Lisbon Account Protocol: Practical Distributed Externally-Owned Accounts

with Key Resharing and Public Key Encryption with Forward Secrecy

Amira Bouguera, James Bourque

February 22, 2023

Abstract

Decentralized networks such as blockchains rely on asymmetric keypairs for external participant addressing, known as externally-owned accounts, or EOAs, and require end-users to safely maintain private keys. This private key management often becomes a single point of failure for end-users, and dictates a poor user experience, particularly for a mainstream audience. As a result, many solutions have been put forward to reduce the use or significance of private keys in these networks, including the use of Smart Contract Accounts for account abstraction, variations of custodial services, or implementing sharing or signature schemes. These solutions, however, often unacceptably compromise on core network values, such as decentralization, self-custody, or interoperability/composability.

In response, we propose a practical account protocol that is non-custodial, fully decentralized, non-interactive, and asynchronous. The result of the protocol is an externally-owned account with no single private key for use on decentralized networks. The protocol includes three main phases: distributed key generation, threshold signature formation, and key resharing. The protocol supports both ECDSA and BLS keypairs, is network-agnostic, and requires no third party dependencies, or additional data stored by end-users locally or otherwise. It has been designed for real-world implementation, and supports many on- and off-chain environments.

Keywords: Secure Multiparty Computation. Threshold Signature Cryptography. Key Resharing. Distributed Key Generation. Verified Secret Sharing.

Contents

1 Introduction **2**
1.1 State of the art 3
1.1.1 Technology 4

1.1.2 Similar Solutions . . . 7
1.1.3 Our solution . . . 9

2 Lisbon Protocol 10
2.1 System Context Overview . . . 10
2.2 Distributed Key Generation (DKG) . . . 11
2.3 Threshold signature . . . 13
2.3.1 ECDSA Threshold signatures scheme . . . 14
2.3.2 BLS Threshold signatures scheme . . . 16
2.4 Key Resharing . . . 17
2.5 Public key encryption with forward secrecy . . . 18
2.5.1 CCA-secure public-key encryption with forward secrecy . . 18

3 Threat Model 19
3.1 Malicious behavior . . . 19
3.2 Man In the Middle . . . 20
3.3 Key compromise . . . 20

4 Conclusion 21

5 Appendix 23

1 Introduction

As Web3 continues to evolve, it is storing and securing digital assets of increasing value and complexity. At the outset, blockchain networks' value was rooted in simple tokens and smart contracts. Web3 now includes sophisticated financial protocols such as staking and yield farming, digitally unique assets such as NFTs, and Digital Identity, all which have much greater financial and social value than before.
All participation in Web3 requires a cryptographic keypair, however, the means for storing and utilizing keys have evolved little since the introduction of blockchain networks. Existing key management solutions fall into three broad categories:

- Original blockchain solutions, such as hardware wallets, mnemonic phrases, and wallet applications
- Web2 Solutions, such as Hardware Security Modules and centralized, custodial services
- Blockchain Solutions (1st generation), such as vault services or smart-contract wallets

Each of these solutions have categorical short-comings for high-value accounts and assets, such as single points-of-failure, dependencies on specialized networks or hardware, or centralized/custodial services.

As the value of individual accounts' assets increases and network value/lost assets, so does the need for a fully-decentralized key management protocol.

The protocol described in this document addresses many of these existing vulnerabilities in a fully decentralized, trustless, network-agnostic way, with the express purpose of serving as an accessible and democratized solution for the Web3 community. It solves the single point of failure (private key) by introducing three key characteristics without cryptographically impacting a static public address or identifier:

- Decentralized Key Generation across user devices, to minimize hardware risk
- Distributed ownership and risk, to reduce dependence on a single key
- Account recovery, through both key resharing and interpolation

1.1 State of the art

Our protocol relies on threshold cryptography which allows for both distributed key generation and signature. Threshold cryptography allows a group of participants, identified by externally-owned accounts to hold a key and use it for signature or encryption. A single individual keyholder cannot recover the key on their own nor use it for encryption or signature. In order to recover the key, there must be a minimum number of accounts, a "threshold" number working together to access the key and use it. However, threshold cryptography isn't the only technology allowing for such usage, multisignature schemes allow that as well.

Another cryptographic method used in our technology, MultiParty Computation (MPC). The goal of MPC is for a set of parties to jointly compute a function over their inputs each on a separate device without revealing anything but the output. MPC has been a topic of research in academia since the 1980s and has recently become efficient enough to be used in practice. There are many potential applications where MPC is useful including key generation in a decentralized fashion.
In this section, we will give the current state of the art by comparing threshold cryptography and MPCs each to similar technologies that solve similar problems. Threshold signatures to Multisignature schemes and MPC to HSMs with the goal of identifying both advantages and disadvantages of each and justify our choice for using these technologies. We will compare key generation and storage solutions (MPC and HSMs), as well as means of utilizing these keys (Threshold and Multisignature Schemes) with the goal of identifying advantages for each, and ultimately justify our design decisions in how we apply these technologies. We will also examine similar projects and their approaches building in the blockchain space to identify the strengths and weaknesses of their methodology.

1.1.1 Technology

MPC vs HSM

Hardware security modules, HSMs, are dedicated computing devices designed for the safe storage and use of cryptographic keys. HSMs have the ability to wipe all key material when they detect attempts to interfere with the device which makes them a good choice for storing private keys. They also allow for faster decryption and thus more real-time access to assets.
Multi-party computation, MPC, is a software solution, based on cryptographic protocols, that give the ability to cooperatively compute a function while keeping each participant's inputs secret without exposing them to other participants. MPC allows several independent parties to cooperatively share ownership and use of a cryptographic account in a variety of different environments, without reduced hardware risk.

Each of these technologies has its advantages and disadvantages and the choice between both will depend on the use case. HSMs protect against external threats since authentication takes place inside the module. Also any attempt to tamper with HSMs leaves a digital trail. However, HSMs have several major drawbacks including slower transaction time and scalability issues as every client must have one or more HSMs, this condition makes the protocol harder to scale as it becomes more expensive to participate. HSMs also have limitations in redundancy and liveness and require physical access for deployment, maintenance and configuration. A single module can protect only a limited number of keys and deployment cannot be automated.

MPCs scale better than HSMs as there is no need to have a hardware device to participate. Key shares remain separate between all the parties and unless the majority (a threshold that is decided depending on the protocol and the level of security needed) colludes together the key is never assembled in one place which means the key stays secure. This is true even when a quorum assembles a key to make a transaction: the key is never assembled on a single machine. MPCs allow for m-of-n parties to co-sign transactions which is similar to traditional multisignature but without the inconvenience of slower and heavy transactions that comes with multisignature schemes since MPC transactions are smaller, faster and incur lower on-chain transaction charges.
One of the biggest challenges in MPC technology is the ability to store key shares securely. The most effective way to do so is by storing keyshares on HSMs, hence these two technologies are complimentary and not exclusive of each other. Another challenge which is sometimes seen as a property is the fact that signers identities stay hidden: this might increase the potential for unethical collusions, it can also be seen as a privacy property that protects identity of signers from attackers who try to influence or hack them.

Threshold Signature Schemes vs Multisignature

Both schemes enable multiple parties to sign transactions together, either by using secret shares of one key to produce a single signature (threshold signatures) or by allowing a group of signers (each possessing its own private/publi key pair) to produce a single signature on a message m and verifying the validity of that signature against the set of public keys of all signers. Both methods allow for distributed nature of signature creation but they differ in the following points:

In TSS (threshold Signature Schemes), signing occurs off-chain and we see a single-key transaction on chain (the aggregated signature), this is why participants' identities are hidden. On the other hand, Multisignature Schemes use an on-chain process, which means that the identities of participants are publicly visible on the blockchain. This introduces security risk by exposing the identities of who is involved in the process. In addition, because Multisignature transactions happen on-chain, the transaction costs increase as well while in TSS, the cost is low due to their off-chain nature.

There are two types of multisgnature schemes that we should mention: first type are multisignature smart contracts: in order to send a transaction from the contract a majority of signers need to approve it. The second scheme is based on Schnorr signatures commonly used in Bitcoin, this scheme allows for combining multiple signatures into one through aggregation, while still producing a small on-chain size signature with faster validation and better privacy than the smart contract multisignatures scheme. The problem with both though is that each one of these schemes necessites each participant to have their own private key stored in one place and thus the single point of failure problem isn't solved.
Threshold signature aggregation for BLS or ECDSA is an example of TSS which ensures that private keys never exist on a single system at any point in time. Secret shares are generated independently on each participants' device and the signing algorithm uses secret shares in turns, without the need to reconstruct private keys.

Threshold signature, Key resharing and Rotation

We will focus on the usage of MPC algorithms to compute a digital signature in a distributed way. This process has three steps: Distributed key generation, Signing and verification. The two DKG (Distributed Key Generation) protocols mostly used nowadays are the Joint Feldman DKG (JF DKG) (1991) and Gennaro et al.'s DKG (2006). Many protocols have been developed as a variation of these two. To use one or the other, one had to do a tradeoff between efficiency or security. JF DKG is more efficient in computation, storage, and communication complexity. On the other hand, Gennaro et al.'s DKG protocol

is more secure and resolves a known attack on the JF DKG protocol, which prevents the public key pk from being uniformly distributed. Gennaro et al's scheme wasn't efficient because it needed the participation of $2t + 1$ players to produce a signature where n is number of participants and t is threshold value.

Gennaro and Goldfeder have developed a threshold signature scheme (GG18)[1] that enables only $t + 1$ to sign which was a huge improvement over the previous $2t + 1$ by Gennaro et al. The GG18 was the first scalable ECDSA TSS protocol with no trusted dealer and was considered the industry standard by many. However, with GG18's algorithm, the communication latency between the MPC-shares goes up to 9 signature rounds. The authors of GG18 have since developed a faster, more scalable version (GG20)[2] that reduces the number of rounds to one-round signing instead of 9, while adding significant functionality: identifiable abort and noninteractivity.

The non-interactivity property means participants don't need to exchange information during rounds of communication and whatever information needed to complete the protocol will be either available publicly or computed offline in a preprocessing round. This is an interesting property as it decreases the number of communication rounds and thus makes the protocol more efficient. The CGGMP21 protocol by Canetti et al[3] also known as MPC-CMP is considered the newest innovation in MPC and was inspired by Gennaro and Goldfeder's design. It enables digital asset transactions to be signed in just 1 round, the same as GG20.

The team behind this protocol has identified a security issue with the GG18 and GG20 protocols, showing that some information related to the private key can be leaked and showed a mitigation method using their security model. However, the vulnerability had not yet proven to be exploitable. CGGMP21 has two versions: the 4 round "online" (interactive) and the 7 round "presigning" (non-interactive). The signing process in the first version can be split into two phases: A first, preprocessing, phase that takes 3 rounds and can be performed before the message is known, followed by a non-interactive step where each signatory generates its own signature share, after the message to be signed becomes known. Only the last round of the protocol requires knowledge of the message, and the other rounds can take place in a preprocessing stage, lending to a non-interactive threshold ECDSA protocol.

Both CGGMP21 and GG20 provide a security property called "identifiable abort", which essentially means that if the protocol fails to generate an output to all honest parties, one corrupt party will be reliably identified while assuming a dishonest majority. This property helps in detecting the identity of malicious parties that have failed to participate in the generation of the signature in the case of a non valid signature.

The table below lists the different MPC protocols and highlights their capa-

bilities in terms of efficiency (number of rounds), cold storage compatibility and non-interactivity.

Algorithm	Transaction Rounds	Non-Interactivity	Cold Storage
GG18	9	No	No
Lindell et al.	8	No	No
Doerner et al.	6	No	No
GG20	1	No	No
CGGMP21	4	No	Yes
CGGMP21	1	Yes	Yes

Another very important property that wasn't mentioned in the table above is the asynchronous communication. In the asynchronous model, there is no need for all participants to be online for message delivery and communication happens over a period of time (a bound can be set) while synchronous communication takes place in real time. We consider the synchronous model to be an unrealistic one for a decentralized setting where nodes are distributed around the globe. Jens Groth has written a paper describing an asynchronous non-interactive distributed ECDSA signing scheme [4]. This protocol guarantees output delivery which means that corrupted parties cannot prevent the honest parties from receiving output in any case. This is true in a system where we assume an honest majority. If a protocol guarantees output delivery, then the parties always obtain output and cannot abort. All of the protocols in [3] and [2] cannot guarantee output delivery while providing identifiable aborts because if a single node in the network loses network connectivity (or crashes) for a period of time that could be considered as an abort of the protocol. The notion of "identifiable abort" does not translate to the asynchronous communication model, as there is no way to differentiate between a corrupt party that is unresponsive and an honest party with a slow network connection.

1.1.2 Similar Solutions

We can divide similar technologies into two categories

1. **Custodial or Non-Custodial** Custodial means another party controls customers' private keys and commits to securely store their funds. Most custodial solutions (centralized exchanges for example) hold funds in cold storage (hardware wallets or hardware security modules) which are highly secure. The alternative -non-custodial solutions - do not store customer's private keys, providing them full ownership and responsability of their account. This is often managed by software such as wallets, or hardware wallets. While this method may sound more secure than the first one as it does not require trusting a third party with sensitive information, it requires customers to trust themselves with that information. However, if the customer loses access to their wallet, key management software or

forgets their password or recovery seed phrase, there are no options for recovering that account key pair. This is why many prefer to use a custodial solution as it does not require as much responsibility on their individual, and is typically more convenient. even if this decreases decentralization by introducing dependencies on the custodian.

2. **Network supported solutions** We can think of this category as Decentralized custody which relies on MPC technology as a building block to manage private keys. as it does not hold users' keys directly but can provide validator nodes. Decentralized custody removes the single point of failure of the private key, making theft or loss almost impossible. This is guaranteed because private keys are divided into multiple shares and stored with different entities in different locations (as explained previously) which makes the key secure. This is only possible if not more than t (threshold) validator nodes are controlled by a single entity otherwise the system will fail when that entity is hacked, malicious or ceases to exist.

   Most actors in the space who use MPC technology to secure key storage and allow threshold signature while providing validator nodes aren't transparent about how many nodes they actually control. There is an approach to collect validators shares called dWallet (decentralized wallet) where the user creates and stores their own share locally and every validator of the network will do the same. The validators' shares constitute the blockchain share which when combined with the user's creates the user's private key (the key is never actually generated). Validators shares will be used to create a partial signature which is stored on the blockchain and the user will also generate their partial signature and queries the blockchain signature to finally generate a valid signature and broadcast it to the target blockchain.
   The problem with such an approach is that it relies heavily on the solution provider's blockchain network and if that last ceases to exist, users won't be able to access their private keys anymore nor sign transactions. The fact that the network validators hold these shares introduces a reliability and trust issue where users trust their blockchains will continue to exist and no risk of ddos attack against their validators or a collusion.

   Most validator nodes use Hardware Security Modules (HSM like intel SGX) to store encrypted shares. This solution is more secure than other private key storage methods like hot and cold crypto wallets which have a single point of failure as every key share is stored in a different enclave. However Hardware solutions like Intel SGX can be dangerous in some cases. In the computation context, data should be protected from any type of modification or access. However, most Intel Hardware chips have been previously found to be vulnerable to some attacks (Meltdown, Spectre and Foreshadow). The first two attacks (Meltdown, Spectre) allow an attacker to access private data by misleading speculative scouts

into a speculative execution attack. Compared to other Intel Hardware chips, the SGX is resilient to speculative attacks but unfortunately, it is vulnerable to another type of attack called Foreshadow. This vulnerability enables an attacker to create a shadow copy of the protected data into a different unprotected location. Another reason why hardware approaches are not privacy preserving is that data is encrypted on the client side with the public key of the server (which is an untrusted party), instead of being encrypted with the client public key. This system requires data within the enclave to be decrypted before being processed. If an attacker manages to access the enclave, it will then be easy to recover plain text data. Last but not least, hardware enclaves introduce centralization risks and limits availability as it becomes more expensive to be a validator of the network and thus these solutions don't scale.

1.1.3 Our solution

Our proposed solution - the Lisbon Account Protocol - consists of four principal phases, which are described in more detail in the following section:

- Decentralized Key Generation, including a pre-registration phase, which utilizes Secure Multiparty Computation (SMPC) and Verified Secret Sharing (VSS)
- Threshold Signature Formation, supporting both ECDSA and BLS Signatures
- Key Resharing and Rotation, for account recover, participant rotation, and proactive security
- Public Key Encryption with Forward Secrecy, enabling on-chain data storage and inter-participant communication on public channels without requiring additional data or passwords for end-users

The protocol is designed to be fully decentralized, utilized in public blockchain environments, with no additional dependencies. The protocol creates asymmetric keypairs, but only returns the public address, with the private key distributed across qualified participants as key shares. These shares are used to form threshold signatures, which are verified on-chain using a smart contract.

We consider the use of private or gated networks an introduction of an additional point of failure; should the network be unavailable or cease to exist, the end-user is incapable of recovering shares and forming transactions. We believe this is an unacceptable compromise in return for eliminating a single private key point of failure. We also consider dependence on hardware security modules or trusted execution environments an additional barrier to entry for most users, as well as a limited factor in scaling adoption and use.

Ultimately, the protocol achieves the following characteristics:

- No Single Point of Failure
- No additional dependencies
- Broad compatibility and availability
- Minimized hardware risk
- Forward Secrecy
- Account Recovery and Distributed Ownership

2 Lisbon Protocol

2.1 System Context Overview

Generally speaking, the protocol requires only two components: a local light client and access to a public, EVM-compatible blockchain. Individual participants in the protocol are expected to bring their own EOA keypair, but are free to manage and utilize this as they wish. This keypair is their identifier, and is utilized in account creation, communication, and signature formation.

The protocol assumes all communication between participants are public and unsecured, and thus make use of public key encryption with forward secrecy. These encryption keys are derivatives of participants' EOAs, and require the individual participant to store no additional data or passwords.

All data relevant to the use of Lisbon EOAs are safely stored on-chain, with the same encryption employed in communication. This includes Lisbon shares, information for public verification of signatures and Lisbon Account participation, and qualified participants in the Lisbon Account. This encrypted data can be cached and stored locally, and can always be recovered from the on-chain record.

The majority of computation in the Lisbon Account creation and threshold signature formation occurs locally on a participant's device, and is performed by the light client. The light client, developed by INTU, is available as an SDK for developers, for integration into browser-based applications, as well as offline executables. This includes the pre-computation that enables the protocol to limit communication rounds and satisfy asynchronous, non-interactive requirements.

The corresponding smart contract system serves three purposes: coordinating communication between participants in the absence of offchain channels, storing critical data for both the Lisbon Account and the participants, and cryptographic validation in account use.

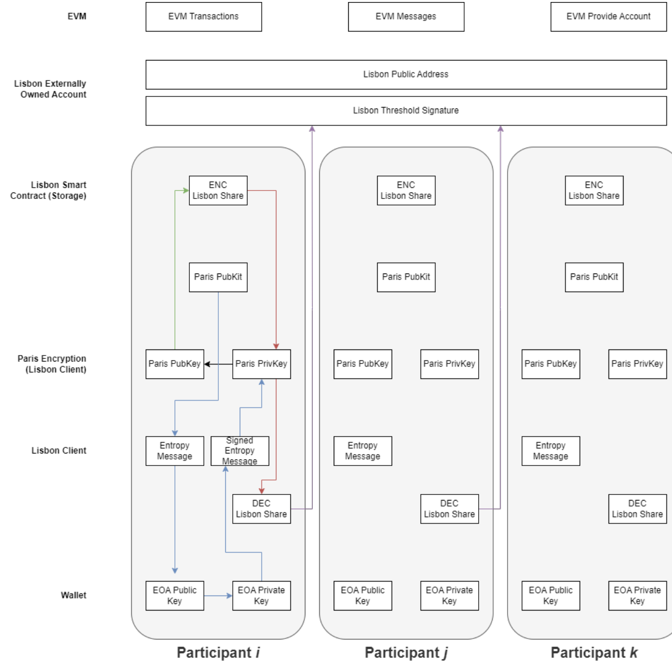

Figure 1: System Context Diagram

2.2 Distributed Key Generation (DKG)

Our key generation protocol follows the description in the Canetti paper [3]. The steps are as follow:

- **Pre-registration phase** During this phase, proposer P_i initiates the protocol by submitting other participants EOAs, choosing a threshold t and a generating random message msg. The smart contract is then created and participants who wish to be part of the vault will each sign msg and create their Paris key pair which is stored in the smart contract (Paris is an asymmetric encryption key scheme that we developed for the purpose of on-chain storage of shares, will be explained further in the document). Once each participant signs msg, and create their Paris key pair, the proposer, (or anyone involved with the smart contract at this moment), can start the key generation process.

- **Sharing phase** Each $P_i \in P$ (where P is the set of participants) generates their secret share $s_i \in \mathbb{F}_q$ for $\mathbb{F}_q$ a finite field with q elements and broadcasts it to all the other participants. The share is sent encrypted using Paillier encryption which is used as a commitment scheme. In order for other participants to verify whether P_i knows the secret exponent s_i or not, P_i publicly shares a Schnorr proof of knowledge.

As a security measure, Canetti explains that each participant P_i must commit to (S_i, A_i) to prevent any adversary from choosing the master public key $S = \sum_j S_j$ as a function of the honest participants's keys, where $S_i = s_i g$ is their public key share for g a generator of the group $\mathbb{G}$ of prime order q and A_i is the Schnorr proof's first message.

Let $\prod^{sch}$ bet the Schnorr PoK (Proof of Knowledge) that takes as input $(\mathbb{G}, q, g, S)$ such that the prover has a secret input s where $sg = S$. Prover sends $A = \alpha g$ for $\alpha \in \mathbb{F}_q$ to verifier who then replies with another value $e \in \mathbb{F}_q$. Prover sends $z = \alpha + es$ to verifier who checks the equality $zg = A + eS$.
The key generation protocol works as follows:

1. Every participant P_i for $i \in \{1, n\}$ generates a secret share s_i such that $s = s_1 + ...s_n$ and set $S_i = s_i g$ where n is number of total participants. P_i samples $srid_i \in \{0,1\}^k$ and computes (A_i, τ) for A_i Schnorr PoK's s first message. P_i samples $u_i \in \{0,1\}^k$ and set $V_i = H(ssid, i, srid_i, S_i, A_i, u_i)$ where $ssid = (..., \mathbb{G}, q, g, P)$. Each participant P_i brodcasts $(ssid, i, V_i)$.
2. For every P_i such that $j \neq i$ upon receiving $(ssid, j, V_j)$ from all P_j, sends $(ssid, i, srid_i, S_i, A_i, u_i)$ to all participants.
3. Each P_i now verifies the following equality:

$$H(ssid, j, srid_j, S_j, A_j, u_j) = V_j$$

   after receiving $(ssid, j, srid_j, S_j, A_j, u_j)$ from P_j. Next, P_i sets $srid = \oplus_j srid_j$ and computes

$$\psi_i = M(prove, \prod^{sch}, (ssid, i, srid), S_i, s_i, \tau)$$

   and sends $(ssid, i, \psi_i)$ to all participants P_j.
4. Last but not least, P_i upon receiving $(ssid, j, \psi_i)$ from all P_j, interpret $\psi_j = (\hat{A}_j, ...)$ and verifies:

$$\hat{A}_j = A_j$$

$$M(vrfy, \prod^{sch}, (ssid, j, srid), S_j, \psi) = 1$$

   and outputs $S = \prod_j S_j$.

- **Refresh phase** Key refresh is the process of periodically generating and exchanging new session share keys s_i and acts as a security measure in an adaptive party corruptions situation as assumed in the Canetti paper [3], this process is known as proactive security. In such an approach, the lifetime of the secret is divided into epochs and secret shares are refreshed each epoch. The scheme refreshes keys automatically and refreshment is executed as follows:

1. Each participant P_i samples a Paillier modulus N_i, ring-Pedersen parameters (s_i, t_i). P_i then creates a secret sharing $(s_i^1, s_i^2, ..., s_i^n)$ of 0 and computes

$$S_i = (S_i^1 = s_i^1 g, ..., S_i^n = s_i^n g)$$

and broadcasts (S_i, N_i, s_i, t_i) to all participants.

2. For every P_i such that $j \neq i$ receives (S_j, N_j, s_j, t_j) from every P_j, P_i encrypts every s_i^k using Paillier public key N_k and obtains the ciphertext C_i^k which he then sends to all participants.

3. For all P_i recipient of ciphertext C_j^k from P_j, P_i refreshes his share to a new one

$$s_i^* = s_i + \sum_l s_l^i modq$$

and public keys of other participants as

$$S_j^* = S_j + \sum_l S_l^i$$

2.3 Threshold signature

A threshold signature scheme is a protocol used to create a cryptographic signature from a shared private key that uses a distributed set of participants. This way, there is no single point of failure in case one of the key shares are lost or tampered with. Threshold Signature Schemes provide a means of distributing governorship, enabling a group of participants to decide and sign together on one decision. The security is defined that any less than t participants learn nothing about the shared private key and cannot produce a signature such that t is the threshold.

Our protocol supports both BLS and ECDSA threshold signatures. This is due to the fact that ECDSA is the most popular and widely used signature scheme and thus by supporting it we are compatible with protocols like Bitcoin and Ethereum which use ECDSA for generating keys and signing transactions. We support BLS signatures because they are known to be a better solution for doing secret sharing for multiple reasons, including the fact that threshold ECDSA needs multiple rounds of synchronous communication, while the communication in threshold BLS is asynchronous. This is a downside since more rounds of communication means slower speed of threshold signature generation even if ECDSA is known to be faster in key generation and signature verification. Secondly, BLS signatures are deterministic, unlike ECDSA, which requires a new random value for each signing. Being deterministic prevents several types of attacks and guarantees immutability of the resulting signature. Last but not least, BLS signatures can be aggregated and their aggregation verification is known to be fast while individual signature verification is slow. ECDSA on the other hand doesn't offer a natural way to be aggregated.

While the concept of threshold ECDSA has been around for some time, recent innovation made it more practical and usable minimize the overall number of rounds — including the pre-signing rounds where the message is not known yet. As a result, threshold ECDSA is increasingly being adopted as a key tool for secure and scalable decentralized applications.

2.3.1 ECDSA Threshold signatures scheme

Our goal in the near future is to have both asynchronous and non-interactive communication which we can achieve by following the Jens Groth paper [4] for threshold ecdsa signature.

ECDSA threshold signing with additive key derivation

Additive key derivation is a widely used process in cryptocurrency for either Heirarchical Deterministic Key Derivation as described in Bitcoin Improvement Proposal 32 (BIP32) standard for deriving many subkeys from a master key or for ECDSA threshold signing. Let s be the secret key and S be the associated public key. The additive derivation works by deriving child keys from the master key s by generating a random element "tweak" $\epsilon \in \mathbb{Z}_q$ creating a subkey $s + \epsilon$. We can compute the corresponding public key as $S + \epsilon g$. In BIP32's case, the tweak is derived via a chain of hashes applied to the public key

Another variation of ECDSA signatures is presignatures that allows the values r and $R = rg$ to be precomputed in advance of the signing phase as they are independent from the message. We have previously mentioned several protocols that use this property in their protocols like [2] and [3]. Groth and Shoup explain in their paper [5] that the combination of the two variations additive key derivation and presignatures introduce an attack on ECDSA that produces a forgery in signature in time significantly faster than $O(q^{1/2})$ which is much faster than the best known, square-root attack on plain ECDSA.
The paper presents two possible mitigations against these weaknesses:

- **Re-randomized presignatures:** is of the form $k = k' + \delta$ and $R = R' + \delta g$ such that $k' \in \mathbb{Z}_q$ and $R' = k'g$ for $\delta \in \mathbb{Z}_q$ a pseudo-randomly generated value.
- **Homogeneous key derivation:** an alternative additive key derivation mechanism with better security properties.
  Goal is to derive a pair of public and private key from the master public key $(S, S') = (sg, s'g)$. Giving a tweak $\epsilon \in \mathbb{Z}_q$. The derived secret key is $s + \epsilon s'$ and derived public key $S + \epsilon S'$.

Our threshold ECDSA signing protocol follows the Groth design [4] which utilizes re-randomized presignatures and additive key derivation. This design works

in an asynchronous communication model and provides a very efficient non-interactive signing phase. Groth assumes a system with an honest majority with $f < n/3$ corrupt (Byzantine) participants. We will describe the signing protocol in both an ideal and realistic scenario. An ideal scenario means identities of corrupted participants f are known and the environment Z will only give inputs to honest parties to generate signature. A realistic world on the other hand assumes that both environment Z and adversary A can pass messages back and forth freely. It also has same assumptions as in the ideal world for Z providing inputs to honest parties who run the protocol.

- **Pre-signature:** The ECDSA signature is of the form $(r = g^k \mid_{x-axis}, \sigma = k^{-1}(m+rs))$ so the goal of this step is to compute $g^{k^{-1}}$ and ks such that s_i is P_i's the secret share and s is the master secret key. In an ideal scenario, we assume each honest party P_i is given an initialization request $init$ only once, generates a presignature once per presignature identifier $presigID$, uses it once and also given a signature request sig which is received after presignature request. Each party should generate a signature only once. All honest parties receive same request in same order.

  1. Upon receiving $init$ request from P_i, if P_i is first to receive, the ideal functionality F_{ecdsa} runs key generation pub key $S \in \mathbb{G}$ and secret key s, then records $(init, S, s)$, and gives $(init, i, S)$ to simulator L.
  2. P_i receives $presig$ request, F_{ecdsa} runs presig generation to get $(R = g^k, k)$, records tuple $(presig, presigID, R, k)$ and gives to simulator L the following:
  $$(presig, i, presigID, R)$$

  By applying additive key derivation and re-randomized signature to the previous protocol, we add a public tweak ϵ derived as a hash of some strings that identify signing key so it becomes $s + \epsilon$ instead of s and re-randomize R with public randomness $\delta \in \mathbb{Z}_q$ such that $r = (g^\delta R) \mid_{x-axis}$ and replace k with $k + \delta$.

- **Signing**: Once a message msg is known, P_i receives sig request
  $$(sig, sigID, presigID, msg, \epsilon)$$
  F_{ecdsa} fetches tuple $(presig, presigID, R, k)$ that was recorded and runs signature algorithm to generate (r, σ) with the value δ, records
  $$(sig, sigID, presigID, msg, \epsilon, r, \sigma, \delta)$$
  and gives to L
  $$(sig, i, sigID, presigID, msg, \epsilon, r, \sigma, \delta)$$
  Finally, P_i forwards output $(output - sig, sigID, r, \sigma)$ to Z.

The assumption in the real world is that the ideal functionality F generates a sequence of random seeds $s_1, s2, ...$ which are used by only honest participants to issue a sequence of next-seed $(next - seed, i, j, s_j)$ requests to F which then gives them to A.

2.3.2 BLS Threshold signatures scheme

BLS signatures BLS is a digital signature scheme that is built over the BLS curve, specifically BLS12-381. BLS curve has the following properties that are quite unique in signature schemes and not possible for Elliptic Curves (EC):

- *Aggregatable:* With BLS, it's possible to aggregate all types of primitives (secret keys, public keys, signatures) and the result is always another valid primitive. This property is hard and very limited using EC.
- *Uniqueness and determinism:* For any given pair of public key and message, there can only be one valid BLS signature. Adding randomness in ECDSA however results in uncountable amounts of possible signatures for the same public key and message.
- *Shamir's Secret Sharing:* Creating secret key shares with BLS results in valid secret keys which are then used independently to sign a transaction and then signature is aggregated which means no need to ever compute the master private key and have a single point of failure. Computing secret shares with ECDSA doesn't necessarily results in valid shares and thus the master secret key must be computed and used for signature.

BLS Threshold Signatures Signing in threshold BLS works the same as normal BLS signatures, but instead of using the private key you sign using the share of the private key.
Let $H : \{0,1\}^* \to G$ be a collision-resistant hash-function.
Each participant generates a partial signature σ_i and a verification key

$$\sigma_i = H(msg)^{s_i}$$

where s_i is P_i's secret key that was generated in the key distribution phase and msg is the message to be signed.

Signature Aggregation After t signatures have been collected, they can be aggregated into a signature under the master private key msk. This is done by doing the shamir reconstruction on the partial signatures.
We use Lagrange interpolation as follows:

$$\begin{aligned}
\sigma &= \sum_{i=0}^{t-1} \sigma_i \prod_{j=0, j \neg i} \frac{x_i}{x_j - x_i} \\
&= \sum_{i=0}^{t-1} (H(m).sk_i) \prod_{j=0, j \neg i} \frac{x_j}{x_j - x_i} \\
&= H(m) \cdot \sum_{i=0}^{t-1} k_i \prod_{j=0, j \neg i} \frac{x_j}{x_j - x_i} \\
&= H(m) \cdot msk
\end{aligned}$$

By aggregating $t+1$ partial signatures for the message m we form the signature σ which verifies under the public key mpk.

2.4 Key Resharing

Benefits of key resharing proactive security, participants rotation , account recovery and forward secrecy the execution of a resharing algorithm at various times (both to support proactive security measures, and to support a change in the membership of the protocol); there may also be protocols and mechanisms to securely backup these keys.
Key resharing is a non-interactive publicly verifiable secret sharing scheme used by Lisbon protocol to allow for managing and changing the composition of qualified participants in a privacy preserving way. Our work is based on the Non-interactive distributed key generation and key resharing paper [6] but we only use their key resharing scheme while following the key generation process from the Canetti paper [3], threshold ECDSA signature from Groth [4] and threshold BLS from EthDKG paper [7].

We can think of the following scenarios for which key resharing is used:

1. Construct a DKG of a secret and confidential, yet verifiably distribute the shares to multiple receivers.
2. Existing shareholders of a secret can create a new DKG of the same secret and distribute it to a set of receivers which may or may not overlap with the original set of shareholders. This could happen in the case where shareholders want to add and/or delete participants to the same pool for example.
3. Create a new secret while keeping the associated public key and hands it to a set of receivers, which may or may not overlap with the original set of shareholders. This can happen in the case the secret was compromised and new secret needs to be generated for the pool of participants.

The resharing steps are as follow:

- Parent secret S is divided (in reality these shares are created by each dealer independently and parent key isn't generated at any time) among n receivers (we call them dealers) into n shares $S = (s_1, s_2, ..., s_n)$
- A set of t dealers will divide their key shares and share them with n' receivers. So basically each dealer i divides their share s_i into n' sub-shares $s_i = (s_{i1}, s_{i2}, ..., s_{in'})$ and send each to a receiver from the set of n' receivers with n not necessarily equals to n'.
- At the end, every receiver j will have a sub share s_{ij} from dealer i ($i \in 1, t$ and $j \in 1, n'$. For example receiver 1 will have these sub-shares $s_{11}, s_{21}, ..., s_{t'1}$

- Now every receiver is able to reconstruct their new secret share s'_i from the set they have $s'_i = \{s_{11}, s_{21}, ..., s_{t'1}\}$
- The new set of receivers will be able to reconstruct the secret parent key S only by recombining t' shares and doing polynomial interpolation of the shares $(s'_1, s'_2, ...s'_{t'})$

Combining either the shares s_i or shares s'_i will result in same original key S

2.5 Public key encryption with forward secrecy

Forward Secrecy (FS), also called Perfect forward secrecy (PFS), is an encryption method that changes the encryption keys frequently as a way of adding more security in case these keys get hacked.
The Lisbon Protocol uses FS to mitigate the damage caused by the exposure of secret key information that was used to decrypt the shares and thus an adversary that was able to retrieve secret keys will not manage to decrypt all past shares from previous time epochs. We build an hierarchical identity based encryption scheme that achieves that by allowing the dealer to keep a static long term public key and only refresh his or her private keys periodically (each block epoch). In addition to protecting from compromise secret keys risk, the Lisbon protocol provides a mitigation against Chosen Ciphertext Attacks (CCA-secure) so if a malicious dealer sends faulty ciphertexts (encrypted shares) they are detected and not accepted in the protocol.

2.5.1 CCA-secure public-key encryption with forward secrecy

The state-of-the-art for Hierarchical identity based encryption schemes (HIBE) show that pairing based HIBE schemes provide efficient and provably secure identity-based encryption schemes. In particular, The first constructions of such schemes were considered CPA-secure (Chosen Plaintext Attacks secure) but we can easily extend them to CCA-secure schemes without sacrificing efficiency. Therefore, the Lisbon protocol follows the same approach and uses a pairing based HIBE scheme that was constructed by Jens Groth and is described in his paper [6].

Construction The encryption algorithm takes as input groups G_1, G_2 and G_T of prime order p such that $e : G_1 \times G_2 \to G_T$ is a pairing with generators g_1, g_2 and $e(g_1, g_2)$, a message space $M = [-R..S] \subset \mathbb{Z}_p$, group elements $f_0, f_1......f_\lambda, h \in G_2$ and a tree height λ.
The first step is key generation which outputs the root public and private key pair (y, dk) by computing $y = g_1^x$ as the public key for a random $x \in \mathbb{Z}_p$ and the dk the decryption key as follows:

$$dk = (g_1^\rho, g_2^x f_0^\rho, ..., f_\lambda^\rho, h^\rho)$$

Each leaf may be associated with a decryption key, and a holder of that decryption key can recover the plaintext. Using the decryption key for the root we

can derive decryption keys for all leaves The randomized derivation algorithm takes a decryption key for epoch τ and returns a decryption key for $\tau + 1$ this way we can't decrypt messages of previous epochs. The randomized encryption algorithm then given the public key y and the plaintext m with the associated epoch for a leaf $\rho_1 ....\rho_\lambda \in \{0, 1\}^\lambda$ picks randomly $r, s \in \mathbb{Z}_p$ and returns the ciphertext c where

$$c = \left(y^r g_1^m, g_1^r, g_1^s, \left(f_0 \prod_{i=1}^{\lambda} f_i^{\tau_i} \right)^r h^s \right)$$

The decryption algorithm is deterministic and takes as input c and a decryption key $dk_{\tau_1 ......\tau_\lambda}$ for a leaf with the tree path $\tau_1 ......\tau_\lambda$ and outputs the plaintext m as:

$$M = e(C, g_2) \cdot e(R, b^{-1}) \cdot e(a, Z) \cdot e(S, e)^{-1}$$

such that

$$c = (C, R, S, Z) \in G_1^3 \times G_2$$

and

$$dk_{\tau_1 ......\tau_\lambda} = (\tau_1 .....\tau_\lambda, a, b, e) \in \{0, 1\}^\lambda \times G_1 \times G_2^2$$

The algorithm does a brute force search for m such that $M = e(g_1, g_2)^m$.

3 Threat Model

In this section we identify the different security weaknesses and attack vectors facing the Lisbon protocol and the proposed solution to mitigate each of them. Some of these weaknesses have already been mitigated by implementing their countermeasures and others are still a work in progress and we plan to address them in the near future.

3.1 Malicious behavior

We can imagine two scenarios where either the dealer is malicious or another participant of the protocol. In case of a malicious dealer, the dealing phase reveals whether everybody received a correct key share or not but what they don't know is whether the dealer is honest or not so we cannot trust a single entity generating these shares and therefore we need a distributed key generation protocol. So the setup that we have is multiple dealers and each of these dealers provide a dealing and we get security if at least one of these dealers is honest, we can actually tolerate quite few bad dealers

In the case where a participant is malicious, the scenario is in the key generation process, participants receive their secret shares encrypted by the dealer and verify their integrity using the Feldman's VSS protocol. If the share is proven to be invalid, the participant P_j issues a dispute claim in order to disqualify the dealer (share issuer) as explained in the dispute phase.

However, P_j cannot verify the integrity of the other shares of the rest of participants. In the case where a participant P_w sends no message of verifying their share? P_w could be disconnected or malicious by accepting an invalid share sent by dealer and not reporting it.

Solution: Publicly verifiable secret sharing: In this case, the dealer encrypts shares using the participants' public keys and generates a non-interactive zero knowledge proof which guarantees that what shares have been encrypted in an actual valid secret share that corresponds to the public key material. This way every participant knowing the public encryption key is able to verify the integrity of other shares. If one or more verifications fails the dealer fails and the protocol is aborted.
A scheme with such a verifiability property is also used to verify dealing integrity to prevent malicious behavior by the dealer. Feldman developed the first efficient and non-interactive VSS protocol which we use. He used a commitment with computational security and unconditional share integrity to achieve it. However, since the adversary can access the public key, unconditional security for the secret is impossible.
The Lisbon protocol provides this property as it's using asymmetric key encryption to encrypt shares and thus shares are publicly verifiable by everyone.

3.2 Man In the Middle

An attacker can record and modify communications between two participants of Diffie Hellman key exchange by impersonating one of the two participants. While Forward Secrecy (FS) protects against the decryption of such communication, it cannot prevent it from being collected if an attacker positions themself in the middle. In principle, obtaining and keeping such records leaves the door open for them to be deciphered in the future. As a result, the Lisbon Account Protocol employs as a countermeasure to Man-in-the-middle attacks ECDHE (Elliptic Curve Diffie Hellman Ephemeral) with ECDSA signature.

This means participant A signs the public elements they send to participant B with their private key during the key exchange so that B knows for sure the elements they received come from A, and not some malicious actor listening to the communication by verifying the signature against A's public key.

3.3 Key compromise

The usage of static long term keys to encrypt shares introduces the risk of key-compromise which could reveal information about the key shares. An attacker instantly has access to all encrypted shares from previous sessions that were transmitted between a dealer and participant using a particular key and be able to decrypt them. This type of attack is the most common one since cryptographic keys are directly exposed.

Solution: Forward Secrecy: Using ephemeral keys during key exchange protects session keys even if a long-term key is compromised which guarantees forward secrecy. Combining distributed trust with ephemeral keys protects a system against key compromise. Here, the system's time is divided into phases. At the start of each phase, participants' secret shares are renewed such that new shares are independent of previous ones, except for the fact that they interpolate to the same secret key. With an assumption that the adversary may corrupt at most t participants in each phase, the system now becomes secure
liveliness which we address with async

4 Conclusion

The Lisbon Account Protocol provides a practical means of generating and utilizing distributed externally-owned accounts (dEOAs) for use on decentralized networks in real-world conditions. The resulting distributed externally-owned accounts (dEOAs) provide end-users with several desirable benefits without compromising on core values such as self-custody, decentralization, and interoperability. These benefits include:

- Eliminating private key single-point-of-failure risk through distributed ownership
- Account recovery and proactive security through key resharing
- Broad usability through both ECDSA and BLS Signature Schemes

The Lisbon Account Protocol consists of four core components, including:

- Decentralized Key Generation, through Secure Multiparty Computation (SMPC) and Verified Secret Sharing (VSS)
- Threshold Signature Formation, for both BLS and ECDSA Signature Schemes
- Key Resharing, and consequently, rotation, for dynamic ownership and security
- Public Key Encryption with Forward-Secrecy for on-chain data storage

We consider the protocol practical for real-world applications as it:

- Enjoys broad compatibility with decentralized networks by providing ECDSA-based EOA public addresses with off-chain signature formation, as well as BLS keypairs for next-generation networks
- Is independent of any third-party dependencies, such as MPC systems or validator networks, hardware such as HSMs, APIs, Tokens, or proprietary communications channels

- Achieves Public-key Encryption with Forward Secrecy without requiring end-users to actively store any additional data or passwords
- Is non-interactive and asynchronous, assumes public and unsecured channels of communication, and therefore is better suited for decentralized network environments

The protocol provides the necessary infrastructure to better meet human expectations in regards to accounts: No single point-of-failure, forgiveness for human error, and more natural ownership structures, all while providing a high degree of usability without prerequisite understanding of cryptographic or technical concepts. We envision applications ranging from single-user accounts with recovery for decentralized applications and games to complex decentralized autonomous organizations and businesses, with sophisticated governance, operational policies, and business logic.

In designing the protocol, we have addressed many safety and security concerns common in Web3, both cryptographic and practical, such as forward secrecy, denial of service, man-in-the-middle, and active malicious adversaries. The considerations in making the protocol practical allow developers to confidently build on the protocol and corresponding infrastructure, and empower them to focus on additional functionality and programmability.

The result is an externally-owned account that is better fit for human end-users, functionally shielding them from the risks and complexities of decentralized networks, while providing a higher degree of usability and safety. Ultimately, we expect the protocol to enable developers to better meet the expectations of mainstream users, fueling the adoption of decentralized networks as the next generation of the internet - Web3.

References

[1] Rosario Gennaro and Steven Goldfeder. Fast multiparty threshold ecdsa with fast trustless setup. Cryptology ePrint Archive, Paper 2019/114, 2019. https://eprint.iacr.org/2019/114.

[2] Rosario Gennaro and Steven Goldfeder. One round threshold ecdsa with identifiable abort. 2020. https://eprint.iacr.org/2020/540.

[3] Ran Canetti, Rosario Gennaro, Steven Goldfeder, Nikolaos Makriyannis, and Udi Peled. Uc non-interactive, proactive, threshold ecdsa with identifiable aborts. Cryptology ePrint Archive, Paper 2021/060, 2021. https://eprint.iacr.org/2021/060.

[4] Jens Groth and Victor Shoup. Design and analysis of a distributed ECDSA signing service, 2022.

[5] Jens Groth and Victor Shoup. On the security of ecdsa with additive key derivation and presignatures. Cryptology ePrint Archive, Paper 2021/1330, 2021. https://eprint.iacr.org/2021/1330.

[6] Jens Groth. Non-interactive distributed key generation and key resharing. *IACR Cryptol. ePrint Arch.*, page 339, 2021.

[7] Philipp Schindler, Aljosha Judmayer, Nicholas Stifter, and Edgar R. Weippl. ETHDKG: distributed key generation with ethereum smart contracts. *IACR Cryptol. ePrint Arch.*, page 985, 2019.

5 Appendix

In this section, we will introduce all the cryptographic concepts used in our protocol.

Groups

A binary operation $*$ on a set G is a mapping from $G \times G$ to G, which associates to elements x and y of G a third element $x * y$ of G.

Definition: A group $(G, *)$ consists of a set G together with a binary operation $*$ for which the following properties are satisfied:

- Associativity: $(x * y) * z = x * (y * z),\ \forall x, y, z \in G$
- Neutral element: $\exists!\ e \in G,\ e * x = x = x * e,\ \forall x \in G$
- Inverse element: $\forall x \in G,\ \exists!\ x' \in G,\ x * x' = e = x' * x$ where e is the neutral element of G.

A group G is Abelian (or commutative) if: $x * y = y * x, \forall x, y \in G$

Cyclic groups:

Definition: A group G is said to be cyclic, with generator x, if every element of G is of the form x^n for some integer n.

Fields

A field F is a set with two binary operations $+$ and $*$ that satisfies the following field axioms:

- Closure under addition: $\forall x, y \in F,\ x + y \in F$
- Closure under multiplication: $\forall x, y \in F,\ x * y \in F$
- Additive inverses: $\forall x \in F, y \in F$ such that $x + y = 0$

- Multiplication inverses: $\forall x \in F$ such that $x \neq 0$, $\exists\ y \in F$ such that $x * y = 1$, y is called the multiplicative inverse of x and is denoted x^{-1} or $\frac{1}{x}$
- The distributive law: $\forall x, y, z \in F,\ x * (y + z) = x * y + x * z$

Finite Fields:

A finite field is a field F_q with a finite number of elements. The order of a finite field

$$|F_q| = q = p^k$$

for some integer $k \geq 1$ and p prime equals the number of elements in the field.

Elliptic curves

We use pairing friendly elliptic curves defined over very large finite fields $|F_p| \approx 2^{256}$. In the following we will explain what is an elliptic curve, pairings and elliptic curves over finite fields.

Definition An Elliptic curve E is a mathematical object defined over a field F and generally expressed in the following Weierstrass form:

$$y^2 = x^3 + ax + b$$

for some $a, b \in F_q$ where (x, y) are called affine coordinates.

Elliptic curves over finite fields

We will mainly focus on elliptic curves over finite fields since they are the ones used for cryptographic applications. An elliptic curve over a finite field F_q is an abelian group G with a finite number of points n such that $n = |G|$ (order of the group G).

The discrete logarithm problem

The security of many cryptographic techniques depends on the intractability of the discrete logarithm problem.

Definition: Let G be a multiplicative group. The discrete logarithm problem (DLP) is: Given $g, h \in G$ to find a, if it exists, such that $h = g^a$.

Elliptic Curve Discrete Logarithm Problem (ECDLP)

Let E be an elliptic curve of the Weierstrass form defined over a finite field F_q. Let S and T be two points in $E(F_q)$. Find an integer m such that:

$$T = mS$$

The fastest method to solve the ECDLP problem in $E(F_q)$ is the Pollard Rho method which has exponential complexity $O(\sqrt{|G|})$. In order for this algorithm to be exponential, we need to define elliptic curves over very large fields $|F_p| \approx 2^{256}$.

Pairings

Pairing based-cryptography is used in many cryptographic applications like signature schemes, key agreement, zero knowledge...etc. For example, pairings are used to create efficient circuit-based zero knowledge proofs.

Definition A pairing e is a bilinear map , defined as:

$$e : G_1 \times G_2 \to G_T$$

Such that G_1, G_2 and G_T are abelian groups. The bilinear property means that:

$$e(P + P', Q) = e(P, Q) + e(P', Q)$$

$$e(P, Q + Q') = e(P, Q) + e(P, Q')$$

for $P, P' \in G_1$ and $Q, Q' \in G_2$ and $a, b \in \mathbb{Z}$

Montgomery curve

Definition: A Montgomery curve over F_q is an elliptic curve defined as

$$E_{(A,B)} : By^2 = x(x^2 + Ax + 1)$$

where A and B are parameters in F_q satisfying $B \neq 0$ and $A^2 \neq 4$.

Curve25519 Curve25519 is a Montgomery curve providing 128 bits of security defined as:

$$y^2 = x^3 + ax^2 + x$$

over prime field p where: $b = 1$. The curve is birationally equivalent to a twisted Edwards curve used in the Ed25519 signature scheme.

Barreto-Lynn-Scott curves (BLS curves) A BLS curve is a pairing over BLS curves that constructs optimal Ate pairings. BLS12-381 is optimal for zk-SNARKs at the 128-bit security level and is implemented by the zcash team. Bls12-381 has an embedded Jubjub curve.

Signature schemes

We will give the definition of the following signature schemes: ECDSA and BLS.

- ECDSA works over a general elliptic curve and is used in Bitcoin and Ethereum.
- BLS signature scheme uses a bilinear pairing for verification, and signatures are elements of an elliptic curve group, it is already integrated into major blockchain projects such as Ethereum, Algorand, Chia and Dfinity.

ECDSA: Elliptic Curve Signatures

The (Elliptic Curve Digital Signature Algorithm) is a cryptographically secure digital signature scheme, based on the elliptic-curve cryptography. ECDSA is defined over a group $\mathbb{G}$ of prime order q, $\mathbb{G}$ is defined as the group of points on an elliptic curve and $g \in \mathbb{G}$ is a generator for $\mathbb{G}$. Given a secret key $s \in \mathbb{Z}_q$ and a message $msg \in \{0,1\}^*$, the signing algorithm runs as follows:

- Compute the hash m of the message msg as $m = hash(msg) \in \mathbb{Z}_q$ (we can use SHA256).
- Choose a random number $k \in \mathbb{Z}_q^*$ and compute a point $R = kg$
- Select the x-coordinate of R as $r = R.x$
- Compute the signature $\sigma = k^{-1} \times (m + rs)$
- The signature consists of the pair (r, σ)

BLS signatures

The BLS signature scheme is a cryptographic digital signature scheme based on bilinear pairing operations. In the BLS signature scheme, a signer generates a public and private key pair (x, g^x), where the public key g^x is a point on an elliptic curve and the private key x is an integer.

To sign a message m, the signer hashes the message $h = H(m)$ to a point on the curve and then raises the point to the power of the private key. The resulting point is the signature

$$\sigma = h^x$$

To verify the signature, the verifier computes a pairing e operation between g^x and h and compares it to another pairing operation between the signature σ and a fixed point on the curve. If the two pairings match, the signature is valid.

$$e(\sigma, g) = e(H(m), g^x)$$

The BLS signature scheme is efficient and has several desirable properties, including being resistant to certain types of attacks such as existential forgery and key-only attacks.

Diffie-Hellman Key Exchange

The Diffie-Hellman key exchange is a cryptographic protocol that allows two parties Alice and Bob to generate a shared secret over an insecure communication channel. The protocol involves both parties agreeing on a public domain parameters and each generating their own private key. They then use their private keys and the public parameters to compute a shared secret, which can be used for secure communication. The security of the Diffie-Hellman key exchange is based on the computational difficulty of the discrete logarithm problem.
Let's outline the process step by step:

1. Alice and Bob agree on a prime modulus p and a generator g such that $g \in \mathbb{Z}_p^*$ is a generator of a group of a prime field $\mathbb{Z}_p^*$.
2. Alice selects a private random number a such that $1 < a < p - 1$ and calculates $l = g^a \bmod p$ sending the result publicly to Bob;
3. Then Bob selects his private random number b such that $1 < b < p - 1$ and calculates $m = g^b \bmod p$ sending the result publicly to Alice;
4. Alice takes Bob's public result m and raises it to the power of her private number obtaining $m^a \bmod p$;
5. Bob takes Alice's public result l and raises it to the power of his private number obtaining $l^b \bmod p$;
6. Finally the shared key s is

$$s = m^a modp = (g^b)^a modp = (g^a)^b modp = l^b modp$$

Lagrange basis and polynomial interpolation

Polynomial interpolation is a process where a given set of points (x_i, y_i), $i \in [n]$ allows us to construct a polynomial $f(x)$ that passes through all of them. We will assume that $x_i \neq x_j$ for all distinct i, j pairs, otherwise there is a repeated pair or it is not possible to construct the polynomial as it would have to take two different values at the same x-point.

Notice that for a set of 2 points, we can find a line that crosses both of them, for a set of 3 points, a parabola, and in general, for a set of n points there is a polynomial of degree $n - 1$ that contains all of them.
The Lagrange interpolation consists of 2 steps:

1. Construct a **Lagrange basis**: This is a set of n polynomials of degree $n-1$ that take the value 0 at all points of the set except one, where their value is Expressed in a formula:

$$L_i(x) = \begin{cases} 0, & \text{if } \ x = x_j, j \in [n], j \neq i \\ 1, & \text{if } \ x = x_i \end{cases}$$

The polynomials L_i can be constructed in the following way:

$$L_i(x) = \prod_{0 \le j < n,\ j \ne i} \frac{x - x_j}{x_i - x_j}$$

Notice that this product has $n-1$ terms, and therefore results in a degree $n-1$ polynomial.

2. Scale and sum the polynomials of the basis.

$$f(x) = \sum_{i=0}^{n-1} y_i \cdot L_i(x)$$

The properties of the Lagrange basis now allow us to scale each polynomial to its target value by multiplying by y_i and then add up all the terms.
The important observation we can extract from the Lagrange interpolation is that given a fixed set of points $x_1, \ldots, x_n$ (an evaluation domain) we can represent any polynomial of degree $d < n$ by its evaluations $f(x_i)$ at $d+1$ points in the set. As it turns out, this representation is much more convenient than the usual coefficient representation as it provides a very simple and fast way of computing sums and multiplication of polynomials. However, the coefficient form is still useful for evaluating the polynomial at points outside the evaluation domain.
Switching between these two forms of representation is very useful. The coefficient form is preferred when the polynomial must be evaluated at a random point (outside of the evaluation domain). The evaluation form is better suited for operations between polynomials such as addition, multiplication and exact quotients. The algorithm that allows us to efficiently switch between representations is the Fast Fourier Transform (FFT). This is an efficient algorithm for the more general discrete Fourier Transform (DFT). It has a complexity of $\mathcal{O}(n \cdot log(n))$ with n being the degree of the polynomial.

Zero Knowledge Proof systems

Before we start talking about zero knowledge proof systems, let's first define what a proof system is: A proof system is a protocol by which one party (prover) wants to convince another party (verifier) that a given statement is true.

Zero knowledge proof system: In zero-knowledge proofs, the prover convinces the verifier about the truthfulness of the statement without revealing any information about the statement itself.

Properties A zero-knowledge proof needs to fulfill each of the following properties to be fully described:

- **Completeness:** An honest prover is always able to convince the verifier of the truthfulness of their claim.

- **Soundness:**If the prover's claim is false (malicious prover), the verifier is not convinced.
- **Zero knowledge:**The proof should not reveal any information to the verifier beyond the truthfulness of the given claim.

Types of Zero knowledge proofs

There are two types of zero knowledge proofs (interactive and non interactive ones) Interactive zero-knowledge proof systems were first introduced in 1985 by Goldwasser, Micali and Rackoff. Non-interactive schemes were introduced later on by Blum et al. The main difference between both schemes is that interactive proofs require interaction between both parties which means both have to be online in order to do so; this can be seen as inconvenient, especially for modern cryptography applications, while non-interactive proofs need a shared setup preprocessing phase instead. The shared setup phase will allow the participating parties to know which statement is being proved and what protocol is being used.

Interactive Zero Knowledge Proofs

A prover P has a secret s and correctly responds to challenges to convince a verifier V it has knowledge of s using rounds of interaction between the two parties.

Non interactive zero knowledge proofs (NIZK)

Non-interactive zero-knowledge proofs, also known as NIZKs are another type of zero-knowledge proof which require no interaction between the prover and the verifier. In order to transform interactive proofs into NIZK proofs, cryptographers used the Fiat-Shamir heuristic hash function. This hash function allows one to compute the verifier challenges and offer very efficient NIZK arguments that are secure in the random oracle model. More recent works have started using bilinear groups to improve efficiency.

The two major types of NIZK proofs are zkSNARKs and zkSTARKs; zk-SNARKs are based on elliptic curve cryptography and need a trusted setup phase, whereas zkSTARKs rely on hash functions and do not have any trusted setup. We will focus on zkSNARKs since PLONK is in this category. zkSNARKs stands for zero-knowledge Succinct Non-interactive ARguments of Knowledge.

- - Succinct: proof length needs to be short
- Non-interactive: needs to be verifiable in a short amount of time
- ARKs: need to show that we know an input (witness) which yields to a certain computation.

zkSNARKs cannot be applied to any computational problem directly; rather, you have to convert the problem into the right “form” for the problem to operate on.