The DATA Network

The Data Foundation

Version 0.2 - June 2026

1 The Data Problem

We are living through a foundational shift in AI progress. Transformer-based large language models have redefined what machines can do, from reasoning and code generation to complex decision-making. Diffusion models are improving every day, making it possible to create realistic and artistic images and videos, and robots are consuming more real-world data to navigate unknown environments more effectively.

The appetite for more capable models shows no signs of slowing. The cornerstone of this progress is two core ingredients: computation and data. On the computation side, progress has been steady. Hardware improvements, including faster GPUs, TPUs with more memory, along with software optimizations like quantization and tiling that take more out of existing infrastructure, have kept pace with growing model ambitions. On the other hand Data is increasingly the binding constraint. The seminal 2022 DeepMind paper, "Training Compute-Optimal Large Language Models" [1] (often called the Chinchilla paper), made this explicit: to avoid under-training a larger model, the number of tokens required must scale with the parameter count. More parameters, which is what everyone is pursuing these days, demand proportionally more data, not just more compute.

In the realm of large language models (LLMs), as model builders push for more parameters and support for longer context windows, the public web data used heavily for pretraining is nearly exhausted. Villalobos et al., in their paper[2], argue that human-written text is a finite resource and may become a bottleneck for scaling LLMs. They estimate the stock of publicly available human-generated text on the internet, as well as in books, articles, forums, and other sources. The paper predicts that training datasets could soon reach the limits of available public, human-generated text. At current trends, language models will fully utilize that stock between 2026 and 2032, if not earlier. This limits the size of models that can be properly trained.

In response, many labs have turned to synthetic data and multi-modal learning. While synthetic data unlock scale in the short term, they carry a structural weakness, models trained on their own outputs risk compounding errors, narrowing over time, and failing to encounter the long-tail, noisy, genuinely novel situations that real-world data naturally contain. Synthetic pipelines can improve a model within a known distribution, but they cannot expand the frontier of what a model has actually seen. Several recent papers show[3] that synthetic data is a supplement, not a complete replacement for scarce human-generated data,

and warn that self-learning on synthetic data could cause 'model collapse.' But there is another way; the vast majority of web data (estimates of 80%) remain private and inaccessible, leaving a significant and largely untapped reservoir of human knowledge out of reach. We aim to solve the challenges of unlocking these valuable resources.

Moreover, the next wave of AI is not only about LLM-based applications. Robotics, autonomous systems, physical AI, and multi-modal agents require data that does not yet exist at meaningful scale: sensor streams from real environments, interaction logs from embodied systems, and edge cases in unpredictable physical conditions. These modalities cannot be synthesized from text or scraped from the web. They must be captured, structured, and made available, which means that the data problem is not just unsolved. It is growing in scope and urgency.

This only means that companies that can provide clean and effective data at scale are going to become some of the most valuable businesses ever built. And it doesn't stop at dedicated data companies. Every platform sitting on top of human behaviour, every app with engaged users, every business producing proprietary content is and will be monetizing their data in the age of AI.

2 Challenges

In previous section, we have highlighted the value of data and the need to unlock existing private data, or to create incentives and structures to capture new data. Yet there are several challenges ahead.

2.1 Challenges for Data Providers

A key barrier is data ownership and the risks that come with it. Today, there is no easy, fast way to define how ownership is split between the original data owner, the entity that curates or aggregates the data, and the model builder that ultimately consumes it. In practice, permission, attribution, and compensation are often negotiated ad hoc, and that uncertainty makes even well-intentioned data sharing feel dangerous. Many digital businesses rely on private, proprietary data and hard-won institutional knowledge, and they worry that granting access could expose their differentiated workflows, reveal sensitive user behavior, or leak commercially valuable insights that competitors can reverse-engineer. They also worry about downstream misuse: data being re-sold without consent, combined with other sources in ways that violate internal policies, or used to train models that directly substitute the business's core product.

These concerns have intensified as many model-building companies have taken a "scrape first, ask later" approach, pushing legal and ethical questions to the margins in order to maintain a competitive edge. Even when a buyer is trustworthy, providers still lack the tooling to share data while retaining control: clear licensing terms, enforceable usage restrictions, auditable provenance, and confidentiality guarantees that prevent the data from becoming broadly accessible. We believe the solution is to build an open, fair economic model that

defines ownership, standardizes licensing and payment, and makes confidential data transfer possible, so providers can participate in the AI economy without risking their secret sauce.

2.2 Challenges for Data Users

The higher the data quality, the better the models. Given the volume of data consumed, it is difficult to catch noise, duplicates, and spam. Beyond quality, distribution is often a challenge. It can be skewed, and many languages do not have enough data. For example, there is an abundance of resources for English, while many other languages face a long-tail data shortage.

More recently, it has become clear that end-to-end data tracing is needed for both legal protection and security. This means tracing data from the point of capture through the entire pipeline and being able to provide a "data receipt" for every data point used for training. Early on, many model builders took a "scrape first, ask later" approach. Regulations like GDPR and CCPA impose strict consent requirements and hefty fines, up to 4% of global revenue, making unauthorized collection a serious financial liability. Terms of service violations also expose companies to costly litigation, as seen in high-profile suits from publishers and platforms asserting copyright and contract claims. A related challenge is proving that a data provider is the rightful owner of the data being sold. Buyers need a clear receipt showing the data was acquired from rightful owners.

Attack vectors range from simple tactics, such as AEOs (SEOs for agents) that steer users toward specific products in search results, to more complex attacks. In the latter, training data can influence coding agents to use malicious or third-party packages, enabling supply chain compromises.

3 The DATA Network

We propose the DATA network as a unified, secure, open data infrastructure and blockchain designed to solve AI's biggest bottleneck: access to rights-cleared, specialized, real-world data.

Within the Data network, Data is a native object and every piece of data can be:

- *Proven*: a verifiable, portable record of where it came from and under what terms it was contributed
- *Confidential*: encrypted so only the parties you authorize can ever decrypt it
- *Governed*: owned, licensed, and monetized under transparent on-chain terms

This creates an open and flexible framework that anyone on the internet can participate in. For instance, someone could create a data marketplace where all data is stored confidentially, alongside proof of authenticity and a tamper-proof public audit trail. The data owner could define the terms of use and require data consumers to acquire a license and set up payments in a stablecoin.

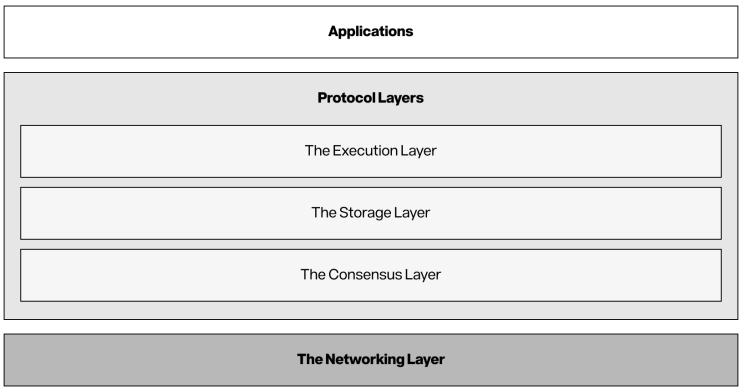

Figure 1: The DATA Protocol: Layers

Under the hood, the DATA Protocol defines how a set of decentralized nodes can participate over a peer-to-peer network, building the required infrastructure. These nodes, open to all, work together to maintain consensus, process transactions, facilitate confidential data exchanges, and protect the integrity of the protocol. Following the principle of separation of concerns, the protocol is divided into three layers:

- *a consensus layer* that ensures nodes follow the protocol and can reach agreement over the state of the protocol

- *an execution environment* that acts as a virtual machine where data is registered, and where ownership, authenticity, attestations, and licensing terms are stored, enforced, and recorded in a tamper-proof audit trail.
- *a storage layer* where data are stored and access controlled securely and confidentially

Lets take a closer look at each layer.

3.1 The Consensus Layer

The consensus layer ensures network integrity, security, and stability by orchestrating block formation among all network participants. Recent years have witnessed remarkable advances and research breakthroughs in consensus mechanisms.

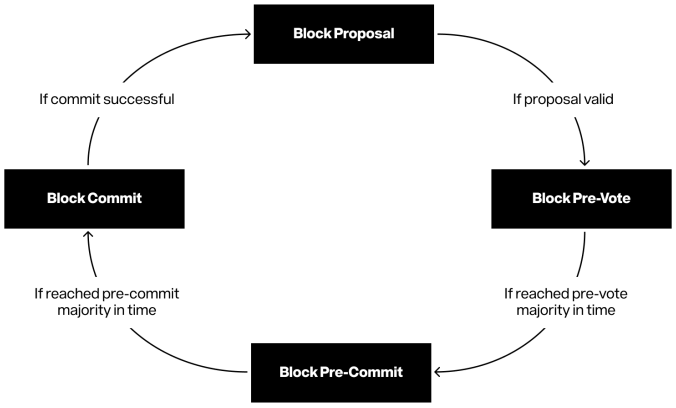

Figure 2: Tendermint's block formation cycle (the happy path)

The DATA network uses a modern, performant implementation of Tendermint called CometBFT[4]. Tendermint[5] is a Byzantine Fault Tolerant consensus engine that offers well-defined properties and customization options that align with our design needs:

- **Fault Tolerance:** maintaining security even when 33% of validators are malicious or offline. The system ensures all honest nodes reach consensus on committed blocks and uses cryptographic proofs to identify and penalize bad actors.
- **Liveness Guarantees:** The network continues to progress when more than two-thirds of validators (measured by voting power) remain active and participating.
- **Instant Finality:** Transactions achieve finality quickly after being included in a block. This immediate finality ensures reliability for applications needing immutability - a key factor in user experience.

- **Modular Design:** The Application Blockchain Interface (ABCI) decouples consensus from execution, enabling future consensus upgrades without impacting execution.

The consensus layer of the DATA Protocol uses a Proof-of-Stake (PoS) mechanism, with validators playing a crucial role in maintaining blockchain security and integrity. This creates a direct economic incentive for an honest and efficient operation, as malicious behavior results in penalties, including loss of staked tokens. The layer manages all operations related to staking, lockup requirements, and reward distribution.

3.2 The Storage Layer

The storage layer stores data confidentially and provides a programmable interface for access control and for securely transferring data to the execution layer. To achieve this, nodes participate in and run the *Confidential Data Rails (CDR)*[6] protocol.

The CDR protocol is powered by a decentralized collection of trusted execution environments (TEEs) run by the validator nodes that operate off-chain and are managed and synchronized by a execution layer. Using this infrastructure validators can participate in distributed key generation and perform threshold decryption.

This core functionality lets users securely upload confidential data to the network and define access conditions. The data then becomes automatically accessible to users and systems that meet those conditions. Conditions can be satisfied dynamically based on the terms set in the execution layer.

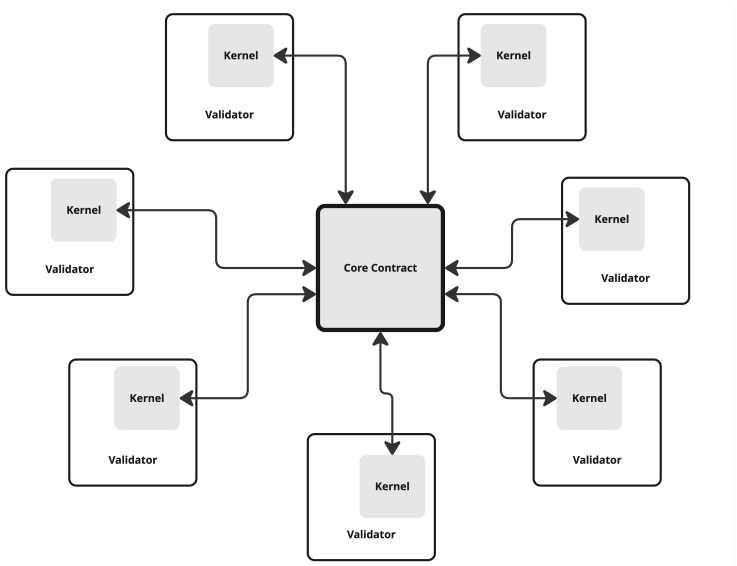

Figure 3: Validators running CDR protocol

This storage layer can support any size and any format of data at scale and while maintaining confidentiality of the data is highly compatible with data availability protocols like IPFS[7] and Walrus[8].

3.3 The Execution Layer

On top of all previous layers is *the Execution Layer*. It runs the virtual machine where all data points are registered, and where ownership, authenticity, attestations, and licensing terms are recorded, stored, and enforced. Every interaction with the virtual machine is tamper-proof and auditable, and it happens through user-submitted transactions (for example, a data deal).

At the core of this layer, the Ethereum Virtual Machine (EVM) is used, with performance optimizations to support scalable data registration, operations, exchanges, and other required functionality.

At the center is a smart contract responsible for checking authenticity and registering every data point at scale. Alongside another contract allows attestations such as proof of authenticity or proof of quality to be submitted by anyone on any data points. This constructs a ledger for data points that is:

- *Immutable*: records are permanent once written; deletion or modification is computationally infeasible on a well-secured chain
- *Timestamped*: each interaction and updates is timestamped and ordered, giving you a reliable sequence of events
- *Verified*: Verified through the decentralized network of nodes where no single party can alter the record without consensus
- *Cryptographic provenance*: each entry can be signed by its author (a wallet, a key, a node), proving who wrote what and when

Note that the data itself is stored on the Storage layer using CDR and only a cryptographic-verifiable reference of it gets to be registered on-chain. The data owner could setup terms and conditions for each data point and every terms and conditions are defined as an smart contract where this contract has the power to automatically access control the data. Many existing templates and smart contracts are available to customize terms and license agreements and support payments in cryptocurrencies and stable coins.

Trace

A core application that runs on the execution layer is Trace. Even when data is properly licensed and securely transferred, a critical question remains unanswered: how does anyone verify that a given record was legitimately collected, that consent was obtained at the moment of contribution, and that nothing changed between collection and use? Without a reliable answer, data buyers carry legal exposure, contributors have no enforcement mechanism, and regulators cannot audit at scale. Trace solves this by providing every human-contributed record with a verifiable, end-to-end receipt that travels with the data for its entire lifecycle.

When a record enters the DATA Network through Trace, a tamper-proof, cryptographically signed receipt is generated immediately at the point of contribution. This receipt

captures the full provenance of the record: its origin, the contributing platform, the consent terms under which it was collected, a timestamp, and a content hash that detects any subsequent modification. The receipt is written to the execution layer's immutable ledger and remains permanently accessible. As the record moves through curation, cleaning, bundling, and eventual sale, the receipt travels with it, so that at any point downstream, any party can trace that record back to its source and verify its integrity.

Critically, this auditability does not come at the cost of contributor privacy. What is recorded on-chain is a cryptographic commitment to the contribution and its terms, not the personal data of the person who made it. Contributor identity is separated from contribution metadata at the protocol level, ensuring that a regulator auditing millions of consent records or a lab verifying an entire training dataset never gains access to the personal information of any individual contributor.

The result is a public audit trail that serves every participant in the data economy. AI labs can verify the provenance of entire datasets in seconds rather than weeks of legal review. Contributors can confirm at any time that their terms are being honored. And regulators can audit consent at scale without burdening providers. By anchoring every record to a verifiable receipt on a public, immutable ledger, Trace transforms data provenance from an afterthought into a first-class guarantee.

4 Conclusion

Data is now the binding constraint on AI progress. Public web data is nearly exhausted, synthetic alternatives cannot fully replace real-world signal, and the vast majority of private data remains inaccessible due to trust and tooling gaps. The DATA Protocol closes this gap by combining BFT consensus, TEE-backed confidential storage via CDR, and a tamper-proof EVM execution layer — giving providers a way to share data without losing control, and consumers access to rights-cleared, verifiable data at scale. As demand for specialized, real-world data grows across LLMs, robotics, and autonomous systems, this kind of open data infrastructure becomes foundational to AI's next era.

References

[1] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals, and Laurent Sifre. Training compute-optimal large language models, 2022.

[2] Pablo Villalobos, Anson Ho, Jaime Sevilla, Tamay Besiroglu, Lennart Heim, and Marius Hobbhahn. Will we run out of data? limits of llm scaling based on human-generated data, 2024.

[3] Feiyang Kang, Newsha Ardalani, Michael Kuchnik, Youssef Emad, Mostafa Elhoushi, Shubhabrata Sengupta, Shang-Wen Li, Ramya Raghavendra, Ruoxi Jia, and Carole-Jean Wu. Demystifying synthetic data in llm pre-training: A systematic study of scaling laws, benefits, and pitfalls, 2025.

[4] CometBFT. https://cometbft.com/, accessed February 2025.

[5] Jae Kwon. Tendermint: Consensus without mining. https://tendermint.com/static/docs/tendermint.pdf, 2014.

[6] Ramtin M. Seraj, Wenxing Duan, Hansol Lee, Steven Wang, Jongwon Park, and Hao (Leo) Chen. Confidential Data Rails: Secure, confidential, programmable transfer of data. https://www.story.foundation/confidentialdatarails.pdf, 2025.

[7] 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.

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

Appendix - Confidential Data Rails

Confidential Data Rails

Secure, Confidential, Programmable Transfer of Data

Version 0.14 - 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 the DATA Protocol 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. Built on the same secure foundation of the Data Protcol, this architecture ensures complete data confidentiality while offering flexibility through programmable access control and decentralized infrastructure.

1 Introduction

Computation has scaled steadily, but data is now the true bottleneck for AI progress. Public human-generated text is approaching exhaustion, synthetic data risks model collapse, and the modalities needed for the next wave of AI, robotics, physical agents, and multi-modal systems cannot be scraped or synthesized at all. Meanwhile, roughly 80% of web data remain private and untapped, making the ability to unlock and monetize that reservoir one of the most valuable opportunities in the AI economy.

To address this gap, **the DATA Network**[1] was launched earlier this year as a unified, scalable and decentralized data layer. The DATA Network uses a novel *multi-core* architecture where a main EVM-compatible core automatically triggers a collection of *specialized cores* for enhanced performance. However, a challenge remained in the end-to-end journey of data assets: How can data be transferred confidentially and securely from the owner to the license holder? Data spans a wide range of formats and sizes, personal or business data for AI training, API keys for Web2 platforms, digital records, or even keys to external accounts.

Although many decentralized storage solutions (IPFS[2], Walrus[3], 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 assets. 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[4] 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. data 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 the Data Protocol'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 satisfies the on-chain condition (e.g. acquiring a license to a data asset), they can automatically access the CDR without requiring further involvement from the initial data uploader.

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

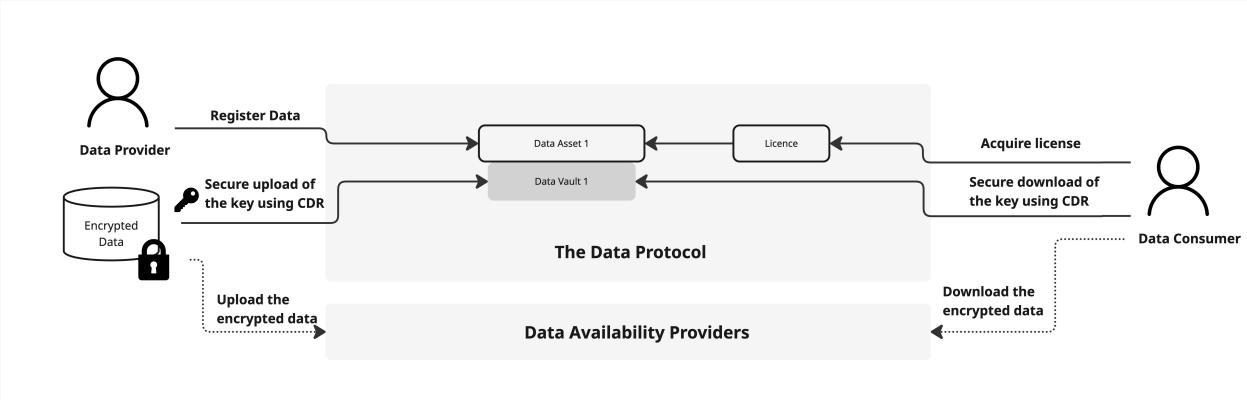

Figure 1: Secure and confidential transfer of Data assets between the data provider and the data consumer

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 asset on the Data Network 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.

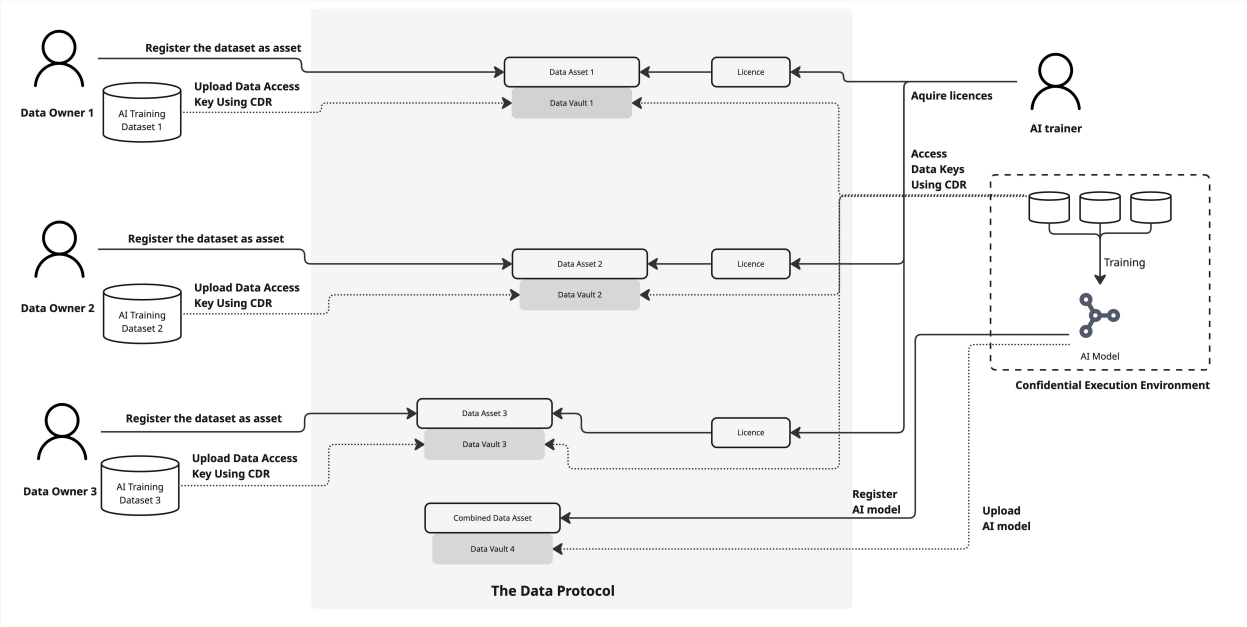

Figure 2: Data providers and model trainers participate to train and own a new AI model using the DATA Network and Confidential Data Rails

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[5].

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.

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.

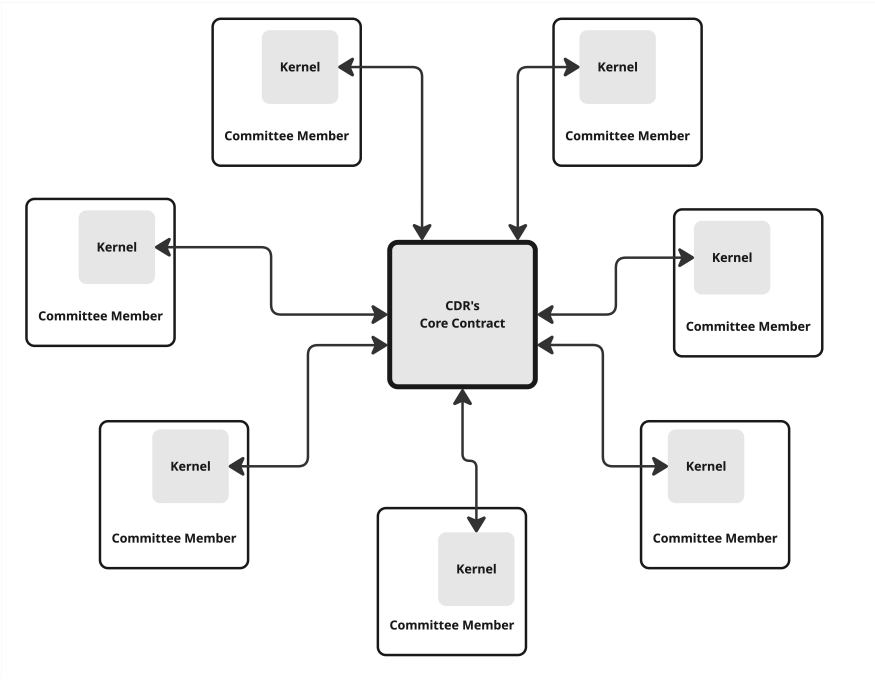

Figure 3: Confidential Data Rails High-Level Architecture Overview

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 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 4 for more details).

Rewards come from two sources: CDR usage fees and the Data Protocol's UBI (Universal Basic Income) pool. UBI is a fund of newly minted 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. Validators who participate in the CDR protocol cannot remove themselves from the 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[6] 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 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 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.

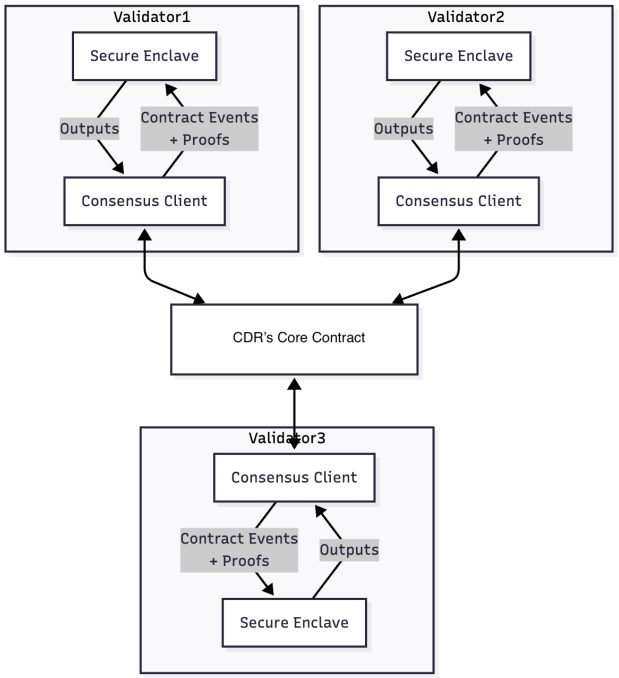

Figure 4: Kernels are synchronized through core contract order logs (events)

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 the network, **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. Data 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 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.

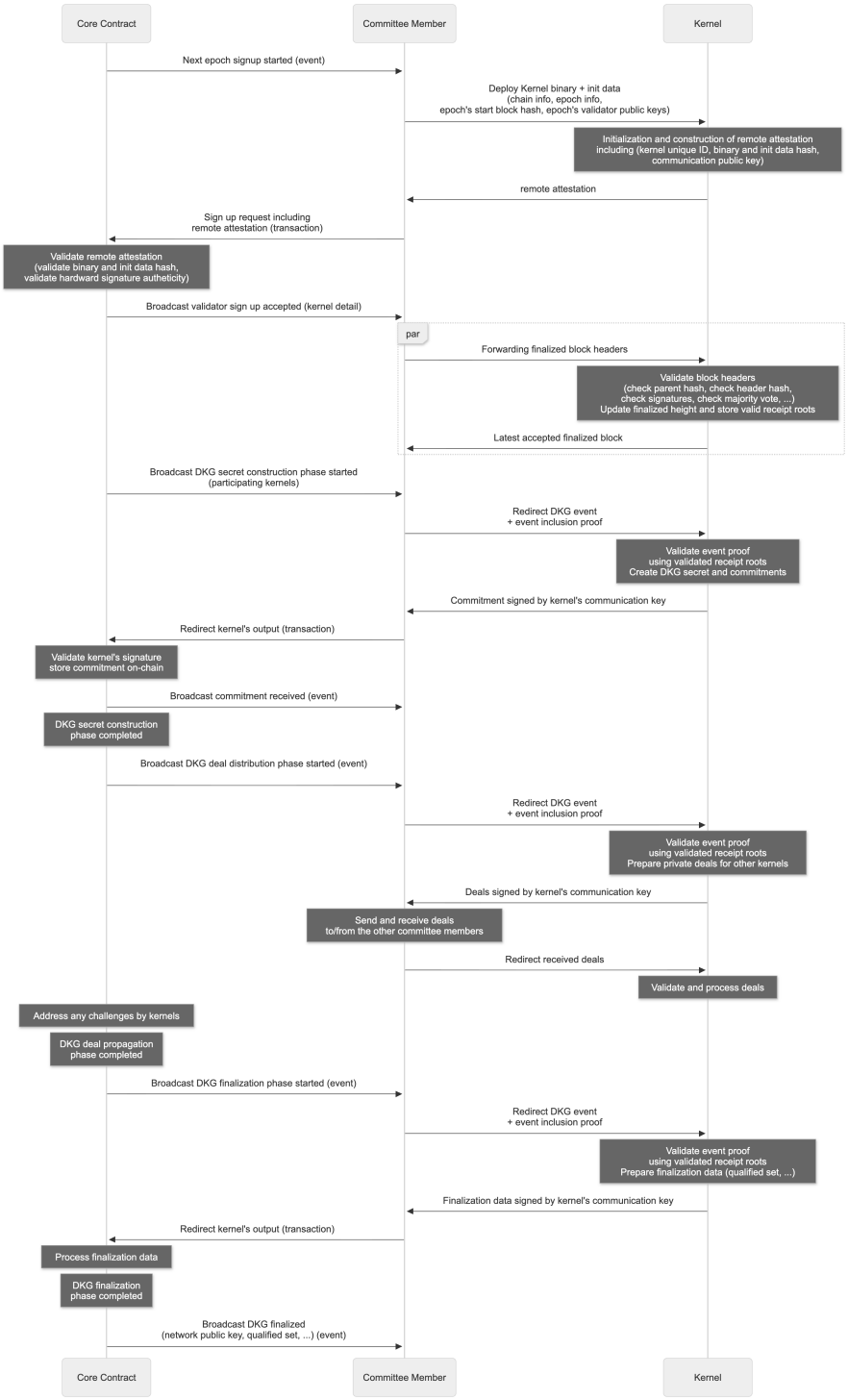

Figure 5: DKG process

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.

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[7] 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 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.

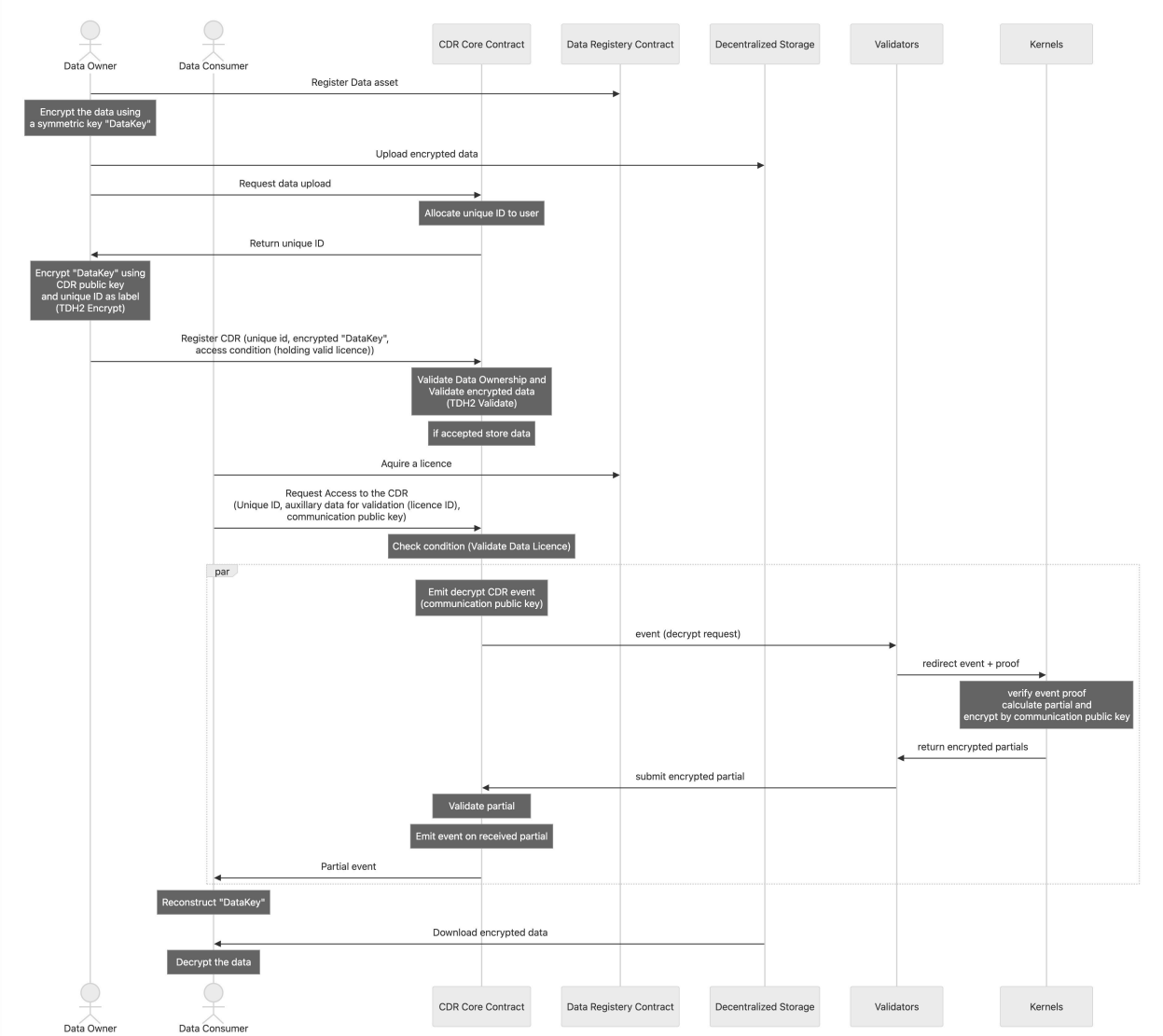

Figure 6: Secure transfer of data between Data owner and Data consumer using Confidential Data Rails protocol

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 4 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 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[8] introduced the first verifiable DKG protocol based on Verifiable Secret Sharing (VSS)[9,10,11], 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.[12] 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[13] later developed a large-scale DKG protocol optimized for Internet-sized networks, substantially reducing communication overhead. Building on these, Kate, Zaverucha, and Goldberg[14] 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[15] leveraged DKG as a core primitive for decentralized randomness generation with public verifiability, while Hanke et al.'s Threshold Relay in DFINITY[16] 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.[17] proposed an R-LWE-based DKG supporting threshold encryption and decryption, while Gur et al.[18] 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 [10]: 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 [19,20], with Shamir's secret sharing [9] providing the algebraic backbone. Building on ElGamal's multiplicative structure [21], Shoup and Gennaro formalized CCA-secure threshold decryption via TDH1/TDH2 [7,22], 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 [23], while adaptive security is captured by subsequent formal treatments [24]. To scale beyond long-lived committees, non-interactive, large-scale threshold cryptosystems in the YOSO model [25] decouple participation from persistent identities while preserving public verifiability. In applied settings, threshold ElGamal/TDH2 support verifiable e-voting (e.g., Helios [26]), front-running–resistant commit-reveal designs [27], and practical asynchronous BFT systems [28]. The same "verifiable partial computation" paradigm informs modern threshold signatures, e.g., FROST's two-round Schnorr [29] and dealerless

distributed ECDSA [30].

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:

$$m = H_1(Y) \oplus c.$$

4 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[31]. 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[32,33,34].

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[32,34]. 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[32,33,34]. 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)[35], 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 the data protocol; for example, it enables the programmable transfer of encrypted data between data 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] The Data Foundation. The Data Protocol: A unified data infrastructure for ai. https://www.datafdn.org/whitepaper.pdf, 2024.

[2] 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.

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

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

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

[6] 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.

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

[8] 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.

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

[10] 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.

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

[12] 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.

[13] 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.

[14] 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.

[15] 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.

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

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

[18] 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.

[19] 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.

[20] 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.

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

[22] 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.

[23] 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.

[24] 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.

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

[26] 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.

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] 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.

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

[33] 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.

[34] 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.

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