Confidential Data Rails

Secure, Confidential, Programmable Transfer of Data

Version 0.11 - November 2025

Ramtin M. Seraj
ramtin.seraj@piplabs.xyz

Wenxing Duan
wenxing.duan@piplabs.xyz

Hansol Lee
hansol.lee@piplabs.xyz

Steven Wang
steven.wang@piplabs.xyz

Jongwon Park
jongwon@piplabs.xyz

Hao (Leo) Chen
leo@piplabs.xyz

Abstract

We introduce *Confidential Data Rails (CDR)*, a secure and programmable on-chain storage and transfer protocol for private data. This novel core functionality of Story allows users to securely upload confidential data to the network and define on-chain access conditions. The data then becomes automatically accessible to other users who meet those conditions. The CDR protocol is powered by a decentralized collection of trusted execution environments (TEEs) that operate off-chain and are managed and synchronized by a smart contract on Story. Built on the same secure foundation of Story, this architecture ensures complete data confidentiality while offering flexibility through programmable access control and decentralized infrastructure. IP Vault is one of the first applications of this powerful protocol. It allows IP owners to securely attach confidential data to their registered on-chain IP assets, data that becomes automatically accessible to license holders without further IP owner involvement. IP Vault combined with Story's existing powerful infrastructure for IP, streamlines the entire journey of IP assets, from registration and licensing to monetization and automatic confidential data delivery. This integrated ecosystem makes IP assets truly programmable and creates an open, composable marketplace for IP data assets (AI datasets, API keys, ...) with privacy preserved throughout the lifecycle.

1 Introduction

Earlier this year, **Story**[1] was launched as a scalable layer 1 blockchain designed to help creators register, manage and monetize their intellectual property (IP), such as art, music, data and AI models, as programmable assets on the chain.

Story uses a novel *multi-core* architecture where a main EVM-compatible core automatically triggers a collection of *specialized cores* for enhanced performance. For instance, the *IP Core*, the first specialized core on Story, handles IP registration, licensing, and tracking derivative works through large and complex IP webs with thousands of connections. It

achieves this using natively-supported graph data structures and high-performance traversal algorithms for efficient queries. Using Story, creators and IP owners can register their IP assets on-chain and use the *Programmable IP License (PIL)*[2] framework to set programmable licenses and automate royalty payments with legal bindings. The release of Story unlocks IP as a new class of programmable on-chain asset, gaining significant attention from ecosystem applications that register various forms of IP on-chain.

However, a challenge remained in the end-to-end journey of IP assets: How can IP's data be transferred confidentially and securely from the owner to the license holder? IP data spans a wide range of formats and sizes, personal or business data for AI training, API keys for Web2 platforms, digital records from artists and creators, or even keys to external accounts.

Although many decentralized storage solutions (IPFS[3], Walrus[4], Shelby[5], etc.) have recently gained attention, most of these solutions focus primarily on data availability and liveness guarantees through cost-effective replication. They either do not address critical confidentiality requirements or require complex interactions with secondary protocols, which results in fragmented user experience and inefficiency. These challenges inspired us to design the Confidential Data Rails protocol, a novel solution that unlocks new applications.

1.1 Problem Definition

The goal is to build a unified system that securely, efficiently, and confidentially facilitates the transfer of data between users that satisfy the following properties:

Security and Reliability It must provide the same level of security and decentralization as the other on-chain asset. Thus, it must be operated by a decentralized set of participants N, maintaining liveness and security even if up to one-third of participants ($N/3$) are offline or act malicious.

Confidentiality and Privacy The data must remain confidential and protected from everyone including protocol participants except the data owner and authorized users. It must maintain privacy even when a certain ratio (t) of protocol participants act maliciously and/or collude. For this setup, we consider a reconstruction threshold $t + 1$ and validity of $N > 2t + 1$ to guarantee liveness.

Note that the requirements only cover secure storage and transfer of data. Malicious actions by a valid recipient, such as leaking data post-transfer, are outside the scope of this work. Other methods, such as cryptographic fingerprinting[6] and on-chain incentives, can be used to identify and adjudicate such actions as an additional safeguard.

Scalability The design must be scalable and support various forms and any size of data. This includes AI training data, digital assets, biomedical data, and more. Scalability also means practical affordability, making it financially feasible to store data. If storage costs grow exponentially with the size of the data, the system is not truly scalable.

Functionality The system should support *Dynamic Access Control* and Programmability. Dynamic Access Control enables automated data access based on-chain logic (e.g. IP

ownership and licensing), eliminating the need for direct interactions between users. The system adapts to the evolving set of authorized consumers, which is not known in advance and changes over time based on on-chain state. The system should also natively support *programmability*, so that the data provider can require a wide range of on-chain conditions. This covers a wide range from basic ones like time-limited access to more advanced capabilities where the data provider can specify which computational environments can access the data and enforce usage restrictions. This is a powerful property that unlocks many forms of applications (more on this topic later in this chapter).

Usability The user experience is a critical factor that is often overlooked when designing decentralized systems. The system has to be designed for a seamless user experience. It has to minimize fragmented user experience, where a user has to go to multiple providers setup with different payment methods and has to often tolerate huge delays when these providers have to synchronize and settle.

1.2 Confidential Data Rails

We propose **Confidential Data Rails (CDR)**, a new core in Story's multi-core architecture. A CDR is a secure, confidential, on-chain storage space that can be attached to any on-chain asset or smart contract. Each CDR is only accessible by users who satisfy the conditions set initially. Whenever a user satisfy the on-chain condition (e.g. acquiring a license to an IP asset), they can automatically access the CDR without requiring further involvement from the initial data uploader.

An example application of the CDR protocol on Story is IP Vault. IP Vault is an instance of CDR that restricts write access to the IP owner and read access to license holders. On Story, each *IP Asset* is an ERC-721 (NFT). The owner of this NFT (IP Owner) has the power to setup the terms and conditions for other users to acquire a license to use the IP asset and can also attach confidential data to the IP Asset using IP Vault.

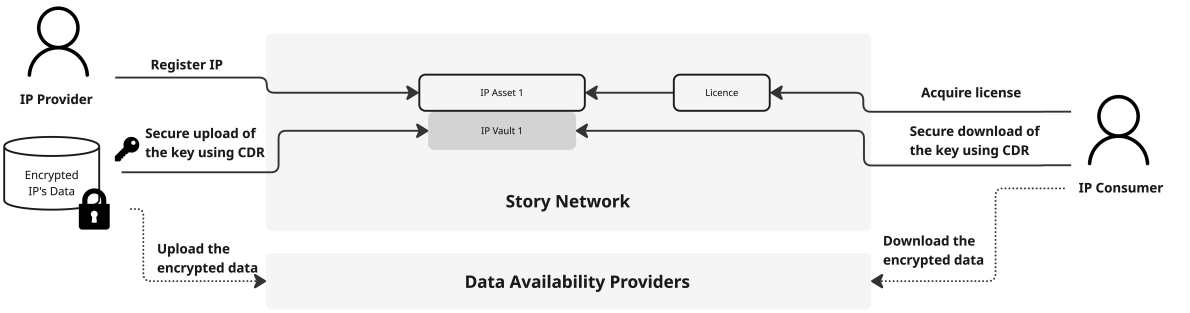

Figure 1: Secure and confidential transfer of assets between the IP provider and the IP consumer

Because on-chain data storage is often costly, the CDR protocol is designed to be compatible with existing data availability layers. Users can upload encrypted data to any of these providers and use CDR's space only for the encryption key. This makes the CDR protocol highly scalable and cost-effective while not relying on underlying data availability layers for data privacy.

Furthermore, the CDR protocol supports full on-chain programmability, including time-bounded access, time-locks, role-based access, and restrictions on data processing methods. This flexibility unlocks a wide range of applications. An interesting application is limiting the type of operations on data, where the data provider sets conditions that require other authorized users to provide remote attestation for a Trusted Execution Environment (TEE) loaded with a specific binary, exposing only the final result. This powerful concept can be used to restrict the types of operations allowed to be performed or prevent full access to the data. using this technique, AI data providers might require unlocking of their data only to a TEE where a model is trained and automatically registers the AI model as a new IP on Story with distributed ownership to the data providers and AI trainer. This allows for creating powerful AI models with shared ownership based on each party's contributions.

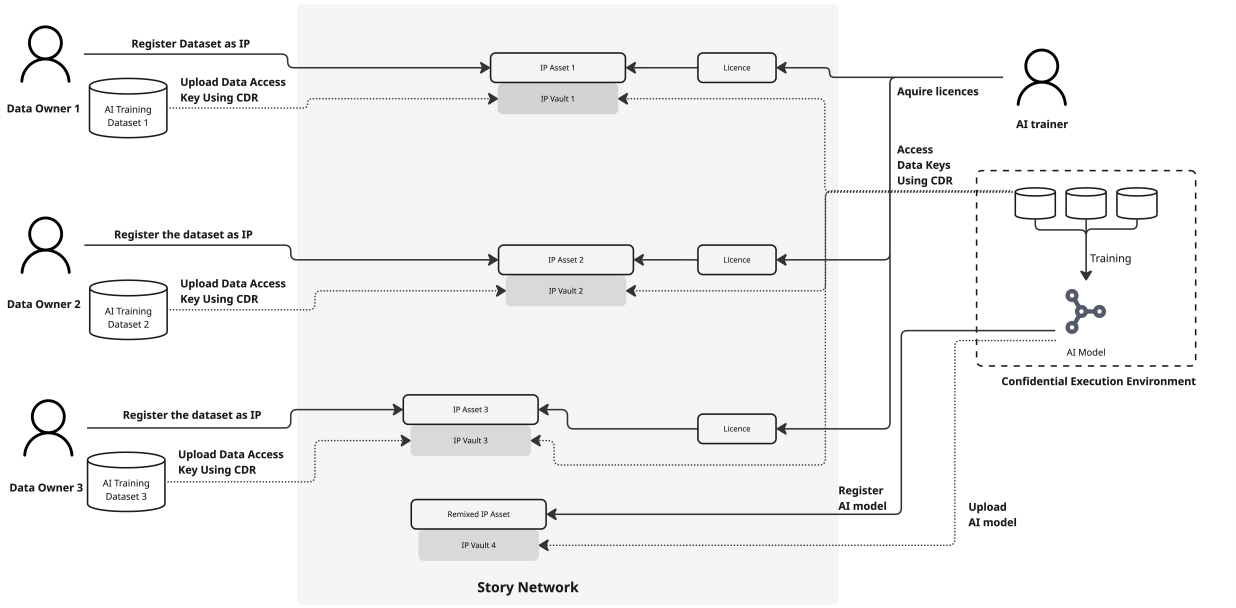

Figure 2: Data providers and model trainers participate to train and own a new AI model using Story and Confidential Data Rails

Similarly, the data access can be limited to other privacy-preserving verifiable execution environment (e.g. private smart contract). For instance, an AI model can limit the user's access to inference results only while preserving both the user's input data and the model parameters. These are just some examples of how data and computation can work together, with the CDR protocol serving as the building blocks that power protocols like Agent TCP/IP [7].

2 Architecture Overview

The Confidential Data Rails protocol's architecture consists of three components: a committee of participants (CDR Committee), a collection of Trusted Execution Environments (TEEs) running specific binaries (kernels), and a smart contract deployed on Story (CDR's core contract).

Upon joining, each committee member must spin up and register a TEE instance loaded with a specific binary. We call this instance a *kernel*. Kernels collectively execute cryptographic protocols such as Distributed Key Generation (DKG) and Threshold Decryption to securely manage and utilize encryption keys without exposing private key material to any single entity. At the heart of this design, the CDR's core contract manages a wide-range of on-chain responsibilities. This smart contract orchestrates the kernels to jointly follow the steps of DKG. this process generates a public/private key pair without any single one of them ever knowing the full private key. Instead, each participant ends up with a share of the secret key, and a threshold of participants is required to reconstruct or use it.

With this foundation in place, the core contract accepts user requests to upload or access CDRs. Before accepting upload requests, the contract assigns a unique ID to the request. This ID is later used for encryption of the data. After uploading the encrypted data and setting the access conditions, the data is stored on-chain and indexed by the unique ID. Upon receiving a download request, it validates on-chain conditions set by the data uploader. If all conditions are satisfied, it signals the kernels (through events) to perform distributed threshold decryption and decode the content for the recipient.

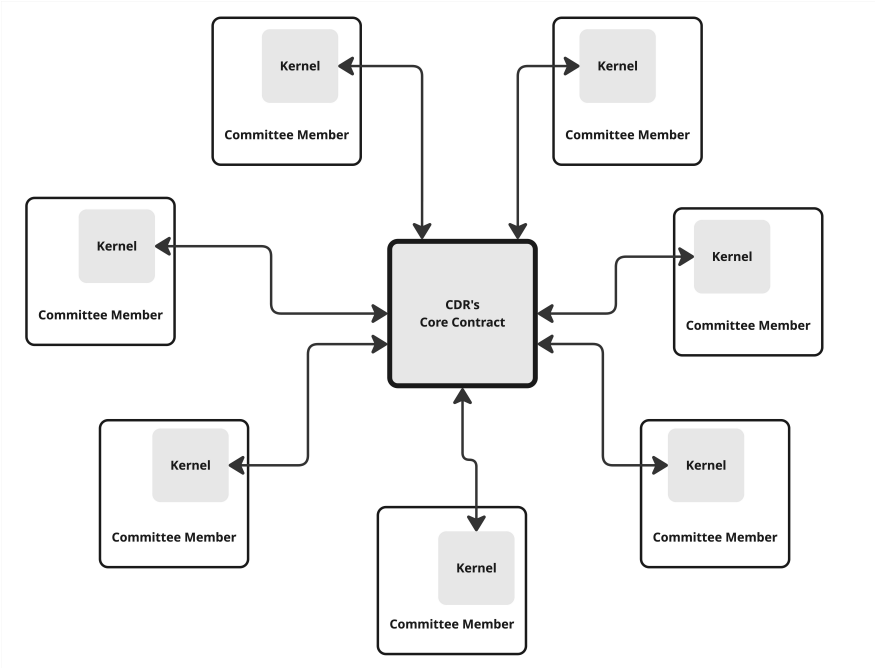

Figure 3: Confidential Data Rails High-Level Architecture Overview

2.1 Core Components

In this section, we examine the architecture's core components in detail.

2.1.1 CDR Committee

The CDR committee consists of Story node operators (validators) who have opted-in to participate in the protocol. They receive additional rewards based on their participation and performance, and risk having their stakes slashed if they act maliciously or become inactive for a long period of time.

The main responsibility of a committee member is to run kernel and facilitate its communication with other kernels and the core contract. The kernels only operate when they receive ordered events (logs) from the core contract or when they broadcast messages from other kernels. Committee members cannot access the internal state of the kernels or request arbitrary actions from them. Thus, the committee members act only as intermediaries, forwarding events from the core contract (along with inclusion proofs) and submitting results back to the core contract. They need active participation to receive rewards and receive penalties for not participating or submitting invalid data. As part of future improvements to this protocol, in addition to penalties for inactivity, cryptographic methods are used to identify whether a secret share stored in a kernel has been leaked or used maliciously. If such a case is reported, the validator's stake is at risk (checkout section 3 for more details).

Rewards come from two sources: CDR usage fees and Story's UBI (Universal Basic Income) pool. UBI is a fund of newly minted Story tokens that incentivizes validators to provide new network functionalities. Initially, UBI helps cover validator rewards and TEE-friendly server costs. Over time, CDR usage fees will fully fund these rewards.

The CDR protocol operates in *epochs*. In every n block, the committee can change and secret shares of the public key are updated as a security measure. Story validators who participate in the CDR protocol cannot remove themselves from the Story network until they are completely off-boarded.

2.1.2 Kernels

As mentioned earlier, a kernel is a Trusted Execution Environment (TEE) instance loaded with a specific binary. A TEE is a protected execution environment inside a CPU (e.g., Intel TDX, AMD SEV, ARM TrustZone) that isolates code and data from the rest of the system, including the OS and hypervisor. Its memory is encrypted and accessible only to the specified binary. If the binary is tampered with, the memory becomes inaccessible. To verify that a kernel is healthy and untampered, these environments provide *remote attestation*, a digitally signed report that proves the integrity and authenticity of the environment. An attestation report is typically signed with a key rooted in hardware (e.g., Intel's attestation keys), and a remote verifier checks this against a trusted certificate authority (e.g., Intel Attestation Service). The CDR protocol requires that each kernel's remote attestation report includes a unique identity, the cryptographic hash of the initial code and data, and a public key for

authenticated and encrypted communication with the smart contract and other kernels. The CDR's core contract uses decentralized on-chain protocols such as Automata DCAP [8] to validate the authenticity of the signature.

The kernel stores secret shares and consumes events from the core contract to update its internal state and produce results that are sent to the smart contract and other kernels. Committee members redirect finalized Story block headers and core contract events to their kernels. Kernels do not trust committee members and construct their own confidential and authenticated communication channel with the smart contract. This is accomplished through several components:

First, every piece of output is signed by the communication key of the kernel. This key was reported as part of the remote attestation and kernel registry and is used by the core contract and other kernels to validate the authenticity of the message and sometimes for confidential async transfer of messages between kernels (used during DKG).

Second, every log (event) received by the kernel has to be received along with event inclusion proofs. To be able to validate these events, the kernel uses the initialization data (committed in remote attestation) and runs a light client internally to validate block header hashes and signatures, and follows Story's finalized canonical chain. By following the block headers and using Merkle root of logs (events), kernels can verify inclusion proofs for events taking any action. The smart contract assigns to each log (event) a sequence number, ensuring that the events are consumed in order.

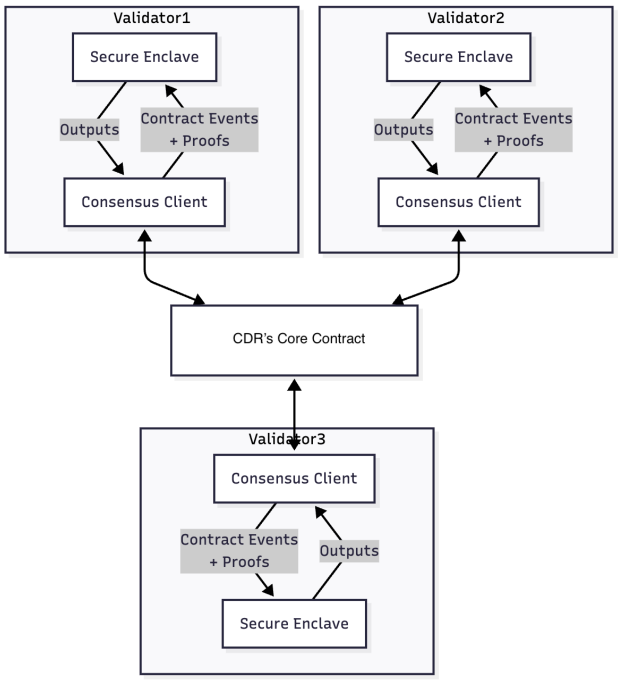

Figure 4: Kernels are synchronized through core contract order logs (events)

Although TEE promises no direct access to secret shares by committee members, the safety of the CDR protocol does not depend on a single TEE instance. A successful attack

would require compromising a threshold majority of these environments during the same epoch. To minimize this risk, the protocol incentivizes committee members to use diverse, state-of-the-art TEE architectures from various hardware providers. This prevents a majority of participants from using the same hardware. If issues are discovered with a TEE architecture (e.g., Intel SGX), the majority will not be compromised, system confidentiality will remain intact, and the kernel switch process can be triggered. Additionally, the CDR protocol does not assume that these environments are secure against side-channel attacks. Therefore, the binary is implemented using constant-time cryptography with secret-independent memory access and branching.

2.1.3 The Core Contract

Deployed on Story, **the core contract** plays several roles in the CDR protocol, including:

- On-boarding and off-boarding validators joining the committee and managing epoch timing and kernel switch process (more on this later)
- Serving as the source of truth for the kernel binary hash and initialization data, and verifying remote attestations provided by committee members based on these data
- Managing the distributed key generation (DKG) process and serving as the source of truth for its state (tracking commitments) and outcome
- Managing CDR upload and download requests and enforcing access conditions (e.g. IP ownership and valid license possession)
- Collecting fees and distributing rewards based on committee member participation
- Adjudication of reported violations and slashing the stakes of malicious members

The contract can be upgraded through Story's network governance process.

2.2 Process Overview

This section explains how these components work together and outlines the main processes. We keep it high-level for clarity. For more technical details, please refer to the cryptographic primitives section.

2.2.1 Distributed Key Generation

The goal of this process is for the kernels to participate in the Distributed Key Generation (DKG) and jointly construct one or several cryptographic key pair(s). For this goal, the smart contract acts as the source of truth, registers participants, and synchronizes the major steps of Pedersen DKG. Pedersen DKG is a verifiable distributed key generation protocol based on Pedersen's verifiable secret sharing (VSS). We set the parameters so that more than 50% of the participants ($t = N/2$) must participate to use the private key. It ensures:

- No single party knows the private key.
- Participants can verify that all shares are consistent.
- The public key is jointly computed and agreed on.

This process is done in phases, each lasting several blocks (dynamically adjusted by the smart contract if needed). The start of each phase is emitted as a signal by the smart contract.

Phase 1 - Signup
The process begins when the core contract announces the sign-up phase. During this phase, validators submit their requests to join the committee and provide remote attestations of their kernels. During sign-up, the smart contract validates remote attestations using on-chain verifiers. For large committees, this process can be optimized using an optimistic structure: proofs are received, but only verified if challenged within a specific time window. If challenged, on-chain verification is executed. Based on the results, the validator may be penalized and the challenger may receive rewards. At the end of this phase, the set of participants and other parameters (e.g. threshold, ...) are emitted as events and would be known to everyone.

Phase 2 - Secret Construction
This phase starts when the validator redirects the setup events to the kernel. Each kernel generates two random polynomials, one for the secret and one for hiding and verification. Each kernel then computes commitments to their polynomial coefficients and returns the results to the validator to be uploaded to the core contract and received by the other validators. These commitments allow others to verify that their shares are valid later.

Phase 3 - Deal Propagation and Verification
Next, each kernel prepares a set of secret shares (aka deals) and privately sends one share to other participants. This is done by encrypting the shares and transferring them to the final recipient through a secure channel constructed using other kernels' DKG public keys. The transmission of these data is done by the validator and does not need to be uploaded to the smart contract for performance reasons. During this phase, each kernel validates the deals it receives from other kernels using the commitments originally submitted to the smart contract. If a deal is found invalid, the kernel can submit a complaint to the smart contract and raise a challenge. At the end of the phase, the set of qualified kernels is confirmed and any challenges have been addressed.

Phase 4 - Finalization
In this phase, each qualified kernel combines valid shares to construct their own final secret and construct the DKG public key. Each qualified kernel then has to report their calculated public key to smart contract as a signal to be ready and done with DKG steps. At this phase all the public keys have to match and this is an extra step to make sure that all the kernels have agreement over the public key.

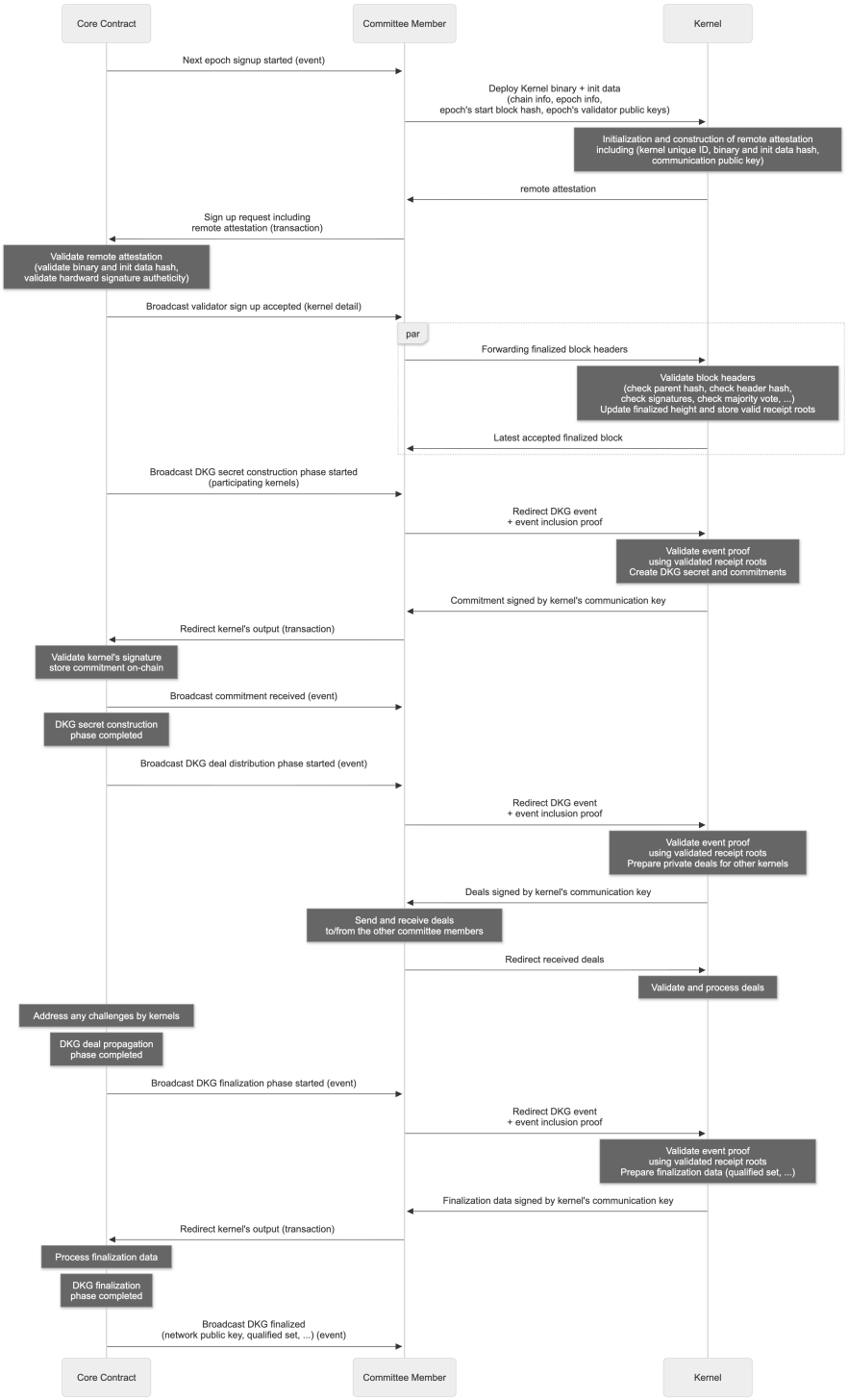

Figure 5: DKG process

2.2.2 Kernel Rotation and Proactive Refresh

This process can be triggered when any of these conditions occurs:

- The epoch is over (the specific number of blocks has passed)
- The number of active committee members are dropping closer to the liveness threshold, we need to adjust the committee members.
- The kernel binary needs to be updated following the network governance process.
- A vulnerability is identified in one type of TEE architecture, requiring the migration of the affected kernels to a different architecture. (re-balancing diversification of TEE architectures)

During the kernel rotation process, the active set of kernels is replaced with a new set of kernels; thus, validators can join or leave the committee by sending their request to the core contract. The new committee members have to provide remote attestation of their kernel, and the core contract validates remote attestations and ensures that the validator has enough stake. During the time that a validator serves as a committee member, it cannot request the release of its stakes.

In addition, during this process, a proactive share refresh occurs between old and new committee members. This changes the secret shares for all members, including existing ones, without changing the overall secret. Periodically updating the' secret shares of participants is an additional security measure that improves the long-term security of the CDR protocol.

2.2.3 Secure Data Transfer

After successful DKG finalization, the core contract is ready to accept requests to upload and download the content of the Confidential Data Rails. To securely upload data to a CDR, the data provider first requests a unique ID from the core contract. Then it generates and uses a symmetric *data key* and encrypts the data using data key. It can use any choice of data availability layer for uploading the encrypted data. Then the data provider encrypts the data key using the DKG's public key and received unique ID and submits it along side access conditions to the core contract. The contract validates whether the submitter is the initial receiver of the unique ID before storing the content on-chain.

For encryption and threshold decryption, the CDR protocol uses a slightly modified version of TDH2[9] where the message encryption of the original paper (hashing the key and XOR-ing the results with the message) is replaced with standard AES-GCM. This allows us to use unique ID as additional authenticated data, where we can prevent attacks such as replay attacks.

After successful data upload, any user that can satisfy access condition, can send a request to the core contract to access the data. As part of this request, the user reports a public key that is going to be used for secure transfer of confidential data to the user. After on-chain validation of conditions, the core contract emits an event and signals Kernels

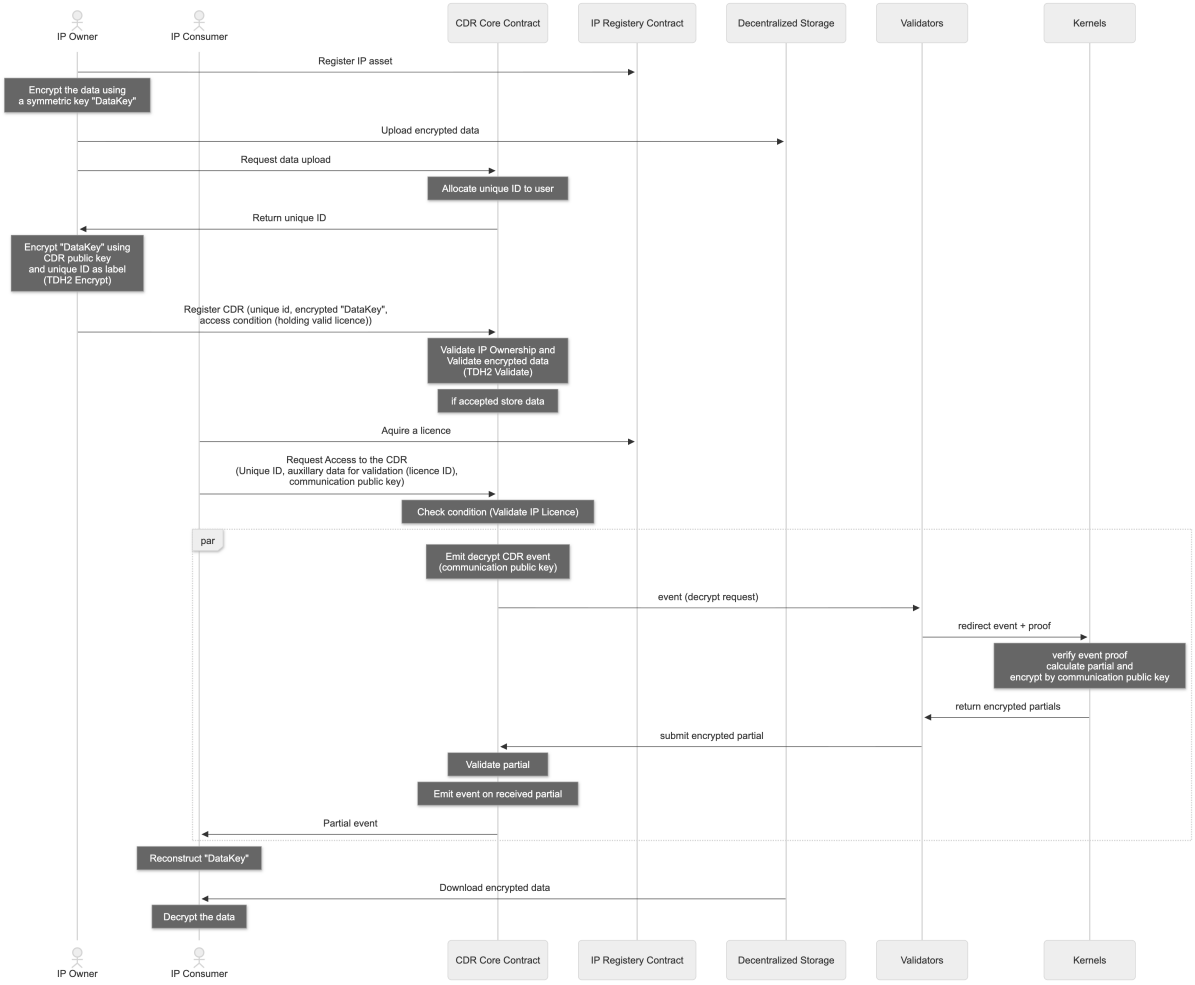

Figure 6: Secure transfer of data between IP owner and IP consumer using Confidential Data Rails protocol

to participate and execute threshold decryption for that specific user. Following the signal kernels, the partial decryption shares are prepared, encrypted, and submitted to the core contract for validation. When the threshold of kernels uploads their shares, the user can collect the decryption shares and reconstruct the data of the CDR.

Note that for simplicity we only mentioned a single DKG key in this document, in practice to minimize the risk from time to time, a new set of DKG public keys can be generated and used by the same set of committee members to secure different group of assets.

2.3 Security Considerations

This section summarizes attack vectors and explains how Confidential Data Rails' architecture prevents or mitigates them.

2.3.1 Privacy Attacks by Malicious Committee

A major risk in any threshold decryption system is secret leakage or malicious use of secrets by participating members. A key challenge is that colluding parties could decrypt encrypted data without leaving any trace to be identified. Although we assume that no more than one-third of the committee will act maliciously and require half of the committee to collude to reveal the information, we believe that multiple layers of defense are needed to make such attacks extremely difficult.

The first layer of defense prevents the committee from directly accessing secrets or running operations on them. This is achieved using Kernels; isolated state machines running inside TEEs that only act when smart contract events are provided with proofs. As discussed previously, the kernel is designed to remove direct access to secrets by committee members and prevent committee members from being able to trigger unauthorized malicious actions. Also, every output of the kernel is signed by its communication key and can be authenticated on-chain, which makes the committee member unable to fabricate an output. The only malicious action by the committee member can be blocking communication between the kernel and the smart contract, which would result in a loss of reward for the committee member.

While the CDR protocol uses cryptographic implementations that support constant times, non-deterministic memory access, etc., TEEs are sometimes vulnerable (e.g., attacks on Intel SGX) and can be compromised from time to time (e.g. side-channel attacks). Although such attacks are often expensive and time-consuming and might not be feasible to carry out during the life cycle of a kernel (during an epoch), the CDR protocol does not rely on a single TEE's guaranty for protocol safety. An attack would require breaking a threshold number of TEE environments. The use of frequent kernel switch processes (epochs) and proactive secret refresh combined with the use of diversified TEE architecture makes it really hard to gain enough secrets to be able to attack during an epoch. The old kernel after the epoch switch destroys its secret share and becomes inactive. If a vulnerability is identified in a TEE architecture at any time, the kernel switch process is triggered, and the protocol prevents use of the vulnerable environment.

Additionally, in the upcoming updates, an extra layer of protection will be introduced to identify leaks or misuse of shares, detect malicious parties, and penalize them, since all committee members are staked. Recent state of the art techniques allow to leave identifiable traces on secrets that are tamper proof (see section 3 for more details).

2.3.2 Chosen Cipher Text Attack

A Chosen Ciphertext Attack (CCA) is a cryptographic attack in which an adversary can obtain decryptions of chosen ciphertexts and use this information to break the encryption

scheme or recover secret keys. A common attack in ElGamal-based threshold decryption systems exploits the scheme's homomorphic properties. The attacker alters a user's uploaded encrypted data and submits it. TDH2 strengthens ElGamal-style threshold decryption by using zero-knowledge proofs and hash functions. This allows partial decryptions to be verified without exposing secret key shares. TDH2 is secure under Decisional Diffie-Hellman (DDH) in the random oracle model, and the threshold decryption scheme is CCA2 secure. This ensures the protocol is secure even if an attacker can obtain decryptions of arbitrary ciphertexts, except the one they are trying to break.

2.3.3 Replay Attack

In this particular setup and scenario, a malicious attacker might carefully observe and monitor encrypted data as it is being submitted to the network, and then strategically front-run the legitimate submission by uploading the exact same encrypted data before the original submitter completes their transaction. By doing this, the attacker could falsely claim ownership and authorship of those data and subsequently, at a later point in time, submit a request to have those data decrypted, thereby gaining unauthorized access to information they do not rightfully own.

To address and mitigate this vulnerability, the CDR protocol implements a variation of the TDH2 (Threshold Decryption of Hybrid ciphertexts) scheme that specifically replaces the enclosed symmetric encryption component with AES-GCM (Advanced Encryption Standard in Galois/Counter Mode). Within this implementation, the allocated unique ID is incorporated as additional authenticated data (AAD) in the AES-GCM encryption process. The core contract is responsible for verifying that the entity making the upload is indeed the initial requester, and captures and stores the unique allocated ID alongside the encrypted data for subsequent use during the threshold decryption process performed by committee members. In cases where an invalid or incorrect AAD is provided or used during the decryption attempt, the decryption process will fail and produce an authentication error, thus preventing unauthorized access to protected data.

2.3.4 Liveness Attack

Given that our system operates on the basis of a threshold of participation among committee members to maintain operational functionality, the overall liveness and availability of the system becomes vulnerable and is placed at risk in scenarios where more than the accepted threshold of the committee members experience downtime, go offline, or otherwise lose access to their respective kernel instances.

To effectively mitigate this potential risk and maintain system integrity, if the active committee membership falls below the required operational threshold, an automated kernel switch process is immediately triggered and initiated with a completely new setup configuration and infrastructure. Additionally, as part of this mitigation strategy, those committee members who have become inactive or not responding will receive appropriate penalties according to the protocol's enforcement mechanisms.

Note that in case of committee member TEE failure, if the committee member still has access to the same hardware, it can redeploy the kernel and continue the operations since kernel states are persistent. In case of failure that is not recoverable (e.g. change of hardware), the member has to wait until the end of the epoch and request to continue with a new TEE instance.

3 Upcoming Improvements

In previous sections, we outlined our multi-layered architecture designed to ensure the security of Confidential Data Rails. To further strengthen the foundation, we are already working on adding additional layers of defense to the protocol across DKG, runtime decryption, and TEE layers. At a high level, we are working accountable threshold operations, stronger bias resistance, explicit proofs of misbehavior in DKG, and post-quantum readiness, together with incentive updates aligned with these capabilities.

Share leakage / misuse detection and attribution. We are upgrading the protocol to add the ability to identify and punish committee members if we detect share leakage or malicious use of their share—including cases where a subset colludes off-protocol. This would act as an extra layer of protection. Concretely, we will integrate *traceable secret sharing* (TSS) so that a black-box "reconstruction oracle" can be traced to at least one leaking identity without exposing honest shares[10]. At DKG/refresh/reshare time, each Shamir share (x_i) will be wrapped with an identity-bound correlation tag (per-epoch) that (i) preserves standard interpolation for TDH2 and (ii) enables *black-box tracing* with a bounded number of challenge shares. If a leak is suspected, a tracer (attested enclave or quorum) will run the TSS procedure to output a succinct accusation artifact, tied to the on-chain transcript and epoch. To strengthen forensics beyond cryptography, Data owners may opt into *content/model watermarking* (robust canaries; neural watermarks) for post-transfer attribution, while TSS will cover pre-transfer validator misuse. We will evaluate efficient building blocks that keep overhead near-constant per share and remain compatible with our Shamir/TDH2 interface[11,12,13].

Identifying malicious actions during DKG. We are also updating the DKG so that malicious actions can be distinguished from simple inactivity and so that misbehavior yields succinct, reusable proofs. Concretely, we will design a *publicly-verifiable* path (PVSS→DKG) where commitments, encrypted deals, openings, and complaints carry lightweight NIZKs that the core contract can verify, following a Mt. Random discipline with modern YOLO YOSO primitives[11,13]. Each sent share will be accompanied by: (a) a consistency proof that it opens the dealer's commitments and (b) a proof that the ciphertext actually encrypts that opening to the recipient. Complaints will include signed transcripts; invalid complaints will be provably punishable. The same pattern will apply to proactive reshare: deals and challenges will produce on-chain verifiable evidence, enabling precise slashing and minimizing reliance on TEE honesty for *correctness*. DKG/reshare transcripts will be epoch- and identity-marked.

Bias resistance. We will harden the setup against delay-reveal and abort bias. Although kernels use sealed randomness and measured binaries, a member could still bias PK by strategically withholding openings. We will enforce strict *commit–then–open* schedules and derive reveal windows from unbiased, publicly verifiable beacons (VRF/VDF/class-group) anchored on-chain[11,12,13]. Aborts after the beacon will incur higher penalties; repeated aborts will trigger exclusion. Only constant terms (needed for PK) will be revealed in the end; all other coefficients remain information-theoretically hidden. Outputs and per-node fragments will be audit-marked by epoch for post-incident analysis.

Post-quantum readiness. We will pursue hybrid payload protection that encapsulates the symmetric key under both DL/DDH (TDH2) and MLWE KEMs in *AND*-mode (or policy-selected hybrid) without changing the data-path format: the recovered session key will be derived via HKDF over both decapsulations. In parallel, we will evaluate lattice-based DKG/decapsulation tracks: initially, the discrete-log track will run as today, while a PQ track (e.g., MLWE KEM with Shamir-style threshold decapsulation inside TEEs) will run in tandem, allowing epoch-wise migration. We will also adopt *signature agility* so that validator attestations can move to PQ signatures without disrupting decryption (dual-sign during transition).

Incentives and Proof of Cloud. We will revise incentives with graduated penalties backed by verifiable artifacts: (i) inactivity (time-outs) vs. (ii) bias-sensitive aborts vs. (iii) cryptographic misbehavior (invalid shares, false complaints) identified by on-chain checks. As an additional TEE safeguard, a threshold of kernels will periodically furnish *Proof of Cloud* (or equivalent)[14], attesting that enclaves run in physically secured, policy-compliant cloud environments. Combined with proactive refresh/reshare and multi-TEE diversity, this will raise the bar for at-scale share extraction while preserving auditability and performance.

Conclusion

In this paper, we proposed the Confidential Data Rails (CDR) protocol, a secure and decentralized framework for confidential on-chain data storage and transfer. This unlocks many applications on Story; for example, it enables the programmable transfer of encrypted data between IP owners and authorized license holders. We demonstrate that by using a multi-layer architecture, the CDR protocol preserves end-to-end confidentiality of data without sacrificing decentralization or usability. This is achieved through three pillars: decentralized key generation and threshold decryption, restricting participant access to secret shares through TEEs, and identifying share leakage and malicious actions with appropriate penalties.

Acknowledgments

The authors thank Bernardo David, Dimitris Karakostas, Andrea Muttoni, and Arash Afshar for valuable feedback on the early design of this work, and Raúl Martinez, Meng Li, Yao Wang, and Woojin Kim for their contributions to the first version of Confidential Data Rails.

References

[1] Story Foundation. Story: A peer-to-peer intellectual property network. https://www.story.foundation/whitepaper.pdf, 2024.

[2] Story Foundation. Story programmable ip license (pil) - a legal framework for ip licensing on-chain. https://docs.story.foundation/concepts/programmable-ip-license/overview, 2024.

[3] Yiannis Psaras and David Dias. The InterPlanetary file system and the Filecoin network. In *2020 50th Annual IEEE-IFIP International Conference on Dependable Systems and Networks-Supplemental Volume (DSN-S)*, page 80. IEEE, 2020.

[4] George Danezis, Giacomo Giuliari, Eleftherios Kokoris Kogias, Markus Legner, Jean-Pierre Smith, Alberto Sonnino, and Karl Wüst. Walrus: An efficient decentralized storage network, 2025.

[5] Guy Goren, Andrew Hariri, Timothy D. R. Hartley, Ravi Kappiyoor, Alexander Spiegelman, and David Zmick. Shelby: Decentralized storage designed to serve, 2025.

[6] D. Boneh and J. Shaw. Collusion-secure fingerprinting for digital data. *IEEE Transactions on Information Theory*, 44(5):1897–1905, 1998.

[7] Andrea Muttoni and Jason Zhao. Agent TCP/IP: An agent-to-agent transaction system. https://arxiv.org/pdf/2501.06243, 2025.

[8] Automata Team. Automata 2.0: the machine trust layer for verifiable computation. https://1690124894-files.gitbook.io/~/files/v0/b/gitbook-x-prod.appspot.com/o/spaces%2FtYKuUrKWPlgYjy0suCeT%2Fuploads%2FRkErQDfHOf1byYf75vb1%2Flightpaper-v2.pdf?alt=media&token=47a81c68-41c2-40fa-a156-b39e467a145f.

[9] Victor Shoup and Rosario Gennaro. Securing threshold cryptosystems against chosen ciphertext attack. *Journal of Cryptology*, 15(2):75–96, 2002.

[10] Dan Boneh, Aditi Partap, and Lior Rotem. Traceable secret sharing: Strong security and efficient constructions. In *Annual International Cryptology Conference*, pages 221–256. Springer, 2024.

[11] Ignacio Cascudo, Bernardo David, Omer Shlomovits, and Denis Varlakov. Mt. random: Multi-tiered randomness beacons. Cryptology ePrint Archive, Paper 2021/1096, 2021.

[12] Ignacio Cascudo and Bernardo David. Publicly verifiable secret sharing over class groups and applications to DKG and YOSO. Cryptology ePrint Archive, Paper 2023/1651, 2023.

[13] Ignacio Cascudo, Bernardo David, Lydia Garms, and Anders Konring. YOLO YOSO: Fast and simple encryption and secret sharing in the YOSO model. Cryptology ePrint Archive, Paper 2022/242, 2022.

[14] Proof Of Cloud. https://proofofcloud.org/.

[15] Torben Pryds Pedersen. A threshold cryptosystem without a trusted party. In *Workshop on the Theory and Application of of Cryptographic Techniques*, pages 522–526. Springer, 1991.

[16] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[17] Paul Feldman. A practical scheme for non-interactive verifiable secret sharing. In *28th Annual Symposium on Foundations of Computer Science (sfcs 1987)*, pages 427–438. IEEE, 1987.

[18] Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Annual international cryptology conference*, pages 129–140. Springer, 1991.

[19] Rosario Gennaro, Stanisław Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 295–310. Springer, 1999.

[20] John Canny and Stephen Sorkin. Practical large-scale distributed key generation. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 138–152. Springer, 2004.

[21] Aniket Kate, Gregory M Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In *International conference on the theory and application of cryptology and information security*, pages 177–194. Springer, 2010.

[22] Ewa Syta, Philipp Jovanovic, Eleftherios Kokoris Kogias, Nicolas Gailly, Linus Gasser, Ismail Khoffi, Michael J Fischer, and Bryan Ford. Scalable bias-resistant distributed randomness. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 444–460. Ieee, 2017.

[23] Timo Hanke, Mahnush Movahedi, and Dominic Williams. Dfinity technology overview series, consensus system. *arXiv preprint arXiv:1805.04548*, 2018.

[24] Ferran Alborch, Ramiro Martínez, and Paz Morillo. R-lwe-based distributed key generation and threshold decryption. *Mathematics*, 10(5):728, 2022.

[25] Kamil Doruk Gur, Jonathan Katz, and Tjerand Silde. Two-round threshold lattice-based signatures from threshold homomorphic encryption. In *International Conference on Post-Quantum Cryptography*, pages 266–300. Springer, 2024.

[26] Yvo Desmedt. Society and group oriented cryptography: A new concept. In *conference on the theory and application of cryptographic techniques*, pages 120–127. Springer, 1987.

[27] Benny Chor, Shafi Goldwasser, Silvio Micali, and Baruch Awerbuch. Verifiable secret sharing and achieving simultaneity in the presence of faults. In *26th Annual Symposium on Foundations of Computer Science (sfcs 1985)*, pages 383–395. IEEE, 1985.

[28] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE transactions on information theory*, 31(4):469–472, 1985.

[29] Victor Shoup and Rosario Gennaro. Securing threshold cryptosystems against chosen ciphertext attack. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 1–16. Springer, 1998.

[30] Christian Cachin, Klaus Kursawe, Anna Lysyanskaya, and Reto Strobl. Asynchronous verifiable secret sharing and proactive cryptosystems. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, pages 88–97, 2002.

[31] Ran Canetti, Rosario Gennaro, Stanisław Jarecki, Hugo Krawczyk, and Tal Rabin. Adaptive security for threshold cryptosystems. In *Annual International Cryptology Conference*, pages 98–116. Springer, 1999.

[32] Andreas Erwig, Sebastian Faust, and Siavash Riahi. Large-scale non-interactive threshold cryptosystems in the yoso model. *Cryptology ePrint Archive*, 2021.

[33] Philippe Bulens, Damien Giry, and Olivier Pereira. Running {Mixnet-Based} elections with helios. In *2011 Electronic Voting Technology Workshop/Workshop on Trustworthy Elections (EVT/WOTE 11)*, 2011.

[34] Haoqian Zhang, Louis-Henri Merino, Vero Estrada-Galinanes, and Bryan Ford. Flash freezing flash boys: Countering blockchain front-running. In *2022 IEEE 42nd International Conference on Distributed Computing Systems Workshops (ICDCSW)*, pages 90–95. IEEE, 2022.

[35] Sisi Duan, Michael K Reiter, and Haibin Zhang. Beat: Asynchronous bft made practical. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 2028–2041, 2018.

[36] Chelsea Komlo and Ian Goldberg. Frost: Flexible round-optimized schnorr threshold signatures. In *International Conference on Selected Areas in Cryptography*, pages 34–65. Springer, 2020.

[37] Rosario Gennaro and Steven Goldfeder. Fast multiparty threshold ecdsa with fast trustless setup. In *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*, pages 1179–1194, 2018.

Appendix A - Cryptographic primitives

In this section, we detail the cryptographic primitives that secure Confidential Data Rails.

Notation. We use a cyclic group $G = \langle g \rangle$ of prime order q, and an *independent* generator $h_{\text{ped}} \in G$ such that $\log_g(h_{\text{ped}})$ is unknown (for Pedersen commitments). We also fix an *independent* generator $\bar{g} \in G$ with $\log_g(\bar{g})$ unknown (used in Chaum–Pedersen proofs). The threshold is $t \leq n$. The DKG output is a distributed secret $x \in \mathbb{Z}_q$ with the network public key $h := g^x$ (used exclusively as the encryption key). We write $\mathbb{Z}_q$ consistently, and use "Chaum–Pedersen" and "Fiat–Shamir" with typographic en-dashes. Hash functions are typed as follows: $H_1 : G \to \{0,1\}^\ell$ (KDF/XOF to a message-length mask), and $H_2, H_4 : \{0,1\}^* \to \mathbb{Z}_q$, each with domain separation.

Distributed Key Generation via Pedersen Scheme

Distributed Key Generation (DKG) is a foundational component of threshold cryptography, designed to enable a group of participants to jointly generate a cryptographic key pair without requiring any single party to know the private key. The goal is to eliminate reliance on trusted third parties while ensuring robustness, verifiability, and fault tolerance.

Early work by Pedersen [15] introduced the first verifiable DKG protocol based on Verifiable Secret Sharing (VSS) [16,17,18], allowing participants to confirm the integrity of each share without revealing secrets. This seminal construction established the basic principles of verifiable randomness and public commitments that underpin all later DKG designs.

Subsequent improvements focused on efficiency, scalability, and robustness. Gennaro et al. [19] provided a formal treatment of DKG in the presence of malicious adversaries, identifying biasing attacks on Pedersen's protocol and providing a provably secure construction that ensures uniform key distribution and secrecy for discrete-log systems. Canny and Sorkin [20] later developed a large-scale DKG protocol optimized for Internet-sized networks, substantially reducing communication overhead. Building on these, Kate, Zaverucha, and Goldberg [21] introduced pairing-based polynomial commitments, achieving constant-size verification data and near-linear scalability.

Beyond traditional discrete-log settings, DKG has also evolved in the context of large distributed systems and blockchains. Syta et al.'s RandHound and RandHerd protocols [22] leveraged DKG as a core primitive for decentralized randomness generation with public verifiability, while Hanke et al.'s Threshold Relay in DFINITY [23] integrated a BLS-based DKG mechanism directly into blockchain consensus, demonstrating DKG's applicability to global-scale, long-lived networks.

In parallel, the DKG paradigm has been extended to post-quantum settings. Recent research has explored lattice-based DKG constructions under the (Module/R-)LWE and MSIS assumptions, offering quantum-resistant alternatives. Notably, Alborch et al. [24] proposed an R-LWE-based DKG supporting threshold encryption and decryption, while Gur et al. [25] designed a two- round lattice-based DKG achieving arbitrary t-of-n thresholds with tight security proofs. These advances show that robust DKG mechanisms can be realized even under quantum-resistant assumptions, although practical deployment remains challenging due to communication and performance overhead.

Despite these developments, Pedersen DKG remains one of the most practical and trusted choices in real-world distributed systems, particularly within architectures constrained by Trusted Execution Environments (TEEs). Its enduring relevance stems from three key advantages: (1) simplicity and auditability based on standard discrete logarithm assumptions, (2) deterministic verifiability of shares and commitments, and (3) computational efficiency compatible with TEE-based execution.

Protocol Description. We outline a Pedersen-style DKG tailored for Confidential Data Rails protocol. The protocol enables a set of n validators $\mathcal{P} = \{P_1, \ldots, P_n\}$ to jointly generate a public/private key pair (PK, x), where the private key $x \in \mathbb{Z}_q$ remains secret shared among participants and the public key $\mathsf{PK} = g^x$ is globally verifiable on-chain. The DKG proceeds over a cyclic group $G = \langle g \rangle \subset \mathbb{Z}_p^*$ of order q, with an

independent second generator $h_{\text{ped}} \in G$ such that $\log_g h_{\text{ped}}$ is unknown. The global parameters are

$$\mathsf{params} = (p, q, g, h_{\text{ped}}, \bar{g}, n, t),$$

where t denotes the reconstruction threshold.

1. **Polynomial Commitment.** Each participant P_j independently samples two random degree-$(t-1)$ polynomials over $\mathbb{Z}_q$:
$$f_j(X) = \sum_{k=0}^{t-1} a_{jk} X^k, \qquad f'_j(X) = \sum_{k=0}^{t-1} r_{jk} X^k,$$
where coefficients a_{jk}, r_{jk} are drawn uniformly at random. P_j then commits to its coefficients using Pedersen commitments:
$$C_{j,k} = g^{a_{jk}} h_{\text{ped}}^{r_{jk}} \in G, \quad k = 0, \ldots, t-1,$$
and broadcasts the commitment vector $\{C_{j,k}\}$ to the blockchain for public verifiability.

2. **Share Distribution.** For each participant P_i, P_j computes
$$s_{ij} = f_j(i), \qquad r_{ij} = f'_j(i),$$
and securely transmits (s_{ij}, r_{ij}) to P_i.

3. **Verification and Complaint.** Each P_i verifies consistency:
$$g^{s_{ij}} h_{\text{ped}}^{r_{ij}} \stackrel{?}{=} \prod_{k=0}^{t-1} C_{j,k}^{i^k} \pmod p.$$
If false, P_i raises an on-chain complaint. Invalid responses trigger the exclusion or penalization of P_j, enforced by the CDR's core contract.

4. **Aggregation and Public Key.** After verification, each validator computes local aggregates
$$x_i = \sum_{j=1}^{n} s_{ij} \bmod q, \qquad r_i = \sum_{j=1}^{n} r_{ij} \bmod q.$$
Since $C_{j,0} = g^{a_{j0}} h_{\text{ped}}^{r_{j0}}$, the network public key cannot be obtained by $\prod_j C_{j,0}$. Each P_j therefore publishes
$$Y_j = g^{a_{j0}}$$
together with either (i) an opening r_{j0} such that $C_{j,0} = Y_j h_{\text{ped}}^{r_{j0}}$, or (ii) a zero-knowledge consistency proof that $C_{j,0}$ and Y_j share the same a_{j0}. The network public key is
$$\mathsf{PK} = g^x = \prod_{j=1}^{n} Y_j.$$
(Only the constant terms are opened/linked; the rest of the DKG transcript remains information-theoretically hiding.)

Proactive Share Refresh. To mitigate share aging and maintain forward secrecy, the protocol supports a *proactive refresh* that rerandomizes private shares without changing the global secret x or the public key PK.

1. **Zero-constant polynomials.** Each participant P_j samples two random degree-$(t-1)$ polynomials with zero constant terms:
$$g_j(X) = \sum_{k=1}^{t-1} b_{jk} X^k, \qquad g'_j(X) = \sum_{k=1}^{t-1} b'_{jk} X^k.$$

2. **Commitment broadcast.** P_j publishes commitments to the coefficients:
$$D_{j,k} = g^{b_{jk}}\, h_{\text{ped}}^{b'_{jk}}, \quad k = 1, \ldots, t-1.$$

3. **Private share distribution.** For each P_i, P_j sends the corresponding evaluations:
$$\delta_{ij} = g_j(i), \qquad \rho_{ij} = g'_j(i).$$

4. **Verification.** Each receiver verifies the correctness of its received shares via:
$$g^{\delta_{ij}}\, h_{\text{ped}}^{\rho_{ij}} \stackrel{?}{=} \prod_{k=1}^{t-1} D_{j,k}^{i^k}.$$

5. **Share update.** Upon successful verification, P_i locally refreshes its share and (re)derives the public fragment:
$$x_i \leftarrow x_i + \sum_j \delta_{ij} \bmod q, \qquad r_i \leftarrow r_i + \sum_j \rho_{ij} \bmod q, \qquad h_i \leftarrow g^{x_i}.$$

6. **Public key invariance.** Since all $g_j(X)$ have zero constant terms, the aggregate secret remains unchanged:
$$\mathsf{PK} = g^x.$$

Proactive Share Reshare. In addition to periodic proactive refresh, the protocol supports a *proactive reshare* procedure that reassigns secret shares to a new participant set $\mathcal{P}' = \{P'_1, \ldots, P'_{n'}\}$ while preserving the global secret x and public key $\mathsf{PK} = g^x$. This allows the committee to dynamically add or remove validators (e.g., during kernel/committee rotation) without re-running a full DKG.

1. **Select a qualified old subset and precompute anchors.** To enable dynamic reconfiguration without re-running a full DKG, the protocol selects any qualified subset $S \subseteq \mathcal{P}$ of size t or more to act as the re-dealers. Let $S \subseteq \mathcal{P}$ be any qualified old subset with $|S| \geq t$. For notational clarity, old indices are $j \in S$ and new indices are $i \in \{1, \ldots, n'\}$. Compute the Lagrange coefficients at 0 over $\mathbb{Z}_q$ for the old points S:
$$\lambda_j^S = \prod_{\substack{\ell \in S \\ \ell \neq j}} \frac{-\ell}{j - \ell} \pmod q.$$
Using the on-chain Pedersen commitments from the original DKG, form the public anchors for each old index j:
$$\widehat{C}_j \;:=\; \prod_{\ell=1}^{n} \prod_{k=0}^{t-1} C_{\ell,k}^{j^k} \;=\; g^{x_j}\, h_{\text{ped}}^{r_j},$$
where $x_j = \sum_\ell f_\ell(j)$ and $r_j = \sum_\ell f'_\ell(j)$ are the (hidden) aggregated share and blinding values held by P_j from the latest epoch. The elements $\widehat{C}_j$ are *publicly computable*.

2. **Per-dealer resharing polynomials (weighted constants).** Each old participant $P_j \in S$ samples two random degree-$(t'-1)$ polynomials with zero constant terms:

$$h_j(X) = \sum_{k=1}^{t'-1} c_{jk} X^k, \qquad h'_j(X) = \sum_{k=1}^{t'-1} c'_{jk} X^k,$$

and defines the weighted resharing polynomials

$$F_j(X) = \underbrace{\lambda_j^S x_j}_{\text{constant}} + h_j(X), \qquad F'_j(X) = \underbrace{\lambda_j^S r_j}_{\text{constant}} + h'_j(X).$$

The λ_j^S weights ensure that the new aggregate secret remains x.

3. **Commitment broadcast.** Each P_j publishes Pedersen-style commitments to the (non-constant) resharing coefficients:

$$E_{j,k} = g^{c_{jk}} \, h_{\text{ped}}^{c'_{jk}}, \qquad k = 1, \ldots, t'-1.$$

4. **Private share distribution to new members.** For every new participant P'_i, P_j computes

$$s'_{ij} = F_j(i) = \lambda_j^S x_j + h_j(i), \qquad r'_{ij} = F'_j(i) = \lambda_j^S r_j + h'_j(i),$$

encrypts (s'_{ij}, r'_{ij}) to P'_i's enclave communication key, and transmits privately.

5. **Per-share verification at the receiver.** Each recipient P'_i verifies every received pair (s'_{ij}, r'_{ij}) by checking the publicly verifiable relation

$$g^{s'_{ij}} \, h_{\text{ped}}^{r'_{ij}} \stackrel{?}{=} \left(\widehat{C}_j\right)^{\lambda_j^S} \cdot \prod_{k=1}^{t'-1} E_{j,k}^{\,i^k} \pmod p.$$

This binds the reshared values to the original DKG transcript (via $\widehat{C}_j$) and to the new polynomial coefficients (via $E_{j,k}$).

6. **Aggregation by new participants.** Each new participant P'_i aggregates all verified contributions to obtain its fresh local share and blinding factor:

$$x'_i = \sum_{j \in S} s'_{ij} \bmod q, \qquad r'_i = \sum_{j \in S} r'_{ij} \bmod q, \qquad h'_i = g^{x'_i}.$$

(No extra Lagrange weighting is needed here because the constants were already multiplied by λ_j^S on the dealer side.)

7. **Public key invariance.** Because $\sum_{j \in S} \lambda_j^S x_j = x$, the new aggregate polynomial over the index set of $\mathcal{P}'$ has constant term x, hence the network public key remains unchanged:

$$\mathsf{PK} = g^x \;=\; \prod_{j=1}^{n} g^{a_{j0}} \;=\; \prod_{i \in S'} (h'_i)^{\lambda_i^{S'}},$$

where $S' \subseteq \{1, \ldots, n'\}$ is any qualified subset with $|S'| \geq t'$, and $\lambda_i^{S'}$ are the Lagrange coefficients at 0 for the *new* index set. This ensures continuity of the cryptographic identity of the network, while redistributing trust across a new validator set.

This mechanism enables periodic renewal of validator state without reinitializing the entire key generation process. It protects against partial compromise, supports long-term operation, and facilitates adaptive membership changes or audit-based re-randomization in the CDR committee, ensuring that stale shares never accumulate into a systemic vulnerability.

Integration Discussion. In the first version of implementation of the CDR protocol, each committee member publicly releases its per-party key component $h_i = g^{x_i}$ after the DKG phase *and* after each proactive refresh, enabling standard Chaum–Pedersen equality proofs $\log_g h_i = \log_u u_i$ within the TDH2 decryption protocol. Once these public shares are available, the online threshold decryption is operationally equivalent to Feldman's construction [17]: each node provides a verifiable partial decryption, and any qualified subset interpolates the result. The main modifications lie in the DKG setup and maintenance phases rather than in the TDH2 runtime.

Compared with a pure Feldman approach, this design retains Pedersen's advantages during key generation and refresh. The commitment form $C_{j,k} = g^{a_{jk}} h_{\text{ped}}^{r_{jk}}$ supports bias-resistant, delay-reveal publication of the final public key via the $Y_j = g^{a_{j0}}$ values with openings or consistency proofs, while keeping the *rest* of the DKG transcript information-theoretically hiding. It also preserves privacy during proactive resharing and supports migration to configurations that omit public h_i values if stronger confidentiality is later required, while keeping the decryption path lightweight and publicly verifiable.

Threshold Decryption (TDH2)

Threshold decryption is the natural complement to DKG. While DKG enables a set of participants to collaboratively create a shared key without disclosing secret material, threshold decryption ensures that *no single node* can decrypt ciphertexts alone. Instead, a subset of at least t participants must cooperate, each contributing a partial decryption that can be publicly verified and aggregated to recover the plaintext.

Threshold cryptosystems distribute decryption authority across multiple parties [26,27], with Shamir's secret sharing [16] providing the algebraic backbone. Building on ElGamal's multiplicative structure [28], Shoup and Gennaro formalized CCA-secure threshold decryption via TDH1/TDH2 [9,29], combining verifiable partial decryptions with NIZKs to achieve full CCA2 security (under DDH) in the *random oracle model (ROM)* using the Fiat–Shamir transform. Robustness for asynchronous networks and mobile adversaries is addressed by asynchronous VSS and proactive techniques [30], while adaptive security is captured by subsequent formal treatments [31]. To scale beyond long-lived committees, non-interactive, large-scale threshold cryptosystems in the YOSO model [32] decouple participation from persistent identities while preserving public verifiability. In applied settings, threshold ElGamal/TDH2 support verifiable e-voting (e.g., Helios [33]), front-running–resistant commit-reveal designs [34], and practical asynchronous BFT systems [35]. The same "verifiable partial computation" paradigm informs modern threshold signatures, e.g., FROST's two-round Schnorr [36] and dealerless distributed ECDSA [37].

We adopt TDH2 as the network's threshold decryption primitive because it matches our security and systems constraints. Cryptographically, TDH2 delivers CCA2 security in the ROM under discrete-log/DDH assumptions while binding decryptions to context via labels, preventing replay and cross-protocol misuse. Operationally, each participant outputs a partial decryption with a non-interactive proof that is statelessly verifiable on-chain, enabling transparent auditing, accountability, and straightforward slashing policies. Systems-wise, TDH2's online path uses modular exponentiations and simple proofs (no pairings or heavy ZKs), fitting TEE budgets and keeping gas/latency predictable. The scheme composes naturally with our Pedersen-based DKG outputs and equality proof checks, so integration requires no bespoke crypto glue and preserves public verifiability end-to-end. TDH2 also accommodates proactive share refresh and key-rotation without altering the public key.

Protocol Description. We describe the *Threshold Decryption Handler (TDH2)* used in CDR for verifiable, labeled public-key encryption and distributed decryption. The scheme enables any subset of t out of n validators, each holding a secret share x_i of the global key x, to collaboratively decrypt ciphertexts under public verification. All operations take place in the same group $G = \langle g \rangle$ of order q, using an independent generator $\bar{g}$ to support Chaum–Pedersen-style NIZKs. The network public key is $h = g^x$, derived from the DKG phase. All exponentiations are modulo p; all exponents are in $\mathbb{Z}_q$. Implementations must enforce group-member and canonical-encoding checks for all group elements.

Encryption (Labeled PKE). To encrypt $m \in \{0,1\}^\ell$ under a public label L (associated data) and context string ctx (for domain separation), the sender does:

1. Sample $r, s \xleftarrow{\$} \mathbb{Z}_q$.
2. Compute
$$u = g^r, \quad \bar{u} = \bar{g}^r, \qquad w = g^s, \quad \bar{w} = \bar{g}^s,$$
$$c = H_1(h^r) \oplus m, \qquad e = H_2(\text{ctx}, c, L, u, w, \bar{u}, \bar{w}), \qquad f = s + re \pmod q.$$
3. Output the ciphertext
$$C = (c, u, \bar{u}, e, f, L, \text{ctx}).$$

The tuple $(u, \bar{u}, e, f)$ forms a Chaum–Pedersen-type NIZK of discrete-log equality $\log_g u = \log_{\bar{g}} \bar{u}$ via Fiat–Shamir. A verifier parses $C = (c, u, \bar{u}, e, f, L, \text{ctx})$, recomputes $w = g^f/u^e$ and $\bar{w} = \bar{g}^{\,f}/\bar{u}^{\,e}$, checks group membership, and accepts iff $e = H_2(\text{ctx}, c, L, u, w, \bar{u}, \bar{w})$. (If ctx is a fixed system parameter, it may be omitted from C and treated as implicit.)

Partial Decryption and Proof of Correctness. Each validator P_i, holding share x_i and public fragment $h_i = g^{x_i}$, on input $C = (c, u, \bar{u}, e, f, L, \text{ctx})$:

1. **Ciphertext verification.** Recompute $w = g^f/u^e$ and $\bar{w} = \bar{g}^{\,f}/\bar{u}^{\,e}$; check membership; reject if $e \neq H_2(\text{ctx}, c, L, u, w, \bar{u}, \bar{w})$.
2. **Share computation and proof.** Compute $u_i = u^{x_i}$. Sample $s_i \xleftarrow{\$} \mathbb{Z}_q$, set $u_i' = u^{s_i}$, $h_i' = g^{s_i}$, and
$$e_i = H_4(\text{ctx}, g, u, h_i, u_i, u_i', h_i'), \qquad f_i = s_i + x_i e_i \pmod q.$$
Output the decryption share (u_i, e_i, f_i).

Combining and Message Recovery. Given any valid set of t decryption shares $\{(u_i, e_i, f_i)\}_{i \in S}$ for indices $S \subseteq \{1, \ldots, n\}$ with $|S| = t$, the aggregator proceeds:

1. **Per-share verification.** For each $i \in S$, compute
$$u_i'' = u^{f_i}/u_i^{e_i}, \qquad h_i'' = g^{f_i}/h_i^{e_i},$$
check membership, and verify
$$e_i = H_4(\text{ctx}, g, u, h_i, u_i, u_i'', h_i'').$$
Discard invalid shares.
2. **Threshold interpolation.** Using Lagrange coefficients at zero over $\mathbb{Z}_q$ for points $\{1, \ldots, n\}$,
$$\lambda_i^S = \prod_{\substack{j \in S \\ j \neq i}} \frac{-j}{i - j} \pmod q,$$
compute
$$Y = \prod_{i \in S} u_i^{\lambda_i^S} = \prod_{i \in S} (u^{x_i})^{\lambda_i^S} = u^{\sum_{i \in S} \lambda_i^S x_i} = u^x = h^r.$$
3. **Message recovery.** Recover the plaintext:
$$m = H_1(Y) \oplus c.$$