Common Interest Protocol - dCommon

Stakehouse CIP Application

Bingsheng Zhang, Justinas Žaliaduonis, Matt Shams(Anis)

Blockswap Labs

Abstract

In this paper we introduce dCommon - auditable and programmable MPC as a service for solving multichain governance coordination problems throughout DeFi and Web3; Along with its on-chain part Common Interest Protocol (CIP) - an autonomous and immutable registry smart contract suite. CIP enables arbitrary business logic for off-chain computations using dCommon's network/subnetworks with Ethereum smart contracts. In Stakehouse, CIP facilitates a trustless recovery of signing keys and key management for validator owners on demand. The paper elucidates a formal overview of the MPC system cryptography mechanics and its smart contract business logic for the Stakehouse CIP (SH-CIP) application implementation.

1 Introduction & Rationale

Ethereum node operations are an off-chain activity performed by validator node operators for Ethereum consensus; They are controlled through on-chain deposits of ETH. Stakehouse is an extended business logic of the consensus protocol [13] [8] of Ethereum. It is implemented on the execution layer using solidity smart contracts where it tokenizes the validator deposits into two distinct token balances (dETH and SLOT). dETH is the base capital that accrues all consensus rewards from validation, SLOT represents the off-chain operators and has exclusivity on revenue rights from transaction building. Ethereum network has two sets of distinct revenue flows for its staked deposits. One is pure inflation yield from newly minted ETH for keeping the blockchain live and secure, and the second is for injecting the transactions into the block. This is performed by off-chain actors using an RPC-endpoint for transaction aggregation and using Ethereum client software for ordering and building blocks as per Ethereum builder specification. Blockspace and fees are determined by a dynamic transaction fee market protocol EIP 1559 [17] [10] [6].

Ethereum validator balances are considered as stake weight in consensus validation. They are represented by a validator's cryptographic address (as opposed to an IP address) for consensus scheme and block building, which means every block in Ethereum has a disclosure attached to it, in the form of validator addresses who had attested and proposed. Further, the network keeps tracking this disclosure history for dispensing rewards for validators who are active in the validation process, penalizing those who are inactive, and slashing those who are malicious based on a whistleblower report.

Ethereum stake weight has a range-bound maximum of 32 ETH to be eligible as a validator in the network and a minimum of 16 ETH to be included for validation duties; please note that the active balance of a validator can be more or less than this stake weight. However, if a validator falls below the minimum stake weight, it will be kicked

from the consensus validation process. A leakage mechanism consistently penalizes inactive validators and eventually kicks them out when the balance falls below the minimum stake weight.

1.1 Stakehouse and Ethereum Consensus

The rewards from consensus validation will go to dETH holders in Stakehouse, where SLOT token holders will bear penalty and slashing. In return, SLOT tokens have the exclusive right to transaction fee revenue earned according to the EIP-1559 protocol and from their custom RPC transaction ordering logic, normally referred to as MEV (maximal extraction value) for every block added to the chain.

In Stakehouse protocol, a critical measure is to keep all its registered validators active and in optimal performance for Ethereum. Hence it re-configures the stake weight representation on its business logic to a minimum of 24 and a maximum of 32. Thus it is reflected as 24 dETH and 8 SLOT tokens per validator.

In the Stakehouse protocol, SLOT tokens are managed by a standalone SLOT Registry [2] [5]. The registry records every change in token balances at the validator level. Validators are grouped as an arbitrary collection of validators in the registry (a.k.a, Stakehouse) and accounted on a native token, sETH, that serves as a unit of account for SLOT balance movement in the house. Every 1 ETH decrease in the stake weight of validator balance is registered in the Stakehouse protocol as a 1 SLOT token decrease. Anyone can top-up ETH for the decreased amount and get newly issued SLOT tokens from the Stakehouse protocol. There is also a specific logic on how SLOT tokens are classified in Stakehouse; 8 SLOT per validator as four free floating and, four collateralized (ref. Stakehouse position paper [5]). If a validator within Stakehouse is slashed then the KNOT (validator) will be removed from the protocol's registry. For both leakages and slashing, it results in the SLOT being burned from the collateralized SLOT vault.

Ethereum staking requires and incentivizes the validator's liveness to be active all the time. Being a multi-client network, the network coordination cannot be checked by a single tracker. So with leakage for inactive validators and sudden kicking of validators from the network after a slashing, it is imperative that the validators shall have an on-demand Signing Key retrieval for managing the off-chain node (a device that is connected to the Ethereum network running a client software).

Ethereum PoS design is focused on onboarding long tail users as validators and it allows anyone with 32 ETH to become a validator and a node in the network, this is a very brave attempt for decentralizing the PoS network thus increasing its resilience and longevity. Stakehouse protocol takes this mission a step further and allows mainstream users to have fractional ownership of a validator on SLOT tokens, and it continuously incentivizes its registered validator's performance using Price of Anarchy principles [16], first by applying a decay on existing house payoff rate and then by allowing permissionless topup of SLOT tokens and collect transaction revenue of that validator.

1.2 Ethereum Staking Keys

Every Ethereum Validator registered in the Consensus layer maintains three distinct cryptographic addresses [7] (keys):

- Public Key - BLS [3].
- Validator Private Key - a.k.a. Signing Key - BLS

- Withdrawal Credential Key (BLS or Ethereum ECDSA [8]), where in Stakehouse protocol withdrawal credential is assigned to its Account Manager contract serving the KNOT token addresses - dETH and SLOT owners.

The signing key is an operational key that is required to perform off-chain node operations of Ethereum staking, it has no right to receive either validation consensus rewards or the underlying deposit withdrawal.

The Stakehouse protocol's immutable and permissionless nature demands trustless and autonomous coordination for both onchain and off-chain elements of Ethereum staking since the validator ownership is fractionalized and openly traded in the market. Every validator in the stakehouse and its value generation is actively priced by market actors, hence any small downtime of validators may reflect in market reflexivity on house payoff rates on transaction revenue earning, consequently, the dETH yield rates may vary negatively.

The relationship between Stakehouse protocol and Ethereum keys is displayed in the figure below (Fig. 1).

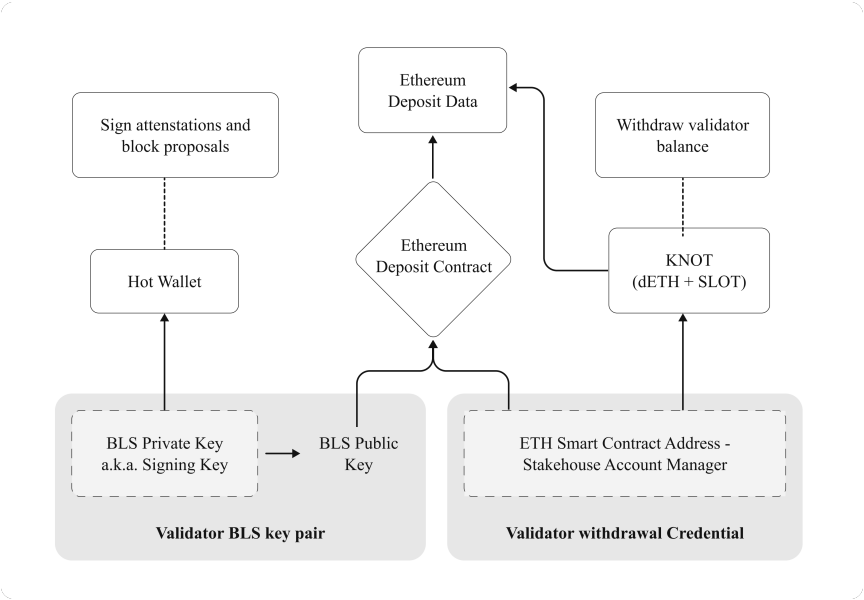

Figure 1: Ethereum Staking Keys & Stakehouse Protocol

2 Stakehouse - CIP

Stakehouse CIP application enables stakehouse validators to have an on-demand trustless backup and recovery of their validator signing key [8] that is encrypted and stored on the Ethereum blockchain using dCommon subnet DKG derived master public key.

A core primitive of dCommon is its CIP applications are standalone networks within dCommon, they have complete autonomy to have their own MPC solvers or appoint a solver sub-network registered under dCommon using DKGRegistry and SafeBox contracts, and can freely ragequit via resignation mechanism anytime. For Stakehouse we define a business logic to have Stakehouse protocol users (Ethereum stakers) act as MPC solvers

with its own exclusive MPC network instance serving other stakers. An MPC network serves its common interest for its well-being and continuity.

3 *Key building blocks*

3.1 *Cryptographic Primitives*

Throughout the CIP system, it uses four Probabilistic Polynomial Time Algorithms to derive a hybrid encryption scheme for ensuring operations of secure Multiparty Computation, referred to as Hybrid Encryption.

Formally, the cryptographic primitives can be described by incorporating Probabilistic Polynomial Time - PPT algorithms.

The first algorithm – $Gen_{gp}(1^{\lambda})$ is used for generating the group parameter σ given the security parameter $\lambda \in \mathbb{N}$, using elliptic curve group P-256 (secp256r1) [18].

The other 3 algorithms are from the Hybrid Encryption set HE and are mostly used for the Hybrid Cryptography scheme. It serves three operations for the protocol: **Key Generation**, **Plaintext Encryption**, and **Ciphertext Decryption**.

$HE.Keygen(\sigma)$ - the algorithm is used for generating the hybrid credential pair $(pk, sk) = ((g, h), s)$ and works by picking a private key s from the prime integer group $\mathbb{Z}_q$ and obtaining the public key by raising the generator point g to the power of s: $h = g^s$.

$HE.Enc_{pk}(\sigma, m)$ is used for forming a ciphertext C hiding the message m. The ciphertext is formed by picking a random element $r \in \mathbb{Z}_q$, computing the generator point raised to the power of r, and the public key of the recipient h raised to the power of r: $c_1 := g^r$, $c_2 := h^r$. Then, the joint AES key k is computed by hashing c_2: $k = hash(c_2)$, and the final ciphertext is formed by utilizing the AES encryption: $u := AES-GCM_k(m)$.

$HE.Dec_{sk}(\sigma, m)$ is used to obtain the message m given the ciphertext u and the encryptor hybrid public key c_1. The algorithm works by computing the shared key seed $c_2 := (c_1)^{sk}$ and hashing it to compute the joint AES key $k = hash(c_2)$. Further, the shared key k together with the ciphertext u is fed into the AES decryption function $m = AES - GCM_k(u)^{-1}$.

Notation: For the purpose of formally presenting the operations used in the SH-CIP protocol we will simplify the notation by emitting the group parameter σ.

- $HE.Enc_{pk}(\sigma; m)$ - Instead we will use the notation $HE.encrypt_{pk}(m, sk)$. This should be read as the message m is encrypted for the recipient who has a public key pk while using the secret key sk.
- $HE.Dec_{sk}(\sigma; m)$ - will be denoted as $HE.decrypt_{pk}(m, sk)$. This should be read as decrypting a message m sent by the holder of the public key pk, while using the secret key sk

To have a better understanding on crypto primitives used in the context of Stakehosue CIP we give a briew listing below (Tab. 1). All adopted primitives are widely used in the cryptography field and have been tested in many other applications.

Table 1: Cryptographic primitives used in the Common Interest Protocol

CRYPTOGRAPHIC PRIMITIVE	BUILDING BLOCKS	CIP APPLICATION USAGE
Generic group (Elliptic Curve p256k)	ECDSA address	Ethereum wallet and token addresses
Hybrid KeyGen (AES Keys)	Secure communication using block ciphers	DKG and MPC decryption coordination.
Hybrid Encryption	Symmetric and asymmetric encryption combination	Secure data dissemination for DKG and MPC operations
Hybrid Encryption: encryption	AES based encryptions	Message broadcast in DKG & MPC
Hybrid Encryption: decryption	AES based decryption	Verifiable recipient delivery
Multi-Party computation	Arbitrary custom message computations	Computation of partial authentication of Signing Key decryption
NIZK - proof	Non-interactive deterministic verification	Authentication of an encrypted envelope in DKG & MPC
DKG - Distributed Key Generation	Shared private credential of MPC network	Setting up a trustless adhoc MPC committee
Shamir Secret	Recombine split values into a secret value	Combine received solver solutions and decrypt the signing key

4 Multi-Party Computation (MPC)

Multiparty computation (MPC) is a cryptographic protocol primitive, as the name implies it distributes computation across multiple parties, who do not trust each other or any common third party to jointly compute a function that depends on all of their private inputs. Here, no individual party can see the other parties' data and only disclosing the result of the function to a specific counterparty. The MPC scheme has been around for more than 30 years in academic and theoretical realms; with the advent of blockchains and the proliferation of smart contract applications for real-time applications such as digital asset trading, auction, privacy, etc, it became a core primitive for mainstream usage.

4.1 Stakehouse CIP MPC

Stakehouse CIP application utilizes MPC with Distributed Key Generation (DKG) for registering its MPC solvers as having shared credentials and expanding the set from time to time using a handover mechanism.

Additionally, the dCommon design offers a DKG that can be performed in an ad-hoc network with an asynchronous setting discarding inactive participants from the committee and ensuring robustness for the system. This is a great advantage over traditional systems that operate in synchronized settings, where single-party failure requires the restart of the whole system, thus becoming practically unusable for systems that require coordination at scale.

5 SH-CIP Configuration

The Stakehouse Common Interest Protocol (SH-CIP) application allows a user who holds more than 2 collateralized SLOT tokens in a stakehouse tokenized validator (KNOT) to decrypt the encrypted message (validator BLS signing key) from the Ethereum blockchain. SH-CIP application is designed to serve exclusively Stakehouse protocol Collateralized SLOT token holders facilitating backup and on-demand trustless recovery option for their validators signing keys, however, it remains independent from an operational standpoint from the Stakehouse protocol. *To avoid any doubt, Stakehouse protocol can continue its full operations in absence of SH-CIP services.*

5.1 Role Matrix

Table 2: The matrix describing role relationships in the Stakehouse CIP

ACTIONS	REQUESTER	STAKEHOUSE	SAFEBOX	MPC
Signing Key Backup (Encryption)	YES	YES	NO	NEUTRAL
Decryption Request	YES	NEUTRAL	YES	NO
Decrypiton Solution	NO	NEUTRAL	YES	YES
Decryption Key	YES	NO	NEUTRAL	NO
Joining DKG Solvers	NO	NEUTRAL	YES	NO
Expanding Solver Set	NO	YES	YES	YES
Solver Eligibility	YES	NEUTRAL	YES	NO
Decryptor Eligibility	NO	YES	YES	YES

5.2 Architecture

Stakehouse CIP application leverage dCommon for its off-chain cryptographic operations and CIP smart-contract suite for its on-chain validation and intermediation for signing key decryption.

Table 3: Architectural base module roles

CORE COMPONENTS	CONTEXT	APPLICATION UTILITY
SafeBox Smart Contract	Registry Smart Contract as BulletinBoard	Preserves computation request integrity and availability
Solver off-chain Client S_BOX	Off-chain MPC nodes	Perform cryptographic operations for MPC and DKG
Cryptography Module	Off-chain library or package	Cryptography logic for key generation and computations.
SLOT token	Stakehouse Validator	Tokenized access with identity for decryption request, DKG and MPC Solver

5.2.1 SafeBox Smart Contract

CIP consist of two smart contracts: *SafeBox* and *DKGRegistry*. SafeBox is primarily responsible for orchestrating the entire lifecycle management of the CIP application and dCommon MPC instance. It's responsible for the following:

- Complete information broadcasting as events and coordination of actors
- Implement all checklists for decryption and MPC solver set DKG services
- Enable custom business logic between operations

Decryption Rule: *SafeBox*

- When a user executes the decryption request for a KNOT's signing key by calling *applyForDecryption()*, then
  - the KNOT is active and has not rage-quit
  - the KNOT has no slashed SLOT with the latest balance update
  - the user owns more than 2 collateralized SLOT for the KNOT
  - The user has not submitted any decryption request for the KNOT within the last 6300 blocks.
- When a solver submits information for a decryption request by calling *submitDecryption()*, then

- the solver must be registered with the SafeBox contract
- the decryption request must be less than 6300 blocks old,
- the solver must not already have submitted any information for this specific decryption request
- the solver must not have relinquished his duties

DKG Registration rule: *SafeBox*

The SafeBox contract also imposes the following requirements for DKG committee:

- Initial DKG committee consists only of the ECDSA addresses and Hybrid public keys specified on the SafeBox smart contract deployment procedure. Further expansions can be achieved during the Handover procedure.
- Guardians who have relinquished their duties cannot further participate in the protocol.
- The data sent by the participants must pass various length checks.

5.2.2 DKG Registry

DKG registry is a contract that keeps track of Distribution Generation Process participants' active status. It also Keeps track of initial progress through the DKG procedure, performs submitted data sanity checks, and makes sure the round process is sequential ($i \rightarrow i + 1$).

5.2.3 SlotRegistry - SLOT token

In addition to the above contracts CIP also leverages SlotRegistry from Stakehouse Protocol for querying information related to SLOT tokens of the KNOT(validator) that requested the Signing Key and the MPC solver eligibility.

5.2.4 MPC solver Client - S_BOX

A custom standard software for dCommon MPC solvers to run computation for a puzzle and generate proof satisfying the conditions of CIP onchain SafeBox smart contract for both decryption services and Solver committee DKG process.

5.2.5 Cryptography Module

Cryptography module is an off-chain software utilized by the MPC solvers and the *SafeBox*. The module holds all of the cryptographic operations, credential generation, and other computations, and can be used in isolation or hosted environment.

5.2.6 SLOT token

SLOT token is an ERC20 token issued by Stakehouse protocol from minting a KNOT and against an Etheruem Validator. Collateralized SLOT is the specific token that is accepted by CIP for both decryption requests and registering as an MPC solver.

6 MPC Operation

SH_CIP application process description in pseudo formal spec for Validator signing Key encryption performed by a user and then get decrypted using CIP application can be seen in the figure (Fig. 9).

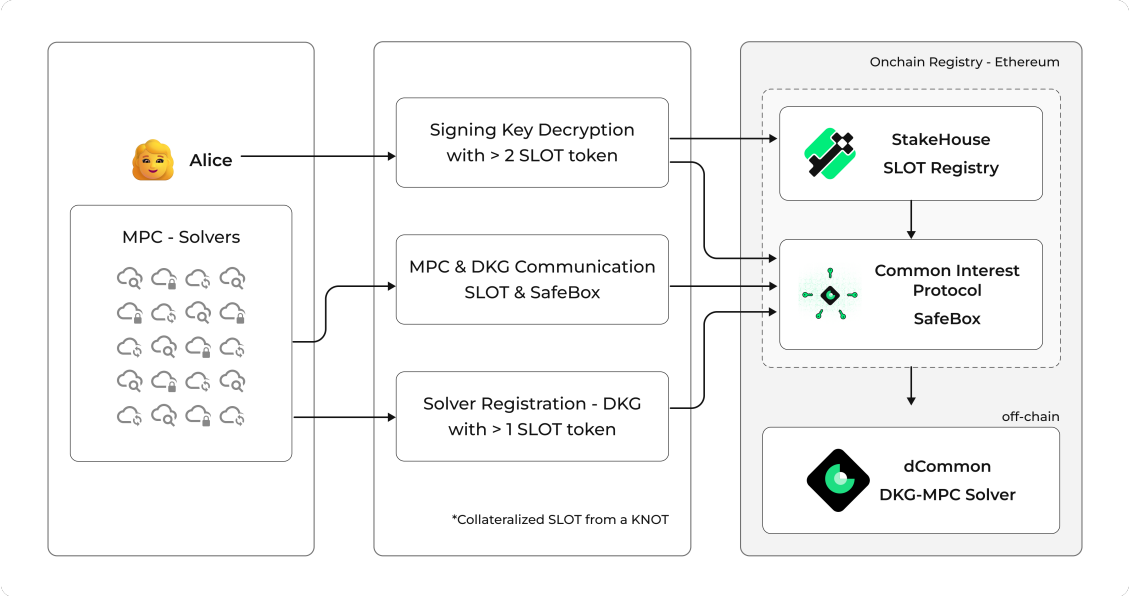

Figure 2: Identity and Responsibility management through registries

7 Core Assumptions

Recovery Requestor: Must hold > 2 Collateralized SLOT tokens
Corllateralized SLOT Token: Non-transferable SLOT held in KNOT Collateralized Vault, deposited by initial validator owner at the time of tokenizing the staked Validator, subject to have ownership change based on validator leakage/slashing top-ups.

We define a mapping $b : (v, a) \to \mathbb{N}_0$ that takes in a BLS public key v and an ECDSA address a and returns a collateralized SLOT token balance owned by the address a for the validator v.

Then, whenever the request for a validator v key recovery is submitted by an address a, the system requires *SafeBox* smart contract to satisfy the following condition:

$$b(v, a) > 2 \tag{1}$$

Recovered validator: Must have no untopped-up penalties reported
Leakage penalties: Stakehouse protocol allows reporting consensus layer accrued penalties and reducing the collateralized SLOT balance of the validator. The balance can also be recovered by topping up the collateralized SLOT balance by sending an equivalent amount of Ethereum to the Ethereum Deposit Contract [11].

Here, the system requires the validator to have no reported penalties i.e. have a full 4 SLOT Collateralized tokens balance. It is checked by defining a mapping $\psi : v \to \mathbb{N}_0$ which takes a validator BLS public key v and gives reported and untopped-up validator penalties in the Stakehouse protocol. The system requires the following equation to hold:

$$\psi(v) = 0 \tag{2}$$

Recovery Requestor: Must submit the recovery request when no active request is present for the requested validator.
Request timing: In order to prevent spam and give sufficient time for solver request

processing, the requests can only be submitted periodically. Consider the decryption validity period $\alpha = 6300$ blocks, the block during which the decryption request is submitted λ and the last request block r, then specified as follows:

$$\lambda > r + \alpha \tag{3}$$

8 SH-CIP MPC operation overview

In this section we will discuss the essential SH-CIP operations based on the algorithms and primitives described in the crypto primitives section.

8.1 Helper functions

Here we define a few helper functions that abstract the logic specified in the Zero-Knowledge proof section for forming and verifying a ZK proof. Additionally, we add a function to compute an EIP712 [15] signature over specified data.

- *formNIZK(encryptor public key ,shared secret key, partial decryption solution)* = Function to form a proof verifying the correctness of partial decryption solution
- *verifyNIZK(shared public key, encryptor public key, partial decryption solution, NIZK)* = Function to verify the NIZK proof for partial decryption solution correctness.
- *eip712Sign([data], destination, deadline)* = Function to form an EIP712-compliant signature over the specified data together with a destination and a deadline.
- *assembleHybridKey([partialDecryptionSolutions])* = Assemble the hybrid decryption key k that unlocks the ciphertext u while using the Shamir secret sharing primitive.

8.2 Signing Key Encryption on Ethereum

Every validator in the Stakehouse validator shall backup its Signing Key using CIP as a part of the staking process prior to depositing 32 ETH with Etheruem Deposit Contract, generating a random hybrid key pair (τ, h), and encrypting the validator signing key m for the SH-CIP DKG key $\overline{PK}$.

$$u = HE.encrypt_{\overline{PK}}(m, \tau) \tag{4}$$

After the encryption is completed, the validator BLS public key m, ciphertext u and an encryptor public key h is registered in the Stakehouse registry.

8.3 Signing Key Decryption MPC

8.3.1 Recovery request

There are two distinct scenarios of decryption requests that were submitted, a) a single party initiated the recovery request b) The request was initiated by a multi-party coordinated address via EIP-712 signature. For successful recovery, the system must pass the Core assumptions.

8.3.2 Single-party decryption request

Requestor will generate a Hybrid Encryption (AES) key for instigating the recovery process communication, with the MPC solvers (sk_r, pk_r).

Further, the requester will add the BLS public key v of the Signing Key validator, and submit pk_r and v to the *SafeBox* smart contract as a blockchain transaction, upon successfuly meeting core assumptions, an event message is broadcasted to the MPC solver network.

8.3.3 Multi-party decryption request (fractionalized/pooled validator)

For multiple owner's requests for a signing key recovery, they shall coordinate to generate the above-mentioned requestor key pair (sk_r, pk_r), and then consolidate their consent of recovery request by forming a set of EIP712 signatures (Eq. 5) containing a deadline t, destination $d = SafeBox$, hybrid public key pk_r, and a BLS public key v.

$$S := \{\beta_i | \beta_i = eip712Sign([v, pk_r], d, t)\} \tag{5}$$

The set of above signatures need to satisfy the aggregate signature holds Core Assumption property, that is the signatures would be issued only by collateralized SLOT owners totaling a balance over 2 Collateralized SLOT tokens.

Eventually, S, v, and pk_r is sent to the smart contract by an arbitrary party, and the work is terminated.

8.4 Decryption procedure

Solvers listen to *SafeBox* events for receiving decryption requests, and upon fetching the request data from *SafeBox* perform the **Core Assumption** feasibility checks.

Once the Core Assumption checks pass, solvers fetch the ciphertext encryptor public key h, and perform the following computation involving their secret shared key $\overline{sk_i}$:

$$s_i = h^{\overline{sk_i}} \tag{6}$$

Here, $i \in \{1, ..., N\}$ is the solver index, s_i is the partial decryption solution, and the is an *Elliptic Curve* multiplication.

Once a partial decryption solution (Eq. 6) is formed, the value is encrypted for the recipient.

$$e_i = HE.encrypt_{pk_r}(s_i, sk_i) \tag{7}$$

Further, the MPC solver forms a *Non-Interactive Zero Knowledge* (NIZK) Proof to prove that the partial decryption solution corresponding to the encryption public key h and the solver shared secret key $\overline{sk_i}$ formed correctly:

$$NIZK_i = formNizkProof(h, \overline{sk_i}, s_i) \tag{8}$$

Once the NIZK formation is completed, each solver posts $NIZK_i$ and e_i to the *SafeBox* smart contract and terminate the requested computation.

Note: The ciphertext u hiding the BLS signing key is not necessary to form a partial decryption solution, and the only information used in forming partial decryption solution is the encryptor signing key h.

8.5 NIZK Proof

Note: $c_1 = g^r$, $c_2 = h^r$, $h = g^{sk}$, where $r \in \mathbb{Z}_q$ is a random point drawn from the field.

Non-Interactive Zero Knowledge Proof (NIZK) will enable a deterministic validation of the correctness of received encrypted solution from an MPC solver by the recipient, i.e; prove the correctness of the plaintext m with respect to the given ciphertext u.

We are given the public key of the encryptor h, the generator point g, the encryption elements c_1, and c_2 and the ciphertext u. In this case we want to ensure that the contents of the ciphertext correspond to the global DKG public key $\overline{PK}$.

1. Pick a random integer $t \in Z_q$ and compute $w_1 := g^t$, $w_2 := c_1^t$
2. Compute a hash $e = hash(pk, c_1, u, c_2, w_1, w_2)$
3. Compute $z = t + e \cdot sk \bmod q$

Finally the NIZK proof is summarized in an output $\pi = (w_1, w_2, z)$

Here it is assumed that the verifier(recipient) already has pk, c_1, c_2 and u, and is provided by the prover. The proof is then checked as follows:

1. Verifier computes $e = hash(pk, c_1, u, c_2, w_1, w_2)$
2. Checks the condition $h^e \cdot w_1 = g^z$
3. Checks the condition $c_2^e \cdot w_2 = c_1^z$

If the conditions from steps 2 and 3 hold, the proof is considered as passed, else the proof has failed.

8.6 Key assembling procedure

Decryption procedure first starts with the recipient fetching the following data points from the chain:

- e_i - Ciphertext hiding the partial decryption solution from solver i
- $NIZK_i$ - Non-Interactive Zero Knowledge proof of data correctness from solver i
- $\overline{PK}$ - Global public key generated during the DKG procedure
- u - Ciphertext hiding the signing key

Then, the decryptor makes sure that the solver submitting the decryption solution has at least 1 collateralized SLOT tokens in the Stakehouse universe (Core Assumption) and is not resigned; also, no historic complaints exist are recorded against the MPC solver in the SafeBox data. Although SafeBox has performed the checks, the decryptor re-checks to tighten the security. Upon receiving the encrypted partial decryption pieces e_i and the ZK proofs proving the content validity, the recipient does the following.

First, the recipient decrypts the encrypted decryption solutions using their secret key sk_r and the solver's public key pk_i:

$$s_i = HE.decrypt_{pk_i}(e_i, sk_r) \tag{9}$$

Further, the decryptor verifies each NIZK proof to make sure that the partial decryption solution is valid:

$$verifyNIZK(pk_s, pk_e, s_i, NIZK_i) \tag{10}$$

If the partial decryption solution is invalid, it is discarded, and if the remaining number of pieces is bigger than the threshold T, we can apply the Shamir's secret sharing principle and assemble the hybrid key.

$$k = assembleHybridKey(\{s_i | verifyNIZK(pk_s, pk_e, s_i, NIZK_i) = 1\}) \tag{11}$$

Finally, the recipient uses the assembled hybrid key k to decrypt the ciphertext u:

$$m = HE.decrypt_{\overline{PK}}(u, k) \tag{12}$$

In the case solvers failed to provide T valid partial decryption solutions or went offline for a period of time the application can be filed again once the current application expires (approximately 21 hours/6300 Ethereum Blocks), where the identical sequence of actions will be performed.

9 SH-CIP DKG

9.1 Distributed Key Generation (DKG)

Distributed key generation (DKG) enables untrusted parties to collaborate for generating and hold a shared secret without individually sufficient access. In a distributed setting, the DKG protocol allows a set of nodes to collectively generate a secret with its shares maintained among the participating nodes, such that a subset of defined quorum with a threshold (T) can process requests. A quorum subset size greater than or equal to the specified threshold would be able to use the shared secret, whilst the other members don't have any knowledge about it (in simpler terms, we don't need the participation of all N signers, just M out of N signers).

In dCommon, we use DKG for having shared credentials to operate a Multi-Party Computation network and its solvers to compute NIZK proofs for requested work from a CIP application via SafeBox contract.

9.2 DKG Actors

Table 4: Distributed Key Generation actor roles

ACTORS	STAKEHOUSE	MPC - ACTIONS
Applicant	Collateralized SLOT token holder	Register for DKG procedure to become an MPC solver for Stakehouse
MPC Solver	A Registered SLOT token holder with CIP-DKG	Serve decryption request for stakehouse users performing computation
DKG Member	SLOT token holder who has shared private key of DKG public key	Perform DKG process and validate the committee to acquire credential for MPC

The distributed key generation procedure happens in five rounds. The progress of each participant is tracked via the status system, and is summarized in the table below (Tab. 5).

During the DKG process, the CIP "DkgRegistry" contract ensures the following:

- Holding DKG-related constants:
  - *threshold* - Quorum required to successfully complete the DKG procedure. Also describes the polynomial degree used in cryptography

Table 5: Distributed Key Generation lifecycle roles

STATUS ID	DKG LIFE CYCLE STATUS NAME	DESCRIPTION
0	NON_BOOTSTRAP_GUARDIAN	Does not have a right to participate in the DKG procedure
1	NO_SUBMISSION	Has a right to participate in the DKG procedure, but haven't started yet
2	ROUND1_SUBMITTED	Round 1 validation has been submitted to the SafeBox
3	ROUND2_COMPLAINT_PROCESSED	Complaints pertained to Round 1 submitted or no complaints received
4	ROUND3_SUBMITTED	Round 3 validations got accepted to the SafeBox
5	ROUND4_COMPLAINT_PROCESSED	Complaints pertained to round 3 were processed or no complaints received
6	ROUND5_SUBMITTED	Final Validations submitted to the SafeBox

  - *startBlock* - Starting block of the DKG procedure, set future to deployment block
  - *roundTime* - Block intervals for participants to complete each round.
  - *initialParticipantCount* - Total participant count at the start of the DKG procedure

- Keeping track of DKG complaints
- Enforcing block-based time boundaries. Participants who failed to complete the round within the stipulated time period will be disqualified for the succeeding round.
- Enforcing basic constraints on submitted data like length-checks and checking if the status for each solver is increased monotonically
- Emitting submitted data in easy-to-query Ethereum events

Setup: A set of participants $\chi := \{P_1, ..., P_N\}$ each with its own index $\mathcal{N} := \{1, ..., N\}$ and public hybrid key pk_i and a secret hybrid key sk_i.

9.2.1 Round 1: Initializing and Distributing credentials

Each participant P_i establishes the initial numerical credentials for himself and other participants. As follows:

First, each participant P_i picks random elements $\{a_{i,0}, ..., a_{i,t}\}$, $\{b_{i,0}, ..., b_{i,t}\}$ from the prime integer field $\mathbb{Z}_q$ and uses them to define 2 polynomials $f_i(x) := \Sigma_{l\in\{0,..,t\}} a_{i,l}x^l$ and $f'_i(x) := \Sigma_{l\in\{0,..,t\}} b_{i,l}x^l$. Here $t := floor(N/2) - 1$ is defined as the *threshold number*.

Further, each participant computes $t+1$ cryptographic reference points $E_{i,l} := g^{a_{i,l}}u^{b_{i,l}}$ and posts them to the *SafeBox* smart contract. Here g and u are generator point and a common base respectively and are specified in the *SafeBox* smart contract.

Finally, for every participant excluding themselves $P_j \in \chi \setminus P_i$, the participant P_i evaluates both polynomials $s_{i,j} := f_i(j)$ and $s'_{i,j} := f'_i(j \in \mathcal{N})$ and posts the encrypted result $e_{i,j} := HE.encrypt_{pk_j}([s_{i,j}, s'_{i,j}], sk_i)$ to the *SafeBox* contract.

9.2.2 Round 2: Cryptographic Credential Validation

Now, each participant P_i fetches the cryptographic reference points $\{E_{i,j}\}$ and individually encrypted credentials $\{e_{j,i}\}$ ($(j = i)$ credential can be obtained off-line) from the *SafeBox* smart contract and will dedicate this round for the credential validation.

First, each participant P_i decrypts the personalized polynomial values $s_{j,i}, s'_{j,i} = decrypt_{pk_j}(e_{j,i}, sk_i)$ and performs the following equation check involving the hybrid public key h of P_i:

$$g^{s_{j,i}} h^{s'_{j,i}} = \Pi_{l \in \{0,...,t\}} (E_{j,l})^{i^l} \tag{13}$$

In the case when Equation (Eq. 13) is not satisfied, a Non-Interactive Zero Knowledge complaint $\pi := NIZK(s_{j,i}, s'_{j,i}, sk_i, e_{j,i})$ is formed and posted on the *SafeBox* together with the polynomial evaluations $s_{j,i}$ and $s'_{j,i}$.

It is important to note, that if the NIZK proof π is verified successfully by all the other participants except P_j, then the participant P_j is disqualified from the procedure, otherwise the participant P_i is disqualified for false accusations.

9.2.3 Round 3: Obtaining the shared secret key

At this point each participant P_i defines a set of participants who passed the data checks in the previous round $P_j \in \Omega$ and their indices $\Im := \{i | P_i \in \Omega\}$. Each participant P_i dedicates this round to generate the shared secret key $\overline{sk_i}$ and set the ground for generating the shared public key $\overline{pk_i}$ and the DKG public key $\overline{PK}$.

The process for obtaining the shared public key $\overline{pk_i}$ and the DKG public key $\overline{PK}$ is started by each participant posting reference points $A_{i,l} := g^{a_{i,l}}$ $(l \in 0, ..., t)$ to the blockchain and the shared private key is assembled by adding all the parts $s_{j,i}$ that were deemed valid from the previous round $\overline{sk_i} := \Sigma_{j \in \Im} s_{j,i}$

9.2.4 Round 4: Secondary reference checks

In this round, each participant P_i validates secondary cryptographic credentials $A_{j,l}$ $(j \neq l)$ submitted by other participants $P_j \in \Omega \setminus P_i$

In order to perform validation, each participant checks the following equation:

$$g^{s_{j,i}} = \Pi_{l \in \{0,...,t\}} (A_{j,l})^{i^l} \tag{14}$$

If the equation (Eq. 14) is violated, then NIZK complaint $\pi := NIZK(s_{j,i}, s'_{j,i}, sk_i, e_{j,i})$ is revealed together with the $s_{j,i}$ and $s'_{j,i}$. Here same complaint check rules hold as in Round 2.

9.2.5 Round 5: Deriving final distributed credentials

Round 5 is the final round for distributed key generation procedure and forming the DKG public key $\overline{PK}$.

Each remaining participant who has successfully performed validation for the Round 2 and Round 4 checks defines a set Θ which contains other participants who made it to round 5. Additionally, each participant defines a set of indices of the remaining participants $\Re := \{i | P_i \in \Theta\}$.

To give more context based on the above mentioned **setup**, we arrive at the following set relations $\Theta \subseteq \Omega \subseteq \chi$, and $\Re \subseteq \Im \subseteq \mathcal{N}$.

Further, every party excludes all the information received by solvers in the set $P_i \in \chi \backslash \Theta$, and reconstructs their zero'th coefficient $a_{j,0} = \Sigma_{i \in \Re} s_{i,j} \cdot \lambda_i$. Where $\lambda_i = \Pi_{j \in \Re \backslash \{i\}} \frac{l}{l-i}$ is the *Lagrange* coefficient.

Finally, each participant P_i can obtain the shared public key $pk_i = \Pi_{j \in \Re} \Pi_{l \in \{0,\ldots,t\}} (A_{j,l})^{i^l}$ and their DKG public key $\overline{PK} = \Pi_{j \in \Re} A_{j,0}$.

After 5-round DKG procedure, each participant $P_i \in \Theta$ will have a shared private key $\overline{sk_i}$, a shared public key $\overline{pk_i}$ together with 1 joint DKG public key $\overline{PK}$.

The outcome of this 5-round procedure for each participant is the following:

- **DKG public key** - Every remaining DKG participant arrives at the same value hence qualify for being a solver for MPC. This key will be used for both encrypted backup and recovery requests of the Signing Key, and also for all related computation requests with the MPC solver network for the Stakehouse protocol CIP application.
- **Shared private key** - Each participant arrives at an individual value. This value is used as a solver key for forming partial decryption pieces
- **Shared public key** - Individual to each owner and publicly known by the network. This value is utilized for correctness proofs while providing decryption requests. (Hybrid encryption key)

Transitions in the Distributed Key Generation can be seen below (Fig. 3):

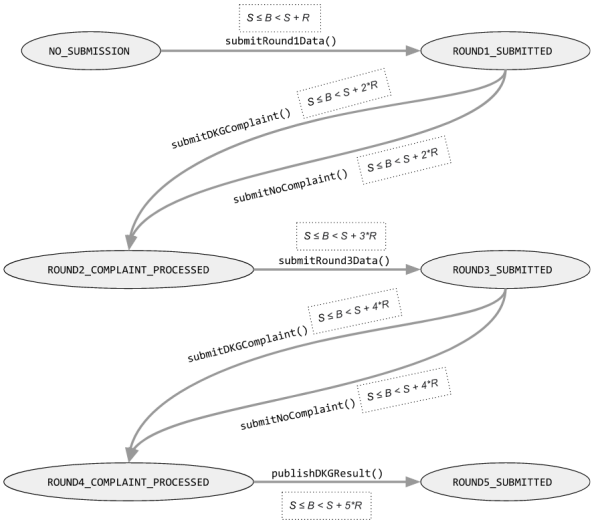

Figure 3: CIP-DKG state transitions

10 Handover DKG - Solver Committee Expansion

The initial setup consists of the old committee members $P_i \in P$ each with a pair of "old" pair of hybrid credentials $(\overline{sk_i}, \overline{pk_i})$ and a set of new committee members $S_i \in S$ with a

new set of hybrid credentials $(\hat{sk}_i, \hat{pk}_i)$. Here we will denote the number of old solvers by N and a number of new solvers by M.

10.1 Old solvers

Each old solver P_i sets up the ground for new solvers S by first picking a set of random elements $\{a_{i,1}...a_{i,k}\} \subset \mathbb{Z}_q$ and defining the zero'th element to be $a_{i,0} := \overline{sk}_i$.

Further, each solver P_i defines a cryptographic reference point $E_{i,l} := g^{a_{i,l}}$ for $l \in \{1, ..., k\}$ and posts it to the *SafeBox* smart contract. Here $k = floor(M/2) - 1$ is the new decryption threshold.

And finally, similar to the DKG procedure, each solver defines a polynomial $f_i(x) = \Sigma_{l\in\{0,..,k\}} a_{i,l}x^l$, and evaluates the polynomial for each solver except oneself $s_{i,j} := f_i(j)$ $(i \neq j)$ and posts the encrypted result to the *SafeBox* $e_{i,j} := HE.encrypt_{\hat{pk}_j}(s_{i,j}, \overline{sk}_i)$.

10.2 New solvers

Once the old solvers submitted at least t handover pieces, each new solver S_i fetches the encrypted polynomial values $e_{j,i}$, and reference points $E_{j,l}$ and obtains the personalized polynomial values $s_{j,i} = HE.decrypt_{\overline{pk}_j}(e_{j,i}, \hat{sk}_i)$ by decrypting $e_{j,i}$.

Further, each new solver evaluates the following equation with the adjusted reference point $E_{j,0} = \overline{pk_j}$:

$$g^{s_{j,i}} = \Pi_{l\in\{0,..,k\}}(E_{j,l})^{i^l} \tag{15}$$

As in the handover procedure, in case the equation (Eq. 15) is violated, the NIZK proof $NIZK(s_{j,i}, e_{j,i}, \hat{sk}_j)$ together with the personalized polynomial value $s_{j,i}$ is posted to the *SafeBox* smart contract. As before, a valid complaint against P_j will disqualify P_j, and an invalid complaint against P_j will disqualify S_i.

After data filtering is completed, each new participant defines a set of solvers who passed the previous checks Θ and a set of indices belonging to those solvers $\Re := \{i | P_i \in \Theta\}$. The new solvers then have enough information to compute their shared credentials $\tilde{sk}_i = \Sigma_{l\in\Re} s_{l,i} \cdot \gamma_l$ and $\tilde{pk}_i = \Pi_{l\in\Re}\Pi_{p\in\{0,...,k\}}(E_{l,p})^{l^p\cdot\gamma_l}$, where $\lambda_i = \Pi_{j\in\Re\setminus\{i\}} \frac{l}{l-i}$ is the previously mentioned *Lagrange coefficient*.

The completed handover process finally yields an extended set of shared credentials $\{(\tilde{sk}_i, \tilde{pk}_i)\}$ without changing the DKG public key $\overline{PK}$.

10.3 Solver Registration

Any SLOT token holder who holds 1 or more Collateralized SLOT tokens from a KNOT on unique ECDSA can register to become a solver for SH-CIP application. Registration can be performed with a DKGRegistry contract, however, until the user completes the DKG procedure and obtain a shared key credential he/she cannot partake as a solver.

11 Economic analysis for Ad-hoc committee

Stakehouse CIP - MPC DKG committee selection and handover(expansion) are to be performed in a decentralized and trustless manner, ensuring permissionless access to become an MPC solver by any Stakehouse Validator who holds a minimum of 1 SLOT token from a unique ECDSA address and is registered through DKGRegistry.

Like any ad-hoc network selection, SH-CIP DKG demands a strong entropy for candidate selection; Hence all candidates are required to go through a randomized selection process using RANDAO [14] points and pseudo-stochastic bifurcations. The randomization process will ensure any frontrunning, monopoly attacks; this process will strengthen the DKG committee threshold requirement from coordinated attacks.

The table below summarizes the risk matrix of the DKG committee of 100 members as MPC Solver from more than 100 candidates selected randomly. The DKG Committee requires a minimum of 100 requests for candidacy, so in the below table, we assume the number of requests $s \geq 100$. Further, the initial committee number of candidates is fixed at 100. The committee size n satisfies $n \leq 100$, also the threshold value $t = floor(N/2) - 1$ satisfies $t \leq 49$. To compromise the committee, an attacker needs to control at least $t+1 = 50$ candidates, so we get $f(s_a, s) = 0$ for the number of malicious candidates $s_a < 50$. The attacker needs to submit at least 50 requests for candidacy, and because each request requires the requester to hold at least 1 collateralized SLOT from a KNOT(validator), it requires the attacker to form at least 50 unique KNOTS (each 32 ETH).

The tables (Tab. 6a, Tab. 6b) summarize $f(s_a, s)$ for $s_a = 50$ and $s_a = 60$.

Table 6: Probabalistic committee takeover analysis for 50 and 60 malicious solvers

s	$f(50, s)$
100	100.000%
105	3.603%
110	0.161%
115	0.009%
120	0.001%
125	0.000%
130	0.000%
135	0.000%
140	0.000%

(a) Attack success probabilities with 50 malicious solvers

s	$f(60, s)$
100	100.000%
105	100.000%
110	100.000%
115	93.230%
120	59.647%
125	25.162%
130	8.042%
135	2.180%
140	0.538%

(b) Attack success probabilities with 60 malicious solvers

A security analysis of the Stakehouse CIP application for the initial ad-hoc committee of 100 members can be seen in the audit report done by Runtime Verification [2].

Additionally to the above analysis, an empirical simulation with a committee size set to 100 and the number of malicious applicants set to 49 was executed to confirm the

analytical calculations [19].

12 Initial DKG and MPC service

DKG gives a strong base for a trustless committee to operate a non-interactive MPC network, regardless of its size and liveness in an adversarial environment. We believe Stakehouse shall have a DKG-based ad-hoc MPC network that is derived from its own stakeholders - SLOT tokens and able to expand the committee set and possibly bring subcommittee for further modularization of the network. Like any permissionless network having a Sybil-resistant self-governance is an evolving process with a cold start, we set to begin the Stakehouse initially bootstrap with a 20-member DKG that will not be eligible to participate in any future DKG committees. As soon as the Stakehouse will attain enough validators to maintain the initial threshold of 100 members set participants from its registered validators, the initial bootstrap committee will terminate and a new DKG will be performed. The initial service phase will be more of an alpha phase with controlled performance, during this time period CIP will be an "as is" service for Stakehouse facilitated by Blockswap.

13 Reset DKG - Switching the SafeBox

dCommon allows to reset the committee ensuring robustness and secure continuity of the system as a backstop, this could be done via social consensus of Stakehouse stakers anytime abandon their exiting DKG set and re-encrypt signing Key with another DKG-PublicKey for appointing new MPC solver set for decryption service. Reset also act as a healing mechanism in case the existing committee falls below the quorum and consistently fails to achieve the set threshold for decryption service, CIP application can switch to a new committee with a new threshold and committee size.

The process is fairly straightforward by simply broadcasting the message to the blockchain by specifying the new ciphertext c' and the encryptor dCommon public key k.

14 Complexities

The below table summarizes the asymptotic complexities of each process in the CIP application, the system design to handle a large set of actors.

Table 7: Asymptotic runtime complexities for core procedures

PROCESS	ASYMPTOTIC COMPLEXITY	SCALING CLASS
Distributing messages to other participants	O(N)	Linear
Key recovery request	O(1)	Constant
Key Recovery execution	O(N)	Linear
Distributed Key Generation	O(N)	Linear
Handover	O(N)	Linear

Proof size & Gas cost:

ZK proof size in bytes: **259 Bytes**
Gas cost per decryption of 1 solver: **82684 gas**

15 Solver Resignation

A solver in SH-CIP MPC can exit the system at their choice via submitting a request by calling a SafeBox smart contract function *refuseGuardianDuties()*. Irrespective of the phase (application, execution, or handover).

It is of paramount importance that **RageQuit** is available for all participants upholding the permissionless principle. However once resigned from being a solver, it is permanent and the user cannot rejoin in the process in the future both off-chain and on-chain elements of dCommon - CIP

15.1 Solver Blacklist

Registries gives the ability to reconcile its data in isolation deterministically, and without incurring on-chain transactions with its oUTXO. CIP leverages this construct for maintaining a blacklist registry - an append only registry that gives the ability to reveal private values submitted by the solver in combination with their AES key and NIZK proof that verifies the correctness of revealed information and serves as a validation for an encryption complaint.

CIP as it was named weighed on the common interest of the network and the participants, to maintain an optimal MPC network. The registry allows discarding any solver who had a history of misbehavior complaints. The mechanism is highly useful for a on-chain reputation for dCommon continuous hygiene on solver selection for CIP applications and its users, to filter out solvers who have a record of submitting malicious information or wrongly accusing other parties of misbehaviour.

16 CIP Monitoring

CIP is an auditable and publicly verifiable MPC scheme for Ethereum applications, hence observability is very key for all user-driven actions for others to monitor. CIP uses the widely used indexer service graph protocol to expose the application data in a queriable manner as a dedicated subgraph for other protocols to utilize and bring more transparency and accountability for Stakehouse assets and node performance.

Alternate to this, CIP application and its MPC operational data are always accessible from any Ethereum execution node or any other indexing services of users' choice.

16.1 The Graph

The graph [1] is an indexing service that works by being triggered by the Ethereum events. The data points tracked by the CIP subgraph [4] are the following:

- **totalNumberOfGuardianRegistrations** - Number of users who signed up to be solvers
- **solverPass** - The number of partial decryption solutions to be obtained for assembling the private key necessary for decryption
- **numberOfInitialGuardians** - Number of initial Participants in the DKG procedure

- **totalNumberOfDecryptionRequests** - Total number of times a request been made to compute the partial decryption solution for key recovery

Decryption Request:

- **id** - BLS public key and internal nonce concattenation to indicate the decryption request target and how many times was recovery requested
- **requesters** - Array of addresses that requested decryption
- **stakehouse** - The stakehouse to which the recovered validator belongs
- **blsPublicKey** - The BLS public key of the validator
- **nonce** - Decryption request number
- **recipientAesKey** - Hybrid public key which will receive the information posted by the committee (needed to establish private communication via public space)
- **totalNumberofPiecesReceived** - Total number of partial decryption solutions posted on the SafeBox
- **blockNumber** - Block Number during which the request was made

Decryption Piece:

- **id** - Transaction hash during which the partial decryption solution was submitted
- **solver** - The committee memeber who submitted the partial decryption solution
- **recipientAesKey** - Hybrid public key under which the partial decryption solution is encrypted
- **blsPublicKey** - BLS public key for which the recovery is being assisted
- **ciphertext** - Ciphertext hiding the partial decryption solution
- **zkProof** - ZK proof for the user to verify that the solution was correct
- **nonce** - The number of recoveries assisting requests that were made for this BLS public key

17 Disclaimer

This document is not final; hence the title "position paper," is for general information purposes only. It does not constitute investment advice or a recommendation or solicitation to buy or sell any investment and should not be used in the evaluation of the merits of making any investment decision. The opinions and analysis reflected herein are subject to change or update without being updated. This paper reflects the current opinions of the authors and is not made on behalf of Blockswap Labs, or its affiliates, and does not necessarily reflect the opinions of Blockswap Labs, Blockswap Foundation, or their affiliates, or individuals associated with them. You may refer to Stakehouse and Common Interest Protocol documentation for their latest published information.

Cryptographic Safe Box Specification

18.1 Building Blocks

18.1.1 Elliptic Curve Over $\mathbb{F}_p$

The implementation of our scheme is based on elliptic curve groups for efficiency. Let $\sigma := (p, a, b, g, q, \zeta)$ be the elliptic curve domain parameters over $\mathbb{F}_p$, consisting of a prime p specifying the finite field $\mathbb{F}_p$, two elements $a, b \in \mathbb{F}_p$ specifying an elliptic curve $E(\mathbb{F}_p)$ defined by $E : y^2 \equiv x^3 + ax + b \pmod{p}$, a base point $g = (x_g, y_g)$ on $E(\mathbb{F}_p)$, a prime q which is the order of g, and an integer ζ which is the cofactor $\zeta = \#E(\mathbb{F}_p)/q$. We denote the cyclic group generated by g by $\mathbb{G}$, and it is assumed that the DDH assumption holds over $\mathbb{G}$, that is for all p.p.t. adversary $\mathcal{A}$:

$$\mathsf{Adv}^{\mathsf{DDH}}_{\mathbb{G}}(\mathcal{A}) = \left| \Pr \left[\begin{array}{l} x, y \leftarrow \mathbb{Z}_q; b \leftarrow \{0,1\}; h_0 = g^{xy}; \\ h_1 \leftarrow \mathbb{G} : \mathcal{A}(g, g^x, g^y, h_b) = b \end{array} \right] - \frac{1}{2} \right| \leq \epsilon(\lambda) \ ,$$

where $\epsilon(\cdot)$ is a negligible function.

18.1.2 Hybrid Encryption

The hybrid encryption scheme consists of the following 4 PPT algorithms.

- $\mathsf{Gen}_{\mathsf{gp}}(1^\lambda)$: take input as security parameter $\lambda \in \mathbb{N}$, and output a group parameter σ.
- $\mathsf{HE.KeyGen}(\sigma)$: pick random $s \leftarrow \mathbb{Z}_q^*$ and set $h := g^s$, and output $(\mathsf{pk} := (g, h), \mathsf{sk} := s)$.
- $\mathsf{HE.Enc}_{\mathsf{pk}}(\sigma; m)$: pick random $r \leftarrow \mathbb{Z}_q$; compute $c_1 := g^r$ and $c_2 := h^r$; set $k \leftarrow \mathsf{hash}(c_2)$; compute $u := \mathsf{AES\text{-}GCM}_k(m)$; output $C = (c_1, u)$.
- $\mathsf{HE.Dec}_{\mathsf{sk}}(\sigma; C)$: compute $c_2 := (c_1)^{\mathsf{sk}}$; set $k \leftarrow \mathsf{hash}(c_2)$; output $m := \mathsf{AES\text{-}GCM}_k^{-1}(u)$.

18.1.3 ZK Proof for Decryption

We construct a non-interactive zero-knowledge (NIZK) proof to show the correctness of decryption w.r.t. the hybrid encryption scheme. Actually, the prover only needs to provide $c_2 := (C_1)^{\mathsf{sk}}$ and show its correctness. The verifier then computes $k \leftarrow \mathsf{hash}(c_2)$ and checks the decryption by $m = \mathsf{AES\text{-}GCM}_k^{-1}(u)$. The NIZK protocol is depicted in (Fig. 4).

NIZK proof for decryption correctness $\Pi_{\mathsf{nizk}}^{\mathsf{HE.Dec}}(\mathsf{pk}, \mathsf{sk})$

Statement: $\mathsf{pk} := (g, h)$, $C := (c_1, u)$ and c_2

Witness: sk

Prove:

- Pick random $t \leftarrow \mathbb{Z}_q$ and compute $w_1 := g^t$ and $w_2 := c_1^t$;
- Compute $e \leftarrow \mathsf{hash}(\mathsf{pk}, C, c_2, w_1, w_2)$;
- Compute $z := t + e \cdot \mathsf{sk} \pmod{q}$;
- Output $\pi := (w_1, w_2, z)$;

Verify:

- Compute $e \leftarrow \mathsf{hash}(\mathsf{pk}, C, c_2, w_1, w_2)$;
- Output 1 if and only if the following holds:
  - $h^e \cdot w_1 = g^z$;
  - $c_2^e \cdot w_2 = c_1^z$;

Figure 4: NIZK proof for decryption correctness $\Pi_{\mathsf{nizk}}^{\mathsf{HE.Dec}}$

18.1.4 Distributed Key Generation

Distributed key generation (DKG) is a fundamental building block of the proposed cryptographic safe box system. Ideally, the protocol termination should be guaranteed when up to $t = \lceil \frac{n}{2} \rceil - 1$ out of n committee members are corrupted. We will adopt the distributed key generation protocol proposed by Gennaro et al. [9]). Denote the committee as $\mathcal{P} := \{P_1, \ldots, P_n\}$. In a nutshell, the protocol lets the committee members P_i first post a "commitment" of pk_i. After sharing the corresponding sk_i via $(t+1, n)$-threshold VSS, the committee members P_i then reveal pk_i. We will use the blockchain to realize the broadcast channel and peer-to-peer channels. Our distributed key generation protocol $\Pi_{\mathsf{DKG}}^{t,n}$ is depicted in (Fig. 5). It allows us to accommodate up to $t < n/2$ malicious players in the protocol. That is, guaranteeing that with $\lfloor \frac{n}{2} \rfloor + 1$ honest players, all the players should be able to agree on a uniformly random public key pk such that no malicious players can influence the distribution of the generated public key. The corresponding secret key is shared among all committee members.

Distributed key generation $\Pi^{t,n}_{\mathsf{DKG}}$

Setup: Each party $P_i \in \mathcal{P}$ is associated with a long-term public key pk_i, and P_i holds the corresponding secret key sk_i, $i \in [n]$.
CRS: The commitment key $u \in \mathbb{G}$.

Round 1: Each party $P_i \in \mathcal{P}$ does the following:

- Pick random $a_{i,0}, a_{i,1}, \ldots, a_{i,t}, b_{i,0}, b_{i,1}, \ldots, b_{i,t} \leftarrow \mathbb{Z}_q$.
- Define two polynomials $f_i(x) := \sum_{\ell=0}^{t} a_{i,\ell} x^\ell$ and $f'_i(x) := \sum_{\ell=0}^{t} b_{i,\ell} x^\ell$.
- For $\ell \in \{0, \ldots, t\}$, post $E_{i,\ell} := g^{a_{i,\ell}} u^{b_{i,\ell}}$ on the blockchain.
- For every other $P_j \in \mathcal{P}$, $j \neq i$, compute $s_{i,j} := f_i(j)$ and $s'_{i,j} := f'_i(j)$;
  Post $e_{i,j} \leftarrow \mathsf{HE.Enc}_{\mathsf{pk}_j}(s_{i,j}, s'_{i,j})$ on the blockchain.

Round 2: Each party $P_i \in \mathcal{P}$ does the following:

- Fetch $\{e_{j,i}\}_{j\in[n], j\neq i}$ from the blockchain, and use sk_i to decrypt them, obtaining the corresponding shares $\{(s_{j,i}, s'_{j,i})\}_{j\in[n], j\neq i}$.
- For $j \in [n], j \neq i$, check if $g^{s_{j,i}} h^{s'_{j,i}} = \prod_{\ell=0}^{t} (E_{j,\ell})^{i^\ell}$. If not, post complain against P_j by revealing the evidence: $(s_{j,i}, s'_{j,i})$ and
  $\pi \leftarrow \mathsf{NIZK}\left\{((s_{j,i}, s'_{j,i}, \mathsf{pk}_i, e_{j,i}), (\mathsf{sk}_i)) : (s_{j,i}, s'_{j,i}) = \mathsf{HE.Dec}_{\mathsf{sk}_i}(e_{j,i}) \ \wedge \ (\mathsf{pk}_i, \mathsf{sk}_i) \in \mathcal{R}_{\mathrm{PKE}}\right\}$
- (One valid complain against $P_j \in \mathcal{P}$ will disqualify P_j.)

Round 3: Define the indices of the qualified set of parties as $\mathcal{J}$. Each qualified party P_i does:

- For $\ell \in [t]$, post $A_{i,\ell} := g^{a_{i,\ell}}$ to the blockchain.
- Return its secret key share as $\overline{\mathsf{sk}}_i := \sum_{j\in\mathcal{J}} s_{j,i}$.

Round 4: Each qualified party P_i does the following:

- For $j \in \mathcal{J}, j \neq i$, check if $g^{s_{j,i}} = \prod_{\ell=0}^{t} (A_{j,\ell})^{i^\ell}$. If not, post complain against P_j together with the evidence $(s_{j,i}, s'_{j,i})$ on the blockchain. Such that $g^{s_{j,i}} h^{s'_{j,i}} = \prod_{\ell=0}^{t} (E_{j,\ell})^{i^\ell}$ and $g^{s_{j,i}} \neq \prod_{\ell=0}^{t} (A_{j,\ell})^{i^\ell}$.

Round 5: Each qualified party P_i do the following:

- If there is a valid complain against P_j, $j \in \mathcal{J}$, then post the share $s_{j,i}$ on the blockchain.
  (Everyone can reconstruct $a_{j,0} := \sum_{i\in\mathcal{J}} s_{i,j} \cdot \lambda_i$ and re-define $A_{j,0} := g^{a_{j,0}}$, where $\lambda_i := \prod_{\ell\in\mathcal{J}\setminus\{i\}} \frac{\ell}{\ell - i}$ are the Lagrange coefficients.)
- For $m \in \mathcal{J}$, compute P_m's partial public key $\overline{\mathsf{pk}}_m := \prod_{j\in\mathcal{J}} \prod_{\ell=0}^{t} (A_{j,\ell})^{i^\ell}$
- Return the election public key as $\overline{\mathsf{pk}} := \prod_{j\in\mathcal{J}} A_{j,0}$ and partial public keys $\{\overline{\mathsf{pk}}_m\}_{m\in\mathcal{J}}$.

Figure 5: Distributed key generation $\Pi^{t,n}_{\mathsf{DKG}}$

18.1.5 Threshold Decryption

The threshold decryption protocol $\Pi^{t,n}_{\mathsf{TDec}}$ is executed among a set of parties $\mathcal{P} := \{P_1, \ldots, P_n\}$. The adversary is allowed to corrupt up to $t = \lceil \frac{n}{2} \rceil - 1$ parties, and the remaining parties can still compute a sufficient amount of partial decryption solutions. As depicted in (Fig. 6), the protocol uses blockchain as the broadcast channel. In a nutshell, we let the committee members to compute partial decryption solutions and encrypt the partial decryption solutions under the recipient's public key pk. Subsequently, the recipient can use its secret key sk to decrypt them and assemble the key that allows to decrypt the message. To ensure partial decryption solution correctness, the committee members are

also required to submit a NIZK proof.

Multi-Party Computation $\Pi^{t,n}_{\mathsf{TDec}}$

Setup: Each party P_i has a partial decryption key pair $(\overline{\mathsf{pk}}_i, \overline{\mathsf{sk}}_i)$.

Input: Encryption hybrid public key c_1 and the recipient's public key pk.

Multi-Party computation: Each party $P_i \in \mathcal{P}$ does the following:

- Compute $D_i := (c_1)^{\overline{\mathsf{sk}}_i}$ and encrypt $e_i \leftarrow \mathsf{HE.Enc}_{\mathsf{pk}}(D_i)$.
- Compute the corresponding NIZK proof $\Pi^{\mathsf{HE.Dec}}_{\mathsf{nizk}}(\overline{\mathsf{pk}}_i, \overline{\mathsf{sk}}_i)$ (cf. Fig. 4):

$$\sigma_i \leftarrow \mathsf{NIZK}\left\{(g, \overline{\mathsf{pk}}_i, c_1, D_i), (\overline{\mathsf{sk}}_i) : D_i = (c_1)^{\overline{\mathsf{sk}}_i} \ \wedge \ \overline{\mathsf{pk}}_i := g^{\overline{\mathsf{sk}}_i}\right\}$$

- Post (e_i, σ_i) on the blockchain.

Figure 6: Multi-Party Computation $\Pi^{t,n}_{\mathsf{TDec}}$

The following algorithm describes the decryption procedure performed by the recipient (Fig. 7).

Decryption by Recipient

Setup: Each party P_i has a partial decryption key pair $(\overline{\mathsf{pk}}_i, \overline{\mathsf{sk}}_i)$, recipient has a public and private hybrid keys sk and pk

Input: Encryption hybrid public key c_1 and the recipient's public key pk.

Recipient: The recipient holds $C = (c_1, u)$ and does the following:

- Fetch $\{(e_i, \sigma_i)\}_{i\in[n]}$ from the blockchain, and decrypt $D_i \leftarrow \mathsf{HE.Dec}_{\mathsf{sk}}(e_i)$, $i \in [n]$;
- For $i \in [n]$, check the correctness of D_i by verifying the NIZK proofs σ_i.
- Compute $D := \prod_{i\in\mathcal{I}}(D_i)^{\gamma_i}$ and $k \leftarrow \mathsf{hash}(D)$, where $\mathcal{I} \subseteq [n]$, $|\mathcal{I}| = t+1$ and $\gamma_i := \prod_{\ell\in\mathcal{I}\setminus\{i\}} \frac{\ell}{\ell - i}$;
- Decrypt $m := \mathsf{AES\text{-}GCM}^{-1}_k(u)$;

Figure 7: Decryption by Recipient $\Pi^{t,n}_{\mathsf{TDec}}$

18.1.6 Handover Protocol

Suppose a committee $\mathcal{P} := \{P_1, \ldots, P_n\}$ jointly hold the secret key sk for the public key pk in the $(t+1, n)$-Shamir secret sharing form. They want to handover the secret key sk to the new committee $\mathcal{S} := \{S_1, \ldots, S_m\}$ in the $(k+1, m)$-Shamir secret sharing form for any $k < m$. As described in (Fig. 8), the handover protocol let each committee member $P_i \in \mathcal{P}$ to share its secret key share to everyone in committee $\mathcal{S}$ using Feldman's VSS. Each new committee member then combines the received shares to obtain its final secret key share.

The Handover Protocol $\Pi^{t,n,k,m}_{\mathsf{handover}}$

Setup: Each old committee member $P_i \in \mathcal{P}$ holds a partial decryption key pair $(\overline{\mathsf{pk}}_i, \overline{\mathsf{sk}}_i)$. Each new committee member $S_j \in \mathcal{S}$ holds a pair of public key for PKE $(\hat{\mathsf{pk}}_j, \hat{\mathsf{sk}}_j)$. Here, $\{\overline{\mathsf{pk}}_i\}_{i\in[n]}$ and $\{\hat{\mathsf{pk}}_j\}_{j\in[m]}$ are publicly known.

Old committee members: Each party $P_i \in \mathcal{P}$ does the following:

- Pick random $a_{i,1}, \ldots, a_{i,k} \leftarrow \mathbb{Z}_q$ and set $a_{i,0} := \overline{\mathsf{sk}}_i$.
- Define a polynomial $f_i(x) := \sum_{\ell=0}^{k} a_{i,\ell} x^\ell$.
- For $\ell \in \{1, \ldots, k\}$, post $E_{i,\ell} := g^{a_{i,\ell}}$ on the blockchain.
- For every $S_j \in \mathcal{S}$:
  - Compute $s_{i,j} := f_i(j)$;
  - Post $e_{i,j} \leftarrow \mathsf{HE.Enc}_{\hat{\mathsf{pk}}_j}(s_{i,j})$ on the blockchain.

New committee members: Each party $S_j \in \mathcal{S}$ does the following:

- Fetch $\{e_{i,j}\}_{i\in[n]}$ from the blockchain, and use $\hat{\mathsf{sk}}_j$ to decrypt them, obtaining the corresponding shares $\{s_{i,j}\}_{i\in[n]}$.
- For $i \in [n]$, check if $g^{s_{i,j}} = \prod_{\ell=0}^{k} (E_{i,\ell})^{i^\ell}$, where $E_{i,0} = \overline{\mathsf{pk}}_i$. If not, post complain against P_i by revealing the evidence: $s_{i,j}$ and

  $$\pi \leftarrow \mathsf{NIZK}\left\{((s_{i,j}, \hat{\mathsf{pk}}_j, e_{i,j}), (\hat{\mathsf{sk}}_j)) : s_{i,j} = \mathsf{HE.Dec}_{\hat{\mathsf{sk}}_j}(e_{i,j}) \ \wedge \ (\hat{\mathsf{pk}}_j, \hat{\mathsf{sk}}_j) \in \mathcal{R}_{\mathrm{PKE}}\right\} \text{ (cf. Fig. 4)}$$

- (One valid complain against $P_i \in \mathcal{P}$ will disqualify P_i.)
- Set the new partial public key as $\tilde{\mathsf{pk}}_j := \prod_{i\in\mathcal{I}} \prod_{\ell=0}^{k} (E_{i,\ell})^{i^\ell \cdot \gamma_i}$, where $\mathcal{I} \subseteq [n]$, $|\mathcal{I}| = t+1$ and $\gamma_i := \prod_{\ell\in\mathcal{I}\setminus\{i\}} \frac{\ell}{\ell - i}$;
- Set the new partial secret key as $\tilde{\mathsf{sk}}_j := \sum_{i\in\mathcal{I}} s_{i,j} \cdot \gamma_i$, where $\mathcal{I} \subseteq [n]$, $|\mathcal{I}| = t+1$ and $\gamma_i := \prod_{\ell\in\mathcal{I}\setminus\{i\}} \frac{\ell}{\ell - i}$;
- Return $(\tilde{\mathsf{pk}}_j, \tilde{\mathsf{sk}}_j)$.

Figure 8: The handover protocol $\Pi^{t,n,k,m}_{\mathsf{handover}}$

Appendix

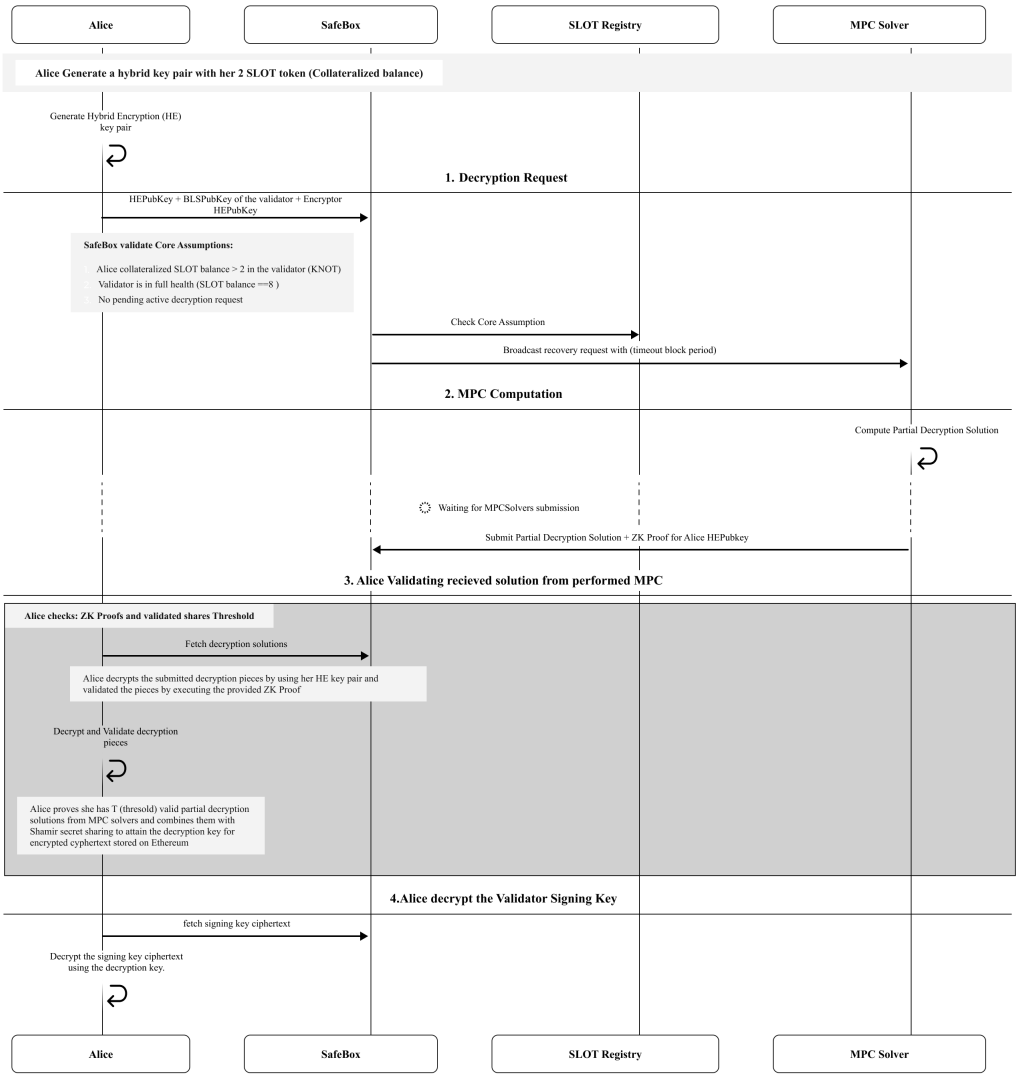

Figure 9: Signing key recovery using CIP MPC

19.2 Subgraph queries

```
{
  cipGlobalValues(first: 5) {
    id
    SafeBox
    solverPass
    numberOfInitialGuardians
    totalNumberOfEncryptions
```

```
    totalNumberOfDecryptionRequests
  }
}

Guaridan queries:

{
  guardians(first: 10) {
    id
    aesPublicKey
    guardianIndexPointer
    relinquished
    initial
    sharedPublicKey
  }
}

Encryptions:

{
  encryptions(first: 10) {
    id
    ciphertext
    encryptorAesKey
    encryptor
    native
  }
}
```

19.3 Glossary

The following table (Tab. 8) summarizes the notation used for cryptographic credentials used in the CIP operations:

Table 8: Description for notation used in the CIP procedures

Symbol	Description	Actor
$\overline{sk_i}$	Shared hybrid secret belonging to the solver i	MPC Solver
$\overline{pk_i}$	Shared hybrid public key belonging to the solver i	MPC Solver
$\overline{PK}$	Master public key resulting from the DKG procedure	Recovery requester
sk_i	Personal hybrid secret key belonging to the solver i	MPC Solver
pk_i	Personal hybrid public key of the solver belonging to the solver i	MPC Solver
u	Ciphertext hiding the BLS signing key	Recovery requester
h	Hybrid public key used to create the ciphertext of the BLS signing key	MPC solver
τ	Hybrid secret key used to create the ciphertext of the BLS signing key	MPC solver
k	Hybrid secret key assembled from Shamir secret sharing that is used to unlock the ciphertext u	Recovery requester
m	BLS signing key	Recovery requester
sk_r	Personal requester hybrid secret key	Recovery requester
pk_r	Personal requester hybrid public key	Recovery Requester

References

[1] APIs for a vibrant decentralized future. URL: https://thegraph.com/en/.

[2] Blockswap Stakehouse 2nd audit by Runtime Verification. URL: https://github.com/runtimeverification/publications/blob/main/reports/smart-contracts/Blockswap_Stakehouse_2nd_Audit.pdf.

[3] Dan Boneh et al. Bls signature scheme. Tech. rep. Technical Report draft-boneh-bls-signature-00, Internet Engineering Task Force, 2019.

[4] Common Interest Protocol Subgraph. URL: https://thegraph.com/hosted-service/subgraph/stakehouse-dev/common-interest-protocol.

[5] Vincent Almeida Derek Rickert Matt Shams. Stakehouse Protocol and Multichain ETH. https://github.com/stakehouse-dev/papers/blob/main/position_paper.pdf. Blockswap Labs, 2022.

[6] Ethereum. Ethereum Improvement Proposal 1559. May 2022. URL: https://github.com/ethereum/EIPs/blob/master/EIPS/eip-1559.md.

[7] Extended overview of Ethereum 2.0 keys. URL: https://kb.beaconcha.in/ethereum-2-keys.

[8] Ethereum Foundation. Consensus Specification Ethereum. May 2022. URL: https://github.com/ethereum/consensus-specs/blob/dev/specs/phase0/beacon-chain.md.

[9] Rosario Gennaro et al. "Secure distributed key generation for discrete-log based cryptosystems". In: Journal of Cryptology 20.1 (2007), pp. 51–83.

[10] Yulin Liu et al. "Empirical Analysis of EIP-1559: Transaction Fees, Waiting Time, and Consensus Security". In: arXiv preprint arXiv:2201.05574 (2022).

[11] Daejun Park, Yi Zhang, and Grigore Rosu. "End-to-end formal verification of Ethereum 2.0 deposit smart contract". In: International Conference on Computer Aided Verification. Springer. 2020, pp. 151–164.

[12] Torben Pryds Pedersen. "Non-interactive and information-theoretic secure verifiable secret sharing". In: Annual international cryptology conference. Springer. 1991, pp. 129–140.

[13] Proof of stake FAQ. URL: https://vitalik.ca/general/2017/12/31/pos_faq.html.

[14] RANDAO. URL: https://eth2.incessant.ink/book/06__building-blocks/02__randomness.html#enter-randao.

[15] Leonid Logvinov (@LogvinovLeon) Remco Bloemen (@Recmo). EIP-712: Typed structured data hashing and Sept. 2017. URL: https://eips.ethereum.org/EIPS/eip-712.

[16] Tim Roughgarden. "Intrinsic robustness of the price of anarchy". In: Journal of the ACM (JACM) 62.5 (2015), pp. 1–42.

[17] Tim Roughgarden. "Transaction fee mechanism design for the Ethereum blockchain: An economic analysis of EIP-1559". In: arXiv preprint arXiv:2012.00854 (2020).

[18] Standard curve database (P-256). URL: https://neuromancer.sk/std/nist/P-256#.

[19] Justin Zal. Numerical Malicious Committee Takeover Simulation. URL: https://gist.github.com/JustinZal/08b8d216a131090f22d2b5378e7abd53.