35th European Modeling & Simulation Symposium
20th International Multidisciplinary Modeling & Simulation Multiconference

2724-0029 © 2023 The Authors.
doi: 10.46354/i3m.2023.emss.022

BlockVerify: Privacy-Preserving Zero-Knowledge Credentials Verification Framework on Ethereum

Theodoros Constantinides[1,*] and John Cartlidge[1]

[1]Department of Computer Science, University of Bristol, Bristol, BS8 1UB, UK

[*]Corresponding author. Email addresses: theodoros.constantinides@bristol.ac.uk; john.cartlidge@bristol.ac.uk

Abstract

We present a general purpose, privacy-preserving framework for verifying user attributes. The framework is designed for users (e.g., a job candidate) to allow a challenger (e.g., a prospective employer) to verify whether the user meets a particular requirement (e.g., does the candidate hold a valid driving license?), without leaking any other information about the user. Importantly, the user is an active part of the challenge-verification process, which ensures that challenges cannot be made without the user's full knowledge and participation. The framework is decentralized and requires a public blockchain. A smart contract is used to manage the challenge-verification process, and zero-knowledge proofs are used to verify challenges in a privacy-preserving manner. We implement a simplified version of the framework using smart contracts deployed on the Ethereum blockchain, and we simulate some simple use cases. All simulation code is available open-source (https://github.com/lifeisbeer/BlockVerify).

Keywords: Blockchain; smart contract; zero-knowledge proofs; privacy-preserving; verification

1. Introduction

People are regularly required to prove certain attributes about themselves. For example, a candidate for a job might need to prove to the hiring company that they have the required visa permissions to work, that they meet a certain educational requirement such as a degree, etc. Traditionally, people use identity documents (such as passports, birth certificates, national identify cards, etc) and other forms of certification (such as university degree, driving license, pay slip, etc) to prove their attributes. More often than not, these documents are in physical (paper) format, which makes it difficult for someone else to verify them. Even when the document information is available in digital form (such as a biometric passport), the verifier will often need specialized equipment (such as near field communication (NFC) readers) to access the information, and it may be necessary to navigate a number of different systems and processes to verify each document. The verification of such documents can therefore become a lengthy and error-prone process that requires manual labour and does not scale well. Indeed, the current processes are so complex that it has led to the establishment of paid services (such as notaries and screening companies) that carry out verification on behalf of clients. However, this is not ideal, as it incurs unnecessary costs, allows for collusion, does not completely eliminate the possibility of fraud (for example if someone is able to forge a passport, they could also forge the seal of a notary), and leads to repeated costly verification of the same individuals and documents (for example when a candidate applies for multiple jobs at different companies and each company hires a screening agency to perform their own verification checks).

In this paper, we introduce a new general purpose and privacy preserving system for verifying user identities and attributes. In our system, every document (e.g., passport, birth certificate, driving licence, university degree, etc) is represented by a smart contract running on the Ethereum

blockchain. In each smart contract, there are a number of *verifiers* that are tasked with verifying the authenticity of the corresponding document. These verifiers are known to all users of the smart contract and include issuing agencies (e.g., the government, or a public body such as the Driver and Vehicle Licensing Agency) and other independent trusted bodies (e.g., a Notary). A *user* who owns a document can then visit one of the recognised verifiers to get their document verified. The verifier will assign the user's blockchain address with a privacy-preserving representation of the user's document on the smart contract. This representation contains all the user's attributes that are present in the document (such as date of birth, address, etc) in a way that is only meaningful to the user and does not reveal any information to other observers. More importantly, from that point on, only the user can use that representation to verify their attributes; everyone else, including the public body or notary that verified the user's attributes, cannot. Other users – the *challengers* (e.g., a hiring company) – can then post challenges to the smart contract. These challenges could be of binary form (e.g., does the user have a valid driving licence?) or of ordinal form (e.g., is the user's age greater than 18?). Each challenge could specify all the fields contained in a document, or any chosen subset. Finally, users see the challenges and are able to respond by providing a Zero-Knowledge Proof (ZKP) that their attributes meet the specified attributes in the challenge. The smart contract automatically verifies each submitted ZKP and informs the challenger. The ZKP does not reveal the user's attributes; the only information revealed is that the user's attributes meet the minimum requirements set in the challenge. Finally, the challenger can see who verified each user that responded to the challenge, and decide if they are willing to accept their verification or not (for example, some challengers might only accept government verification, while others might be willing to be less restrictive).

Contribution: We introduce a general purpose privacy-preserving system for verifying user attributes using smart contracts deployed on the Ethereum blockchain. Our system, simplifies and combines many of the best elements of previous systems, while introducing new design choices that provide some clear benefits for the applications we are interested. We implement a proof of concept system, and through simulation of some example use cases, demonstrate system use and performance. All system code is made available open source for research and education.

2. Structure of the paper

In the following sections, we first introduce the relevant literature and the state of the art in section 3. We split this section into subsection 3.1 which contains literature related to anonymous credential systems, subsection 3.2 which is focused on blockchain based systems and subsection 3.3 which specifies the aim of our paper. Then we present our implementation in section 4. Our results are presented in section 5. Finally, section 6 is a conclusion that summarizes our finding and presents our plans for future work.

3. Literature review of state of the art

3.1. Anonymous credential systems

The problem of anonymous credentials that are owned by individual users and are transferable between different organizations/applications was first proposed in the 1980s by Chaum (Chaum, 1985). Chaum's anonymous credential system consisted of organizations that know their users by their pseudonyms only, and issue credentials to those pseudonyms. Under a different pseudonym, the user can then prove ownership of that credential to a different organization, thus providing an anonymous credential system.

Zero-Knowledge Proofs (ZKP) (Goldwasser et al., 1989) were invented to allow a party (the prover) to prove a statement to another party (the verifier) without revealing anything apart from the statement that is being proven. One of the first applications of ZKPs was to solve the identification problem. The first zero knowledge identification scheme (Fiege et al., 1987), allowed a user to verify their identity to a challenger by replying to several challenges posted by the challenger. In this system, the verifier learns nothing about the identity of the user, and crucially the challenger is unable to impersonate the user even after the user has verified themselves to the challenger. Thus, this system allows for the creation of unforgeable identities.

Initially, ZKPs were interactive, meaning that the prover had to communicate with the verifier. This changed with the introduction of non-interactive zero-knowledge (NIZK) proofs, and especially with zero-knowledge succinct non-interactive argument of knowledge (zk-SNARK) (Bitansky et al., 2012). Variations of NIZK proofs are now commonly used, particularly in decentralized blockchain-based credential systems, where non-interactive argumentation is required. We consider these in Section 3.2.

In recent years, there has been a big shift in creating credential systems that are not only of academic interest, but also of practical use. In the past, credentials were usually envisioned as single access tokens along with methods to prove ownership of such tokens. In contrast, more modern systems allow for credentials that contain multiple user attributes and allow for more complex processes. A milestone was IBM's *idemix* system (Camenisch and Van Herreweghen, 2002). In idemix, users obtain credentials from organizations under pseudonyms that contain arbitrary attributes. An idemix credential is defined as a certificate showing that a pseudonym has certain attributes, along with a signature of that certificate from the issuing organization. Users owning such credentials can then use ZKPs to prove to other organizations that they own a certificate with certain attributes, and that the certificate was signed by the issuing organization, without revealing anything else (such as their pseudonyms with different organiza-

tions or other attribute details). The main innovation of this system is the signature scheme used (Camenisch and Lysyanskaya, 2001), which is efficient for ZKPs.

3.2. Decentralised blockchain-based systems

Blockchain technology to achieve anonymous credentials was first used in (Garman et al., 2013). This not only provided anonymous, but also decentralized credentials. In this system, a user sends their identity attributes, a public identity assertion, a commitment of their attributes, and a proof that the commitment matches their attributes to a network of parties (i.e., blockchain nodes) who verify the commitment proof, verify (using an external procedure) the user's identity, and if correct add the user's commitment to a set containing all other commitments. At a later point, the user can produce a ZKP that they know how to open one of these commitments and any other statement about their attributes that are contained in the commitment. This system also demonstrated that it was possible to switch from blind signature to ZKPs as the main component of an anonymous credential system.

The previous application required the participation of blockchain nodes in the verification. With the introduction of general purpose blockchain systems that support smart contracts (e.g., Ethereum), this participation of nodes is no longer needed. This has allowed for more general applications of anonymous credentials and self-sovereign identities. Examples of these include Know Your Customer (KYC) checks (Biryukov et al., 2018; Pauwels, 2021), safe ride-sharing (Li et al., 2020), and car-sharing (Gudymenko et al., 2020). More importantly, it has allowed for more general and flexible credential and identity management systems (Sonnino et al., 2020; Mukta et al., 2022; Luong and Park, 2023; Namazi et al., 2022; Lee et al., 2021; Rosenberg et al., 2022).

Coconut (Sonnino et al., 2020) is a *selective disclosure* credential system that is based on a threshold issuance signature scheme, i.e., in order for a user to obtain a credential in coconut, a set number of different authorities from the set of all authorities must issue a partial credential (signature) to the user. The user can then collect the required number of shares and establish their credential, in the form of a valid signature that some attributes belong to them. The user can then disclose part of their attributes or some statement about them using a ZKP to a verifier. An Ethereum implementation of the Coconunt smart contact library is available: coconut-ethereum. Another selective disclosure identity management system (Luong and Park, 2023) allows for anonymous identities while allowing the system to trace users who violate the system's policy.

CredTrust (Mukta et al., 2022) is a credentials platform built on top of a blockchain. Unlike the other applications we have described, CredTrust is not concerned with anonymity of users. Rather, it is concerned with building a trust propagation method where higher authorities, starting from an official issuer of credentials, endorse other lower-level issuers of credentials. This builds a "chain of trust" where users who interact with those lower-level issuers can be sure that they are legitimate as they have been verified by a more trusted authority.

zkFaith (Namazi et al., 2022) is a protocol where users have their documents verified by an authority and create commitments of their attributes, which are signed by an issuer. The issuer in this protocol also maintains a list of all revoked credentials. Users can then prove they are eligible for a service by using a ZKP to prove that they hold the right attributes, that they have a valid signature, and that they are not part of the revoked credentials list. Additionally, the protocol allows users to update their records by repeating the above procedure. Importantly, the zkFaith protocol uses a smart contract to verify ZKPs, which means that the verification process is automatic and does not require a challenger to run specialized software.

Another system (Lee et al., 2021) follows the same approach as zkFaith. This system has a similar design but introduces a verifiable data registry where all credentials are stored. A user who holds credentials in this system, can then prove to a verifier some statement about their attributes using a zk-SNARK. The zk-SNARK takes as public input the credentials that are stored in the public registry, which can eliminate any signature checks. The authors also use a different version of zk-SNARKs, the Commit-and-Prove zk-SNARKs (CPSNARKs) which are more efficient for this application as they remove the need for commitment checks in the circuit.

zk-creds (Rosenberg et al., 2022) is an anonymous credential system that uses ZKPs instead of blind signatures as its basis. Users of the system first convince a credential issuer that they own certain attributes that form their credential, which in this system is represented by a commitment to those attributes. The issuer then adds the user's commitment to a Merkle tree (or Merkle forest) which is publicly available. The user can then prove an arbitrary statement about their attributes by providing a ZKP that their commitment is included in a Merkle tree and that it meets the criteria set in the statement.

3.3. Research Aim

The aim of this paper is to introduce a new credential system, BlockVerify, which simplifies and combines many of the best elements of previous systems, while introducing some new design choices that provide some clear benefits. Similar to the systems mentioned so far, BlockVerify uses ZKPs as its main building block, a distributed ledger (a blockchain) as a public registry of issued credentials, and a smart contract to automatically verify ZKPs. As such, BlockVerify allows for multi-attribute credentials, multiple credential issuers, easy revocation of credentials, and does not require endorsement from official issuers.

Additionally, BlockVerify provides users and challengers the option to re-use previous proofs, for example a user who has proven that they are an adult in the past does

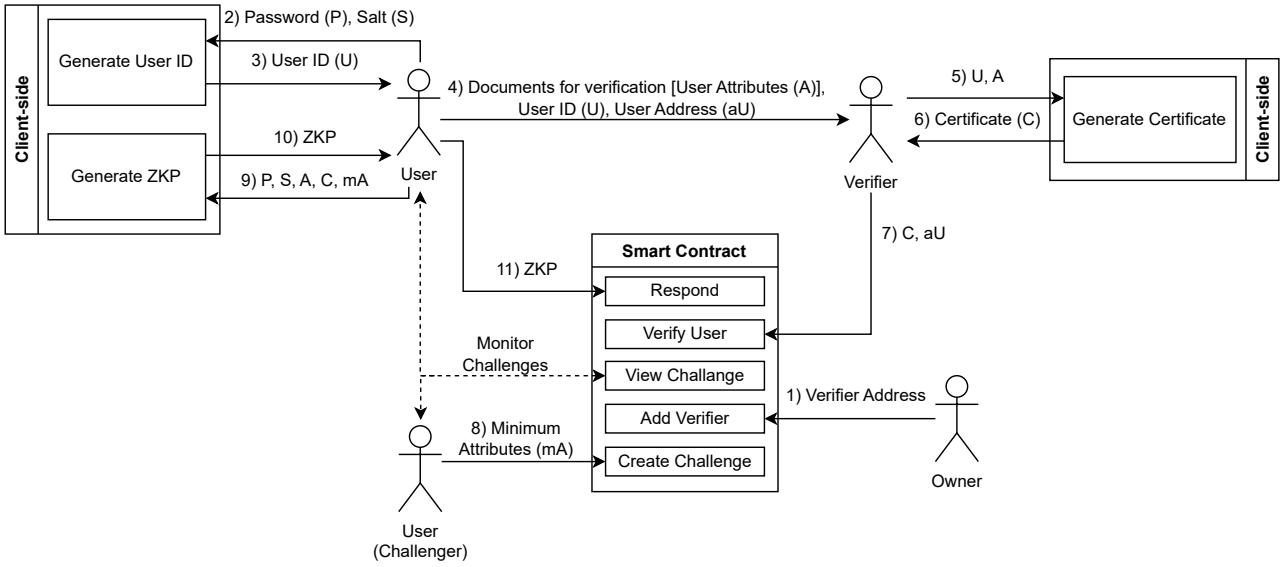

Figure 1. System overview showing the smart contract, system actors and their client-side processes. Multiple actors of each kind and multiple smart contracts can exist in real usage.

not have to prove that statement again in the future. While re-use of proofs might be possible in previous systems, it would require additional work to store exchange of proof information outside the system, or work to search for this information, and replay it. In contrast, proof re-use is built into the design of the BlockVerify framework.

Finally, we release our implementation open source.

4. Materials and Methods

4.1. System design

Figure 1 presents an overview of the system. There is only one smart contract in the figure but, in real usage, every document (e.g. UK passport, Cypriot driving licence, etc) would have its own smart contract. The system has four actors, described below:

Owner: Smart contract owner. Deploys the smart contract and adds the verifiers. We envision the contract owner as the entity who originally issued the document, e.g., DVLA for UK driving licences; GRO for UK birth or marriage certificates; DBS for UK criminal records, etc. This is however not a requirement, and in practise anyone can be the contract owner.

Verifier: An entity that can provide verification of user attributes. The contract owner can also be a verifier if they wish to. Other verifiers could be typical document verifiers, such as screening companies and notaries. As before, in practise anyone can be registered as a verifier.

User: An individual with documents that require verification (e.g., a driving license, a birth certificate, a criminal record check).

Challenger: A user, such as an employer, that wants to check whether another user, e.g., a potential employee, meets some requirement (e.g., user can drive, is aged over 18, has no criminal record).

Obviously, the contract owner should be someone that is generally trusted. That is why we have envisioned the contract owner as the issuing authority of the document. However, even if the contract owner is not trusted, challengers can still be sure about the validity of documents as long as they trust the individual verifier that verified a user's document. The reverse is also true, challengers should not trust the validity of user documents if they do not trust the verifier that verified them, even if they trust the contract owner.

This design choice makes the system more flexible and practical for real-life use. First, it allows the implementation of such a system without the involvement of the issuing authority of a document, as we assume that in many cases it would be difficult to get the issuing authority to participate. Second, it also allows individual challengers to decide if they trust a certain verifier and accept or reject the users they have verified. Challengers can also tailor the level of trust they are willing to accept based on the use case. For example, for a very important document or for a high-risk user, the challenger might only accept verification by a government authority; while for a less important document or for a less risky user, the challenger might be more relaxed and accept verification from a third party. Finally, it is more convenient for users, as they can get the same document verified multiple times by different verifiers. For example, a user can get a document verified somewhere locally or somewhere that they do not have to wait for a long time if they know that this verification will be acceptable for their use case. In this way, simpler and/or less important tasks can avoid more formal (and

more lengthy) processing.

The contract owner first adjusts a constant that defines how many user attributes are present in the system. For example, a birth certificate might only have one attribute, the date of birth, while a university degree might have four, the type (certificate, diploma, bachelors, etc), the grade, the issuing date, and the issuing body. Note that it is also possible to have a tailored instance that is a combination of many documents, but this is likely to be less reusable and will make it more difficult for users to get verified. The contract owner then deploys the system's smart contracts.

The following steps (also annotated on figure 1) describe how the system works:

1 The contract owner adds a verifier. This step can be repeated for as many verifiers as the contract owner wants to allow.

2-3 Each user of the system creates a User ID (U). A user first selects a password and a random number (the password salt). The hash of the password and the salt acts as U. A user can use the same ID between different instances of the system, or can choose to use different ones. Note that U is generated off-chain for privacy.

4 The user gets their document verified by one of the approved verifiers. The user provides the verifier with U and a blockchain address (aU) they wish to use. Importantly, the user does not reveal their password and salt, but just U.

5-7 The verifier verifies the authenticity of the provided document and constructs a certificate (C) that incorporates U and all the attributes of the user's document (A). In practise, C is the hash of U along with all elements of A and is done off-chain. Then, the verifier assigns this certificate to the user's address (aU) on the smart contract. Note that any observer looking at C is not able to extract any of the user's attributes. A verifier, also has the ability to revoke any certificates they have issued.

8 The challenger creates a challenge on the smart contract by posting a list of minimum attributes. The challenge can then be answered by multiple users that fulfil these criteria. A challenger can monitor the smart contract for replies. Note that multiple challengers and multiple challenges from the same challenger can exist simultaneously on the smart contract.

9-11 Users can monitor the smart contract for new challenges. Any user can then reply to a challenge by first creating a zero-knowledge proof (ZKP) proving that a certificate (C) they hold, meets the minimum requirements set by the challenge. Note that if the user holds multiple certificates, they can select which one is used. Then, the user provides the ZKP to the smart contract, which verifies it. If the ZKP is valid, the user's address along with the address of the verifier who verified them are added to a list of replies to that particular challenge.

Below, we present a concrete system use-case, where a challenger (for example an employer) creates a challenge and a user (a prospective employee) responds, proving that they have certain attributes (for example hold a valid driving licence of a certain vehicle category for more than three years), without revealing anything else:

1 DVLA adds verifiers.

2-3 User creates a user ID.

4 User sends their physical driving licence for verification (assuming by DVLA).

5-6 DVLA verifies the user's driving licence and create a certificate (C).

7 DVLA assigns C to the user's address on the smart contract.

8 An employer creates a challenge, requesting some minimum requirements (e.g. user can drive a truck, has licence for more than 3 years etc). This can be represented as $vehicleCategory > 1$, $issuingDate < 1589065200$ (for date and time we use Unix timestamps)

9-10 User creates a ZKP that minimum attributes are met.

11 User provides the ZKP to the smart contract. The employer will now be able to see the user's response.

4.2. System Components

The system relies on multiple components including arithmetic circuits, smart contracts, and some auxiliary JavaScript code. All the code is open-source and available on our code repository: BlockVerify.

4.2.1. *Arithmetic Circuits*

All arithmetic circuits were built using Circom, a domain-specific language for writing arithmetic circuits.

We are using three fairly simple circuits for this application: one to calculate user IDs, one to calculate document certificates, and the final one to calculate the constraints that a certificate meets some minimum requirements for use by the ZKPs. All circuits rely on the Poseidon hash function (Grassi et al., 2019). This hash function was chosen as it was designed to work efficiently with zk-SNARKs.

The first arithmetic circuit is used to compute the user IDs. The input signals to the circuit are a user's password and a password salt. The circuit hashes both and outputs the result. A salt is used to prevent rainbow table attacks on the user ID (i.e., protect against pre-computed hashes). The second arithmetic circuit similarly performs a hash to compute a certificate. Its inputs are a user ID and a list of attributes from a user's document. The circuit hashes these together and outputs the result, which acts as the certificate.

The third circuit is the most complex, as it is responsible for the constraints that are used in the ZKP. It takes as input the user's data (password, salt, and a list of user attributes), the user's certificate, and the minimum requirements (both a list of values and a same size list of zeros or ones, which indicates if each comparison is a *less than or equal* or *greater than*). The circuit then checks that

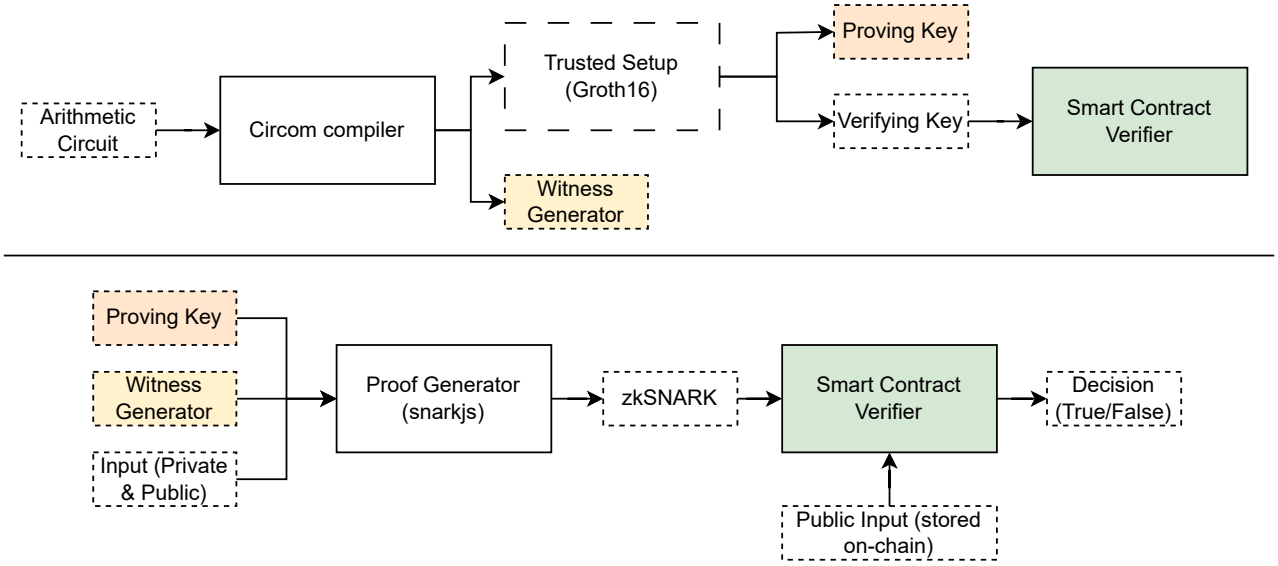

Figure 2. Simplified zk-SNARK generation and verification procedure. The top half is performed once and its outputs (Witness Generator, Proving Key, and Smart Contract Verifier) are used in the bottom half. The bottom half is performed every time a user wants to create and verify a ZKP.

all user attributes satisfy the minimum requirements, recomputes the certificate, and makes sure it matches the original certificate provided. By requiring the password and salt instead of the user ID, we ensure that only the user can run this operation, as they are the only ones that know their password and salt. This means that even a verifier who knows the user's attributes is unable to imitate the user.

4.2.2. Compiling the Circuits and Trusted Setup

The arithmetic circuits used by this system must be compiled before a document smart contract is deployed on chain. Additionally, a process known as "trusted setup" must be performed. The whole process is illustrated in figure 2.

Documents that contain the same number of attributes use exactly the same arithmetic circuits. This implies that compiling the circuits and performing the trusted setup once would cover all documents in a document family (i.e., for all documents with one/two/three/… attributes). Therefore, in an ideal usage of the system, circuits for a number of document families would be compiled, a trusted setup for each family would be performed, and the results would be stored somewhere they can be reused.

4.2.3. Smart Contracts

As described before (see Section 4.1), each document is associated with a different smart contract. Additionally, each smart contract also uses a second smart contract that verifies the validity of ZKPs. This verifier smart contract is produced automatically using the snarkjs library at the end of the trusted setup (see Figure 2). There are many variants of zk-SNARK proving schemes and for our use case the choice of scheme is largely arbitrary. We use *Groth16* (Groth, 2016), one of the most popular zk-SNARK schemes, but this choice can be easily changed in future. Indeed, snarkjs also supports *PLONK* (Gabizon et al., 2019) and *fflonk* (Gabizon and Williamson, 2021), so switching to one of these alternative schemes is trivial.

So, for each document, there exist two smart contracts. However, as with circuits, the verifier smart contract is universal between all documents that contain the same number of attributes. Therefore, this smart contract can be deployed only once and then be shared by all similar documents (with the same number of attributes). An instance of the verifier smart contract is available on the testnet: Etherscan.

The other smart contract uses the verifier smart contract for verification of zk-SNARKs. This smart contract contains all public data and most of the functionality of the system, and thus must be unique for each document. An instance of this smart contract is available on the testnet: Etherscan.

4.2.4. Auxiliary Functions

The rest of the code is written in JavaScript and is essentially used for any user or verifier actions that take place off-chain. Specifically, these actions are the creation of user IDs, the creation of certificates, and the creation of ZKPs. This code, can be packaged into an application or a front-end in a production-ready application.

This implies that ZKPs are produced on the client side, which can be a security risk. Without proper care, a malicious user could provide a false certificate or minimum requirements that do not match the ones set by the challenger on the smart contract. To avoid this, the certificate and the minimum requirements are all set to public inputs, which means that they are populated when a ZKP is verified on the smart contract (as shown in the bottom half of Figure 2). This ensures that the correct values were used in

Table 1. Gas, ETH and Monetary Cost of Operations

Operation	Gas Cost	ETH Cost*	USD Cost*
Deploy Verifier SC	1,651,322	0.03303	$59.45
Deploy Document SC	2,388,458	0.04777	$85.98
Add Verifier	72,629	0.00145	$2.61
Verify User	47,141	0.00094	$1.70
Create Challenge	153,981	0.00308	$5.54
Respond	378,199	0.00756	$13.62

* If the system were to be used on the Ethereum mainnet, using a gas price of 20gwei and ETH price of $1800.

the creation of the ZKP, as otherwise this mismatch would render the proof invalid, resulting in the smart contract rejecting the transaction.

5. Results and Discussion

We have deployed and tested an instance of our system in the Sepolia testnet. This particular instance, uses a document that contains three attributes. Table 1 summarizes the gas cost associated with each operation in the protocol.

The monetary cost of deploying and using this system on the Ethereum mainnet might be okay for certain applications where traditional verification processes cost a lot of money anyway. But the cost associated with using the system can be prohibiting for lower value applications. A potential solution that would drastically reduce costs would be to deploy on a different blockchain, sidechain or a layer-2 solution (e.g. Polygon: *respond* would cost only $0.03).

5.1. Re-use of Proofs and Addresses

Apart from simplicity, assigning certificates to users' addresses provides one additional benefit. This design choice, allows for the re-use of proofs by users. For example, a user who has responded to a challenge in the past (e.g., age>18) doesn't have to prove this again, as a proof of this statement is now linked to their address. In the future, if they need to prove this statement to a different challenger, they can prove that they own an address (e.g., by signing a message) which is associated with a reply to this challenge instead of replying to a new challenge. This reduces the cost for commonly used proofs to a fixed initial cost rather than a cost per proof.

However, this approach also introduces some potential drawbacks. As all proofs are visible and directly linked to addresses, there is a risk that linking of proofs can be used to expose some attribute information. For example, if a user proves in one challenge that their age>18 and in another that their age<30, then an observer would be able to conclude that the user's age is in the range (18,30). We believe that this is not a big issue for the applications we are considering as usually all checks will be in the same direction (usually a user would only prove that age>12, age>18 etc). Additionally, users need to be careful when having certificates of different documents. If a user uses the same address across multiple certificates, an observer would be able to link these certificates together to identify a user. Moreover, even if the user is using different addresses between documents, users could be identified between different documents through network analysis of their transactions. However, we believe that enough privacy tools (such as mixers and relayers) are available that as long as users are careful this risk can be safely avoided.

5.2. Linking Document Issuance to Verification

An interesting use-case of our system is the linking of different documents for verification and issuance. The issuer of a secondary document (e.g. car insurance certificate) could be a challenger in a different document (e.g. driving licence). In this way, a user can verify that they hold some attributes, a valid driving licence in this case, to get a different document, without having to show the actual document and reveal their identity completely.

5.3. Age verification for online services

Age verification for online services is a topic of interest for ensuring child safety online. We can consider two forms of verification: the "simple" problem of age-checking for platform usage restrictions (e.g., verifying a Netflix account to be adult/minor); and the "hard" problem of allowing a user to anonymously prove they are an adult while browsing online (e.g., to access age-restricted services without browsing being traced back to a single user).

The current system can handle the simple problem as follows:

1. The online service provider first issues a challenge in a relevant document (smart contratc) that the user holds.
2. A user replies to the challenge once.
3. Every time the user wants to access the service, the service provider asks the user to sign a selected message using the address they used to respond to the challenge. Alternatively, the user could connect their wallet with the service provider's webpage, as it's commonly used in many applications.
4. The user signs the message.
5. The service provider checks if the address signing the message is in the list of replies, and if it is, allows the user to access the service.

The same solution can be used for the harder problem if the user is willing to reveal their address. However, in this case, the address acts as a pseudonym for the user and can be tracked and profiled. A user could change addresses regularly, but this would mean that they need to go through the verification process every time. To address the harder problem, the system would need adapting. The system would need to allow users to prove ownership of a certificate without revealing their address. This can be achieved by storing certificates in Merkle trees. The user in this case would need to respond to the service provider's

challenge every time they want to access the service; i.e., they would provide a ZKP that their certificate is in the Merkle tree and is suitable to access the online content.

6. Conclusions

We have introduced a blockchain-based framework for verifying user credentials in a privacy-preserving manner. Our framework, combines the best parts from previous systems, and is simple and flexible. Our main contribution, is that our system efficiently allows reuse of proofs. We open-source release a concrete implementation of the framework, and demonstrate how it can be used for tasks such as checking that a user holds a valid driving licence, without revealing anything else.

In future work, we plan to expand our system simulation to explore more realistic use cases and will attempt a real-world application for the sharing economy. While the sharing economy offers wider benefits to society (reducing waste, lowering carbon footprint, etc.), it offers specific benefits for deprived communities (e.g., areas with high unemployment) where people may have spare time but little spare money. Users are able to earn tokens in return for performing simple jobs (e.g., babysitting; cleaning; cooking a meal; collecting and delivering groceries; etc.), and these tokens can then be exchanged for needed goods and services (e.g., a lift to a medical appointment; rental of a power tool; a loaf of bread at a local store; etc.). Clearly, to enable the sharing economy to work safely, privacy-preserving credential verification is necessary.

References

Biryukov, A., Khovratovich, D., and Tikhomirov, S. (2018). Privacy-preserving KYC on Ethereum. In *Proceedings of 1st ERCIM Blockchain Workshop 2018*. European Society for Socially Embedded Technologies (EUSSET). https://dl.eusset.eu/handle/20.500.12015/3165.

Bitansky, N., Canetti, R., Chiesa, A., and Tromer, E. (2012). From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ITCS '12, page 326–349. https://doi.org/10.1145/2090236.2090263.

Camenisch, J. and Lysyanskaya, A. (2001). An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In Pfitzmann, B., editor, *Advances in Cryptology — EUROCRYPT 2001*, pages 93–118. https://doi.org/10.1007/3-540-44987-6_7.

Camenisch, J. and Van Herreweghen, E. (2002). Design and implementation of the idemix anonymous credential system. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, CCS '02, page 21–30. https://doi.org/10.1145/586110.586114.

Chaum, D. (1985). Security without identification: Transaction systems to make big brother obsolete. *Commun. ACM*, 28(10):1030–1044. https://doi.org/10.1145/4372.4373.

Fiege, U., Fiat, A., and Shamir, A. (1987). Zero knowledge proofs of identity. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, STOC '87, page 210–217. https://doi.org/10.1145/28395.28419.

Gabizon, A. and Williamson, Z. J. (2021). fflonk: a fast-fourier inspired verifier efficient version of plonk. Cryptology ePrint Archive, Paper 2021/1167. https://eprint.iacr.org/2021/1167.

Gabizon, A., Williamson, Z. J., and Ciobotaru, O. (2019). PLONK: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Paper 2019/953. https://eprint.iacr.org/2019/953.

Garman, C., Green, M., and Miers, I. (2013). Decentralized anonymous credentials. Cryptology ePrint Archive, Paper 2013/622. https://eprint.iacr.org/2013/622.

Goldwasser, S., Micali, S., and Rackoff, C. (1989). The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–208. https://doi.org/10.1137/0218012.

Grassi, L., Khovratovich, D., Rechberger, C., Roy, A., and Schofnegger, M. (2019). Poseidon: A new hash function for zero-knowledge proof systems. Cryptology ePrint Archive, Paper 2019/458. https://eprint.iacr.org/2019/458.

Groth, J. (2016). On the size of pairing-based non-interactive arguments. Cryptology ePrint Archive, Paper 2016/260. https://eprint.iacr.org/2016/260.

Gudymenko, I., Khalid, A., Siddiqui, H., Idrees, M., Clauß, S., Luckow, A., Bolsinger, M., and Miehle, D. (2020). Privacy-preserving blockchain-based systems for car sharing leveraging zero-knowledge protocols. In *2020 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPS)*, pages 114–119. https://doi.org/10.1109/DAPPS49028.2020.00014.

Lee, J., Choi, J., Oh, H., and Kim, J. (2021). Privacy-preserving identity management system. Cryptology ePrint Archive, Paper 2021/1459. https://eprint.iacr.org/2021/1459.

Li, W., Meese, C., Guo, H., and Nejad, M. (2020). Blockchain-enabled identity verification for safe ridesharing leveraging zero-knowledge proof. In *2020 3rd International Conference on Hot Information-Centric Networking (HotICN)*, pages 18–24. https://doi.org/10.1109/HotICN50779.2020.9350858.

Luong, D. A. and Park, J. H. (2023). Privacy-preserving identity management system on blockchain using zk-snark. *IEEE Access*, 11:1840–1853. https://doi.org/10.1109/ACCESS.2022.3233828.

Mukta, R., Paik, H.-Y., Lu, Q., and Kanhere, S. S. (2022). CredTrust: Credential based issuer management for trust in self-sovereign identity. In *2022 IEEE International Conference on Blockchain (Blockchain)*, pages 334–339. https://doi.org/10.1109/Blockchain55522.2022.00053.

Namazi, M., Ross, D., Zhu, X., and Ayday, E. (2022). zkFaith: Soonami's zero-knowledge identity proto-

col. arXiv:2212.12785. https://doi.org/10.48550/arXiv.2212.12785.

Pauwels, P. (2021). zkKYC: A solution concept for KYC without knowing your customer, leveraging self-sovereign identity and zero-knowledge proofs. Cryptology ePrint Archive, Paper 2021/907. https://eprint.iacr.org/2021/907.

Rosenberg, M., White, J., Garman, C., and Miers, I. (2022). zk-creds: Flexible anonymous credentials from zk-SNARKs and existing identity infrastructure. Cryptology ePrint Archive, Paper 2022/878. https://eprint.iacr.org/2022/878.

Sonnino, A., Al-Bassam, M., Bano, S., Meiklejohn, S., and Danezis, G. (2020). Coconut: Threshold issuance selective disclosure credentials with applications to distributed ledgers. https://doi.org/10.48550/arXiv.1802.07344.