Towards Smart Contract-based Verification of Anonymous Credentials

Robert Muth, Tarek Galal, Jonathan Heiss, and Florian Tschorsch

Technische Universität, Berlin, Germany
{muth,j.heiss,florian.tschorsch}@tu-berlin.de
tgalal@mail.tu-berlin.de

Abstract. Smart contracts often need to verify identity-related information of their users. However, such information is typically confidential, and its verification requires access to off-chain resources. Given the isolation and privacy limitations of blockchain technologies, this presents a problem for on-chain verification. In this paper, we show how CL-signature-based anonymous credentials can be verified in smart contracts using the example of Hyperledger Indy, a decentralized credential management platform, and Ethereum, a smart contract-enabled blockchain. Therefore, we first outline how smart contract-based verification can be integrated in the Hyperledger Indy credential management routine and, then, provide a technical evaluation based on a proof-of-concept implementation of CL-signature verification on Ethereum. While our results demonstrate technical feasibility of smart contract-based verification of anonymous credentials, they also reveal technical barriers for its real-world usage.

Keywords: blockchains · anonymous credentials · zero knowledge

1 Introduction

Blockchains are increasingly used to host decentralized applications (DApps), which are implemented as smart contracts. In contrast to centralized applications, each blockchain node maintains its own application copy. Local copies are synchronized through a consensus protocol that eventually establishes a globally consistent application state. This equips DApps with novel characteristics, e.g., transparency and censorship-resistance which make DApps interesting in various domains including decentralized finance [1], autonomous organizations [2], and the Internet of Things [3].

Interacting with DApps often requires their users to demonstrate identity-related information. For example, in DApps for crypto-token offerings, where an initial amount of free tokens is released to attract new community members, identity-related information is used to prevent malicious users from exploiting these *airdrops* through Sybil-attacks [4], i.e., repeatedly requesting free tokens using different blockchain accounts. Another example is provided by voting DApps, where voters are required to prove their eligibility through identity-related information, e.g., they need to be over age or live within a certain address range.

Unfortunately, in current practices, verification is often realized off-chain using centralized services. For example, during the Stellar airdrop[1], users were required to prove that they have a valid GitHub[2] account with a past registration date to prevent attackers from creating multiple GitHub accounts as their Sybils. The account validity has not been verified on-chain but through an allegedly trusted off-chain verifier. Similarly, in the Open Voting Network implementation for Ethereum [5], the voting initiator approves voters' eligibility off-chain and publishes an acceptance list of eligible voters to the smart contract. In both examples, a dishonest verifier can cheat without being noticed, e.g., by unjustifiably denying access to tokens or the voting.

In order to preserve decentralization and censorship-resistance of such DApps, the verification must be executed on-chain. This, however, introduces two new challenges: runtime isolation of DApps and privacy. Verifying identity-related information typically requires a trusted third party that vouches for the correctness of the information. In the airdrop example, it is not sufficient for the user to claim possession of a GitHub account, but instead GitHub itself must attest to it. Similarly, in the voting example, the required identity information could be attested to by a public institution. However, smart contracts run in an isolated execution environment and can, hence, only access information existent in the same runtime. To access off-chain information, they require an oracle that, however, implies trust [6].

Furthermore, verifying identity-related information typically reveals personal identifiable information to the verifier. This already presents a problem for off-chain usage. However, if the verification happens on-chain, sensitive information becomes accessible to unauthorized blockchain nodes and immutably anchored on-chain for an unknown period of time. Consequently, identity information cannot naively be verified on-chain to protect the users' privacy rights and comply with prevailing privacy regulations.

One approach towards privacy-preserving verification of identity information are *anonymous credentials*. They can be implemented by using zero-knowledge proofs to enable credential verification without revealing sensitive identity attributes to the verifier. While anonymous credentials already existed for long [7], they have increasingly gained attention. For example, they are part of Hyperledger Indy [8] a decentralized credential management system where they are realized based on Camenisch-Lysyanskaya (CL) signatures [7].

Anonymous credentials present a promising solution to overcome the blockchain's privacy limitations and enable non-revealing verification of identity-related information. But anonymous credentials verification happens as part of an interactive procedure among the identity issuer, holder, and verifier. This clashes with a smart contract's limited communication capabilities with off-chain resources. Furthermore, smart contracts are constrained by technical peculiarities that originate from the underlying execution environment, e.g., the Ethereum Virtual Machine [9], and must be considered when implementing the verification of anonymous credentials. To date, no smart contract-based implementation exists for verifying CL signature-based anonymous credentials.

[1] https://www.coindesk.com/business/2019/12/13/stellar-tried-to-give-away-2b-xlm-tokens-on-keybase-then-the-spammers-came/

[2] https://github.com

In this paper, we propose a mechanism for smart contract-based verification of anonymous credentials that are issued as part of the Hyperledger Indy credential management routine and, thereby, make the following contributions:

- We propose a procedure for integrating smart contract-based credential verification into Hyperledger Indy. The procedure connects two previously disconnected worlds and, thereby, allows blockchain-enabled DApps to verify anonymous credentials originating from Hyperledger Indy-based systems.
- We present a technical specification that explains the verification of CL-signature-based anonymous credentials in a developer-friendly way. The specification is based on formal descriptions [10] on the one hand, and insights gained from an analysis of the Hyperledger Indy code repository[3] on the other hand.
- We provide a proof-of-concept implementation for verifying CL signatures-based anonymous credentials on Ethereum in Solidity, and we document technical challenges that we encountered during implementation. Our evaluation demonstrates the technical feasibility of smart contract-based verification of anonymous credentials, but it also reveals technical barriers for its real-world usage.

The remainder of this paper is structured as follows: In Section 2, we give an overview of anonymous credentials in Hyperledger Indy and show how smart contract-based credential verification can be integrated with the Hyperledger Indy procedure. In Section 3, we take a look at the verification of CL signature-based anonymous credentials. Our proof-of-concept implementation is described and evaluated in Section 4, and open issues are discussed in Section 5. In Section 6, we present related work and conclude the paper in Section 7.

2 Anonymous Credentials

To set the scene, we first introduce concepts and roles related to anonymous credentials in Hyperledger Indy. Based on this overview, we propose a procedure for integrating smart contract-based verification of anonymous credentials into Hyperledger Indy's standard credential management routine [11].

2.1 Anonymous Credentials in Hyperledger Indy

Credentials can be understood as a set of claims about its holder's identity [12], i.e., statements about specific identity attributes such as the holder's name, address, or date of birth. Hyperledger Indy [8] provides a decentralized system for managing such credentials that is built upon principles of the self-sovereign identity paradigm [13] where, instead of letting third parties control a holder's identity claims, the holder is in control. To independently verify a credential, additional evidence is required. The evidence is typically attached to the credential by a trusted third party that is able to attest to the holder's attributes, e.g., in form of a cryptographic signature. This makes them *verifiable credentials* [14].

[3] https://github.com/hyperledger/indy-sdk

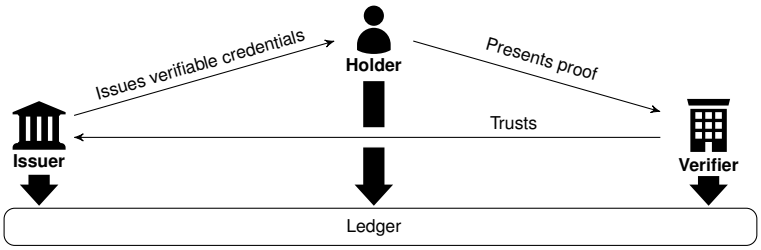


Fig. 1. SSI Model adopted from [12].

However, a naive verification of signatures constructed on identity attributes could reveal potentially sensitive identity attributes to the verifier which may violate the holder's privacy rights. The concept of *anonymous credentials* [10] addresses this problem by enabling selective disclosure for credential verification, i.e., verifying predicates on sensitive identity attributes without revealing them to the verifier. Therefor, Hyperledger Indy utilizes Camenisch-Lysyanskaya signatures (CL) [15], a signature scheme with zero-knowledge properties.

As depicted in Figure 1, credential verification requires three roles: the identity *holder* who owns an identity, the *issuer* that issues and attests to identity attributes through verifiable identity claims, and the *verifier* that verifies the identity claims. In Hyperledger Indy, additionally, a public, permissioned blockchain, the *ledger*, is applied as a public and decentralized storage system to make public artifacts accessible to all involved parties. Private credentials of holders are stored in personal wallets that keep them protected from unauthorized access.

2.2 Smart Contract-based Integration of Anonymous Credentials Verification

In the following, we show how anonymous credentials of Hyperledger Indy can be used for smart contract-based verification, i.e., Ethereum smart contracts. In order to establish compatibility between Hyperledger Indy and Ethereum we define the following requirements:

- No infrastructure modifications: Smart contract developers should be able to draw on existing Hyperledger Indy systems for credential verification. Therefore, on-chain verification should integrate without any modification to the native Hyperledger Indy system.
- No further trust assumption: In Hyperledger Indy, issuers are trusted by the verifier to truthfully attest to a holder's credentials, and no further trust assumption should be introduced. More precisely, integration should work without trusted oracles [6].

The integration procedure builds upon the native Hyperledger Indy credential verification flow described in [11]. As an extension, we propose a new verifier role, i.e., the *verifier contract*, and introduce the *developer* that is responsible for implementing and deploying the smart contract. As depicted in Figure 2, the complete proof verification procedure can be divided in three phases: setup, proving, and verification.

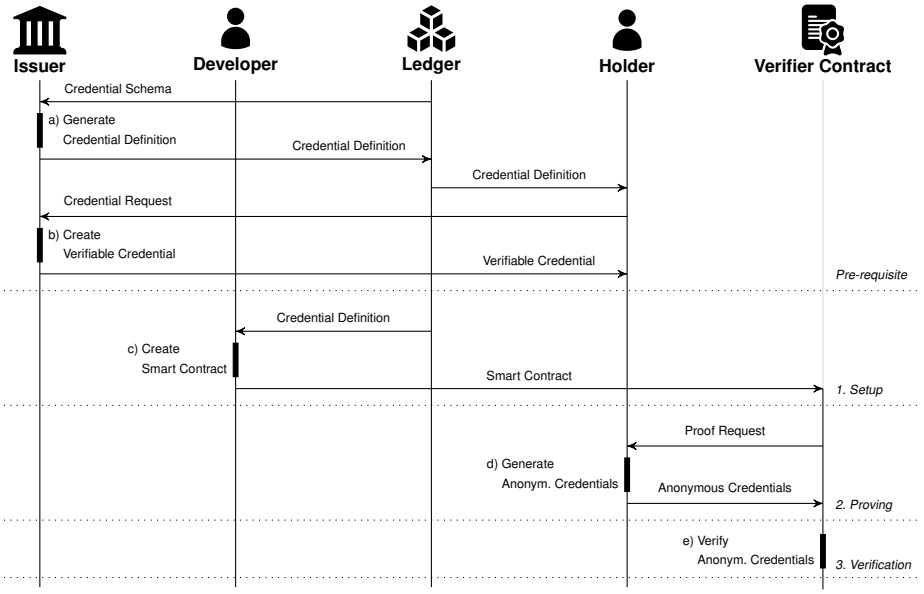


Fig. 2. Adopted proof verification flow of the Hyperledger Indy proof verification procedures with a smart contract verifier.

Pre-requisite: As a pre-requisite, we assume that a *credential schema* exists that defines a set of identity attributes. From that, the issuer has generated a *credential definition* and registered it on the Hyperledger Indy ledger (Step a). The credential definition is a public artifact that is typically specified by the issuer and accessible on the ledger. Among others, it contains the set of attributes to be verified, the issuer's public key, and a reference to the applicable signature algorithm which, in our case, is the CL-signature scheme.

Furthermore, we assume that the holder is already in possession of the verifiable credential that is required by the smart contract. In Hyperledger Indy, this is done in the form of a *credential request* [11] submitted from the holder to the issuer. On receiving the request, the issuer creates the evidence, here the CL-signature, and returns the verifiable credential to the holder where it is stored in her wallet (Step b).

1. **Setup**: During the initial one-time setup, the developer determines the set of attributes and predicates to be verified on-chain and the attesting issuer in the form of a *proof request*, which in Hyperledger Indy is typically created by the verifier. Correspondingly, she selects an appropriate credential definition from the ledger. For simplicity, we assume that the determined proof request matches a single credential definition. Based on the issuer's public key, attributes, predicates, and a reference to the credential definition, the developer creates the smart contract (Step c). In Hyperledger Indy, these information are contained in the credential definition and the proof request. Then, the developer deploys the smart contract to the blockchain.

2. **Proving**: The holder obtains the proof request from the smart contract which also includes a reference to the credential definition on the ledger. Based on the credential definition and schema obtained from the ledger as public information, and the verifiable credential taken from the wallet as private information, the holder constructs a zero-knowledge proof based on her CL-signature (Step d) to obtain anonymized credentials. Finally, the holder submits the resulting anonymous credentials to the smart contract.
3. **Verification**: On reception, the smart contract (i.e., DApp) verifies the proof using the on-chain credential definition (Step e).

Successfully verified credentials can then be used as part of the DApp's logic. As shown by the motivating examples, the credential verification can represent a precondition for using specific functionality provided by the DApp, e.g., a voting or an airdrop.

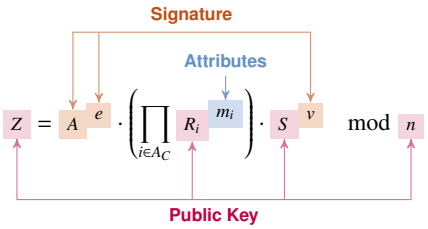
3 Proof Verification

Given the high level description of integrating smart contract-based anonymous credential verification into the native Hyperledger Indy credential management routine, we can now focus on the details of proof verification. Therefore, we first introduce the basics of CL-signature verification that, in Hyperledger Indy, are used to construct Zero-Knowledge Proofs (ZKP). Based on this, we describe the actual anonymous credentials proofs, i.e., a primary proof that builds on equality proofs and inequality predicate proofs.

3.1 Signature Proof of Knowledge

Conceptually, ZKPs in anonymous credentials prove knowledge of some discrete logarithms modulo a composite [10], and are referred to as *Signature Proofs of Knowledge*. They enable a credential holder to prove possession of a CL-signature over certain attribute values without revealing other attributes, as well as, to prove that an attribute value lies within a certain range without revealing it. In order to understand how this is accomplished in anonymous credentials, we give an overview of what a CL-signature looks like, and show how a signature proof of knowledge is generated for it. We do this by referencing corresponding steps in the procedure presented in Section 2.2.

As part of the pre-requisites, the credential issuer creates the credential definition (Step a). Therefore, she generates a CL-signature key pair based on a credential schema which contains a set of predefined attributes (e.g., attributes of a driver's license). The CL-signature scheme [15] defines the public key in this key pair as the quadruple $(Z, S, \{R_i\}_{i \in A_C}, n)$ where A_C is the set of indices of attributes in the credential schema, n is a Special RSA Modulus [15], and $Z, S, \{R_i\}$ are random quadratic residues modulo n. Then, on receiving a credential request from the holder, the issuer attests to the attribute values $\{m_i\}$, and, for creating a verifiable credential (Step b), generates a CL-signature as explained in [15], such that the following holds:



$$Z = A^{e} \cdot \left(\prod_{i \in A_C} R_i^{m_i} \right) \cdot S^{v} \mod n \tag{1}$$

While the public key information $Z, \{R_i\}, S$ and n are publicly available on the ledger, the signature (A, e, v) can only be calculated by the issuer, who owns the private key, in order to keep the whole equation true. Together with the values of the attributes $\{m_i\}_{i \in A_C}$, the holder is now able to prove possession of the credential. Therefore, she generates a ZKP (Step d) which proves knowledge of the exponents $e, \{m_i\}_{i \in A_C}, v$ but keeps them secret when presenting the proof to a verifier. By doing that, the verifier can still verify that the prover knows those exponents, and thereby, is in possession of a valid signature (Step e).

Additionally, a holder can prove that a credential contains an attribute with a specific value that she reveals. For this, we say $A_C = A_r \cup A_{\bar{r}}$ such that A_r contains revealed attributes and $A_{\bar{r}}$ contains unrevealed attributes which are kept secret. Since A_r and $A_{\bar{r}}$ are mutually exclusive, we can adjust Equation 1 as follows

$$Z = A^{e} \left(\prod_{i \in A_r} R_i^{m_i} \right) \left(\prod_{i \in A_{\bar{r}}} R_i^{m_i} \right) S^{v} \mod n \tag{2}$$

$$\frac{Z}{\left(\prod_{i \in A_r} R_i^{m_i} \right)} = A^{e} \left(\prod_{i \in A_{\bar{r}}} R_i^{m_i} \right) S^{v} \mod n \tag{3}$$

This allows us to prove that a credential contains a set of attributes with the revealed values $\{m_i\}_{i \in A_r}$, by proving knowledge of the exponents $(e, \{m_i\}_{i \in A_{\bar{r}}}, v)$ in Equation 3. We use these insights as a basis for the following proof descriptions.

3.2 Primary Proof Verification

Anonymous credentials, as explained in [16], are based on the previously introduced CL-signature proofs of knowledge. Those proofs are similar to the Schnorr Protocol [17], and are made non-interactive (i.e., only one round instead of commitment, challenge, and response) by implementing the Fiat-Shamir Heuristic [18]. Anonymous credentials structures a proof as a combination of different sub-proofs belonging to a *primary proof* that is used for verifying them altogether. To this end, a primary proof consists of a set of equality sub-proofs $\{\text{Pr}_C\}$ and a set of inequality predicate sub-proofs $\{\text{Pr}_P\}$:

- An *Equality proof* proves that a credential contains expected values
- An *Inequality predicate proof* proves that a credential contains a value that lies within a certain range (e.g., zip code between a given range)

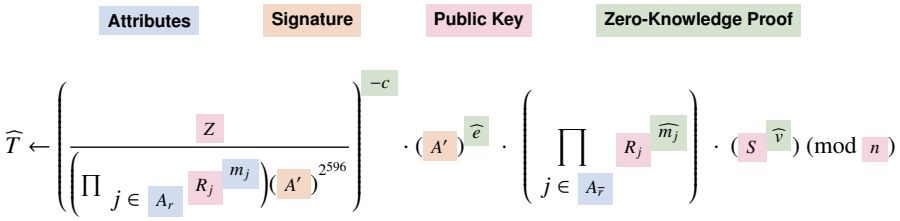
Additionally, a primary proof also contains a set C with necessary information for the non-interactive ZKP execution, and a ZKP challenge c to verify the correct execution. Along that, the verifier requires the proof generator (i.e., the holder) to include a given nonce η, so that the verifier can make sure that the proof corresponds to a particular proof request. During verification, sub-proofs are processed one after the other, and their results are appended to a set $\widehat{\mathcal{T}}$ which is shared across all the sub-proofs. At the end, the ZKP responses in each sub-proof are individually processed and the results are aggregated into $\widehat{\mathcal{T}}$. Once $\widehat{\mathcal{T}}$ is complete, the verifier hashes the final result of $\widehat{\mathcal{T}}$, C, and the nonce η. The proof verification succeeds if c equals $H(\widehat{\mathcal{T}} \parallel C \parallel \eta)$.

3.3 Equality Proof Verification

In the following, we introduce equality proof verification. We therefore present the cryptographical procedure, as we did for the signature proof of knowledge introduction in Section 3.1. In fact, equality proof verification is based on the same technique, but is implemented over a randomized version of the CL-signature [10]. However, we do not intend to recap all proof details, as they are explained in [10].

In a nutshell, an equality proof attests the possession of a CL-signature (i.e., credential) over a set of expected attributes, but without revealing any other information. Still, A_r contains the indices of revealed attributes and $A_{\bar{r}}$ contains the unrevealed attributes. The difference is, a verifier receives $\mathrm{Pr}_C = (\widehat{e}, \widehat{v}, \{\widehat{m_j}\}_{j \in \bar{r}}, A')$ and the revealed attribute values $\{m_j\}_{j \in A_r}$ from a prover, but the original values A, e, v (cf., Equation 1) are kept secret. Therefore, A' belongs to a randomized CL-Signature (A', e', v') [10] where e', v' are exponents in that signature. The prover generates this randomized signature based on the original CL-signature she possesses by following the procedure described in [10] to guarantee unlinkability across different proofs.

For completeness, we present the equality proof equation which calculates the sub-proof results $\widehat{T}$. For the sake of simplicity and easy recognition of relevant components, we highlighted the parts in the equality proof equation that belong together:



In the end, computing $\widehat{T}$ is a pre-verification step for the given ZKP parameters, and is completed as part of the primary proof verification. At this point, we want to underline that the same c is used for exponentiation as for the final hash comparison in the primary proof.

3.4 Inequality Predicate Proof Verification

An inequality predicate consists of an attribute, one of $>, \geq, <, \leq$ as a comparison operator, and a constant value to compare to. Once again, the credential holder proves that a specified inequality is satisfied without revealing the actual value of the attribute. In order to do that, the prover constructs a zero knowledge proof that the inequality predicate is satisfied. Additionally, the prover attests that the attribute indeed belongs to her, by constructing another zero knowledge proof. This zero knowledge proof is in fact just an equality proof that does not reveal the attribute's value.

Verifying an inequality proof requires processing the associated equality proof first where the index of the attribute in the predicate belongs to $A_{\tilde{r}}$ (unrevealed attributes). Afterwards, predicate-specific zero knowledge proofs are processed, each extending $\widehat{\mathcal{T}}$ in the primary proof verification. As the computations performed for processing an inequality predicate proof closely resemble those of an equality proof, we leave them out of this section and refer to the anonymous credentials specification [16].

4 Implementation

In this section, we introduce our proof-of-concept implementation for on-chain anonymous credential verification. We therefore present our smart contract implementation and give an overview of its technical design. In reference to Section 2, we show how the complex proof verification can be implemented and executed with limited and isolated EVM resources. At the end, we evaluate the transaction costs for proof verification with our implementation.

4.1 Proof-of-Concept

As part of our implementation for anonymous credentials verification, we provide a Truffle Suite project with a single smart contract implementation for the proof verification. In addition to that, we include Mocha (Node.js) test cases for single proof verifications and full proofs with combined equality and predicate proofs. We consider our implementation a proof-of-concept, since we limit it to verification only and focus on the technical feasibility. We also stress, that our implementation should not be considered production ready, yet. The code is available on Github[4].

We use the Hyperledger Ursa cryptography library[5] as a reference implementation and we replicate relevant parts of their test cases to ensure our implementation generates the same results. We also provide additional test cases which are used for our evaluation.

For the smart contract development, we face several challenges regarding the CL-signatures' key size. Firstly, proof verification requires arithmetic operations $(+, -, \cdot)$, exponentiation, and multiplicative inversion modulo a large number. As explained in Section 3.3, n is defined by the public key as the modulus. In our case, anonymous credentials and the implemented test cases use n of size 2050 bits[6], which exceeds the size

[4] https://github.com/robmuth/eth-ac-verifier

[5] https://github.com/hyperledger/ursa/blob/34ef392/libursa/src/cl/prover.rs#L2532

[6] n is 3074 bits in anonymous credentials [16], but HyperLedger Ursa sets n to 2050 bits.

of the EVM's largest data type for numbers (max. 256 bits for unsigned integer). To this end, we integrate a big number library[7] for on-chain computations which ports parts of the OpenSSL big number implementation to Solidity and YAL (inline assembly). The library receives numeric inputs as byte arrays and executes computations in the EVM memory space. With EIP-198 [19] the EVM offers a precompiled contract for computing $a^b \mod n$ where a, b and n can be 256-bit unsigned integers. It is noteworthy that the big number library takes full advantage of this precompiled contract, as it also uses it for efficient multiplication.

Additionally, the big number library and the precompiled contract do not support computing modular multiplicative inverses. However, given a, b, n, it is possible to check if a is the inverse of b modulo n by utilizing the available exponentiation. Our implementation therefore computes the required modular multiplicative inverses off-chain and passes them together with the proof. For this reason, we use another big number library for off-chain computations in JavaScript[8] which pre-computes intermediary results (i.e., modular multiplicative inverses) for a proof verification. As explained above, our implementation verifies that the passed values are correct and aborts the execution otherwise.

Lastly, the number of variables necessary for proof calculations leads to capacity shortages for the Solidity compiler. The EVM is designed as a stack machine [9] with a maximum size of 1 024 words (each 256 bits), which is just enough to pass a complete proof in a transaction. Unfortunately, the limit of 16 parameters and local variables per function call renders all proof calculations difficult, since our required parameters and calculations (mainly big integer computations) cause stack overflow exceptions during compilation. Therefore, and for better code readability, we utilize pre-defined *struct* data structures to pack multiple parameters, and split computations into multiple functions which allocate and release local variables from the stack for interim calculations.

The Truffle project provides basic migration scripts for the verification contract and the linked big number library. The test cases provide unit tests for the verification procedures, as well as, exemplary credentials. Once passed, they return the corresponding transaction costs for each verification in gas, which we evaluate in the following.

4.2 Evaluation

We evaluate the costs for the deployment of the smart contracts and for transactions created by test cases with exemplary credentials. We therefore compile the smart contracts with Solidity 0.5.16 (enabled optimizer with 200 runs), and analyze the transactions with Ganache 2.5.4. We implement the Hyperledger Ursa cryptography library unit tests into our PoC test cases and additionally generate our own exemplary anonymous credentials, issued with Hyperledger Indy.

Table 1 shows gas costs for the deployment of the smart contracts and function calls of our test cases. While the deployment of a verification contract only puts the compiled byte code on the blockchain storage, executing a proof verification requires passing a full proof to the smart contract and executing the anonymous credentials verification

[7] https://github.com/firoorg/solidity-BigNumber
[8] https://github.com/indutny/bn.js

Table 1. Transaction costs in Gas for smart contract deployments and different proof verification test cases in Ethereum.

Transaction	**Gas** $\cdot 10^3$
Verifier contract deployment	6 711
Big number library deployment	77
1. Test credential: Primary proof verification with an equality sub-proof	32 001
2. Test credential: Primary proof verification with an inequality predicate sub-proof	84 823
3. Test credential: Primary proof verification with combined sub-proofs	84 031

process as explained in Section 3.2. In our first test, the credential contains a set of 14 identity-related attributes (e.g., name, date of birth, address, etc.) and reveals the first name with a primary proof and an equality sub-proof. Our second test contains the same attributes and an inequality predicate (i.e., date of birth has to be before a specified date), which is realized as a combination of an equality sub-proof and an inequality predicate sub-proof in the primary proof. By looking at the difference between the gas cost for the first and second test credentials, one can see that the gas costs increase with the number of sub-proofs, especially with inequality predicate proofs. This explains why the gas costs of the third test, which reveals the first name and verifies the date of birth together, is not significantly different from the second test case.

Since gas represents the resource consumption per EVM command [9], proof verification becomes expensive for three reasons: First, the passed arguments (i.e., the proofs) contain large byte arrays, e.g., for each attribute and the corresponding zero-knowledge proof parameters. Second, the complex data structures for handling big numbers require allocation of EVM memory space, which consumes extra gas [9]. Third, calling the precompiled contract for exponentiation and modulo operations [19] allows cheaper computations than an on-chain implementation; but due to the involvement of a large number of these operations in a proof verification, the gas cost add up quickly. We also point out, that the number of attributes influences the proof size and therefore has an impact on the transaction costs, as well.

In the end, our evaluations show that the gas costs are very high, so the resulting transaction fees (i.e., gas multiplied by the demand-regulated gas price) render it difficult for most DApp use cases. Considering a current exemplary gas price of 100 Gwei, the full proof verification of our test case would cost approx. 8.4 Ether. However, we stress that our proof-of-concept implementation is not optimized for reduced gas costs and there is certainly great potential for optimization (e.g., more efficient memory allocations). Furthermore, gas costs cannot be translated directly to transaction fees in Ethereum, since a transaction sender specifies how much Ether she is willing to pay per gas. This means, the actual transaction costs depend on the blockchain's current transaction load.

5 Discussion

In this section, we discuss remaining open issues that should be considered for practical usages of our proposed solution. To this end, we put the evaluated transaction costs into

context and present further options to implement our solution. For completeness, we also take a look at revocability of credentials and explain how it could be integrated into our smart contract implementation. At the end, we briefly discuss our considerations regarding Sybil resistance and unlinkability.

5.1 Transaction Costs

As the evaluation in Section 4.2 shows, our proof-of-concept demonstrates the general ability to verify anonymous credentials in a smart contract on Ethereum, and therefore enables compatibility with Hyperledger Indy-based identity platforms. However, the expected transaction costs are too high for current DApp implementations.

Nevertheless, blockchain technologies continue adopting new techniques to address rising transaction costs, scalability limits, and performance issues of execution engines, respectively [20]–[22]; Especially since hype-driven blockchain applications (e.g., Cryptokitties or ICOs) caused fees to skyrocket [23] multiple times, in the past. The constantly increasing block size limit in Ethereum also suggests that the overall demand will continuously increase. However, since our implementation is EVM-based but not limited to the Ethereum Mainnet, we envisage other blockchains that are compatible with our Solidity implementation. For example, Polygon/Matic[9] enables second execution environments which has the potential to decrease the cost of our verification implementation. As well as, Polkadot[10] which is a based on a past EVM fork and uses proof-of-stake as consensus algorithm with a fundamental different transaction pricing.

In the meantime, we consider two possible ways to implement anonymous credentials verification on-chain: First, the computationally expensive big number operations could be implemented as precompiled contracts into the EVM, i.e., implementing big number data types. Doing so, the overall transaction costs of our implementation could significantly decrease in the same way, as EIP-198 [19] decreases costs for big integer modulo operations. Second, in the same manner, the whole verification procedure could be implemented as a precompiled contract. The latter, we at least consider reasonable for instantiating new EVM-based blockchains with anonymous credentials capabilities, since implementing precompiled contracts is common practice for new private networks.

5.2 Non-Revocation Proof

Our solution works for credentials that are generally valid, that is, credentials over attributes that neither have an expiry date nor are revocable by their issuer (e.g., date of birth). In contrast, an issuer can create *revocable credentials* for attributes that are subject to change (e.g., address). In this case, it is possible that a revocable credential could have already been revoked by its issuer, by the time a verifier receives a proof based on

[9] https://polygon.technology
[10] https://polkadot.network

it. Therefore, a verifier must be able determine a credential's revocation status in order to accordingly decide whether to accept or reject the given primary proof.

Anonymous credentials in [16] allow a holder to prove that the credentials which were used during the primary proof construction have not been revoked. To this end, tracking of the revocation status utilizes CKS accumulators [24] which is based on cryptographic primitives different from the ones used by credential attestation proofs. Namely, it makes use of weak Boneh and Boyen signatures [25], [26], and is based on BN-254 curves [27] and type-3 pairing. The process of checking the revocation status takes place as part of the proof verification process. Thus, in addition to the primary proof, a credential holder also sends a non-revocation proof, which is a zero-knowledge proof constructed over accumulator parameters. In order to incorporate verification of non-revocation proofs into our implementation, the verifier would need to add a so-called *accumulator value*, which is explained in [24].

5.3 Sybil Resistance and Unlinkability

In our system, Sybils are re-submissions of already verified anonymous credentials by different holders. Since Sybil attacks can effectively be defeated by a trusted party that guarantees uniqueness [4] we can instrument the only trusted party for that purpose: the issuer. If the issuer provides a unique identifier as an identity attribute, it can be verified on-chain as part of the proof. Thereby, a verifiable one-to-one mapping of proof and holder is established. The unique identifier could be represented as a revealed attribute that is part of the proof construction and verifiable on-chain.

An alternative solution that does not require cooperation from issuers is described in [16]. A verifier can require provers to turn off unlinkability in their proofs. We have shown in Section 3.3 that this is possible if a holder does not generate a randomized CL-signature and instead, constructs proofs based on her original CL-signature. Thus, the smart contract keeps track of all proof identifiers and, hence, can identify Sybils.

However, while storing unique identifiers of proof-holder mappings helps to defeat Sybil attacks, it also introduces linkability. Linkability is a privacy-related characteristic and applies in this context if the linkage of anonymous credentials allows unauthorized third parties to derive private information of the holder. In smart contract-based applications, linkability is critical since the history of blockchain transactions is available to all blockchain nodes.

While holders can protect against linkability by using different blockchain accounts each time they interact with the blockchain, this protection becomes ineffective if unique identifiers of holders are stored on-chain. They should, consequently, not naively be used for Sybil resistance but require further considerations to keep holders protected from linkability.

6 Related Work

Smart contract-based verification of anonymous credentials can be regarded as an instantiation of Verifiable Off-chain Computations (VOC) [28], where an on-chain computation, here the credential verifications, is delegated to an off-chain node, here the

holder, and only the computational result and a cryptographic proof attesting the computation's correctness are published to the blockchain for verification. In the context of VOCs, the CL-signature-based approach complements other applicable non-interactive zero-knowledge technologies, e.g., zkSNARKs, Bulletproofs, or zkSTARKs [28].

In literature, various papers can be found where zkSNARKs are applied for privacy preserving authentication in DApps, e.g., through ZoKrates [29], a toolkit for zkSNARKs-based VOC. Examples are provided in the context of smart electric vehicle authentication at charging stations [30], patient authentication in health care [31], or consent-based access control [32]. However, these works focus on a particular use case where a particular claim of a particular issuer is verified on-chain. In contrast, our approach enables smart contracts to verify Hyperledger Indy-based credentials and, thereby, to reach multiple issuers.

Hyperledger Indy has become one of the most popular SSI management technologies [33] and is adopted in multiple SSI management projects. IDUnion[11], for example, is a German consortium of public and private organizations that uses Hyperledger Indy to implement a SSI management system. Similarly, the Verifiable Organizations Network[12] is an initiative that leverages Hyperledger Indy to realize SSI and enable digitization of identities in a secure, user-centric manner. With our proposed solution, smart contracts can now verify credentials of any issuer that is part of these projects.

Beyond Hyperledger Indy, other blockchain-based credential management systems exist. In the context of SSI, uPort [34] and Jolocom [35] are two approaches that, instead of building upon a public, *permissioned* blockchain, leverage Ethereum as a public, *permissionless* blockchain. These approaches, however, currently only store non-revealing identity information on-chain but do not provide anonymous credential verification. Given the privacy requirement, this makes them not applicable for smart contract-based credential verification. However, recent improvement proposals (e.g., EIP-725 [36] and EIP-735 [37]) show that there is an interest in establishing Ethereum-based SSI-Systems, which has even yielded the formation of a self-proclaimed SSI alliance[13].

Furthermore, Public Key Infrastructures (PKI) exist as centralized credential management systems, and are traditionally used for managing identity card infrastructures. For example, PKIs are used by universities to manage digital student IDs. On a national scale, the German Identity Cards are managed based on PKIs[14], and on an international scale, the ICAO uses PKIs to manage the global passport infrastructure[15]. However, in contrast to a Hyperledger Indy, in centralized credential management systems identity attributes are limited and can only be attested by a single issuer.

[11] https://idunion.org
[12] https://vonx.io
[13] https://erc725alliance.org
[14] https://www.bsi.bund.de/EN/Topics/ElectrIDDocuments/securPKI/pki_node.html
[15] https://www.icao.int/Security/FAL/PKD/Pages/default.aspx

7 Conclusion

Motivated by prevailing limitations observed in different DApp contexts, in this paper, we have made a first step towards verifying identity-related information in DApps in a privacy-preserving and trustless manner. Using the example of Hyperledger Indy as a popular credential management system, we have shown how CL signature-based anonymous credentials can be verified by Ethereum-based smart contracts without introducing further trust assumptions.

With our approach, DApps can verify identity-related information entirely on-chain and, thereby, preserve transparency and censorship resistance as desirable characteristics of DApps. In the airdrop example, account ownership claims issued by GitHub as anonymous credentials could be fully verifiable on-chain and used to mitigate illegitimate token spendings. Similarly, in the voting example, a public authority could issue eligibility claims as anonymous credentials which are on-chain verifiable and, thereby, help voting DApps yield more reliable results.

However, we also revealed aspects that stand between our prototype and a production-ready system. As seen in our technical evaluation, gas costs are currently impractically high due to high verification costs of CL signature-based zero-knowledge proofs. Furthermore, for verifying revocable identity attributes, a revocation system must be integrated into our solution. We also point out that security- and privacy-related aspects as Sybil-resilience and linkability must still be considered for real-world deployments.

The active research and ongoing developments in the field of verifiable credentials and smart contracts, however, let us expect technological advances to emerge from which smart contract-based credential verification may benefit. As one example, a new anonymous credential design, called Anonymous Credentials 2.0, has been proposed in [38] that might be adopted in Hyperledger Indy in the future. However, while it promises more efficient zero-knowledge proofs and, hence, cheaper on-chain verification, we were not able to find public information about the current development state or release dates. In the end, we have shown that CL-signature-based anonymous credentials can be verified in smart contracts and, at the same time, paved the way for the integration of further identity infrastructures and technologies.

References

[1] K. Qin, L. Zhou, Y. Afonin, L. Lazzaretti, and A. Gervais, “Cefi vs. defi - comparing centralized to decentralized finance,” *CoRR*, vol. abs/2106.08157, 2021.

[2] C. Jentzsch, “Decentralized autonomous organization to automate governance,” *White paper*, 2016.

[3] M. S. Ali, M. Vecchio, M. Pincheira, K. Dolui, F. Antonelli, and M. H. Rehmani, “Applications of blockchains in the internet of things: A comprehensive survey,” *IEEE Communications Surveys Tutorials*, vol. 21, no. 2, pp. 1676–1717, 2019.

[4] J. R. Douceur, “The sybil attack,” in *IPTPS*, ser. Lecture Notes in Computer Science, vol. 2429, Springer, 2002.

[5] P. McCorry, S. F. Shahandashti, and F. Hao, "A smart contract for boardroom voting with maximum voter privacy," in *Financial Cryptography*, ser. Lecture Notes in Computer Science, vol. 10322, Springer, 2017, pp. 357–375.
[6] J. Heiss, J. Eberhardt, and S. Tai, "From oracles to trustworthy data on-chaining systems," in *IEEE International Conference on Blockchain*, 2019.
[7] J. Camenisch and A. Lysyanskaya, "An efficient system for non-transferable anonymous credentials with optional anonymity revocation," in *EUROCRYPT*, ser. Lecture Notes in Computer Science, 2001, pp. 93–118.
[8] Hyperledger White Paper Working Group, *An introduction to hyperledger*, https://www.hyperledger.org/wp-content/uploads/2018/07/HL_Whitepaper_IntroductiontoHyperledger.pdf, 2018.
[9] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger, berlin version fabef25," 2021.
[10] J. Camenisch and T. Gross, "Efficient attributes for anonymous credentials," 2008, pp. 345–356.
[11] H. I.-s. Repository, *Indy walkthrough - a developer guide for building indy clients using libindy*, https://github.com/hyperledger/indy-sdk/blob/master/docs/getting-started/indy-walkthrough.md, 2018.
[12] A. Mühle, A. Grüner, T. Gayvoronskaya, and C. Meinel, "A survey on essential components of a self-sovereign identity," *Comput. Sci. Rev.*, vol. 30, 2018.
[13] Christopher Allen, *The path to self-sovereign identity*, http://www.lifewithalacrity.com/2016/04/the-path-to-self-soverereign-identity.html, 2016.
[14] World Wide Web Consortium (W3C), *Verifiable credentials data model v1.1 - expressing verifiable information on the web*, https://www.w3.org/TR/vc-data-model/, 2021.
[15] J. Camenisch and A. Lysyanskaya, "A signature scheme with efficient protocols," Jan. 2002, pp. 268–289.
[16] D. Khovratovich and M. Lodder, *Anonymous credentials with type-3 revocation*, https://github.com/hyperledger/ursa-docs/blob/62bc87b/specs/anoncreds1/anoncreds.tex, 2018.
[17] C. Schnorr, "Efficient signature generation by smart cards," *Journal of Cryptology*, vol. 4, pp. 161–174, Jan. 1991.
[18] A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems," in *Advances in Cryptology*, Springer, 1987, pp. 186–194.
[19] V. Buterin, *Big integer modular exponentiation*, https://github.com/ethereum/EIPs/blob/master/EIPS/eip-198.md), 2017.
[20] S. Popov, "IOTA: Feeless and free," *Blockchain Technical Briefs*, 2019.
[21] T. Roughgarden, "Transaction fee mechanism design for the ethereum blockchain: An economic analysis of EIP-1559," *CoRR*, vol. abs/2012.00854, 2020.
[22] A. Busse, J. Eberhardt, and S. Tai, "EVM-Perf: High-precision EVM performance analysis," in *IEEE International Conference on Blockchain and Cryptocurrency*, 2021, pp. 1–8.

[23] M. Spain, S. Foley, and V. Gramoli, "The impact of ethereum throughput and fees on transaction latency during icos," in *Tokenomics*, ser. OASIcs, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[24] J. Camenisch, M. Kohlweiss, and C. Soriente, "An accumulator based on bilinear maps and efficient revocation for anonymous credentials.," *IACR Cryptology ePrint Archive*, vol. 2008, p. 539, Jan. 2008.

[25] D. Boneh and X. Boyen, "Short signatures without random oracles and the SDH assumption in bilinear groups," 2, vol. 21, 2008, pp. 149–177.

[26] D. Boneh, X. Boyen, and H. Shacham, "Short group signatures," in *CRYPTO*, ser. Lecture Notes in Computer Science, vol. 3152, Springer, 2004, pp. 41–55.

[27] P. Barreto and M. Naehrig, "Pairing-friendly elliptic curves of prime order," vol. 3897, Aug. 2005, pp. 319–331.

[28] J. Eberhardt and J. Heiss, "Off-chaining models and approaches to off-chain computations," in *Proceedings of the 2Nd Workshop on Scalable and Resilient Infrastructures for Distributed Ledgers*, ser. SERIAL'18, ACM, 2018.

[29] J. Eberhardt and S. Tai, "Zokrates - scalable privacy-preserving off-chain computations," in *IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, 2018, pp. 1084–1091.

[30] D. Gabay, K. Akkaya, and M. Cebe, "A privacy framework for charging connected electric vehicles using blockchain and zero knowledge proofs," in *IEEE 44th LCN Symposium on Emerging Topics in Networking*, 2019, pp. 66–73.

[31] B. Sharma, R. Halder, and J. Singh, "Blockchain-based interoperable healthcare using zero-knowledge proofs and proxy re-encryption," *2020 International Conference on COMmunication Systems & NETworkS (COMSNETS)*, pp. 1–6, 2020.

[32] J. Heiss, M.-R. Ulbricht, and J. Eberhardt, "Put your money where your mouth is – towards blockchain-based consent violation detection," in *IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, 2020, pp. 1–9.

[33] R. Soltani, U. T. Nguyen, and A. An, "A survey of self-sovereign identity ecosystem," *CoRR*, vol. abs/2111.02003, 2021.

[34] N. Naik and P. Jenkins, "uPort open-source identity management system: An assessment of self-sovereign identity and user-centric data platform built on blockchain," in *ISSE*, 2020.

[35] JOLOCOM, *A decentralized, open source solution for digital identity and access management (whitepaper)*, `https://jolocom.io/wp-content/uploads/2019/12/Jolocom-Whitepaper-v2.1-A-Decentralized-Open-Source-Solution-for-Digital-Identity-and-Access-Management.pdf`, 2019.

[36] F. Vogelsteller and T. Yasaka, *Erc-725 smart contract based account*, `https://github.com/ethereum/EIPs/issues/725`), 2020.

[37] F. Vogelsteller, *Claim holder*, `https://github.com/ethereum/EIPs/issues/735`), 2019.

[38] D. K. Michael Lodder, *Anonymous credentials 2.0*, `https://wiki.hyperledger.org/download/attachments/6426712/Anoncreds2.1.pdf`, 2019.