JOHNS HOPKINS
WHITING SCHOOL of ENGINEERING | Information Security Institute

J-Card+: Zero Knowledge Identity using Iden3
Mid-Project Progress Report

Team ZKSnacks - Anshul Singhal, Liyin Li, and Pratik Kayastha

I. What has been done?

A. We met with Professor Green and discussed the scope of the project. As ZK identity verification is actively researched and a broad field, we had to narrow our scope. We decided to go with the practical implication of Zero Knowledge Identity.

B. We have completed the literature review and acquired the background knowledge required to complete the project. The following is the list of research papers and articles we went through:

1. "What are zk-SNARKs? & A review of zk-SNARKs."
2. "New privacy practices for blockchain software"
3. "CIRCOM: A Robust and Scalable Language for Building Complex Zero-Knowledge Circuits"
4. "Baby Jubjub elliptic curve - Ethereum Improvement Proposal."
5. "Poseidon: A new hash function for Zero-Knowledge proof systems."
6. "Iden3: Sparse Merkle Trees"

C. We have contacted the Iden3 community through email about our interest in the project.

D. We have started contributing to the Iden3 documentation and submitted merge requests for our contributions.

E. We have installed the tools we will need to develop the application, i.e., snark, Circom 2.0, and Iden3 Go Libraries. We have run them on our local to verify various APIs provided by iden3.

F. We have built a couple of iden3 modules libraries which will be required to build the project.

G. We have designed the JHU wallet application (J-Card+) on paper and have completed a threat analysis of our design. We have identified different components we will need to develop. The list of the components we have identified:

1. **Trusted Issuer:** This backend server will be a trusted setup on the JHU end which will be responsible for submitting claims for students' identity(with their approval)
2. **JHU wallet application**: This will be a web application based on Golang. Students will interact with this web application to generate claims and proofs.
3. **A sample dApp:** We will build a sample dApp to demonstrate the usage of our wallet application.
4. **Smart Contracts:** For on-chain verification and storage.
    a) **Verification.sol:** Smart contract used for on-chain verification of claims.
    b) **JHUIdentity.sol:** Smart contract where we will store Identity State(IdS) for the students participating to use our new Zero knowledge wallet.
    c) **StateTransition.sol**: Smart contract for Identity State Transition(Iden3 provided).

H. We have spoken to the Student DAO community about the scope and adoption of the application and have received exciting feedback and reviews.

II. What still needs to be done?

A. Code Implementation of JHU Wallet Application
   1. Develop a front-end web app using any Javascript framework that creates a wallet to store claims issued by a trusted issuer
   2. Develop a Backend application in Go using iden3 libraries to issue claims to holders.
   3. Create different arithmetic circuits and claim schema to issue the following claims for the JHU wallet application
      a) The student belongs to JHU
      b) Student currently enrolled in JHU
      c) The student is a graduate/undergraduate student
      d) The student's age is more than 18.
   4. Create a sample verifier application that can query the claim holder to generate zero-knowledge proof

B. An alternate way to prove identity to the Issuer
   1. We plan to use a scanner similar to that installed on MSSI Lab to generate claims by fetching information from student JHU cards.
   2. Research on APIs available to fetch student data through JHU systems

C. Contribute to iden3 GitHub open source project as needed

D. Create documentation on how to add more claims to our application

E. Perform threat modeling on our JHU application
   1. Identify different threats
   2. Document and rate different threats
   3. Identify the different scenarios that can reduce privacy provided by the iden3 framework.

F. Research
   1. Research about a way to remove the trust component between issuer, holder & verifier
   2. Research about a way to introduce noise without affecting the protocol of iden3 to improve privacy even if the verifier is a centralized entity

G. Expand Literature review as needed

III. Significant changes to the project Proposal?

A. Instead of building a dApp using ZK Identity, we shifted to building our own ZK JHU Wallet.

B. Initially, we planned to build a wrapper around Iden3 libraries to increase the protocol's adoption. But PolygonID, in July 2022, partnered with Iden3 and published what we were planning to build. Therefore, we dropped the idea to avoid duplication.

J-Card$^+$: Zero Knowledge Identity using Iden3

Capstone Project
Annotated Report Outline
Fall 2022

Anshul Singhal, Liyin Li, Pratik Kayastha (Team ZKSnacks)
Capstone Mentor: Dr. Matthew Green

Abstract

Temporal: Identities are not passive, so should be the proof of access right. When using online services, the most existing authentication process is third-party oriented, and the verifying ID information is not fully autonomous. The key to the problem boils down to the inefficiency of current centralized authentication schemes, using excessive and unnecessary personal information to verify users' access rights. To solve the issue, this project intends to renovate the authentication process through cryptographic means to realize user-centric identity verification with minimal information grants. As a proof-of-concept, an application with identity-based zero-knowledge proof will be developed and delivered by the end of the project.

1 Introduction

1.1 Significance

The proof of access rights contributes to an essential part of modern social functioning. Most online application accesses are third-party authorized, and the governance of verifying information is never user-centric. Evaluating such systems from a software engineering and a computer security perspective is neither maximizing system efficiency nor concerning best security practices. On the one hand, duplicate copies of information must be maintained (on the server side) for verification during authentication. On the other hand, the principle of least privileges is failed to en-place upon users' data concerning from the owner's angle. However, it is an important asset for the overall system.

1.2 Objectives

Empowered by modern cryptography, this project aims to engineer privacy by designs. An authentication module based-off Iden3 [1] which is an innovative identity management protocol will be developed. It also will be integrated into the *J-Card*$^+$, a proof-of-concept online application that targets the Johns Hopkins community. By its design nature, user information is stored in an open, decentralized manner on the Blockchain. And the process of identity verification of authorization is zero-knowledge, meaning that it is the access right being proved but the leverage of person-hood disclosure.

1.3 Definition

Under this project's scope, the authentication problem is characterized by three agents: 1) an identity Issuer, 2) an identity Holder, and 3) an identity Verifier. Corresponding relations and interactions can be visualized in the following figure. An Issuer can create claims for an identity while the Holder manages his identity claims. In requests by a Verifier, a Holder generates and sends associate identity proof based on the respective claim that has private data concealed.

The very property that individual claims are restrictively distributed and controlled by the identity owner is said to be self-sovereign, namely, Self-sovereign Identity (SSI). To further experiment how SSI helps revolutionize social interactions in decentralized autonomous organizations (DAOs), this project

intends to build the *J-Card*$^+$ application, which layouts the potential use cases in the Johns Hopkins University (JHU) community. A JHU student can govern his identity claims issued by the school administration office. And one's identity can be validated by generating and verifying 1) the proof of personhood in school events and/or 2) the proof of membership in a student organization, constructed upon the Zero-knowledge Proofs (ZKPs) cryptographic primitive.

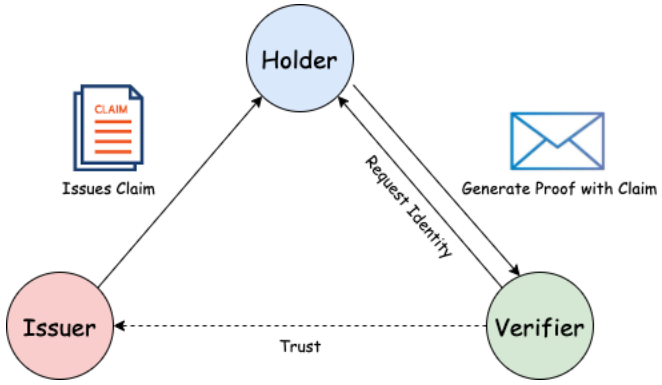

Figure 1: Trust triangle between the identity Issuer, Holder, and Verifier

2 Literature Review

This section provides the necessary background to understand zero-knowledge identity based on the iden3 framework. We first describe the role of zero-knowledge proof (§2.1) in developing software with privacy in mind. Then we describe different components such as zkSNARK (§2.2), Circom (§2.3), BabyJubJub Elliptic Curve (§2.4), Poseidon Hash (§2.5) and Sparse Merkle Tree (§2.6) used by iden3 to create a framework to develop privacy-focused blockchain software.

2.1 Zero Knowledge Proof

A Zero Knowledge Proof (ZKP) is a protocol that allows one party, known as the prover, to convince another, known as the verifier, that a statement about certain data is true without leaking or disclosing any additional information about the statement apart from the fact that statement is indeed true. The property of proving that someone possesses knowledge about specific data without revealing the data itself is desirable for privacy-focused blockchain software. For example, one can prove the following statements by creating zero-knowledge proofs for statements [2].

- "I know a private key associated with this public key": in this case, the prover does not reveal any information about the private key it is holding.
- "I know a pre-image of this hash value": in this case, the proof does not reveal any information about the pre-image value, but it still convinces the verifier that the prover knows the pre-image.
- "The transaction privately transfers a coin": in this case, the proof would not disclose any information about transaction properties such as origin, destination, or amount, but it guarantees that the coin has not been double spent.

Zero-knowledge proofs are classified into interactive zero-knowledge (ZK) and non-interactive zero-knowledge proofs. As the name implies, interactive ZK proofs require interaction between two parties, the prover (proving their knowledge of specific data) and the verifier (validating the proof). The interaction would be in the form of a challenge-response. The verifier challenges the prover, and the prover responds without revealing any additional information other than the statement. The more correct responses the verifier receives from the prover, the less likely the prover is to lie. Non-interactive ZK proofs, on the other hand, are generated once on the prover's side and then sent to the verifier for verification. Non-interactive ZK-proof techniques such as zkSNARK and zkSTARK have been used in recent technologies. (Add references of zcash, zkEVM, etc.). We limit our scope in this project to the

Types of SNARKs				
	Size of Proof π	Size of S_p	Verification Time	Trusted Setup
Groth'16	O(1)	O($\|C\|$)	O(1)	yes (per circuit)
Plonk	O(1)	O($\|C\|$)	O(1)	yes (universal setup)
Bulletproofs	O($\log \|C\|$)	O(1)	O($\|C\|$)	no
STARK	O($\log \|C\|$)	O(1)	O($\log \|C\|$)	no
DARK	O($\log \|C\|$)	O(1)	O($\log \|C\|$)	no

Table 1: Comparison between different types of SNARKs

zkSNARK protocol because the iden3 framework uses it [1]. We examine what zk-SNARK is and its desired characteristics in the following section.

In this project, we focus our scope on only the zkSNARK protocol because, under the hood iden3 framework uses it.

2.2 zk-SNARK

ZK succinct non-interactive arguments of knowledge (zk-SNARKs) are one of the most efficient, popular, and general-purpose zero-knowledge protocols for generating proof [3,4]. It has gained popularity due to its application in ZCash [5], a public blockchain based on bitcoin that uses this proof to ensure that parties involved in the transaction are verified without revealing any information about the parties to each other or the network. The zero-knowledge proof has the following properties, according to SNARK.

- **Succinct** - the size of the proof is minimal (around 200 bytes) regardless of the size of the statement or the witness
- **Non-interactive** - it does not require challenge-response communication between the prover and verifier
- **Argument** - we consider it secure only for provers with limited computational resources, which means that provers with sufficient computational power can persuade the verifier of an incorrect statement
- **Knowledge-sound** - The prover cannot construct proof without knowing a specific so-called witness for the statement

To create a zero-knowledge proof, we primarily focus on the short and non-interactive proof generation characteristics of zk-SNARKs. These characteristics make it very easy to create smart contracts for distributed ledgers that can execute code, such as Ethereum, to verify zero-knowledge proofs in a very short period of time. There are numerous tools available to use zk-SNARK, including circom, zokrates, zinc, snarky, and others. Most tools require developers to create arithmetic circuits for the use case to generate zkSNARK proofs. The main disadvantage of the zk-SNARK is that it necessitates an initial phase known as a trusted setup. The steps include generating random values to obtain a "proving key" and a "verifying key" that must be destroyed. If these random values are leaked, the security of the entire zk-SNARK protocol is jeopardized. However, new protocols, such as zk-STARK, are being proposed that do not require a trusted setup. We will concentrate on the zk-SNARK protocol for this project because the circom language uses zk-SNARKs to generate circuit proofs.

Table 1 shows different time complexity for different implementations of SNARKs. As we can observe from the table, Groth'16 and Plonk/Martin SNARK implementation can be efficient in terms of proof size and verification because they do not depend on the circuit, but proof generation takes linear time.

2.3 Circom 2.0

Circom is a newly developed domain-specific language (DSL) to define arithmetic circuits and its associated compiler written in Rust [6]. The circom's primary goal is to give programmers a comprehensive framework for building arithmetic circuits using a simple user interface while abstracting

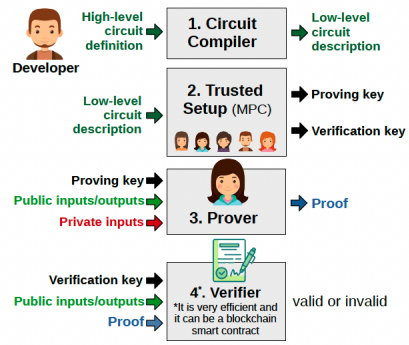

Figure 2: Generate and verify ZK proofs using Circom [2]

```
1  pragma circom 2.0.0;
2
3  /*This circuit template checks that c is the multiplication of a and b.*/
4
5  template Multiplier2 () {
6
7     // Declaration of signals.
8     signal input a;
9     signal input b;
10    signal output c;
11
12    // Constraints.
13    c <== a * b;
14 }
```

Listing 1: Multiplication Code in Circom - multiplier2.circom [6]

the complexity of the proving procedures [7]. While zk-SNARKs, like most ZKPs, allow the proof of computational assertions, they cannot be used to solve the computational problem directly; instead, the statement must be transformed into the appropriate form. The computational statement for zk-SNARKs must be represented by an arithmetic circuit. A circom compiler can therefore be used to produce a zero-knowledge proof from an arithmetic circuit.

The figure (2) depicts the general steps for creating and validating proof using circom. Each step will be explained with the assumption that we want to prove the statement "I can factor number 33." To begin, we must create a program in arithmetic circuits using circom. We will build the multiplication circuit shown in Listing (1) to demonstrate the preceding statement. In line 5, we create a template or, as we commonly refer to it, a function in languages like Python, Java, C++, etc. In lines 8 & 9, we define private inputs or factors of some number. At 10, we define output of the arithmetic circuit which is a multiplication of two factors, as shown in line 13. First, we will take this higher-level code written in circom, compile it using circom compiler and convert it into lower-level language files with extension R1CS and Wasm format by running the following command.

circom multiplier2.circom –r1cs –wasm –sym –c

R1CS contains optimized constraints in quadratic, linear, or constant equations defined in higher-level circom code. The WASM file contains data to generate witnesses for zero-knowledge proofs. Second, once we get a low-level circuit description (r1cs & wasm files), we need to create a proving and verification key by performing an initial phase called trusted setup. For simplicity, we are not discussing different utilities developed by circom to create these keys, but https://docs.circom.io/getting-started/proving-circuits/ has a tutorial on creating them. Third, once we obtain these keys, we create proof using the proving key, private inputs (in our example cases 11 & 3), and public input/outputs. In the Last step, the verifier can verify the proof by verification key, public input/output, and proof. The verifier can be a web2 application or a smart contract on the blockchain.

2.4 BabyJubJub Elliptic Curve

Baby JubJub is an Elliptic curve optimized for use with zk-SNARKs [8]. Traditionally, public and private key pairs are used to authenticate users. The public and private keys can be generated by performing addition or multiplication operations on the Elliptic curve. The public and private key pairs are used in the Iden3 Protocol to manage identity and authenticate in the name of identity. As a result, we must demonstrate ownership of the public key without revealing the private key. To put it another way, we must create an arithmetic circuit that verifies that a given secret key corresponds to a given public key. The BabyJubJub elliptic curve makes it possible to build this circuit quickly.

Definition of parameters of curve
Baby JubJub is an elliptic curve defined over a prime field F_p with

$$\text{p} = 18882428718392752222464057452572750885483644004160343436982041865758084956 17$$

and described with the equation,

$$ax^2 + y^2 = 1 + dx^2y^2 \tag{1}$$

where a=168700 & d=68696

For simplicity, we will not discuss how circom implements different elliptic curve operations, such as multiplication and addition of points on the curve. We will use those circuits as black box circuits, but the definition of those circuits can be found in circomLib [9].

2.5 Poseidon Hash

knowledge-proof systems such as zk-SNARK, zk-STARK, bulletproofs, etc. A deterministic one-way function known as a cryptographic hash function converts data of any size to data of a specific size. They are mainly employed to recognize and validate the integrity of files, papers, and other kinds of data. Because they rely so heavily on bit operations, traditional hash algorithms like SHA256 are particularly ineffective when used in ZK protocols to create arithmetic circuits. For instance, the SHA256 implementation in circom includes almost 59,000 limitations. On the other hand, Poseidon hash construction offers a performance advantage in addition to being compactly stated as a circuit that can be customized for different proof systems using specifically created polynomials [10]. The Poseidon hash function is frequently used in the iden3 framework with Baby JubJub Elliptic curve, Merkle tree, and any other use cases needing a secure hash function.

2.6 Sparse Merkle Tree

A Merkle Tree or a hash tree is a tree data structure in which each leaf node contains the cryptographic hash of some data, and each non-leaf node contains the cryptographic hash of its child nodes. Merkle trees enable the linking of a set of data to a unique value, which is very efficient and useful in blockchain technology because it provides a secure and efficient way to verify large sets of data by storing only a small amount of information (the Merkle tree's root) on the blockchain. If most of the leaf nodes in a Merkle tree are empty, it is called a "Sparse Merkle Tree." The Merkle trees used in the iden3 framework are Sparse Merkle trees [11]. Moreover, in sparse Merkle trees, each data block has an associated index that indicates its position as a leaf inside a tree. The sparse Merkle trees used in iden3 protocols have the following properties:

- **Binary** - Each node can only have two children
- **Sparse and Deterministic** - The contained data is indexed, and each data block is placed at the leaf corresponding to that data block's index, so insert order does not influence the final Merkle tree Root. This also means that some nodes are empty.
- **ZK friendly** - The hash function used for the Merkle tree, Poseidon, plays well with the zero-knowledge proofs (ZKP) used in different parts of the protocol.

Figure (3) shows what a Merkle tree looks like. Apart from inheriting temper-resistance, proof of membership, and scalability features from the Merkle tree, the sparse Merkle tree also provides the feature of proof of non-membership. It is easy to prove "proof non-membership" in sparse Merkle trees

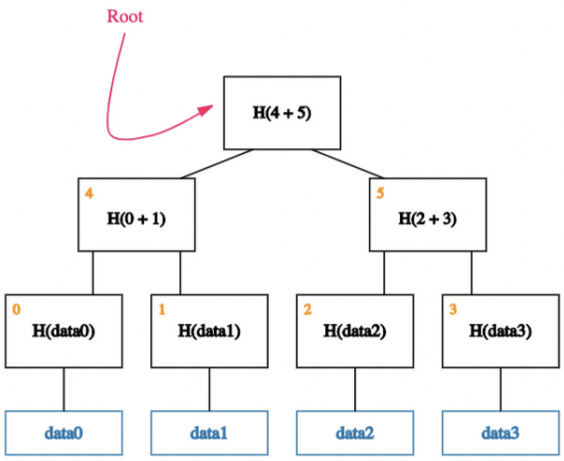

Figure 3: Merkle Tree
Source: https://docs.iden3.io/basics/key-concepts

because most of the leaf node contains null data (empty). It is equivalent to proving the membership of a null data block.

At iden3, sparse Merkle trees allow scalability. Any identity or user should be able to generate as many claims as he/she wants. Achieving this goal requires minimizing the amount of data stored on-chain. Sparse Merkle Tree provides a way to achieve that goal by storing only the root node on the chain. In other words, Merkle trees allow prolific claim (a statement about something) generators to add/modify millions of claims in a single transaction. This makes it easy to scale the claims. We discuss more about claims in the iden3 section of the report.

3 Technical Solution, Design, and Analysis

3.1 Iden3 Overview

To understand iden3, we first have to understand a few terminologies.

- **Identity** - In the blockchain world, when we talk about identity, we mean accounts, which will be smart contracts. So you can think of identities as smart contracts, where the address of the contract is the identifier of that identity. [12]
- **Claims** - Claims are statements said by an identity about itself or another identity. Claims can be public or private. For example, "I" as an identity on the blockchain can claim that I am a JHU student.

Now that we understand the terminologies, it brings us to the question, what is iden3 providing us? Iden3 is a protocol to create and maintain identities and claims with the following properties:

- Scalability
- Temper resistance
- Zero-Knowledge verification.

The Iden3 organization has implemented libraries for their new protocol, mainly in Go.

3.2 Iden3 Protocol Specification

As the introduction states, Iden3 is a protocol to create and maintain identities and claims. Let's see how we store these components, what operation we can perform, and how the iden3 protocol guarantees the above properties, i.e., Scalability, temper resistance, and ZK-verification.

3.2.1 Claims

3.2.1.1 Claim Data Structure

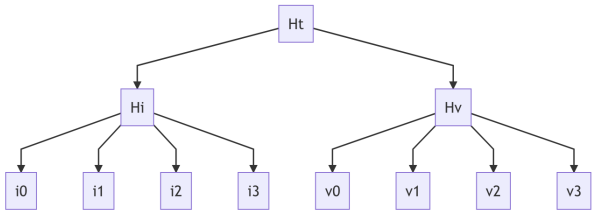

Figure 4: Claim Data Structure
Source: https://docs.iden3.io

Each claim is composed of two parts: the index part and the value part. The Indexes are i0, i1, i2, and i3, and the Values are v0, v1, v2, and v3. The protocol reserves i0, i1, and v0, v1 for its internal use; therefore, they cannot be utilized to store other information. The following defines these reserve slots in the claim data structure.

```
i_0 [253 bits]:[128 bits] claim schema hash(see below)
               [32 bits] header flags
                 [3] Subject:
                   000: A.1 Self
                   001: invalid
                   010: A.2.i OtherIden Index
                   011: A.2.v OtherIden Value
                   100: B.i Object Index
                   101: B.v Object Value
                 [1] Expiration: bool
                 [1] Updatable: bool
                 [27] 0
               [ 32 bits ] version (optional?)
               [ 61 bits ] 0 - reserved for future use


i_1 [253 bits]:[248 bits] identity (case b) (optional)
               [5 bits] 0

v_0 [253 bits]:[ 64 bits ]  revocation nonce (see below)
               [ 64 bits ]  expiration date (optional)
               [ 125 bits] 0 - reserved
v_1 [253 bits]:[ 248 bits] identity (case c) (optional)
               [  5 bits ] 0
```

Figure 5: Bit Structure

Note that the hash of the index slots (hash(i_0,i_1,i_2,i_3)) is unique for the whole tree. That means you cannot have two claims with the same index inside the tree. This property is critical; this is how we decide if the data corresponding to the claim will be stored in index slots(i2, i3) or value slots(v2, v3).

3.2.1.2 Claim Schema

Claim Schema is a JSON-LD document that defines which data should be stored inside which data slots(i2, i3, v2, v3). The Hash of these schemas is stored in the i0 slot when we create a claim.

3.2.2 Keys

In Iden3 or any cryptographic protocol, keys are used to authorize and authenticate new transactions. In Iden3, a transaction would be creating/updating/revoking claims; thus, keys are required to au-

thorize these transactions. In Iden3, we use Baby Jubjub(BJJ) Elliptic Curve algorithm to generate public and private keys. This type of key is designed to be efficient while working with zkSNARK.

3.2.2.1 Auth Claim or Key Authorization Claim

Auth claim is a claim which claims that you own a private key or a public key. In other words, as an identity, you claim that you hold the BJJ private key of this BJJ public key stored in the claim. This claim is very important because whenever you issue a new claim, you need to prove that you are the owner of this identity. You do this using a digital signature. Cryptographically, you will issue a digital signature to a transaction (new/update/revoke a claim) using your private key. Now, any verifier can verify the validity of this transaction by confirming the digital signature (without revealing the private key). This is important because this will be the first claim you will issue when creating an identity.

3.2.3 Identity

Now that we understand what the claims look like, it is finally time to understand how identity is defined. An identity constitutes all the claims the identity has issued or received from other identities. For example, "I" as an identity can put forward many claims, i.e., I am a JHU student, an MSSI Student, and an International student. All these claims say different statements about me, but all these claims are made for one identity, which constitutes an identity. In other words, identity is built by what the identity and others have said about the identity.

3.2.3.1 Identity Data Structure

Three Sparse Merkel Trees define an Identity [12].

- Clt: Clt is the claims tree. This tree is where we store all of our claims.
- Ret: Ret is called a revocation tree. Remember, in the claims data structure, we stored revocation nonce in the v0 slot. Those nonces can be used to revoke a claim by simply adding the nonce to the Revocation Tree(Ret).
- RoT: This tree stores all the historical versions of Clt roots. In Simple words, when we add a new claim to the claims tree, the root of the Clt changes. But before changing the Clt, we add the previous root tree value to the Root tree (RoT). In future sections, we will explain why this is required.

That's it; these three above Merkel Tree are all that how we represent an Identity. With this construction, the identities can issue, update, and revoke claims. An identity must issue at least one Auth Claim to operate correctly. This is the first claim issued by identity, which must be added to the Clt.

3.2.3.2 Identity State

An Identity State ***IdS*** is represented by the hash of the roots of these three Merkle trees.

$$IdS = H(rootofClt||rootofRet||rootofRoT) \tag{2}$$

3.2.3.3 Genesis State

The very first identity state of identity is defined as Genesis State. To create a genesis state, we need at least one claim in the claims tree(Clt). As mentioned before, the first claim is always an AuthClaim, and the Rot and Ret will be empty. The IdS that we get at this state is called Genesis State.

3.2.3.4 Genesis ID or Identifier

We calculate the Genesis ID or Identifier (ID) from the Genesis State. This ID is unique and permanent for the entire lifecycle of an identity. This ID is stored as a mapping in the smart contract; more specifically, we store the mapping of ID $\Rightarrow$ IdS in a smart contract. Therefore, as we add more claims to the claims tree, the IdS changes, and we update the value in the map. This way, we get timestamped data of all the history of identity.

3.2.3.5 Identity State Diagram

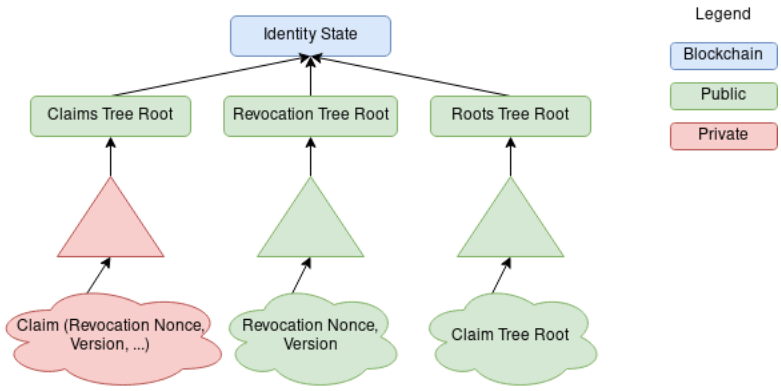

Figure 6: Identity State Diagram
Source: https://docs.iden3.io

This diagram denotes where we store the data we created we talked about. All the claims that we created would be off-chain and will be private. We will store IdS on the blockchain; everything else will be public information. This can be directly shared with the verifier or hosted on an HTTPS website.

3.3 Iden3 Protocol Analysis

At the beginning of the discussion, we introduced three properties that the Iden3 protocol provides. This section will cover them in more detail.

3.3.1 Scalability

There are several reasons why the protocol is very scalable.

- We only store IdS on the blockchain and the rest of the data is stored off-chain. This approach saves us a bunch of gas money and scales the protocol.
- Sparse Merkel Trees are designed to provide quick proof of membership and proof of non-membership. Thus, claim lookup is very efficient.
- We utilize BJJ and Posiden Hash functions designed to work efficiently with the zkSNARK.

3.3.2 Zero-Knowledge Verifiable

ZK verification is the most crucial property of the protocol. The property states that any verifier can verify the claims without learning new information while executing the protocol. The zero-knowledge verification is done through zkSnarks.

3.3.3 Temper resistance

Temper resistance is the property that claims that data stored in the Identity Trees cannot be corrupted. Temper resistance is a property we inherit from the Merkel Trees. Also, changing the state requires proof of ownership, and this verification happens on-chain through zkSnark before changing the IdS. Therefore, we can be sure only the person who holds the private key has put forward claims, and these claims are not tampered with.

3.4 Applied Iden3 Overview

J-Card$^+$ is a mobile/web application that implements the Iden3 protocol. It is an example of a Wallet application that stores the credentials required for operations on blockchains. As identity data is virtual assets registered on the blockchains, *J-Card*$^+$ is built to manage its updates and retrievals. An

identity Holder keeps an inventory of identity claims in his/her wallet app. As a new identity claim is issued, the wallet fetches it from the Issuer. At request by Verifier, the wallet computes zero-knowledge proof locally for identity verification. As a result, user data is guarded by the *J-Card*$^{+}$ securely that sensitive personal data can only be distributed under the wallet owner's consent.

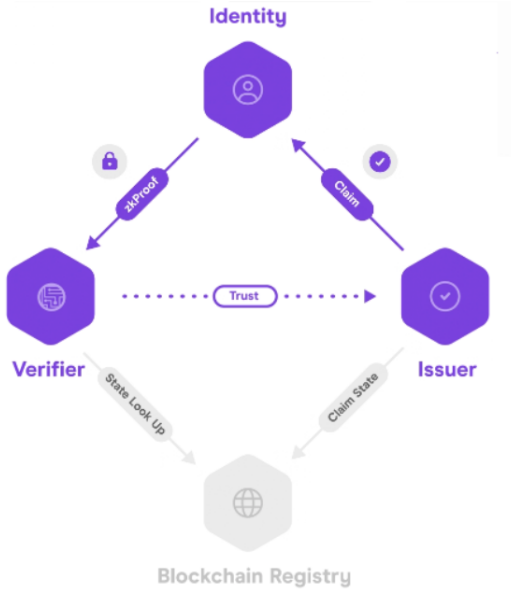

Figure 7: Trust triangle among Issuer, Holder, and Verifier [13].

In this building system, two critical role interactions are 1) Issuer-Holder, and 2) Holder-Verifier. Corresponding control flow graphs are displayed in the following accordingly.

3.4.1 Interaction between the Issuer and Holder

As the first system function, the Issuer-Holder interaction results in the creation of identity claims and the registration of identity data on blockchains.

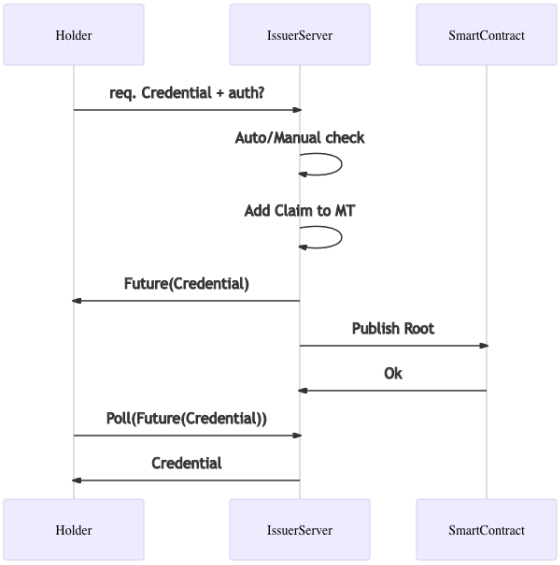

Figure 8: Sequence diagram the interaction between the Issuer and the Holder [13].

3.4.2 Interaction between the Holder and Verifier

As another critical system function, the Holder-Verifier interaction is responsible for a zero-knowledge proof generation and verification upon a particular claim.

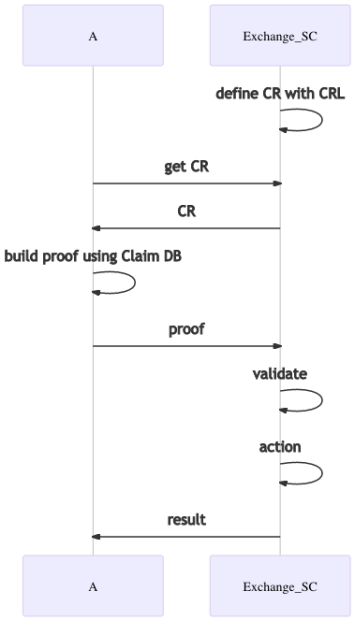

Figure 9: Sequence diagram for the interaction between the Holder and the Verifier [13].

4 Experimentation, Evaluation, and Result Analysis

Here we will add observations in the final report.

5 Conclusion

Once we complete the implementation of our project, we will add a conclusion in our final report.

6 References

[1] "iden3." [Online]. Available: https://iden3.io

[2] M. Bellés Muñoz, J. Baylina Melé, V. Daza Fernández, and J. L. Muñoz Tapia, "New privacy practices for blockchain software," *IEEE software*, 2021.

[3] "What are zk-snarks?" [Online]. Available: https://z.cash/technology/zksnarks/

[4] T. Chen, H. Lu, T. Kunpittaya, and A. Luo, "A review of zk-snarks," *arXiv preprint arXiv:2202.06877*, 2022.

[5] E. B. Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza, "Zerocash: Decentralized anonymous payments from bitcoin," in *2014 IEEE symposium on security and privacy*. IEEE, 2014, pp. 459–474.

[6] "Circom 2 documentation." [Online]. Available: https://docs.circom.io/

[7] J. L. Muñoz-Tapia, M. Belles, M. Isabel, A. Rubio, and J. Baylina, "Circom: A robust and scalable language for building complex zero-knowledge circuits," 2022.

[8] B. WhiteHat, J. Baylina, and M. Bellés, "Baby jubjub elliptic curve," *Ethereum Improvement Proposal, EIP-2494*, vol. 29, 2020.

[9] "Circomlib." [Online]. Available: https://github.com/iden3/circomlib

[10] L. Grassi, D. Khovratovich, C. Rechberger, A. Roy, and M. Schofnegger, "Poseidon: A new hash function for Zero-Knowledge proof systems," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 519–535.

[11] J. Baylina and M. Bellés, "Sparse merkle trees."

[12] "Iden3 documentation." [Online]. Available: https://docs.iden3.io/

[13] "Polygon id documentation." [Online]. Available: https://0xpolygonid.github.io/tutorials/

7 Appendices

We will add our code snippet of Issuer, Verifier & Holder implementation. And rest of the code will be pushed to GitHub.