

FORT: Right-proving and Attribute-blinding Self-sovereign Authentication

Xavier Salleras^{1,2}, Sergi Rovira¹, and Vanesa Daza^{1,3}

¹Department of Information and Communication Technologies,
Universitat Pompeu Fabra, Barcelona, Spain

²Dusk Network, Amsterdam, The Netherlands

³CYBERCAT - Center for Cybersecurity Research of Catalonia
xavier@dusk.network, {sergi.rovira, vanesa.daza}@upf.edu

Abstract

Nowadays, there is a plethora of services that are provided and paid for online, like video streaming subscriptions, car or parking sharing, purchasing tickets for events, etc. Online services usually issue tokens directly related to the identities of their users after signing up into their platform, and the users need to authenticate using the same credentials each time they are willing to use the service. Likewise, when using in-person services like going to a concert, after paying for this service the user usually gets a ticket which proves that he/she has the right to use that service. In both scenarios, the main concerns are the centralization of the systems, and that they do not ensure customers' privacy. The involved Service Providers are Trusted Third Parties, authorities that offer services and handle private data about users. In this paper, we design and implement FORT, a decentralized system that allows customers to prove their right to use specific services (either online or in-person) without revealing sensitive information. To achieve decentralization we propose a solution where all the data is handled by a Blockchain. We describe and uniquely identify users' rights using Non-Fungible Tokens (NFTs), and possession of these rights is demonstrated by using Zero-Knowledge Proofs, cryptographic primitives that allow us to guarantee customers' privacy. Furthermore, we provide benchmarks of FORT which show that our protocol is efficient enough to be used in devices with low computing resources, like smartphones or smartwatches, which are the kind of devices commonly used in our use case scenario.

Keywords: Zero-Knowledge Proofs; zk-SNARKs; Bulletproofs; Applied Cryptography; Self-sovereign; Internet of Things; Authentication; NFT.

This is the authors version of a work accepted and published in the special issue *Advances in Blockchain Technology* of the journal *Mathematics* (2022).

Contents

1	Introduction	3
1.1	Motivation	3
1.2	Our Contributions	4
1.3	Roadmap	4
2	Background	4
2.1	Blockchain	4
2.2	Smart contracts	5
2.3	Zero-Knowledge Proofs	5
2.4	zk-Rollups	6
3	Related work	6
4	Cryptographic building blocks	7
4.1	Preliminaries	7
4.2	Bulletproofs	7
4.3	zk-SNARKs	8
5	Our solution: FORT	8
5.1	Overall description	8
5.2	Protocol details	9
5.3	Security analysis	11
6	Implementation and benchmarks	12
6.1	Bulletproofs module	12
6.2	Solution deployment	12
6.2.1	Generate rights	13
6.2.2	Generate the certificate	13
6.2.3	Prove the attributes	14
7	Discussion on future works	14
8	Conclusions	15
	Acknowledgements	15
	References	15

1 Introduction

Recently, smart cities have evolved with the inclusion of static internet-connected devices [1, 2, 3] like pollution sensors, traffic lights, surveillance cameras, etc. Moreover, other mobile Internet of Things (IoT) [4] devices, like autonomous cars, will be soon populating the cities. If we take into account all the computers, smartphones, smartwatches, etc., we observe how the density of devices is achieving high numbers. Technically speaking, handling this huge amount of connections will be possible thanks to 5G communications [5], which introduce *network slicing* for splitting the network into many virtual and logical networks, providing specific features for different services.

A high density of devices is also translated to more data shared over the network. A concerning fact is what happens with the data shared by users, especially when such data is sensitive or it can simply be used to profile users with no permission. Even when the usage of Internet technologies is increasing very fast, some security and privacy concerns [6, 7] still need to be addressed. In this paper, we address the problem of Trusted Third Parties (TTP), which are still required in many scenarios [8]. For instance, GPS applications or autonomous driving are applications tracing our location and collecting much data about us. Even if the company behind says no personal data is collected, we can only trust them, with no possibility of detecting misbehavior. Other examples can be medical devices sharing sensitive information about patients through a trusted web server, or any other service that requires a TTP.

A natural solution to avoid the need for a TTP is the decentralization of its role. In this direction, several approaches [9, 10] raised in the last decade using Blockchain technologies to connect devices, skipping the need for a TTP in many use cases. In the same context, new digital services have appeared in the market, changing the way how users interact with them. Among many use cases, we can find car sharing, buying tickets for events, subscriptions to streaming services, etc. As centralization was a property of these applications that used to lead to control of all the network by some individuals, Blockchains [11] started to change the way people interact with online services. The most common example, cryptocurrencies, has become a payment method without central authorities (i.e. banks) controlling the stream of the issued transactions and all the collateral information. Moreover, beyond being a payment solution, Blockchains like Ethereum [12] offer a way to execute programs *on-chain*. Those programs, called *smart contracts*, allow to issue a payment to a specific party as soon as this party proves that he/she meets some requirements specified in the contract. This same approach is used in many decentralized applications (DApps) [13] nowadays, like paying the subscription to some service.

1.1 Motivation

Decentralization implies that public data stored in the Blockchain can be accessed by anyone. This leads to some privacy concerns: as Blockchains publicly store all the network activity, user tracking or profiling becomes an issue to be addressed. In such regard, the problem gets worse when users of a Blockchain-based service need to interact with real-world services (i.e. proving to the staff of an event that you paid for the ticket), so if anyone learns your Blockchain identity, he learns all your history.

To solve the privacy concerns that arose on Blockchain applications, Zero-Knowledge Proofs (ZKP) started to be integrated within Blockchain projects like Zcash [14]. ZKPs are cryptographic primitives allowing a user to prove to anyone else that some statement is true for some secret information, without leaking such information. In the Zcash example, these primitives allow users to issue transactions without leaking their identity nor the amount of money they are spending while proving that they are solvent.

Furthermore, Dusk Network [15] is a Blockchain taking a step further and introducing a way to program *private-by-design smart contracts*, programs whose execution and the parameters involved are kept private while being validated by the network. This leads to a toolset capable of building new privacy-preserving applications, solving many privacy and security concerns.

In such a scenario, the concept of *self-sovereign authentication* [16] appeared: authentication systems where users can manage their identities in a fully transparent way, deciding which information they are willing to reveal to other parties. Some solutions like SANS [17] allow users to prove to Service Providers (SP) that they own a token that proves their right to use a specific service. Such a solution is suitable in many scenarios, but in some cases can have efficiency drawbacks since it relies on a ZKP construction called zk-SNARKs [18], which requires high computing power. This scheme is executable on Internet of Things (IoT) devices thanks to implementations like ZPiE [19], but taking a fair amount of time. This fact makes such a solution infeasible in use cases where IoT devices must prove several things in a short amount of time (i.e. willing to use a smartwatch to prove a right, having a door sensor with a cheap CPU verifying proofs, etc.). Besides, this solution is still centralized, which means that if the SP disappears, the user no longer owns the right.

1.2 Our Contributions

In this paper, we introduce **FORT**, a novel self-sovereign authentication protocol, combined with Blockchain technologies to provide a solution where users of a service acquire *rights*, which are a set of different provable *blinded attributes*. Such attributes are portions of personal information which have been blinded: they are invisible to the SP, and only the user can decide how much information about them has to be leaked. These attributes are represented by Non-Fungible Tokens (NFT) [20] on the Blockchain, which can be granted *on-chain* by entities providing services, the SPs, and verified *off-chain*. For instance, a car willing to access a smart city would have to prove its right to do so, that is having two attributes: a certificate stating that the car has a low emissions level, and a fee payment receipt for entering the city. Once the right is represented in the Blockchain using an NFT, the car will be able to prove off-chain the possession of such a right, by using a cryptographic primitive called Zero-Knowledge Proof [21]. Such proof will state the possession of a valid NFT, without leaking the identifier of such NFT nor the identity of the car owner. Furthermore, our solution also skips third-party fees: for instance, in the scenario of buying tickets online for some event, in many cases, the ticket is issued by a third party who handles all the ticketing management, and who needs to be trusted. Furthermore, this party charges the users a service fee. Our solution relies on the Blockchain. Thus, the event organizer does not need to rely on third parties, and the user does not share his identity nor pay a service fee.

Moreover, even when Zero-Knowledge Proofs require high computing resources, our solution can be deployed in IoT devices thanks to ZPiE [19], the library we used to implement a proof-of-concept of our solution. This allows users to use our protocol using devices with low computing resources such as smartwatches.

Our contribution relies on zk-SNARKs, but also on range proofs, another ZKP scheme where users prove that a value lies within a given range, without leaking such a value to other parties. In particular, we use the Bulletproofs [22] range proofs scheme. For that reason, our second contribution in this paper is the implementation of a Bulletproofs module for ZPiE. Our implementation achieves excellent benchmarks, and using such a module, we implement our protocol and show its efficiency in IoT devices.

1.3 Roadmap

In Section 2, we introduce the background needed to understand the paper. In Section 3, we introduce the related work to our solution. In Section 4, we introduce the cryptographic building blocks of our solution. In Section 5, we introduce **FORT**, our solution. The implementation is detailed in Section 6, along with its security analysis and several benchmarks. Section 7 discusses the future works that could be done to integrate our solution into more use cases. We conclude in Section 8.

2 Background

In this section, we introduce the building blocks of our solution. We first introduce Blockchain technologies and their applications to IoT, and later we focus on the details and the specific use cases of smart contracts. We finally review Zero-Knowledge Proofs and how they are used to scale Blockchains by means of zk-Rollups.

2.1 Blockchain

A Blockchain [23] is a decentralized set of interconnected nodes, which share a unique and immutable set of data called ledger. Such a ledger is split into small portions of data called transactions, which are issued by different nodes in the network. In the scenario of cryptocurrencies like Bitcoin [11], these transactions are cryptographically validated by the network (i.e. a user sending some amount of bitcoins has enough funds to do so). Moreover, the whole network is ruled by a consensus agreed among all users of the network to keep the network safe (i.e. Proof-of-Work [24], Proof-of-Stake [25], etc.). The decentralization properties of Blockchains and their security and privacy features led researchers to their integration with smart cities and IoT scenarios. It has been a hot research topic in recent years, for instance in surveys like the one provided in [26], where some of the challenges and opportunities in such regard are stated. Many use cases emerged in the last years, including renting, sharing, or selling specific assets, like cars or apartments. Other approaches are Blockchains applied to Wireless Sensor Networks [27] or e-health devices [28], among many others.

As such, huge amounts of data regarding IoT devices can be found in Blockchains, data that could be useful for improving smart cities traffic control, energy consumption, or pollution. Interesting approaches like the one proposed in [29] allow for an easy IoT discovery in Blockchains. Regarding the security model, there are contributions like trust systems for IoT [10], where nodes of a Blockchain can be trusted upon checking their reputation, which changes depending on the behavior of the nodes in the network.

2.2 Smart contracts

One of the most useful Blockchains in regards to our scenario is Ethereum [12]. It is a network whose purpose is not being a currency for making payments but a way to run distributed applications (DApps). DApps are possible thanks to smart contracts [30], pieces of code executed on the Ethereum Virtual Machine (EVM) [31]. Such contracts and the EVM allow users, for instance, to be paid upon fulfilling some conditions. That is, for instance, distributed exchanges: applications where users buy or sell their cryptocurrencies to other users.

In order to execute transactions, Ethereum requires *gas*. This is the amount of Ether (Ethereum’s coin) per amount of bytes needed to run a transaction. Depending on how busy the Ethereum network is, the price of gas increases or decreases. This can make using Ethereum very expensive. To overcome such a problem, Zero-Knowledge-Rollups (zk-Rollups) have been proposed recently [32]. They basically group several transactions into a single transaction of the main Ethereum Blockchain. Whereas the Ethereum network is called *Layer 1*, the zk-Rollup is commonly called application of *Layer 2*. zk-Rollups are possible thanks to Zero-Knowledge Proofs. Both Zero-Knowledge Proofs and zk-Rollups are introduced in Section 2.3 and Section 2.4 respectively.

Similarly, as Ethereum does, Dusk Network [15] is a Layer 1 Blockchain that provides a virtual machine called *Rusk* which enables the deployment and execution of smart contracts. However, they introduce the *Confidential Security Contract Standard (XSC)*, which ensures the preservation of transactional confidentiality while simultaneously guaranteeing compliance through the use of Zero-Knowledge Proofs. This opens the door to a wide variety of use cases where privacy is a must, but accountability is required at the very same time.

2.3 Zero-Knowledge Proofs

A Zero-Knowledge Proof [21] is a cryptographic primitive which allows a prover P to convince a verifier V that a statement is true, without leaking any secret information. A statement is a set of elements known by both parties, defined as u , and the secret information only known by P is called the witness w . P wants to convince V that he knows w , which makes a set of operations involving u , to hold. Such operations are defined by a *circuit*, a graph composed of different wires and gates, which leads to a set of equations relating to the inputs and the outputs of these gates. Each of these equations is called *constraint*. As depicted in Figure 1, P executes a proving algorithm using u as the set of public inputs, and w as the private inputs. This execution outputs a set of elements of an elliptic curve defined over a finite field, which we call the proof π . We send π to V , who will use a verifying algorithm to verify that u is true, for a given w only known by P . Formally speaking, ZKPs must satisfy 3 properties:

- **Completeness:** If the statement is true, P must be able to convince V .
- **Soundness:** If the statement is false, P must not be able to convince V that the statement is true, except with negligible probability.
- **Zero-knowledge:** V must not learn any information from the proof beyond the fact that the statement is true.

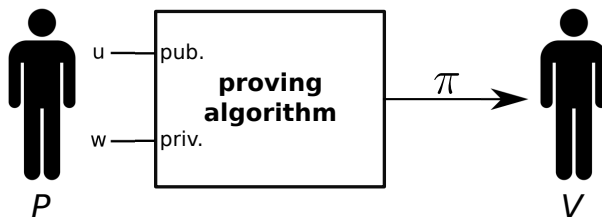


Figure 1: Zero-Knowledge Proof scenario.

Even when first schemes required P and V to interact several times, Non-Interactive ZKPs (NIZKPs) [33] emerged, allowing P to prove statements to V by sending him a single message. However, the first schemes were expensive in terms of computing resources, and this made them not useful in real applications. More recently, zk-SNARKs appeared, which are Zero-Knowledge Succinct and Non-interactive ARGuments of Knowledge [34]. This kind of proof is short and succinct: it can be verified in a few milliseconds, which makes it suitable for on-chain verification on Blockchains using smart contracts, being relatively cheaper in terms of gas consumption than other solutions.

2.4 zk-Rollups

zk-Rollups [32], as depicted in Figure 2, create batches of several transactions in a Layer 2 scenario, and publish the whole batch into a single Layer 1 transaction. This saves a lot of gas that would be consumed if each transaction was executed directly on the main Blockchain. To do so, we have two actors, the *transactors* willing to create a rollup transaction, and the *relayers* computing the required operations to make the rollup work. In such regard, transactors send transactions to the relayers containing information about the sender, the receiver, the amount of tokens to be sent, etc. Such transactions also include a signature of the transaction. As stated previously, ZKPs require an elliptic curve, as proofs are sets of elements on such curves. For instance, the Barreto-Naehrig elliptic curve [35] called BN128, is the currently used curve for zk-SNARKs in Ethereum. The signature scheme used is EdDSA [36], which also requires an additional elliptic curve where parameters are compatible with the zk-SNARKs elliptic curve (BN128). In this scenario, the Baby JubJub [37] elliptic curve is used, for its compatibility with the parameters of BN128.

Once the relayer has received a bunch of transactions, he computes a Merkle tree of the previous accounts' state and the new state. Later, he computes a zk-SNARK which verifies all the signatures, and posts on the Blockchain a transaction containing this batch: the rollup transactions, the previous and new root states, and the zk-SNARK. This transaction is verified by a smart contract previously deployed on the Blockchain.

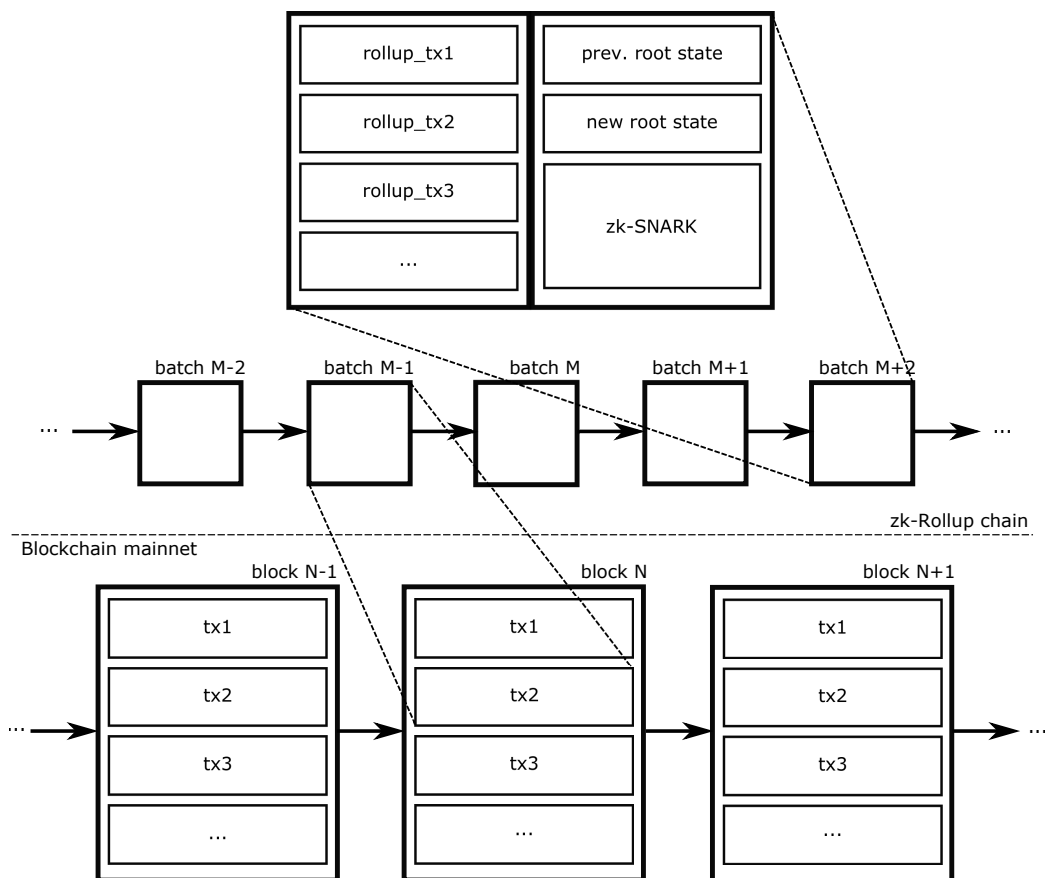


Figure 2: zk-Rollups overview.

3 Related work

Self-sovereign identity systems [38] have the premise of deploying protocols where users of different services can manage their identities in a secure, transparent, and private way. A general idea in this regard, and similar to our solution, was envisioned as a system where users can claim and prove possession of different rights associated with their identities, without compromising their privacy [39]. Furthermore, the combination of Self-sovereign identity systems with Zero-Knowledge Proofs has become a new research topic in the last years [40]. In this regard, solutions like SANS [17] introduce a private authentication mechanism based on Zero-Knowledge Proofs. Using such tools, SANS allows users to prove their rights to access several services, without the Service Provider (SP) knowing the identity of the users, while guaranteeing that the users are allowed to use the service (e.g. the users have paid a subscription fee).

There are some differences between SANS and this work. In all cases, ownership of a given token can be proved (Proof of Ownership). Moreover, this work can prove that the token exists in the Blockchain (Proof of Transaction). Our solution is meant to have protection against malleability: we allow the Service Providers (SP) to be sure that a given token has been used only once. We also deploy attributes blinding, where our solution becomes completely self-sovereign: users reveal their data in a transparent and private way. Furthermore, the efficiency of our scheme is increased and its usage in IoT devices is totally feasible.

Regarding privacy in online transactions, other research papers like [41] explore an interesting way to provide a privacy-preserving authentication protocol by means of Physical Unclonable Functions (PUFs), providing a solid and efficient protocol.

Besides, ongoing research regarding how zk-SNARKs can contribute to Blockchains scalability is done in [42], where research on distributed proofs generation making use of recursive zk-SNARKs is done.

On the other hand, combining IoT devices and NFTs is not an unexplored research area. Recent research [43] introduced a solution to manage IoT devices securely. They associate NFTs stored in Blockchains with IoT devices, to grant them a unique and indivisible identity.

Finally, to the best of our knowledge, there are no other solutions that provide a private-by-design and self-sovereign system to authenticate users, providing at the very same time a decentralized architecture, just like FORT does.

4 Cryptographic building blocks

In this section, we provide the necessary background on zk-SNARKs and Bulletproofs needed for the rest of the paper. We begin with a very high-level description of commitment schemes, and later move to an explanation of Bulletproofs and zk-SNARKs.

4.1 Preliminaries

We start by discussing a cryptographic primitive which is at the core of almost all modern cryptographic constructions, *commitment schemes*. A commitment scheme allows us to select a secret value and commit to it, in the sense that the party performing the commitment cannot change that value for another in the future. The scheme gives the capability to reveal the value later on, but this is not a mandatory task. We are particularly interested in Non-interactive Commitment Schemes, defined as follows:

Definition 4.1 (Non-interactive Commitment Schemes) *A non-interactive commitment scheme consists of a pair of probabilistic polynomial time algorithms (Setup, Commit). The setup algorithm $pp \leftarrow \text{Setup}(1^\lambda)$ generates public parameters pp given the security parameter λ . Given the public parameters pp , the commitment algorithm Commit defines a function $\mathcal{M} \times \mathcal{R} \rightarrow \mathcal{C}$ for message space \mathcal{M} , randomness space \mathcal{R} and commitment space \mathcal{C} . Given a message $x \in \mathcal{M}$, the commitment algorithm samples $r \leftarrow \mathcal{R}$ uniformly at random and computes $\text{Commit}(x; r) \in \mathcal{C}$.*

A useful commitment scheme for us is the Pedersen Commitment which we define as follows:

Definition 4.2 (Pedersen Commitment Scheme) *Let \mathbb{G} be a group of order p and set $\mathcal{M}, \mathcal{R} = \mathbb{Z}_p$ and $\mathcal{C} = \mathbb{G}$. The Setup and Commit algorithms for Pedersen commitments are defined as follows:*

- *Setup: Sample $g, h \leftarrow \mathbb{G}$ uniformly at random.*
- *Commit($x; r$): For a given $x \in \mathcal{M}$, and a random value $r \leftarrow \mathcal{R}$, we compute $g^x h^r \in \mathcal{C}$.*

4.2 Bulletproofs

Bulletproofs [22] are short non-interactive zero-knowledge arguments of knowledge that require no trusted setup. This means that the prover P sends a single message to the verifier V , and this is enough to prove knowledge of the secret information. There is no need to rely on any prior information generated by a trusted party.

Bulletproofs were designed to enable efficient confidential transactions in cryptocurrencies, but they have found many other applications, such as shortening proofs of solvency or enabling confidential smart contracts [44]. The main technical feature of Bulletproofs is to prove that a committed value lies within a certain interval. For example, in the context of Blockchains, it is very useful to have an efficient protocol to prove that a secret value lies in the interval $[0, 2^n - 1]$ for some large value of $n \in \mathbb{Z}_{\geq 0}$. In the cryptographic community, this feature is called a *range proof*. Range proofs allow us to prove that a secret value (previously committed to) lies within a certain range. They do not leak any information about the secret value but the fact that it lies within the desired range.

Let \mathbb{G} be a cyclic group of prime order p and let \mathbb{Z}_p be the ring of integers modulo p . An *inner-product argument* lets P convince V that he/she knows two vectors (bold font denotes a vector) $\mathbf{a}, \mathbf{b} \in \mathbb{Z}_p^n$ such that

$$C = \mathbf{g}^{\mathbf{a}} \mathbf{h}^{\mathbf{b}} \quad \text{and} \quad c = \langle \mathbf{a}, \mathbf{b} \rangle,$$

where $\mathbf{g}, \mathbf{h} \in \mathbb{G}^n$ are independent generators, $c \in \mathbb{Z}_p$, and $C \in \mathbb{G}$. Now, let $c \in \mathbb{Z}_p$ and let $C \in \mathbb{G}$ be a Pedersen commitment to c using randomness r . An inner-product range proof allows P to convince V that $c \in [0, 2^n - 1]$ by proving the relation

$$\{(g, h \in \mathbb{G}, C, n ; c, r \in \mathbb{Z}_p) : C = h^r g^c \wedge c \in [0, 2^n - 1]\}.$$

Now consider the case where P needs to provide multiple range proofs at the same time. The idea of aggregated range proofs is to build a system that can provide a proof for multiple secret values and its efficiency is better than doing one proof for each of the secrets. Since the inner-product range proofs provided by Bulletproofs have logarithmic size, it is possible to build efficient aggregated logarithmic range proofs. That is, it is possible to efficiently prove the relation

$$\{(g, h \in \mathbb{G}, \mathbf{C} \in \mathbb{G}^m ; \mathbf{c}, \mathbf{r} \in \mathbb{Z}_p^m) : C_j = h^{r_j} g^{c_j} \wedge c_j \in [0, 2^n - 1] \forall j \in [1, m]\},$$

where m corresponds to the number of proofs. Bulletproofs can be computed in $O(n)$, and verified in linear time as well. The communication complexity (the size of the proofs) is $O(\log n)$.

4.3 zk-SNARKs

One of the most used ZKP systems in practice are zk-SNARKs [18]. This kind of proofs are short and succinct: they can be verified in only a few milliseconds. zk-SNARKs require a trusted setup that is used both by the prover and the verifier to generate and verify proofs. The set of parameters obtained during the setup phase is commonly called the Common Reference String (CRS). If an attacker was able to get the secret random values used to generate the CRS, it would be able to generate false proofs. For this reason, the initial setup is commonly made through a secure Multi-Party Computation (MPC) protocol [45], which generates the required parameters using a distributed computation protocol.

Definition 4.3 (zk-SNARKs) *A zero-knowledge succinct non-interactive argument of knowledge is a triple of algorithms (Setup, Prove, Verify) that works as follows:*

- $(pk, vk) \leftarrow \text{Setup}(1^\lambda, \text{circuit})$: The setup algorithm outputs a proving key pk and a verification key vk given the security parameter λ and a circuit. Both keys (the CRS) are made public and can be used by the prover and the verifier to generate and verify proofs.
- $\pi \leftarrow \text{Prove}(pk, u, w)$: The proving algorithm produces a proof π using the proving key pk that attests that a statement u and a witness w are a correct solution to the set of equations derived from the circuit.
- $1/0 \leftarrow \text{Verify}(vk, u, \pi)$: The verification algorithm uses the verification key vk to check whether π is a correct proof for the public statement u .

zk-SNARKs can be computed in $O(n \log n)$. Both the verification and the communication complexity are $O(1)$.

5 Our solution: FORT

In this section, we introduce our solution with all details. We start with an overall description of our protocol, to later move to the specific details and its security analysis.

5.1 Overall description

Our solution is meant to be used in scenarios where users need to prove their right to use a service, for instance, accessing a house rented online. In such a scenario, we envision the usage of a certificate installed in the user's smartphone (or smartwatch, or any similar device) which can be validated by some sensor installed in the door of the house. Once validated, the SP might want a proof of meeting some requirements, linked with the previously validated certificate. We detail this workflow (as depicted in Figure 3) in this section, as follows:

1. **Read on-chain information:** the user acquires some attributes granted by third parties, which can be a SP to whom the user is buying a ticket or subscription, a governmental entity verifying your personal information, a bank providing a proof of solvency, etc. Such attributes are granted through an NFT stored in a Blockchain. The SP issues an NFT representing user's attributes. The SP mints this NFT on-chain, and later transfers it to the user's address. Now, the user can read these attributes from the Blockchain.

2. **Compute proof (the certificate):** the user acts as a prover, and computes a ZKP from the information collected from the NFT, as detailed in the circuit of Figure 4, and installs this certificate in his device.
3. **Send proof (read certificate):** The user tries to use the service by showing the certificate to the SP, who reads it.
4. **Verify on-chain information:** the SP needs to partially read the Merkle tree of the Blockchain (as detailed in next section) to be able to verify (in the next step) that the attributes the user wants to prove are really on-chain (the NFT).
5. **Verify proof (validate the certificate):** The SP verifies the ZKP, thus verifying the rights of the user.

After performing this protocol, the SP can ask the user for some information about the attributes, for instance, if they lay within a specific range. To do it, the user computes a bulletproof and sends it to SP, the verifier. Then, the SP verifies that the bulletproof sent by the user is correct, and knows for sure that the value is within a specific range.

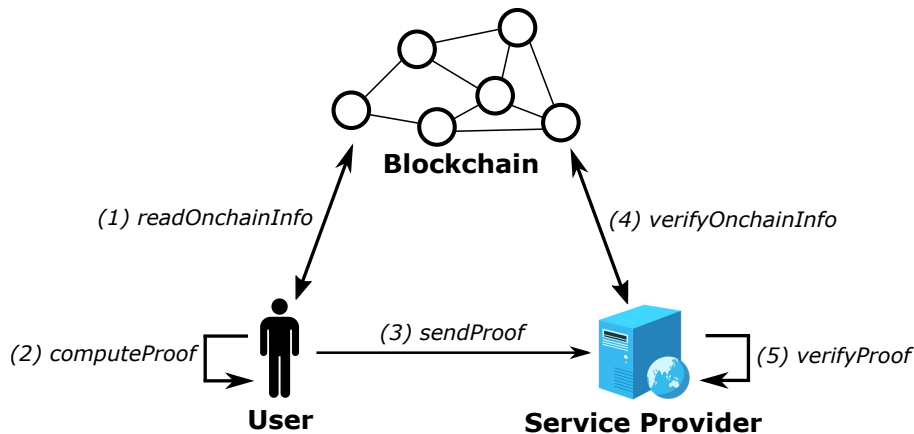


Figure 3: FORT protocol scenario overview.

Ideally, a desirable way to implement our protocol would be creating the NFTs directly using a Blockchain, and proving that we own them using the signature details involving the Blockchain transaction representing each NFT. However, this has some constraints regarding scalability and efficiency: first, the gas fees in the case of Ethereum can become very expensive, so using a transaction for a single right is far from being optimal. Second, the elliptic curve used to sign Ethereum transactions, the *secp256k1*, is not pairing-friendly, so generating proofs proving ownership of the private key used to sign the transaction will not be efficient. To solve this problem, FORT relies on zk-Rollups for scalability, and on EdDSA for proofs off-chain.

5.2 Protocol details

To be able to prove rights to access the service, the first thing a user needs to do is to receive an NFT stating some attributes about him. To do so, the user needs to contact the SP and provide him/her a proof of meeting some requirements. Then, SP executes Algorithm 1: after validating the requirements, it issues an NFT representing the user's attributes. The SP mints this NFT on-chain, and later transfers it to the user's address.

Algorithm 1: Create NFT

Env: vector of x attributes: $attributes[x]$; user's address: pk_{user} ; user's conditions: cnd

```

nft ← create_nft(attributes[x]);
if (verify_conditions(cnd)) then
  nft.id = rand();
  nft.attr = attributes[:];
  nft.S ← signskSP(H(nft.id||nft.attr||pkuser));
  mint_nft(nft);
  transfer_nft(nft, pkuser);
end

```

Upon receiving the NFT, the user is ready to anonymously prove possession of such an NFT. To do so, the user will follow the Algorithm 2. Then, at some point the user will want to prove some rights, and to do so he/she will send / show the proof to the SP and it will execute Algorithm 3.

Algorithm 2: Create certificate

Env: User and Service Provider (SP).

1. The user reads the NFT transaction to be proved, which is published on the Blockchain, and the IDs of a set of NFT transactions in the batch $batch_ids$, where $|batch_ids| = 2^x$ and x is agreed by consensus.
 2. The user requests access to the service offered by SP, and the SP sends a random value, a challenge c to the user.
 3. The user computes a proof π using the above parameters as stated in the circuit depicted in Figure 4.
-

Algorithm 3: Verify certificate

Env: User and Service Provider (SP).

1. The SP receives / reads π from the user.
 2. SP collects the ID of the transactions in the batch and computes the merkle tree $mtree$.
 3. SP executes Algorithm 4, if it returns 1, SP grants the service.
-

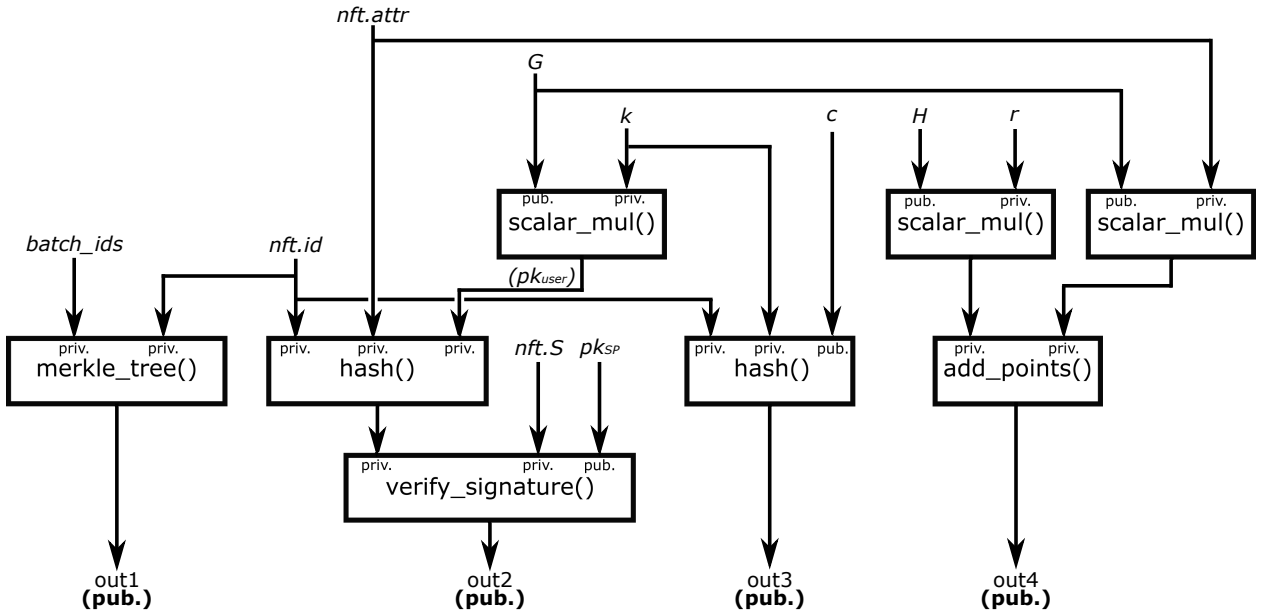


Figure 4: Circuit for our solution.

Algorithm 4: Verify right

Env: Zero-knowledge proof π .

$1/0 \leftarrow verify_right(\pi)$:

```
if (out1, out2, out3  $\leftarrow$  verify_proof( $\pi$ )) and (out1 == mtree) and (out2 == 1) and !(is_seen(out3)) then
| return 1;
else
| return 0;
end
```

The proof used in Algorithm 2 uses the circuit depicted in Figure 4. As can be seen, the circuit includes a verification of the signature $nft.S$, using the public key of the SP pk_{SP} . The inputs of the signature were the attributes $nft.attr$ along with the ID $nft.id$ and the public key of the user pk_{user} . The challenge c is hashed along with $nft.id$ and the user’s private key k . $nft.id$ is also hashed along with $batch_ids$.

Finally, and before the SP grants the service, the prover might have to create a bulletproof to reveal some information about the attributes. This is done using $out4$, a Pedersen commitment. This process is detailed in Algorithm 5.

Algorithm 5: Create Bulletproof

Env: secret key k ; vector of x attributes: $attributes[x]$; vector of x commitments: $C[x]$
 $\pi_b[x] \leftarrow create_bulletproof(C[x], attributes[x])$:
for i **in** x **do**
 | $\pi_b[i] \leftarrow bulletproof(C[i], attributes[i])$
end

5.3 Security analysis

An important element to consider when analyzing the security of FORT is the ZKP scheme to use. The main drawback of some ZKP constructions like zk-SNARKs, when used in scenarios like cryptocurrencies, is the need for a trusted setup. An untrusty setup could lead to huge losses of money if a malicious party gets the seed used to compute it, so it could create false transactions. This is not a problem in our solution: a different setup can be generated by each SP, as the proofs are verified off-chain by a single entity, the SP, and he is the main interested in not leaking the secret seed.

Moreover, the soundness property of each scheme relies on different security assumptions [46] (e.g. zk-SNARKs in [47] use a strong assumption, the q -Power Knowledge of Exponent (q -PKE) assumption). Furthermore, the security of these schemes relies on the security of elliptic curves, where breaking the security of the selected curve would lead to being able to generate false proofs. One of the most used curves in ZKPs is the BN128, which security level in practice is estimated to be 110-bits [48]. Other curves like BLS12-381 [14] estimate around 128-bits of security, with the drawback of heavier group operations. More recent research is introduced in [49], where a new curve called BW6-761 is introduced. As stated by its authors, verification of proofs is at least five times faster than other state-of-the-art curves.

Regarding the circuit we have designed, our solution grants several privacy and authentication features:

- **Proof of Ownership:** the circuit used in FORT verifies a signature $nft.S$ of an input $nft.id, nft.attr, pk_{user}$, using the public key of SP, pk_{SP} . Also, pk_{user} is the output of the scalar multiplication kG , where k is the user’s private key. This ensures that the user owns the NFT, as only him can compute the public key using the private key, while keeping both values private, so SP cannot learn the identity of the user.
- **Proof of Transaction:** the circuit computes a Merkle tree of two private inputs: the $nft.id$ and the IDs of some other issued NFTs in the same batch $batch_ids$. This ensures that the NFT the user is proving ownership of has been transacted in the Blockchain. SP can compute the Merkle tree $mtree$ itself, and check if it equals $out1$ as stated in Algorithm 4.
- **Malleability protection:** the circuit computes the hash of $nft.id$, the private key k , and a challenge c . The format of this value could change in different scenarios. Taking the example of proving ownership of a ticket for an event, ideally, c would be the date of such event. If $is_seen(out3, previous) == 1$, it means that someone already entered the event with the same NFT. This is true because neither $nft.id$ nor k can change, so $out3$ will always be the same for a given public input c . This prevents a user to use the same right multiple times, and to compute valid proofs for other users.
- **Attribute blinding:** the private information the user wants to share only when required, the attributes, are private inputs of the circuit. Such values are committed using a Pedersen commitment (i.e. $out4$, but as many as required can be included in the circuit), so the verifier learns these commitments, and the prover later uses a Bulletproof to prove knowledge of them, and to prove that they are within a specific range.

FORT, as introduced in this section, can also be seen as framework to be modified to match the needs of every use case our solution could be deployed to. This means, selecting the proper ZKP scheme to be used, recompute the certificate each time instead of using Bulletproofs, select a different challenge c , etc.

6 Implementation and benchmarks

In this section, we explain the capabilities and implementation details of the Bulletproofs module we developed, and later explain how we implemented our specific solution using our module.

6.1 Bulletproofs module

We implemented Bulletproofs as a module integrated into ZPiE, a Zero-Knowledge Proofs library coded in C. The library uses GMP and MCL as dependencies: GMP is a pure C library used to handle big numbers and operations involving them, and MCL is a library written in C++, which offers a C wrapper for using it in pure C projects, used to do elliptic curve operations. The library also supports the elliptic curves BN128 and BLS12-381, which are thus also supported by our implementation. We implemented an API that allows us to generate aggregated range proofs using the Bulletproofs scheme above referred, and to verify them. The instructions on how to compile and use the library can be found in the README of the repository. The code can be used as explained in Listing 1.

```
1 #include "../src/zpie.h"
2
3 int main()
4 {
5     // init the bulletproofs module for 2 aggregated proofs of 64 bits
6     bulletproof_init(64, 2);
7
8     // set some values to prove knowledge of and compute the bulletproof
9     unsigned char *si[] = {"1234", "5678"};
10    bulletproof_prove(si);
11
12    // verify the bulletproof (../data/bulletproof.params)
13    if(bulletproof_verify()) printf("Bulletproof verified.\n");
14    else printf("Bulletproof cannot be verified.\n");
15 }
```

Listing 1: Bulletproof generation example. Generation of 2 aggregated proofs of 64 bits.

We benchmarked our implementation as depicted in Figure 5. Moreover, we improved the efficiency of our solution by using multi-threading in several parts of the prover and the verifier, where splitting the operations in different cores was possible. As we can see, we are benchmarking the time it takes by the prover, either in single-core (SC) or multi-core (MC), to compute the proofs. We performed the experiments for several amounts of aggregated proofs of 64 bits, using a 4-cores CPU, and the BN128.

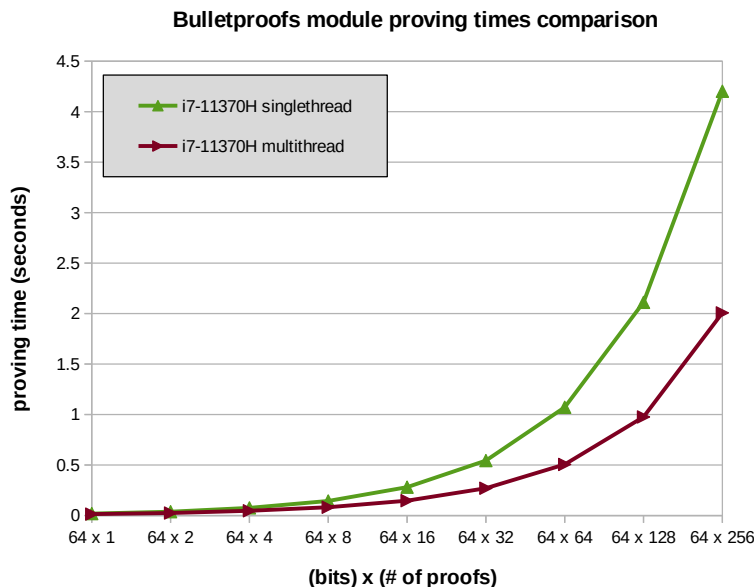


Figure 5: CPUs proving times of our solution.

6.2 Solution deployment

In this subsection, we detail the deployment of the three main parts of our protocol, *generate rights*, *generate the certificate*, and *prove the attributes*.

6.2.1 Generate rights

The first step to use our solution is to generate the rights that our users will need to prove. To do so, a SP needs to provide a service and sell its subscription using an NFT minted to a smart contract-based Blockchain. For testing purposes, we used an Ethereum testnet where we created test NFTs using a reference implementation of ERC-721 (the Ethereum NFT standard)⁰. After deploying an NFT to the Blockchain, a user can buy it. Once done, he is ready to generate the certificate.

The computational costs for generating the NFTs are negligible, as no heavy cryptographic computations are involved in the process. Regarding the time it takes to be reflected on the Blockchain, it would depend on how crowded it is (typically it will take only a few minutes). On the other hand, one of the main concerns regarding this step when deploying it into the mainnet is the amount of gas required to execute the smart contract that mints the NFT. As discussed before, using zk-Rollups would be the best choice to reduce the cost when moving our solution to a production environment.

In Section 7 we discuss further work to be done regarding the deployment of our solution into a Blockchain network, considering how to boost even further the capabilities of our solution when using other Blockchains as the backbone of FORT.

6.2.2 Generate the certificate

As explained previously, the prover precomputes the certificate, which is a zk-SNARK, required to use a specific service. The SP will verify the certificate and will be sure of the prover’s right to use the service. We used `circomlib`¹ to estimate the number of constraints of the circuit used in our solution and thus, its efficiency. To create our circuit, we rely on four main functions:

- `scalar_mul()`: the circuit needs to multiply a number k by a point on an elliptic curve G . To do this scalar multiplication using BN128, `circomlib` uses **776 constraints**.
- `hash()`: the circuit needs to perform 2 fixed hashes, plus a variable number of hashes to compute a merkle tree. A fairly secure and efficient hash function is Poseidon [50], which only uses **210 constraints** in `circomlib`.
- `verify_signature()`: we use the state-of-the-art signature scheme EdDSA [36] over BN128 provided in `circomlib`, which uses **4018 constraints**.
- `merkle_tree()`: the circuit needs to compute a merkle tree. Assuming that $|batch_ids| = 256 = 2^8$, our solution will need to compute 8 Poseidon hashes. This sums up to **1680 constraints**.

In total, our circuit can be implemented using **6894 constraints**. We coded a proof-of-concept using ZPiE², and executed the code using a laptop, a smartphone and a Raspberry Pi Zero. To demonstrate the scalability of our solution, we also executed the circuit using `snarkjs`³, a JavaScript implementation of zk-SNARKs which can be executed in web browsers. This is perfect for scalability in web applications, with the performance drawback it involves, compared with binaries executed directly in the kernel. Table 1 shows the results.

Table 1: Performance results of FORT in different devices using different implementations. All experiments use Groth’16 and BN128.

Device	Prover	Verifier
Raspberry Pi Zero W (ZPiE)	79.058 s	0.134 s
Snapdragon 732G (ZPiE)	0.830 s	0.005 s
i7-11370H (ZPiE)	0.157 s	0.000733 s
i7-11370H - Firefox (snarkjs)	0.694 s	0.022 s

As can be seen, either in high-end devices (a laptop CPU like i7-11370H) or in mobile CPUs (snapdragon 732G), the proofs used in our protocol can be computed in a fair small amount of time using ZPiE. On the other hand, the time increases a lot when talking about extremely low-end CPUs like the one used in the Raspberry Pi Zero. Nevertheless, computing the proof in about a minute taking into account the single-core 700MHz CPU that it uses (aprox. 10\$), is a good result. Furthermore, an advantage of FORT is that proofs can be precomputed much before to be used. Plus, even in worst-case scenarios, protocols like the one introduced in [51] would allow those devices to rely computations on other servers owned by the same user, using a secure channel.

⁰<https://github.com/nibbstack/erc721>

¹<https://github.com/iden3/circomlib>

²<https://github.com/xevalle/zpie>

³<https://github.com/iden3/snarkjs>

Regarding the verification of these proofs, as we stated previously, the verifier is succinct: all the proofs can be verified in just a few milliseconds, with no relation with the size of the circuit. As can be seen, ZPiE outperforms here even in the Raspberry Pi, where it takes roughly 0.1 seconds to verify proofs.

Finally, we can see how either the prover and the verifier in `snarkjs` are much slower than ZPiE for the same CPU. However, such a result was expected, and taking into account the trade-off between performance and scalability, still it is a great result.

6.2.3 Prove the attributes

The SP, after verifying the certificate, might want to be sure that some of the attributes `nft.attr` meet some additional requirements (e.g. being within a given range). For such purpose, we compute a Bulletproof from the Pedersen commitment described in the zk-SNARK circuit. We use the module introduced in the last section to achieve this outcome. In Listing 2 we show how to deploy our solution, where the prover proves knowledge of the Pedersen commitment, and that the secret lies within the range $[0, 2^8 - 1]$.

```
1 #include "../src/zpie.h"
2
3 int main()
4 {
5     // we init the bulletproofs module, for a bulletproof of 8 bits
6     bulletproof_init(8, 1);
7
8     // we get the context (G, H, V[], gammas[])
9     context ctx;
10    bulletproof_get_context(&ctx);
11
12    // we state that we will provide the random gamma and we assign it
13    // according to the one used in the certificate
14    bulletproof_user_gammas(1);
15    mclBnFr_setInt(&ctx.gammas[0], 1234); // r = 1234
16
17    // we need to create a bulletproof for this commitment:
18    // out4 = attr*G + r*H
19    // we set the input attr = "250"
20    unsigned char *si[] = {"250"};
21    bulletproof_prove(si);
22
23    // now P -> V: Bulletproof
24    // V reads out4 from the certificate, and verifies the Bulletproof:
25    if(bulletproof_verify()) printf("Bulletproof verified.\n");
26    else printf("Bulletproof cannot be verified.\n");
27 }
```

Listing 2: Implementation of our solution.

The above code for proving knowledge of an 8-bit attribute takes only **0.3 seconds** on a Raspberry Pi Zero. This time increases as the size of the attributes does the same, but being a fair amount of time to be able to use our solution in IoT devices without problems. Executing the same approach using a zk-SNARK will require around 776 constraints, and the benchmark gives us **10.5 seconds**. As such, it is clear that Bulletproofs are a much better approach for this specific use case, where provers will be able to execute the protocol instantly using low-powered devices.

7 Discussion on future works

We introduced a protocol that allows a user to get some rights to be used in different scenarios: the right to use a service (i.e. demonstrate to have some attributes like not being underage, having a salary above some threshold, etc.) or the right to access an event (i.e. demonstrate to have the attribute, in this case, the ticket for entering to an event, a performance, etc.). Our protocol works as-it-is in such scenarios. Needless to say, some changes shall be made if other constraints arise, or in other use cases. The usage of standard NFTs on Ethereum opens a wide range of features to implement. For instance, NFTs offer the feature of being transferred from one user to another, while charging a percentage of the selling price to the original creator of the token (e.g. the event planner). At the very same time, a SP organizing a performance could allow users to resell the tickets if they cannot attend, but prevent them to increase the price while preventing price speculations as well.

We have seen how FORT could be easily deployed using Blockchains like Ethereum or Dusk. Regarding the latter, which at the time of writing is still under development, we have to take into account the private nature of the execution of their smart contracts. We envision how future work on a fully integrated solution within their network would lead to new privacy models, enhancing our protocol by even blinding the data we need to store on-chain.

8 Conclusions

In this paper, we introduced a protocol for proving ownership of a right for using a service or accessing an event, in a self-sovereign manner. Our protocol grants the chance to buy or request to be granted different attributes, which are grouped into rights, using a Blockchain. We can prove ownership of such rights using Zero-Knowledge Proofs, the main element of our FORT protocol. After stating the details of FORT and its security analysis, we performed several tests to show as using only 6894 constraints, it can be executed very efficiently in a wide variety of devices and environments: desktop, mobile, and web applications. As a reference, we have seen how we can compute a certificate with the users' rights in less than a second using a conventional smartphone. Later, the attributes of the certificate can be proved in just a few milliseconds. As future work, we discussed how our protocol could be modified to fit in other use cases, like ticket reselling or rights transferring, and we also discussed how integrating our solution into the Dusk Network Blockchain would lead to a higher level of privacy.

Acknowledgements

The authors are supported by Project RTI2018-102112-B-100 (AEI/FEDER, UE).

References

- [1] Francisco Ramos, Sergio Trilles, Andrés Muñoz, and Joaquín Huerta. Promoting pollution-free routes in smart cities using air quality sensor networks. *Sensors*, 18(8), 2018.
- [2] Syed Misbahuddin, Junaid Ahmed Zubairi, Abdulrahman Saggaf, Jihad Basuni, Sulaiman A-Wadany, and Ahmed Al-Sofi. Iot based dynamic road traffic management for smart cities. In *2015 12th International Conference on High-capacity Optical Networks and Enabling/Emerging Technologies (HONET)*, pages 1–5, 2015.
- [3] Fadi Al-Turjman and Joel Poncha Lemayian. Intelligence, security, and vehicular sensor networks in internet of things (iot)-enabled smart-cities: An overview. *Computers & Electrical Engineering*, 87:106776, 2020.
- [4] Sakshi Painuly, Priya Kohli, Priya Matta, and Sachin Sharma. Advance applications and future challenges of 5g iot. In *2020 3rd International Conference on Intelligent Sustainable Systems (ICISS)*, pages 1381–1384, 2020.
- [5] ETSI (3GPP). Procedures for the 5G System (5GS), v15.5.1, release 15, May 2019.
- [6] Sidra Ijaz, Munam Ali Shah, Abid Khan, and Mansoor Ahmed. Smart cities: A survey on security concerns. *International Journal of Advanced Computer Science and Applications*, 7(2):612–625, 2016.
- [7] Liesbet Van Zoonen. Privacy concerns in smart cities. *Government Information Quarterly*, 33(3):472–480, 2016.
- [8] Liehuang Zhu, Meng Li, Zijian Zhang, and Zhan Qin. Asap: An anonymous smart-parking and payment scheme in vehicular networks. *IEEE Transactions on Dependable and Secure Computing*, 17(4):703–715, 2018.
- [9] Gbadebo Ayoade, Vishal Karande, Latifur Khan, and Kevin Hamlen. Decentralized iot data management using blockchain and trusted execution environment. In *2018 IEEE International Conference on Information Reuse and Integration (IRI)*, pages 15–22, 2018.
- [10] Roberto Di Pietro, Xavier Salleras, Matteo Signorini, and Erez Waisbard. A blockchain-based trust system for the internet of things. In *Proceedings of the 23rd ACM on Symposium on Access Control Models and Technologies*, pages 77–83, 2018.
- [11] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008. <https://bitcoin.org/bitcoin.pdf>, Accessed on 06/02/2022.
- [12] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger (istanbul version), June 2021.
- [13] P. Shamili, B. Muruganantham, and B. Sriraman. Understanding concepts of blockchain technology for building the dapps. In Subhransu Sekhar Dash, Swagatam Das, and Bijaya Ketan Panigrahi, editors, *Intelligent Computing and Applications*, pages 383–394, Singapore, 2021. Springer Singapore.

- [14] Daira Hopwood, Sean Bowe, Taylor Hornby, and Nathan Wilcox. Zcash Protocol Specification - Version 2019.0.2, 2019. <https://github.com/zcash/zips/blob/master/protocol/protocol.pdf>, Accessed on 28/09/2021.
- [15] Toghrul Maharramov, Dmitry Khovratovich, and Emanuele Francioni. The dusk network whitepaper, 2021. https://dusk.network/uploads/The_Dusk_Network_Whitepaper_v3_0_0.pdf, Accessed on 04/02/2022.
- [16] Geovane Fedrecheski, Jan M. Rabaey, Laisa Caroline de Paula Costa, Pablo C. Calcina-Ccori, William Takeshi Pereira, and Marcelo Knörick Zuffo. Self-sovereign identity for iot environments: A perspective. *2020 Global Internet of Things Summit (GIoTS)*, pages 1–6, 2020.
- [17] Xavier Salleras and Vanesa Daza. Sans: Self-sovereign authentication for network slices. *Security and Communication Networks*, 2020, 2020.
- [18] Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016*, pages 305–326, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.
- [19] Xavier Salleras and Vanesa Daza. Zpie: Zero-knowledge proofs in embedded systems. *Mathematics*, 9(20), 2021.
- [20] Jacob Evans William Entriken, Dieter Shirley and Nastassia Sachs. Eip-721: Erc-721 non-fungible token standard, January 2018.
- [21] S Goldwasser, S Micali, and C Rackoff. The knowledge complexity of interactive proof-systems. In *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*, STOC '85, pages 291–304, New York, NY, USA, 1985. ACM.
- [22] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 315–334, 2018.
- [23] Stephan Leible, Steffen Schlager, Moritz Schubotz, and Bela Gipp. A review on blockchain technology and blockchain projects fostering open science. *Frontiers in Blockchain*, 2:16, 2019.
- [24] Arthur Gervais, Ghassan O Karame, Karl Wüst, Vasileios Glykantzis, Hubert Ritzdorf, and Srdjan Capkun. On the security and performance of proof of work blockchains. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 3–16, 2016.
- [25] Iddo Bentov, Charles Lee, Alex Mizrahi, and Meni Rosenfeld. Proof of activity: Extending bitcoin’s proof of work via proof of stake [extended abstract] y. *ACM SIGMETRICS Performance Evaluation Review*, 42(3):34–37, 2014.
- [26] Ana Reyna, Cristian Martín, Jaime Chen, Enrique Soler, and Manuel Díaz. On blockchain and its integration with iot. challenges and opportunities. *Future generation computer systems*, 88:173–190, 2018.
- [27] Zhihua Cui, XUE Fei, Shiqiang Zhang, Xingjuan Cai, Yang Cao, Wensheng Zhang, and Jinjun Chen. A hybrid blockchain-based identity authentication scheme for multi-wsn. *IEEE Transactions on Services Computing*, 13(2):241–251, 2020.
- [28] N. Rifi, E. Rachkidi, N. Agoulmine, and N. C. Taher. Towards using blockchain technology for ehealth data access management. In *2017 Fourth International Conference on Advances in Biomedical Engineering (ICABME)*, pages 1–4, 2017.
- [29] V. Daza, R. Di Pietro, I. Klimek, and M. Signorini. Connect: Contextual name discovery for blockchain-based services in the iot. In *2017 IEEE International Conference on Communications (ICC)*, pages 1–6, 2017.
- [30] Anastasia Mavridou and Aron Laszka. Designing secure ethereum smart contracts: A finite state machine based approach. In *International Conference on Financial Cryptography and Data Security*, pages 523–540. Springer, 2018.
- [31] Everett Hildenbrandt, Manasvi Saxena, Nishant Rodrigues, Xiaoran Zhu, Philip Daian, Dwight Guth, Brandon Moore, Daejun Park, Yi Zhang, Andrei Stefanescu, et al. Kevm: A complete formal semantics of the ethereum virtual machine. In *2018 IEEE 31st Computer Security Foundations Symposium (CSF)*, pages 204–217. IEEE, 2018.

- [32] Ethworks Reports. Zero-knowledge blockchain scalability, 2020.
- [33] Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, STOC '88, pages 103–112, New York, NY, USA, 1988. ACM.
- [34] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Succinct non-interactive zero knowledge for a von neumann architecture. Cryptology ePrint Archive, Report 2013/879, 2013. <https://eprint.iacr.org/2013/879>.
- [35] Paulo S. L. M. Barreto and Michael Naehrig. Pairing-friendly elliptic curves of prime order. Cryptology ePrint Archive, Report 2005/133, 2005. <https://eprint.iacr.org/2005/133>.
- [36] Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. High-speed high-security signatures. *Journal of Cryptographic Engineering* 2, 2012.
- [37] Jordi Baylina and Marta Bellés. Eddsa for baby jubjub elliptic curve with mimc-7 hash.
- [38] Christopher Allen. The path to self-sovereign identity. Accessed 2020-07-17. <http://www.lifewithalacrity.com/2016/04/the-path-to-self-sovereign-identity.html>.
- [39] Sovrin Foundation. Sovrin: A Protocol and Token for Self-Sovereign Identity and Decentralized Trust. <https://sovrin.org/wp-content/uploads/Sovrin-Protocol-and-Token-White-Paper.pdf>, January 2018.
- [40] Alexander Mühle, Andreas Grüner, Tatiana Gayvoronskaya, and Christoph Meinel. A survey on essential components of a self-sovereign identity. *Computer Science Review*, 30:80–86, Nov 2018.
- [41] Georgios Fragkos, Cyrus Minwalla, Jim Plusquellic, and Eirini Eleni Tsiropoulou. Artificially intelligent electronic money. *IEEE Consumer Electronics Magazine*, 10(4):81–89, 2021.
- [42] Yuri Bepalov, Alberto Garoffolo, Lyudmila Kovalchuk, Hanna Nelasa, and Roman Oliynykov. Probability models of distributed proof generation for zk-snark-based blockchains. *Mathematics*, 9(23), 2021.
- [43] Javier Arcenegui, Rosario Arjona, and Iluminada Baturone. Secure management of iot devices based on blockchain non-fungible tokens and physical unclonable functions. In *Applied Cryptography and Network Security Workshops*, pages 24–40. Springer International Publishing, 2020.
- [44] Benedikt Bünz, Shashank Agrawal, Mahdi Zamani, and Dan Boneh. Zether: Towards privacy in a smart contract world. Cryptology ePrint Archive, Report 2019/191, 2019. <https://eprint.iacr.org/2019/191>, Accessed on 28/09/2021.
- [45] Sean Bowe, Ariel Gabizon, and Ian Miers. Scalable Multi-party Computation for zk-SNARK Parameters in the Random Beacon Model. Cryptology ePrint Archive, Report 2017/1050, 2017. <https://eprint.iacr.org/2017/1050>, Accessed on 28/09/2021.
- [46] Shafi Goldwasser and Yael Tauman Kalai. Cryptographic assumptions: A position paper. In Eyal Kushilevitz and Tal Malkin, editors, *Theory of Cryptography*, pages 505–522, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.
- [47] Jens Groth. On the size of pairing-based non-interactive arguments. Cryptology ePrint Archive, Report 2016/260, 2016. <https://eprint.iacr.org/2016/260>.
- [48] Alfred Menezes, Palash Sarkar, and Shashank Singh. Challenges with assessing the impact of nfs advances on the security of pairing-based cryptography. Cryptology ePrint Archive, Report 2016/1102, 2016. <https://eprint.iacr.org/2016/1102>.
- [49] Youssef El Housni and Aurore Guillevic. Optimized and secure pairing-friendly elliptic curves suitable for one layer proof composition. Cryptology ePrint Archive, Report 2020/351, 2020. <https://eprint.iacr.org/2020/351>.
- [50] Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. Starkad and poseidon: New hash functions for zero knowledge proof systems. Cryptology ePrint Archive, Report 2019/458, 2019. <https://eprint.iacr.org/2019/458>.
- [51] Howard Wu, Wenting Zheng, Alessandro Chiesa, Raluca Ada Popa, and Ion Stoica. Dizk: A distributed zero knowledge proof system. Cryptology ePrint Archive, Report 2018/691, 2018. <https://eprint.iacr.org/2018/691>.