# Abstract

This white paper presents Lit Protocol, a protocol for creating decentralized key management networks that extends upon the foundations of public key cryptography established over the past several decades by luminaries like Diffie, Hellman, Boneh, and Shamir. Lit's work aims to address contemporary challenges faced in our digitally-mediated world, providing a novel solution to current shortfalls in secure communication, data integrity, interactions with distributed ledgers and digital identity.

Applying threshold secret schemes (TSS) and secure encrypted virtualization (SEV), Lit Protocol provides encryption, digital signatures, and programmable wallets as-a-service. This paper aims to provide a comprehensive overview of Lit Protocol and its features, including applicable use cases, underlying architecture, and references for further reading.

# Background

## Public Key Cryptography

Public key cryptography is a fundamental building block of modern security infrastructure on the Internet, playing a pivotal role in enabling secure communication, preserving data integrity, and facilitating trust in digital interactions. Its applications are ubiquitous, encompassing secure online transactions, payments, email communication, and access to cloud-based services. Public key cryptography is also an integral component of the web3 stack and the functioning of distributed systems (such as blockchains), providing the underlying mechanism that enables network transactions and consensus.

The seminal concept of public key cryptography was first introduced in the 1976s by Whitfield Diffie and Martin Hellman in their groundbreaking paper, "New Directions in Cryptography[1]." Their proposal outlined the use of two distinct keys—a public key and a private key—for encrypting and decrypting messages, a marked departure from traditional symmetric cryptography, which relied on a single shared key for both encryption and decryption.

Diffie and Hellman's method, also known as "asymmetric cryptography," is predicated on the mathematical properties of certain one-way functions[2], such as prime factorization and discrete logarithms. These functions enable the mathematical association between the two keys while making it virtually impossible to derive one from the other.

The advent of public key cryptography facilitated the establishment of trust in digital communication without necessitating a secure prior exchange of a shared secret key. Public key infrastructure (PKI) underpins widely employed security protocols, such as Transport Layer Security[3] (TLS) and Secure Socket Layer (SSL), which are indispensable for securing data transmission over the internet. These protocols ensure the privacy and protection of sensitive information, such as login credentials, financial data, and personal details, from potential attackers. At a high level, PKI can be implemented for:

1. *Encryption* for securing communication and the transfer of information on the web.
2. *Digital signatures* for verifying the integrity and attribution of information on the web.

### Encryption

Encryption is a cryptographic technique used to secure information by converting it into a scrambled, unreadable form known as “ciphertext”. This transformation is achieved using a specialized algorithm and an encryption key. Only those possessing the corresponding decryption key have the ability to revert the ciphertext back to its original, “plaintext” form.

---

[1] https://www-ee.stanford.edu/~hellman/publications/24.pdf
[2] https://en.wikipedia.org/wiki/One-way_function
[3] https://www.internetsociety.org/deploy360/tls/basics/

Encryption can be used to protect data stored or transmitted across the web from unauthorized access.

### Digital Signatures

Digital signatures can be used to validate the authenticity and integrity of a particular piece of digital information. They are created using a signer's private key and can be mathematically verified by anyone with access to the corresponding public key. By signing a message with a digital signature, the sender provides proof of origin, identity, and the message's unaltered state since the signature was applied.

## Decentralizing Public Key Infrastructure with Threshold Cryptography and Secure Encrypted Virtualization

At its core, Lit Protocol is an attempt to decentralize public key cryptography through the use of multi-party threshold secret schemes (TSS) and secure encrypted virtualization (SEV). The combination of these technologies provides a level of security and fault tolerance not present in traditional centralized key management systems and custody models.

Running the protocol is a thirty node network (the “Lit Network”) that provides generalizable key management infrastructure for identity-based encryption and programmable, serverless signing. The network is governed by an ecosystem of token holders who steer direction and development through tokenized governance.

With Lit, the nodes perform a distributed key generation (DKG) to create new public/private key pairs where no one party ever holds the entire key. Instead, each node holds a key share which they can use to sign and decrypt data with. These operations (signing or decryption) are performed by nodes in parallel and require participation from at least two-thirds of the network in order to return a complete signature or decryption key. This process never exposes the underlying private key itself.

The use of AMD’s Secure Encrypted Virtualization (SEV)[4] compliments the security provided by the distributed nature of the network itself, providing advanced hardware-level isolation for every Lit node. SEV ensures that node operators never have access to any key shares directly, nor the computation processed inside of the secure hardware.

## Core Functionality

The functionality provided by Lit can be divided into three distinct but interrelated services: decentralized access control, programmable wallets, and serverless signing.

[4] https://www.amd.com/en/developer/sev.html

## Decentralized Access Control

Lit can be used to introduce privacy to a host of web3 applications, specifically offering a solution to the “public-by-default” nature of blockchains and public storage networks (ex. IPFS[5]) without requiring placing trust in a centralized custodian. With Lit, access control is achieved through a novel implementation of identity-based encryption[6]. Both encryption and decryption are handled client side and data regulation is enforced through the definition of programs formally referred to as “Access Control Conditions” (ACCs).

ACCs support the use of both on and off-chain data. On-chain data includes token ownership, DAO membership, and the result of any smart contract call. For example, using a “subscription” NFT to gate access to your creator profile on a decentralized social network. Only those who held your token could access your content, empowering owners with greater control over their personal data. This feature is blockchain agnostic, meaning state from any supported chain can be used. New chains can be added by adding the desired RPC configuration into the Lit node, which can be accomplished via decentralized governance.

Off-chain inputs are supported as well, which includes any data not stored on a blockchain that can be accessed via API. Just the same as above, this data can be used to define the certain conditions under which access should be granted. An arbitrary example would be pulling in data from a weather API and requiring the recorded temperature be above a certain predefined value in order for the content to be decrypted.

When a user requests access to encrypted data, a multi-node verification process is initiated across the Lit Network. Each node independently verifies whether the user meets the conditions set for accessing the data. Only when these conditions are met do the nodes create decryption shares for the user. The user then aggregates these shares until at least two-thirds of the total is reached to successfully decrypt the data locally on their device.

The complete flow for encrypting and decrypting data with Lit is as follows:

## Alice wants to encrypt

1. Alice specifies her access control conditions, ***A***
2. Alice hashes the data she wants to encrypt to get ***ID***
    1. This is an important security parameter that ensures if a given user ceases to satisfy ***A***, they are not able to decrypt future content.
3. Alice picks the message that will be signed to become the decryption key, calculated using hash(***A***) and ***ID***. We will call this ***D***.
4. Alice encrypts the data so that only a signature of ***D*** generated from the network is able to decrypt it.

---

[5] https://ipfs.tech/
[6] https://crypto.stanford.edu/ibe/

5. Alice stores ***A***, ***ID***, and the encrypted data wherever she wants.

## Bob wants to decrypt

1. Bob presents ***A*** and ***ID*** to the nodes and asks them to sign it
2. The nodes check access control conditions ***A*** and if he meets them. If he does, proceed.
3. The nodes create ***D*** (calculated using hash(***A***) and ***ID***), and sign it with the distributed network key to create the signature shares.
4. Bob combines the signature shares from the nodes to get the complete signature.
5. Bob can then decrypt the data using the signature as the decryption key.

### Use Cases

Some possible use cases for decentralized access control include:

1. **Encrypted data, stored in the public:** Use Lit to encrypt files, images, videos[7], archives[8], and other data for private storage on the dWeb.

2. **User-owned social and identity graphs**: Empower users to take full control over how their personal data and identity[9] is managed on the web through generalizable access control and selective disclosure.

3. **Credential-gated spaces**: Use token and credential ownership as “keys” to accessing exclusive spaces[10], content, and experiences, introducing additional utility for digital assets.

4. **Private NFTs**: Release NFTs with private embedded content that can only be accessed by the NFT owner themselves[11].

5. **Open data marketplaces**: Open data marketplaces facilitate the exchange of data between individuals and organizations, allowing users to buy, sell, or share information[12] in a secure and transparent manner. These systems promote data-driven innovation by making diverse datasets accessible to researchers, developers, and businesses, while also providing data creators with the opportunity to monetize[13] and permission access to their own content through customizable access controls.

---

[7] https://docs.livepeer.org/tutorials/developing/token-gate-videos-using-lit.en-US
[8] https://github.com/starlinglab/archive-explorer/
[9] https://www.lens.xyz/
[10] https://www.gather.town/
[11] https://blog.spheron.network/incognfto-a-private-nft-gallery
[12] https://streamr.network/
[13] https://index.network/

6. **Backup and recovery for private key material**: Use Lit to configure robust backup and recovery[14] solutions for private key material (such as multi-factor authentication or social recovery methods), helping users avoid the catastrophic loss of access to their assets due to lost or compromised keys.

## Serverless Signing and Programmable Wallets

### Serverless Signing

This feature gives developers the capacity to build serverless functions and application backends that have the ability to sign data with their own private key.

This functionality is provided through the generation of decentralized keys referred to as *Programmable Key Pairs (PKPs)*. Much the same as with access control, these keys are stored as shares across the Lit nodes, requiring consensus of at least two thirds of the network before executing a given action or signature. PKPs are inherently chain and platform agnostic, interoperable with blockchains and state machines using ECDSA for digital signatures.

Associated with these PKPs are *Lit Actions*, immutable JavaScript programs stored on IPFS that dictate each PKPs signing logic. Each Lit Action supplies “rules” that dictate when or under what conditions a given PKP should produce a signature, a system functionally equivalent to smart contracts that possess their own private key. Lit Actions also have the ability to use off-chain data through the use of the JavaScript “fetch”[15] function.

This can be used to facilitate condition-based automation within and across decentralized applications, as well as to generate proofs for verifying arbitrary data. A simple example would be a Lit Action and corresponding PKP that check if a provided input is divisible by two, meaning a signed response will only be returned if the number is in fact divisible. Since the Lit Action is immutable, and every signature requires participation from at least two-thirds of nodes, there is a provable chain of trust. Instead of having to do the math to ensure a number is divisible, you could simply use the number as an input in your Lit Action and use the signature as proof.

### Programmable Wallets

Wallets serve as the gateway to web3, providing a repository for digital assets and an endpoint from which to express your digital identity. Though critically important, current implementations suffer from many shortfalls. Current wallet solutions range from self-custody options, where users are solely responsible for their private keys, to centralized custodial services that manage keys on behalf of users (such as a CEX). While self-custody offers greater control and security, it introduces many challenges and frictions that are undesirable to the majority of users and stand as a barrier to widespread adoption (one such example being seed phrases). On the

[14] https://github.com/Joseph-Gross/key-recovery
[15] https://javascript.info/fetch

other hand, centralized solutions pose censorship risks and introduce an additional layer of trust, undermining the very ethos of decentralization.

Lit PKPs can be used to develop white-labeled MPC[16] “wallets-as-a-service”, offering a novel solution to many of the shortcomings described above. The two-thirds threshold provides a level of censorship resistance and fault-tolerance that typical Shamir-based[17] 2-of-2 MPC designs do not. In addition to any 2-of-2 provider being able to deny the user access to their funds or censor transactions, most of these systems also require the end user to custody a key share. This means the goal of a seamless, “web2” style onboarding UX is not possible (onboarding without seed phrases or private key management), instead delivering the UX of self-custody with additional steps.

With Lit, the entire key lives in the network and arbitrary logic can be assigned to the keys through the use of Lit Actions and smart contract logic. This gives the application developer or end user full control over designing how interactions with the MPC wallet should be managed. For example, enabling the use of social login or setting up multi-factor authentication (MFA) schemes for asset management. Examples of methods that can be used to interact with and create PKPs include WebAuthn from FIDO Alliance, Google oAuth, SMS, and email, all enabling more intuitive “seed-phraseless” onboarding flows.

## Use Cases

Potential use cases for serverless signing and programmable wallets include:

1. **Event listening and condition-based transaction execution**: Automate interactions with blockchain ecosystems using condition-based execution[18], enabling use cases such as on-chain limit orders or recurring payments that don’t require manual input (i.e. signing off on the transaction) from the end user.

2. **Native cross-chain messaging and swaps**: Seamlessly transfer assets and data across blockchain networks[19] without relying on a trusted intermediary or centralized asset bridge.

3. **Seed-phraseless wallet onboarding using “web2” authentication and sign-on flows (such as SMS, Discord OAuth, Passkey)**: Create easier onboarding experiences for non-crypto native users using familiar sign-on methods and authentication schemes. Abstract away seed phrases and complex private key management, while also providing the full web3 capabilities of an EOA[20].

---

[16] https://medium.com/1kxnetwork/wallets-91c7c3457578
[17] https://en.wikipedia.org/wiki/Shamir%27s_secret_sharing
[18] https://github.com/Sling-Protocol/pkp-dex-sdk
[19] https://spark.litprotocol.com/xchain-bridging-yacht-lit-swap/
[20] https://iglootools.xyz/

4. **Backup, recovery, and progressive self custody for account abstraction (AA)**: Use Lit to configure onboarding[21], backup, and recovery solutions for AA wallets (such as multi-factor authentication, social recovery, or oAuth), helping users keep their funds secure while avoiding the complexities of full self custody.

5. **Trustless vault applications**: Each wallet generated by Lit is represented by an ERC-721 token on the blockchain. This means that any assets sent to the wallet can be traded or sold in a single transaction by selling the NFT that controls the underlying key pair. This facilitates potential trustless "vault" applications where an array of assets may be managed together according to the rules associated with the key itself.

6. **Programmable verifiable credential issuance**: Verifiable credentials are digital certifications attesting to particular user attributes or qualifications. Using condition-based signing, the issuance of these credentials can be programmed[22] based on specific conditions being met, helping eliminate the possibility of fraud or human error.

7. **Enterprise "signed data" applications**: There are numerous use cases for "signed data" in institutional and enterprise environments. An example is a logistics company using digital signatures to authenticate and track goods down physical supply chains.

8. **Authentication for AI generated content**: A registry of identities and associated keys (which do the signing) in order to verify who made a given claim is neither ideal for privacy nor in line with the way that people use the Web. Threshold signing offers a unique solution to the data integrity problem, distributing trust among a set of parties to act as a "signer of last resort"[23].

9. **Powers of Tau as-a-service**: The Powers of Tau ceremony, also known as a "trusted setup[24]" is an essential part of initializing zkSNARK-based systems. This ceremony is used to generate a common reference string (CRS), basically a shared secret that the system uses to generate and verify proofs. By fusing TSS, MPC, and Secure Enclaves[25], the Lit Network becomes an ideal environment to facilitate such ceremonies, providing greater privacy, security, and integrity guarantees to participants.

# Protocol Architecture

The Lit Protocol architecture is composed of the Lit node and Chronicle rollup.

---

[21] https://app.patchwallet.com/
[22] https://spark.litprotocol.com/krebitxlitactions/
[23] https://spark.litprotocol.com/authenticity-matters/
[24] https://a16zcrypto.com/posts/article/on-chain-trusted-setup-ceremony/
[25] https://www.amd.com/content/dam/amd/en/documents/epyc-business-docs/white-papers/SEV-SNP-strengthening-vm-isolation-with-integrity-protection-and-more.pdf

# The Lit Node

## Node Protocol

The Lit node protocol operates as a Finite State Machine (FSM), with each node transitioning through a series of states as it interacts with and operates within the Lit network. Below is an overview of the protocol, focusing on the primary states a node can transition through:

### Initialization and Offline State

Every node begins in an "offline" state and transitions to "online" upon successful initialization. While offline, the node is essentially dormant and doesn't participate in any network activities. It can return to this state at any time, either voluntarily (unstake and leave) or involuntarily (slashing).

### Online State: The Polling Phase

When a node is "online," it actively polls the blockchain to check if it has been chosen as part of a validator set for the current epoch. If selected, the node moves to the "locked" state.

### Locked State: Staked and Committed

A "locked" node signifies a node that has staked (on Chronicle) and is part of an active validator set. It remains in this state until all other participants become "active" or a timeout occurs. If all conditions are met, including successful participation in at least one Distributed Key Generation (DKG) process, the node transitions to the "active" state.

### Active State: Fully Operational

In the "active" state, the node has successfully completed at least one DKG process and is fully operational and available to serve client requests for signing, encryption, and / or decryption. It remains in this state until the end of the epoch, unless it chooses to leave the network or the DKG process fails. If the DKG process fails and it's not this node’s fault, then it shall transition back to the “locked” state. If the DKG process fails and it is this node’s fault, then it shall transition to the “suspend” state and will lose its stake (slashing). Slashing conditions are outlined in the security section of this paper.

Nodes in the “active” state automatically become candidates for the next validator set and will remain in this state unless they opt out or until a new validator set has successfully completed a DKG.

### Suspend State: Temporary Exclusion

Nodes transition to the "suspend" state if they leave the network or are slashed. While in this state, they are ineligible to be selected for the next validator set until both the node owner and the network agree to let the node rejoin.

### Network Requirements and Fail-Safes

The Lit Network starts empty and becomes populated as nodes join and stake. The network employs robust mechanisms to securely choose the next validator set during epoch changes. If the network loses quorum due to a significant number of nodes leaving, it transitions to a suspended state, effectively falling prey to anarchy until the issue is resolved (see Backup and Recovery).

### Hardware

Lit makes use of AMD's Secure Encrypted Virtualization with Secure Nested Paging (SEV-SNP) as a hardware requirement to isolate key material and other sensitive data from node operators and other external agents carrying out the functionality of the Lit Network.

AMD SEV-SNP extends upon previous implementations of AMD's SEV technology, addressing limitations present in prior releases and further enhancing the security guarantees of the secure hardware. SEV is designed to provide an additional layer of protection for virtual machines (VMs) running on cloud-based servers by providing *hardware-based* memory encryption for each VM running on a shared server. These VMs are known as a "Guest", whereas the server itself is referred to as the "Host". Hardware-based encryption ensures that each Guest is isolated from each other, and from the actual Host system itself, which protects the data and applications that are running inside of the VM.

Each Guest uses main memory encryption to ensure that their in-use data remains private and secure. The CPU register state is further protected by encrypting the VM register on each hypervisor transition. While SEV has made leaps and bounds in helping protect the memory of Guest VMs, it does not guarantee the *integrity* of the data stored within memory.

Secure nested paging (SNP) addresses this issue by adding additional hardware-based security protections, providing strong memory integrity guarantees that prevent malicious hypervisor-based attacks like data replay and memory re-mapping. This creates a fully isolated execution environment, increasing the Guest's resiliency to side-channel attacks[26], which may attempt to exploit information gained from the physical implementation of an operating system.

## The Lit Rollup: Chronicle

While Lit is not a blockchain itself, the protocol does contain several on-chain elements, namely smart contracts that have been deployed to an OP stack[27] rollup called Chronicle. The primary function of this rollup is to help coordinate and manage shared state among the nodes participating in Lit Network validation.

Chronicle facilitates the minting and management of new Programmable Key Pairs, which exist on-chain as ERC-721 tokens. Every time a new PKP is minted, a key derivation process is triggered within the node. This will be explored further in subsequent sections. In addition to the

[26] https://csrc.nist.gov/glossary/term/side_channel_attack
[27] https://stack.optimism.io/

PKPs themselves, the specific permissions and authentication logic associated with each key pair is also registered on-chain. Every time a signing request is initiated by a client, the nodes verify that the applicable permissions are met by querying the rollup.

Node operator staking is also handled on Chronicle, which is tightly coupled with the Lit nodes' FSM-based architecture (described in the previous section). Staking dictates the active validator set, refreshed on an epoch-by-epoch basis.

In addition to PKPs and staking, Chronicle is also used by developers who are paying for the service provided by Lit. A rate limiting system is used to manage consumption of network resources where costs vary according to network demand and request type.

Chronicle itself is fully EVM compatible while also supporting unique features such as BLS 12-381 precompiles, which enable BLS signatures to be verified on-chain. This functionality is not yet supported on Ethereum.

### Chronicle Nodes and Sequencing

All Lit Node operators also run a Chronicle node. This speeds up the retrieval of blockchain data and tightly syncs on-chain and off-chain operations, enhancing overall network performance. Furthermore, this setup ensures high levels of data redundancy and holds the sequencer, currently operated centrally by the Caldera team, accountable. In essence, each node acts as a check and balance on the sequencer, ensuring that transactions are valid and not being censored. Efforts to decentralize the sequencer are ongoing.

# Cryptographic Primitives

## Digital Signatures

Each Programmable Key Pair is generated collectively by a set of participants using a DKG. The underlying protocol supports DKG among an arbitrary number of participants while also providing the ability to modify the desired participation threshold (the "default" threshold is set to two-thirds of nodes), creating flexibility for a wide array of potential implementations. Lit supports ECDSA with secp256k1 and BLS signatures in G1 with BLS12-381 curves with plans to expand to support ECDSA with secp256k1 or prime256v1 and BLS signatures in G2 with BLS12-381 and schnorr signatures with secp256k1 and ed25519.

### Overview of Key Generation

The protocol begins with key generation, carried out by an interactive DKG process that requires participation from each node. The end result of the DKG is a state where each node only holds a *share* of the underlying private key (a private key share) and collectively the same public key.

The DKG protocol is divided into several phases, each involving synchronous communication among participants. During the first phase, participants agree on the specific DKG parameters to be used, such as the number of participants and desired threshold. Once the parameters and participant set are established, each node generates their own private key share (S) and derives the associated public key (P). Nodes then share their public keys, known as "verification vectors," with other participants and create private key contributions for each using S. These contributions are encrypted and distributed to the corresponding nodes.
Each node verifies the received private key contribution and verification vector. Then, the DKG is executed for the required keys used by the system.

In case of failure, the affected node issues a complaint to the responsible node, alerting other nodes of the issue. Participants who receive complaints have an opportunity to publish a justification, containing the unencrypted private key contribution initially sent to the complaining node.

After resolving all complaints, the verification vectors of valid participants are aggregated to create a shared verification vector and public key. A preliminary commitment message is published, signed by each node's private key share. Valid commitments are then collected and grouped based on identical parameters. If a group has enough commitments (i.e., greater than the threshold), a final commitment is created using the aggregated signatures from the first type and a recovered signature from the second type.
The current DKG method is based on Gennaro et.al.’s work in Secure Distributed Key Generation for Discrete-Log Based Cryptosystems with additional work looking into asynchronous methods like BINGO.

## Signature Generation

Signatures are computed using the Cait-Sith method which models security using Modular Protocol Security (MPS, like UC security, in essence). After generating the key, the signing process is several stages, beginning with the formation of beaver triples. All nodes concurrently initiate this interactive process. This creates the necessary computational data ahead of time such that it’s not dependent on the key to be used for signing. This is done before any client request and even before the actual message to be signed is known. By doing so, the system can quickly fulfill signing requests, enhancing both its efficiency and overall performance.

The final rounds of signing occur upon receipt of the message. In this stage, a beaver triple is used to create the complete signature and sign a given message.

## Key Refresh and Resharing

Periodically, key shares are changed with a refresh scheme. This rotates the private key shares among an existing set of participants without changing the underlying private key, also known as proactive secret sharing. This method helps ensure the integrity of private key material for long periods of time and minimize the risk of key compromise.

A key refresh protocol is performed by first linearizing the shares, running the same DKG algorithm with existing participants, then checking that the public key doesn't change.

Key resharing includes the additional ability to update the set of participants used in the key management process. This scheme allows nodes to leave and join without disrupting the service of the network and making the protocol highly adaptable to varying requirements and group dynamics within the Lit Key Mesh. A key resharing protocol can be performed as well. This involves transitioning from the existing participant set to the new participant set, and can be performed as long as there are enough old participants with shares. The end result is that the new participant holds threshold shares of the same private key.This works by having existing participants linearize their shares, and new participants set their share to zero and run the same DKG while checking that the same public key is generated.
Key refresh can be seen as a natural case of key resharing, with the participants and threshold potentially changing.

## Encryption

Lit's encryption access control protocol relies on identity based encryption and BLS signatures. Thus a network-wide BLS key is generated by participating nodes using a DKG when the system comes online. The public key acts as the encryption key. When a user encrypts with this key, the decryption policy serves as the identity. This identity is signed when decryption is executed. Since the identity serves this purpose, the nodes check if the policy requirements are met prior to creating a signature share. If the requirements are satisfied, the node signs the identity and returns their signature share which serves as a decryption share. Once enough shares have been collected, the user combines them to yield the signature or decryption key and can decrypt the ciphertext.

## Key Refresh and Epoch Advancement

Each epoch changeover (initial key generation, refresh, or recovery event) has a unique identifier and deadline, determined upon locking in the chosen validator set for the period (as detailed in the node protocol section). If the epoch does not advance by the deadline, anyone can call a function to return the contract and nodes to their unlocked state and discard the changeover in progress.

During the changeover, the key generation process is repeated and recorded. Upon successful epoch advancement, the node replaces the previous DKG results with the new data. If the advancement does not occur by the deadline, the pending changes are discarded, and the process restarts. This approach ensures the system remains secure and functional, even if the epoch changeover does not proceed as expected.

# Security Considerations

## Backup and Recovery

Maintaining the resilience and robustness of the Lit Network is paramount. If too many nodes go offline such that the network loses its threshold, it's essential to have a recovery mechanism. This ensures Lit can always meet the necessary threshold of active nodes, especially if some node operators become unresponsive.

### Verifiable Backups

All keys (except the BLS encryption key) in the Lit Network are derived hierarchically from a set of root keys. During the DKG, each Lit node generates and holds a share of these keys. To ensure these key shares are safe and can be recovered as needed, they're backed up regularly. The backups are created using verifiable encryption, a protocol that ensures the ciphertext meets certain properties such as the encrypted value is an encrypted key share and not anything else, which allows Lit to confirm that the encrypted root key shares are genuine i.e. that the ciphertext encrypts a valid share and not non-related data. The original idea for this was given in 2003 by Jan Camenisch and Victor Shoup. This approach however uses a combination of El-Gamal encryption with discrete-log ZKPs i.e. Verifiable Encryption of ECDL but facilitates decryption of the actual value. Since El-Gamal encryption leaves the value in-the-exponent and would thus be impossible to find, the key share is broken in bytes and encrypted with El-Gamal to enable reconstruction when decrypted. To ensure the bytes are actual bytes or ≤ 256 bits, bulletproofs are used to verify the value’s range.

Every time new root keys are produced, nodes update their stored backups with the new data.

### The Recovery Party

To assist with recovery, there is a designated group called Guardians. Guardians play a crucial role in the recovery process by facilitating the root key share decryption process. More than two-thirds of the Guardians must participate to successfully decrypt the stored backups.

### Encrypting the Backups

The process for encrypting the root key backups involves:

1. The Guardians create a public encryption key and the decryption key shares with a DKG.

2. This public key is used for encrypting the root key backups.

3. The encrypted backups are stored on IPFS.

### Recovery Process

If the network needs to be restored:

1. Guardians validate their identities with a Lit node and produce decryption shares for the required root key shares.

2. Node operators spin up a new node environment and input their encrypted backup, and petition the Guardians for the decryption shares.
   Decryption key shares are combined to fully decrypt the ciphertext backups.

3. With the decrypted root key shares in the new node environment, the network can be restored.

# Scaling Lit

The Lit Network scales both horizontally – through clusters – and vertically through the establishment of subnets.

### Clustering

Clustering allows node operators to replicate their key shares across multiple instances, increasing transaction throughput and performance, as well as strengthening overall data availability and resiliency. Each instance that forms part of a cluster is identical, however one of them assumes the role of the entry-point (or load balancer). This is achieved by assigning a floating IP to the cluster and nominating one of the instances to be the bearer of this IP. If that instance should go offline, one of the other instances will take over bearing the IP automatically. This IP is the one that is recorded on-chain and is discoverable by clients.

Requests are terminated on this main instance and directed to the other instances in the cluster in a “round-robin” fashion. Each cluster automatically replicates their key shares amongst every instance. This means that each instance has the capacity to handle any request and guarantees data redundancy.

### Subnets

Lit can also be scaled by adding new networks into a more expansive architecture known as the Lit Key Mesh. Each subnet within the Key Mesh is ‘app-specific’, holding a distinct set of node operators, functionality and security assumptions. Unlike a monolithic, "all-encompassing" network, these subnets allow for greater flexibility, specialization, and scalability while catering to distinct use-cases and regulatory environments.

Some potential examples of subnets include:

- A network for EVM-based DeFi signing vaults specialized in automation and private orders.
- A network where all node operators are part of a particular industry consortium.

- A network that only signs transactions from accounts that hold a KYC NFT.
- A network focused on signing data destined for decentralized storage providers like IPFS or Ceramic.
- A fully insured network for managing digital assets.

# Conclusion

In an era where digital identity continues to transform from a mere digital footprint into a valuable and often commodified asset, the need for enhanced security, decentralization, and user agency has never been more important. The prevailing challenges of centralized platforms and their monopolistic tendencies have underscored the systemic risks and inherent power imbalances that arise from these traditional “web2” models. While web3 technologies have presented a beacon of hope for truly self-sovereign identity management, their complexities have continued to render them inaccessible to a vast majority of users.

Lit stands out as a pioneering solution, addressing the challenges that have plagued both web2 and web3 landscapes. By leveraging state-of-the-art cryptography and hardware-level security measures, Lit Protocol champions a first-of-its-kind decentralized key management network to enable each individual and application builder to become a beacon of digital independence. Lit provides web3 native identity that embodies the shift from web2, where everyone is merely a user, to a new web, where every individual is a sovereign entity that owns their assets and data.

As the digital landscape continues to evolve, Lit remains steadfast in its mission: to democratize digital identity management and Free The Web to the control of its users.

# References

[x] Michael Backes, Aniket Kate, and Arpita Patra. Computational verifiable secret sharing revisited. In International Conference on the Theory and Application of Cryptology and Information Security, pages 590–609. Springer, 2011.

[x] Joshua Baron, Karim El Defrawy, Joshua Lampkins, and Rafail Ostrovsky. Communication-optimal proactive secret sharing for dynamic groups. In International Conference on Applied Cryptography and Network Security, pages 23–41. Springer, 2015.

[x] 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.

[x] Ran Canetti, Rosario Gennaro, Steven Goldfeder, Nikolaos Makriyannis, and Udi Peled. UC non-interactive, proactive, threshold ECDSA with identifiable aborts. In Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security, pages 1769–1787, 2020.

[x] 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.

[x] Rosario Gennaro and Steven Goldfeder. One round threshold ECDSA with identifiable abort. IACR Crypto. ePrint Arch., 2020:540, 2020.

[x] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. Journal of Cryptology, 20(1):51–83, 2007.

[x] David A Schultz, Barbara Liskov, and Moses Liskov. Mobile proactive secret sharing. In Proceedings of the twenty-seventh ACM symposium on Principles of distributed computing, pages 458–458, 2008.

[x] Jens Groth and Victor Shoup. On the security of ECDSA with additive key derivation and presignatures. IACR Crypto. ePrint Arch., 2022.

[x] Jens Groth and Victor Shoup. Design and analysis of a distributed ECDSA signing service. IACR Crypto. ePrint Arch., 2023.

[x] Gennaro, R., Jarecki, S., Krawczyk, H. et al. Secure Distributed Key Generation for Discrete-Log Based Cryptosystems. J Cryptology 20, 51–83 (2007). https://doi.org/10.1007/s00145-006-0347-3

[x] Abraham, I., Jovanovic, P., Maller, M., Meiklejohn, S., Stern, G. (2023). Bingo: Adaptivity and Asynchrony in Verifiable Secret Sharing and Distributed Key Generation. In: Handschuh, H., Lysyanskaya, A. (eds) Advances in Cryptology – CRYPTO 2023. CRYPTO 2023. Lecture Notes in Computer Science, vol 14081. Springer, Cham. https://doi.org/10.1007/978-3-031-38557-5_2

[x] Lúcás Críostóir Meier. Cait-Sith Security, https://cronokirby.com/notes/2023/04/cait-sith-security/

[x] Lúcás Críostóir Meier. Towards Modular Foundations for Protocol Security. https://eprint.iacr.org/2023/187

[x] Bulletproofs: Short Proofs for Confidential Transactions and More , Benedikt B¨unz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille and Greg Maxwell, IEEE S&P 2018

[x] Practical Verifiable Encryption and Decryption of Discrete Logarithms , Jan Camenisch and Victor Shoup, CRYPTO 2003