Sovereign System

Privacy-preserving identity protocol employing zero-knowledge technology and decentralized identifiers for secure self-custody and user-owned data across distributed ledger networks

Douglas Broughton, Mehmet Kiraz, Foteinos Mergoupis-Anagnou, Kadir Pekel, Gwin Scott

Friday 14th July, 2023

https://www.vendiblelabs.com/, info@vendiblelabs.com

Contents

DOCUMENT APPROVAL INFORMATION	
Approved by	VENDIBLE (BVI) INC

CHANGE RECORDS			
Version No	**Reason**	**Chapter**	**Date**
1.0	The Sovereign System whitepaper has been created.	-	12 Dec 2022
1.1	Correction of minor typos. New use cases have been added.	Typos correction in all chapters. Also, Chapter 13 and 14 have been added.	14 Apr 2023

Chapter 1

Introduction

1.1 Innovation vs Technology Adaptation

Blockchain technology allows participants to transact directly with peers without intermediaries or custodians [Nak08]. The earliest adopters of this technology include technologists and internet-savvy individuals who ideologically agree with the principles behind self-sovereignty or self-custody. For most, however, self-custody is a technical burden with risks, complexities, and costs that outweigh the current perceived benefits. Without a new system for interacting with decentralized technologies, this friction will keep most individuals from acceptance of the technology without the assistance of custodians.

Examining technology acceptance leads us to critical factors necessary for adopting distributed ledger technology. Rogers' Diffusion of Innovations [RS71, Rog79, Rog, Rog03, Rog10] (DOI) describes how new technologies gain acceptance in social systems. The first users are deemed innovators and acknowledge the practical advantages of a novel system over the perceived costs. Innovators are often willing to interface with a new technology regardless of the costs, as they are the first to recognize a need for innovation. Innovators represent 2.5 percent of the total market share, and their acceptance brings continued advancements that increase the ease of interface and reduce perceived cost. With continued innovation comes the approval of the Early Adopters, the next 13.5 percent of the market. This adoption group generally identifies as technology enthusiasts, understands the value versus the cost, and accepts the technology once the Innovators overcome the initial limitations. With continued technological improvements, adopting new technologies follows an S-shape adoption curve, following Metcalfe's law.

As an adaptation of DOI, Moore [Moo22] has suggested a gap exists between Early Adopters and the rest of the market. Described as a chasm, Moore implies that there is a tremendous amount of friction that a technological movement must overcome in the early adoption cycle. While primarily focused on product marketing, Moore's position is that the Innovators and Early Majority have different expectations than the remaining adoption groups, even when initially following the S-curve of adoption. While Rogers agreed [Rog03] that there are essential differences between adoption groups, Moore's chasm theory does not hold with Roger's DOI results as he measured no recognized breaks in adoption between adjacent categories. Yet, a technologies' failure to innovate to the point that it can meet the majority's expectations could

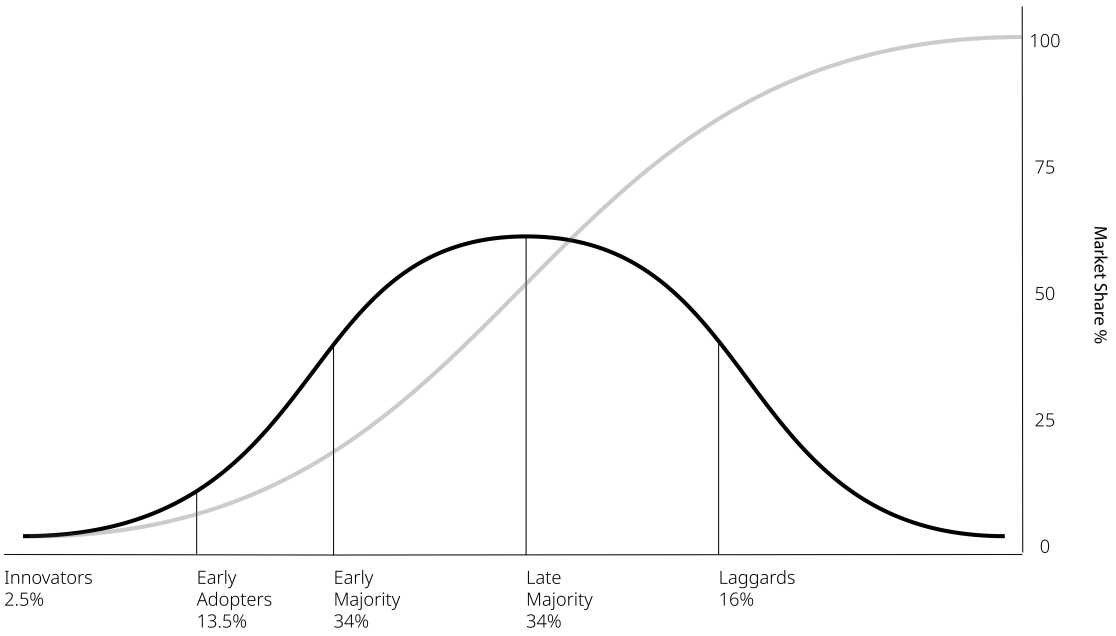

Figure 1.1: Categories of adopters and adoption curve, based on Diffusion of Innovations (Rogers, 2003)

suggest why specific innovations fail to reach the masses while others succeed [RSF07].

The Technology Acceptance Model [Lam22] (TAM) posits that a specific technology's perceived usefulness and ease of use must outweigh the perceived effort required to adopt if that innovation is to be accepted. Costs in TAM can be monetary outflow, physical exertion, and mental effort. This model aligns with the Value-based Adoption Model [Lam22] (VAM), where the benefits, defined as usefulness and enjoyment, must outweigh the monetary and non-monetary costs.

All three models point to similar models for adoption. New technologies must have advantages over previous systems and provide more value than incumbent technologies. They must continually innovate and become easier to use so that the majority see more value in their application than cost, thus becoming socially acceptable. Once a great enough share of the market perceives more benefits than difficulties, intentions to adopt convert into acceptance, leading to a network effect that takes over a market.

Public permissionless blockchains must continue to innovate until a tipping point, where 15 and 20 percent of the population accept the technology, to reach a level of social acceptance for that stated network effect to take hold and accelerate adoption through the Early and Late Majority groups [Lam22]. While there has been a tremendous focus on technological improvements, permissionless systems face challenges as incumbent technologies, such as payment systems, continue to innovate. Private permissioned blockchains, as well as the growing number of service providers and applications that act as custodians for digital assets, also pose risks to the sovereign model of permissionless systems.

1.2 Custodial solutions vs Self-Custody

Having custodians remove complexities helps aid adoption, and the acceptance of others helps lead the majority to accept new technology [Rog03]. While custodians have their uses and will play a significant role in the use of blockchain technology, the current trends suggest that the Early Majority, representing roughly 34 percent of the population, will interact with blockchain technology primarily through the assistance of custodians. Current trends and adoption models suggest they will follow this path because service providers offering custodial solutions offer the net lowest cost compared to self-custody solutions. Custodians provide significant reductions in physical and mental effort compared to self-custody while perceptually portraying nearly the same advantages as self-custody. While there are profound differences in advantages between custodial services and self-custody, they are perceived as insignificant by new entrants into the market as, generally, they are more concerned with the function of a product rather than the underlying structure or delivery of the product. For most, the importance of self-custody is a foreign concept that requires a tremendous amount of physical and mental effort; more than most wish to expend when so many other areas of their daily life require focus and effort.

Businesses, institutions, and governments accept distributed ledger technology at an increasing rate as they see process improvement and cost reduction advantages [Gar22]. Primarily, these entities choose permissioned networks where the key stakeholders own the network nodes, which provide consensus to control business logic tailored to a specific vertical or use case. As businesses continue their acceptance and integrate this technology into their standard practices, it will eventually lead to new product offerings on permissioned blockchains that will likely only provide custodial offerings. For businesses, this choice to build on closed systems is evident as they see clear advantages over public distributed ledger technology [Men18]. However, as use of these permissioned blockchains continue to gain support for consumer product offerings, trends suggest the Late Majority and Laggards will primarily, if not entirely, interface with blockchain technology through custodians.

Self-sovereign control over assets brings freedom. However, this freedom comes with many technical complexities and risks making the barrier to entry and the burden of self-custody high. Custodians mitigate many of these risks and remove complexities at the expense of some freedom as the custodian controls the flow of transactions and, in most cases, maintains ownership of individuals' assets.

The current adoption trends towards custodial solutions may lead to the negation of self-custody and public blockchain systems. By public systems, we are referring to public permissionless blockchains, which allow anyone to participate in the network (we will refer to these as pubic blockchains, public systems, and public networks depending upon the context). Examples of public networks include Bitcoin, Ethereum, Zcash, and Algorand. Public systems must also incorporate ways to protect consumer data in transactions for widespread acceptance. Public permissioned blockchains (such as Ripple and NEO) or private permissioned blockchains (such as Hyperledger, Quorum, or a private Ethereum side-chain) already include different levels of data protection. While transparency and traceability are crucial for trust in public blockchain networks, seeking ways to protect sensitive data is necessary. Companies almost unanimously choose public and private permissioned networks for data protection. Therefore, public systems must also include consumer protections to gain a majority of acceptance in the market.

1.3 Motivation and Contributions: The Sovereign System

This paper presents a third option between self-sovereignty and custodians to address the discussions in the previous section. We title this option *the Sovereign System*. On a very high level, the Sovereign System has the following features:

- It is rooted in identity and provides participants on public networks full ownership and control over their data and assets, with safeguards based on identity to ensure data protection.
- Further variations of this system ensure privacy preservation and compliance for individuals and institutions.
- The structure presented removes the complexities of self-custody while ensuring the participant always maintains full ownership of their assets and data.
- We have constructed this system to remove a significant amount of cost with advantages for consumers and institutions so that sovereign control is a viable option for most users who have yet to enter the market.

1.4 Roadmap

The rest of the paper is as follows.

- In Chapter 2, we present the challenges which are blocking the mass adoption of permissionless networks.
- In Chapter 3, we present other issues such as data, identity, and privacy.
- In Chapter 4, we present the high-level solution of the Sovereign System and general assumptions.
- Chapter 5 presents the necessary cryptographic background.
- In Chapter 6, we present the Σ protocols which are necessary to prove the correctness of calculations without disclosing the private information.
- Chapter 7 gives the basic definitions behind the Sovereign System.
- Chapter 8 revisits the Merkle Trees and Authentication Paths which are necessary to use the membership proofs.
- Chapter 9 presents the open source implementations of ZKSNARKs which are used in the Sovereign System.
- Chapter 10 introduces the architectural flow of the Sovereign System.
- In Chapter 12, we present the first use case for the recovery of private keys of other chains.

- Chapter 15 presents how a user can disable himself/herself.
- Chapter 16 presents how it can be easily extended to social recovery, solving the long standing problem.
- In Chapter 17, we present how to secure the Sovereign System further.
- Finally, in Chapter 18, we present other potential and immediate use cases of the Sovereign System.

Chapter 2

Challenges Blocking Adoption of Permissionless Networks

2.1 Importance of Public Permissionless Networks

From its inception with Bitcoin, the original intent of blockchain technology was to facilitate a shift in ownership rights from centralized authorities directly to those interacting with the network through a shared, decentralized ledger that was public, agreed upon, and tamper-resistant. The importance of this technology cannot be understated; it provides a clear path to sovereignty over one's assets, a rare opportunity in our modern financial system. The shared ledger also offers parties who do not know or trust one another security while transacting through trustless mechanisms.

Innovators and many Early Adopters recognized the value of owning and controlling one's assets. They embraced a solution to systemic issues of trust-based financial systems that routinely fail due to institutions' mismanagement of assets. To achieve the freedom outlined by the Innovators, one must be comfortable assuming self-custody of one's assets. Blockchain technology achieves self-custody through a secure private key generation (which can also be represented by a passphrase), storage, and management. Digital assets on blockchain networks exist as records on the distributed ledger, controlled by the private keys stored in wallets and other devices by the owner. With possession of the private keys comes complete positive control of the assets. When an individual or entity relies on a third party to secure their digital assets, they give or transfer the ownership rights to that party, once again forming a trust-based relationship and breaking the original intent of the technology. Therefore, innovation in managing private keys must continue so that sovereign control on public permissionless networks continues to gain acceptance in the market.

2.2 Consumer Challenges

2.2.1 Private Key (or Passphrase) Generation and Management

While critical to the function of public blockchain networks, the generation of private keys presents a primary barrier to entry for most of the population. The ease of account generation through email and password and the near frictionless integration of single-sign-on technologies have become the de facto authentication mode. These methods for managing access to services are widely viewed as secure and painless to generate and manage. When initially presented with the novelty of securing a private key or passphrase, coupled with the warning that this task is critical and that one should employ the utmost seriousness in management, many entrants may feel overwhelmed and confused. Without an understanding of the history and intent of the technology, those new to the space, looking to explore the potential value, will find this friction a deterrent and look for more straightforward paths to adoption through custodians.

Furthermore, many reports state that many who understand the importance of self-custody to safeguard their private keys or passphrases from loss have made mistakes leading to permanent asset loss. Current studies estimate that as much as 20 percent of the digital asset Bitcoin (BTC) has been lost forever due to mismanagement or loss of private keys [KH20]. Assuming a 20 percent loss of the current circulating supply, approximately 3,800,000 BTC, this represents USD 60 billion in lost value at the time of writing this paper. These assets were lost by new entrants and those experienced in self-custody alike, so we cannot expect the early and Late Majority, most without technical inclinations, to shoulder the burden of self-custody without support.

2.2.2 Trust Issues

Blockchains facilitate internet transactions between unknown, untrusted parties through trustless mechanisms on shared distributed ledgers. This technology is secure and achieves the intended goal. However, the early and Late Majority have different trust thresholds than the Early Adopters. Generally, most people trust governments, institutions, and companies to perform the function or service they have been created to serve. If they do not trust the entity, they at least have trust in a process or system the entity operates within to ensure that they perform the service to minimum standards.

Decentralized wallets and applications remove the need for trust but introduce uncertainty and ambiguity for most people accustomed to knowing who they are dealing with in transactions. The trustless setup and operation of blockchain, which ensures confidence in transactions, ironically leads those unfamiliar with how the technology works to distrust the process. Most internet users will need more time to dive deep to understand the security features of blockchain networks. The use of blockchain networks for illicit activities and highlighting these issues over beneficial use cases in the public have led many to view decentralized application use and digital assets as a space to be wary of or only viable as a speculative investment.

While some decentralized and centralized options and structures are beginning to emerge, the need for more consumer protections, guidance on regulations, and taxation for actions taken with digital assets in decentralized and centralized applications creates confusion. A

long-documented history of projects launching and not delivering after receiving funding, some pulling funds and not providing without any notice, creates chaos and drives people away from blockchain-based applications and services.

Over the past decade, hundreds of centralized digital asset exchanges and custodians have failed [Wis22]. In many of these cases, the assets and funds of the users who had credits from deposits of digital assets had their assets frozen, stolen, or used to settle other company debts, leaving many without recourse. With each occurrence, there is a call for greater enforcement of regulations on these entities. Still, with technology moving faster than lawmakers are sometimes capable of acting, it leaves space for further exploitation of consumers, which, in turn, drives people away from decentralized technologies.

The traditional financial industry, and the products they provide, have many consumer protections built into their offerings. Consumers expect certain assurances that their deposits are safe from loss and that there are channels to address and mitigate adverse outcomes in the event of fraudulent charges. Backstops and recourse are an option when incidents occur. Public systems must address this directly as it is a significant practical advantage for incumbent technologies.

2.2.3 Public Data

Public blockchains have distributed shared ledgers that act as a source of truth for all who engage with the network. The sharing of publicly available data is one of the most important features of blockchain. However, many participants fail to understand their lack of privacy when transacting with known parties or sharing account information across social platforms. We will revisit the original intent of blockchain, where entities can have trust in transactions across the internet without knowing the other party involved in a transaction. However, as we have reached the early adopter phase of the technology adoption curve, transactions are no longer siloed to unknown entities. Individuals and companies that transact on public networks expose the state and activity of their assets, including historical, present, and future data, to anyone who wishes to research and track them.

The majority of public blockchain networks are pseudonymous. Public accounts or addresses are used instead of the participants' names or identifying information. With wider popularity and sharing of information across social media, the popularity of non-fungible assets, and broader acceptance of payments with digital assets, people are openly sharing their account information with friends, families, businesses, and the wider general public. As soon as one entity is aware of the public address of another (in public blockchains that do not have a privacy component), they can trace the state and activity of the assets as the pseudonymity of the account is broken. This feature exposes any entity that discloses its ledger account to potential tracking, manipulation, or theft attempts. No other payment system, whether cash or electronic, operates in such a public manner. Institutions and individuals alike will not accept a fully open payment system where each party you trade with instantly has access to your net worth and transaction history. Public blockchain networks will only achieve mass adoption with consumers, businesses, and institutions when data protections are in place.

2.3 Institutional Challenges

2.3.1 Institution Focus on Permissioned Networks

Governments, institutions, and businesses are fully aware of the issues caused by open data on pseudonymous blockchain networks. In a recent survey, a majority of Fortune 500 Chief Information Officers, and the companies they serve, were actively working, building, or designing solutions that include a blockchain component [Gar22]. However, almost all of these solutions will not operate on public blockchains. Instead, closed networks such as Hyperledger, Quorum, or a private instance of a public blockchain such as Ethereum with nodes hosted by the key stakeholders are custom-built for specific use cases. Companies building public blockchains are primarily creating fiat ramps or custodial services as a new revenue source through exchange and management fees. Without privacy, competitors, and possibly even trade partners, would exploit information that should otherwise only be known to customers and suppliers and, only then, disclose what is necessary for each particular action or transaction. One of the most significant issues for institutions is the need for accepted industry standards, both technically and legally, to operate either on permissioned or public networks.

2.3.2 Permissionless Networks Lack of Structure for Compliance

Governments, institutions, and businesses must operate within clearly defined structures. Structures for operation usually stem from regulations and mutually agreed-upon standards specific to each industry or market. Institutions have implemented such measures to avoid pitfalls and failures made by previous entities. Regulatory reporting requires proper data management to ensure accuracy in disclosing activities to authorities. Data management also includes strict controls for access to data and hierarchy rules across organizations, including records of access to data.

Another primary requirement for institutions is understanding or knowing those with whom you trade or serve. The degree to which an entity requires disclosure and diligence depends upon the type of service administered. Typically known as know-your-customer (KYC) and know-your-business (KYB), these checks help protect businesses, ensure regulations compliance, and protect against bad actors. Financial institutions or those involved with the transfer or exchange of money have additional legal and monitoring requirements, such as monitoring for money laundering (AML) or suspicious transactions. By their pseudonymous nature, public blockchain networks do not have business logic built into their platforms to manage data, identity, and compliance properly. Unless an available solution exists to address all of these concerns, we will see institutions only adopt blockchain technology to streamline backend business processes rather than as a transformative tool to empower all participants.

2.4 Closed Networks

Permissioned networks, where key stakeholders (or validators) control the network nodes and entrants are provisioned onto the network and managed by stakeholders, are at odds with the original intent for blockchain. Network decentralization is essential to ensure the proper setup

and execution of trustless mechanisms. Without a fair degree of decentralization, or consensus distribution, the ledger may be subject to modification.

If the validators of a closed network are not decentralized and have common interests, ownership and sovereign control of assets are at risk. Without decentralization, whoever runs the nodes owns and has positive control over the network, including the assets generated and maintained on the ledger, as logic can change, and transactions may be submitted on behalf of a user without consent. Furthermore, in private permissioned networks, node operators and stakeholders could construct hierarchy views to ledger data, preventing some participants from some or all transaction views, leading to trust, transparency, and control issues.

The current trend for industry to opt for permissioned over public networks will lead many of the Early Majority and most of the Late Majority to interact with blockchain technology solely through permissioned networks where the node operators (stakeholders) are the custodians maintaining positive control over all network assets. While the improved processes will help accelerate the speed and efficiency with which entities do business, the ideal of the Early Adopters of blockchain will not be realized.

Chapter 3

Other Issues and Trends

3.1 Data

Businesses have developed data management practices where records for companies and consumers are repeatedly duplicated and housed in numerous data centers. This practice has left personal and financial data at risk for exposure. In the last decade, the 50 most significant reported data breaches have exposed nearly 8 billion user accounts with trillions of records in the hands of attackers. These breaches and the subsequent loss of customers, fines, legal fees, and reputation loss have cost businesses and financial institutions over USD 5 billion each year. The millions who have dealt with identity theft ramifications are on the other end of these breaches. The energy expended to protect against and deal with data breaches is immeasurable, representing untold lost productivity globally.

Even with the threat of potential attacks and breaches, corporations are designing platforms to capture, store, process, analyze, and utilize even more consumer data. Companies view every record stored in their databases as potential energy in new monetization schemes. Consumers have an abstract knowledge that the data they share with service providers drives monetization beyond the application interface. Still, the methods and frequency with which this occurs are unknown to the user. These practices create a trust gap between consumers and service providers, which instills an underlying distrust, whether conscious or not, in our internet experience.

3.2 Identity

Furthermore, a trend is emerging toward integrating identity into every action, from financial services, travel, telecommunications, e-commerce, social platforms, healthcare, gaming, and more. While the authors of this paper agree that identity should be the root of all interactions, the implementation method is of concern. Suppose identity is the key in all interactions connected to the internet. In that case, the individual, or business, should have a majority, if not complete control, of their identity. Suppose identity data is left to third parties to manage in a centralized fashion. In that case, abuses will undoubtedly lead to disenfranchisement and non-equitable treatment of people groups based on algorithms, prevailing biases, or plans laid

out by decision-makers.

3.3 Privacy

Privacy, or protecting sensitive identifying information, is a fundamental right that all entities should enjoy. Entities can give up this right by abusing systems or freely sharing information. However, disclosure and use of personal or corporate data should be up to the owner or creator of that data and not used or manipulated without consent. Our current information age is opening us to a reality where privacy is impossible due to the proliferation of collection and processing of our PII, data, and physical and online interactions.

3.4 Decentralized Identifier Ambiguity

Public, decentralized ledgers, databases, and the transactions on them represent an opportunity for consumers and businesses to regain trust in their internet activities. These tools can also empower increased ownership over personal and financial activities and data. Self-sovereign identity has emerged as a potential answer to the identity issue. A significant body of work now exists with many proofs-of-concept, working applications, and technical standards set forth by World Wide Web Consortium (W3C) for Decentralized Identifiers [Con22] (DIDs). However, solutions must include a well-defined structure to ensure data integrity. Sources of identity claims can range from verified issuers to self-issued. Data linked to these claims can go from SOC2-compliant storage to not being stored by any entity or initially non-existent. This ambiguity leads to confusion and misunderstanding over the potential weight of a claim. Unless a previously agreed-upon path where a verifier recognizes and accepts claims from an issuer exists, verifiers will need to expend a great deal of energy to source the validity of a claim. In addition, the lack of precise data management standards for handling, storage, and access can lead to further confusion and friction.

Flexible frameworks for DIDs coupled with pseudonymous blockchains present two issues. First, when an individual or entity can spin up as many claims from as many sources as they choose, the weight of all claims is reduced. Second, suppose an individual's public account holds a DID and data or assets. In that case, the holder of that claim will expose certain identifying information and the state and movement of their assets or transactions to outside parties. This exposure leads, once again, to privacy issues on public networks.

Chapter 4

Solution Overview

4.1 Sovereign System

We present a structured asset custody and data management solution situated between self-sovereign and custodial models. This framework, with its foundation rooted in the unique identity of each participant, provides users complete control of their data and assets, protections against asset loss, account management, and enhanced mechanisms for preserving privacy on public networks. The following is a high-level overview of the Sovereign System.

4.1.1 Initialization

A structured approach outlines each network function and each entity's specific role within the network. Each entity must meet particular requirements to interact with the system.

4.1.2 Registration

Regardless of function, each entity that joins the network is considered a Member of the network. Each Member must act in good faith to support the network and its Members. Each Member must complete a unique identification process, resulting in a unique identity (uID) on-chain. This process occurs in a closed channel between a Validator and a potential Member. The purpose of the process is to ensure that each Member has only one individual account on the network. This process ensures network integrity and validates the network as a trust layer over public blockchains.

4.1.3 Secure communication between Member and Validator

There is end-to-end secure communication between Member and Validator. Namely, no other entity can engage in or access information, identifying or otherwise, sent between these two entities.

4.1.4 Verification Checks by the Network Participants

Upon creating uID, the Validator returns the encrypted identity credentials to the Member. The Member and Validator send zero-knowledge proofs to the Producer that this event was successful and request the Producer admit a new Member to the network. Before admission, the Producer reviews and verifies claims made by the Validator and Member.

4.1.5 Key Share of Main Account through Personal Security Questions

In addition to the proof of uniqueness, the Member also commits to a key share of their Main Account with participating Organizations to ensure fallback measures are in place to recover their accounts if they lose access to their private keys. Key share generation includes some personal and confidential information of the Member. The key share design ensures that even if all participating Organizations collude to gain access to a Member's accounts, they do not possess enough key shares to accomplish this action. Involving organizations prevents any internal or external adversaries to brute-force/attack to the key shares of the Main Account.

4.1.6 Creation of DID and its Protection through On-Chain

After confirming the matching claims made by the Member, Validator, the Producer, and the Organizations, the Producer creates and submits a DID object to decentralized storage containing all the proofs, encrypted identity credentials, encrypted key share information, and all other general setup items. Anyone can publicly verify that each process was calculated correctly. The Producer then submits a transaction on-chain to register this Content IDentifier (CID), a hash of the DID. This certification establishes an official record on-chain of the Main Account as part of the network.

4.1.7 Accumulation of Main Account DID objects through Merkle Trees

Main Account DID objects (i.e., the public key of the Main Account) are added to a universal Merkle Tree (called DID Merkle Tree). Namely, whenever a new Member joins the network, its public key is added to the DID Merkle Tree, and the new DID Merkle Root is updated on-chain. This way, all users can be integrated and accumulated in a single DID Merkle Root.

4.1.8 Creating Associated Accounts through Private Membership Proof

Once the DID object of a Main Account has been created, its owner can privately use it for different use cases without disclosing any private information about its identity. This privacy-preserving feature is accomplished through the creation and use of Associated Accounts. More concretely, a Member first creates an Associated Account and proves that it is generated deterministically from their Main Account and is also privately linked to the current DID Merkle

Root. In this way, anyone can publicly verify that this new Associated Account is valid and part of the Sovereign System.

4.1.9 Selective Disclosure through Associated Accounts

A Member can re-hash (obfuscate) the Main Account DID object field, which contains their credential attributes and creates proof that this obfuscated value is privately linked to the current DID Merkle Root. The user can now selectively disclose their credentials' attributes (either confidentially or outright) along with authenticity proofs.

4.2 General Assumptions

- The underlying cryptographic primitives (e.g., elliptic curves (ed25519, JubJub), hash functions (SHA256, Poseidon), ZKSNARKs, Σ-proofs, ElGamal Encryption, Pedersen Commitments) are assumed to be secure.
- The participating parties are supposed to utilize secure random number generators.
- The Algorand public blockchain used to implement the Sovereign System is assumed to be secure.

Chapter 5

Cryptographic Background

5.1 Elliptic Curves

As in Bitcoin, secp256k1 [Res10] will be used. The elliptic curve domain parameters over $\mathbb{F}_p$ associated with a Koblitz curve secp256k1 are specified by the sextuple $T = (p, a, b, G, n, h)$ where the finite field $\mathbb{F}_p$ is defined by:

$$\begin{aligned} p &= FFFFFFFF\ FFFFFFFF\ FFFFFFFF\ FFFFFFFF\ FFFFFFFF\ FFFFFFFF \\ &\quad FFFFFFFE\ FFFFFC2F \\ &= 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1 \end{aligned}$$

The curve $E : y^2 = x^3 + ax + b$ over $\mathbb{F}_p$ is defined by:

$$\begin{aligned} a &= 00000000\ 00000000\ 00000000\ 00000000\ 00000000\ 00000000\ 00000000\ 00000000 \\ b &= 00000000\ 00000000\ 00000000\ 00000000\ 00000000\ 00000000\ 00000000\ 00000007 \end{aligned}$$

The base point G in compressed form is:

$$G = 02\ 79BE667E\ F9DCBBAC\ 55A06295\ CE870B07\ 029BFCDB\ 2DCE28D9\ 59F2815B\ 16F81798$$

and in uncompressed form is:

$$G = 04\ 79BE667E\ F9DCBBAC\ 55A06295\ CE870B07\ 029BFCDB\ 2DCE28D9\ 59F2815B\ 16F81798\ 483ADA77\ 26A3C465\ 5DA4FBFC\ 0E1108A8\ FD17B448\ A6855419\ 9C47D08F\ FB10D4B8$$

Finally the order n of G and the cofactor are:

$$\begin{aligned} n &= FFFFFFFF\ FFFFFFFF\ FFFFFFFF\ FFFFFFFE\ BAAEDCE6\ AF48A03B\ BFD25E8C\ D0364141 \\ h &= 01 \end{aligned}$$

5.2 Hash Functions

A hash function

$$H : \{0,1\}^* \to \{0,1\}^k$$

takes a string of arbitrary length as input and produces a fixed-size output k (i.e., hash function "compresses" the input).

On a high level, it has the following properties:

- **Deterministic:** The same message always outputs in the same hash.
- **Efficient:** It is very fast to compute the hash value for any given message.
- **Non-Correlation of Input and Output (Avalanche effect):** A small change in a message should change the hash value dramatically such that the new hash value is indistinguishable from the old hash value (i.e., statistically indistinguishable from random).
- **One-way (Second Preimage Resistance):** Given $H(x)$, it is infeasible to find x except by trying all possible messages (except with negligible probability).
- **Collision Resistance (Strong Collision Resistance):** No efficient adversary can find two arbitrary messages x and y such that $x \neq y$ and $H(x) = H(y)$.

SHA256 is used in Bitcoin while Keccak (SHA-3) is used in Ethereum. However, these functions are extremely expensive inside a SNARK circuit. Therefore, in practice, SNARK frindly hash functions such as Poseidon [GKR+21], MIMC [AGR+16], Pedersen hash [ZCa17], and Sinsemilla [ZCa21] are quite efficient compared to SHA family hash functions. Note that Sinsemilla is an instance of the Pedersen hash function optimized for table lookups in custom gates. The security properties of Sinsemilla are similar to Pedersen hashes, however, it is not designed to be used where a random oracle, PRF, or preimage-resistant hash is required. The only claimed security property of the hash function is collision-resistance for fixed-length inputs.

In the Sovereign System, Poseidon will be used in ZKSNARK proofs due to its security and efficiency features. Also, in case of security questions and answers, slow hashes such as PBKDF2 will be used.

Remark. Whenever needed, output of the has functions can be interpreted as a point on the underlying curve. How this encoding takes place is an implementation matter.

5.3 Pedersen Commitment

5.3.1 Commitment Schemes

Definition 5.3.1. Let $Commit : \{0,1\}^k \times \{0,1\}^* \to \{0,1\}^*$ be a deterministic polynomial time algorithm, where k is a security parameter. A (non-interactive) commitment scheme consists of two protocols between a sender and a receiver:

Commitment phase:

The sender

1. chooses (or is given) a secret message $m \in \{0,1\}^*$.
2. chooses a random secret $r \in \{0,1\}^k$.
3. produces a commitment $Com = Commit(m, r)$ by applying some public method (i.e., the commitment algorithm $Commit$) defined by the scheme.
4. makes Com public.

Reveal phase:

1. The sender reveals m and r.
2. Given Com, m, r, the receiver can check if indeed $Commit(m, r) \stackrel{?}{=} Com$.

5.3.2 Pedersen Commitment Scheme

Pedersen Commitment uses a public group $\mathbb{G}$ of large order q in which the discrete logarithm is hard, and two random public generators G and P (such that the discrete logarithm is not known to any party).

Commitment phase:

The sender

1. chooses (or is given) a secret message $m \in \{0,1\}^*$ taken in some public message space with at least two elements
2. chooses a random secret $r \in \mathbb{Z}_q$
3. produces from that m and r a commitment $Com = Commit(m, r) = mG + rP$.
4. makes Com public

Reveal phase:

1. The sender reveals m and r.
2. Given Com, m, r, the receiver can check if indeed $mG + rP \stackrel{?}{=} Com$.

5.4 Threshold ElGamal Encryption

Suppose we have a threshold ElGamal cryptosystem (t, n) over an Elliptic Curve E_p where p is a large prime and a secret is shared between n parties $\mathcal{P}_i$s while only if t parties could reconstruct it.

5.4.1 Setup and Distribution of Private Key Shares Through a Dealer

The following Feldman's (t, n)-threshold VSS scheme for sharing a secret $s \in \mathbb{Z}_q$ is defined as an extension of Shamir's scheme [Sch22].

Distribution

- G is a generator point of E_p with order q (i.e., $q = |G|$).
- The dealer has a public and private key pair (pk_d, sk_d).
- Each participant already knows pk_d.
- A dealer chooses a polynomial $f(x) = a_0 + a_1 x + \ldots + a_{t-1} x^{t-1} \in \mathbb{Z}_q[x]$ where $a_j \in_R \mathbb{Z}_q$ with $j \in [0, \cdots, t-1]$.
- $X = s \cdot G$ is the public key where $s = a_0$.
- The dealer computes and distributes the secret private share $s_i = f(i)$ to $\mathcal{P}_i$ for all $i \in [1, \cdots, n]$.
- The dealer finally broadcasts the commitments $B_j = a_j \cdot G$ for all $0 \leq j \leq t-1$.
- The dealer broadcasts $Sign_{sk_d}(h_1, \cdots, h_n)$ where $h_i = s_i G$.

Verification of the shares

Upon receipt of share s_i, each participant $\mathcal{P}_i$ verifies its validity by evaluating the following equation:

$$s_i \cdot G = \sum_{j=0}^{t-1} i^j \cdot B_j.$$

Note that this equation holds if $s_i = f(i)$ because

$$s_i \cdot G = f(0) \cdot G = \sum_{j=0}^{t-1} a_j i^j \cdot G = \sum_{j=0}^{t-1} a_j i^j \cdot G = \sum_{j=0}^{t-1} i^j \cdot B_j.$$

Therefore, any attempts at cheating by the dealer can be detected by the participants.

Reconstruction

- The secret $s = f(0)$ is recovered as in Shamir's scheme from t valid shares.

Remark (Reconstruction of the Private Key through Lagrange Interpolation)**.**

$$s = \sum_{i=0}^{t-1} s_i \prod_{j=0}^{t-1} \frac{j}{j-i} \text{ with } i \neq j.$$

5.4.2 Setup and Distribution of Private Key Shares without a Dealer

The goal now is to let parties $\mathcal{P}_1, \cdots, \mathcal{P}_n$ jointly generate the random polynomial $f(x)$. This is basically done by having each party $\mathcal{P}_i$, $1 \leq i \leq n$ picking a random polynomial $f_i(x) \in \mathbb{Z}_q[x]$ of degree at most $t-1$ (to achieve (t, n) threshold security), and then defining $f(x) = \sum_{i=0}^{t-1} f_i(x)$. Let again G be a generator point of E_p with order q (i.e., $q = |G|$).

1. Each party $\mathcal{P}_i$ runs an instance of Feldman's VSS scheme by choosing a random $t-1$-degree polynomial, using $s_i \in \mathbb{Z}$ as a secret value (as described in Section 5.4.1). Namely, Party $\mathcal{P}_i$ plays the role of the dealer, and parties $\mathcal{P}_1, \cdots, \mathcal{P}_n$ play the role of the participants. Hence, $\mathcal{P}_i$ plays a double role, namely as a dealer and as a participant. Let $f_i(x) \in \mathbb{Z}_q[x]$ be a random polynomial of degree $t-1$ generated by a party $\mathcal{P}_i$ where $f_i(x) = a_{i0} + a_{i1}x + \cdots + a_{it-1}x^{t-1}$.

2. Each party $\mathcal{P}_i$ broadcasts the commitments $< B_{ij} = a_{ij} \cdot G : 0 \leq j \leq t-1 >$ along with $s_i \cdot G$, where a_{ij} are the coefficients of its private polynomial f_i and $s_i = f_i(0) = a_{i0}$. Let $H_i = B_{i0}$.

3. For $1 \leq i, j \leq n$, each party $\mathcal{P}_i$ privately shares s_{ij} to party $\mathcal{P}_j$ where s_{ij} denotes the share of s_i. Note that $s_{ij} = f_i(j)$, and since $f(x) = \sum_{i=1}^{t-1} f_i(x)$, it follows that $x_i = f(i)$.

4. Each party $\mathcal{P}_j$ verifies the shares he received by checking for $i = 1, \cdots, n$:

$$s_{ij} \cdot G = \sum_{k=0}^{t-1} j^k \cdot B_{ik}$$

   If a check fails for an index i then $\mathcal{P}_j$ broadcasts a complaint against $\mathcal{P}_i$.

5. Party $\mathcal{P}_i$ reveals share s_{ij} if it receives a complaint against him by party $\mathcal{P}_j$. If any of the revealed shares s_{ij} fails to satisfy previous check, then $\mathcal{P}_i$ is disqualified. Let us define the set $QUAL \neq \emptyset$ as the set of qualified players.

6. Each party $\mathcal{P}_i$ calculates the overall public key $H = \sum_{i \in QUAL} H_i$.

7. Each party $\mathcal{P}_i$ sums all its received shares s_{ij} to obtain its share $x_i = \sum_{j=1}^{t-1} s_{ij}$ of the overall private key $x_{org} = \sum_{i \in QUAL} s_i$ (which is not known by anyone).

8. Each party $\mathcal{P}_i$ broadcasts its verification key $vk_i = \sum_{i \in QUAL} s_{ij} \cdot G = x_i \cdot G$ and $Sign_{sk_{\mathcal{O}_i}}(vk_i)$ where $sk_{\mathcal{O}_i}$ is its signing private key which was certified by a Trusted Certificate Authority.

5.4.3 Threshold ElGamal Encryption

The following encryption algorithm outputs $C = Enc_P(m)$ where $H(= x \cdot G)$ is a public key and M is a message over the given field.

1. $r \leftarrow\!\!\$\ \mathbb{Z}_p^*$

2. $A = r \cdot G$
3. $B = M + r \cdot H$
4. return $C = (A, B)$

5.4.4 Threshold ElGamal Decryption

Let $(A, B) = (rG, M + rH)$ be a ciphertext where $r \in \mathbb{Z}_q$. The threshold decryption protocol is follows:

1. Each party $\mathcal{P}_i$ takes A as input and uses its private share x_i to produce $d_i = x_i \cdot A$ along with a Σ-proof showing that $\log_G H_i = \log_A d_i$ (using EQ proofs presented in Section 6.5).
2. Let Q be a set of t parties who produced valid d_i values. Then, the plaintext M can be recovered by computing:

$$B - \sum_{i \in Q} \lambda_{Q,i} \cdot d_i = B - x \cdot A = M.$$

5.5 Elliptic Curve Integrated Encryption Scheme (ECIES)

Let $H : \mathbb{G} \to \{0,1\}^k$ be a hash function that maps the elements of $\mathbb{G}$ to bit strings of length k. The hashed ElGamal encryption algorithm is as follows [GKR04]:

5.5.1 Setup

1. Generate a random number $x \in_R \mathbb{Z}_p$ and compute $x.G$.
2. The public and private key pair is $(pk, sk) = (x.G, x)$.

5.5.2 Encryption

Given the public key pk and a message m:

1. Generate a random number $s \in_R \mathbb{Z}_p$ and compute $R = s \cdot G$.
2. Compute the shared secret P_x where $P = (P_x, P_y) = s.pk$.
3. Compute $k_E || k_M = Hash(P_x)$ where Hash is a hash function like SHA512.
4. Compute the ciphertext as $C = Enc_{k_E}(m)$ where Enc is a symmetric encryption scheme like AES256.
5. Compute the tag of the encrypted message $d = MAC_{k_M}(C)$ where MAC is a hash-based message authentication code like HMAC-SHA256.
6. Output $R||C||d$.

5.5.3 Decryption

Given the private key sk and a ciphertext $R||C||d$:

1. Derive the shared secret P_x where $P = (P_x, P_y) = xR$.
2. Derive the keys: $k_E||k_M = Hash(P_x)$
3. Use MAC to check the tag and output failed if $d \neq MAC_{k_M}(C)$.
4. Decrypt the ciphertext: $m = Dec_{k_E}(C)$.

Remark. The Sovereign System uses ECIES due to its practical ECC encryption scheme and strong security property. The Sovereign System extends ECIES to achieve (2,2) encryption scheme where two entities public keys' are used simultaneously. The entities individually could only compute the private shares (but not the whole key) during the protocol executions and are expected to share them with users privately, so that they can recombine and obtain the whole key.

5.6 Digital Signatures

5.6.1 Edwards-Curve Digital Signature Algorithm (EdDSA)

EdDSA [JL17] is currently used by Cardano, NANO, Stellar Lumens, WAVES, and Libra, and will be supported by many more blockchains. Compared to ECDSA's signing and verification steps, EdDSA is simpler and easier to understand and to implement. Both signature algorithms have similar security strength for curves with similar key lengths. The EdDSA algorithm is slightly faster than ECDSA for the most popular curves like edwards25519 and edwards448. Unlike ECDSA, it is not possible for EdDSA to recover the signer's public key from the signature and the message.

EdDSA is a digital signature system with 11 parameters. The generic EdDSA digital signature system with its 11 input parameters is not intended to be implemented directly. Choosing parameters is critical for secure and efficient operation. Instead, you would implement a particular parameter choice for EdDSA (such as Ed25519 or Ed448). Therefore, a precise explanation of the generic EdDSA is thus not particularly useful for implementers. For background and completeness, a succinct description of the generic EdDSA algorithm is given on RFC8062 [JL17].

Public Parameters

- Finite field $\mathbb{F}_q$ over odd prime power q.
- Elliptic curve E over $\mathbb{F}_q$ whose group $E(\mathbb{F}_q)$ of $\mathbb{F}_q$-rational points has order $\#E(\mathbb{F}_q) = 2^c\ell$ where ℓ is a large prime and 2^c is called the cofactor. [1].

- of base point $G \in E(\mathbb{F}_q)$ with order ℓ.
- of hash function H with $2b$-bit outputs, where $2^{b-1} \geq q$ so that elements of $\mathbb{F}_q$ and curve points in $E(\mathbb{F}_q)$ can be represented by strings of b bits.

Keys

- Secret key: b-bit random string x (where $b = 256$).
- $H(x)$= SHA512$(x) = (h_0, \ldots, h_{2b-1})$.
- Derive integer $s = 2^{b-2} + \sum_{3 \leq i \leq b-3} 2^i h_i$ (s is a multiple of 8).
  - Use $H_L(x)$ the first half of $H(x)$ to generate the public key by setting the first three bits of the first octet and the last bit of the last octet to zero and setting the second last bit of the last octet to one. Hence, we set $h_0 = h_1 = h_2 = h_{b-1} = 0$ and $h_{b-2} = 1$. Determine from this new bit string as an integer $s \in \mathbb{F}_q$ using little-endian convention.
- Compute $P = s.G$.
  - The public key is encoded as compressed EC point: the y-coordinate, combined with the lowest bit (the parity) of the x-coordinate. For Ed25519 the public key is 32 bytes. For Ed448 the public key is 57 bytes
- Public key: Encoding $\tilde{P}$ of $P = (x_P, y_P)$ as y_P and one (parity) bit of x_P (needs b bits).
- Compute P from $\tilde{P}$: $x_P = \pm \sqrt{(y_P^2 - 1)/(dy_P^2 + 1)}$.
  - The private key is generated from a random number, called seed (which should have similar bit length, like the curve order). The seed is first hashed, then the last few bits, corresponding to the curve cofactor (8 for Ed25519 and 4 for X448) are cleared, then the highest bit is cleared and the second highest bit is set. These transformations guarantee that the private key will always belong to the same subgroup of EC points on the curve and that the private keys will always have similar bit length (to protect from timing-based side-channel attacks). For Ed25519 the private key is 32 bytes. For Ed448 the private key is 57 bytes.

HashEdDSA Signing

1. Input: (secret key x, message: M).
2. Compute $(h_0, \ldots, h_{2b-1})$ =$H(x)$.
3. Derive integer $s = 2^{b-2} + \sum_{3 \leq i \leq b-3} 2^i h_i$ (s is a multiple of 8).
4. $H_R(x) = h_b || h_{b+1} || \cdots || h_{2b-1}$

5. Deterministically compute $r = SHA512(H_R(x)||SHA512(M)) \in \{0, \cdots, 2^{2b} - 1\}$.
   - This r will be 64-octets long, and we treat it as a little-endian integer modulo q.
6. Calculate the public key point behind r by multiplying it by the curve generator: $R = r.G$.
7. Calculate $h = SHA512(\tilde{R}||\tilde{P}||SHA512(M))$.
8. Calculate $S = r + hs \mod \ell$.
9. Return the signature $(\tilde{R}, \tilde{S})$, with $\tilde{S}$ the b-bit encoding of S
   - The digital signature is 64 bytes (32 + 32 bytes) for Ed25519 and 114 bytes (57 + 57 bytes) for Ed448 (confirming that the signer knows m and x).

Verification

1. Signature $(\tilde{R}, \tilde{S})$, public key $\tilde{P}$, message M
2. Parse P from $\tilde{P}$, R from $\tilde{R}$, and S from $\tilde{S}$.
3. Verify that s lies in the half open interval $[0, \ell]$.
4. Calculate $h = SHA512(\tilde{R}||\tilde{P}||SHA512(M))$.
5. Calculate $2^3S.G = 2^3.R + 2^3h.P$.
6. Check $S' \stackrel{?}{=} S$ in E (and reject if parsing fails or equation does not hold).

Edwards Curves: Point Addition for $q = 5 \mod 8$

The following algorithm is taken from [JL17, Section 5.1.4]. For point addition, the following method is recommended. A point (x, y) is represented in extended homogeneous coordinates (X, Y, Z, T), with $x = X/Z$, $y = Y/Z$, $x * y = T/Z$.

The neutral point is $(0, 1)$, or equivalently in extended homogeneous coordinates $(0, Z, Z, 0)$ for any non-zero Z.

The following formulas for adding two points, $(x3, y3) = (x1, y1) + (x2, y2)$, on twisted Edwards curves with $a = -1$, square a, and non-square d are described in Section 3.1 of [HWCD08, HD08]. They are complete, i.e., they work for any pair of valid input points.

- $A = (Y_1 - X_1) * (Y_2 - X_2)$

- $B = (Y_1 + X_1) * (Y_2 + X_2)$
- $C = T_1 * 2 * d * T_2$
- $D = Z_1 * 2 * Z_2$
- $E = B - A$
- $F = D - C$
- $G = D + C$
- $H = B + A$
- $X3 = E * F$
- $Y3 = G * H$
- $T3 = E * H$
- $Z3 = F * G$

Edwards Curves: Doubling

The following algorithm is taken from [JL17, Section 5.1.4]. For point doubling, $(x_3, y_3) = (x_1, y_1) + (x_1, y_1)$, one could just substitute equal points described in Section 5.6.1 (because of completeness, such substitution is valid) and observe that four multiplications turn into squares.

- $A = X_1^2$
- $B = Y_1^2$
- $C = 2 * Z_1^2$
- $H = A + B$
- $E = H - (X_1 + Y_1)^2$
- $G = A - B$
- $F = C + G$
- $X_3 = E * F$
- $Y_3 = G * H$

- $T_3 = E * H$
- $Z_3 = F * G$

Edwards Curves: Modular Inversion mod $p = 2^{448} - 2^{224} - 1$

For inversion modulo $p = 2^{448} - 2^{224} - 1$, it is recommended to use the identity $x^{-1} = x^{p-2}$ mod p. Inverting zero should never happen, as it would require invalid input, which would have been detected before, or would be a calculation error.

5.6.2 Edwards Curves: Square root for $q = 5 \mod 8$

The following algorithm is taken from [AFH20, Appendix F.2 q = 5 (mod 8)].

Parameters: $\mathbb{F}$, a finite field of characteristic p and order $q = p^m$.

- **Input:** x, an element of $\mathbb{F}$.
- **Output:** z, an element of $\mathbb{F}$ such that $z^2 \stackrel{?}{=} x$, if x is square in $\mathbb{F}$.

Constants:

1. $c_1 = sqrt(-1) \in \mathbb{F}$, i.e.,$c_1^2 == -1$ in $\mathbb{F}$.
2. $c_2 = (q + 3)/8$. # Integer arithmetic

Procedure:

1. $tv_1 = x^{c_2}$
2. $tv_2 = tv_1 * c_1$
3. $e = (tv_1^2) == x$
4. $z = CMOV(tv_2, tv_1, e)$ # If e is False, $CMOV$ returns tv_2, otherwise it returns tv_1.
5. return z

To prevent against timing attacks, this operation must run in constant time, without revealing the value of c. Commonly, implementations assume that the selector c is 1 for True or 0 for False. In this case, given a bit string C, the desired selector c can be computed by OR-ing all bits of C together. The resulting selector will be either 0 if all bits of C are zero, or 1 if at least one bit of C is 1.

Ed25519

Ed25519 is the underlying curve of the EdDSA signature scheme using SHA512 and Curve25519 [LHT16] where

- $q = 2^{255} - 19$ (p of edwards25519 in [LHT16])
- c: base 2 logarithm of cofactor of edwards25519 [LHT16] (i.e., 3).
- $n = 254$
- $E/\mathbb{F}_q$ is the twisted Edwards curve

$$-x^2 + y^2 = 1 - \frac{121665}{121666}x^2y^2$$

- d of edwards25519 in [LHT16] (i.e., `-121665/121666=37095705934669439343138083508754565189542113879843219016388785533085940283555`)
- $a = -1$
- $B = (X(P), Y(P))$ of edwards25519 in [LHT16] (i.e., (`15112221349535400772501151409588531511454012693041857206046113283949847762202`,`46316835694926478169428394003475163141307993866256225615783033603165251855960`))
- ℓ: the order of edwards25519 in [LHT16] where $\ell = 2^{252} +$ `27742317777372353535851937790883648493`.
- B is the unique point $E(\mathbb{F}_q)$ whose y coordinate is $4/5$ and whose x coordinate is "positive" where "positive" is defined in terms of bit-encoding.
  - "positive" coordinates are even coordinates (least significant bit is cleared)
  - "negative" coordinates are odd coordinates (least significant bit is set)
- H is SHA-512 ($SHA - 512(dom2(phflag, context)||x)$ [Hr11]).
- $b = 256$.
- $PH(x) = x$ (identity function).

The curve $E(\mathbb{F}_q)$ is birationally equivalent to the Montgomery curve known as Curve25519. The equivalence is

$$x = \frac{u}{v}\sqrt{-486664},$$

$$y = \frac{u-1}{u+1}$$

Ed448

Ed448 is the underlying curve of the EdDSA signature scheme using SHA512 and Curve25519 [LHT16] where

- $q = 2^{448} - 2^{224} - 1$ (p of edwards448 in [LHT16])
- c: base 2 logarithm of cofactor of edwards448 [LHT16] (i.e., 2).
- $n = 447$
- d of edwards448 in [LHT16] (i.e., - -39081)
- $a = 1$
- $B = (X(P), Y(P))$ of edwards448 in [LHT16] (i.e., (`224580040295924300187604334099896036246789641632564134246125461686950415467406032909029192869357953282578032075146446173674602635247710,298819210078481492676017930443930673437544040154080242095928241372331506189835876003536878655418784733982303233503462500531545062832660`))
- ℓ: the order of edwards448 in [LHT16] where $\ell = 2^{446}$ - 138180668098951153520073867485154268803366924748821786098945475038 85.
- H is $SHAKE256(dom4(phflag, context)||x, 114)$ where SHAKE256(x, y) is the y first octets of SHAKE256 [FP18] output for input x.
- $b = 456$.
- $PH(x) = x$ (identity function).

The SHA-3 family consists of four cryptographic hash functions, called SHA3-224, SHA3-256, SHA3-384, and SHA3-512, and two extendable-output functions (XOFs), called SHAKE128 and SHAKE256. Ed448 uses SHAKE256 as a hash function, even if SHAKE256 is specifically defined not to be a hash function. The first potentially troublesome property is that shorter outputs are prefixes of longer ones. This is acceptable because output lengths are fixed. The

second potentially troublesome property is failing to meet standard hash security notions (especially with preimages). However, the estimated 256-bit security level against collisions and preimages is sufficient to pair with a 224-bit level elliptic curve [JL17]

5.7 Recommended Curve Sizes [MSS17], [BD18]

- **Pairing Friendly Curves:** Except the broken parameters, Tate and Ate pairings presented in Table A.3 in [IEE13] can be chosen for the curves $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$. Apart from BN curves, BLS12 and KSS curves are also recommended to be used for efficiency reasons [MSS17, Section 1], [BD18, Section 2].
- **The current state of-the-art Curve Sizes:** According to the most recent paper of Menezes and Barbulescu [MSS17, Section 6.3], for the 128 bit security, if BN curve with embedding degree 12 is used then the size of $\mathbb{G}_1$ should be 383 bits and the size of $\mathbb{G}_2$ should be 4596 bits [2].
- Elliptic curve points: $P \in \mathbb{G}_1 = E(\mathbb{F}_q)$, and $Q \in \mathbb{G}_2 = E(\mathbb{F}_{q^k})$.
- Evaluates $e(P, Q)$ as an element in the multiplicative group $\mathbb{F}_{q^k}$.
- The order n is a large prime number such that $n|\#E(\mathbb{F}_q)$ and $gcd(n, p) = 1$. The integer k is known as the embedding degree of the curve and is the smallest integer k such that $n|q^k - 1$.

5.7.1 Jacobian coordinates

It is often convenient to represent points (x, y) on $E(\mathbb{F})$ in Jacobian projective coordinates $(X : Y : Z)$, which are defined as follows:

$$(x, y) \to (x : y : 1)$$
$$(X : Y : Z) \to (X/Z^2, Y/Z^3)$$

Projective coordinates are generally useful to avoid computing inversions, and Jacobian coordinates give some of the fastest group operations for this curve shape. Moreover, the point addition law in this representation is independent of the coefficients of the curve equation.

5.7.2 Parameters

- $u = -0xd201000000010000$
- $k = 12$
- $q =$ `0x1a0111ea397fe69a4b1ba7b6434bacd764774b84f38512bf6730d2a0f6b0f6241eabfffeb153ffffb9feffffffffaaab`

[2] The parameters in *IEEE Std 1363.3-2013: IEEE Standard for Identity-Based Cryptographic Techniques using Pairings* are outdated and no longer correct. For example, if BN curve with embedding degree 12 was used then 256 bits for $\mathbb{G}_1$ and 3072 for $\mathbb{G}_2$ would be used. These sizes are no longer secure.

- $r =$ `0x73eda753299d7d483339d80809a1d80553bda402fffe5bfeffffffff00000001`

$$E(\mathbb{F}_q) := y^2 = x^3 + 4$$

$$\mathbb{F}_{q^2} := \mathbb{F}_q[i]/(x^2 + 1)$$
$$E'(\mathbb{F}_{q^2}) := y^2 = x^3 + 4(i + 1)$$

5.7.3 Sizes:

- **private key (32 bytes):** Big endian integer.

  The private key is just a scalar that your raise curve points to the power of. The subgroup order for $\mathbb{G}_1$ and $\mathbb{G}_2$ is r 2^{255}, so for private keys higher than this the point just wraps around. Therefore, useful private keys are $< 2^{255}$ and fit into 32 bytes.

- **pubkey (48 bytes):** 381 bit affine x coordinate, encoded into 48 big-endian bytes. Since we have 3 bits left over in the beginning, the first bit is set to 1 if and only if y coordinate is the lexicographically largest of the two valid ys. The public key fingerprint is the first 4 bytes of $SHA256(serialize(pubkey))$.

- **signature (96 bytes):** Two 381 bit integers (affine x coordinate), encoded into two 48 big-endian byte arrays. Since we have 3 bits left over in the beginning, the first bit is set to 1 if and only if the y coordinate is the lexicographically largest of the two valid ys. (The term with the i is compared first, i.e. $3i + 1 > 2i + 7$). The second bit is set to 1 if and only if the signature was generated using the prepend method, and should be verified using the prepend method.

  The signature is a point on the $\mathbb{G}_2$ subgroup, which is defined over a finite field with elements twice as big as the $\mathbb{G}_1$ curve ($\mathbb{G}_2$ is over $\mathbb{F}_{q^2}$ rather than $\mathbb{F}_q$. $\mathbb{F}_{q^2}$ is analogous to the complex numbers).

5.8 Ed25519 Clamping and Selection of Scalar to make compatible with JubJub

5.8.1 Selection of Scalar in ED25519

In Ed25519 signatures, some "bit-twiddling" is done on the private key and this process is called "clamping". See [Cra20] for further details. Clamping is the action of applying some deterministic manipulation to some input bytes, typically using bitwise operations. It can be used for many purposes, but one common use is to force arbitrary values into a particular integer range by setting or zeroing bits in a particular way.

First, in the key generation, it derives the public key using the following logic. Ed25519 also requires that you SHA512 the seed and use the first 32-bytes for the clamp. That step is done simply to protect against bad seed randomness which is non-uniform.

```
x||y := sha512.Sum512(seed)
x[0] &= 248
x[31] &= 127
x[31] |= 64
```

Next, we calculate $scalar = x \mod ord(ed25519)$ in the standard reference implementation. However, if *scalar* is larger than the order of the subgroup for JubJub (i.e., $\ell = 2736030358979909402780800718157159386076813972158567259200215660948447373041$), then start over with a new random seed to calculate the scalar.

Clearing the lowest three bits. In the first line, we are "clearing" or "zeroing" the lowest three bits of the first byte. This works because if we & with 0, it will always be 0 in the output, but if we & with 1, it will retain whatever value was already there.

Remark. Cleaning these bits does something known as "clearing the cofactor". A cofactor is one of the parameters that make up an elliptic curve. The number of points on the elliptic curve can be described as $= r * h$ where r is the prime order of a subgroup and h is the cofactor. If the cofactor is 1, like in many of the standardized curves, we do not need to worry about. However, if it is not one, careful consideration needs to be made for its implications in cryptographic schemes to avoid a few types of attacks, notably small-subgroup attacks and more nuanced malleability attacks that affected Monero.

Setting the highest bit. The second line and third line work together to clear the 256th bit and set the 255th bit to 1. The second line does the clearing using the same logic as the first and the third uses | to "set" a specific bit. It works using similar logic to how & works, if we | with 0 we retain whatever value was there, and if we | with 1 it will always produce 1, so we can set a specific bit.

Remark. The purpose of this is to ensure that the highest bit is always at a fixed position. X25519 only deals in x-coordinates and there is a simple & efficient way to implement scalar multiplication of x-coordinates known as the Montgomery ladder. The problem with this is that some implementations implement it in variable-time based on the position of the highest bit. To avoid implementations having to care about this the creator of the ed25519 (Bernstein) decided to make this part of the standard so that if the implementation is variable time in that way, it will run in constant time because of this clamping.

5.8.2 Additional Changes in JubJub Clamping

JubJub Clamping is currently using Blake function is the reference implementation, However, in the Sovereign system, we replace SHA512 with the Blake function.

Chapter 6

Honest-Verifier Zero Knowledge (Σ-proofs)

See [Sch22] for further details of the protocols presented below.

6.1 Schnorr's Protocol: Proving the knowledge of r such that $P = rG$

Proof Generation

`eqProofGen(public keys:` P, G`,`
`random values:` r`)`

1. $u_1 \leftarrow\!\!\$\ \mathbb{Z}_n^*$
2. $u_2 \leftarrow\!\!\$\ \mathbb{Z}_n^*$
3. $A_1 = u_1 G$
4. $A_2 = u_2 G$
5. $c_1 = hash(A_1, A_2)$ and $c_2 = hash(A_2, A_1)$.
6. $w_1 = u_1 + c_1 r$
7. $w_2 = u_2 + c_2 r$
8. return (A_1, A_2, w_1, w_2)

Verification

`eqProofVer(proof` (A_1, A_2, w_1, w_2)`,`
`public values` P, G`)`

1. $c_1 = hash(A_1, A_2)$ and $c_2 = hash(A_2, A_1)$.
2. The proof is valid if the following equalities hold, invalid otherwise:
   - $w_1 G = A_1 + c_1 P$
   - $w_2 G = A_2 + c_2 P$

6.2 Proving the equality of messages in different Pedersen Commitments

Prove in (v, r) in $C = vG + rH$ is consistent with that in the second part of the ciphertext $(\alpha, \beta) = (s \cdot G, vG + sP)$.

Proof Generation

`eqProofGen(public keys:` G, P, H, C, β`,`
`random values:` v, r, s`)`

1. $u_1 \leftarrow\!\!\$\ \mathbb{Z}_n^*$
2. $u_2 \leftarrow\!\!\$\ \mathbb{Z}_n^*$
3. $u_3 \leftarrow\!\!\$\ \mathbb{Z}_n^*$
4. $A_1 = u_1 G + u_2 H$
5. $A_2 = u_1 G + u_3 P$
6. $c = hash(G, H, P, C, \beta, A_1, A_2)$.
7. $w_1 = u_1 + vc$
8. $w_2 = u_2 + rc$
9. $w_3 = u_3 + sc$
10. return $(A_1, A_2, w_1, w_2, w_3)$

Verification

`eqProofVer(proof` (A_1, A_2, w_1, w_2)`,`
`public values` G, P, H, C, β`)`

1. $c = hash(G, H, P, C, D, A_1, A_2)$.
2. The proof is valid if the following equalities hold, invalid otherwise:
   - $w_1 G + w_2 H = A_1 + cC$
   - $w_1 G + w_3 P = A_2 + c\beta$

6.3 Proof of Knowledge in a Pedersen Commitment

The following proof proves knowledge of x, y such that $B = xG + yP$, for given B.

Proof Generation

`PedersenProofGen(public keys:` G, P, B`,`
`secret values:` x, y`)`

1. $u_1 \leftarrow\$ \mathbb{Z}_n^*$
2. $u_2 \leftarrow\$ \mathbb{Z}_n^*$
3. $A = u_1 G + u_2 P$
4. $c = hash(A)$
5. $r = u_1 + cx$
6. $s = u_2 + cy$
7. return (A, r, s)

Verification

`PedersenProofVer(proof` (A, r, s)`)`

1. $c = hash(A)$

2. The proof is valid if the following equality holds, invalid otherwise:
   - $rG + sP = A + cB$

6.4 AND Composition of Schnorr's Protocol

We use Schnorr's protocol with equality composition which is basically an and-composition with common witness [Sch22].

Given two relations $R_1 = \{(v_1; w_1)\}$ and $R_2 = \{(v_2; w_2)\}$, a Σ - protocol is obtained for relation $R_1 \wedge R_2 := \{(v_1, v_2; w_1, w_2) : (v_1; w_1) \in R_1, (v_2; w_2) \in R_2\}$ by running a Σ-protocol for R_1 and a Σ-protocol for R_2 in parallel, using a common challenge (assuming that both protocols use the same challenge space). Given two public keys h_1 and h_2, a proof of knowledge of both $\log_g h_1$ and $\log_g h_2$ is obtained, by running two instances of Schnorr's protocol in parallel, using a common challenge.

Proof Generation

`ANDProofGen(public keys:` h_1, h_2`,`
`random values:` r_1, r_2 `)`

1. $u_1 \leftarrow\!\!\$\ \mathbb{Z}_n^*$
2. $u_2 \leftarrow\!\!\$\ \mathbb{Z}_n^*$
3. $A_1 = u_1 G$
4. $A_2 = u_2 G$
5. $c = hash(A_1, A_2)$
6. $w_1 = u_1 + cr_1$
7. $w_2 = u_2 + cr_2$
8. return (A_1, A_2, w_1, w_2)

Verification

`ANDProofVer(proof` (A_1, A_2, w_1, w_2)`)`

1. $c = hash(A_1, A_2)$
2. The proof is valid if the following equalities hold, invalid otherwise:
   - $w_1 G = A_1 + ch_1$
   - $w_2 G = A_2 + ch_2$

6.5 Equality Composition of Schnorr's Protocol

Let $Enc_{pk_{associate}}(m) = (A, B) = (rG, mG + rP)$ where $pk_{associate} = P$ with $P = xG$. We prove that

$$\{(P, G, A, B, m; r) : A = rG, B - mG = rP\}.$$

Given two public values A and $B - mG$, we prove that both $\log_G A$ and $\log_P(B - mG)$ is equal, by running two instances of Schnorr's protocol in parallel, using a common challenge.

Proof Generation

`EQUALITYProofGen(public values:` P, G, A, B, m`,`
`random values:` r `)`

1. $u \leftarrow\$ \mathbb{Z}_n^*$
2. $A_1 = uG$
3. $A_2 = uP$
4. $c = hash(A_1, A_2)$
5. $w = u + cr$
6. return (A_1, A_2, w)

Verification

`ANDProofVer(proof` (A_1, A_2, w)`)`

1. $c = hash(A_1, A_2)$

2. The proof is valid if the following equalities hold, invalid otherwise:
   - $wG = A_1 + cA$
   - $wP = A_2 + c(B - mG)$

6.6 Proving the Equality of the Secret Between JubJub and ed25519

- Let $|\mathbb{G}_1| = p$ and $|\mathbb{G}_2| = q$ where $\mathbb{G}_1$ is the ed25519 and $\mathbb{G}_2$ is the JubJub curve.
- Let n be the bit length of q (since $q < p$).
- Let G_1, H_1 be generators of $\mathbb{G}_1$.
- Let G_2, H_2 be generators of $\mathbb{G}_2$.
- Let $pk = x \cdot G_1$ where $x = x_1||\cdots||x_n$ with $x_i \in \{0, 1\}$. Note that pk is a public key which will be used on Algorand.
- Let $Com_{1,i} = x_i \cdot G_1 + s_i \cdot H_1$ for some randomness s_i.
- Let $Com_{2,i} = x_i \cdot G_2 + t_i \cdot H_2$ for some randomness t_i.
- Given public values $Com_{1,i}, Com_{2,i}$ and private values (x_i, s_i, t_i), we want to prove that

$$(Com_{1,i} = s_i \cdot H_1 \wedge Com_{2,i} = t_i \cdot H_2) \vee (Com_{1,i} = G_1 + s_i \cdot H_1 \wedge Com_{2,i} = G_2 + t_i \cdot H_2).$$

Namely, if $x_i = 0$ then $Com_{1,i} = s_i \cdot H_1$ and $Com_{2,i} = t_i \cdot H_2$, and if $x_i = 1$ then $Com_{1,i} - G_1 = s_i \cdot H_1$ and $Com_{2,i} - G_2 = t_i \cdot H_2$. Therefore, we will prove that

$$(s_i = \log_{H_1} Com_{1,i} \wedge t_i = \log_{H_2} Com_{2,i}) \vee (s_i = \log_{H_1}(Com_{1,i} - G_1) \wedge t_i = \log_{H_2}(Com_{2,i} - G_2)).$$

6.6.1 orProofGen: OR-composition Between JubJub and ed25519

Proof Generation

`orProofGen(public:` $G_1, H_1, G_2, H_2, Com_{1,i}, Com_{2,i}$,
`private:` (x_i, s_i, t_i)

1. If $x_i = 0$
   (a) $c_{2,i}, w_{2,i}, u_{1,i} \in_R \mathbb{Z}_p$
   (b) $c'_{2,i}, w'_{2,i}, u'_{1,i} \in_R \mathbb{Z}_q$
   (c) $A_{1,i} = u_{1,i} \cdot H_1$
   (d) $A_{2,i} = w_{2,i} \cdot H_1 - c_{2,i}(Com_{1,i} - G_1)$
   (e) $A'_{1,i} = u'_{1,i} \cdot H_2$
   (f) $A'_{2,i} = w'_{2,i} \cdot H_2 - c'_{2,i}(Com_{2,i} - G_2)$

(g) $c = SHA256(A_{1,i}, A_{2,i}, A'_{1,i}, A'_{2,i}, Com_{1,i}, Com_{1,i})$

(h) $c_{1,i} = c - c_{2,i}$

(i) $c'_{1,i} = c - c'_{2,i}$

(j) $w_{1,i} = u_{1,i} + c_{1,i}s_i$

(k) $w'_{1,i} = u'_{1,i} + c'_{1,i}t_i$

2. If $x_i = 1$

(a) $c_{1,i}, w_{1,i}, u_{2,i} \in_R \mathbb{Z}_p$

(b) $c'_{1,i}, w'_{1,i}, u'_{2,i} \in_R \mathbb{Z}_q$

(c) $A_{1,i} = w_{1,i} \cdot H_1 - c_{1,i}Com_{1,i}$

(d) $A_{2,i} = u_{2,i} \cdot H_1$

(e) $A'_{1,i} = w'_{1,i} \cdot H_2 - c'_{1,i}Com_{2,i}$

(f) $A'_{2,i} = u'_{2,i} \cdot H_2$

(g) $c = SHA256(A_{1,i}, A_{2,i}, A'_{1,i}, A'_{2,i}, Com_{1,i}, Com_{2,i})$

(h) $c_{2,i} = c - c_{1,i}$

(i) $c'_{2,i} = c - c'_{1,i}$

(j) $w_{2,i} = u_{2,i} + c_{2,i}s_i$

(k) $w'_{2,i} = u'_{2,i} + c'_{2,i}t_i$

3. return $(A_{1,i}, A_{2,i}, A'_{1,i}, A'_{2,i}, c_{1,i}, c_{2,i}, c'_{1,i}, c'_{2,i}, w_{1,i}, w_{2,i}, w'_{1,i}, w'_{2,i})$

Verification $(A_{1,i}, A_{2,i}, A'_{1,i}, A'_{2,i}, c_{1,i}, c_{2,i}, c'_{1,i}, c'_{2,i}, w_{1,i}, w_{2,i}, w'_{1,i}, w'_{2,i})$

1. $c = SHA256(A_{1,i}, A_{2,i}, A'_{1,i}, A'_{2,i}, Com_{1,i}, Com_{2,i})$
2. The proof is valid if the following equalities hold, invalid otherwise:
   - $c_{1,i} + c_{2,i} \stackrel{?}{=} c$
   - $c'_{1,i} + c'_{2,i} \stackrel{?}{=} c$
   - $w_{1,i} \cdot H_1 \stackrel{?}{=} A_{1,i} + c_{1,i} Com_{1,i}$
   - $w'_{1,i} \cdot H_2 \stackrel{?}{=} A'_{1,i} + c'_{1,i} Com_{2,i}$
   - $w_{2,i} \cdot H_1 \stackrel{?}{=} A_{2,i} + c_{2,i}(Com_{1,i} - G_1)$
   - $w'_{2,i} \cdot H_2 \stackrel{?}{=} A'_{2,i} + c'_{2,i}(Com_{2,i} - G_2)$

6.6.2 eqProofGen: EQ proof to use in JubJub: Proving the knowledge of s where $s \cdot H_1 = Com_1 - pk$

Once the OR proof in the previous section has been verified, both the prover and verifier can compute:

1. $Com_1 = \sum_1^n 2^{i-1} \cdot Com_{1,i} = x \cdot G_1 + s \cdot H_1$.
2. $Com_2 = \sum_1^n 2^{i-1} \cdot Com_{2,i} = x \cdot G_2 + t \cdot H_2$.

Next, the prover proves that $Com_1 = x \cdot G_1 + s \cdot H_1$ and $pk = x \cdot G_1$ indeed correct using the following equality proof:

Proof Generation

`eqProofGen(public keys:` Com_1, pk`,`
`random values:` s`)`

1. $u_1, u_2 \leftarrow\!\!\$\ \mathbb{Z}_p$
2. $A_1 = u_1 \cdot H_1$
3. $A_2 = u_2 \cdot H_1$
4. $c_1 = SHA256(Com_1, pk, A_1, A_2)$ and $c_2 = SHA256(A_2, A_1, pk, Com_1)$.
5. $w_1 = u_1 + c_1 s$
6. $w_2 = u_2 + c_2 s$
7. return $(Com_1, pk, A_1, A_2, w_1, w_2)$

Verification$(Com_1, pk, A_1, A_2, w_1, w_2)$

`eqProofVer(public values` Com_1, G_1`)`

1. $c_1 = SHA256(Com_1, pk, A_1, A_2)$ and $c_2 = SHA256(A_2, A_1, pk, Com_1)$.
2. The proof is valid if the following equalities hold, invalid otherwise:
   - $w_1 \cdot H_1 = A_1 + c_1(Com_1 - pk)$
   - $w_2 \cdot H_1 = A_2 + c_2(Com_1 - pk)$

6.6.3 Additional EQ proof to reduce the number of constraints in ZKSNARK: Proving the knowledge of t where $t \cdot H_2 = Com_2 - pk'$

Next, the prover creates $pk' = x \cdot G_2$. This is added to the protocol to reduce the size of the ZKSNARK circuit. pk' will be an additional public input where the number of constraints will be smaller than giving Com_2 as an input. Then, he proves that $Com_2 = x \cdot G_2 + t \cdot H_2$ and $pk' = x \cdot G_2$ indeed correct using the following equality proof:

Proof Generation

`eqProofGen(public keys:` Com_2, pk', `random values:` t`)`

1. $u_1, u_2 \leftarrow\!\!\$\ \mathbb{Z}_q$
2. $A_1 = u_1 \cdot H_2$
3. $A_2 = u_2 \cdot H_2$
4. $c_1 = SHA256(Com_2, pk', A_1, A_2)$ and $c_2 = SHA256(A_2, A_1, pk', Com_2)$.
5. $w_1 = u_1 + c_1 t$
6. $w_2 = u_2 + c_2 t$
7. return $(Com_2, pk', A_1, A_2, w_1, w_2)$

Verification$(Com_2, pk', A_1, A_2, w_1, w_2)$

`eqProofVer(public values` Com_2, G_2`)`

1. $c_1 = SHA256(Com_2, pk', A_1, A_2)$ and $c_2 = SHA256(A_2, A_1, pk', Com_2)$.
2. The proof is valid if the following equalities hold, invalid otherwise:
   - $w_1 \cdot H_2 = A_1 + c_1(Com_2 - pk')$
   - $w_2 \cdot H_2 = A_2 + c_2(Com_2 - pk')$

Chapter 7

Definitions for the Sovereign System

7.1 Unique identification (uID)

Unique identification, or uID, is a process whereby a user presents biometrics and identity documentation to confirm they are a unique Member of the network. The uniqueness identification includes encryption of a Member's data, storage of that data in decentralized storage, creation of a DID object with claims to said data, a zero-knowledge setup which includes fallback authentication for recovery of data and assets, and an on-chain CID record of the DID object. Members performing the initial setup of their uID will have a user experience in the application interface similar to Know-Your-Customer or KYC. Members complete biometric and document submissions as in the KYC process. However, data is managed quite differently in the case of uID. Before being admitted to the network, a Validator must conform to requirements laid out by the Producer to comply with the network's specific requirements. Second, and most importantly, Validators share no collected data collected for uID with any other entity besides the Member. The Member also can request the deletion of all data except for biometrics. Biometrics information will be stored with the Validator for future uID checks. For the proposed framework based on uniqueness, the Validator must adhere to the specific requirements presented in 2.2.2.

7.2 Main Account

The Main Account is the hub for all activity in the sovereign system. The Main Account has two functions. The first function is producing an on-chain link to the CID of the DID object created in the uID process. The second function is to act as the signatory for the Member during the zero-knowledge setup for all Associated Accounts. The Main Account should hold no assets and perform no transactions other than the establishment and maintenance of the DID object connected to the account. With uID linked to the Main Account and all Associated Accounts created through the Main Account, all accounts are recoverable with uID through the Main Account.

7.3 Associated Account

The Member creates Associated Accounts for each specific process employed by a Member. These processes can include holding and sending assets, connecting with peers, connecting to applications, voting, etc. The Main Account signs a transaction authorizing the creation of the Associated Account as part of a zero-knowledge setup of the account. This setup is critical to the recovery methods described in 2.2.2 and allows for the discovery of all Associated Accounts by the Member without having those connections discoverable to outside parties viewing the transactions on the network.

7.4 Self Sovereign versus Sovereign versus Custody

In reference to this work, Self Sovereign is the ability to have complete ownership and positive control over one's data and assets. Self Sovereign implies that the burden for accomplishing this lies solely with the owner and all actions, including custody, is accepted in full by the owner with no recourse in the event of an error on the owner's part.

In the Sovereign System, outside parties can ensure avenues available to protect assets and data from loss or manipulation and create structures to automate complex processes to reduce friction. The owner can rely on assistance from outside parties so long as those outside parties cannot view, access, alter, or transact with the owner's data and assets without the owner's express consent. Further protections, including recourse for transactions, are possible but not explored in this work.

If any outside organization can view, access, alter, or transact an owner's data or assets without the owner's express consent, the Sovereign model is broken and should be considered a custodial model.

Chapter 8

Merkle Tree and Authentication Path

The Sovereign System uses Merkle Tree and Authentication path to allow users to generate Associated Accounts without disclosing the link between the Main Accounts [GJ20, sem22].

First of all, a complete binary tree T is said to have height H if it has 2^H leaves, and 2^H -1 interior nodes. By labelling each left child node with a "0" and each right child with "1". Furthermore, a Merkle hash tree is a complete binary tree equipped with a function hash and an assignment θ which maps the set of nodes to the set of k-length strings:

$$x \to \theta(x) \in \{0,1\}^k.$$

For the two child, x_{left} and x_{right} of any interior node, x_{parent}, the assignment θ is required to satisfy

$$\theta(x_{parent}) = hash(x_{left}||x_{right}).$$

Merkle Tree

`MerkleTree(public: start, maxheight)`

1. Set $leaf = start$ and create empty stack.
2. **Consolidate:** If top 2 nodes on the stack are equal height:
   - Pop node value x_{right} from stack.
   - Pop node value x_{left} from stack.
   - Compute $x_{parent} = hash(x_{left}||x_{right})$.
   - If height of $x_{parent} = maxheight$, output x_{parent} and stop.
   - Push x_{parent} onto the stack.
3. **New Leaf:** Otherwise:

- Compute $x_{left} = LeafCalc(leaf)$ where $LeafCalc$ produces $\theta(x)$ at the cost of single computational unit.
- Push x_{left} onto stack.
- Increment $leaf$.
- Loop to step 2.

Authentication Paths.

The goal of Merkle tree traversal is the sequential output of the leaf values, with the associated authentication data. For each height $h < H$, we define $Auth_h$ to be the value of the sibling of the height h node on the path from the leaf to the root. The authentication data is then the set $\{Auth_i | 0 \leq i < H\}$.

The correctness of a leaf value may be verified as follows: It is first hashed together with its sibling $Auth_0$, which, in turn, is hashed together with $Auth_1$, etc., all the way up to the root. If the calculated root value is equal to the published root value, then the leaf value is accepted as authentic. Fortunately, when the leaves are naturally ordered from left to right, consecutive leaves typically share a large portion of the authentication data.

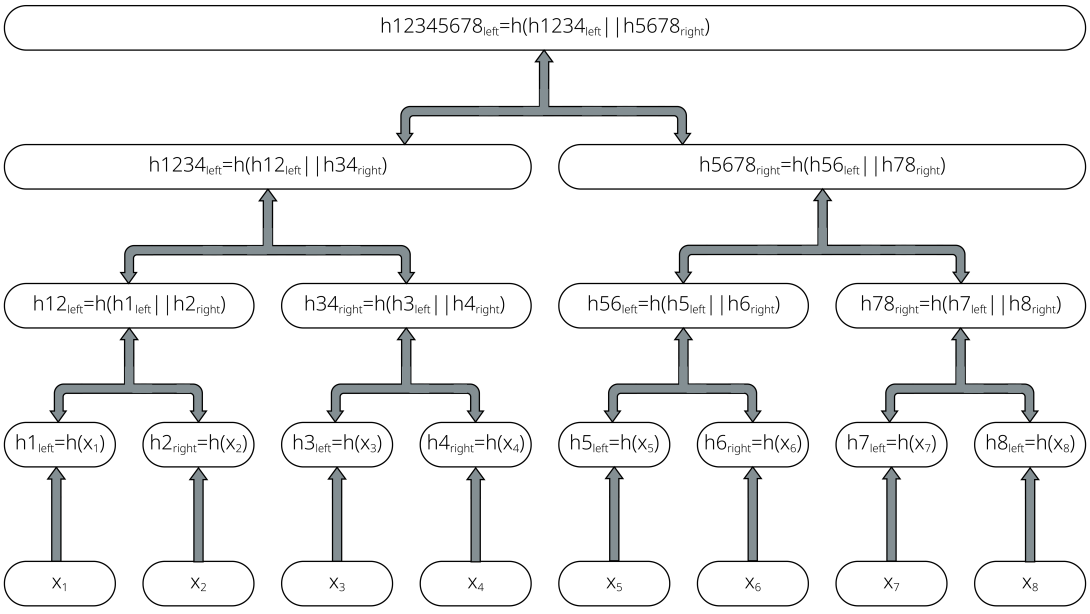

Figure 8.1: Merkle Tree.

Chapter 9

Open Source Crypto Libraries for ZKSNARKs

The Sovereign System uses ZKSNARKs to ensure that participating parties are doing calculations correctly without disclosing any information about their private values. This chapter overviews the details of the implementation of the underlying ZKSNARK proofs.

9.0.1 SNARK friendly curves and hash functions

Remark (SNARK friendly curves)**.** *The Algorand network uses the ed25519 curve for signature generation (in particular, RFC8032 [JL17]). Due to the performance issues, it is very expensive to utilize RFC8032 in the ZKSNARK computations. Therefore, we need to use SNARK friendly curves such as JubJub during the proof generation. In order to utilize JubJub in the proofs, we need to make sure that the private key used in the JubJub curve is the same as in ed25519. The protocol presented in Section 6.6.1 is used to prove this relation.*

Remark (SNARK friendly hash functions)**.** *The ZKSNARK proofs are also very expensive if one uses SHA functions or Blake (as they are used in RFC8032 [JL17]). Therefore, the architecture of the Sovereign system has been designed in such a way that only SNARK friendly hash functions such as Poseidon are going to be used.*

9.1 Circuit Generation: Circom

- `https://github.com/iden3/circomlib`
- It contains the implementation of different cryptographic primitives in Circom language.
- Circuits are basically statements that we need to prove, and they will be written in this language.

9.2 Proof Generation & Verify

- **Rust based compiler:** `https://github.com/iden3/circom`

- **JS based compiler:** `https://github.com/iden3/snarkjs`. This is a JavaScript and Pure Web Assembly implementation of ZKSNARK and PLONK schemes. It uses Groth16 (only 3-point multiplication and 3 pairings) and PLONK.
- `https://github.com/iden3/go-circom-prover-verifier`
- **Go implementation:** ZKSNARK prover & verifier is compatible with Circom. It uses bn256 (used by go-ethereum) for the Pairing curve operations.

Remark. The Sovereign system aims to utilize the Plonk construction due to its advantages such as being trustless, universal, and having an updatable setup.

9.3 Communication: Use gRPC

- **Go:** `https://grpc.io/docs/languages/go/basics/`
- **Node:** `https://grpc.io/docs/languages/node/quickstart/`

9.4 Additional Tutorial & Documentations:

- **ZK Background:** `https://docs.circom.io/background/background/`
- **Circom Language:** `https://docs.circom.io/circom-language/signals/`. For more circuits, `https://docs.circom.io/more-circuits/more-basic-circuits/`.
- **More Documentation:** `https://iden3-docs.readthedocs.io/en/latest/iden3_repos/circom/TUTORIAL.html`
- **A tutorial for using Circom:** `https://hackmd.io/@n2eVNsYdRe6KIM4PhI_2AQ/SJJ8QdxuB`

Chapter 10

The Sovereign System

10.1 Entities

10.1.1 Member

A Member is a unique participant in the public network. Members include individuals, businesses, and institutions (from not-for-profits to government entities). To avoid issues noted previously, having a method to ensure uniqueness is critical to providing structure to the proposed Sovereign System. Each Member has a decentralized identifier (DID) attributed to their Main Account that notates their uniqueness on the network. This DID is the Member's anchor for all web3 data and asset activity.

10.1.2 Issuer

An Issuer is an entity that issues documentation acceptable to a Validator for provision onto the network or addition to a Member's identity vault. An Issuer must be approved by the Producer(s) and any governing Organization(s) before admittance. Formal acceptance of Issuers and the documentation they produce ensures Members, Verifiers, and consumers of DID claims have confidence in the data stored on the network.

10.1.3 Validator

A Validator is an entity that qualifies and authenticates documents and data produced by Issuers, as well as biometric or other identifying data presented to the Validator directly by the Member to verify uniqueness through a process known as unique identification (uID). The Validator must be able to employ specific checks against material presented by the Member including liveness (proof of being alive, present, and not under duress at time of authentication), biometric scans, proof that supporting documentation matches biometrics, and most importantly that a prospective Member does not already have an established main identity account on the network.

Remark. During the initial implementation, Identity Proxy Server (IPS) will be a trusted entity between the Validator and the Producer. In the long-term, the functionalities of IPS are

expected to be integrated with the Validator.

10.1.4 Independent parties: Organization$_1$, $\cdots$, Organization$_n$

An Organization is any network entity that supports and manages the network's operation. In this paper, we explore an implementation with two Organizations, a Producer and a Foundation.

10.1.5 Producer

A Producer is an Organization responsible for properly operating the network by acting as a relayer for the network as it receives data from Members and Organizations and approvals from Validators. The Producer receives these relays and verifies messages through cryptographic and other calculations, produces DIDs, and leads key management. The Producer can be a centralized entity, a centralized entity employing smart contracts for operation, or a fully decentralized set of smart contracts with key management managed by the contract owner(s) or the contracts themselves.

10.1.6 Foundation

A Foundation is an Organization responsible for the proper operation of the network by representing and advocating for the Members of the network, and auditing the Producer. In this paper, we envision the representatives of the Foundation are proposed for short-term appointment to the Foundation by the Producer and the Members of the network who stake dA beyond their Minimum Commitment vote whether to admit each representative through a majority yes or no vote. The Members who stake dA beyond their Minimum Commitment also have the authority to vote a representative out of the Foundation before their term ends. This structure creates a check and balance system between the Organizations and the Members of the network.

10.1.7 Service Providers

Service Providers leverage uID for authentication, Associated Accounts for creating and managing connections to Members of the network, management of user assets, anonymous or user-provisioned KYC and ongoing AML, and attribute custom data to the DID to perform user-owned data operations.

10.2 Trust Assumptions and Requirements

- Only registered users to the Validator are allowed to use the Sovereign System (for the uniqueness of the real users).
- Let $(tpk_{ed25519}, tsk_{ed25519})$ and $(tpk_{jubjub}, tsk_{jubjub})$ be the public and private key pairs of a user's Main Account on Ed25519 and JubJub curves, respectively. Each user will have one single $(tpk_{jubjub}, tsk_{jubjub})$ (similarly, one single $(tpk_{ed25519}, tsk_{ed25519})$) in the system which will be used to create a Main DID object for the user U , called DID_u.

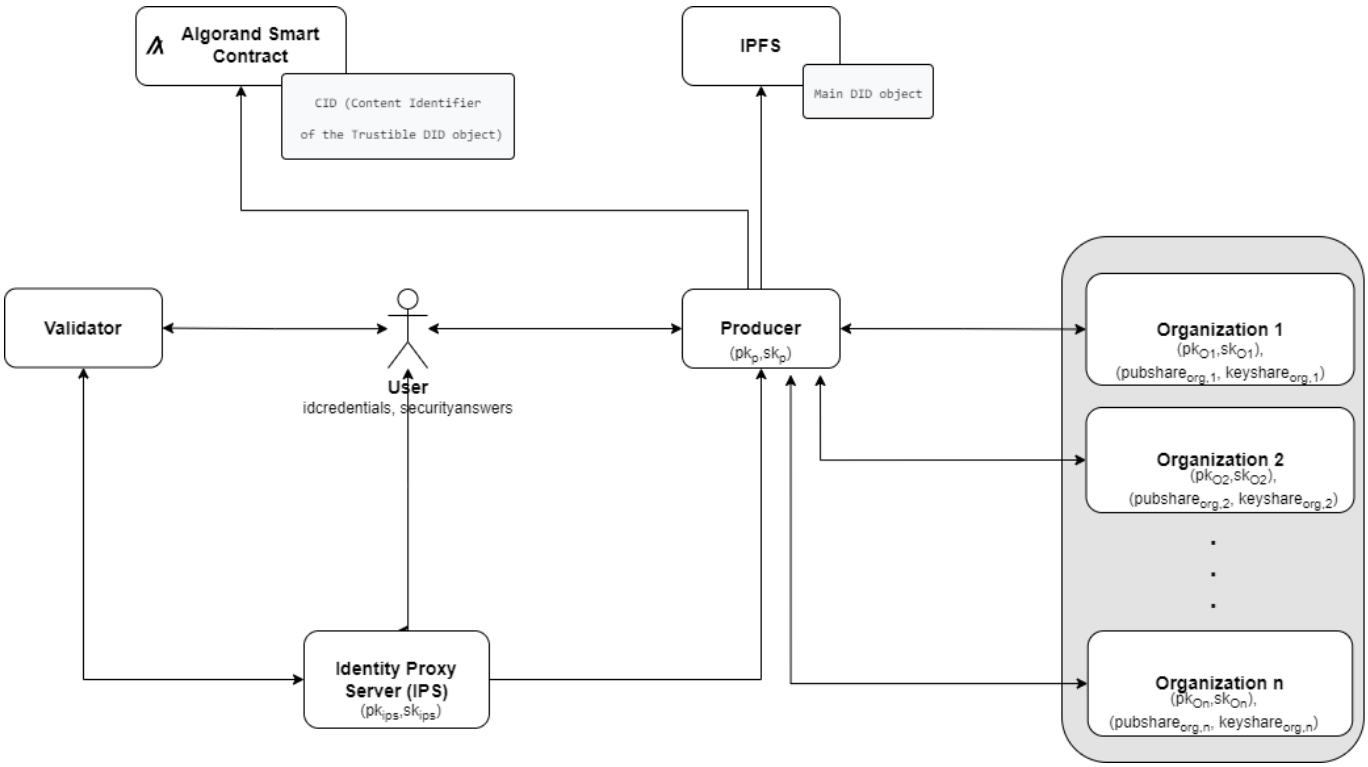

Figure 10.1: A High level Architecture of Sovereign.

  - Note that existing systems like Polygon ID do not consider the uniqueness of the real users. In the Sovereign System, there will be only one main identifying DID object for one single user assuming that the Validator is honest.
  - Also, the user can use the main identifying DID object in other services (such as authentication, voting in electronic polls, and privacy preserving complaint transactions) through selective disclosure by means of only revealing certain information of the credentials. The Main Account which resolves to the main identifying DID object is not exposed in this process.

- Each private key of other chains is encrypted with a newly generated public and private key pair. This new key is deterministically derived from tsk_{jubjub} (which is also equal to $tsk_{ed25519}$).
- The Sovereign System uses (t, n) threshold secret sharing scheme where at least t shares must exist to be able to obtain the private key. Since the Sovereign System always requires user involvement during the key recovery process, the user is supposed to hold the controlling interest of the shares of the private key of the Main Account. Therefore, even if all participating parties (except the user) are malicious and colluding, they will not be able to recover the private key.
- The user's private key shares of tsk_{jubjub} (or $tsk_{ed25519}$) are generated from selected and pre-defined security questions.

  - *A pool of security questions need to be selected carefully.*
  - *Security answers should be difficult to predict. However, they should be easy to remember even years later.*

- The security questions are encrypted through (t, n) threshold encryption scheme (between n organizations). The selected security questions are not known by any party individually. During the registration, they are encrypted through (t, n) encryption scheme. The user is expected to encrypt the questions correctly during the registration.
- If users lose everything (all public and private values), they must
  - provide their credentials to the Validator and answer to a publicly known pre-defined security question correctly to be able to identify the DID object, and
  - answer to the additional security questions which were selected by the user (and stored in the DID object) during the registration.
- Users can only recover their private keys as long as they remember their answers to the selected security questions correctly. Furthermore, their private keys and the private keys of other chains cannot be recovered if the security questions are not answered correctly.
- The overall Sovereign System is resistant against threshold number of corruption. Namely, the Sovereign System can recover all the keys of users if at least one organization is honest or participates in the protocol (in addition to the user).
- The encrypted private keys of other chains are all indistinguishable. Even if DID_u of a user is disclosed, no one can learn any information whether the user has protected his/her private keys of other chains.
- The Producer cannot obtain any private information about users (i.e., private keys or credential attributes).
- Associated Accounts can be proven that they are linked to an existing Main Account without disclosing any information about the users or the Main Account.
- The Validator will be a trusted entity. The Validator is assumed to see the credential details of the users but is not supposed to share with anyone except sharing the validity of the user with the Producer. Note that since Validator sees the identifying information during the protocol executions, it can learn which DID_u belongs to which user. However, it cannot obtain any information about the activity of the user because Associated Accounts cannot be linked to any of the Main Accounts. For example, it also cannot learn whether the user has created an associated or used in a use case scenario (e.g., encrypting a private key, transferring assets, or voting).

  Remark. During the initial implementation, Identity Proxy Server (IPS) will be a trusted entity between the Validator and the Producer. The IPS is assumed to see the credential details of the users as well as which DID_u belongs to which user, however it is should not share with anyone except sharing the validity of the user with the Producer. In this case, the Validators cannot learn which DID_u belongs to which user.

10.3 Unique Identification (uID)

As the Sovereign System involves many parties in a cooperative model, a Member must be able to make authoritative claims to their data and assets. Multiple parties, or Multiple accounts held by one party, could make claims over an account(s) without authentic claims. Adding unique identity to the network helps build a trust layer over all actions taken on the network. uID ensures that a Member may only create one Main Account on the network. Not only is this crucial in helping Members and Organizations transact with greater peace of mind, even when the counterparties are anonymous or pseudonymous, ensuring uID checks are factual is also critical to a tamper-proof recovery process for private keys.

10.3.1 Validator Requirements to ensure uID integrity on the network

- Validators performing services for other entities must run a separate environment to specifically handle the custom workflow of uID to ensure no leakage of personal identifying information, or PII, of the registrants and Members. The Producer must have no access to identifying information.
- The Validator must be able to identify previous Registrants and accepted Members while processing uID biometrics and documentation and appropriately flag the registration attempt to prevent a fictitious or duplicate entry into the network.
- The Validator must return specific data attributes containing personal identifying information, or PII, we term *idcredentials* to the Member's device through secure channels upon confirmation of uniqueness.
- The Validator must accept a signature of the compressed *idcredentials* from the Member.
- The Validator must agree to verify the signed *idcredentials* and present this hashed value with randomness to the Producer for verification.
- If there are multiple Validator's, they must agree to share biometric data to ensure entrants to the network are unique.

10.4 Registration and Creation of a Main DID Object

Let G_1, H_1 be the base elements of ed25519 (denoted by $\mathbb{G}_1$) and G_2, H_2 be the base elements of JubJub (denoted by $\mathbb{G}_2$).

10.4.1 Pre-condition

- Organization$_i$ has a public and private key pair $(pk_{\mathcal{O}_i}, sk_{\mathcal{O}_i})$ for all $i = m, \cdots, n$.
- The Identity Proxy Server has (pk_{ips}, sk_{ips}) and the Producer has (pk_p, sk_p) public and private key pairs.

- The organizations', the Identity Proxy Server's, and the Producer's public keys $pk_{\mathcal{O}_i}$, pk_{ips}, and pk_p are certified by a trusted Certificate Authority (i.e., signed certificates).
- The organizations execute a threshold key generation process as described in Section 5.4.2. At the end of this setup, each Organization$_i$ creates a public and private key pair $(pubshare_{org,i}, keyshare_{org,i})$ with $i = m, \cdots, n$, where $pubshare_{org,i} = keyshare_{org,i} \cdot G_2$, respectively. If the threshold is ℓ, then the system will be $(\ell, n-m+1)$ threshold security. This is required to eliminate the security issues in case of a possible corruption on the organization side. These values will not only make the system more robust but also will be used to identify the user Main DID object deterministically.
- Once user opens the Sovereign browser/application, the certificates of all the organizations are downloaded and verified immediately.

10.4.2 Registration Flow

1. The client app will ask the user to select and answer ℓ security questions

$$securityquestions = question_1 || \cdots || question_\ell.$$

   Let's denote the answers by

$$securityanswers = answer_1 || \cdots || answer_\ell$$

   where $answer_i$ is the answer to $question_i$.

   Remark. *These answers will be part of the Sovereign Private Key tsk_{jubjub} (which is also equal to $tsk_{ed25519}$) as described in the next step. Neither the questions nor the answers are supposed to be seen by anyone individually (including the Validator, the Identity Proxy Server (IPS), and the Producer). This is necessary to prevent a possible corruption on the Validator side (e.g., Onfido) and the Identity Proxy Server (IPS) side, otherwise it would be possible to impersonate users.*

2. The user utilizes (t, n) verifiable secret sharing scheme (e.g., Feldman's VSS scheme) where the user is the dealer and organizations are the members (see [Sch22, Chapter 6.2.1]). The Sovereign system always requires user involvement during the key recovery process. Therefore, initially the Sovereign system will use (5,7) secret sharing scheme where the user holds 3 private key shares while there are also 4 organizations each holding a private key share (i.e., $m = 4$ (organizations), $n = 7$ (total number of users), $t = 5$ (the threshold level)). To construct such a key management structure, the user

   (a) i. generates a new and fresh random number *seed*.
   ii. computes the following:

```
sk||y := sha512.Sum512(seed)
sk[0] &= 248
sk[31] &= 127
sk[31] |= 64

```

iii. calculates $x = sk \mod ord(ed25519)$ in the standard reference implementation.

iv. If x is larger than the order of the subgroup for JubJub (i.e., ℓ= 2736030358979909402780800718157159386076813972158567259200215660948447373041), then start over with a new random seed to calculate the scalar (i.e., go to step 1(b)i.) [1]

(b) generates a public and private key pair $(tpk_{ed25519}, tsk_{ed25519})$ and $(tpk_{jubjub}, tsk_{jubjub})$ such that

- $tsk_{ed25519} = tsk_{jubjub} = x$,
- $tpk_{ed25519} = x \cdot G_1$, and
- $tpk_{jubjub} = x \cdot G_2$.

Note that $(tpk_{ed25519}, tsk_{ed25519})$ and $(tpk_{jubjub}, tsk_{jubjub})$ are the key pairs belong to the main account on the ed25519 and the JubJub curves, respectively.

(c) encrypts *seed* through tpk_{jubjub}.

i. generates a new random key $K \in_R \{0,1\}^{256}$ and computes

$$CT_{seed} = AESEnc_K(seed).$$

ii. generates an ephemeral (fully random) key pair: $(eprivkey_{seed}, epubkey_{seed})$ where

$$epubkey_{seed} = eprivkey_{seed} \cdot G_2.$$

iii. computes the shared key:

$$ss = SHA512((eprivkey_{seed} \cdot tpk_{jubjub}).x)$$

where $(eprivkey_{seed} \cdot tpk_{jubjub}).x$ is the x axis of $eprivkey_{seed} \cdot tpk_{jubjub}$ with $eprivkey_{seed} \cdot tpk_{jubjub} = tsk_{jubjub} \cdot epubkey_{seed}$.

iv. computes the ciphertext: $CT_{seed,key} = AESEnc_{ss_{enc}}(K)$ and $d_{seed,key} = MAC_{ss_{mac}}(CT_{seed,key})$ where $ss_{enc}||ss_{mac} = ss$.

(d) generates a random number $salt_{secanswers}$.

(e) calculates $m-1$ private key shares $(tsk^{u,1}_{jubjub}, \cdots, tsk^{u,m-1}_{jubjub})$ for himself/herself and private key shares $tsk^{\mathcal{O},m}_{jubjub}, \cdots, tsk^{\mathcal{O},n}_{jubjub}$ for organizations as follows:

i calculate $tsk^{u}_{jubjub,secans} = PBKDF2(salt_{secanswers}, securityanswers)$.

ii generate $tsk^{u,i}_{jubjub} = PBKDF2(tsk^{u}_{jubjub,secans}, i)$ for $i = 1, \cdots, m-1$.

iii generate a random number $tsk^{\mathcal{O},m}_{jubjub}$ (to be able to calculate the unique polynomial.).

iv through Lagrange interpolation, calculate the polynomial $p(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{t-1} x^{t-1}$ such that

$$(p(0), p(1), \cdots, p(m-1), p(m)) = (tsk_{jubjub}, tsk^{u,1}_{jubjub}, \cdots, tsk^{u,m-1}_{jubjub}, tsk^{\mathcal{O},m}_{jubjub}).$$

[1] We need this step to make sure that the public keys generated on both curves have the same private keys. See Section 5.8 for the reasoning of the calculations.

v calculate $tsk_{jubjub}^{\mathcal{O},i} = p(i)$ for $i = m+1, \cdots, n$.

where PBKDF2 is a password based key derivation function [2].

Remark. *Note that the constant term of the polynomial $p(x)$ is tsk_{jubjub} (i.e., $a_0 = tsk_{jubjub}$). While computing the shares of the organizations, we need to make sure that none of these shares are equal to a_0. Surely, the probability of being equal is negligible, however, it's still important to remove this exceptional case in the implementation. Namely, if it becomes equal, then we go back to Step 2.b.iii where new fresh random numbers for the organizations are regenerated.*

Remark. The user has now a public and the private key pair $(tpk_{jubjub}, tsk_{jubjub})$ where the shares of tsk_{jubjub} are defined as

$$(tsk_{jubjub}^{u,1}, \cdots, tsk_{jubjub}^{u,m-1}, tsk_{jubjub}^{\mathcal{O},m}, \cdots, tsk_{jubjub}^{\mathcal{O},n})$$

and $tpk_{jubjub} = tsk_{jubjub} \cdot G_2$.

(f) calculates $B_j = a_j \cdot G_2$ for $j = 0, \cdots, t-1$.

(g) The user submits $(name, surname, tpk_{ed25519})$ to the Identity Proxy Server.

3. The Identity Proxy Server

   (a) creates an ApplicantID (through the Validator).

   (b) generates and stores a random number r_{ips} for the user.

   (c) sends the ApplicantID and r_{ips} to the user.

4. The user

   (a) sends the identifying information and the ApplicantID to the Validator.

   (b) informs the Identity Proxy Server about the submission of the identifying information.

5. The Validator checks if the identifying information is validated through the Validator to confirm the uniqueness of the user in the network. The Validator also runs a check to ensure the user is both live and present at time of capture and whether the user had previously registered.

   (a) If the user fails uID for submission, the Validator must alert the user directly to the failure's nature so that the user may attempt to correct the submission error or underlying issue.

[2] Instead of PBKDF2, one can also use other slow hashing-functions Scrypt, Bcrypt, and Argon2. Note that PBKDF2 is commonly implemented as 2048 iterations of hashing to slow down the crackers who have acquired a list of hashes.

(b) If the Validator detects the user is a Member, meaning they have previously completed uID successfully, the Validator will prompt the Application to alert the Member to either log into the Application with their Main Account key or begin the recovery process.

(c) The Validator accepts the Registrant's submission if documentation matches their biometrics, there are no issues with the Registrants documentation, and they have not previously registered with the network through any approved Validator.

(d) The Validator sends the validation result to the Identity Proxy Server.

6. The Identity Proxy Server

(a) checks if the identifying information is validated through the Validator to confirm the uniqueness of the user in the network.

(b) obtains $idcredentials$ from the Validator

(c) calculates $hash_{ips} = Poseidon(idcredentials, tpk_{ed25519}, r_{ips})$

(d) stores $hash_{ips}$ for the user.

(e) calculates $Signature_{ips} = Sign_{sk_{ips}}(hash_{ips})$.

(f) calculates the encryption of $(idcredentials, Signature_{ips})$ using $tpk_{ed25519}$ outputting $Ciphertext_{ips}$.

(g) sends $Ciphertext_{ips}$ to the user.

7. The user

(a) decrypts $Ciphertext_{ips}$ and obtains $(idcredentials, Signature_{ips})$.

(b) validates the credential attributes $idcredentials$ (which is displayed as read only), and if confirmed, calculates $hash_{uid} = Poseidon(idcredentials, tpk_{ed25519}, r_{ips})$.

(c) There will be k additional pre-defined and fixed security questions for everyone (initially, k can be set to 1) [3]. Let's denote $securityquestions_{authrecovery}$ and $securityanswers_{authrecovery}$ for the security questions and their answers, respectively. More concretely, the user

i. verifies $Signature_{ips}$ through pk_{ips}, r_{ips}, $tpk_{ed25519}$, and $idcredentials$.

ii. generates r_u, s_u and calculates

$$hash_u = r_u \cdot Poseidon(idcredentials, securityanswers_{authrecovery})$$ [4]

and

$$hash_{uprivateshare} = s_u \cdot Poseidon(salt_{secanswers}, securityanswers).$$

[3] Note that these additional security questions will be used in the recovery phase to authenticate users in addition to the identifying information. If the answer(s) are given correctly then organizations will be convinced to decrypt the ciphertext of $securityquestions$. To recover the key, the user not only needs to present the identifying information but also needs to answer the security questions correctly.

[4] $hash_u$ may include another private random value like r'_{ips} which belongs to the Validator. For each revocation, a new r_{ips} can be used which could make the system anonymous and indistinguishable.

iii. calculates $hash_{ips}$ (since r_{ips} is already known).

iv. creates $ZKSNARK_{uips}$ to prove that $hash_u, hash_{ips}$, and $hash_{uprivateshare}$ are consistent (i.e., the pre-images of these hashes contain $idcredentials$). More concretely, given public values $hash_u, hash_{ips}, hash_{uprivateshare}, tpk_{jubjub}$, $tpk_{ed25519}$, $salt_{secanswers}$ and private values r_u, s_u, r_{ips}, $idcredentials$, tsk_{jubjub}, $securityanswers_{authrecovery}$, and $securityanswers$ prove that

- $hash_u = r_u \cdot Poseidon(idcredentials, securityanswers_{authrecovery})$
- $hash_{uprivateshare} = s_u \cdot Poseidon(salt_{secanswers}, securityanswers)$
- $hash_{ips} = Poseidon(idcredentials, tpk_{ed25519}, r_{ips})$.
- $tpk_{jubjub} = tsk_{jubjub} \cdot G_2$.

(d) computes

$$CT_{privateshare,i} = Enc_{pk_{\mathcal{O}_i}}(Sign_{tsk_{ed25519}}(hash(tsk^{\mathcal{O},i}_{jubjub})), tsk^{\mathcal{O},i}_{jubjub})$$

for $i = m, \cdots, n$ where $pk_{\mathcal{O}_i}$ denotes the public key of Organization$_i$.

- *Optional:* To detect any possible corruption on the user side, the user can provide an additional proof that $CT_{privateshare,i}$ indeed contains a share being part of tsk_{jubjub}.

(e) encrypts $securityquestions$ through (k, n) encryption scheme between the organizations. Let $H(= x \cdot G_2)$ be the overall public key of the organizations (as described Section 5.4.2).

i. generates a random key $K \in_R \{0,1\}^{256}$ and computes

$$CT_{sq} = AESEnc_K(securityquestions).$$

ii. generates an ephemeral (fully random) key pair: $(eprivkey_{sq}, epubkey_{sq})$ where

$$epubkey_{sq} = eprivkey_{sq} \cdot G_2.$$

iii. computes the shared key:

$$ss = SHA512((eprivkey_{sq} \cdot H).x)$$

where $(eprivkey_{sq} \cdot H).x$ is the x axis of $eprivkey_{sq} \cdot H$ with $eprivkey_{sq} \cdot H = x_{org} \cdot epubkey_{sq}$.

iv. computes the ciphertext: $CT_{sq,key} = AESEnc_{ss_{enc}}(K)$ and $d_{sq,key} = MAC_{ss_{mac}}(CT_{sq,key})$ where $ss_{enc}||ss_{mac} = ss$.

(f) also encrypts the symmetric key K through his/her private key tsk_{jubjub} of the main account as:

i. computes his/her shared key:

$$ss_u = SHA512((eprivkey_{sq} \cdot tpk_{jubjub}).x)$$

where $(eprivkey_{sq} \cdot tpk_{jubjub}).x$ is the x axis of $eprivkey_{sq} \cdot tpk_{jubjub}$ with $eprivkey_{sq} \cdot tpk_{jubjub} = tsk_{jubjub} \cdot epubkey_{sq}$.

ii. computes the ciphertext: $CT_{key,user} = AESEnc_{ss_{u,E}}(K)$
and $d_{key,user} = MAC_{ss_{u,M}}(CT_{key,user})$ where $ss_{u,E}||ss_{u,M} = ss_u$.

(g) calculates

$$\mathsf{CTUser}_{sq} = (CT_{sq}, CT_{sq,key}, d_{key}, CT_{key,user}, d_{key,user}, epubkey_{sq}, salt_{secanswers}).$$

(h) The user computes the proof of relation between JubJub and ed25519 address as follows:

i. computes $Com_{main,1,i} = tsk^i_{jubjub} \cdot G_1 + s_i \cdot H_1$ for some randomness s_i where $tsk_{jubjub} = tsk^1_{jubjub}|| \cdots ||tsk^n_{jubjub}$ with $tsk^i_{jubjub} \in \{0, 1\}$.

ii. computes $Com_{main,2,i} = tsk^i_{jubjub} \cdot G_2 + t_i \cdot H_2$ for some randomness t_i.

iii. creates a proof orProofGen as $\texttt{orProofGen}((G_1,\ H_1,\ G_2,\ H_2,\ Com_{main,1,i},\ Com_{main,2,i}),\ (tsk^i_{jubjub}, s_i, t_i))$.

iv. computes $Com_{main,1} = \sum_1^n 2^{i-1} \cdot Com_{main,1,i} = tsk_{jubjub} \cdot G_1 + s \cdot H_1$.

v. computes $Com_{main,2} = \sum_1^n 2^{i-1} \cdot Com_{main,2,i} = tsk_{jubjub} \cdot G_2 + t \cdot H_2$.

vi. creates a proof eqProofGen1 as $\texttt{eqProofGen}((Com_{main,1}, tpk_{ed25519}),\ s)$.

vii. creates a proof eqProofGen2 as $\texttt{eqProofGen}((Com_{main,2}, tpk_{jubjub}),\ t)$.

viii. sends $< Com_{main,1,i} : i = 1, \cdots, n >, < Com_{main,2,i} : i = 1, \cdots, n >$, orProofGen, eqProofGen1, and eqProofGen2 to the Producer.

Remark. Note that both the prover (the user) and verifier (the Producer and the organizations) can compute the following:

- computes $Com_{main,1} = \sum_1^n 2^{i-1} \cdot Com_{main,1,i} = x \cdot G_1 + s \cdot H_1$.
- computes $Com_{main,2} = \sum_1^n 2^{i-1} \cdot Com_{main,2,i} = x \cdot G_2 + t \cdot H_2$.

(i) signs as

$$Signature_{uid} = Sign_{tsk_{ed25519}}(hash(userdata))$$

where $userdata = (\ (CT_{seed}, CT_{seed,key}, epubkey_{seed}), hash_u, hash_{uid}, h_{ips}, hash_{uprivateshare}, tpk_{jubjub}, tpk_{ed25519}, ZKSNARK_{uips}, Signature_{ips}, \mathsf{CTUser}_{sq}, < B_j : j = 1, \cdots, n > , < CT_{privateshare,i} : m, \cdots, n >, < Com_{main,1,i} : i = 1, \cdots, n >, < Com_{main,2,i} : i = 1, \cdots, n >$, orProofGen, eqProofGen1, eqProofGen2).

(j) sends

$$Signature_{uid}, userdata$$

to the Producer (the backend of the Sovereign system).

8. The Producer

(a) calculates $Com_{main,1}$ and $Com_{main,2}$.

(b) verifies orProofGen, eqProofGen1, and eqProofGen2.

(c) checks if $hash_{uid} \stackrel{?}{=} hash_{ips}$.

(d) validates $Signature_{uid}$ through $tpk_{ed25519}$ and $hash_{uid}$.

(e) verifies $ZKSNARK_{uips}$ (which makes sure that the identifying information just shared by the user is the same as the one shared by the Identity Proxy Server).

(f) forwards $(Signature_{uid}, tpk_{ed25519}, hash_{uid})$ to the Identity Proxy Server.

9. The Identity Proxy Server

(a) checks if $hash_{uid} \stackrel{?}{=} hash_{ips}$.

(b) sends $Signature_{ips}, hash_{ips}$ to the Producer.

10. The Producer

(a) verifies $Signature_{ips}$ through pk_{ips},

(b) calculates $Sign_{sk_p}(hash(usertotaldata_u))$ where $usertotaldata_u = (userdata, Signature_{uid})$.

(c) sends $(Sign_{sk_p}(hash(usertotaldata_u)), usertotaldata_u)$ to Organization$_i$ $\forall\, i = m, \cdots, n$.

11. Let's say the set $Q = (Organization_{j_1}, \cdots, Organization_{j_\ell})$ where $j_i \in \{1, \cdots, n\}$ is going to participate in the following calculations. For $k = j_1$ until j_ℓ, Organization$_k$

(a) calculates $Com_{main,1}$ and $Com_{main,2}$.

(b) verifies orProofGen, eqProofGen1, and eqProofGen2.

(c) verifies $Sign_{sk_p}(usertotaldata_u)$ through pk_p.

(d) verifies $Signature_{ips}$ through pk_{ips}.

(e) validates $Signature_{uid}$ through $tpk_{ed25519}$ and $hash_{uid}$.

(f) verifies $ZKSNARK_{uips}$.

(g) decrypts $CT_{privateshare,k}$ and obtains $(Sign_{tsk_{ed25519}}(hash(tsk^{\mathcal{O},k}_{jubjub})), tsk^{\mathcal{O},k}_{jubjub})$.

(h) verifies $Sign_{tsk_{ed25519}}(hash(tsk^{\mathcal{O},k}_{jubjub}))$ through tpk_{jubjub}.

(i) checks if $tsk^{\mathcal{O},k}_{jubjub} \cdot G_2 \stackrel{?}{=} \sum_{z=0}^{t-1} k^z \cdot B_z$ to make sure that the calculations have been done correctly (see [Sch22, Chapter 6.2.1]).

(j) computes $h_k = keyshare_{org,i} \cdot hash_u$ and $h'_k = keyshare_{org,i} \cdot hash_{uprivateshare}$ [5] and creates a Schnorr proof $(\mathsf{SchnorrProofOrg1}_k)$ which proves the knowledge of $keyshare_{org,i} = \log_{G_2} pubshare_{org,i} = \log_{hash_u} h_k$ and also creates another Schnorr proof $(\mathsf{SchnorrProofOrg2}_k)$ $keyshare_{org,i} = \log_{G_2} pubshare_{org,i} = \log_{hash_{uprivateshare}} h'_k$.

(k) computes

$$Signature_{\mathcal{O}_k} = Sign_{sk_{\mathcal{O}_k}}(h_k, h'_k, hash(usertotaldata_u), \mathsf{SchnorrProofOrg1}_k, \mathsf{SchnorrProofOrg2}_k).$$

(l) sends $(Signature_{\mathcal{O}_k}, h_k, h'_k, hash(usertotaldata_u), \mathsf{SchnorrProofOrg1}_k, \mathsf{SchnorrProofOrg2}_k)$ to the Producer.

[5] This is done to have a threshold level of security on a system level.

12. The producer

    (a) verifies $Signature_{\mathcal{O}_k}$ for all $k \in Q$.

    (b) verifies $\mathsf{SchnorrProofOrg1}_k, \mathsf{SchnorrProofOrg2}_k$ for all $k \in Q$.

    (c) computes $hash_{u,O} = \sum_{k \in Q} \lambda_{Q,k} \cdot h_k$ and $hash'_{uprivateshare,O} = \sum_{k \in Q} \lambda_{Q,k} \cdot h'_k$ where $\lambda_{Q,k} = \sum_{j \in Q\ k} \frac{j}{j-k}$ denote the Lagrange coefficients as in the Shamir's scheme.

    (d) forwards the final computed value $(hash_{u,O}, hash'_{uprivateshare,O})$ to the user.

13. The user

    (a) calculates $hash_{u,selectivedisclosure} = Poseidon(idcredentials, tsk_{jubjub})$.

    (b) creates a $ZKSNARK_{selectivedisclosure}$ proof that $hash_{u,selectivedisclosure}$ is consistent with $hash_{uid}$. More concretely, given public values $hash_{u,selectivedisclosure}$, $hash_{uid}$, $tpk_{ed25519}$, tpk_{jubjub} and private values $idcredentials, tsk_{jubjub}, r_{ips}$

    - $hash_{u,selectivedisclosure} = Poseidon(idcredentials, tsk_{jubjub})$
    - $hash_{uid} = Poseidon(idcredentials, tpk_{ed25519}, r_{ips})$.
    - $tpk_{jubjub} = tsk_{jubjub} \cdot G_1$.

    (c) computes $identifier_u = r_u^{-1} \cdot hash_{u,O}$ and creates a Schnorr proof ($\mathsf{SchnorrProof1}$) which proves the knowledge of $r_u^{-1} = \log_{hash_{u,O}} identifier_u$.

    (d) computes $commit_{tsku} = s_u^{-1} \cdot hash'_{uprivateshare,O}$ and creates another Schnorr proof ($\mathsf{SchnorrProof2}$) which proves the knowledge of $s_u^{-1} = \log_{hash'_{uprivateshare,O}} commit_{tsku}$.

    - *Optional:* The user can add an additional proof that he/she indeed uses the same r_u and s_u. Note that all these values are already signed by the user but still we can add this proof to detect any possible corruption on the user side.

    (e) sends $hash_{u,selectivedisclosure}$, $ZKSNARK_{selectivedisclosure}$, ($identifier_u$, $commit_{tsku}$, $\mathsf{SchnorrProof1}$, $\mathsf{SchnorrProof2}$) to the Producer.

14. The Producer

    (a) verifies $ZKSNARK_{selectivedisclosure}$.

    (b) verifies $\mathsf{SchnorrProof1}$ and $\mathsf{SchnorrProof2}$.

    (c) calculates $Sign_{p,u,selectivedisclosure} = Sign_{sk_p}(hash_{u,selectivedisclosure})$

    (d) creates a Sovereign DID object called DID_u which contains

    - $Sovereign$
    - $identifier_u$,
    - $tpk_{ed25519}$,
    - tpk_{jubjub},
    - $commit_{tsku}$,
    - $hash_u$,

- $hash_{ips}$,
- $hash_{uprivateshare}$,
- $hash_{u,selectivedisclosure}$ [6]
- $(CT_{seed}, CT_{seed,key}, epubkey_{seed}), d_{seed,key}$
- $Sign_{sk_p}(hash(usertotaldata_u))$,
- $Sign_{p,u,selectivedisclosure}$
- $Signature_{uid}$,
- $Signature_{ips}$,
- $ZKSNARK_{uips}$,
- $ZKSNARK_{selectivedisclosure}$,
- $(hash_{u,O}, hash'_{uprivateshare,O})$
- $\mathsf{SchnorrProof1}, \mathsf{SchnorrProof2}$,
- $\mathsf{SchnorrProofOrg1}_k$, $\mathsf{SchnorrProofOrg2}_k$ for all $k \in Q$,
- orProofGen and eqProofGen1 and eqProofGen2,
- CTUser_{sq}, $d_{ct,key}$
- $< B_j = a_j \cdot G_2 : j = 1, \cdots, t-1 >$,
- $< CT_{privateshare,i} : i = m, \cdots, n >$,
- $< Signature_{\mathcal{O}_{j_i}}, h_{j_i}, h'_{j_i} : i = 1, \cdots, \ell >$.

(e) uploads DID_u to IPFS.

(f) calculates $CID_u = Hash(DID_u)$.

(g) adds $(tpk_{jubjub}, hash_{u,selectivedisclosure})$ to the Merkle tree. Let's say $newMerkleRoot$ is the updated Merkle root of the tree (see Chapter 8).

Remark. *Note that the internal calculations can either be stored on Algorand (if it is not expensive) or be stored on offchain.*

(h) creates and sends TxID_{merkle} which contains $newMerkleRoot$ to Algorand (to update the Merkle root of all the DID objects).

(i) creates and sends TxID_u which contains $(identifier_u, tpk_{ed25519}, CID_u)$

(j) sends TxID_{merkle}, TxID_u back to the user to make ensure that the Sovereign DID object has been created successfully.

15. The user verifies if DID_u has been indeed created and $CID_u \stackrel{?}{=} Hash(DID_u)$, and $identifier_u, tpk_{ed25519}, CID_u)$ is stored in the given transaction through TxID_u for $tpk_{ed25519}$. The user also checks if his/her $tpk_{ed25519}$ and $hash_{u,selectivedisclosure}$ have been added to the Merkle tree correctly by obtaining $newMerkleRoot$ from Algorand and checking with internal calculations (i.e., the authentication path to the root).

[6] this information will later be masked again to be used to prove selective disclosure in a private manner (i.e., it allows users to choose what and how much they share).

Chapter 11

A Sovereign System Use Case: Encryption of a Private Key of Other Chains

Members can only have one single Main Account associated with their real identities, while they may have as many Associated Accounts as they want. Anyone can publicly verify whether a given Associated Account is valid in the Sovereign System without disclosing the Main Account. The Associated Accounts are generated from the Main Accounts deterministically (like in BIP32 HD wallets). More concretely, the Associated Accounts are created through the hash of the private key of the Main Account (together with some incremented index). This creation process will help the users to retrieve all their Associated Accounts by only using the Main Account private key.

Members can import private keys generated outside the Sovereign System to protect them against loss by generating Associated Accounts specifically to manage the protection of these keys for retrieval through the Main Account. The method is not network dependent and can extend to any public or private permissionless or permissioned system. This use case provides a general layer of asset loss protection across all activity in the market for a Member, removing one of the heaviest burdens in self-custody, key management. The reduction in physical and mental costs and increase in ease of use is a crucial innovation initiative to help drive the Early Majority's intentions towards accepting self-custody as their primary model for the adoption of blockchain technology.

Assumption 1. *The user is assumed to know his/her private key* tsk_{jubjub} *(which is also equal to* $tsk_{ed25519}$*). Therefore, the user can trivially calculate* $tpk_{ed25519} = tsk_{jubjub} \cdot G_1$.

11.1 Login to Trustible

The user

1. enters his/her mnemonic words
2. calculates tsk_{jubjub}
3. performs the following

(a) computes $identifier_i = Poseidon(tsk_{jubjub}, i)$ where i is an incremented number.

(b) obtains the $(identifier_i, CID_i)$ from Algorand through $identifier_i$

(c) stops finding until a record $(identifier_i, CID_i)$ is not found on Algorand.

4. finds and downloads the corresponding Sovereign DID object DID_u from IPFS through CID_i.

11.2 The Protocol for Secure Storage Keys of Other Chains

1. The user

(a) logins to Trustible by running the protocol presented in Section 11.1.

(b) computes $sk_{associated,jubjub,protection,i} = Poseidon(tsk_{jubjub}, i, random)$ where $random$ is a fresh random number.

(c) defines $seed := sk_{associated,jubjub,protection,i}$.

(d) computes the following:

```
sk||y := sha512.Sum512(seed)
sk[0] &= 248
sk[31] &= 127
sk[31] |= 64

```

(e) calculates $x_i = sk \mod ord(ed25519)$ in the standard reference implementation.

(f) If x_i is larger than the order of the subgroup for JubJub (i.e., ℓ= 2736030358979909402780800718157159386076813972158567259200215660948447373041), then start over with a new random seed to calculate the scalar (i.e., go to step 1e.) [1]

(g) calculates $pk_{associated,clamped,ed25519,protection,i} = x_i \cdot G_1$.

(h) calculates $pk_{associated,clamped,jubjub,protection,i} = x_i \cdot G_2$.

(i) computes $pk_{associated,jubjub,protection,i} = sk_{associated,jubjub,protection,i} \cdot G_2$.

(j) obtains the most recent $newMerkleRoot$ from Algorand.

(k) generates a $ZKSNARK_{assoc,merkle}$ which proves that $pk_{associated,jubjub,protection,i}$ is linked to his/her main account tpk_{jubjub} (=$tsk_{jubjub} \cdot G_1$) without disclosing tpk_{jubjub}.

More concretely, given public values $newMerkleRoot$, $pk_{associated,jubjub,protection,i}$, $identifier_{coin}$, and private values i, tsk_{jubjub}, tpk_{jubjub}, $hash_{u,selectivedisclosure}$, $random$, $sk_{associated,jubjub,protection,i}$, $(h_{1,internal}, \cdots, h_{depth-1,internal})$ [2].

procedure $ZKSNARK_{assoc,merkle}((newMerkleRoot, pk_{associated,jubjub,protection,i}), (tsk_{jubjub}, psk_{jubjub}, sk_{associated,jubjub,protection,i}, (h_{1,internal}, \cdots, h_{depth-1,internal}))$:)

Check if $identifier_{coin} = Poseidon(tsk_{jubjub}, i)$

[1] We need this step to make sure that the public keys generated on both curves have the same private keys. See Section 5.8 for the reasoning of the calculations.

[2] These internal values can be taken from either onchain or offchain depending on the implementation.

Check if $sk_{associated,jubjub,protection,i} = Poseidon(tsk_{jubjub}, i, random)$
Check if $pk_{associated,jubjub,protection,i} \stackrel{?}{=} sk_{associated,jubjub,protection,i} \cdot G_2$.
Check if $tpk_{jubjub} \stackrel{?}{=} tsk_{jubjub} \cdot G_2$.
$h_0 \leftarrow (tpk_{jubjub}, hash_{u,selectivedisclosure})$
for $i = 1$ `till` $depth - 1$ **do**
    h_i `=` $hash(h_{i-1}||h_{i,internal})$
end for
Check if $h_i \stackrel{?}{=} newMerkleRoot$
end procedure

2. The user now

   (a) computes $Com_{associated,1,i,j} = \alpha_{i,j} \cdot G_1 + s_j \cdot H_1$ for some randomness s_j where $x_i = \alpha_{i,1}|| \cdots ||\alpha_{i,n}$ with $\alpha_{i,j} \in \{0, 1\}$, $j = 1, \cdots, n$.

   (b) computes $Com_{associated,2,i,j} = \alpha_{i,j} \cdot G_2 + t_j \cdot H_2$ for some randomness t_j.

   (c) creates a proof orProofGen as `orProofGen`$((G_1, H_1, G_2, H_2, Com_{associated,1,i,j}, Com_{associated,2,i,j}),\ (\alpha_{i,j}, s_j, t_j))$.

   (d) computes $Com_{associated,1,i} = \sum_1^n 2^{j-1} \cdot Com_{associated,1,i,j} = \alpha_i \cdot G_1 + s \cdot H_1$.

   (e) computes $Com_{associated,2,i} = \sum_1^n 2^{j-1} \cdot Com_{associated,2,i,j} = \alpha_i \cdot G_2 + t \cdot H_2$.

   (f) creates a proof eqProofGen1 as `eqProofGen`$((Com_{associated,1,i}, pk_{associated,clamped,ed25519,protection,i}),\ s)$.

   (g) creates a proof eqProofGen2 as `eqProofGen`$((Com_{associated,2,i}, pk_{associated,clamped,jubjub,protection,i}),\ t)$.

   (h) computes $Signature_{registration} = Sign_{sk_{associated,jubjub,protection,i}}(data)$ where $data = (pk_{associated,clamped,jubjub,protection,i}$, $pk_{associated,clamped,ed25519,protection,i}$, $pk_{associated,jubjub,protection,i}$, $random$, $identifier_{coin}$, $newMerkleRoot$, $Signature_{associatedaddress}$, $ZKSNARK_{assoc,merkle}$, $pk_{associated,clamped,ed25519,protection,i}$, orProofGen, eqProofGen1, eqProofGen2).

   (i) sends $Signature_{registration}$, $data$ to the Producer.

3. The user

   (a) associated account is displayed with $pk_{associated,clamped,ed25519,protection,i}$.

   (b) enters $CoinName$ and its private key of the selected network. Let's say sk_{btc} has been entered as the private key of his/her Bitcoin address (i.e., $CoinName$ = BTC).

Remark. *At this point, sk_{btc} can be verified (e.g., take the hash of the private key and verify with the checksum), the corresponding Bitcoin address can be calculated, the current balance can be read from the Bitcoin network and displayed on the user screen for further validation by the user.*

4. The user is now ready to encrypt $CoinName, sk_{bitcoin}$, and x_i, and provide a proof. Let M be be the message to be encrypted where $\ell = ||M||$. Here below we are using the Poseidon based encryption method presented by Khovratovich which [Kho19]. Note that the following encryption needs to be performed for $CoinName, sk_{bitcoin}$, and x_i separately. More concretely, the user

   (a) generates a random number r and a nonce $N < 2^{128}$.

   (b) generates an expanded shared secret key

   $$ss = r \cdot pk_{associated,jubjub,protection,i} = (ss_u[0], ss_u[1]) \in \mathbb{F}_q^2.$$

   (c) creates a Poseidon input state

   $$S = (0, ss_u[0], ss_u[1], N + \ell * 2^{128}) \in \mathbb{F}_q^4.$$

   (d) repeats for $0 \leq i \lceil \ell/3 \rceil$

      i. iterate Poseidon on

      $$S \leftarrow Poseidon(S)$$

      ii. absorbs three element of the message (in the last iteration the missing elements are set to 0):

      $$S[1] \leftarrow S[1] + M[3i]$$
      $$S[2] \leftarrow S[2] + M[3i+1]$$
      $$S[3] \leftarrow S[3] + M[3i+2].$$

      iii. releases three elements of ciphertext:

      $$CT[3i] \leftarrow S[1]$$
      $$CT[3i+1] \leftarrow S[2]$$
      $$CT[3i+2] \leftarrow S[3].$$

   (e) Iterates Poseidon on S last time:

   $$S \leftarrow P(S).$$

   (f) Release the last ciphertext element:

   $$CT.add(S[1]).$$

   (g) Output (CT_1, CT_2, CT_3).

   (h) creates a ZKSNARK$_{encryption}$ which proves that (CT_1, CT_2, CT_3) has been calculated correctly (i.e., encrypted with the public key $pk_{associated,jubjub,protection,i}$. More concretely, given the public values $random$, $pk_{associated,jubjub,protection,i}$, M, and private values r, N, show that

- $ss = r \cdot pk_{associated,jubjub,protection,i} = (ss_u[0], ss_u[1]) \in \mathbb{F}_q^2$.
- $S = (0, ss_u[0], ss_u[1], N + \ell * 2^{128}) \in \mathbb{F}_q^4$.
- $0 \leq i \lceil \ell/3 \rceil$
  i. calculate $S \leftarrow Poseidon(S)$
  ii. absorb three element of the message (in the last iteration the missing elements are set to 0):

$$S[1] \leftarrow S[1] + M[3i]$$
$$S[2] \leftarrow S[2] + M[3i+1]$$
$$S[3] \leftarrow S[3] + M[3i+2].$$

  iii. release three elements of ciphertext:

$$CT[3i] \leftarrow S[1]$$
$$CT[3i+1] \leftarrow S[2]$$
$$CT[3i+2] \leftarrow S[3].$$

- iterates Poseidon on S last time: $S \leftarrow Poseidon(S)$.
- releases the last ciphertext element: $CT_j.add(S[1])$ for $j = 1, 2, 3$.
- checks if $CT_j \stackrel{?}{=} CT'_j$ for $j = 1, 2, 3$.

(i) computes $Signature_{coin,userdata} = Sign_{sk_{associated,ed25519,i}}(pk_{associated,jubjub,protection,i}, random, ((CT_1, CT_2, CT_3), H_r, N, \ell), \text{ZKSNARK}_{encryption}, hash(Signature_{registration}))$ and sends it to the Producer.

5. The Producer

(a) obtains the current Merkle root $newMerkleRoot'$ from Algorand.

(b) verifies $Signature_{coin,userdata}$ through $pk_{associated,clamped,ed25519,protection,i}$.

(c) verifies $Signature_{registration}$ through $pk_{associated,jubjub,protection,i}$.

(d) verifies $ZKSNARK_{assoc,merkle}$ through $newMerkleRoot'$, $pk_{associated,jubjub,protection,i}$, $random$, $identifier_{coin}$, and $CoinName$.

(e) verifies orProofGen, eqProofGen1, eqProofGen2.

(f) submits the following object DID_{coin} to IPFS:

i. $Sovereign$
ii. $Trustible$
iii. $identifier_{coin}$
iv. $pk_{associated,clamped,ed25519,protection,i}$
v. $pk_{associated,clamped,jubjub,protection,i}$
vi. (CT_1, CT_2, CT_3)

vii. $(random, H_r, N, \ell)$,

viii. $ZKSNARK_{assoc,merkle}$

ix. $\text{ZKSNARK}_{encryption}$

(g) computes $CID_{coin} = SHA256(DID_{coin})$.

(h) submits a transaction which contains $(identifier_{coin}, CID_{coin})$ to the Algorand network for $pk_{associated,clamped,ed25519,protection,i}$. Let $TxID$ be the transaction ID of the transaction.

(i) sends $TxID'$ to the user to ensure that the encryption has been processed successfully.

6. The user verifies if DID_{coin} has been indeed created and $CID_{coin} \stackrel{?}{=} Hash(DID_{coin})$, and $(pk_{associated,jubjub,protection,i}, CID_{coin})$ is stored in the given transaction through $TxID_u$ for $pk_{associated,clamped,ed25519,protection,i}$.

11.3 Recovery of Private Keys of Other Chains

Users can always access the private key of their main account tsk_{jubjub} as long as they hold the identifying information and remember their security answers. We will divide the recovery into the following two cases:

- **The private key tsk_{jubjub} of the main account (which is also equal to $tsk_{ed25519}$) is known:** In this case, the user can already calculate tpk_{jubjub} and $tpk_{ed25519}$. Users can also re-calculate the associated public and private key pairs from tsk_{jubjub} as they were used to encrypt the private keys of other chains. Therefore, if tsk_{jubjub} is known, then private keys can be recovered easily.
- **The private key tsk_{jubjub} of the main account is unknown:** In this case, it is assumed that users have no prior knowledge at all (e.g., tsk_{jubjub} or even tpk_{jubjub} and the main DID object DID_u are all unknown). The goal of this phase is to first retrieve tsk_{jubjub} securely after the user is authenticated through the identifying information and answering the security questions, and then run the previous step again to recover the keys of other chains.

Let's assume that the user already holds tsk_{jubjub}. Next, in order to recover the private key of the other chains, the user

1. Repeats i until all $identifier_{coin}$s are not found:
   (a) compute $identifier_{coin} = Poseidon(tsk_{jubjub}, i)$ where i is an incremented number.
   (b) obtain the $(identifier_{coin}, CID_{coin})$ from Algorand through $identifier_{coin}$.
2. finds and downloads the corresponding Sovereign DID object DID_{coin} from IPFS through CID_{coin}.

3. gets $(random, H_r, N, \ell)$ from DID_{coin}.
4. computes $sk_{associated,jubjub,protection,i} = Poseidon(tsk_{jubjub}, i, random)$.
5. decrypts the ciphertext (CT_1, CT_2, CT_3) as follows:
   (a) Generate
   $$ss \leftarrow k \cdot pk_{associated,jubjub,protection,i}.$$
   (b) Create a Poseidon input state
   $$S = (0, ss_u[0], ss_u[1], N + \ell * 2^{128}) \in \mathbb{F}_q^4.$$
   (c) Repeat for $0 \leq i \lceil \ell/3 \rceil$
      i. Iterate Poseidon on S:
      $$S \leftarrow Poseidon(S).$$
      ii. Release three elements of message for $j = 1, 2, 3$:
      $$M[3i] \leftarrow S[1] + CT_j[3i]$$
      $$M[3i+1] \leftarrow S[2] + CT_j[3i+1]$$
      $$M[3i+2] \leftarrow S[3] + CT_j[3i+2].$$
      iii. Modify state:
      $$S[1] \leftarrow CT_j[3i]$$
      $$S[2] \leftarrow CT_j[3i+1]$$
      $$S[3] \leftarrow CT_j[3i+2].$$
      iv. If 3 does not divide ℓ then check that the last $3 - (\ell \mod 3)$ elements of M are 0. If not, reject the ciphertext.
   (d) Iterate Poseidon on S last time:
   $$S \leftarrow Poseidon(S).$$
   (e) Check last ciphertext element:
   $$CT_j.last = S[1].$$
   (f) Output $sk_{CoinName} := M$.

Chapter 12

Full Recovery of Main Accounts

In this section, we will assume that users do not hold the private keys of their main accounts.

12.1 tsk is unknown

1. The user
   (a) selects the recovery option that does not require the public or private key of the main account.
   (b) i. generates a new and fresh random number *seed*.
   ii. computes the following:

```
sk||y := sha512.Sum512(seed)
sk[0] &= 248
sk[31] &= 127
sk[31] |= 64
```

   iii. calculates $x = sk \mod ord(ed25519)$ in the standard reference implementation.
   iv. If x is larger than the order of the subgroup for JubJub (i.e., $\ell = 27360303589799094027808007181571593860768139721585672592002156609484473730 41$), then start over with a new random seed to calculate the scalar (i.e., go to step 1(b)i.)[1]

   (c) generates ephemeral public and private key pairs $(epubkey_{recover,ed25519}, eprivkey_{recover,ed25519})$ and $(epubkey_{recover,jubjub}, eprivkey_{recover,jubjub})$ such that
   - $eprivkey_{recover,ed25519} = eprivkey_{recover,jubjub} = x$,
   - $epubkey_{recover,ed25519} = x \cdot G_1$, and
   - $epubkey_{recover,jubjub} = x \cdot G_2$.

   Note that $(epubkey_{recover,ed25519}, eprivkey_{recover,ed25519})$ and $(epubkey_{recover,jubjub}, eprivkey_{recover,jubjub})$ are ephemeral key pairs which will be used to recover the private key of the main account.

[1] We need this step to make sure that the public keys generated on both curves have the same private keys. See Section 5.8 for the reasoning of the calculations.

(d) The user computes the proof of relation between JubJub and ed25519 addresses as follows:

i. computes $Com_{main,1,i} = eprivkey^{i}_{recover,jubjub} \cdot G_1 + s_i \cdot H_1$ for some randomness s_i where $eprivkey_{recover,jubjub} = eprivkey^{1}_{recover,jubjub} || \cdots || eprivkey^{n}_{recover,jubjub}$ with $eprivkey^{i}_{recover,jubjub} \in \{0, 1\}$.

ii. computes $Com_{main,2,i} = eprivkey^{i}_{recover,jubjub} \cdot G_2 + t_i \cdot H_2$ for some randomness t_i.

iii. creates a proof orProofGenRecover as `orProofGenRecover`$((G_1,\ H_1,\ G_2,\ H_2,\ Com_{main,1,i},\ Com_{main,2,i}),\ (eprivkey^{i}_{recover,jubjub}, s_i, t_i))$.

iv. computes $Com_{main,1} = \sum_1^n 2^{i-1} \cdot Com_{main,1,i} = eprivkey_{recover,jubjub} \cdot G_1 + s \cdot H_1$.

v. computes $Com_{main,2} = \sum_1^n 2^{i-1} \cdot Com_{main,2,i} = eprivkey_{recover,jubjub} \cdot G_2 + t \cdot H_2$.

vi. creates a proof eqProofGenRecover1 as `eqProofGenRecover`$((Com_{main,1}, epubkey_{recover,ed25519}),\ s)$.

vii. creates a proof eqProofGenRecover2 as `eqProofGenRecover`$((Com_{main,2}, epubkey_{recover,jubjub}),\ t)$.

Remark. Note that both the prover (the user) and verifier (the Producer and the organizations) can compute the following:

- computes $Com_{main,1} = \sum_1^n 2^{i-1} \cdot Com_{main,1,i} = x \cdot G_1 + s \cdot H_1$.
- computes $Com_{main,2} = \sum_1^n 2^{i-1} \cdot Com_{main,2,i} = x \cdot G_2 + t \cdot H_2$.

(e) submits $(name, surname, epubkey_{recover,ed25519})$ to the Identity Proxy Server.

2. The Identity Proxy Server

(a) calculates $Com_{main,1}$ and $Com_{main,2}$.

(b) validates $Signature_{uid}$ through $epubkey_{recover,ed25519}$ and $hash_{uid}$.

(c) creates an ApplicantID (through the Validator).

(d) generates and stores a random number r_{ips} for the user.

(e) sends the ApplicantID and r_{ips} to the user.

3. The user

(a) sends the identifying information (which includes the same credentials used during the registration) and the ApplicantID to the Validator.

(b) informs the Identity Proxy Server about the submission of the identifying information.

4. The Validator verifies the identifying information and confirms the uniqueness of the user in the network, and if so, sends the validation result to the Identity Proxy Server.

5. The Identity Proxy Server

(a) checks if the identifying information is validated through the Validator to confirm the uniqueness of the user in the network.

(b) obtains $idcredentials$ from the Validator.

(c) calculates

$$hash_{ips} = Poseidon(idcredentials, epubkey_{recover,ed25519}, r_{ips}).$$

where $idcredentials$ is exactly the one that was used during the registration.

(d) stores $hash_{ips}$ for the user.

(e) calculates

$$Signature_{ips} = Sign_{sk_{ips}}(hash_{ips})$$

(f) calculates the encryption of $(idcredentials, Signature_{ips})$ using $epubkey_{recover,ed25519}$ outputting $Ciphertext_{ips}$.

(g) sends $Ciphertext_{ips}$ to the user.

6. The user

(a) decrypts $Ciphertext_{ips}$ using $eprivkey_{recover,ed25519}$ and obtains $(idcredentials, Signature_{ips})$.

(b) validates the credential attributes $idcredentials$ (which is displayed as read only).

(c) verifies $Signature_{ips}$ through $pk_{ips}, r_{ips}, epubkey_{recover,ed25519}$.

(d) answers $securityquestions_{authrecovery}$ as $securityanswers_{authrecovery}$.

(e) generates a fresh and random r_u and calculates

$$hash_u = r_u \cdot Poseidon(idcredentials, securityanswers_{authrecovery})$$

(f) calculates the hash value $hash_{ips} = Poseidon(idcredentials, epubkey_{recover,ed25519}, r_{ips})$ (since r_{ips} is already known).

(g) creates $ZKSNARK_{uips,recover}$ such that $hash_u, hash_{ips}$ are consistent (i.e., both hashes contain the same input $idcredentials$). More concretely, Given public values $hash_u$, $hash_{ips}$, $epubkey_{recover,jubjub}$, r_{ips} and private values r_u, $idcredentials$, $eprivkey_{recover,jubjub}$, and $securityanswers_{authrecovery}$, prove that

- $hash_u = r_u \cdot Poseidon(idcredentials, securityanswers_{authrecovery})$
- $hash_{ips} = Poseidon(idcredentials, epubkey_{recover,ed25519}, r_{ips})$.
- $epubkey_{recover,jubjub} = eprivkey_{recover,jubjub} \cdot G_2$.

(h) signs as

$$Signature_u = Sign_{eprivkey_{recover,ed25519}}(hash(userdata))$$

where $userdata = (ZKSNARK_{uips,recover}$, $hash_u$, $hash_{ips}$, $epubkey_{recover,jubjub}$, $epubkey_{recover,ed25519}$, $Signature_{ips}$, $< Com_{main,1,i} : i = 1, \cdots, n >$, $< Com_{main,2,i} : i = 1, \cdots, n >$, orProofGenRecover, eqProofGenRecover1, eqProofGenRecover2).

(i) sends

$$(userdata, Signature_u)$$

to the Producer.

7. The Producer

(a) verifies $Signature_u$ through $epubkey_{recover,ed25519}$,

(b) verifies $Signature_{ips}$ through pk_{ips},

(c) verifies $ZKSNARK_{uips,recover}$ (which makes sure that the shared credential data by the user is the same as the one by the Identity Proxy Server).

(d) calculates $Com_{main,1}$ and $Com_{main,2}$.

(e) verifies orProofGenRecover, eqProofGenRecover1, and eqProofGenRecover2.

(f) calculates $usertotaldata_u = (userdata,\ Signature_u)$,

(g) calculates $Sign_p(hash(usertotaldata_u))$, and

(h) sends $Sign_p(hash(usertotaldata_u))$, $usertotaldata_u$ to Organization$_i$ $\forall\, i = m, \cdots, n$.

8. Let's say the set $Q = (Organization_{j_1}, \cdots, Organization_{j_\ell})$ where $j_i \in \{1, \cdots, n\}$ is going to participate in the following calculations. For $k = j_1$ until j_ℓ, Organization$_k$

(a) verifies $Sign_p(usertotaldata_u)$ through pk_p.

(b) verifies $Signature_u$ through $epubkey_{recover,jubjub}$.

(c) verifies $Signature_{ips}$ through pk_{ips}.

(d) calculates $Com_{main,1}$ and $Com_{main,2}$.

(e) verifies orProofGenRecover, eqProofGenRecover1, and eqProofGenRecover2.

(f) verifies $ZKSNARK_{uips,recover}$.

(g) computes $h_k = keyshare_{org,i} \cdot hash_u$ and creates a Schnorr proof ($\mathsf{SchnorrProofOrg1}_k$) which proves the knowledge of $keyshare_{org,i} = \log_{G_2} pubshare_{org,i} = \log_{hash_u} h_k$.

(h) computes

$$Signature_{\mathcal{O}_k} = Sign_{sk_{\mathcal{O}_k}}(h_k, hash(usertotaldata_u), \mathsf{SchnorrProofOrg1}_k).$$

(i) sends $(Signature_{\mathcal{O}_k}, h_k, hash(usertotaldata_u), \mathsf{SchnorrProofOrg1}_k)$ to the Producer.

9. The producer

(a) verifies $Signature_{\mathcal{O}_k}$ for all $k = j_1, \cdots, j_\ell$.

(b) verifies $\mathsf{SchnorrProofOrg1}_k$ for all $k = j_1, \cdots, j_\ell$.

(c) computes $hash_{u,O} = \sum_{k \in Q} \lambda_{Q,k} \cdot h_k$ where $\lambda_{Q,k} = \sum_{j \in Q\ k} \frac{j}{j-k}$ denote the Lagrange coefficients as in the Shamir's scheme.

(d) forwards the final computed value $hash_{u,O}$ to the user.

10. The user

    (a) computes $identifier_u = r_u^{-1} \cdot hash_{u,O}$ and creates a Schnorr proof (SchnorrProofRecover1) which proves the knowledge of $r_u^{-1} = \log_{hash_{u,O}} identifier_u$.

    (b) obtains CID_u from the Algorand network through $identifier_u$.

    (c) obtains DID_u from the IPFS through CID_u.

    (d) sends $identifier_u$ and SchnorrProofRecover1 to the Producer.

11. The Producer sends the final value $identifier_u$, SchnorrProofRecover1 as well as $Signature_{\mathcal{O}_i}$, h_i, $\mathsf{SchnorrProofOrg1}_k$ $\forall i \in Q$ to the organizations to decrypt the security questions.

12. The organizations are now ready to verify the calculations, and if successfully passed, calculate their partial private keys and share with the user so that the user can learn the security questions. Let's say the set $Q = (Organization_{j_1}, \cdots, Organization_{j_\ell})$ where $j_i \in \{1, \cdots, n\}$ is going to participate in the following calculations. For $k = j_1$ until j_ℓ, Organization_k

    (a) verifies $Signature_{\mathcal{O}_k}$ for all $k = j_1, \cdots, j_\ell$.

    (b) verifies $\mathsf{SchnorrProofOrg1}_k$ for all $k = j_1, \cdots, j_\ell$.

    (c) verifies SchnorrProofRecover1.

    (d) obtains CID_u from the Algorand network through $identifier_u$.

    (e) obtains DID_u from the IPFS through CID_u.

    (f) obtains $epubkey_{sq}, CT_{sq}$, and $salt_{secanswers}$ from the DID_u.

    (g) calculates the shared key as

    $$ss_{org,i} = keyshare_{org,i} \cdot epubkey_{sq}$$

    with $keyshare_{org,i} \cdot epubkey_{sq} = eprivkey_{sq} \cdot pubshare_{org,i}$.

    (h) calculates $CT_k = Enc_{epubkey_{recover,jubjub}}(ss_{org,i})$.

    (i) sends CT_k to the Producer.

13. The Producer forwards $< CT_k : k = j_1, \cdots, j_\ell >$ to the user.

14. The user

    (a) decrypts CT_k using $eprivkey_{recover,jubjub}$ and obtains $ss_{org,k}$.

    (b) applies the threshold decryption process (as described in Section 5.4.4) by recombining the partial decryptions $(ss_m, \cdots, ss_n)$ as $ss_{preimage}$.

    (c) calculates $ss = SHA512(ss_{preimage}.x)$ where $ss_{preimage}.x$ is the x coordinate of $ss_{preimage}$.

    (d) obtains ss_{enc} and ss_{mac} where $ss = ss_{enc}||ss_{mac}$

(e) calculates the decryption as

$$K = AESDec_{ss_{enc}}(CT_{key})$$

and checks if

$$d_{key} \stackrel{?}{=} MAC_{ss_{mac}}(CT_{key})$$

where $ss_{enc}||ss_{mac} = ss$.

(f) calculates $securityquestions = AESDec_K(CT_{sq})$.

(g) answers the security questions. Let's say the answers are denoted as $securityanswers$ [2].

(h) generates a fresh and new random value s_u and calculates

$$hash_{uprivateshare} = s_u \cdot Poseidon(salt_{secanswers}, securityanswers).$$

(i) sends $(hash_{uprivateshare}, Sign_{eprivkey_{recover,jubjub}}(hash_{uprivateshare}))$ to the Producer.

15. The Producer

(a) verifies $Sign_{eprivkey_{recover,jubjub}}(hash_{uprivateshare})$ through $epubkey_{recover,jubjub}$

(b) sends $(hash_{uprivateshare}, Sign_{eprivkey_{recover,jubjub}}(hash_{uprivateshare}))$ to Organization$_i$ $\forall$ $i \in Q$.

16. For $k = j_1$ until j_ℓ, Organization$_k$

(a) verifies $Sign_{eprivkey_{recover,jubjub}}(hash_{uprivateshare})$ through $epubkey_{recover,jubjub}$

(b) computes $h'_k = keyshare_{org,i} \cdot hash_{uprivateshare}$ and creates a Schnorr proof $(\mathsf{SchnorrProofOrg2}_k)$ $keyshare_{org,i} = \log_{G_2} pubshare_{org,i} = \log_{hash_{uprivateshare}} h'_k$.

(c) computes

$$Signature'_{\mathcal{O}_k} = Sign_{sk_{\mathcal{O}_k}}(h'_k, hash(usertotaldata_u), \mathsf{SchnorrProofOrg2}_k).$$

(d) sends $(Signature'_{\mathcal{O}_k}, h'_k, hash(usertotaldata_u), \mathsf{SchnorrProofOrg2}_k)$ to the Producer.

17. The Producer

(a) verifies $\mathsf{SchnorrProofOrg2}_k$ for all $k \in Q$.

(b) verifies $Signature'_{\mathcal{O}_k}$ for all $k \in Q$.

(c) computes $hash'_{uprivateshare,O} = \sum_{k \in Q} \lambda_{Q,k} \cdot h'_k$ where $\lambda_{Q,k} = \sum_{j \in Q\ k} \frac{j}{j-k}$ denote the Lagrange coefficients as in the Shamir's scheme.

(d) forwards the final computed value $hash'_{uprivateshare,O}$ to the user.

18. The user

[2] Note that these answers given must be the same as the ones created during the registration

(a) computes $commit_{tsku} = s_u^{-1} \cdot hash'_{uprivateshare,\mathcal{O}}$ and creates a Schnorr proof (SchnorrProofRecover2) which proves the knowledge of $s_u^{-1} = \log_{h'_{uprivateshare,\mathcal{O}}}(commit_{tsku})$.

(b) sends $commit_{tsku}$ and SchnorrProofRecover2 to the Producer.

19. The Producer forwards $commit_{tsku}$, h'_k, SchnorrProofRecover2, $Signature'_{\mathcal{O}_i}$ to Organization$_i$ $\forall i \in Q$.

20. For $k = j_1$ to j_m, Organization$_k$

(a) retrieves $tsk_{jubjub}^{\mathcal{O},k}$ by decrypting the respective $CT_{privateshare,k}$ included in the fetched DID document using $sk_{\mathcal{O}_k}$ where $sk_{\mathcal{O}_k}$ denotes the private key of Organization$_k$.

(b) verifies SchnorrProofRecover2

(c) obtains $commit'_{tsku}$ from DID_u

(d) verifies $Signature'_{\mathcal{O}_k}$ for all $k = j_1, \cdots, j_m$.

(e) checks if $commit_{tsku} \stackrel{?}{=} commit'_{tsku}$

(f) encrypts their partial private keys through $epubkey_{recover,jubjub}$ and sends back to the user. More concretely, each Organization$_i$ computes

$$C_{share,\mathcal{O}_i} = Enc_{epubkey_{recover,jubjub}}(tsk_{\mathcal{O}}^i)^3$$

and sends $C_{share,\mathcal{O}_i}$ to the Producer.

21. The Producer sends $C_{share,\mathcal{O}_k}$ back to the user for all $k \in Q$.

22. For $k = j_1$ to j_m, the user

(a) calculates

$$tsk_{\mathcal{O}}^k = Dec_{eprivkey_{recover,jubjub}}(C_{share,\mathcal{O}_k})$$

(b) finally reconstructs the Trustible Private Key

$$tsk_{jubjub} = Reconstruct(tsk_{jubjub}^{u,1}, \cdots, tsk_{jubjub,secans}^{u,m-1}, tsk_{\mathcal{O}}^m, \cdots, tsk_{\mathcal{O}}^n)^4$$

through the Lagrange interpolation where

i. calculate $tsk_{jubjub,secans}^u = PBKDF2(salt_{secanswers}, securityanswers)$

ii. generate $tsk_{jubjub}^{u,i} = PBKDF2(tsk_{jubjub,secans}^u, i)$ for $i = 1, \cdots, m-1$

23. Once tsk (which is the constant term of the polynomial $p(x)$) is obtained, the user decrypts $(CT_{seed}, CT_{seed,key}, d_{seed,key})$ using $tsk_{jubjub}^{u,i}$ and $epubkey_{seed}$.

The user executes the protocol presented in Section **??** to obtain the private keys of other chains.

[3]Note that this can be sent through the Producer since it is already encrypted).

[4]Any t values will be sufficient to compute the polynomial $p(x)$

Chapter 13

Integration of Trustible with PolygonID

In this chapter, we present a new protocol that lets Trustible users create their Polygon ID through their Associated Accounts. This allows Polygon ID to connect to the real identity information, such as passports and id cards, via Trustible.

13.1 Creating a Claim for Polygon ID through Associated Accounts

Assumption 2. *In the Polygon integration, Producer is assumed to be the Issuer.*

1. User
   (a) clicks on "Polygon Integration" in order to pull the actual data from IPFS using the main account (if not exists locally).
   (b) creates an associated account from the main account as:

$$sk_{associated,jubjub,i} = Poseidon(tsk_{jubjub}, \text{“}PolygonID'', i, random).$$

   (c) $identifier_{polygonid} = Poseidon(tsk_{jubjub}, i)$
   (d) calculates $claim$ and $hash_{polygon}$ based on $hash_{u,selectivedisclosure}$ where

$$hash_{u,selectivedisclosure} = Poseidon(idcredentials, tsk_{jubjub}).$$

   (e) calculates $hash'_{u,selectivedisclosure} = Poseidon(randomnumber, hash_{u,selectivedisclosure})$.
   (f) creates ZKSNARK$_1$ and ZKSNARK$_2$ proving that the associated account has been calculated correctly and $hash_{u,selectivedisclosure}$ is consistent with $hash_{polygon}$ (e.g., birthday and type of identity document could be public input for the proof).
   (g) submits $claim$, $hash'_{u,selectivedisclosure}$, $hash_{polygon}$, ZKSNARK$_1$, ZKSNARK$_2$ to Producer.

2. Producer

(a) verifies ZKSNARK$_1$ and ZKSNARK$_2$.

(b) issues *claim* as in the Polygon ID flow (and the flow starting from here is exactly the same as in Polygon ID).

(c) submits the following object $DID_{polygonid}$ to IPFS:

i. $Sovereign$
ii. $Trustible$
iii. $ChainID$
iv. $identifier_{polygonid}$
v. $pk_{associated,jubjub,i}$
vi. $hash'_{u,selectivedisclosure}$
vii. $claim$
viii. $hash_{polygon}$
ix. $random$
x. $ZKSNARK_1$
xi. $ZKSNARK_2$
xii. Merkle root of the DID Account Tree

(d) computes $CID_{coin} = SHA256(DID_{polygonid})$.

(e) submits a transaction which contains $(identifier_{polygonid}, CID_{polygonid})$ to the Algorand network. Let $TxID$ be the transaction ID of the transaction.

(f) sends $TxID$ to the user to ensure that the encryption has been processed successfully.

3. The user verifies the calculations.

13.1.1 ZKSNARK$_1$

1. Calculate $hash'_{u,selectivedisclosure} = Poseidon(randomnumber, hash_{u,selectivedisclosure})$ where $randomnumber$ is a freshly generated random number.

2. Given public values $hash'_{u,selectivedisclosure}$, $pk_{associated,jubjub,i}$, $hash_{polygon}$, $random$, $identifier_{polygonid}$, Merkle root of the DID Account tree and private values tpk_{jubjub}, tsk_{jubjub}, $sk_{associated,jubjub,i}$, $idcredentials$, $randomnumber$, $hash_{u,selectivedisclosure}$ prove that

(a) $tpk_{jubjub} \stackrel{?}{=} tsk_{jubjub} \cdot G_2$.

(b) $sk_{associated,jubjub,i} = Poseidon(tsk_{jubjub}, \text{“}PolygonID\text{”}, i, random)$.

(c) $pk_{associated,jubjub,i} \stackrel{?}{=} sk_{associated,jubjub,i,random} \cdot G_2$.

(d) $identifier_{polygonid} = Poseidon(tsk_{jubjub}, i)$

(e) $hash'_{u,selectivedisclosure} = Poseidon(randomnumber, Poseidon(idcredentials, tsk_{jubjub}))$.

(f) $(tpk_{jubjub}, hash_{u,selectivedisclosure})$ has an authentication path to Merkle root.

13.1.2 ZKSNARK$_2$

Given public values $hash'_{u,selectivedisclosure}$, $hash_{polygon}$, Merkle root of the DID Account Tree and private values $randomnumber$, tsk_{jubjub}, $idcredentials$ prove that

1. $hash'_{u,selectivedisclosure} = Poseidon(randomnumber, Poseidon(idcredentials, tsk_{jubjub}))$.
2. $hash'_{u,selectivedisclosure}$ and $hash_{polygon}$ are consistent. More concretely, the preimage of $hash_{u,selectivedisclosure} = Poseidon(idcredentials, tsk_{jubjub})$ contains the preimage of $hash_{polygon}$ without disclosing $hash_{u,selectivedisclosure}$.

Chapter 14

Restricting Associated Accounts for Different Application Domains

In this chapter, we will limit users to have only one account per application domain.

14.1 A New Merkle Tree to Ensure One Associated Account for Each Main Account

We want to link one associated account to each main account for a specific application domain, however we also want to keep the main account private. To achieve this, we will use a new Merkle tree that contains both the main account and the associated account, but hides the main account from others. More concretely, if a user creates a new associated account $pk_{associated,jubjub,chainid,appdomain,i}$ for an Application Domain *AppDomain* on a chain (with $ChainID$), then $H(tsk_{jubjub}, ChainID, AppDomain)$ will be generated and will be added to $MerkleTree_{appdomain}$ unless it has been listed before. By doing this, we can ensure that every main account has only one associated account per application domain.

The Merkle tree of Application Domains in Figure 14.1 ensures that one single main account cannot have more than one associated account for a given application domain.

14.2 The Protocol

1. The user U
   (a) obtains the $(identifier_u, CID_u)$ from Algorand through $tpk_{ed25519}$.
   (b) finds and downloads the corresponding DID object DID_u from IPFS through CID_u.
   (c) selects an application domain *AppDomain* on a chain $ChainID$ from the dropdown list of coins. Let's say the user selects *AppDomain* = "PolygonID" on Polygon.
   (d) computes $identifier_{appdomain} = Poseidon(tsk_{jubjub}, i)$ where i is an incremented number.

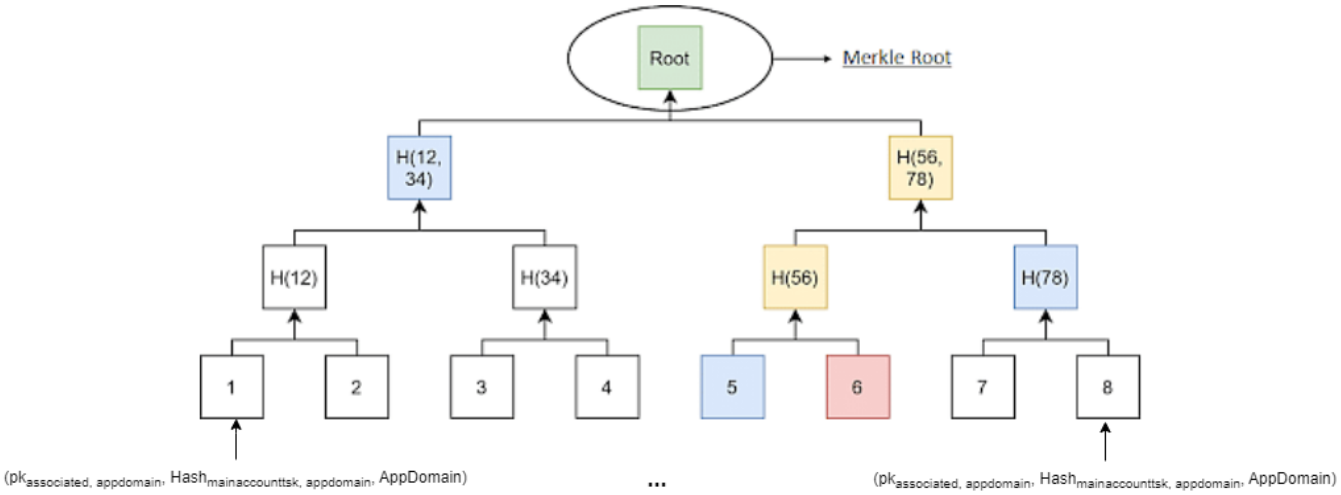

Figure 14.1: $MerkleTree_{appdomain}$: The list of associated accounts and application domains for a given main account where $pk_{associated,chainid,appdomain}$ is the public key of an associated account, $Hash_{mainaccounttsk,chainid,appdomain}$ is the hash of the private key of the corresponding main account and the application domain $AppDomain$ on a chain with $ChainID$.

Remark. Note that $identifier_{appdomain}$ should not be used before. To verify the uniqueness, the user first checks whether it exists on the Algorand network.

(e) computes

$$sk_{associated,jubjub,chainid,appdomain,i} = Poseidon(tsk_{jubjub}, ChainID, AppDomain, i, random)$$

where $random$ is a fresh random number.

(f) defines $seed := sk_{associated,jubjub,chainid,appdomain,i}$.

(g) computes the following:

```
sk||y := sha512.Sum512(seed)
sk[0] &= 248
sk[31] &= 127
sk[31] |= 64

```

(h) calculates $\alpha_i = sk \mod ord(ed25519)$ as in the standard reference implementation.

(i) If α_i is larger than the order of the subgroup for JubJub (i.e., $\ell = 2736030358979909402780800718157159386076813972158567259200215660948447373041$), then start over with a new random seed to calculate the scalar (i.e., go to step 1e.) [1]

(j) calculates $pk_{associated,clamped,ed25519,chainid,appdomain,i} = \alpha_i \cdot G_1$.

(k) calculates $pk_{associated,clamped,jubjub,chainid,appdomain,i} = \alpha_i \cdot G_2$.

[1] We need this step to make sure that the public keys generated on both curves have the same private keys. See Section 5.8 for the reasoning of the calculations.

(l) computes $pk_{associated,jubjub,chainid,appdomain,i} = sk_{associated,jubjub,chainid,appdomain,i} \cdot G_2$.

(m) obtains the most recent Merkle Root $oldMerkleRoot$ from Algorand.

(n) calculates $Hash_{tsk,chainid,appdomain} = Poseidon(tsk_{jubjub}, ChainID, AppDomain)$. [2]

(o) generates a $ZKSNARK_{assoc,merkle,chainid,appdomain}$ which proves that $pk_{associated,jubjub,chainid,appdomain,i}$ is linked to his/her main account tpk_{jubjub} ($=tsk_{jubjub} \cdot G_1$) without disclosing tpk_{jubjub} and the main account is linked to $MerkleTree_{appdomain}$. More concretely, given public values $newMerkleRoot$, $Hash_{tsk,chainid,appdomain}$, $pk_{associated,jubjub,chainid,appdomain,i}$, $identifier_{appdomain}$, $ChainID$, $AppDomain$ and private values i, tsk_{jubjub}, tpk_{jubjub}, $hash_{u,selectivedisclosure}$, $random$, $sk_{associated,jubjub,appdomain,i}$, $(h_{1,internal}, \cdots, h_{depth-1,internal})$ [3].

procedure $ZKSNARK_{assoc,merkle,chainid,appdomain}((newMerkleRoot, pk_{associated,jubjub,chainid,appdomain,i}), (tsk_{jubjub}, psk_{jubjub}, sk_{associated,jubjub,chainid,appdomain,i}, (h_{1,internal}, \cdots, h_{depth-1,internal})) :)$

1) Check if $identifier_{appdomain} = Poseidon(tsk_{jubjub}, i)$

2) Check if

$$sk_{associated,jubjub,chainid,appdomain,i} = Poseidon(tsk_{jubjub}, ChainID, AppDomain, i, random)$$

3) Check if $pk_{associated,jubjub,chainid,appdomain,i} \stackrel{?}{=} sk_{associated,jubjub,chainid,appdomain,i} \cdot G_2$.

Check if $tpk_{jubjub} \stackrel{?}{=} tsk_{jubjub} \cdot G_2$.

4) $Hash_{tsk,ChainID,AppDomain} \stackrel{?}{=} Poseidon(tsk_{jubjub}, ChainID, AppDomain)$.

5) $h_0 \leftarrow (tpk_{jubjub}, hash_{u,selectivedisclosure})$

for $i = 1$ `till` $depth - 1$ **do**

h_i `=` $hash(h_{i-1}||h_{i,internal})$

end for

6) Check if $h_i \stackrel{?}{=} newMerkleRoot$

end procedure

(p) computes $Com_{associated,1,i,j} = \alpha_{i,j} \cdot G_1 + s_j \cdot H_1$ for some randomness s_j where $\alpha_i = \alpha_{i,1}|| \cdots ||\alpha_{i,n}$ with $\alpha_{i,j} \in \{0,1\}$, $j = 1, \cdots, n$.

(q) computes $Com_{associated,2,i,j} = \alpha_{i,j} \cdot G_2 + t_j \cdot H_2$ for some randomness t_j.

(r) creates a proof orProofGen as `orProofGen`$((G_1, H_1, G_2, H_2, Com_{associated,1,i,j}, Com_{associated,2,i,j}),\ (\alpha_{i,j}, s_j, t_j))$.

(s) computes $Com_{associated,1,i} = \sum_1^n 2^{j-1} \cdot Com_{associated,1,i,j} = \alpha_i \cdot G_1 + s \cdot H_1$.

(t) computes $Com_{associated,2,i} = \sum_1^n 2^{j-1} \cdot Com_{associated,2,i,j} = \alpha_i \cdot G_2 + t \cdot H_2$.

(u) creates a proof eqProofGen1 as `eqProofGen`$((Com_{associated,1,i}, pk_{associated,clamped,ed25519,chainid,appdomain,i}),\ s)$.

(v) creates a proof eqProofGen2 as `eqProofGen`$((Com_{associated,2,i}, pk_{associated,clamped,jubjub,chainid,appdomain,i}),\ t)$.

[2] This ensures the uniqueness of an associated accoount per main account.

[3] These internal values can be taken from either onchain or offchain depending on the implementation.

(w) computes $Signature_{registration} = Sign_{sk_{associated,jubjub,chainid,appdomain,i}}(data)$ where $data = (pk_{associated,clamped,jubjub,chainid,appdomain,i}, pk_{associated,clamped,ed25519,chainid,appdomain,i}, pk_{associated,jubjub,chainid,appdomain,i}, random, identifier_{appdomain}, ChainID, AppDomain, newMerkleRoot, Hash_{tsk,chainid,appdomain}, Signature_{associatedaddress}, ZKSNARK_{assoc,merkle,chainid,appdomain}, pk_{associated,clamped,ed25519,chainid,appdomain,i}$, orProofGen, eqProofGen1, eqProofGen2).

(x) sends $Signature_{registration}$, $data$ to the Producer.

2. The Producer

(a) obtains the current Merkle root $newMerkleRoot'$ from Algorand.

(b) verifies $Signature_{registration}$ through $pk_{associated,jubjub,chainid,appdomain,i}$.

(c) verifies $ZKSNARK_{assoc,merkle,chainid,appdomain}$ through $newMerkleRoot'$, $Hash_{tsk,chainid,appdomain}$, $pk_{associated,jubjub,chainid,appdomain,i}$, $random$, $identifier_{appdomain}$, $ChainID$ and $AppDomain$.

(d) checks if $Hash_{tsk,chainid,appdomain}$ exists in the Merkle tree of Application Domains which has the current root $MerkleRoot_{domain,appdomain}$. It aborts the protocol if it already exists (to prevent multiple associated accounts for a given main account.).

(e) verifies orProofGen, eqProofGen1, eqProofGen2.

(f) submits the following object $DID_{appdomain}$ to IPFS:

i. $Sovereign$
ii. $ChainID$
iii. $AppDomain$
iv. $newMerkleRoot$
v. $identifier_{appdomain}$
vi. $Hash_{tsk,chainid,appdomain}$
vii. $random$
viii. $pk_{associated,clamped,ed25519,chainid,appdomain,i}$
ix. $pk_{associated,clamped,jubjub,chainid,appdomain,i}$
x. $ZKSNARK_{assoc,merkle,chainid,appdomain}$

(g) The Producer adds $(pk_{associated,jubjub,chainid,appdomain,i}, Hash_{tsk,chainid,appdomain}, ChainID, AppDomain)$ to $MerkleTree_{appdomain}$ which is the Merkle root of the tree for all application domains. Let's say $NewMerkleRoot_{appdomain}$ is the new Merkle root of the tree $NewMerkleTree_{appdomain}$.

(h) computes $CID_{appdomain} = SHA256(DID_{appdomain})$.

(i) submits a transaction which contains ($identifier_{appdomain}$, $pk_{associated,jubjub,appdomain,i}$, $CID_{appdomain}$) to the Algorand network for $pk_{associated,clamped,ed25519,chainid,appdomain,i}$. Let $TxID$ be the transaction ID of the transaction.

(j) sends $\text{TxID}_{merkle,chainid,appdomain}$, TxID_u back to the user to make ensure that the DID object $DID_{appdomain}$ has been created successfully.

3. The user verifies if $DID_{appdomain}$ has been indeed created and

   $CID_{appdomain} \stackrel{?}{=} Hash(DID_{appdomain})$ and $(pk_{associated,jubjub,chainid,appdomain,i}, CID_{appdomain})$

   are stored in the given transaction through $TxID_u$ for $pk_{associated,clamped,ed25519,chainid,appdomain,i}$.

14.3 Revoking an Associated Account for an Application Domain

To revoke an associated account that is linked to an app domain, users need to use the corresponding private key. Note that we can always recover the associated account because it is derived from the main account, and the main account can always be restored as long as users remember the answers to their security questions.

1. The user calculates and sends

   $Sign_{sk_{associated,jubjub,chainid,appdomain,i}}(pk_{associated,jubjub,chainid,appdomain,i}, Hash_{tsk,chainid,appdomain}, ChainID,$

   to the Producer.

2. The Producer

   (a) finds a record using $pk_{associated,jubjub,chainid,appdomain,i}$ in $MerkleTree_{appdomain}$.

   (b) removes $(pk_{associated,jubjub,chainid,appdomain,i}, Hash_{tsk,chainid,appdomain}, ChainID, AppDomain)$ from $MerkleTree_{appdomain}$.

   (c) updates the new Merkle root of $MerkleTree_{appdomain}$ on the Algorand network.

   (d) stores

   $Sign_{sk_{associated,jubjub,chainid,appdomain,i}}(pk_{associated,jubjub,chainid,appdomain,i}, Hash_{tsk,chainid,appdomain}, Chain$

   on the Algorand network for public verifiability.

   (e) notifies the user about the removal.

Chapter 15

Deactivating Main Accounts

15.1 Case 1: If a user remembers his/her tsk

Assume that a user's main key has been compromised. In this case, the user needs to disable the current main account and re-creates a new one.

1. The user creates a signature using tsk and requests from the Producer to disable his/her corresponding main account.
2. The signature is sent to the Producer. Alternatively, it can also be submitted as a transaction (for public transparency).
3. The Producer obtains the signature and verifies its validity. If the verification is successful, then it disables the user. The disabling can be done in the following two ways:
   (a) Let's assume that $(tpk, 1)$ and $(tpk, 0)$ are used for enabled and disabled users, respectively. We store $(tpk, 1)$ in the Merkle tree instead of having only tpk. If a user requests to be disabled then $(tpk, 1)$ will be updated to $(tpk, 0)$, and the Merkle root will be updated accordingly.
   (b) We can also simply remove tpk from the leaves and update the Merkle root accordingly.

Remark. Once users are disabled, they will not be able to use any Vendible services since a valid proof cannot be generated any more.

Remark. Once users are disabled, they have to perform the registration process again to be able to create new tpk.

15.2 Case 2: If a user does not remember both tsk and the security answers

In this case, the removal cannot be processed. The reason is that the Sovereign system does not hold any relation between the identifying information and tpk, and therefore, the Sovereign system cannot verify whether the user is telling the truth.

Chapter 16

Potential Enhancements: Extending to Social Recovery

The Sovereign System can also easily be extended to support social recovery. Namely, the user can add his/her additional public keys of his friends or family members to be part of the system. The Producer will reduce the number of public keys from the number of the organizations and only share with the remaining number of the organizations. In this way, the system could be more flexible.

For example, during the registration, let's say a user Alice also shares the public key of her father Bob, her aunt Eve, and her close friend Charlie, and also asked to have (4,7) threshold security in the system. hence,

- sk_{alice} (her private key share through security questions)
- sk_{bob} (her father's private key)
- sk_{eve} (her aunt's private key)
- $sk_{charlie}$ (her close friend's private key)
- Organization_1, Organization_2, Organization_3,

Based on this construction, we can conclude the followings:

- The system can be robust up-to three participants' corruption (e.g., system failures or key loss).
- The Sovereign System remains intact (i.e., the Producer colluding with the Organizations maliciously cannot obtain the private key at all (including brute force attacks)) as three organizations is strictly less than the threshold. Namely, the organizations are required to collude with at least one party to apply brute force attacks.
- Alice does not need the Organizations to recover her keys. Namely, the keys can be recovered even if the Sovereign System is corrupted (or down) including the Organizations ability to operate.

Chapter 17

Securing the Sovereign System

The Sovereign System, having a separate elliptic curve that equates to private keys on all significant blockchains, coupled with the assurance that each participant is a unique Member, opens the possibility for a cross-chain identity network that operates as a trust layer for distributed ledgers.

To secure the network, we propose a cooperative network model which includes a digital asset (or token), dA. The token dA provides a measure of each Member's impact on the network, rewards Members for positive support, acts as a deterrent for Members who might otherwise abuse the system or other Members, and helps facilitate network governance.

17.1 Network Minimum Commitment

Each Member has a Main Account which contains the DID object that attests to their uniqueness on the network. This Main Account has one purpose outside of maintaining the DID; acting as the signatory in the construction of Associated Accounts. It is advisable to create an Associated Account for each connection made by the Member to help preserve the privacy of each Member. A connection is a channel formed between two Members to transfer data and assets between one another. These connections exist between contacts (friends and family), merchants, decentralized applications (dApps), and institutions. The first step in securing the network is to leverage the creation of Associated Accounts so that Members have a verifiable measure of their network activity. A small amount of dA (with the amount dependent upon the use case) can lock into each Associated Account upon creation. This dA is paid for by the Member or a Service Provider acting on behalf of the Member. This commitment stake remains in the Associated Account until a vesting period expires or that account is no longer in use. The network minimum commitment is the sum of a Member's dA held in Associated Accounts.

17.2 Cooperative Network

The Minimum Commitment is an incentive to reward behavior that benefits the long-term health of the network. We employ a stake-and-slash mechanism managed by a decentralized

autonomous organization (DAO) to help oversee the direct operation of the network. All Members have the right to privacy and the freedom to steward their data and assets without interference from outside parties. However, if a Member (whether that Member is an individual, business, Service Provider, Organization, Validator, the DAO Foundation or Producer) acts maliciously or counter to the viable operation of the network, they can lose their Minimum Commitment stake and, in some instances, the right to operate on the network. In some variations, the Member can also give up their right to privacy as a direct result of malicious actions. Conversely, members can gain rewards through multiple mechanisms, including the distribution of dA, for activities that support the operation of the network. Participation is the term used for supportive network activities.

17.3 Member Staking

Members may stake dA above the Minimum Commitment. A specific Associated Account holds the stake. Staking provides access to community management of the cooperative network, oversight of the community treasury, voting and administration for Foundation representatives, and direct network rewards.

17.4 Operational Rewards

Suppose the cooperative network generates value from the specific properties of the network (i.e., Service Providers find value in building products that leverage verifiable claims of uniqueness and secure data management). In that case, network Members can share in this value generation. A separate paper will outline specifics on reward distributions by the Producer to Members based on network activity and staking.

Chapter 18

Expansion of the Sovereign System

The Sovereign System outlined in the paper is an open framework for identity, data, privacy, and payment solutions. Additional research and publications will add to this body of work. An outline of upcoming and future expansion follows.

18.1 Identity Vault Questions Research

In a fully Sovereign System, the security and recovery of a Member's Main Account are predicated on the Member remembering the questions. A large body of research has examined various methods for secure fallback authentication. The authors have compiled much of this research and will apply these topics to early testing of Main Account generation. Questions must be memorable but not easily guessed. Testing different question modes, including text, visual, kinetic, persona, location-based, and specific recall prompts, will help ensure a robust system. We will explore further variations of account setup with fail-safe options for Members. Upon completion of testing, we will publish our results to guide best practices.

18.2 Selective Disclosure and User-Owned Distributed Data

One of the most powerful aspects of the Sovereign System is that through Associated Accounts, Service Providers, Institutions, and Organizations can query the Member without the Member revealing any personal information. Leveraging the Merkle Tree setup, a Member can generate an Associated Account for authentication, and outside parties can pose zero-knowledge-based questions to the Associated Accounts and receive answers from the Main Account without exposing the public address of the Main Account.

Service Providers can use the Associated Accounts and the verifiable uniqueness of Members to build new classes of applications. Some advantages of this authentication system include preventing bots from gaining access, restrictions to admittance based on age or geography, and ensuring that each user in an application can only create one account in a system.

All data produced in the Sovereign System is encrypted, stored in a distributed manner to ensure availability, and attributed to DIDs (both in the Main Account and Associated Accounts). The Member entirely owns this data. The Member can selectively share this data

with other Members or service providers. Service Providers can create custom data attributes for their applications and leverage the Sovereign System to store and manage user-owned data securely. In this scenario, a user will agree to share specific data attributes created or named by the Service Provider before gaining access to an application. If the Member no longer wishes to interact with the application, they can revoke the Service Providers' access to their data.

18.3 Automation of Associated Accounts

Returning to the VAM adoption model, applications utilizing the Sovereign System must ensure that the benefits of engaging with the technology outweigh any perceived sacrifices. The system must be enjoyable to use with little monetary and mental costs. Automation of Associated Accounts will help provide the most value for the effort. With all accounts recoverable through the Main Account, it is advisable to create user experiences that obfuscate the entire key generation and management process so that the Member can interact with the technology in ways they are already familiar [Lin98].

A paper outlining best account generation and management practices across multiple networks and applications will help guide this area.

18.4 Specific Use Cases

During the creation of the Sovereign System, multiple use cases were proposed, outlined, and designed to ensure the robustness of the proposed system. Numerous papers will outline each use case and its potential impact on the market. Use cases explored include:

- anonymous polling and voting
- authentication techniques
- anonymous KYC and ongoing AML
- user-disclosed KYC data and ongoing AML
- user-owned decentralized data storage
- single-sign-on systems
- privacy-preserving markets
- confidential transactions
- compliant-confidential transactions

18.5 Variations of the Sovereign System

The Sovereign System is the base layer of an open solution for identity, data protection, and payments. The base layer design is structured to guide the Early Majority, Late Majority, and Laggards (consisting of individuals, businesses, and institutions) to accept self-custody of their data and assets as the de facto mode of transaction across the internet. Organizations and Service Providers can extend the use of the system to fit their specific use cases.

The first notable extensions are user-provisioned data so application developers can choose to build systems where data is user-owned and stored in a distributed manner with no central point of failure. Adopting this setup will shift data storage for centralized data centers and cloud infrastructure to lower-cost options with the added benefit of reducing the chance of data hacks and leaks as data duplication is unnecessary.

The second notable extension is different threshold setups for organizations to comply with financial regulations and required reporting. Users can opt into different key-share configurations to ensure that BSA, travel rules, central banking practices, and government-mandated disclosures. The user can manage this process rather than external service providers. This compliant setup keeps the user in complete control of their identity, data, and privacy while continuing to use desired or required services.

Bibliography

[AFH20] N. Sullivan R.S. Wahby A. Faz-Hernandez, S. Scott, *Hashing to Elliptic Curves draft-irtf-cfrg-hash-to-curve-06)*, Internet Draft, March 2020.

[AGR+16] Martin Albrecht, Lorenzo Grassi, Christian Rechberger, Arnab Roy, and Tyge Tiessen, *Mimc: Efficient encryption and cryptographic hashing with minimal multiplicative complexity*, Advances in Cryptology – ASIACRYPT 2016 (Berlin, Heidelberg), Springer Berlin Heidelberg, 2016, pp. 191–219.

[BD18] Razvan Barbulescu and Sylvain Duquesne, *Updating key size estimations for pairings*, Journal of Cryptology (2018).

[Con22] World Wide Web Consortium, *Decentralized identifiers (dids) v1.0 core architecture, data model, and representations*, https://www.w3.org/TR/did-core/.

[Cra20] Jake Craige, *An explainer on ed25519 clamping*, July 2020, `https://www.jcraige.com/an-explainer-on-ed25519-clamping`.

[FP18] FIPS-PUB-202, *SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions*, `https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf`, 2018.

[Gar22] Gartner, *Gartner 2019 Hype Cycle for Blockchain Business Shows Blockchain Will Have a Transformational Impact across Industries in Five to 10 Years*, 2022, 12.12.2022.

[GJ20] Kobi Gurkan and Koh Wei Jie, *Community Proposal: Semaphore: Zero-Knowledge Signaling on Ethereum*, 2020, 12.12.2022.

[GKR04] Rosario Gennaro, Hugo Krawczyk, and Tal Rabin, *Secure hashed diffie-hellman over non-ddh groups*, Advances in Cryptology - EUROCRYPT 2004 (Berlin, Heidelberg), Springer Berlin Heidelberg, 2004, pp. 361–381.

[GKR+21] Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger, *Poseidon: A new hash function for Zero-Knowledge proof systems*, 30th USENIX Security Symposium (USENIX Security 21), USENIX Association, August 2021, pp. 519–535.

[HD08] Wong K. Carter G. Hisil, H. and E. Dawson, *Extended coordinates with a=-1 for twisted Edwards curves", The 'add-2008-hwcd-3' addition formulas*, December 2008.

[Hr11] Tony Hansen and Donald E. Eastlake 3rd, *US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)*, RFC 6234, May 2011.

[HWCD08] Huseyin Hisil, Kenneth Koon-Ho Wong, Gary Carter, and Ed Dawson, *Twisted edwards curves revisited*, Advances in Cryptology - ASIACRYPT 2008 (Berlin, Heidelberg), Springer Berlin Heidelberg, 2008, pp. 326–343.

[IEE13] *IEEE standard for identity-based cryptographic techniques using pairings*, IEEE Std 1363.3-2013 (2013), 1–151.

[JL17] Simon Josefsson and Ilari Liusvaara, *Edwards-Curve Digital Signature Algorithm (EdDSA)*, RFC 8032, January 2017.

[KH20] Rafael Schultze-Kraft Kilian Heeg, *78 percent of the bitcoin supply is not liquid*, https://insights.glassnode.com/bitcoin-liquid-supply/.

[Kho19] Dmitry Khovratovich, *Encryption with poseidon*, Dec 2019, `https://drive.google.com/file/d/1EVrP3DzoGbmzkRmYnyEDcIQcXVU7GlOd/view?pli=1`.

[Lam22] Alessandro Lampo, *How is technology accepted? fundamental works in user technology acceptance from diffusion of innovations to utaut-2.*

[LHT16] Adam Langley, Mike Hamburg, and Sean Turner, *Elliptic Curves for Security*, RFC 7748, January 2016.

[Lin98] Carolyn Lin, *Exploring personal computer adoption dynamics*, Journal of Broadcasting & Electronic Media - J BROADCAST ELECTRON MEDIA **42** (1998), 95–112.

[Men18] Ingo & Aalst Wil & Brocke Jan vom & Cabanillas Cristina & Daniel Florian & Debois Søren & Di Ciccio Claudio & Dumas Marlon & Dustdar Schahram & Gal Avigdor & García-Bañuelos Luciano & Governatori Guido & Hull Richard & La Rosa Marcello & Leopold Henrik & Leymann Frank & Recker Jan & Reichert Manfred & Zhu Liming Mendling, Jan & Weber, *Blockchains for business process management - challenges and opportunities. acm transactions on management information systems.*, In press, accepted. 10.1145/3183367.

[Moo22] Geoffrey Moore, *Crossing the chasm : Marketing and selling disruptive products to mainstream customers.*, 2022, Rev. ed. HarperBusiness Essentials.

[MSS17] Alfred Menezes, Palash Sarkar, and Shashank Singh, *Challenges with assessing the impact of nfs advances on the security of pairing-based cryptography*, Paradigms in Cryptology – Mycrypt 2016. Malicious and Exploratory Cryptology (Cham), Springer International Publishing, 2017, pp. 83–108.

[Nak08] S. Nakamoto, *Bitcoin: A peer-to-peer electronic cash system.*, https://bitcoin.org/bitcoin.pdf.

[Res10] Certicom Research, *SEC 2: Recommended Elliptic Curve Domain Parameters*, Standards for Efficient Cryptography, 2010.

[Rog] Everett M. Rogers, *Diffusion of Innovations: An Overview*, Use and Impact of Computers in Clinical Medicine (James G. Anderson and Stephen J. Jay, eds.), Computers and Medicine, Springer, pp. 113–131.

[Rog79] Everett M Rogers, *Network analysis of the diffusion of innovations*, Perspectives on social network research, Elsevier, 1979, pp. 137–164.

[Rog03] Everett M. Rogers, *Diffusion of innovations*, 5th ed., Free Press, New York, NY [u.a.], 08 2003.

[Rog10] E.M. Rogers, *Diffusion of innovations, 4th edition*, Free Press, 2010.

[RS71] Everett M. Rogers and F. Floyd Shoemaker, *Communication of innovations: A cross-cultural approach, 2nd ed.*, 1971.

[RSF07] Alastair Robertson, Didier Soopramanien, and Robert Fildes, *Household technology acceptance heterogeneity in computer adoption*, Department of Management Science, Lancaster University, Working Papers (2007).

[Sch22] Berry Schoenmakers, *Lecture notes on cryptographic protocols*, February 2022, https://www.win.tue.nl/ berry/CryptographicProtocols/LectureNotes.pdf.

[sem22] *Semaphore: Signal anonymously*, August 2022, `https://semaphore.appliedzkp.org/`.

[Wis22] Crypto Wisser, *Exchange graveyard*, 2022, https://www.cryptowisser.com/exchange-graveyard/.

[ZCa17] ZCash, *What is jubjub*, 2017, `https://z.cash/technology/jubjub/`.

[ZCa21] ———, *Zcash protocol specification*, February 2021, `https://github.com/zcash/zips/blob/master/protocol/protocol.pdf`.