CERIAS Tech Report 2019-01

PrivIdEx: Privacy Preserving and Secure Exchange of Digital Identity Assets

by H. Gunasinghe, A. Kundu, E. Bertino, H. Krawczyk, K. Singh, S. Chari, D. Su
Center for Education and Research
Information Assurance and Security
Purdue University, West Lafayette, IN 47907-2086

PrivIdEx: Privacy Preserving and Secure Exchange of Digital Identity Assets.

H. Gunasinghe
Purdue University
huralali@purdue.edu

A. Kundu
IBM Research
akundu@us.ibm.com

E. Bertino
Purdue University
bertino@purdue.edu

H. Krawczyk
IBM Research
hugokraw@us.ibm.com

K. Singh
IBM Research
kapil@us.ibm.com

S. Chari*
IBM Research
snchari@gmail.com

D. Su*
IBM Research
sudong.tom@gmail.com

ABSTRACT

User's digital identity information has privacy and security requirements. Privacy requirements include *confidentiality* of the identity information itself, *anonymity* of those who verify and consume a user's identity information and *unlinkability* of online transactions which involve a user's identity. Security requirements include *correctness*, *ownership assurance* and *prevention of counterfeits* of a user's identity information. Such privacy and security requirements, although conflicting, are critical for identity management systems enabling the exchange of users' identity information between different parties during the execution of online transactions. Addressing all such requirements, without a centralized party managing the identity exchange transactions, raises several challenges. This paper presents a decentralized protocol for privacy preserving exchange of users' identity information addressing such challenges. The proposed protocol leverages advances in blockchain and zero knowledge proof technologies, as the main building blocks. We provide prototype implementations of the main building blocks of the protocol and assess its performance and security.

KEYWORDS

Decentralized identity asset exchange; Privacy preserving; Unlinkability; Counterfeit elimination; Blockchain; ZK-SNARK

1 INTRODUCTION

Access to services offered by online service providers (SPs) is controlled by identity verification processes. Such a process requires users to verify different types of identity information, depending on the sensitivity of the service. Some services require to verify individual pieces of identity information, such as email address, phone number and Social Security Number (SSN), and some other services require to verify composite identity information such as driver's license and passport. Certain other services require to perform rigorous due diligence processes to satisfy certain compliance requirements such as Know Your Customer (KYC) compliance in banking/financial services [5]. Similar example scenarios which involve lengthy identity verification processes include, but not limited to, joining a new employer, applying for temporary visa in a foreign country, etc.

The amount of resources and effort that users and SPs have to devote to these identity verification processes vary depending on the type of identity information being verified. For examples, verifying an email address only requires the SP to send an email to the given email address, asking the user to click a link in it, verifying an SSN may require the SP to consume a paid service offered by the SSN authority, and verifying KYC compliance requires the SP to perform background checks on the user to verify the user's status of credit score, Anti-Money Laundering (AML), Counter Terrorist Financing (CTF), Politically Exposed Person (PEP) [5], etc. Once an SP has verified the identity of a user, the package of information associated with such *verified identity* becomes an asset of the SP, which we refer to as *identity asset* (the National Strategy for Trusted Identities in Cyberspace (NSTIC) labels such verified identity information as belonging to LOA (Level Of Assurance) 2+ category [20]). On the other hand, since an identity asset contains personal information about a user, the user also becomes an owner of the identity asset, leading to having two legitimate owners per *identity asset*.

Currently, when a user needs to consume similar services from different SPs, the user is treated as an "alien" by each SP and is required to go through a similar identity verification performed by each SP. These repeated processes not only are costly, but also are inherently error-prone, causing inconvenience to both parties. These issues magnify especially in scenarios involving lengthy identity verification such as consuming financial services from multiple banks, joining multiple employers, applying for temporary visa in multiple countries, etc. If there were a standard protocol through which different SPs could share the same *identity assets* of a user, that would result in substantial cost savings and notable convenience to both parties [5]. The SP who originally performed the identity verification for the user, and hence is one of the owners of the identity asset, can be incentivized for sharing the identity asset, in exchange of a monetary compensation by the subsequent SP(s) that the user interacts with.

Let us consider the following **use case**: *The user Ursula first consumes financial services from bank A where bank A performs identity verification and due diligence steps for KYC compliance on Ursula. Later Ursula needs to consume financial services from bank B as well. Bank B wants to know if Ursula has already performed KYC*

*this work was done when the author was at IBM Research.

WWW '19, May 13–17, 2019, San Francisco, CA, USA

ACM ISBN 978-1-4503-6674-8/19/05.
https://doi.org/10.1145/3308558.3313574

compliance verification and if so, both Ursula and bank B would like to re-use the corresponding identity asset. However, in this use case, Ursula would not like to reveal to bank B which bank(s) she has interacted before, and would not like to reveal to bank A, which other bank(s) she is planning to be a customer of. Bank A and bank B themselves would also not like to reveal their identity to each other during potential identity asset exchange, due to competition in business. On the other hand, Ursula would not like the transactions she carries out with different banks based on the same identity asset to be linkable. Prominent privacy protection regulations, such as General Data Protection Regulation (GDPR) [27], also treats a user's transactional patterns as personal data and prohibits tracking such personal data. Therefore, *anonymity of the parties who exchange the identity asset* and *unlinkability of the transactions* are key privacy requirements to be addressed when designing a protocol to facilitate identity asset exchange during online transactions, in addition to protecting *confidentiality* of identity assets from external parties.

The existing identity management protocols, which facilitate sharing users' identity information, do not address all the key privacy requirements. For examples, OpenIDConnect [22], an industry standard widely used by SPs to obtain a user's identity information from an identity provider (IDP), does not preserve anonymity and unlinkability. Two nation-scale brokered identification systems built by the USA and UK governments, namely, Federal Cloud Credential Exchange (FCCX) [21] and GOV.UK Verify [16], respectively, focus on the aforementioned use case and the first privacy requirement, i.e. protecting from each other the anonymity of the parties who exchange users' identity information, in order to preserve users' privacy. Those systems, however, use a government managed broker to mediate the identity exchange transactions, in which case, the identity of the two exchanging parties is revealed to the broker, although the two parties stay anonymous to each other. Brandao et. al [7] have raised certain other privacy concerns on the introduction of such a centralized broker.

One of our goals is to avoid introducing such a centralized broker. Therefore, the proposed protocol is executed in a decentralized identity management ecosystem backed by a permissioned blockchain network (see Section 2.1). Distributed trust implemented on the basis of the consensus protocol through which blockchain peers validate protocol executions eliminates the requirement of a centralized broker. Participants of the decentralized identity management ecosystem invoke the identity exchange transactions with pseudonyms, in order to preserve anonymity and unlinkability. However, when anonymity and unlinkability are enforced in confidential identity asset exchanges, without a mediating centralized party, it is challenging to achieve the required security properties, such as *correctness*, *ownership assurance* and *counterfeits elimination* of the identity assets, and optionally, *financial fairness* of the identity asset exchange transactions, because the participants, appearing with a new pseudonym in each round of the protocol execution, can violate such security properties, as discussed in Section 3. This problem can be related to the challenge of preventing double spending in bitcoin [19] and ZeroCash [4], without a centralized financial institute managing the payment transactions. However, unlike in bitcoin and ZeroCash, whose goal is to prevent double spending of cryptocurrency, which has a single owner at a time, our goal is to enable multiple exchanges of the same identity asset which has two owners, which poses a different set of challenges. We have designed the dedicated phases of the protocol to address such challenges leveraging the power of ZK-SNARKS (see Section 2.2).

Our main contribution is PrivIdEx - a protocol realizing privacy preserving and secure exchanges of identity assets in a decentralized identity management ecosystem, including its: i) design, ii) implementation and iii) analysis and evaluation.

2 PRELIMINARIES

2.1 Permissioned Blockchain

We identify two main parties in a BC [25] network as follows: i) *peers* - they maintain the transaction ledger (i.e. BC) and host the smart contract(s); ii) *participants* - they perform transactions. The consensus algorithm defines rules to be followed by peers when ordering and validating the transactions to be added to the ledger. A *smart contract* defines the business logic for transaction execution and validation, which is invoked by participants and run independently by each peer for executing the transactions.

There are two types of BCs: i) Permissionless BCs - where any one can join the network and write to/read from the BC. Participants' identity is hidden by the random pseudonyms they choose, which results in lack of accountability. Cheating by the peers is avoided and the correctness of the ledger is preserved by employing an expensive consensus algorithm and the assumption that the majority of the computation power of the network is with honest parties. ii) Permissioned BCs - where a trusted certification authority (CA) issues signed X.509 certificates to actors (i.e. peers and participants), which include the public key of a RSA key pair and other identity attributes that determine their permissions (i.e. read/write access to BC). This preserves accountability and enables employing a less costly consensus algorithm. Permissioned BCs are categorized as public and private based on whether read access is controlled or not, respectively. We assume a decentralized identity management ecosystem backed by a permissioned BC when designing the proposed protocol, in particular, Hyperledger Fabric [13], a private permissioned BC.

2.2 ZK-SNARKs

ZK-SNARKs is an efficient construction to prove in zero-knowledge, a satisfying assignment to the class of problems called Quadratic Span Program (QSP) [15]. QSP is an NP-complete problem. According to the principles in complexity theory, for any NP problem L and an NP-complete problem M, there is a reduction function f, which is computable in polynomial time, s.t. $L(x) = M(f(x))$. Accordingly, ZK-SNARKs can be used to prove in zero-knowledge, a satisfying assignment to any NP problem following these high level steps: i) formulate the decision problem D as an NP statement, which is expressed in the following form: Given a set of public inputs X, I know a set of secret inputs W, s.t. condition D holds on X and W (i.e. the satisfying assignment is constituted by X and W); ii) write an algorithmic program P to solve D; iii) convert P to an arithmetic circuit C; iv) convert C to a QAP (Quadratic Arithmetic Program - a variant of QSP); v) prove/verify satisfiability for the QAP in zero knowledge.

The ZK-SNARKs construction is expressed in following three algorithms: ***i) Generator (G)***: takes as inputs: C and secret parameters λ, and outputs a proving key (pk) and a verification key (vk). This is a one time setup step run by a trusted party, after which λ should be destroyed in order to preserve the soundness of the proofs. (pk, vk) := G(C, λ). ***ii) Prover (P)***: takes as inputs: C, pk and the satisfying assignment - which may have both private inputs w and public inputs x, and outputs the proof Φ. Φ := P (C, pk, w, x). ***iii) Verifier (V)***: takes as inputs: vk, Φ and public inputs x provided by the prover, and outputs the decision d as true, iff (w, x) is a satisfying assignment to P, and false, otherwise. d := V(vk, Φ, x).

In ZK-SNARK (*Zero-Knowledge Succinct Non-interactive Argument of Knowledge*), the *Zero-Knowledge* property enables participants to keep transaction information confidential, and still prove to peers that transactions are valid according to the smart contract. *Succinctness* makes the size of such proofs small (≈2KB) and verification time in the orders of milliseconds, irrespective of the complexity of the business logic defined in the smart contract. The *Non-interactive* property enables multiple peers to verify the proofs independently without interacting with the prover. There are other zero-knowledge proof constructions developed to achieve similar goals without a trusted setup, such as ZK-STARK [3] and Bullet Proof [8]. We use ZK-SNARK since it is more efficient and practical compared to the other constructions.

3 SYSTEM MODEL AND THREATS

3.1 System model

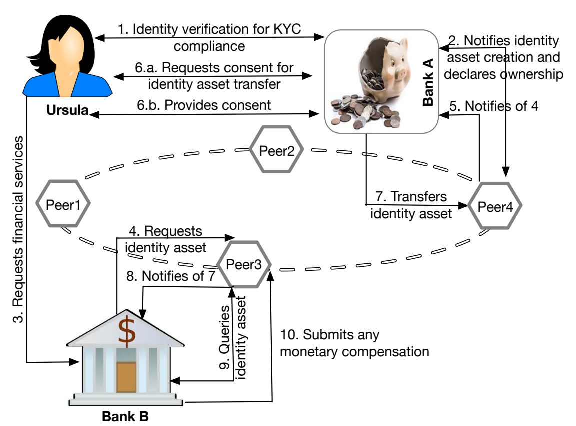

Figure 1: High level steps of a decentralized identity asset exchange protocol

The following steps describe the basic flow of identity asset exchange for the use case mentioned in Section 1, which is illustrated in Figure 1: 1) When Ursula consumes financial services for the first time from bank A, identity verification and due diligence are performed by bank A to verify KYC compliance of Ursula, and the resulting identity asset is stored at bank A. Note that the details of how the identity verification is performed is out of scope of the identity exchange protocol. 2) Bank A notifies the identity ecosystem about the identity asset creation and claims its ownership. 3) At a later point in time, Ursula requests financial services from bank B and they discover by some means (either by querying the BC or Ursula's private records) if the required identity asset is already created for Ursula. 4) If this is the case, Bank B requests from the identity ecosystem the relevant identity asset of Ursula. 5) Bank A receives the request submitted by bank B, via the transaction notification system of the BC. 6.a) If bank A decides to share the identity asset, bank A requests the consent from Ursula to transfer the identity asset to bank B. 6.b) Ursula provides her consent. 7) Bank A transfers the identity asset, along with Ursula's consent, via the identity ecosystem. 8) Bank B receives the notification about the valid identity asset transfer. 9) Bank B queries the identity asset from the ledger. 10) Optionally, if the transferred identity asset is correct, bank B submits a monetary compensation (if it is required, by the policies of the identity ecosystem, in order to ensure financial fairness) to bank A. This could be a bitcoin payment sent to bank A, if the underlying BC supports bitcoin transactions, which bank A can redeem later.

We make the following **three basic assumptions** in the context of decentralized identity asset exchange: 1) There is a criteria to define and verify uniqueness of an identity asset so that if multiple copies of a particular type of identity asset are created by multiple parties using identity information of a given user, they all become digitally identical (e.g. defining a standard format for an identity asset used for a particular identity verification scenario and considering the cryptographic hash (CRH) of the identity asset to be the criteria for verifying uniqueness). 2) If a particular type of identity asset is created for a given user in the identity ecosystem, all the SPs requiring a similar identity asset from the user should re-use it, without re-creating it. 3) The trusted CA, which issues the identity certificates to the actors of the BC network, does not collude with any actor in the BC.

Note that we incorporate the required privacy features into the protocol in an incremental manner. Therefore, we first consider Version 0 of the protocol, which does not include any privacy features, i.e. all the participants appear in their real identities and identity assets are transferred in plain text. Although such a model is not used in a real world deployment, we use it as the baseline to identify what properties should be achieved in order to guarantee that an identity asset exchange protocol is secure, and to analyze various challenges in achieving the those security properties when privacy features are incorporated into the protocol incrementally.

3.2 Threat model for protocol security

In what follows, we first identify the different ways in which an adversary can compromise the security of the plain protocol model. **1) Compromising *correctness***: A malicious identity asset provider (e.g. bank A) transfers an identity asset in step 6, which is different than the one it created in step 2. **2) Compromising *ownership assurance***: A malicious collusion of two of the three parties engaged in an identity asset exchange can compromise the ownership assurance of one of the two legitimate owners as follows: i) a different user can collude with bank A to impersonate Ursula at bank B; ii) bank A and bank B can collude to transfer the identity asset without Ursula's consent; iii) after bank B obtains the identity asset from bank A, Ursula and bank B can collude to act as the original owners and transfer the identity asset to a different bank. Attacks i) and ii) are possible by sending a fake consent in step 6.b and attack iii) is motivated by any benefits obtained from the identity asset exchange, such as monetary compensation paid by the identity asset consumer to the identity asset provider. **3) Creating *counterfeits*** of an identity asset: Due to the same motivation for attack 2.iii, bank B may execute step 2 using the identity asset received from bank A, hence creating a counterfeit. **4) Compromising *financial fairness***: After receiving the identity asset, bank B can abort skipping step 10. Or if bank B makes the payment first, bank A can abort skipping step 7.

Accordingly, *correctness*, *ownership assurance*, *counterfeits elimination* and *financial fairness* are key requirements to be addressed, in order to guarantee the security of an identity asset exchange protocol. In what follows, we describe the simple mechanisms that should be incorporated into protocol V0, in order to address those requirements. **1) *Correctness*** : The smart contract that defines the protocol requires from bank A to submit the cryptographic hash (CRH) of the identity asset in step 2. In step 7, bank A submits a pointer (e.g. transaction ID) to step 2 associated with the identity asset being transferred. Then the peers running the smart contract validate correctness by computing the CRH of the transferred identity asset and comparing it with the CRH submitted in step 2 associated with the same identity asset. **2) *Ownership assurance***: The protocol requires: i) from bank A to submit the CRH of the public keys of the two owners in step 2; ii) from Ursula and bank A to sign, using their private keys, the messages sent in step 6.b (consent by the user) and step 7 (identity asset transfer), respectively. Then the peers verify the signatures and confirm that the original owners of the identity asset indeed performed the transfer. Note that an adversary can replay the message sent in step 7. Therefore, the protocol should also require bank B to send a random nonce in the identity asset request message (step 4), which Ursula and bank A should include in the messages they sign in steps 6.b and 7. **3) *Counterfeits elimination***: To ensure that duplicates of an identity asset do not exist, the peers maintain a hash table which is indexed by the CRH of the identity asset and which stores the information submitted in step 2 of the protocol (i.e. CRH of the public keys of the two owners). Each time step 2 is executed for a newly created identity asset, the peers check if the newly submitted CRH already exists in the hash table, in which case the peers reject it as an attempt to create a counterfeit of an identity asset. **4)*Financial fairness*** : It is unlikely that bank B skips step 10 in protocol V0 where it appears with its real identity, as it would damage its reputation. Even if bank B does so, it is easy to take actions against bank B for the dishonest behavior.

3.3 Threat model for users' privacy

In what follows we discuss an adversary's goals in compromising users' privacy (i.e. learning and tracking information that users do not intentionally share) in protocol V0. **1) Compromising *confidentiality* of users' identity information**: The adversary learns the users' identity information from the identity assets transferred in plain text via the BC in executions of step 7. **2) Compromising users' *transactional privacy***: i) the adversary learns the identity of the parties a user interacts with, because in protocol V0, all participants interact with the identity ecosystem using their real identities; ii) the adversary tracks a user's transaction patterns by linking the transactions that the user carries out with different SPs.

The following modifications to protocol V0 address those privacy concerns. **1) *Confidentiality***: Bank A encrypts the identity asset in step 7, using a key known to bank B. **2) *Anonymity*** of the parties whom a user interacts with: All the participants use a pseudonymous certificate issued by the CA, when interacting with the identity ecosystem. **3) *Unlinkability*** of the user's transactions: i) the participants use different pseudonymous certificates in executing step 2 and each round of identity asset transfer (i.e. steps 4 - 10); ii) bank A does not expose the CRH of the identity asset in plain text in step 2; instead it submits a commitment to the CRH of the identity asset. Otherwise, the identity asset consumers (e.g. bank B), can decrypt the identity asset received in step 9, compute its CRH and track the corresponding identity asset creation transaction (e.g. execution of step 2), in order to infer information such as when is the first time the user has consumed a similar service, etc. Note that the aforementioned mechanisms for enforcing unlinkability, specifically the one mentioned under 3.i, also imply anonymity. The pseudonymous identity certificates obtained by the participants from the CA, do not include any identifiable attributes, but the public key of a new RSA key pair. A pseudonym is considered to be the CRH of the public key of such key pair.

3.4 Challenges in preserving users' privacy and ensuring security of the protocol

In what follows we discuss how the aforementioned mechanisms for ensuring security and privacy properties conflict, which raise challenges in developing a privacy preserving and secure identity asset exchange protocol. We discuss such challenges w.r.t. three versions of the incrementally developed protocol, each of which is a result of incorporating privacy features one by one, into V0.

V1-*Confidentiality*: When the identity asset is encrypted, peers cannot verify its *correctness* simply by computing its CRH as in V0. The mechanisms for preserving *ownership assurance*, *counterfeit elimination* and *financial fairness* used in V0 are not affected though.

V2-*Confidentiality and Anonymity*: When the participants execute the protocol with pseudonyms, the *correctness* enforcement mechanism is not affected compared to V1. However, the mechanism to preserve *ownership assurance* in V0/V1 is affected, because there is the threat of a malicious identity consumer (e.g. bank B) sending a 'contract' which they want the identity asset provider and/or the user to sign, instead of a truly random nonce in step 4. Therefore, parties can not give away a signature on a challenge nonce, as a proof of ownership of the private key. There is no effect on the *counterfeit elimination* mechanism used in V0/V1, except that the owners of the identity asset should use the key pair related to their pseudonymous identity in steps 2, 6.b and 7. The *financial fairness* enforcement mechanism used in V0/V1 is affected, because pseudonymous bank B can intentionally skip step 10. Involving the CA to de-anonymize such identity consumers adds lot of overhead. Instead, this should be addressed by the protocol itself.

V3-*Confidentiality, Anonymity and Unlinkability*: It is more challenging to ensure *correctness* when unlinkability is enforced, because now even the CRH of the identity asset is not exposed in step 2. It is also more challenging to preserve *ownership assurance* than in V2, because now the two owners of the identity asset use different pseudonyms in step 2 and in each round of steps 4-10, in contrast to using a single pseudonym across all transactions as in V2. The hash table based *counterfeit elimination* mechanism is no longer sufficient now, because the hash of the identity asset is not exposed in step 2 and any two parties appearing with new pseudonyms can execute step 2, submitting a commitment to a CRH of any identity asset. Preserving *financial fairness* also needs improved mechanisms because we need to make sure that the underlying monetary payment system also preserves unlinkability; otherwise, parties can be de-anonymized via linkability in the payment system.

Table 1 summarizes how introducing the properties for preserving users' privacy into protocol V0, in an incremental manner, affects the mechanisms for achieving the identified security properties. As mentioned in , the mechanisms for enforcing unlinkability implies anonymity. Therefore, only two properties are mentioned in the heading of the third column under users' privacy, which corresponds to Version 3 of the protocol. A checkmark in a given cell indicates that the combination of privacy properties in the given column pose challenges to the mechanism used in the previous version of the protocol, for ensuring the security property in the given row.

Table 1: Effect of introducing the properties for preserving users' privacy, in an incremental manner, on the properties for ensuring security of the protocol.

Properties ensuring protocol security:	Properties ensuring users' privacy:		
	confidentiality	confidentiality + anonymity	confidentiality + unlinkability
counterfeit elimination	-	-	✓
correctness	✓	-	✓
ownership assurance	-	✓	✓
financial fairness	-	✓	✓

4 PROTOCOL DESIGN

In what follows, we present the design of the proposed protocol, named PrivIdEx, addressing the aforementioned challenges. As shown in Figure 2, PrivIdEx involves four parties: Identity Asset Provider (IAP), User, Identity Asset Consumer (IAC), and Blockchain (BC). PrivIdEx consists of four phases, each of which serves one or more specific purposes and groups together a set of relevant steps from the identity asset exchange flow shown in Figure 1. Phase 0 is executed only once and phases 1-3 are executed each time an identity asset is exchanged. Note that T represents the transactions posted to the BC by participants, by invoking different functions in the smart contract, W represents the validation steps executed by peers on the transactions and M represents the messages exchanged between two parties offline (i.e. without involvement of the BC). The purpose(s) of each phase is (are) described as follows, with examples from protocol V0.

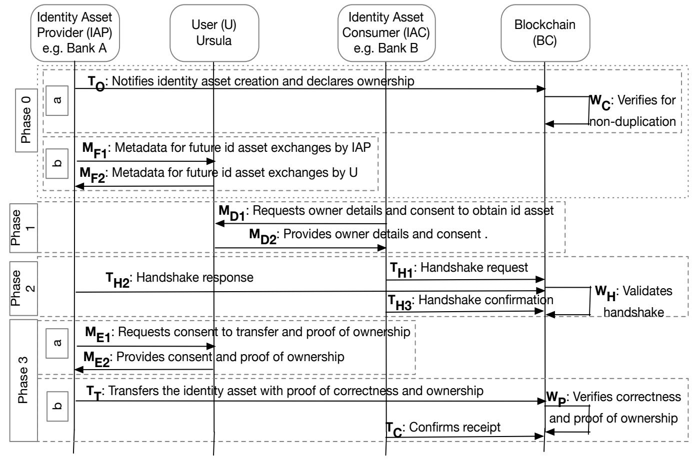

Figure 2: Overview of PrivIdEx design. T represents a transaction submitted to the BC by the participants. W represents a validation (of a transaction) performed by the peers. M represents a message exchanged by the participants offline (without involving the BC).

Phase 0 consists of two sub phases. Phase 0.a, executed between the IAP and the BC, serves for ownership declaration (T_O) by IAPs for newly created identity assets and verification by peers that such identity assets are not counterfeits. E.g. in protocol V0, the IAP sends the CRH of the public keys of the two owners and the CRH of the identity asset in T_O, which the peers verify in W_C using the hash table of ownership declarations maintained in the BC (see Section 3.2). Phase 0.b, executed between the IAP and the user, allows them to exchange meta data to be used in future identity asset exchange(s). For example, in protocol V0, the IAP sends to the user in M_{F1}, the IAP's contact information and the name of the identity asset created for the user, which the user stores in her records to be used in phase 1.

Phase 1 is executed between the IAC and the user. In M_{D1}, the IAC requests contact details of the IAP that has created the required identity asset (if any), and the user's consent for requesting such identity asset from the corresponding IAP. The meta data saved by the user in phase 0.b is queried to construct the user's response (M_{D2}).

Phase 2 consists of a three-way handshake between the IAC and the IAP over the decentralized identity ecosystem. The IAC initiates the handshake by submitting to the BC a message (T_{H1}) addressed to the IAP who owns the identity asset. T_{H1} includes the user's consent received in M_{D2} and is signed by the IAC. All the participants receive a notification about T_{H1} via the BC. The corresponding IAP verifies the user's consent and submits a response handshake message (T_{H2}). The IAC acknowledges T_{H2} by sending a confirmation (T_{H3}). The peers validate a handshake by verifying if T_{H2} and T_{H3} are associated with a corresponding T_{H1} and T_{H2}, respectively.

The handshake phase allows the IAP and the IAC to connect anonymously over the BC and to negotiate certain information pertaining to the identity asset exchange, which is carried out in Phase 3. For examples, in protocol V0, i) the IAC sends the random nonce in T_{H3} to be signed by the two owners during the transfer phase, for the proof of ownership, ii) if financial fairness is enforced by the identity management system, the IAP specifies in T_{H2}, the monetary value required for transferring the requested identity asset and the IAC sends in T_{H3}, a reference to a payment made to the IAP with that value. Note that in protocol V0 (i.e. when the participants appear with their real identity and we do not assume any threats to financial fairness), the payment can be made either before of after the identity asset is transferred (i.e. either in T_{H3} or in T_C respectively). However, in protocol V1 and beyond, when there is a threat to financial fairness from the parties appearing with pseudonyms, certain precautions should be taken in order to ensure financial fairness.

Phase 3 is where the actual identity transfer happens. In phase 3.a, executed between the IAP and the user, the IAP requests the user, via M_{E1}, to provide consent for transferring the identity asset and the proof of user's ownership of the identity asset. The user responds accordingly via M_{E2}. In phase 3.b, the IAP transfers the identity asset along with M_{E2} and the proof of IAP's ownership of the identity asset, via T_T. The peers verify in W_P, that T_T is associated with a valid handshake, and verify correctness and proofs of ownership. After receiving the notification about T_T from the BC, the IAC checks the transferred identity asset and posts to the BC a confirmation or complains about the receipt of the identity asset in T_C.

In what follows, we present how those different phases are utilized and enhanced to address the challenges for achieving the security properties of the three versions of the incremental design of PrivIdEx (see Section 3.4).

4.1 V1- Confidentiality Preserving Protocol

When confidentiality is enforced, the IAP and the IAC agree on a key K for encrypting the identity asset (A), by integrating the Diffie-Hellman key exchange protocol into the three-way handshake in phase 2. In T_T, the IAP submits the encrypted identity asset: $C = Enc_K(A)$. Due to encryption of the identity asset, *correctness* is the only security property that is more challenging to achieve in V1, compared to V0 (see Section 3.4). To prove correctness, the IAP submits in T_T, the transaction id of T_O associated with A and a zero-knowledge (ZK) proof (Φ_1), proving the knowledge of a satisfying assignment to the NP statement - **NS1**: *Given a cipher text C', a hash value a', I know the following secrets: an identity asset A' and a key K' s.t a' = CRH(A') and $C' = Enc_{K'}(A')$*. The IAP proves the satisfying assignment to NS1 with $A' = A$ and $K' = K$ as secret inputs, $C' = C$, and a' = the CRH of A submitted in the corresponding T_O, as the public inputs. If Φ_1 is verified successfully, the peers accept that T_T encrypts the same identity asset whose ownership was declared in the corresponding T_O.

4.2 V2 - Confidentiality and Anonymity Preserving Protocol

When anonymity is enforced: i) the IAP includes the two owners' pseudonyms in T_O instead of their real identities; ii) the IAP and the user records each other's pseudonym in phase 0.b; iii) the user sends to the IAC, via M_{D2}, the pseudonym of the IAP who created the identity asset; iv) the transactions posted to the BC are signed using the pseudonyms; v) when transaction notifications are received from the BC, each participant checks if the transaction messages are addressed to their pseudonym and responds accordingly. Due to the threats by the pseudonymous participants, *ownership assurance* and *financial fairness* are more challenging to achieve in V2 (see Section 3.4).

To prove ownership in V2, the user and the IAP create ZK proofs (Φ_{2U} and Φ_{2P} respectively) on the NP statement **NS2**: *Given a public key PK, a message M, I know a secret signature S, s.t. RSA_Sig_Verify(M, S, PK) = True*, where S = RSA_Sig(M, private key of PK). RSA_Sig outputs a signature, given a RSA private key, and a message M; RSA_Sig_Verify outputs True iff S is the correct signature for M, using the private key associated with PK. The user and the IAP prove satisfying assignments to NS2 with PK = the public key of the pseudonym included in the T_O associated with the identity asset being transferred, M = the random nonce sent by the IAC via T_{H1}, as public inputs, and S = the signature created on such nonce with the private key associated with the pseudonym, as the secret input. Proving the knowledge of S on the nonce sent by the IAC, without revealing S, avoids the risk of giving a signature on a potential 'contract' (see Section 3.4).

Φ_{2U} and Φ_{2P} are integrated in to the protocol design as follows. The IAP sends to the user, via M_{E1}, the nonce it received from the IAC in T_{H1}. The user sends Φ_{2U} via M_{E2}. Then the IAP creates Φ_1 (see Section 4.1) and Φ_{2P}, and sends T_T to the BC along with the transaction id of T_O associated with A, Φ_1, Φ_{2U} and Φ_{2P}. The peers validate T_T, by verifying correctness and ownership assurance via the ZK-proofs provided in T_T.

To ensure *financial fairness* in V2, the handshake phase is used as follows. The IAP informs the IAC about the required monetary compensation for transferring the identity asset via T_{H2},. If the IAC agrees to pay, it includes in T_{H3} a reference to a bitcoin payment made with a locking condition such as: *'the IAP can can unlock the payment only by using either of these: i) a T_C submitted by the IAC, indicating a successful receipt of the identity asset; ii) a successful W_P by the peers, if a T_C is not submitted after time 't' since the time of T_T'*. Such a locked payment [2, 24] made during the handshake phase prevents a pseudonymous IAC from skipping the required payment. The IAP can redeem the payment only if it transfers the correct identity asset, ensuring financial fairness to both parties.

4.3 V3 - Confidentiality, Anonymity and Unlinkability Preserving Protocol

As per Section 3.4, it is more challenging to achieve all four security properties in V3, compared to V2, due to the enforcement of unlinkability property. In what follows we describe how each phase of the protocol is enhanced to address those challenges. Prior to phase 0.a, the user creates a commitment to the public key of her real identity (U_{pk}): $C_U = commit(U_{PK}, r_u)$ and sends C_U to the IAP. The IAP creates a commitment to the public key of its real identity (P_{pk}): $C_P = commit(P_{PK}, r_p)$ and a commitment to a - the CRH of the newly created identity asset: $C_a = commit(a, r_a)$. The IAP sends T_O to the BC, including the ownership declaration $O = C_U|C_P|C_a$, where | denotes concatenation, signed by a new pseudonym key.

The basic idea of *counterfeit elimination* in V3 is as follows. Let $B = \{a_1, a_2, ..., a_n\}$ be the set of CRH values of the identity assets associated with all the previous valid executions of T_O. This set is represented by a unique polynomial P of degree n, that has a_1, a_2,..., a_n as its roots. The polynomial P is represented as $P(x) = \prod_{i=1}^{n}(x - a_i)$. Let P_i be the i^{th} coefficient of P, for $i = 0, 1, ..., n$. P is initialized as $P(x) = 1$, and its degree increases with each new valid T_O. Hence, at any given time, if a number n of valid identity assets have been created in the identity ecosystem, then there is a number $n + 1$ of P_is. If the evaluation of $P(x)$ with $x = a$ results in zero (i.e. $P(a) = \sum_{i=0}^{n} P_i.a^i = 0$), it implies that the identity asset, whose CRH value is a, is a duplicate of an existing one. In order to preserve unlinkability, peers should only learn if $P(a) = 0$ or not, and nothing else. Therefore, the set of P_is are secretly encoded before being stored in the BC and $P(a)$ is computed in the encoded domain before being revealed to peers.

To prove that the created identity asset is not a counterfeit, the IAP submits T_O to the BC, including O and the following four items:
(I1) The result of computing $P(a)$ in the encoded domain, which is denoted by l, i.e. $l = En(P(a))$.
(I2) A ZK-proof Φ_3, proving that l is correctly computed and that the same a is used to compute both C_a (in O) and l.
(I3) The secretly encoded set of coefficients P'_i of the updated polynomial P', which has a as one of its roots (i.e. $P'(x) = P(x).(x - a)$, and therefore, P'_0 = -$a.P_0$, $P'_i = P_{i-1}$ - $a.P_i$, for $i = 1, ..., n$, and P'_{n+1} = 1).
(I4) A ZK-proof Φ_4, proving that I3 is correctly computed, using the same a used in C_a.

Details of the mechanism for counterfeit elimination are as follows. Let E be an additive threshold homomorphic encryption scheme, whose public key is known to everyone, but the secret key is distributed among the peers s.t. a group of at least $t + 1$ of them are required to perform decryption. E is instantiated with the Elgamal encryption scheme over a group G of order q. The public key $h = g^s$, where g a generator in G and s is the private key, which is distributed among the threshold peers. E is initialized with a distributed key generation protocol [14]. The encryption of an element $g' \in G$ is defined as: $E(g') = (g^k, h^k g')$, where $k \in Z_q$ is randomly chosen. An encoding scheme En to encode elements in Z_q is defined based on E as follows. $En(z) = (g^k, h^k g^z)$, where $z \in Z_q$. In fact, $En(z) = E(g^z)$. $En(\cdot)$ is an additively homomorphic encoding of z which allows us to carry out computation on polynomials whose coefficients are presented in encoded form. Moreover, while the value z cannot be recovered in general from $En(z)$, for our purposes we only need to be able to decide for a given $En(z)$ whether z is zero or not. In addition, the secrecy of z is guaranteed by the underlying Elgamal encryption scheme $E(\cdot)$.

Let the set of encoded P_is stored in the BC be $S_n = \{En(P_i) = E(g^{P_i})$, for $i = 0, 1, ..., n\}$. Details of how the IAP computes I1-I4 are as follows.

(I1) Compute $l = En(P(a))$, given the set S_n:
- Compute a fresh encoding of zero as: $e_0 = En(0) = (g^k, h^k)$ for random $k \in Z_q$, in order to randomize the encoding of l.
- Then l is computed as follows:

$$l = (\prod_{i=0}^{n}(En(P_i)^{a^i})).e_0 \tag{1}$$

(I2) Create Φ_3 on the NP statement - **NS3**: *Given a commitment C_{IA}, an encoding L, and a set S of encoded coefficients of a polynomial P, I know secrets: r, a', and k' s.t. C_{IA} = commit(a', r), $L = En(P(a') + 0) = En(P(a')).e_0$ and $e_0 = (g^{k'}, h^{k'})$.*

The IAP proves a satisfying assignment to NS3 with $C_{IA} = C_a$ in O, $L = l$ in (I1), and $S = S_n$ stored in the BC at the time of submitting T_O, as public inputs; and $r = r_a$ and $a' = a$ used in C_a and $k' = k$ used in e_0, as secret inputs.

(I3) Let the set of P'_is in the encoded domain be $S_{n+1} = \{En(P'_i)$ for $i = 0, 1, ..., n + 1\}$. S_{n+1} is computed as follows.
- Compute a fresh encoding of zero as: $e_0 = En(0) = (g^{k_0}, h^{k_0})$ for random $k_0 \in Z_q$, and $En(P'_0) = En((-a).P_0 + 0) = En(P_0)^{-a}.e_0$.
- For $i = \{1, .., n\}$, choose $k_i \in Z_q$ randomly and compute: $e_0^i = En(0)$, and $En(P'_i) = En(P_{i-1} + (-a).P_i + 0) = En(P_{i-1}).En(P_i)^{-a}.e_0^i$.
- Choose $k_1 \in Z_q$ randomly and compute $En(P_{n+1}) = En(1)$.

(I4) Create Φ_4 on the NP statement - **NS4**: *Given a commitment C_{IA}, a set S of encoded coefficients of a polynomial P^n and a set S' of the updated polynomial P^{n+1}, I know secrets: r, a', k'_0, k'_1 and k'_i for $i = 1, ..., n$ s.t.:*
- *C_{IA} = commit(a', r),*
- *$En(P_0^{n+1}) = En(-a'.P_0^n + 0) = En(P_0^n)^{-a'}.e_0$ and $e_0 = (g^{k'_0}, h^{k'_0})$,*
- *for $i = 1, .., n$, $En(P_i^{n+1}) = En(P_{i-1}^n).En(P_i^n)^{-a'}.e_0^i$ and $e_0^i = (g^{k'_i}, h^{k'_i})$,*
- *$En(P_{n+1}^{n+1}) = (g^{k'_1}, h^{k'_1}.g^1)$.*

The IAP proves a satisfying assignment to NS4 with $C_{IA} = C_a$, $S' = S_{n+1}$ computed in I3 above, $S = S_n$ stored in the BC, as public inputs, and with $a' = a$, $r = r_a$ used in C_a, $k'_0 = k_0$ used in $En(P'_0)$, $k'_1 = k_1$ used in $En(P'_{n+1})$ and $k'_i = k_i$ used in $En(P'_i)$ for $i = 1, .., n$, of I3, as secret inputs.

Once the IAP submits T_0 along with O and I1-I4, the validation W_C (see Figure 2) by peers is executed as follows. If the ZK-proof Φ_3 is successfully verified, each peer i executes the following steps to randomize l: choose $r_i \in Z_q$ randomly, compute $l_i = l^{r_i}$ and broadcast l_i to all the other peers together with a ZK-proof Ω_1^i, proving that the peer knows the value r_i. Then each peer computes l' as the sum of the cipher texts received from all the peers, i.e. $l' = \prod_{i=1}^{m}(l^{r_i}) = E(g^{P(a)(\sum_{i=1}^{m}(r_i)})$, where m is the number of peers. Then peers collectively decode l'. Note that we choose to randomize l and then decode l', instead of just decoding l, due to a potential collusion attack by an IAP and a peer to check if a given value matches the CRH of an already created identity asset.

If l' does not decode to an encoding of zero (i.e. l' does not decrypt to 1), peers verify the ZK-proof Φ_4. If Φ_4 is successfully verified, peers accept T_O as a valid ownership declaration, which is not associated with a counterfeit, and S_{n+1} as the encoded set of coefficients of the updated polynomial to be stored in the BC. Accordingly, protocol version V3 of PrivIdEx preserves counterfeit elimination, without revealing the CRH values of the identity assets associated with transactions T_O, thereby preserving unlinkability across T_O and T_T associated with the same identity asset. To enable proof of ownership and correctness, while preserving unlinkability, after successful verification of T_O, the peers add the CRH of the ownership declaration O, i.e. f = CRH(O), as a leaf in the Merkle hash tree (MHT) data structure stored in the BC. This marks the end of phase 0.a for protocol V3.

The basic idea of proving ownership during identity asset transfer (phase 3.b) is to prove that the user and the IAP know a *path P* in the MHT from a leaf f, which contains the CRH of a valid ownership declaration O, to the root RT and that the user and the IAP own the private keys associated with the public keys committed in C_U and C_P of such O, respectively. To prove correctness, the IAP proves a similar statement, that is, the IAP knows a path P from f, which contains O with a commitment C_a to a CRH value that matches the CRH value of the identity asset being transferred, to RT. Note that both the owners of the identity asset should prove in zero-knowledge, the knowledge of the same path P in the MHT.

During the identity asset transfer phase, when the IAP requests the user's proof of ownership to the identity asset via M_{E1}, the user creates a ZK-proof Φ_5 proving her ownership and sends it to the IAP via M_{E2}. The IAP then creates a ZK-proof Φ_6 proving its ownership and correctness and transfers the encrypted identity asset along with Φ_5 and Φ_6 via T_T. Peers verify Φ_5 and Φ_6 in W_P and confirm that T_T preserves correctness and ownership assurance.

In order to ensure financial fairness while preserving unlinkability, protocol V3 should integrate an anonymous and an unlinkable payment system such as Zerocash [4], which also enables making locked payments described under protocol V2.

5 IMPLEMENTATION AND EXPERIMENTS

In what follows, we present the details of the implementation and experiments on the main building blocks of PrivIdEx. Our goals are two folds: i) understanding the challenges and feasibility of the implementation of some of the most complex building blocks, e.g. ZK-proofs for the NP statements used in PrivIdEx; ii) evaluating circuit size, execution times and storage requirements of ZK-proofs for the NP statements. Experiments were run in a desktop machine running Ubuntu 18.04.1 LTS with 16GB memory and Intel i7-4790 CPU @ 3.6GHz.

We used the ZK-SNARK construction (see Section 2.2) to prove/verify satisfying assignments to the NP statements listed in Section 4. ZK-proofs for the NP statements were created using ZK-SNARKs, following the five steps process listed in Section 2.2. First, the cryptographic primitives in the NP statements were instantiated with specific algorithms. Then the circuits for the NP statements were designed and implemented using the Jsnark [17] framework. The Jsnark framework allows one to write circuits in a format compatible with the ZK-SNARK compilers and provides building blocks called 'gadgets' for designing circuits. In order to compile the circuit into a QAP and to prove/verify in zero-knowledge the satisfiability of the assignment given by the prover, Jsnark interfaces with Libsnark [23] - the widely used library implementing the ZK-SNARK construction. The challenges in this process include, but not limited to: i) gadgets for certain cryptographic primitives, such as Elgamal encryption used in NS4, are not yet available in Jsnark; ii) as different existing Jsnark gadgets accept inputs in different formats, we had to standardize the input formats of these gadgets before wiring them together to form the required circuit.

In circuit 1 (see Figure 3) built for the NP statement NS1 used in V1 of PrivIdEx, the cryptographic hash (CRH) algorithm is instantiated with the widely used SHA-256 and the symmetric encryption algorithm is instantiated with SPECK128 [11], due to its light weight properties. The SPECK128 gadget is wrapped with the gadget implementing symmetric encryption in CBC mode. We evaluated the circuit size (see Table 2), running time (see Figure 4), and storage requirements (see Figure 5) associated with the three algorithms of ZK-SNARKs for circuit 1, by varying the size of the identity asset (A'). Increase in the size of A' increases the size of the proving key and the circuit (i.e., number of constraints in the circuit), which in turn affects the running times of the key generator, which takes the circuit as inputs, and the prover, which takes both the circuit and the proving key as inputs. However, the increase in running time of the prover is much less than that of the key generator, which is good because the prover is run each time an identity asset is exchanged, whereas the key generator is run only at system setup. Proof size, verification key size, and verifier running time are constant irrespective of the size of the secret input A'. Note that for the scope of this paper, we assume a fixed size for the identity assets (those with shorter sizes can use padding) exchanged in a given deployment of PrivIdEx, because it is an overhead to deploy multiple circuits for different sizes. In a real deployment, we can have three fixed sizes as small, medium and large, and three different categories of circuits can be created during the bootstrap. The zero-knowledge proofs for a given identity asset can be created using the circuits in the nearest upper size category, after (minimally) padding the identity asset.

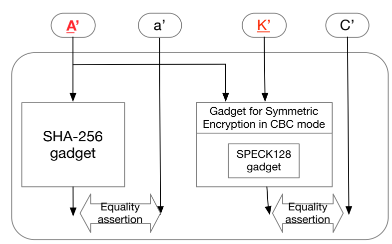

Figure 3: Circuit 1 built for NS1: *Given a cipher text C', a CRH value a', I know the following secrets: an identity asset A' and a key K' s.t a' = CRH(A') and $C' = Enc_{K'}(A')$.*

Size of (A')	64 bytes	128 bytes	256 bytes	512 bytes	1024 bytes
Number of constraints	74,429	126,505	230,675	451,210	855,569

Table 2: Circuit size vs the size of the identity asset (A'), for Circuit 1.

In circuit 2 (see Figure 6) built for the NP Statement NS2 used in V2 of PrivIdEx, the signature S is the only secret input, which is created by the provers (i.e. the IAP and the user) locally (i.e. outside of the circuit). Although PK theoretically consists of both RSA modulus and public exponent, only the RSA modulus is given as input PK, and the public exponent is set to a hard coded constant, according to the implementation details of the RSA algorithm [6]. In order to decide the size of the nonce M with optimal trade-off between security and performance of ZK-SNARKs for circuit 2, we evaluated the performance by varying the size of M. However, as shown in Table 3, there was a negligible impact on the performance

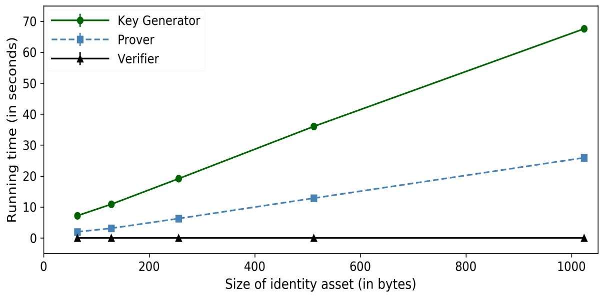

Figure 4: Running time vs the size of the identity asset (A'), for Circuit 1.

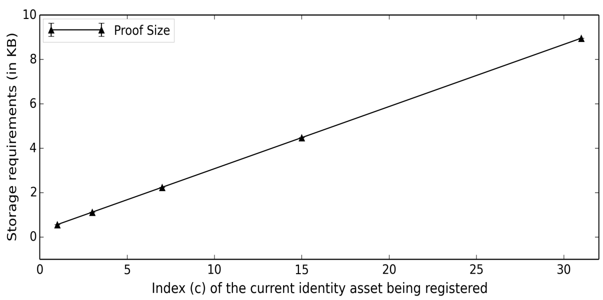

Figure 5: Storage size vs the size of the identity asset (A'), for Circuit 1.

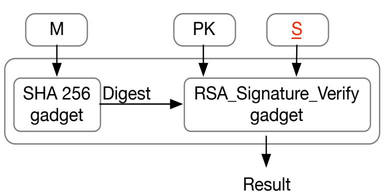

Figure 6: Circuit 2 for NS2: *Given a public key PK, a message M, I know a secret signature S, s.t RSA_Sig_Verify(M, S, PK) = True,* where S = RSA_Sig(M, private key of PK).

when the size of the public input M was doubled. Therefore, we decided to use 128 bits as the size of the nonce.

	64 bits	128 bits
Circuit size (number of constraints)	119,146	119,344
Key gen running time	12.8659(s)	12.8729(s)
Proving key size	32,903(KB)	32,938(KB)
Verification key size	3.2856(KB)	3.597(KB)
Prover running time	3.38(s)	3.3827(s)
Proof size	0.28(KB)	0.28(KB)
Verifier running time	0.0045(s)	0.0045(s)

Table 3: Performance numbers vs the size of the nonce (M), for Circuit 2.

A summary of the insights derived from the above experiments are as follows: 1) Increase in the size of *secret* inputs increases circuit size and prover key size, thereby increasing the running times of the key generator (which is run only once for the entire system lifetime) and the prover (which is run only once at ownership declaration and at each round of exchange of an identity asset). 2) Increase in the size of public inputs has negligible impact on the performance of ZK-SNARKS associated with a given circuit. 3) Proof size, verification key size, and verifier running time are negligibly affected (if at all) by the circuit complexity (e.g. circuit 1 and circuit 2 use different gadgets with varying complexity), size of secret inputs (e.g. experiments on circuit 1,) and size of public inputs (e.g. experiments on circuit 2). This makes using ZK-proofs based on ZK-SNARKs very suitable for use in PrivIdEx to ensure privacy and security properties of identity asset exchange in a decentralized identity ecosystem backed by a BC network, where multiple peers may run the verification algorithms associated with W_C and W_P (see Figure 2). We refer the reader to the Appendix 4.3 for the details of the remaining circuits.

6 SECURITY AND PRIVACY PROOFS

In what follows, we prove that PrivIdEx (V3 - which addresses all three privacy requirements - see Section 4.3) protects against the threats mentioned in the threat models for user privacy (see Section 3.3) and protocol security (see Section 3.2).

The following lemma establishes that an adversary (referred to as Adv1), whose goal is to compromise the user's privacy (see Section 3.3), does not learn any information on the identity asset and the identity of the parties the user interacts with. Based on the information that Adv1 learn from the protocol transcripts of PrivIdEx, Adv1 cannot link different transactions of the same user.

Lemma 6.1. *PrivIdEx (V3) preserves confidentiality of the user's identity asset and anonymity and unlinkability of the user's transactions against Adv1.*

Proof (informal): Based on the security of the Diffie-Hellman key exchange used to establish a key between the IAP and the IAC, during the handshake in phase 2 of the protocol, Adv1 cannot learn the key used to encrypt the identity asset in T_T of phase 3.b (see Figure 2). Hence, *confidentiality* of the identity asset of the user is preserved against Adv1.

Due to the computationally hiding property of the underlying commitment scheme, the actual identities of the IAP and the user are not revealed to Adv1 via commitments: C_P and C_U included in the ownership declaration O of T_O in phase 0.b. Therefore, Adv1 does not learn the identity of the IAP who creates an identity asset for a user as well as the identity of the user for whom the identity asset is created, during phase 0.b. All the transactions posted to the BC include senders' and intended recipients' pseudonyms. A pseudonym P of a participant, i.e. P = CRH(public key in the pseudonymous certificate) is indistinguishable from a random string. Therefore, Adv1, which does not collude with the CA, does not learn the identity of the parties interacting via the BC. Due to the zero-knowledge property of ZK-SNARKs, the identity of which the ownership is proved in Φ_5 and Φ_6 is not revealed to Adv1 in T_T. Anonymity of the underlying payment scheme, which is used to pay any required monetary compensation, ensures that Adv1 does not learn the identity of the IAP or the IAC via the associated payment transactions. Therefore, throughout the protocol execution, *anonymity* of the participants is preserved from Adv1.

The different pieces of information involved in the protocol execution that Adv1 can use to link different transactions of the same user, are as follows: i) identity (i.e. pseudonyms) of the parties; ii) the CRH of the identity asset; iii) cipher text encrypting the identity asset; iv) any payment transactions created to pay monetary compensations for the identity assets. Since new pseudonyms are used by the participants for the execution of each round of the identity asset exchange, Adv1 cannot link such transactions via pseudonyms. Due to the computationally hiding property of the commitment scheme, the CRH of the identity asset is not revealed to Adv1 via commitment: C_h in O of T_O. Due to the encoding scheme En not allowing one to decode the encoded values in l and S_{n+1}, the CRH of the identity asset is not revealed to Adv1 via any of I1-I4 submitted to the BC via T_O. Due to zero-knowledge property of ZK-SNARKs, the CRH of the identity asset, which is used to prove correctness in Φ_6, is not revealed to Adv1 in T_T. Hence Adv1 cannot link transactions via the CRH of the identity asset. Due to semantic security of the underlying symmetric encryption scheme, Adv1 cannot link the exchange transactions encrypting the same identity asset. Unlinkability of the underlying payment scheme ensures that Adv1 cannot link the identity asset exchange transactions via the associated payment transactions. Therefore, Adv1 does not learn any information helping to link transactions of the same user, hence, unlinkability is preserved against Adv1. ■

The following lemma establishes that an adversary (referred to as Adv2), whose goal is to compromise security of the identity asset exchange protocol (see Section 3.2), cannot create a counterfeit of an existing identity asset, transfer a fake identity asset that has not been legitimately created in the identity ecosystem, and claim false ownership to an identity asset.

LEMMA 6.2. *PrivIdEx (V3) preserves correctness, ownership assurance and counterfeit elimination against Adv2.*

Proof (informal): Due to the additive homomorphic property of the encoding scheme based on the Elgamal encryption scheme, and the soundness property of ZK-SNARKs (which is based on the knowledge of coefficient assumption) used to create the ZK-proof Φ_3, Adv2 cannot submit an ownership declaration in T_O for an identity asset which is a counterfeit, without failing the validation W_C run by the peers in phase 0.b. Again, due to the soundness of ZK-SNARKs, used to create Φ_5 and Φ_6, we have that: i) an IAP controlled by Adv2 cannot transfer a fake identity asset because the IAP cannot provide commitments to a valid Merkle hash tree path of an ownership declaration O, which contains a commitment to a CRH value that matches the CRH of the identity asset being transferred, in the satisfying assignment to $\Phi 6$; ii) a user and an IAP controlled by Adv2 cannot claim false ownership because they cannot provide commitments to a valid Merkle hash tree path of an O, which contains commitments to the CRH of public keys for which they own the private keys, in the satisfying assignments to Φ_5 and $\Phi 6$. ■

THEOREM 6.3. *PrivIdEx preserves the identified privacy properties against Adv1 and the security properties against Adv2.*

Theorem 6.3 follows from lemma 6.1 and Lemma 6.2.

7 RELATED WORK

Identity management research has a rich history. Here we focus on the proposals focusing on exchanging users' identity information between SPs. Next we discuss approaches for privacy enhancing techniques for BC applications and show that such approaches alone cannot address the problem that we focus on. OpenID Exchange (OIX) [12] and OpenID Connect [22] are industry standards which address some form of identity exchange. Such protocols, however, have one central IDP from whom other SPs obtain identity information of a user, and do not address a user's transactional privacy requirements. Identity Mixer [9] is an anonymous credential system which enables users to authenticate to SPs in an unconditionally unlinkable manner, while selectively disclosing users' identity information. Identity Mixer also involves a central IDP from which the user obtains identity tokens; the IDP is known to the SPs while the SPs are not known to the IDP. The USA and UK governments have developed nation-scale identity management systems which enable government identity consumers to obtain users' identity information from third party identity providers, where consumers and providers are anonymous to each other, in order to preserve users' privacy. However, such systems introduce a government managed broker to mediate the identity exchange transactions, which learns the identity of the two exchanging parties, and hence, can track the users' transactions. More recently, decentralized identity management systems have been proposed that leverage BC technology to avoid centralized parties managing users' identity [26]. However, such systems do not address all the privacy requirements that we consider.

Zerocash [4] enables a sender to transfer bitcoins to a recipient in an anonymous and unlinkable manner. PrivIdEx differs from Zerocash in multile respects, including: i) Zerocash prevents double spending of bitcoins whereas PrivIdEx enables transferring the same identity asset as many times as needed by the legitimate owners to different consumers; ii) there is only one anonymous owner for bitcoins at a given time, whereas there are two owners for an identity asset. Hawk [18] is a framework for privacy preserving smart contracts. Hawk alone does not address all the privacy and security requirements of a given use case, such as the one we focus on, which involves multiple phases and repeating interactions among the participants, based on the same identity asset. Zero Knowledge Asset Transfer (ZKAT) [1] by Heperledger Fabric is based on the unspent transaction output (UTXO) model of bitcoin. Hence, it supports exchange of monetary transactions which cannot be double spent, which is different from our use case. Therefore, ZKAT alone is not sufficient to enable privacy preserving and secure identity asset exchange.

8 CONCLUSION

We proposed PrivIdEx - a privacy preserving and secure protocol for identity asset exchange over a decentralized identity ecosystem backed by a permissioned BC network. PrivIdEx enables different SPs that a user interacts with to re-use the identity assets created for the user, eliminating the cost of repeated identity verification and due diligence processes, without having to worry about privacy and security concerns in doing so. Analysis of the threat model, protocol design and implementation and experiments are presented in an *incremental* approach to help readers understand the specific challenges posed when achieving each of the identified privacy properties and the mechanisms developed to address them, which also helps in selectively enabling those properties as required by a given identity ecosystem.

One potential future extension of PrivIdEx is to integrate it with the Identity Mixer based CA in Hyperledger Fabric [1] BC netowork to achieve unlinkability against collusions between the CA and an actor in the BC, so that we can eliminate the third assumption mentioned in Section 3.1. Other relevant future work is to generalize PrivIdEx to facilitate privacy preserving and secure exchange of any confidential digital asset with multiple owners, such as song lyrics, music, write-ups, e-books, etc., which has not yet been addressed by the existing digital asset exchanging platforms.

Acknowledgement

We thank Ahmed Kosba for clarifications about jsnark and Fabrice Benhamouda for discussions regarding the project. This work is supported by an IBM PhD fellowship award.

REFERENCES

[1] E. Androulaki, S. Cocco, and C. Ferris. 2018. Private and confidential transactions with Hyperledger Fabric. https://developer.ibm.com/tutorials/cl-blockchain-private-confidential-transactions-hyperledger-fabric-zero-knowledge-proof/ Accessed: 1-Nov-2018.
[2] M. Andrychowicz, S. Dziembowski, and D. Malinowski. 2014. Secure Multiparty Computations on Bitcoin. In *IEEE Symposium on Security and Privacy*.
[3] E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev. 2018. Scalable, transparent, and post-quantum secure computational integrity. In *Cryptology ePrint Archive: Listing for 2018*.
[4] E. Ben-Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza. 2014. Zerocash: Decentralized Anonymous Payments from Bitcoin. In *IEEE Symposium on Security and Privacy*.
[5] David Birch. 2016. Putting identity on the blockchain. http://www.chyp.com/putting-identity-on-the-blockchain-part-1-find-a-problem/.
[6] D. Boneh. 1998. Twenty Years of Attacks on the RSA Cryptosystem. https://crypto.stanford.edu/%7Edabo/pubs/papers/RSA-survey.pdf Accessed: 22-Sept-2018.
[7] Luis T.A.N. Brandao, N. Christin, G. Danezis, and Anonymous. 2015. Toward Mending Two Nation-Scale Brokered Identification Systems. In *Proceedings on Privacy Enhancing Technologies*.
[8] B. Bunz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell. 2018. Zerocash: Decentralized Anonymous Payments from Bitcoin. In *IEEE Symposium on Security and Privacy*.
[9] J. Camenisch and A. Lysyanskaya. 2001. An Efficient System for Non-Transferable Anonymous Credentials with Optional Anonymity Revocation. In *Proceedings of EUROCRYPT '01*. 93–118.
[10] A. Kosba et. al. 2015. CoC0: A Framework for Building Composable Zero-Knowledge Proofs. https://eprint.iacr.org/2015/1093.pdf
[11] R. Beaulieu et. al. 2013. The Simon and Speck Families of Lightweight Block Ciphers. https://eprint.iacr.org/2013/404.pdf Accessed: 22-Sept-2018.
[12] Open Identity Exchange. [n. d.]. OIX - Open Identity Exchange. https://www.openidentityexchange.org/ Accessed: 22-Nov-2017.
[13] Hyperledger Fabric. 2018. A Blockchain Platform for the Enterprise. https://hyperledger-fabric.readthedocs.io/en/release-1.3/ Accessed: 16-Oct-2018.
[14] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. 2007. Secure Distributed Key Generation Protocol.. In *J Cryptology*.
[15] R. Gennaro, C. Gentry B. Parno, , and M. Raykova. 2013. Quadratic Span Programs and Succinct NIZKs without PCPs. In *EUROCRYPT*.
[16] GOV.UK. 2018. Introducing GOV.UK Verify. https://www.gov.uk/government/publications/introducing-govuk-verify/introducing-govuk-verify Accessed: 22-Sept-2018.
[17] Ahmed Kosba. 2017. jsnark. https://github.com/akosba/jsnark Accessed: 22-Nov-2017.
[18] A. Kosba, A. Miller, and E. Shi. 2014. Hawk: The Blockchain Model of Cryptography and Privacy-Preserving Smart Contracts.. In *IEEE Symposium on Security and Privacy*.
[19] S. Nakamoto. 2008. Bitcoin: A Peer-to-Peer Electronic Cash System.
[20] Nat. 2010. Is Expressing Levels Enough for LOA2+? https://nat.sakimura.org/2010/09/03/is-expressing-levels-enough-for-loa2/ Accessed: 22-Sept-2018.
[21] United States Postal Office. 2014. FCCX Briefing. https://csrc.nist.gov/csrc/media/events/ispab-june-2014-meeting/documents/ispab_jun2014_fccx-briefing_glair.pdf Accessed: 22-Sept-2018.
[22] OpenID. 2017. Welcome to OpenID Connect. http://openid.net/connect/ Accessed: 22-Nov-2017.
[23] scipr lab. 2017. C++ library for zkSNARKs. https://github.com/scipr-lab/libsnark Accessed: 22-Nov-2017.
[24] Prabath Siriwardena. 2017. A Deeper Look Into Bitcoin Internals. https://medium.facilelogin.com/pay-with-bitcoin-to-play-with-a-fidget-spinner-86b7b43414c0 Accessed: 22-Sept-2018.
[25] Prabath Siriwardena. 2017. Identity on Blockchain (Part I). https://medium.facilelogin.com/identity-on-blockchain-part-i-a59d7abe75c0 Accessed: 22-Sept-2018.
[26] sovrin. 2017. Identity For All. https://sovrin.org/ Accessed: 22-Nov-2017.
[27] European Union. 2016. General Data Protection Regulation. https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32016R0679 Accessed: 22-Sept-2018.
[28] Zooko Wilcox. 2016. The Design of the Ceremony. https://electriccoin.co/blog/the-design-of-the-ceremony/

9 CIRCUITS FOR ZK-PROOFS INVOLVED IN THE VERSION 3 OF THE PROTOCOL

9.1 Circuits required for the ZK-Proofs in the identity asset registration phase

As explained in Section 4.3, in the identity asset registration phase, the IAP needs to prove to the peers two things: i) the evaluation of the polynomial $P(x)$ on the cryptographic hash value (a) of the created identity asset is computed correctly in the encoded domain (i.e. $En(P(a))$ is computed correctly), using the set of encoded coefficients of the polynomial stored in the blockchain (i.e. S_n). ii) the new set of coefficients for the updated polynomial P' (s.t. P' has 'a' as one of its roots), is computed correctly in the encoded domain. In other words, the IAP has to prove the two NP statements: **NS3** and **NS4** (see Section 4.3) respectively.

The encoding scheme that we have used to compute $P(a)$ and the coefficients of the updated polynomial in the encoded domain is based on Elgamal encryption (i.e. Elgamal encryption in the exponent).

Designing the circuit for NS3: Public parameter generation of ZK-SNARKs (e.g. generation of the prover key and verifier key - see Section 2.2) for a given NP statement used in an application should be performed at the bootstrapping of the system (e.g. see the public parameter generation ceremony of ZCash [28]). Therefore, the trusted party who runs the ***Generator()*** algorithm of ZK-SNARKs has two options for creating the circuit(s) and the associated public parameters for NS3, as follows:

Approach 1: create $m-1$ number of circuits, each allowing to prove NS3 for the i^{th} registering asset where $i \in \{2, 3, ..., m\}$.

Approach 2: create one circuit, which allows to prove NS3 for registering of an identity asset at any index i where $i \in \{2, 3, ..., m\}$, and m is the maximum number of identity assets allowed to be registered in the system.

Remark: Note that NS3 is not needed to be proved for the first asset registered in the system.

The trusted party who bootstraps the system has to run *Generator()* algorithm of ZK-SNARKs for $m-1$ number times w.r.t approach 1 whereas the trusted party has to run *Generator()* algorithm only once w.r.t approach 2. The work done at the bootstrapping stage is considered a one-time cost which does not impact the end user experience. On the other hand, approach 1 makes different IAPs to bear different costs in creating ZK-proofs for NS3 (i.e. in running the *Prover()* algorithm of ZK-SNARKs), based on the index (i) at which the identity asset is being registered in the system. In other words, the IAP who registers the 2^{nd} identity asset has to bear a lower cost than the IAP who registers the m^{th} identity asset, because the former IAP only has to evaluate a polynomial of degree 1 whereas the latter IAP has to evaluate a polynomial of degree $m-1$, in the encoded domain. In contrast, approach 2 makes all IAPs to bear almost similar (high) costs in running the *Prover()* algorithm, because all IAPs have to evaluate a polynomial of degree $m-1$ in the encoded domain. In this case, encodings of zero are used as the encodings of any coefficient a_j s.t. $i < j < m$. Although *Generator()* and *Prover()* algorithms of ZK-SNARKs will incur different costs in approach 1 and 2, *Verifier()* algorithm will incur similar costs in the two approaches, according to the properties of ZK-SNARKs.

The original NS3 in Section 4.3 is re-written to suite the aforementioned two approaches, as follows:

NS3 for approach 1: *Given a commitment C_{IA}, an encoding L, and a set S of encoded coefficients of a polynomial P of order c, where c = the number of identity assets that are already registered in the system, I know secrets: r, a, and k s.t. $C_{IA} = commit(a, r)$, $L = En(P(a) + 0) = (\prod_{i=0}^{c}(En(P_i)^{a^i})).e_0$ and $e_0 = (g^k, h^k)$.*
Set S is initialized at the bootstrapping of the system as follows: $S = \{En(P_0) = En(1)\}$.

NS3 for approach 2: *Given a commitment C_{IA}, an encoding L, and a set S of encoded coefficients of a polynomial P of order $m-1$, where m = the maximum number of identity assets allowed to be registered in the system, I know secrets: r, a, and k s.t. $C_{IA} = commit(a, r)$, $L = En(P(a) + 0) = (\prod_{i=0}^{m-1}(En(P_i)^{a^i})).e_0$ and $e_0 = (g^k, h^k)$.*
Set S is initialized at the bootstrapping of the system as follows: $S = \{$for $i = \{1, ..., m-1\}$, $En(P_i) = En(0)$, $En(P_0) = En(1)\}$. Note that $En(P_m)$ does not need to be stored in set S because no IAP needs $En(P_m)$ for the computations of L.

Irrespective of which approach is used for NS3, an IAP who successfully registers an identity asset, also updates the encoded coefficients of the polynomial at indices from 0 to $c+1$, such that the cryptographic hash of the currently registering identity asset becomes a root of the updated polynomial, and proves the correctness of the updated encoded coefficients using NS4.

Remark: The IAP who is registering the mth identity asset, does not need to perform this step, as the updated coefficients of the polynomial of degree m will not be used by any future IAP.

Designing the circuit for NS4: NS4 defined in Section 4.3 can be broken into three sub NP statements and three separate individual circuits can be created for ZK-SNARKs associated with each of these three sub NP statements. These three NP statements are based on the three different ways the updated encoded coefficients $En(P'_i)$ at different indexes i are computed, as shown in equation 2:
Let the number of identity assets that are already registered in the system be c.

$$En(P'_i) = \begin{cases} En(1) \text{ for } i = c+1; \\ En(P_0)^{-a} \text{ for } i = 0 \\ En(P_i)^{-a}.En(P_{i-1}).En(0) \text{ for i = \{1, ..., c\}}. \end{cases} \tag{2}$$

Accordingly, we can create three different NP statements, based on the three different cases shown in the equation 2, in order to prove the correctness of each updated encoded coefficient of the polynomial. In equation 2, only the cases (ii) and (iii) use the cryptographic hash of the identity asset (a) being registered, in the computation of $En(P'_i)$.

The three sub NP statements which replace NS4 defined in Section 4.3 are as follows.

(1) NS4.1 (corresponding to case (i) of equation 2):
Given the $(c+1)^{th}$ encoded coefficient $En(P'_{c+1})$ of the updated polynomial P', I know a secret k s.t: $En(P'_{c+1}) = En(1) = (g^k, h^k.g)$.
(2) NS4.2 (corresponding to case (ii) of equation 2):
Given the commitment C_{IA}, the 0^{th} encoded coefficient $En(P_0)$ of the existing polynomial P, from set S, and the 0^{th} encoded

coefficient $En(P'_0)$ of the updated polynomial P', I know the secrets a, r, k_0 s.t: C_{IA} = commit(a,r) and $En(P'_0) = En(P_0)^{-a}.En(0)$ and $En(0) = (g^{k_0}, h^{k_0})$.

(3) NS4.3 (corresponding to case (iii) of equation 2):
Given the commitment C_{IA}, the i^{th} and $(i-1)^{th}$ encoded coefficients $En(P_i)$ and $En(P_{i-1})$ of the existing polynomial P, from set S, and the i^{th} encoded coefficient $En(P'_i)$ of the updated polynomial P', I know secrets: a, r and k_i s.t: C_{IA} = commit(a,r) and $En(P'_i) = En(P_i)^{-a}.En(P_{i-1}).En(0)$ and $En(0) = (g^{k_i}, h^{k_i})$.
Note that an IAP, who is registering a new identity asset when there are c number of identity assets registered in the system, should create c number of zero knowledge proofs using this circuit, each proving the correct computation of the i^{th} encoded coefficient of the updated polynomial P', for all $i \in \{1, ..., c\}$.

Combining the proofs for NS3 and NS4: Note that we need to make sure that the prover uses the same secret input 'a' across all zero knowledge proofs created during the identity asset registration phase (i.e. zero knowledge proofs created for NS3, NS4.2 and NS4.3). This is achieved by involving the computation of C_{IA} = commit(a, r) in each NP statement that involves 'a' in its computation. Because C_{IA} is publicly known and one can not come up with different (a, r) pairs that give the same C_{IA} value, without breaking the security (i.e. binding property) of the commitment scheme, we can make sure that the prover uses the same 'a' in all zero knowledge proofs by verifying whether the public input C_{IA} submitted with the zero knowledge proofs for all NP statements are the same and that those zero knowledge proofs are successfully verified.

Discussion on scalability: The maximum number m of identity assets allowed to be registered in the system directly affects the performance of the identity asset registration phase due to the fact that the solution for counterfeit elimination while preserving unlinkability, is based on evaluation of a polynomial in the encoded domain. The number of coefficients of the updated polynomial grow with the number of identity assets being registered which in turn increases the costs associated with the ZK-SNARKs for NS3 and combined NS4. Particularly, the costs of both *Generator()* and *Prover()* algorithms of ZK-SNARKs for NS3 and the cost of *Prover()* algorithm of ZK-SNARKs for combined NS4 increases with m.

In oder to overcome this scalability issue, the identity management system can define a limit for the maximum degree of the polynomial (let it be p) that the system can handle in ZK-SNARKs, without causing unacceptable performance in the identity asset registration phase. Once p number of identity assets have been registered, the system can allow registration of the next set of p number of identity assets using the same circuits created for ZK-SNARKs, by treating the $(p+1)^{th}$ identity asset as the 0^{th} identity asset. Note however, that counterfeit elimination is not preserved between such two different sets of identity assets, as this is a trade-off between scalability and counterfeit elimination. Unlinkability is still preserved between registration and/or transfer of identity assets from two different sets and the ownership declarations associated with the identity assets from two different sets can be stored in the same or different merkle hash trees.

Implementation of the circuits: Our encoding scheme, which is Elgamal encryption in the exponent, can be efficiently implemented over elliptic curves, because the size of the cipher text can be significantly reduced, compared with the same implemented over a field. Therefore, we implement the circuits required for NS3 and NS4 over elliptic curves. The elliptic curve supported by jSNARK is: $y^2 = x^3 + A.x^2 + x$, where $A = 126932$, and it achieves 125-bit security [10]. Three basic operations required to perform computations over elliptic curves are: addition of two points, multiplication of a point by a scalar and negation of a point. Starting from these three basic operations, we build the complex gadgets and circuits required for NS3 and NS4, in a bottom-up and a modular approach.

In what follows, we first present the three circuits associated with the three sub NP statements of NS4 and then present the circuit for NS3, because it illustrates the gradual development of the circuits using the aforementioned basic building blocks. Let B is the base point and P is the public key point on the elliptic curve.

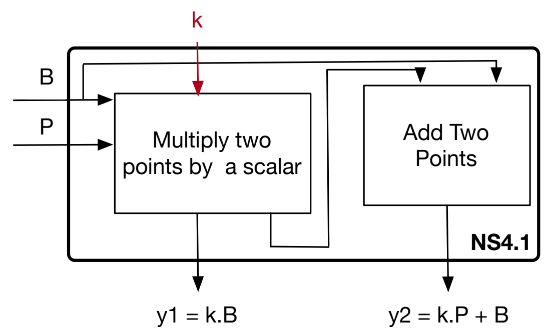

Figure 7: Circuit for NS4.1: ***Given an encoded coefficient*** $En(P'_{c+1}) = (y_1, y_2)$***, I know a secret key*** k ***s.t.*** $En(P'_{c+1}) = En(1) = (k.B, k.P + 1.B)$**.**

Figure 7 illustrates the circuit used to prove NS4.1; in other words, this is the composite circuit used to prove that a fresh encoding of 1 is computed correctly as the $(c+1)$th encoded coefficient of the updated polynomial. This is consisted of a new gadget for multiplication of two points by a scalar, which is built on top of the basic gadget for multiplication of a point by a scalar, in addition to the basic gadget for addition of two points.

Figure 8 illustrates the circuit used to prove NS4.2; in other words, this is the composite circuit used to prove that the zeroth encoded coefficient of the updated polynomial is computed correctly. This circuit uses the new gadget (multiplication of two points by a scalar) introduced in the circuit for NS4.1 above, for two main purposes: i) to multiply $En(P_0)$ by the cryptographic hash of the identity asset, ii) to compute a fresh encoding of zero. Furthermore, this circuit uses the third basic gadget for negation of a point, and introduces a new gadget for addition of two encodings, which is built on top of the basic gadget for addition of two points. Lastly, this circuit also includes the commitment gadget (i.e. SHA256 gadget in jsnark), in order to help verifying that all the ZK-proofs that involves the cryptographic hash of the identity asset as a secret input, use the same value, as discussed before.

Figure 9 illustrates the circuit used to prove NS4.3; in other words, this is the composite circuit used to prove that the ith encoded

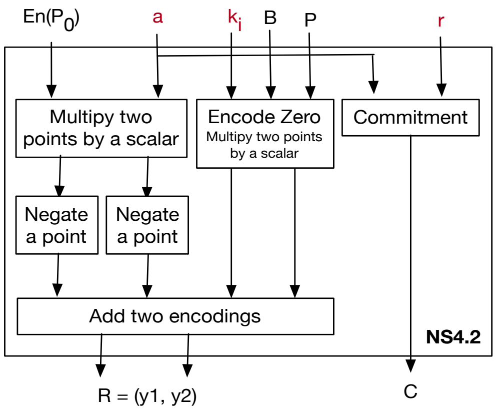

Figure 8: Circuit for NS4.2: ***Given an encoding*** R***, the existing encoded coefficient*** $En(P_0)$***, a commitment*** C***, I know secrets:*** a***,*** r ***and*** k ***s.t.*** $R = -a.En(P_0) + En(0)$ ***and*** $C = commit(a, r)$***.***

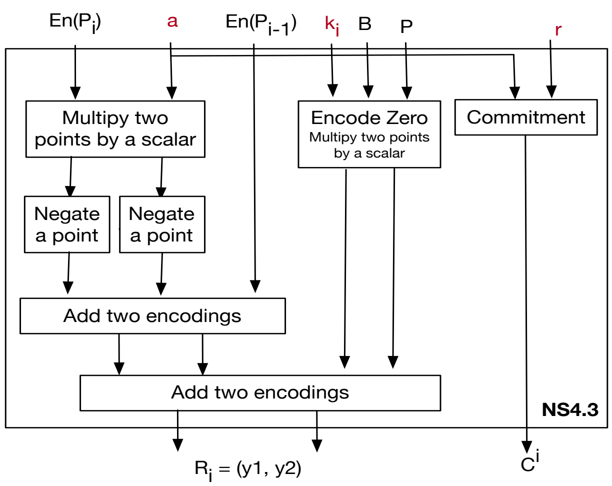

Figure 9: Circuit for NS4.3: ***Given an encoding*** R_i***, the existing encoded coefficients*** $En(P_i)$***,*** $En(P_{i-1})$ ***and a commitment*** C***, I know secrets:*** a***,*** r ***and*** k_i ***s.t.*** $R = -a.En(P_i) + En(P_{i-1}) + En(0)$ ***and*** $C = commit(a, r)$***.***

coefficient of the updated polynomial is computed correctly. This circuit uses all three basic gadgets, and the new gadgets introduced in the previous circuits for NS4.1 and NS4.2. As mentioned before, this circuit is used to create multiple zero knowledge proofs, for each ith coefficient of the polynomial, for all $i \in \{1, 2, .., c\}$).

Table 4 reports the performance numbers for the ZK-SNARKS associated with the three individual circuits for NS4.1, NS4.2 and NS4.3. Using these performance numbers collected for the individual circuits, we calculate the performance numbers for the cases where these circuits are combined to prove that the updated encoded coefficients are computed correctly, for the polynomials of different degree sizes. Since the maximum number (m) of identity assets that can be registered in the system should be a power of 2 (due to the structure of the Merkle hash tree), and ZK-SNARKs for NS4 are created only up to $(m-1)$th identity asset being registered, we calculate the performance numbers by varying the index c of the currently registering identity asset, where $c \in \{1, 3, 7, 15, 31\}$, corresponding to each case of $m \in \{2, 4, 8, 16, 32\}$. Note that since Generator() algorithm is run only once during the bootstrapping of the system for all three individual circuits, the first four performance metrics given in table 4 do not vary with c. Therefore, we only focus on the changes in prover running time and verifier running time (shown in Figure 10) and the proof size (shown in Figure 11), with varying c. Note that the only factor that contributes to these changes, is the fact that multiple ZK-SNARKs should be created using the circuit for NS4.3, for each ith coefficient of the polynomial, for all $i \in \{1, 2, .., c\}$). As shown in Figure 10, each time m doubles, the prover running time increases linearly, whereas the verifier running time stays almost constant. As shown in Figure 11, each time m doubles, the proof size increases linearly.

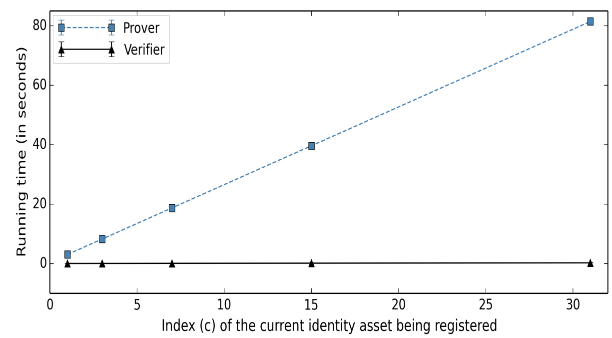

Figure 10: Running times vs the index of the identity asset currently being registered, for the combined circuits for NS4.

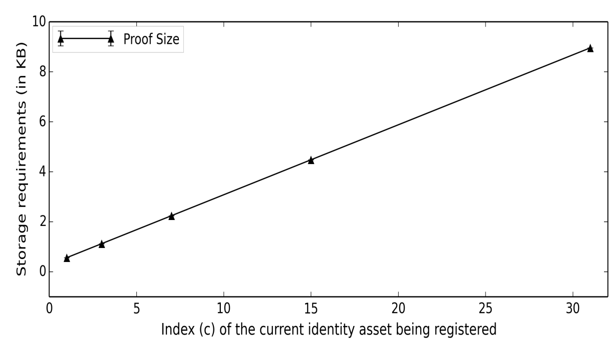

Figure 11: Proof size vs the index of the identity asset currently being registered, for the combined circuits for NS4.

Figure 12 illustrates the circuit used to prove NS3; in other words, this is the composite circuit used to prove that the evaluation of the polynomial on the cryptographic hash of the identity asset currently being registered, is computed correctly in the encoded domain. Without loss of generality, Figure 12 illustrates the circuit used to prove NS3, when registering the last identity asset in a system

	NS4.1	NS4.2	NS4.3
1. Circuit size (number of constraints)	3273	34,674	34,682
2. Key gen running time	0.6890(s)	4.8959(s)	5.2958(s)
3. Proving key size	863.63(KB)	8211.7685(KB)	8241.3403(KB)
4. Verification key size	0.5987(KB)	1.0660(KB)	1.2218(KB)
5. Prover running time	0.6091(s)	2.41(s)	2.6144(s)
6. Proof size	0.28(KB)	0.28(KB)	0.28(KB)
7. Verifier running time	0.0065(s)	0.0062(s)	0.0069(s)

Table 4: performance numbers for the ZK-SNARKS associated with the three individual circuits for NS4.1, NS4.2 and NS4.3.

where $m = 4$. In our implementation, the circuit can be shrunk or expanded according to the index of the currently registering identity asset. This circuit uses all three basic gadgets, and the new gadgets introduced in the previous circuits for NS4.1 and NS4.2.

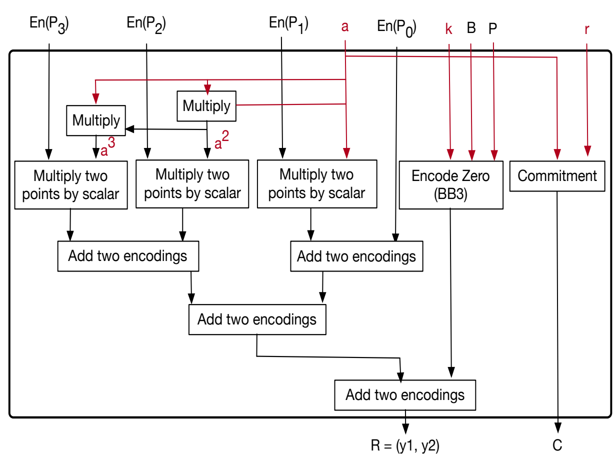

Figure 12: Circuit for NS3: ***Given the existing set of encoded coefficients*** $En(P_3)$***,*** $En(P_2)$***,*** $En(P_1)$***,*** $En(P_0)$ ***of polynomial P of degree 3, an encoding*** R***, and a commitment*** C***, I know secrets:*** a, r ***and*** k ***s.t.*** $R = En(P(a)) + En(0)$ ***and*** $C = commit(a, r)$***.***

Figure 13 reports the running times and Figure 14 reports the storage sizes for the ZK-SNARKS associated with the circuit for NS3, by varying the maximum number m of identity assets that can be registered in a given system, where $m \in \{2, 4, 8, 16, 32, 64\}$.

Note that when measuring the perfomance numbers in an Amazon EC2 instance of type t3a.large, jsnark/libsnark framework times out before reporting the performance results, after reaching $m = 22$. Therefore, for completeness, we include the performance results for $m = 22$, as the last instance of measurement in the Amazon EC2 instance of type t3a.large, although m should be a power of two as described before. In order to decide whether this is due to limitations in computing resources or a limitation in jsnark/libsnark framework, we carried out the same performance measurements in an Amazon EC2 instance of type t3a.2xlarge as well. The ZK-SNARKS associated with the circuit for NS3 could be run without an issue for $m = 32$ and $m = 64$ as well in the larger instance. Therefore, given sufficient computing resources, jsnark/libsnark framework can run ZK-SNARKS for more complex circuits.

In each instance of measurement, we measure the performance of the ZK-SNARKS associated with the circuit for NS3, when registering the last identity asset of the system. Despite the fact that NS3 being the most complex circuit in the identity asset registration phase of protocol V3, and that the complexity increases with the increase of c in factors of two, verifier running time and proof size remains constant and verification key size increases at a very low rate. Proving key size, key generator running time and prover running time increases linearly with the increase of c, although the prover running time is always lower than the key generator running time.

The aforementioned performance numbers for the circuits of protocol V3 are primarily obtained in an amazon EC2 instance of type t3a.large. The ZK-SNARKS associated with the circuit for NS3 were also run in an amazon EC2 instance of type t3a.2xlarge due to the resource limitations in the first instance type, as discussed above.

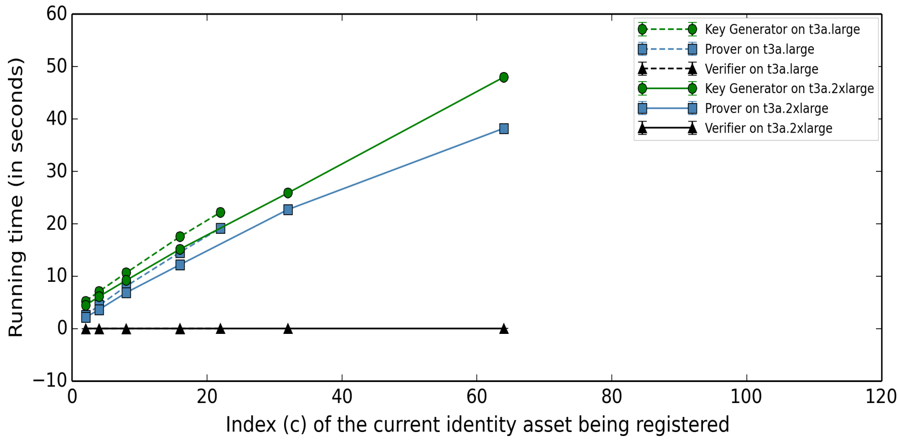

Figure 13: Running times vs the index of the identity asset currently being registered, for the circuit for NS3. The dashed lines show the performance measurements in an Amazon EC2 instance of type t3a.large and the solid lines show the performance measurements in an Amazon EC2 instance of type t3a.2xlarge

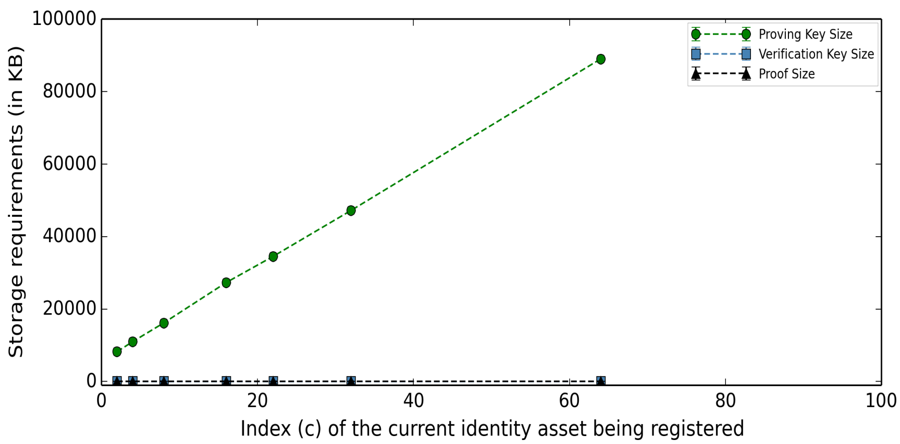

Figure 14: Storage size vs the index of the identity asset currently being registered, for the circuit for NS3. The storage sizes do not change depending on the instance type, although we could collect measurements only up to $m = 22$ in an Amazon EC2 instance of type t3a.large and the remaining measurements are collected in an Amazon EC2 instance of type t3a.2xlarge