cryptosystem.

2. Each group member then computes n number of shares for each member in the group including itself. The shares computed by the i-th member are represented as:

$$\{S_i(1), S_i(2), ..., S_i(n)\}$$

3. The i-th group member then transmits its polynomial's share for the j-th group member via a private and authenticated channel. In the same way, each group member shares its individual polynomial's shares to the other respective group members.

4. Each group member on receiving the shares from other group members securely, performs arithmetic addition of the shares to get the share of the group secret polynomial. The share of group secret polynomial of the i-th group member is:

$$S(i) = \sum_{j=1}^{n} S_j(i)$$

5. Every group member, represented by the i-th member, will have to broadcast their share of the group public key calculated as shown below:

$$Q_i = h^{S(i)}$$

6. Upon receiving the group public key shares, each group member, represented below as the i-th member, can run a check by interpolating in the exponent to verify its public key share.

$$Q_i = ExpInt(Q_1, Q_2, ...Q_n, i)$$

$$Q_i = Q_1^{\lambda_1} * Q_2^{\lambda_2} * ... * Q_n^{\lambda_n}$$

where,

$$\lambda_j = \prod_{1 \leq m \leq n, m \neq j} \frac{m-i}{m-j}$$

and i is the interpolation point.

7. If the check passes, then the group public key is generated by interpolating in the exponent as follows:

$$Q = ExpInt(Q_1, Q_2, ...Q_n, 0)$$

4.3 BIP-32 Extension

This section describes our approach for a group of members to derive child/receive addresses from the group public key. And, update their private key shares appropriately to spend the digital assets held by the child/receive addresses. For the BIP-32 extension we assume, the group has already been set up, the group public key is computed and is known to everyone, and the shares of the group private key are distributed among the group members securely.

This approach mimics the BIP-32 [12] approach to deterministically generate child addresses. To derive group keys, we pre-define a base derivation path as follows:

```
b = m / 67' / 77'
```

Child keys can be derived at any path below the base derivation path as follows:

1. Choose a derivation path d after the base derivation path b to deterministically derive the group public key.

2. Use the BIP-39 [1] approach to create a BIP-39 seed using the group public key hash (256 bits) as the random entropy.

3. Using the BIP-39 seed, compute the child private key at the chosen derivation path. We represent this child private key as r_d.

4. Perform EC addition to the group public key with $r_d.G$, resulting in the the child group public key.

5. To update the group private key shares, individual parties can compute the r_d value using the derivation path and add it to their shares to get the shares of the group child private key.

6. Note, in case the shares of group private key are encrypted using the linearly homomorphic encryption scheme, parties can encrypt r_d value using the same encryption key and perform homomorphic addition of two encrypted values to compute the encrypted share of group child private key.

5 Our Solution

Before we begin to describe our solution to the problem of devising an approach for a group of members to own digital assets together with a dynamic threshold setting to spend those assets defined clearly in an easily manageable policy, we define the following:

5.1 Definitions

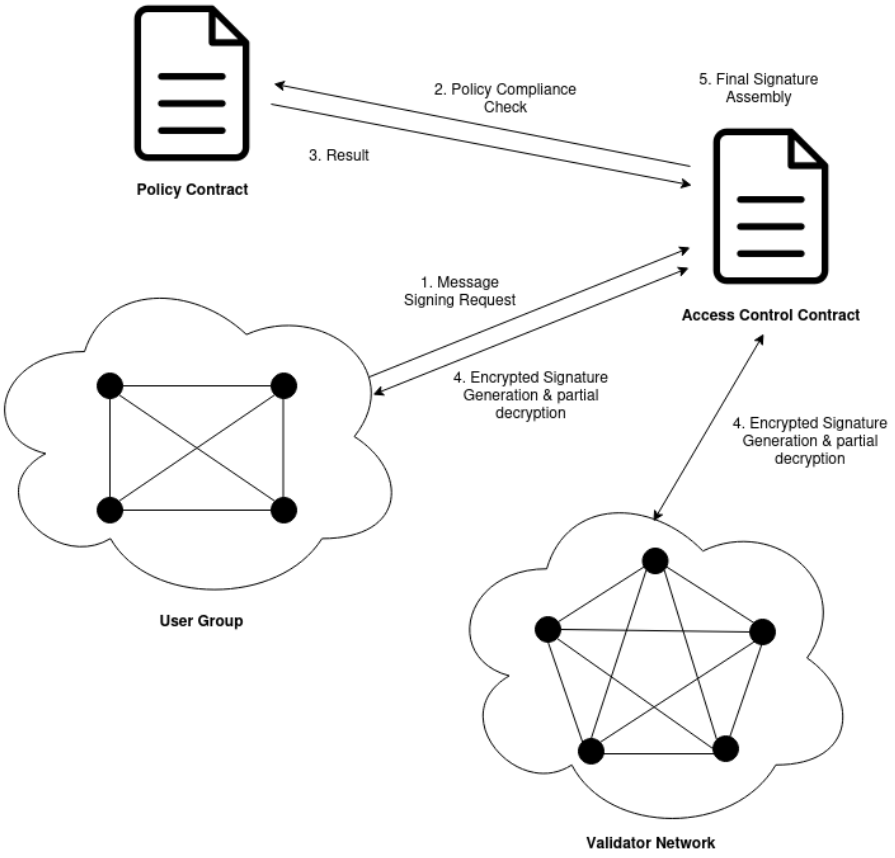

Figure 1: Flow Diagram

1. **User Group** - This term refers to a collective of individuals who jointly possess cryptocurrency assets. The access to these assets by members of the group is governed by an articulated policy managed by the group. The total number of group members is represented as n_p.

2. **Policy** - This is a framework of regulations delineating the required number of group members who must be present to generate the signature on each possible message. We assume T to be an array encapsulating various threshold values corresponding to different messages, as specified in the policy.

3. **Policy Contract** - A blockchain-based smart contract [8] created by the User Group that implements a function that assesses whether a given message can be signed by the given threshold number of group members, returning a boolean value indicative of compliance with established norms.

4. **Validator** - An entity responsible for ensuring policy adherence. The i-th validator is denoted as V_i.

5. **Validator Network** - An interconnected system comprising multiple validators. The validator network operates on predetermined constant parameters enforced by the access control contract defined below, such as $r_v = t_v/n_v$. This represents the ratio of the threshold number of validators required to validate the policy compliance to the total number of validators currently online in the network. In response to the expansion of the validator network, the requisite number of validators for policy validation will proportionally increase.

6. **Access Control Contract** - A pre-formulated smart contract that enforces the compliance of policies utilizing the validator network. This contract is established by the blockchain network and calls the policy contract set up by the group to check for policy compliance.

5.2 Validator Network Setup

1. Initial seed validators will establish the validator network and partake in the Distributed Key Generation (DKG) process to compute the shares of the validator network's private decryption key, denoted as sk_v.

2. After the DKG process, each validator, identified as the i-th validator, will be assigned a share, $S_{sk_v}(i)$, and the network collectively possesses the validator network's public encryption key, pk_v.

3. The DKG process is performed such that any t_v subset of validators can interpolate their shares to derive the validator network decryption key. However, this interpolation will never be performed, instead partial decryptions will be computed using the shares and then aggregated together to get the final decrypted value. This is because of the decentralized nature of the validator network.

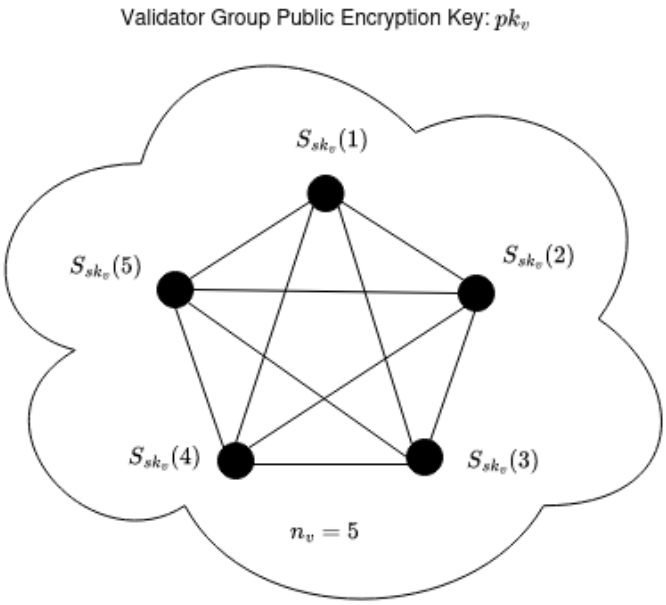

Figure 2: Validator Network Post-Setup

4. The public encryption key of the validator network, generated by the chosen Threshold Homomorphic Encryption Scheme, is publicly accessible and can be retrieved from the blockchain.

5. Figure 2 shows the state of the validator network after the DKG process.

5.3 User Group Setup

1. Initially, the User Group delineates their policy and proceeds to program the policy contract accordingly

2. Subsequently, all group members participate in the Distributed Key Generation (DKG) process such that each member gets a share for each of the $len(T)$ User Group's private tAHE decryption keys.

3. We require $len(T)$ number of shares possessed by each party such that any $T[i]$ number of group members should be able to interpolate their i-th shares to get the i-th decryption key.

4. Each participant in the group, denoted as the j-th member, get the following shares:

$$S_{sk_g^i}(j) \forall i \in T$$

5. Upon the conclusion of the DKG process, $len(T)$ public encryption keys for the User Group are generated, symbolized as $pk_g^i \forall i \in T$.

6. These public keys are then uploaded to the Access Control Contract, making them available for public access.

7. Further, each of the $len(T)$ public keys is multiplied with the public encryption key of the validator network (pk_v) to derive the final set of $len(T)$ global encryption keys:

$$ek_i = pk_g^i \times pk_v \forall i \in T$$

Figure 3 shows the state of the User Group after setting up the User Group encryption keys and the shares of User Group decryption keys.

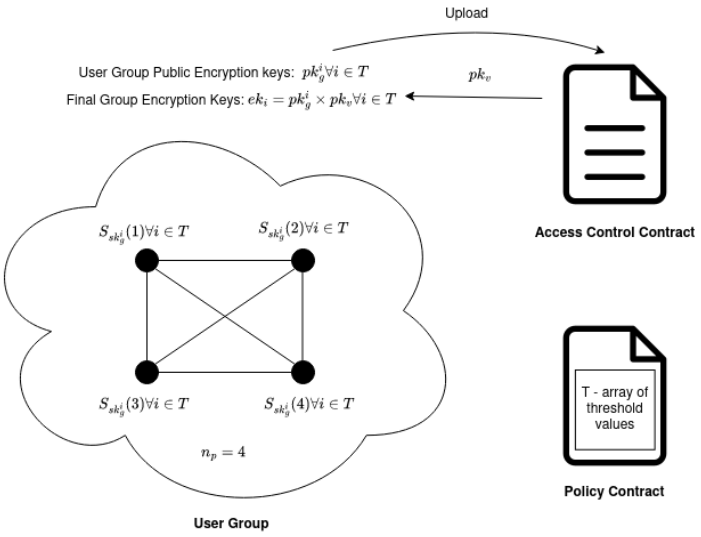

Figure 3: User Group Setup - Part 1

8. The User Group then engages in the encrypted DKG procedure, detailed as follows:

   (a) Each group member generates a unique random number, for instance, the i-th participant creates a_i within the curve order of `SECP256K1`.

   (b) Members then encrypt their random numbers using all the global encryption keys. The calculation done by the i-th members is shows as below:

$$enc(a_i, ek_j) \forall j \in T$$

   .

(c) Post-encryption, these values are uploaded to the contract along with the individual additive share of the group public key computed by the i-th member as follows:

$$Q_i = a_i \cdot G$$

(d) Members also upload zero knowledge proofs to prove the correct usage of encryption function and that the random number used is consistent in the encrypted values and the curve point uploaded to the contract.

(e) Upon receiving all the values from all the group members, the contract verifies the zero knowledge proofs and then proceed to generate group public key and encrypted group private keys using the additive property of tAHE scheme used, as follows:

$$Q = \sum_{i=1}^{n_p} Q_i$$

$$enc(X, ek_i) = \sum_{j=1}^{n_p} enc(a_j, ek_i) \forall i \in T$$

(f) Our BIP-32 extension approach can be used to derive child/receive addresses from this group key, and encrypted shares can be updated accordingly to spend the digital assets held by child/receive addresses.

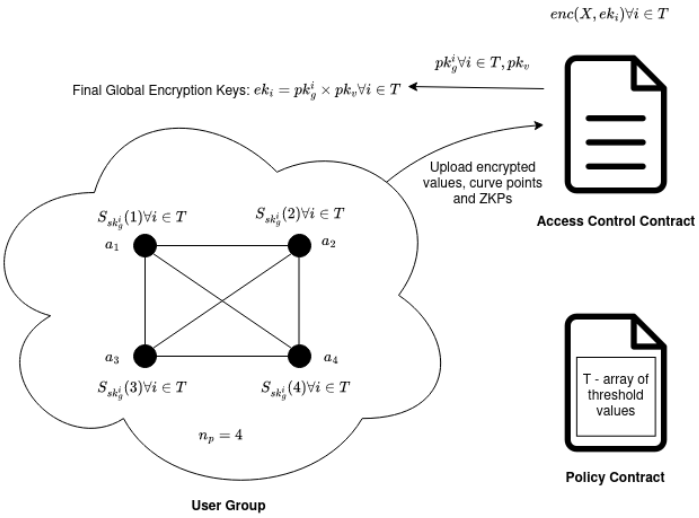

Figure 4: User Group Setup - Part 2

9. At this point, each party possesses their len(T) number of shares of different decryption keys with different threshold values. Each party also possess an additive share a_i of the group private key which is kept as a backup in case the validator network goes down. The contract contains the group private key encrypted with all global encryption keys. See Figure 4.

5.4 Signature Generation Process

1. Consider a scenario where t_p members of the group intend to generate a signature on a message M, which complies with the policy. Specifically, the policy permits the message M to be signed by t_p users, with t_p being an element of the set T.

2. These t_p participants independently generate two random numbers within the curve order of `SECP256K1`. For instance, the i-th participant produces two values, denoted as k_i^g and p_i^g.

3. They also fetch the value $enc(X, ek_{t_p})$ from the contract. Check Figure 5 to see the distribution of the decryption key corresponding to ek_{t_p} among the User Group and the validator network.

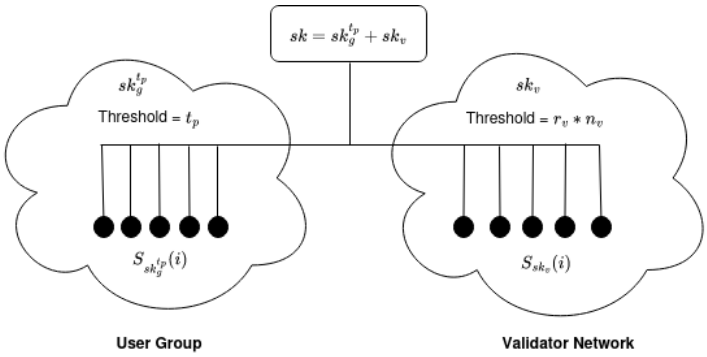

Figure 5: Distribution of Decryption Key

4. Subsequently, the i-th participant encrypts the values k_i^g and p_i^g using the global encryption key ek_{t_p}, resulting in $enc(k_i^g, ek_{t_p})$ and $enc(p_i^g, ek_{t_p})$. The i-th participant also computes $enc(p_i^g * X, ek_{t_p}) = p_i^g * enc(X, ek_{t_p})$ using the scalar multiplication propert of tAHE scheme.

5. The participants also compute zero knowledge proofs to prove the correct usage of encryption function and the scalar multiplication property.

6. All t_p users then interact with the Access Control Contract to initiate the signature generation phase. The i-th user invokes the contract with the following parameters: t_p, M, $enc(k_i^g, ek_{t_p})$, $enc(p_i^g, ek_{t_p})$, $enc(p_i^g * X, ek_{t_p})$, $K_i^g = k_i^g.G$ (where G is the generator point of `SECP256K1`) and the ZKPs.

7. The Access Control Contract inquires with the Policy Contract, established by the group, to validate whether the message M is eligible to be signed by t_p parties.

8. Depending on the verdict of the Policy Contract and the verification of the ZKPs, the Access Control Contract either terminates the process or progresses to the subsequent operation.

9. In the next phase, the Access Control Contract awaits the involvement of $r_v \times n_v$ validators in the signature generation process.

10. Before their participation, these $r_v * n_v$ number of validators retrieve the appropriate User Group encryption key for the threshold value t_p. This key is then combined with the public encryption key of the validator network, pk_v, to formulate ek_{t_p}, as shown:

$$ek_{t_p} = pk_g^{t_p} \times pk_v$$

11. Each validator, denoted as the i-th validator, randomly generates two numbers k_i^v and p_i^v within the curve order of `SECP256K1`.

12. The validators then collectively engage with the Access Control Contract to partake in the signature generation. The i-th validator supplies the contract with M, $enc(k_i^v, ek_{t_p})$, $enc(p_i^v, ek_{t_p})$, $enc(p_i^v * X, ek_{t_p})$, $K_i^v = k_i^v.G$ (where G is the generator point of `SECP256K1`) and the required ZKPs.

13. Upon receipt of function calls from all $r_v \times n_v$ validators, the contract verifies the ZKPs and employs the homomorphic encryption property to aggregate the encrypted values:

$$enc(k, ek_{t_p}) = \sum_{i=1}^{t_p} enc(k_i^g, ek_{t_p}) + \sum_{i=1}^{r_v \times n_v} enc(k_i^v, ek_{t_p})$$

.

14. Concurrently, the contract calculates the R value, integral to the signature, as follows:

$$R = \sum_{i=1}^{t_p} K_i^g + \sum_{i=1}^{r_v \times n_v} K_i^v$$

15. The r component of the signature is derived as the x-coordinate of the elliptic curve point R:

$$r = R.x$$

Figure 6 shows the visualization of Round 1.

16. Each of the t_p number of participants, represented by the i-th participant, retrieves the encrypted value $enc(k, ek_{t_p})$ from the contract and multiplies it with its own random p_i^g value generated before. This computation,

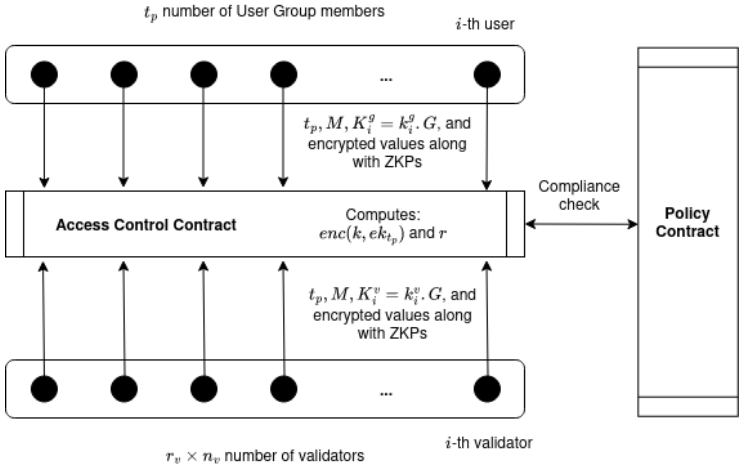

Figure 6: Signature Generation - Round 1

leveraging the scalar multiplication feature of the homomorphic encryption scheme, is executed as follows:

$$enc(p_i^g \times k, ek_{t_p}) = p_i^g \times enc(k, ek_{t_p})$$

17. In a similar fashion, each of the $r_v \times n_v$ validators also acquires the $enc(k, ek_{t_p})$ value from the contract. Utilizing their respective random value p_i^v, the i-th validator computes the following, using the scalar multiplication property of the homomorphic encryption scheme:

$$enc(p_i^v \times k, ek_{t_p}) = p_i^v \times enc(k, ek_{t_p})$$

18. The group participants and the validators also compute ZKPs to prove the correct usage of the scalar multiplication property of the homomorphic encryption scheme.

19. Following these calculations, both the t_p participants and the $r_v \times n_v$ validators transmit their respective encrypted computations and the ZKPs to the contract which first verifies the ZKPs and then processes the inputs collectively to form the encrypted product of the aggregate group secrets p and k, where $p = \sum_{i=1}^{t_p} p_i^g + \sum_{i=1}^{r_v \times n_v} p_i^v$ and $k = \sum_{i=1}^{t_p} k_i^g + \sum_{i=1}^{r_v \times n_v} k_i^v$:

$$enc(p \times k, ek_{t_p}) = \sum_{i=1}^{t_p} enc(p_i^g \times k, ek_{t_p}) + \sum_{i=1}^{r_v \times n_v} enc(p_i^v \times k, ek_{t_p})$$

Figure 7 shows the visualization of Round 2.

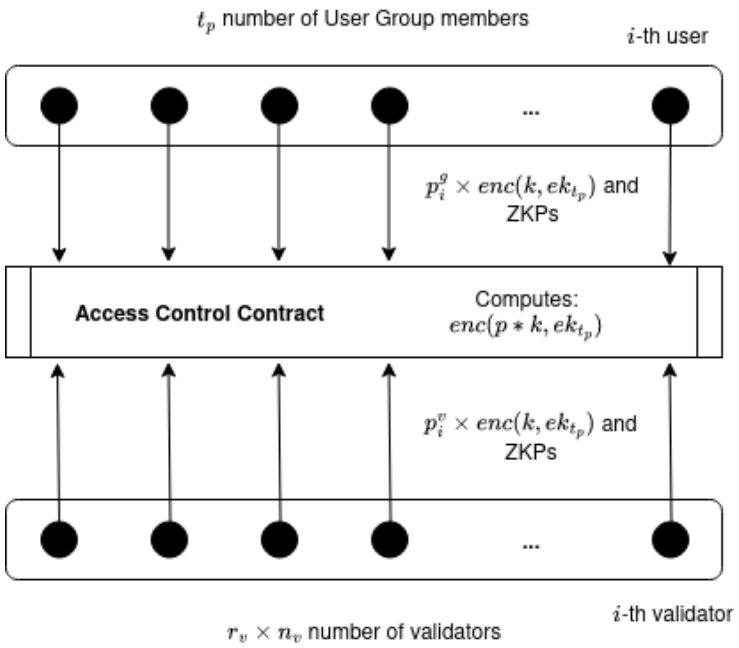

Figure 7: Signature Generation - Round 2

20. Once the contract aggregates the encrypted product, each member of the User Group, denoted as the i-th participant, retrieves this data. They then perform a partial decryption using their specific share of the User Group decryption key, computed as follows:

$$d_i^g = partDec(enc(p \times k, ek_{t_p}), \lambda_i, S_{sk_g{}^{t_p}}(i))$$

21. Concurrently, the validators, each represented as the i-th validator, undertake their partial decryption tasks. This involves applying their share of the validator decryption key to the encrypted product:

$$d_i^v = partDec(enc(p \times k, ek_{t_p}), \lambda_i, S_{sk_v}(i))$$

22. These partial decryption outputs are then propagated back to the contract along with the ZKPs to prove the correct usage of partial decryption functions, where they are collectively synthesized to formulate the final decrypted value, denoted as $w = k \times p$ after ZKP verification:

$$w = aggPartDecs(enc(p \times k, ek_{t_p}),$$
$$\{d_i^g \mid i = 1, \ldots, t_p\} \cup \{d_i^v \mid i = 1, \ldots, r_v \times n_v\})$$

Figure 8 shows the visualization of Round 3.

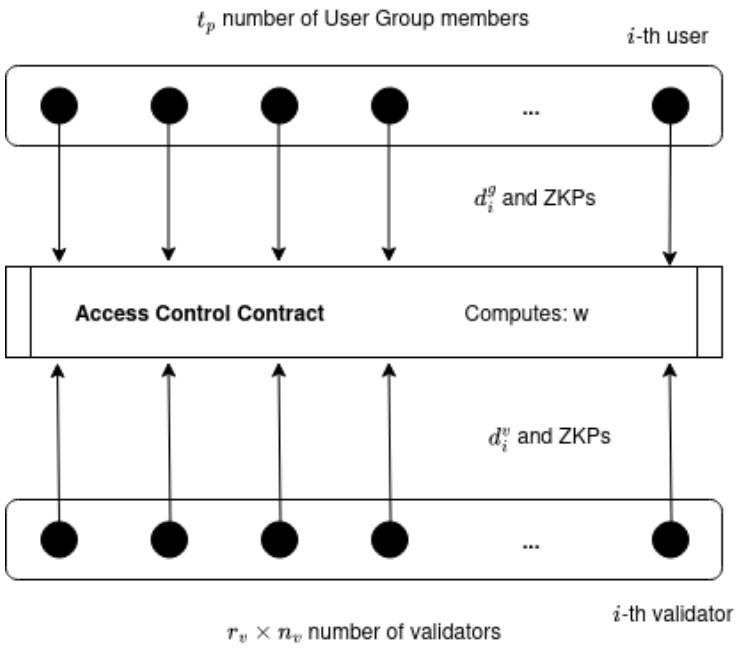

Figure 8: Signature Generation - Round 3

23. The contract the computes the encrypted signature from all the raw materials using additive property of the homomorphic scheme as follows:

$$enc(s, ek_{t_p}) = enc(s_g, ek_{t_p}) + enc(s_v, ek_{t_p}), where$$

$enc(s_g, ek_{t_p})$ is the encrypted signature share of the User Group and $enc(s_v, ek_{t_p})$ is the encrypted signature share of the Validator Network computed using the additive and scalar multiplication properties of the homomorphic encryption scheme, as follows:

$$enc(s_g, ek_{t_p}) = \sum_{i=1}^{t_p}[(w^{-1} \times enc(p_i^g, ek_{t_p}) \times \text{Hash}(M)) + (r \times w^{-1} \times enc(p_i^g * X, ek_{t_p}))]$$

$$enc(s_v, ek_{t_p}) = \sum_{i=1}^{r_v \times n_v}[(w^{-1} \times enc(p_i^v, ek_{t_p}) \times \text{Hash}(M)) + (r \times w^{-1} \times enc(p_i^v * X, ek_{t_p}))]$$

since the contract already has the values, w, $enc(p_i^g, ek_{t_p})$, $enc(p_i^v, ek_{t_p})$, M, r, $enc(p_i^g * X, ek_{t_p})$ and $enc(p_i^v * X, ek_{t_p})$.

24. Each User Group participant, denoted as the i-th participant, retrieves the encrypted final signature. They then perform a partial decryption using their allocated share of the decryption key and broadcast this to

the contract along with the ZKPs to prove the correct usage of partial decryption function:

$$s_i^g = partDec(enc(s, ek_{t_p}), \lambda_i, S_{sk_g{}^{t_p}}(i))$$

25. In parallel, each validator, denoted as the i-th validator, processes their partial decrypted signature by applying their share of the validator decryption key to the encrypted final signature, before broadcasting it to the contract along with ZKPs to prove the correct usage of partial decryption function:

$$s_i^v = partDec(enc(s, ek_{t_p}), \lambda_i, S_{sk_v}(i))$$

26. The contract then aggregates these inputs to compute the complete final signature after verifying all the ZKPs:

$$s = aggPartDecs(enc(s, ek_{t_p}),$$

$$\{s_i^g \mid i = 1, \ldots, t_p\} \cup \{s_i^v \mid i = 1, \ldots, r_v \times n_v\})$$

Figure 9 shows the visualization of the final round.

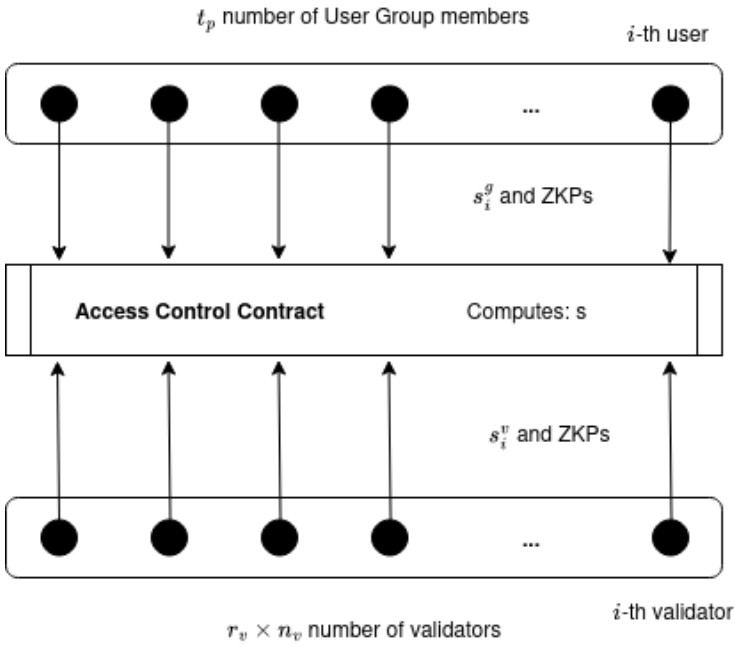

Figure 9: Signature Generation - Round 4

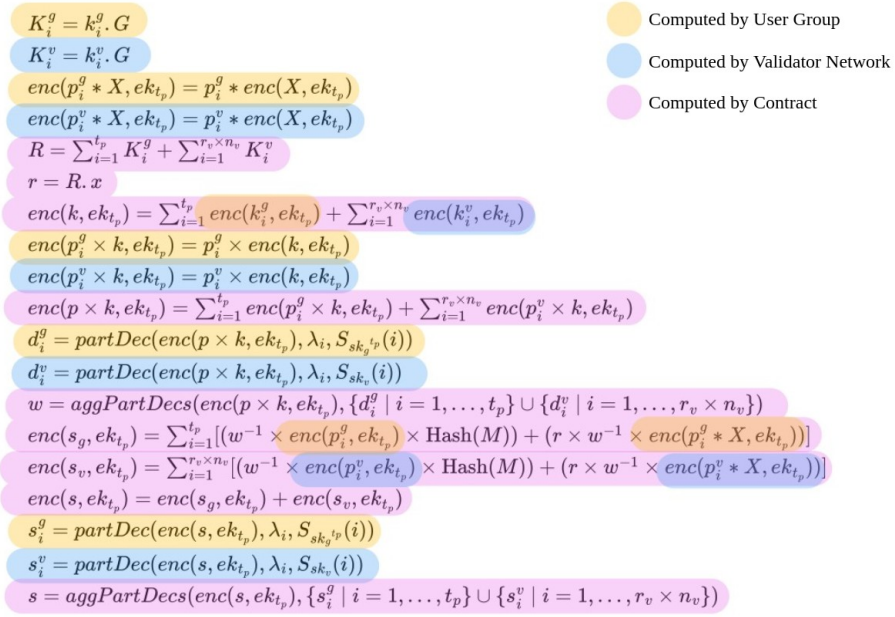

Figure 10: Signature Generation Process Summary

27. Finally, all t_p number of participants of the User Group retrieve the consolidated final signature. This signature is then utilized to authorize the expenditure of the digital assets held collectively by the group.

Figure 10 shows the overall ECDSA circuit computed at the contract with contributions from the User Group and the Validator Network. As can be seen from the equations, the randomness used for generating the ephemeral key k is contributed by both the User Group and the Validator Network. This is architectured in this way to prevent a malicious set of group members from generating the signature with their own ephemeral key and hence leak the group private key.

5.5 Validator Network Outage Scenario

The above scheme supports the availability of the digital assets held by the User Group even in the case when the whole blockchain network where the contracts are hosted is down or even when there are no $r_v \times n_v$ number of validators available to validate the policy compliance. But to transfer the digital assets in such a scenario, all the members of the User Group have to come online to re-generate the shares of the group private key using the random a_i values held by the parties. The parties, after performing the DKG (Distributed Key Generation) process again using the same random values as the coefficient of

x^0 of their individual polynomial, can obtain the non-encrypted shares of the group private key with the threshold value being n_p. Using these shares, they can compute signatures over any arbitrary message and transfer the digital assets.

6 Security Analysis

We now describe the scenarios where the malicious actors can try to sign a message that does not comply with the policy defined, and how our system prevents the signature from being generated in such cases. Assuming t (where $t < n_p$) malicious User Group members want to sign the message M' that does not comply with policy and there's a message M that does comply with the policy. The following scenarios are possible:

1. The malicious User Group members can call the Access Control Contract with the message M' but the Access Control contract will halt before calling out the validators from the network upon confirming that M' doesn't comply with the rules set up by the group, with the Policy Contract. And since all users are required to make updates to the Policy Contract, the t number of malicious users will not be able to generate any signature at all.

2. The malicious User Group members can call the Access Control Contract with the message M and it'll pass the policy compliance check by the contract. However, the signature shares generated by the validators will be on the message M and not M'. Moreover, the malicious users only submit their u_i and v_i to the contract and the message is added by the contract to compute the encrypted signature share of the User Group. This mechanism allows for secure enforcement of the policy.

3. The malicious User Group members may try to generate the complete encrypted signature by themselves since they possess the encrypted private key shares with threshold value t. But they will not be able to decrypt the signature to spend the digital assets, because some of the shares of the decryption key are held by the validator network.

4. Once the signature of the message M is successfully generated, the malicious User Group members may try to figure out the random nonce (k) value used in the signature computation, to get the decrypted group private key to generate further non-compliant signatures. However, they can not compute the value of k as it was not just distributed between the User Group but also the Validator Network. The malicious members are thus not able to guess the shares used by the validators while contributing to the group secret value k.

Although the approach we described is secure against policy compliance breaks, we acknowledge that we do need Zero Knowledge Proofs (ZKPs) at

various steps in our solution to achieve the property of identifiable abort. This is a required property in practical scenarios as the network should be able to figure out which dishonest validator node or group member is trying to sabotage the signature generation process.

7 What's possible with our solution

The innovative blockchain infrastructure proposed in this whitepaper has far-reaching implications for the management of digital assets, addressing the needs of institutions and collectives in a secure, decentralized, and flexible manner. The implementation of this infrastructure enables a variety of applications, catering to specific policy requirements and enhancing operational efficiencies. Below are key features provided by this solution:

7.1 Address Whitelisting

The policy mechanism in this solution allows for the creation of "whitelists" of addresses, ensuring that transactions are only signed and executed if they are destined for these pre-approved addresses. This reduces the risk of unauthorized transactions and enhances the security of asset management.

7.2 Spending Limits

Institutional users can set spending limits within the policy, defining the maximum amount of digital assets that can be transacted in a single operation or over a specified time. Additionally, the policy can dictate the number of members required to authorize transactions exceeding these limits.

7.3 Time-Based Transactions

The solution enables the configuration of policies for time-based transactions, allowing transactions to be executed only during specified time windows. This feature is useful for executing scheduled payments, limiting transactions to business hours, or aligning transactions with specific events or milestones.

7.4 Programmable Rules

One of the most powerful aspects of this solution is its ability to support arbitrary programmable rules within the policy framework. This flexibility allows institutions to encode complex, custom rules that govern how and when transactions are executed. These rules can be tailored to specific organizational needs, regulatory requirements, or risk management strategies, providing unparalleled customization in digital asset management.

7.5 DAO Access Control

This infrastructure is ideal for decentralized autonomous organizations (DAOs), as it aligns with their democratic and decentralized principles. DAOs can use this system to democratically control and update policies, ensuring asset management aligns with the collective will of their members. This also includes robust access control mechanisms, allowing DAOs to define who can manage their assets, under what conditions, and with specific limitations. This functionality is essential for DAOs with significant assets or those needing to adhere to strict regulatory standards.

8 Conclusion

In conclusion, the development of the Dynamic Threshold Digital Signing Scheme for Blockchain marks a significant leap forward in the field of blockchain technology and digital asset management. This scheme not only introduces a novel approach to enforce access control policy settings but also addresses key challenges in policy management, decentralization, network resilience, efficiency, and security.

Through our unique group access control framework supporting dynamic thresholds, we have achieved a flexible yet secure mechanism for managing digital assets. The decentralized nature of the validation network underpins the robustness of the system, eliminating single points of failure and ensuring consistent operation even when parts of the network are down. The scheme further supports the protection of funds even when the validation network becomes unavailable.

This approach, along with the fact that the keys to sign transactions remain within the group, reinforces the security and autonomy of user groups. The ease of deriving child/receive addresses for groups and the capability to support the joining and leaving of validators demonstrate the system's flexibility and scalability. Moreover, the scheme's compatibility with all chains ensures its wide applicability, striking a balance between usefulness and security. This balance is achieved under specific security assumptions, notably the Decisional Diffie-Hellman (DDH) assumption and the Hard Subgroup Membership (HSM) with 2k security.

Looking to the future, the architecture of this scheme lays the foundation for a myriad of applications, including blockchain bridges, swaps, private transactions, and private data processing. These potential applications highlight the versatility and broad impact of the scheme, opening new avenues for secure and efficient blockchain interactions.

9 Future Work

Future works includes further optimisations to practically implement the scheme for real world use cases, and developing a proof of concept to gather performance benchmarks before implementing the scheme using Cypherock X1 devices. As we explore further, further improvements will be added to this paper in the revised editions.

References

[1] *BIP39*. https://github.com/bitcoin/bips/blob/master/bip-0039.mediawiki. Accessed: March 1, 2024.

[2] Cyril Bouvier et al. *I want to ride my BICYCL: BICYCL Implements CryptographY in CLass groups*. Cryptology ePrint Archive, Paper 2022/1466. https://eprint.iacr.org/2022/1466. 2022. DOI: 10.1007/s00145-023-09459-1. URL: https://eprint.iacr.org/2022/1466.

[3] Guilhem Castagnos, Fabien Laguillaumie, and Ida Tucker. *Threshold Linearly Homomorphic Encryption on* $\mathbf{Z}/2^k\mathbf{Z}$. Cryptology ePrint Archive, Paper 2022/1143. https://eprint.iacr.org/2022/1143. 2022. URL: https://eprint.iacr.org/2022/1143.

[4] Cypherock. *TSS in Secure Environment*. https://github.com/Cypherock/MPC-TSS/blob/mascot-iknp-mta/MPC_in_Secure_Environment.pdf. Accessed: March 1, 2024. 2023. URL: https://github.com/Cypherock/MPC-TSS/blob/mascot-iknp-mta/MPC_in_Secure_Environment.pdf.

[5] Ivan Damgård et al. *Fast Threshold ECDSA with Honest Majority*. Cryptology ePrint Archive, Paper 2020/501. https://eprint.iacr.org/2020/501. 2020. URL: https://eprint.iacr.org/2020/501.

[6] Jack Doerner et al. *Threshold ECDSA from ECDSA Assumptions: The Multiparty Case*. Cryptology ePrint Archive, Paper 2019/523. https://eprint.iacr.org/2019/523. 2019. DOI: 10.1109/SP.2019.00024. URL: https://eprint.iacr.org/2019/523.

[7] Jack Doerner et al. *Threshold ECDSA in Three Rounds*. Cryptology ePrint Archive, Paper 2023/765. https://eprint.iacr.org/2023/765. 2023. URL: https://eprint.iacr.org/2023/765.

[8] Ethereum. *Ethereum Whitepaper*. https://ethereum.org/en/whitepaper. Accessed: March 1, 2024. 2023. URL: https://ethereum.org/en/whitepaper.

[9] Rosario Gennaro and Steven Goldfeder. *Fast Multiparty Threshold ECDSA with Fast Trustless Setup*. Cryptology ePrint Archive, Paper 2019/114. https://eprint.iacr.org/2019/114. 2019. URL: https://eprint.iacr.org/2019/114.

[10] Iftach Haitner et al. *Fast Secure Multiparty ECDSA with Practical Distributed Key Generation and Applications to Cryptocurrency Custody*. Cryptology ePrint Archive, Paper 2018/987. https://eprint.iacr.org/2018/987. 2018. DOI: 10.1145/3243734.3243788. URL: https://eprint.iacr.org/2018/987.

[11] Odsy. *Odsy Litepaper*. https://github.com/odsy-network/odsy-litepaper/blob/main/odsy-litepaper-version-1-0-august-2022.pdf. 2022. URL: https://github.com/odsy-network/odsy-litepaper/blob/main/odsy-litepaper-version-1-0-august-2022.pdf.

[12] *Public CKD (Hierarchical Deterministic Wallets)*. https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki#public-parent-key--public-child-key. Accessed: March 1, 2024.