applied sciences

Article

Implementation and Security Test of Zero-Knowledge Protocols on SSI Blockchain

Cristina Vilchez Moya, Juan Ramón Bermejo Higuera *, Javier Bermejo Higuera and Juan Antonio Sicilia Montalvo

Faculty of Engineering, Universidad Internacional de La Rioja, Av. de la Paz, 137, 26006 Logroño, Spain; cris.vmoya@unir.net (C.V.M.); javier.bermejo@unir.net (J.B.H.); juanantonio.sicilia@unir.net (J.A.S.M.)
* Correspondence: juanramon.bermejo@unir.net

Abstract: The problem of digital identity acquires more relevance every day in the eyes of a society that spends more and more time connected to the Internet. It has evolved throughout its history to reach a decentralized model known as Self-Sovereign Identity (SSI), which finds its natural tools in the blockchain technology and Zero-Knowledge Proofs (ZKPs). ZKPs, in this context, allow users to prove that their credentials are legitimate without revealing more information than is strictly necessary, and constitute one of the most promising areas of applied cryptography. In this work, an application is developed for the study of Zero-Knowledge Proof methods and, specifically, in their application for authentication in public-private key encryption systems. It focuses on the study of three ZKP protocols (Feige-Fiat-Shamir, Guillou-Quisquater, and Schnorr, which rely on the problems of large number factorizations and discrete logarithms for security) in the practical use-case where a prover wants to demonstrate knowledge of a private key for a public key without revealing the key itself. The application allows the user to modify the necessary parameters in each method to achieve a better understanding of their role in their safety and efficiency. Several types of attacks are carried out against the above-mentioned protocols to analyze their degree of security and what recommendations can be made to improve it.

Keywords: educational software; Self-Sovereign Identity; Zero-Knowledge protocols

check for updates

Citation: Moya, C.V.; Bermejo Higuera, J.R.; Bermejo Higuera, J.; Sicilia Montalvo, J.A. Implementation and Security Test of Zero-Knowledge Protocols on SSI Blockchain. *Appl. Sci.* **2023**, *13*, 5552. https://doi.org/10.3390/app13095552

Academic Editor: Luis Javier Garcia Villalba

Received: 23 March 2023
Revised: 23 April 2023
Accepted: 27 April 2023
Published: 29 April 2023

1. Introduction

The Internet has taken on an essential role in today's society; thanks to the Internet of Things (IoT), it is encompassing areas of our day-to-day lives that could never have been imagined in its initial conception and, thanks to advances in technologies such as Artificial Intelligence and Big Data, its functions and capabilities are also reaching far beyond expectations. At the same time, digital identity and user data privacy have become major concerns in today's digital age. With the increasing use of the internet and technology, individuals and organizations alike are becoming more vulnerable to security breaches and identity theft. For example, in 2017, Equifax, one of the largest credit reporting agencies in the US, suffered a massive data breach that compromised the personal information of over 140 million people [1]. This breach included sensitive information such as social security numbers, birth dates, and addresses. Another example is the Cambridge Analytica scandal, where the data of millions of Facebook users was harvested without their consent and used for political purposes [2]. These incidents highlight the importance of protecting our digital identities and personal information online. As we continue to rely more and more on technology, the number of people concerned about the security of their personal data on the Internet is also increasing.

In response to this situation, the use of technologies that provide greater control to the user and seek to eliminate the need to rely on third parties to protect their information is becoming increasingly popular. This is the case of blockchain, whose applications today go beyond decentralized digital currencies and, more recently, Zero-Knowledge Proof systems. Blockchain technology is a way of recording information in a secure and tamper-proof manner. It works as a digital ledger that keeps track of transactions, contracts, and other information that can be shared among a network of users. What makes it special is that it uses cryptography to ensure that the information is protected and can only be accessed by those who have the proper permissions.

The ZKP or Zero-Knowledge Proof systems are cryptographic techniques that allow a prover to demonstrate knowledge of a particular fact or statement to a verifier without revealing any additional information beyond the fact or statement itself. This is achieved by having the prover generate a proof that satisfies a specific set of criteria, which the verifier can then use to verify the claim without learning anything else about the statement. Therefore, they allow the minimization and limitation of accessibility to data in distributed contexts, such as Internet services in general, cloud computing, etc. [3] In particular, by combining blockchain technology with ZKP protocols, it is possible to create systems that provide both the transparency and security of a blockchain with the privacy and confidentiality of ZKPs. This can be achieved by storing encrypted data on the blockchain and using ZKP protocols to prove the ownership or validity of the data without revealing the actual data itself. ZKP is already in use in cryptocurrencies such as Zcash, which uses a ZKP protocol known as zk-SNARKs [4] for transaction verification even once encrypted; however, the purpose of this work is to study the use of ZKP in Self-Sovereign Identity (SSI) blockchain solutions.

SSI supports the idea that each individual should have control over their online identity, as opposed to identity authentication handled by a centralized authority. The goal behind using ZKPs in SSI is to be able to verify our identity without giving out our personal information. At the same time, blockchain technology provides us with an immutable, secure, and auditable means of sharing public information, eliminating the figure of a third party (a certification authority) that verifies the validity of the identification.

Despite the growing popularity of ZKPs in recent years, and the consequent increase in the research of its applications, its study is still not common. Therefore, we present the objectives after the development of an application that implements three ZKP protocols, oriented to the study of these methods and their comparison.

The general objective of this work is to develop an application with a graphical interface aimed at students in areas related to computer security, such as computer engineering and mathematics, who are interested in the study of Zero-Knowledge Proof methods and, specifically, in their application for authentication in public-private key systems. It will focus on showing the Feige-Fiat-Shamir, Guillou-Quisquater, and Schnorr protocols step-by-step, allowing the user to modify the necessary parameters in each method to gain a better understanding of their role in the security and efficiency of these.

More specifically, the contributions of this work are:

- Studying the operation of the Feige-Fiat-Shamir, Guillou-Quisquater, and Schnorr protocols for their correct implementation in an authentication system;
- Creating an application that manages the Verifier and Prover roles and allows the user to observe the authentication protocols step-by-step, including the verification calculations performed by the Verifier and the communication between the two;
- Testing the security of the protocols against brute-force attacks that try to take advantage of the vulnerability of the ZKP methods against an adversary who manages to guess the challenge that the Verifier is going to demand from the Prover;
- Allowing the application to be executable in as many environments as possible. As it is a desktop application, this is achieved by its implementation in an executable environment in the main operating systems: Windows, OS, and Linux.

The paper has the following structure: Section 1 is an introduction to the paper and to the problem of Self-Sovereign Identity. It justifies the motivation of the work in the context of today's hyper-connected society where privacy has taken greater importance in the consciousness of citizens and presents the approach of the work. Section 2 consists of a deep dive into the current state of online identity management technologies and protocols, the role of blockchain technology in this field, and Zero-Knowledge protocols. Section 3 presents the methodology followed to test the security of Zero-Knowledge protocols. Thus, an application to test the security of ZKPs is proposed. Section 4 presents the use-case and requirement analysis for the proposed application prior to its development. Section 5 shows its main classes and interfaces. Section 6 executes the security tests using the developed software and presents a final reflection to summarize everything learned and the challenges encountered. Section 7 presents the main conclusions of the obtained results. Finally, Section 8 considers possible future lines of work.

2. Background

As Internet users, there are many occasions when we need to identify ourselves to access a service. In the case of registering on a social network, it is common to use aliases to protect our privacy and hide our personal information. However, when applying for a bank credit, we need to provide digital proof to verify our physical identity. As a result, the level of trust placed in our identification is different in each case.

In the physical world, a person can prove their identity with their ID, driver's license, or similar documents. These have in common that:

- They are issued by a trusted authority; whether it is a Public Administration, an autonomous body or a private company, the validity of the certificate is directly related to the trust granted to this third party.
- They are difficult to falsify; the measures taken to prevent the copying or modification of these documents are numerous and increasingly elaborate.
- They are recognized on a large scale; for example, the Spanish DNI is a valid identification document within the entire European Union.
- They have the above three properties which characterize physical identification methods that are difficult to replicate in the digital world.
- Technology facilitates the falsification of documents; a bank would not accept a photograph of a DNI, as it may have been edited.
- There is no single valid identification method for all services; users are forced to create a specific account for each website where they want to log in.

The problem of digital identity has been present since the Internet was made public, and as such has evolved throughout history. We can speak of three models of digital identity [5]:

- Centralized Identity: This was the first model to be used and the most similar to physical identity models. An identity is established by creating an account with a service, web, or application. If the account on the provider is deleted, or if the provider suffers an attack that causes a loss of information, our identity disappears. This model has several disadvantages:
  - Forces the user to remember their credentials for each site they want to access, possibly with different security requirements.
  - Identity profiles are not transferable or reusable.
  - Centralized user databases are attractive targets for cybercriminal attacks.

The administration of a user's accounts in all of the different services where they are registered can become a real challenge, especially if all of the security recommendations on password policies are considered.

- Federated Identity: This model aims to improve the multiple-accounts problem of the centralized model by inserting an intermediate agent: the Identity Provider (IDP). The idea is that multiple services trust an IDP in such a way that the user proving their identity to the latter is enough to access the former. Protocols such as SAML, OAuth, and OpenID Connect have successfully implemented this identity model. However, there is no global IDP that works with any application or web, and a new problem arises related to privacy and the monitoring of user activity: the position of the IDP as an intermediary in our relationships with different services gives it privileged information about our online activity.

- Decentralized Identity: The third identity model stems from the idea of eliminating the need to trust an identity provider authority, allowing the user to be the owner and creator of their own identity. This is where the concept of SSI arises. The user does not have to register an account in each of the services that they wish to use, but rather has the power to manage their different identity credentials via their virtual wallet, equivalent to the identity documents of the physical world, and use them to establish a reliable connection with the other parts. These virtual credentials are one of the main pillars of SSI and are known as Verifiable Claims.

2.1. Verifiable Credentials

We distinguish three essential roles involved in the exchange of verifiable credentials (VCs):

- Issuer: entity that generates and certifies verifiable credentials;
- Holder: entity that requests a credential and stores it in their digital wallet;
- Verifier: entity that requests a credential to verify information about a user.

A VC has the same role as the physical credentials we use every day: to prove information about our identity to an agent who does not take our word for it. It may contain information about the identity of the subject of the credential (name, photo, ID), information about the entity that issued the credential, and about the specific attributes that it is attesting to about the subject. A VC makes use of technology to increase its reliability and immutability.

2.2. Zero-Knowledge Proof

Now that the concept of SSI and the technology that makes its implementation possible today have been explored, what remains to be understood are the cryptographic mechanisms that allow users to prove that their credentials are legitimate without revealing more information about their identity than is strictly necessary.

A Zero-Knowledge Proof, or Zero-Knowledge protocol, is the method proposed by Goldwasser, Micali, and Rackoff in 1989 [6] by which an actor (the Prover) can prove to another (the Verifier) the veracity of a statement about a secret, without revealing the secret itself. After the execution of the ZKP protocol, the Verifier will not have gained any additional knowledge about the secret than they had to start with, other than the fact that the statement is true.

A ZKP test must meet three basic characteristics:

- Completeness: If the statement is true, the verifier can be convinced of it by the prover.
- Solvency: If what you want to prove is false, the verifier cannot be deceived.
- Zero-Knowledge: The verifier will not have gained additional information except the truth of the statement.

The first two attributes are required by any verification system; it is zero knowledge that characterizes a ZKP method.

A simple example of how a ZKP works would be the following situation: imagine Alice wants to prove to Bob, who is colorblind, that she is able to distinguish between two pens, one red and one blue, without giving him any information about which is which. To perform this, Alice could ask Bob to take a pen in each hand, hide them behind his back, and choose to switch hands or not. Bob then shows Alice the pens again and she will tell him if he has switched them or not. At first, even if Alice got it right, Bob might think it was a fluke, since Alice has a 50% chance of getting it right whether she really knows the answer. However, if they repeat the protocol numerous times, Bob will become more and more convinced that Alice is telling the truth, until after n repetitions the probability that Alice's statement is false is so small (1 in 2^n) that Bob should be convinced.

Now that we have seen a rather metaphorical example of a ZKP protocol, it is interesting to study a real protocol that can be implemented in a computer system.

2.2.1. Feige-Fiat-Shamir Protocol

Let $n = p{\cdot}q$ be the product of two big prime numbers.

Let $s_1, s_2, \ldots, s_k$ be numbers in $\mathbb{Z}_n$ which Alice knows but wants to keep secret.

The FFS protocol allows Alice to prove to Bob that she knows these numbers in two steps, as listed in Table 1.

Table 1. Feige-Fiat-Shamir Protocol Rokeprinted/adapted with permission from Ref. [7]. 2020, Trappe, W. and Washington, L.C.

Feige-Fiat-Shamir Protocol	
1. Alice	Computes $v_i = s_i^{-2}, i = 1, \ldots, k$ Computes $x = r^2 \; mod \; n$ for some random number r Sends $x, v_1, \ldots, v_k$ to Bob
2. Bob	Generate $b_i \in \{0, 1\}, i = 1, \ldots, k$ and sends them to Alice
3. Alice	Computes $y = r{\cdot}\prod_1^k s_i^{b_i}$
4. Bob	Verifies that $y^2{\cdot}\prod_1^k v_i^{b_i} = x$

If Alice is being truthful and knows $s_1, s_2, \ldots, s_k$, then it is certain that Bob's verification will turn out true. However, if they repeat the protocol numerous times, Bob will become more and more convinced that Alice is telling the truth, until after n repetitions the probability that Alice's statement is false is so small (1 in 2^n) that Bob should be convinced.

$$y^2{\cdot}\prod_1^k v_i^{b_i} = \left(r{\cdot}\prod_1^k s_i^{b_i}\right)^2 \prod_1^k v_i^{b_i} = r^2{\cdot}\left(\prod_1^k s_i^{b_i}\right)^2 \prod_1^k v_i^{b_i} = r^2 = x$$

Otherwise, Alice would need to guess the indices b_i that Bob is going to choose before computing x; she could then compute $x = a^2\prod_1^k v_i^{b_i}$ for a random number a and answer Bob's challenge with $y = a$, satisfying the equality check. However, an impostor pretending to be Alice and not correctly guessing the indices that Bob is going to choose will have to compute a root for a number of the form $\prod_1^k v_i^{b_i}$ which, by definition and at least with today's computing power, is a difficult task.

The total number of possible values of the indices b_i is 2^k; if the protocol is repeated t times, there is only a 1 in 2^{kt} chance that Bob will be fooled, so he can rest easy.

2.2.2. Guillou-Quisquater Protocol

Louis Guillou and Jean-Jacques Quisquater proposed in 1988 an alternative to the Feige-Fiat-Shamir protocol which reduces the number of messages exchanged between the Prover and the Verifier, as well as the memory requirements, in exchange for increasing the computational work by a factor of three [8].

Again, let n be the product of two prime numbers. Let v be the coprime to $\phi(n) = (p-1)(q-1)$.

Let J be Alice's public key; then, let us define her private key s such that $J{\cdot}s^v = 1 \; mod \; n$. For Alice to prove that she owns the public key J, she will prove she knows s, as shown in Table 2.

The verification is straightforward:

$$y^v{\cdot}J^e = r^v{\cdot}s^{ev}{\cdot}J^e = r^v(J{\cdot}s^v)^e = r^v \; mod \; n = x$$

The security of this protocol resides in the parameter v, since it defines the range of values between which Bob can choose his challenge e and, consequently, the probability that an impostor will be able to guess it. If Eve were to guess the value of e, she could impersonate Alice even without knowing her private key s. She can choose a random y value in advance that will be the one she sends in step three and define $x = y^v{\cdot}J^e$ in step one accordingly; in this way Bob's check in the final step is trivially true. Recall that, although Eve does not know Alice's private key, she does have access to her public key.

Table 2. Guillou-Quisquater Protocol. Reprinted/adapted with permission from Ref. [9]. 2015, Schneier, B.

Feige-Fiat-Shamir Protocol	
1. Alice	Publishes her public key $[n, v, x]$, where $x = r^v$ for some random number r, with $1 < r < n - 1$
2. Bob	Generates a random number e such that $1 < e < v - 1$ Sends e to Alice
3. Alice	Computes $y = r \cdot s^e$ and sends y to Bob
4. Bob	Verifies that $y^v \cdot J^e \ mod \ n = x$

2.2.3. Schnorr Protocol

Claus Schnorr's authentication scheme bases its security on the discrete logarithm problem. Let p, q be two prime numbers, where $q \mid p - 1$, and let $a \neq 1$ be a random number such that $a^q = 1 \ mod \ p$. Given Alice's public key s, we define her private key $v = a^{-s} \ mod \ p$. Table 3 presents the protocol steps.

Table 3. Schnorr Protocol. Reprinted/adapted with permission from Ref. [9]. 2015, Schneier, B.

Schnorr Protocol	
1. Alice	Computes $x = a^r \ mod \ p$ for a random r Sends x to Bob
2. Bob	Generates a random number e such that $1 < e < 2^i - 1$ Sends e to Alice
3. Alice	Computes $y = (r + s \cdot e) \ mod \ q$ Sends y to Bob
4. Bob	Verifies that $a^y \cdot v^e = x$

Indeed,

$$a^y \cdot v^e = a^{r+se} \cdot a^{-se} = a^r = x$$

Similar to the Guillou-Quisquater protocol, the security of Schnorr's protocol lies in the probability that a third party guesses the challenge e, which in this case is related to the parameter i in step two. Otherwise, the impostor could choose any random value for y and define $x = a^y \cdot v^e$. The probability that Eve guesses the value of e before it being announced is 2^{-i}; Schnorr recommended that i be a 72-bit number [9].

2.3. Benefits and Profits

According to [10–12], the benefits of SSI include:

- Owner-centric: The owner is the root of trust for their devices. Once a user is identified as the owner and can verify their identity, the next step is only to create a network of devices where the common parameter among all is the same owner. This way no third parties are needed to control the device. These devices can also have their own identities that allow them to create trust relationships, for example, devices that have the same owner automatically trust each other.
- Privacy: Users' identities are stored locally, whereas before they were stored at a service provider. Thus, it is located in a place closer to the user, usually in his wallet, and over which they have full control. Furthermore, encrypted identities could also be stored with the user's private keys.
- Decentralization: With the use of SSI, the user is the one who decides how and when to use their IoT devices without the intervention of any third party. It also provides an additional layer of security as the device information is not hosted anywhere outside.

- End-to-end security: By exchanging decentralized identifiers applying asymmetric cryptography, devices authenticate and exchange messages securely.
- Standardization and open source: Decentralized identifiers are being developed according to W3C specifications, where companies as well as independent researchers are free to suggest new approaches.

2.4. Related Work

The currently available bibliography on Zero-Knowledge is extensive and varied. There are numerous ZKP implementation projects accessible in public repositories:

- The ING group published ZKKrypto, a library of ZKP algorithms based on different hash algorithms and elliptic curve cryptography. It is written in Kotlin, a programming language interoperable with Java. Although instructions on how to install the library are provided, it does not explain the operation of the implemented protocols and it is difficult to analyze for a person who is not an expert in the field.
- NoKnow is a Python library [13] that implements the Schnorr protocol in its non-interactive version based on the elliptic curve logarithm problem. It is based on the implementation described by Chatzigiannakis et al. [14]. It provides a simple and intuitive API, but it is still too high a level for a first approach to the Zero-Knowledge world.

In general, there is an effort in the scientific community to reach a standard regarding the implementation of Zero-Knowledge Proof protocols. However, they are not very friendly for people with less expertise on the subject.

On the other hand, there are also numerous publications explaining the fundamentals and advantages of using this methodology, specifically in application to Self-Sovereign Identity.

Yadav et al. [15] analyze the weakness of the NTRF key exchange protocol against Man-in-the-Middle attacks even after ZKP has been applied to the algorithm, finding that it is indeed vulnerable. They explain what the attack steps would consist of one by one, but without actually showing a specific practical example.

Kayathri Devi D et al. [16] make a comparison of the efficiency of the Feige-Fiat-Shamir and Guillou-Quisquater protocols as authentication protocols on a web server. They explain the steps of the protocol without going into the mathematical details or why they work. They carry out an analysis of the efficiency of the protocols by measuring the time spent by the Prober and the Verifier, but do not provide a tool that the reader can use to carry out their own experiments.

Daniele Raffo exposes in [17] an implementation of the Feige-Fiat-Shamir protocol, exploring its possible use in the implementation of an alternative SSH authentication scheme. He does not carry out a study of the possible vulnerabilities of this protocol, referring only to its weakness against MITM attacks.

The goal of the study of the sovereign management of identities decentralized with Blockchain [18] is to create a system that can manage our digital identities in a similar way to how we manage our physical ones. This system would allow us to prove our identity online just like we do in the real world. The authors achieve this by building a virtual environment within the Sovrin Blockchain and leveraging the Hyperledger Indy protocols to create a practical example of this innovative identity management system. The authors of this work have chosen to exclude IoT devices from their study.

There are works on Self-Sovereign Identity that address the issue from a legal perspective. That is, looking at the way in which such a system could be implemented in public entities such as, in this case, the Spanish government and the European Union (EU). This is the subject of the article by blockchain lawyer Ignacio Alamillo [19]. In this article, the author talks about the main system of digital identity in the Spanish Public Administration, Cl@ve system, which involves several entities and third parties. In the article, the author already anticipates the benefits of SSI versus Cl@ve: in a Self-Sovereign Identity system, given that the data subject already possesses the identity data and other authenticated attributes, the user can identify themselves without the intervention of the identity issuers (p. 5). The author then discusses the two main regulations on the electronic relationship between citizens and the Public Administration. They are found in Article 9 of Law 39/2015, of 1 October 2015 on the Common Administrative Procedure of Public Administrations [20], and in Directive 1999/93/93/EC of the European Parliament and of the Council [21]. Directive 1999/93/EC is known as the Electronic Identification and Trust Services (eIDAS) Regulation. These regulations are studied in depth in order to identify whether an SSI system is compatible with them and with the National Security Scheme (ENS) [22] in order to comply with the requirements necessary to conform an information system in the Public Administration. The conclusion of this article would be that an SSI model would be compatible with the Spanish regulation, although it would need some small adjustments to comply with the eIDAS regulation. Because this article focuses on the legal aspects of the problem, no case of actual and functional use is proposed to create said SSI system.

Other authors see the use of Self-Sovereign Identity as an opportunity to achieve improvements and upgrades in the industrial sector in order to increase performance. Following the concept of Industry 4.0, there is an attempt to implement novel technologies in the industry in IoT networks, which will add much more time to the life cycle of the machinery while ensuring higher safety standards. The work of some authors from the University of Aveiro in Portugal [23] addresses the issue of Self-Sovereign Identity in the industry environment to achieve competitive advantages over the rest. They believe that using IoT device networks managed with an SSI system would bring economic benefits, such as better risk management and increased productivity. However, their work is primarily focused on the private sector, and the technology is not being considered as a benchmark available to everyone in the society of the future.

As for the literature on quantum blockchain and quantum signatures, the authors of [24] propose a new quantum Byzantine agreement protocol that leverages quantum digital signatures and the recursion method to achieve almost one-half fault-tolerance and unconditional security. The proposed protocol overcomes the one-third fault-tolerance bound of a classical Byzantine agreement and addresses the security loopholes of classical cryptography methods. The protocol also includes a consistency check between each pair of rounds to ensure the unforgeability and nonrepudiation throughout the whole process. The authors experimentally demonstrate the proposed protocol's effectiveness at achieving a three-party and five-party quantum consensus for a digital ledger, highlighting the potential of quantum blockchain and quantum consensus networks.

Yin, H.L. et al. [25] propose a high-efficiency quantum digital signature (QDS) protocol that achieves integrity, authenticity, and non-repudiation of data with information-theoretical security. The protocol uses asymmetric quantum keys acquired via secret sharing, one-time universal2 hashing, and one-time pad, requiring only a 384-bit key to sign documents of up to 2^{64} lengths with a security bound of 10^{-19}. The proposed protocol achieves a signature efficiency that is more than 10^{8} times higher than previous QDS protocols when signing a one-megabit document. The authors also build the first all-in-one quantum secure network integrating information theoretically secure communication, digital signatures, secret sharing, and conference key agreement, and experimentally demonstrate this signature efficiency advantage. However, the limitations of the proposed protocol need to be explored further, and its implementation in practical scenarios needs to be evaluated. Table 4 resumes related work.

The present work aims to improve or expand the aforementioned works in the following aspects:

- Greater transparency and clarity for the non-expert user who is interested in learning about ZKP protocols.
- A more practical application that allows the user not only to understand the theory, but also to see it in operation and perform their own tests.

- Offers a wide enough variety of ZKP protocols so that the user acquires a natural intuition about how they work and can compare them.

Table 4. Related Work.

Reference	Proposed	Finding	Limitation
[13]	NoKnow: A python library for implementation of Schnorr protocol	Intuitive API	Too high level for first approach to ZKP
[15]	Weakness analysis of NTFR protocol against MITM	Step-by-step theoretical explanation	No practical example that helps understanding
[17]	Implementation of Feige-Fiat-Shamir protocol	Proposal for use in the implementation of an alternative SSH authentication scheme	No weakness analysis
[19]	Analysis on the benefits of SSI vs. Spanish system Cl@ve	In-depth analysis of necessary changes needed to abide with EU regulations	Legally focused, lacking a practical proposal
[23]	Study of SSI in industrial environment	Arguments in favor of IoT device networks managed with an SSI system which would bring economic benefits	Their work only benefits private sector; tech is not accessible to all.
[24]	High-efficiency Quantum Digital Signature	The protocol proposed is 10^8 times more efficient than previous QDS protocols	Limitations of the proposed protocol need to be explored further

As we have seen, a more practical approach that allows us to understand the algorithm behind a ZKP protocol, seeing its execution step-by-step and verifying its security, has not been found. This is the main objective of this work: to provide students who are entering the world of Zero-Knowledge tests with a tool to verify in a practical way the usefulness and operation of these protocols.

3. Method

The process used for testing the security of ZKP protocols can be summed up as:

1. Development of a specific test environment that allows the implementation of attacks against the ZKP protocol and testing its security status.
2. Execution of specific attacks and security tests against the ZKP protocol using the developed software. The security tests to be performed are evaluations of the correctness of the ZKP protocols and User Input Verifications. The attacks to be performed are brute-force attack analysis and MITM attack analysis.
3. Discussion of the results.

4. Test Environment Development

The work method followed was one typical of the application development workflow: firstly, the specific objectives and use cases were defined. From these, the final requirements for the application were extracted and the classes to be implemented were defined in a UML diagram. Figure 1 shows the development process.

Use Cases
UML
Specific goals
Requirements
Implementation

Figure 1. App Development.

Finally, the implementation stage of the workflow was reached, which will be further explained below.

Once the algorithms were studied in the background section, the steps described had to be translated into executable code. With the aim of making the application executable on the main operating systems, it has been decided to work with Python due to its versatility as an open-source multi-platform language. It also offers a solution for the development of the user interface with the tkinter package, which is easy to install. To mimic the Verifier and Prover roles, the two sides of the protocols' logic will be implemented in two separate processes that will communicate over a TCP connection so that the final application may resemble a real-life situation.

Once these design decisions have been made, an incremental sequential work method is proposed which consists of the following steps:

1. Use cases and requirements definition.
2. Development environment configuration: This involves the installation of the latest version of Python (Python 3) and the Visual Studio Code source code editor, as well as the necessary packages for application development (tkinter, socket).
3. Development of the Verifier and Prover processes: Firstly, it is necessary to establish the TCP connection between both processes before encoding the logic of the protocols in each one of them.
4. Implementation of the ZKP protocols: The implementation of each of the three ZKP methods is independent of the rest, so they can be performed in parallel. However, it is necessary to develop the logic on the side of the Prover and the Verifier synchronously in order to verify the communication between them and the correct operation of the protocol.
5. ZKP protocol tests: Ensure the correct authentication of the Prover with each of the methods when changing in the input parameters established by the user.
6. Implementation of brute-force attacks: Once the correct operation of the protocols has been verified, it is a matter of implementing a third actor, the impostor, who tries to pass the Verifier test with only the information about the public key of the Prover, and not their private key.
7. Implementation of the Verifier interface: The work of the Verifier will be indistinguishable whether an honest user or an attacker is trying to pass the test, so it will have a single interface that will allow the user to choose the protocol with which the Prover is expected to work. It will also be from this interface where the general parameters of the specified protocol are configured. In addition, it will be allowed to choose the listening port for the TCP connection.
8. Implementation of the Prover's authentication interface: The Prover's interface must allow the user to configure the address of the Verifier to which they want to connect, indicating its IP address and listening port. The ZKP protocol with which they want to work will also be selected.
9. Implementation of the brute-force attack interface: From the Prover's menu, it will be possible to access the brute-force attack functionality.

Use Cases

The four main use cases that were identified for the application are presented below. Tables 5–7 present the use cases for the execution of the three selected ZKP protocols, while Table 8 corresponds to the use case where the user wants to perform an attack on these protocols.

Table 5. Use Case 1: Feige-Fiat-Shamir.

UC1—Feige-Fiat-Shamir	
Name	Feige-Fiat-Shamir Authentication
Description	Step-by-step visualization of the Feige-Fiat-Shamir protocol
Event stream	1. User selects the "Authentication" option. 2. User selects Feige-Fiat-Shamir protocol. 3. User inserts the parameters: - p,q: prime factors of n; - k: private/public keys length; - t: number of trials demanded by Verifier. 4. User is assigned a private and public key accordingly. 5. The program shows the steps followed during the authentication process.

Table 6. Use Case 2: Guillou-Quisquater.

UC2—Guillou-Quisquater	
Name	Guillou-Quisquater Authentication
Description	Step-by-step visualization of the Guillou-Quisquater protocol
Event stream	1. User selects the "Authentication" option. 2. User selects the Guillou-Quisquater protocol. 3. User inserts the parameters: - p,q: prime factors of n - v: exponent coprime to $\phi(n) = (p-1)(q-1)$; - t: number of trials demanded by Verifier. 4. User is assigned a private and public key accordingly. 5. The program shows the steps followed during the authentication process.

Table 7. Use Case 3: Schnorr.

UC3—Schnorr	
Name	Schnorr Authentication
Description	Step-by-step visualization of the Schnorr protocol
Event stream	1. User selects the "Authentication" option. 2. User selects the Schnorr protocol. 3. User inserts the parameters: - p,q: prime numbers, such that q is a factor of p $-$ 1; - a: $a \neq 1$, such that $a^q = 1$ mod; - t: number of trials demanded by Verifier; - i: exponent which defined the range of values for e. 4. User is assigned a private and public key accordingly. 5. The program shows the steps followed during the authentication process.

Table 8. Use Case 4: Brute-Force Attack.

UC4—Brute-Force Attack	
Name	Brute-Force Attack
Description	Step-by-step exposition of the operation and the efficiency of a brute-force attack when stealing an identity in an authetication protocol
Event stream	1. User selects the "Brute-force attack" option. 2. User chooses which protocol they want to work against. 3. User inserts the authentication protocol parameters similarly to the authentication use case. 4. User is assigned private and public keys accordingly. 5. The program shows the steps followed during the attack as well as time-taken statistics.

5. Classes Architecture

The structure of the program revolves around two main classes: the Verifier class, which implements all of the logic of the Verifier agent, and the Prover class, which does the corresponding for the Prover. Due to the differences in the general parameters needed for each ZKP protocol, subclasses are also defined for each of them that inherit their parameters from the parent class Verifier or Prover.

5.1. Verifier Classes

The Verifier class has the parameters common to all protocols as attributes:

- p,q: Prime numbers that define the module on which we will work.
- server_socket: Communication socket with the Prover.
- pub_key: Prover's public key that they are trying to authenticate.

It contains the following methods:

- _init_ (self, port): Class constructor receives the port number where the connection will be established.

- listen (self, port): Initializes the socket, waits to accept a connection, and receives the port number where the socket should be established.
- layout (): Initializes the general screen for a generic Verifier.
- main (self, protocol): Initializes the protocol requested by the interface.

Classes FFS_Verifier, GQ_Verifier, Sch_Verifier

These particular classes for each of the protocols are subclasses of the Verifier class and have the same structure, with the only difference being the specific parameters that each of them needs to function.

Methods:

- _init_ (self, port, n, p, q, k, t): Constructor of the class. It receives the port where the connection will be established and the list of particular parameters of the specific protocol: n, p, q, k, t for Feige-Fiat-Shamir; n, p, q, t for Guillou-Quisquater; p, q, a, t for Schnorr. It makes a call to the parent constructor and executes the layout function to customize the screen to the specific protocol.
- layout_ffs (self): Function that adds the graphical interface elements specific to the specific protocol (protocol name, parameter selection panel). It corresponds to the functions layout_gq, layout_sch in the other subclasses.
- saveButton (self): Function that checks the validity of the parameters entered by the user and saves them as global parameters if they are valid.
- sendParams (self): Function that sends the list of protocol parameters through the communication socket to the Prover at the beginning of the process.
- ffs_protocol (self): Function that contains the logic of the Verifier for the execution of the protocol. It corresponds to the gq_protocol, sch_protocol functions in the other subclasses.

5.2. Prover Class

Analogous to the Verifier class, the Prover class is the parent class for the subclasses that implement the Prover-side logic of each protocol.

Attributes:

- IP: IP address of the Verifier's machine before which one wants to prove their identity.
- port: Port of the Verifier to which it must connect.
- socket: Communication pipe with the Verifier.

Methods:

- _init_ (self, protocol, port, id): Constructor of the class. Calls the layout and listen functions.
- listen (self, port, id): Function that establishes the connection with the Verifier server at the IP address and port received as parameters.
- layout (self): Initializes the screen for the Prover, which shows the exchange of messages with the Verifier and the keys with which it is trying to authenticate.
- main (self, protocol): Initializes the requested protocol.

5.2.1. Classes FFS_Prover, GQ_Prover, Sch_Prover

These classes inherit from the Prover class and contain the logic of the Prover side for each of the protocols.

Functions:

- _init_ (self, port, ip): Constructor of the class. Calls the constructor of the Prover parent class.
- receiveParams (self): Receives the list of parameters through the connection with the Verifier and assigns them to their attributes.
- key_generator (self): Function in charge of calculating a suitable public and private key for the specific protocol and with the parameters defined by the Verifier.

- ffs_protocol (self): Function that contains the Prover logic of the specific protocol. Corresponds to the functions gq_protocol, sch_protocol.

5.2.2. FakeProver Class

This is the class that implements the logic of brute-force attacks on protocols. It has the same attributes and functions as the Prover class and only differs from it in the logic implemented in the protocol function in each of the subclasses (ffs_protocol, gq_protocol, sch_protocol), where instead of using the private key to get authenticated, it will try to guess the challenge sent by the Verifier.

5.2.3. FFS_FakeProver, GQ_FakeProver, Sch_FakeProver Classes

These classes inherit from the FakeProver class and contain the logic to perform a brute-force attack for each of the protocols.

5.2.4. GUI_Verifier Class

This class is responsible for creating the home screen of the Verifier.
Methods:

- _init_ (self): Constructor that creates the protocol selection window for the Verifier.
- zkp_protocol (self, zkp): Initializes an object of the Verifier subclass corresponding to the selected protocol.

5.2.5. GUI_Prover Class

This class is responsible for creating the Prover home screen.
Methods:

- _init_ (self): Constructor that creates the protocol selection window for the Prover.
- zkp_protocol (self): Initializes an object of the Prover or FakeProver subclass, depending on the selected protocol and functionality.

5.3. Application Interface

For the correct execution of the protocols, it is necessary for the user to execute the application in two independent processes, one that acts as a Verifier and the other as a Prover.

5.3.1. Verifier Interface

The Verifier interface is activated with the python main.py v command (Figure 2). The first screen that appears corresponds to the initial configuration screen of the Verifier. The user can choose the port through which the TCP connection will be established—if none is entered, port 42,424 will be used by default. When choosing one of the protocols by clicking on its corresponding button, the window of the specific protocol is accessed. As an example, the screen corresponding to the Feige-Fiat-Shamir protocol is shown in Figure 2.

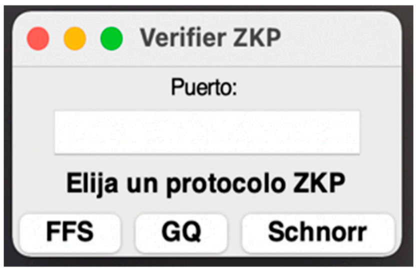
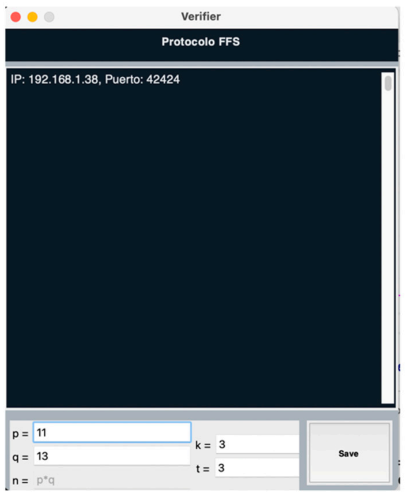


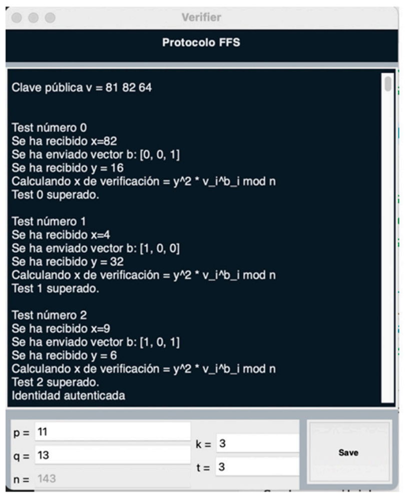
Figure 2. Verifier Home Screen.

From the specific protocol screen (Figure 3), the Verifier's waiting room is shown in chatroom mode, which has already opened a TCP socket in the direction shown above and will remain waiting until a Prover connects. In addition, at the bottom of the screen the user can configure the specific parameters of the current protocol; in this case, for the Feige-Fiat-Shamir method, these are the primes p, q factors of the number n that will act as module, as well as the parameters k, t that determine the length of the public key and the number of times that is required to pass the verification algorithm successfully, respectively.



```
IP: 192.168.1.38, Puerto: 42424
```

p = 11
q = 13
n = p*q
k = 3
t = 3



```
Clave pública v = 81 82 64


Test número 0
Se ha recibido x=82
Se ha enviado vector b: [0, 0, 1]
Se ha recibido y = 16
Calculando x de verificación = y^2 * v_i^b_i mod n
Test 0 superado.

Test número 1
Se ha recibido x=4
Se ha enviado vector b: [1, 0, 0]
Se ha recibido y = 32
Calculando x de verificación = y^2 * v_i^b_i mod n
Test 1 superado.

Test número 2
Se ha recibido x=9
Se ha enviado vector b: [1, 0, 1]
Se ha recibido y = 6
Calculando x de verificación = y^2 * v_i^b_i mod n
Test 2 superado.
Identidad autenticada
```

p = 11
q = 13
n = 143
k = 3
t = 3

Figure 3. Verifier Screen.

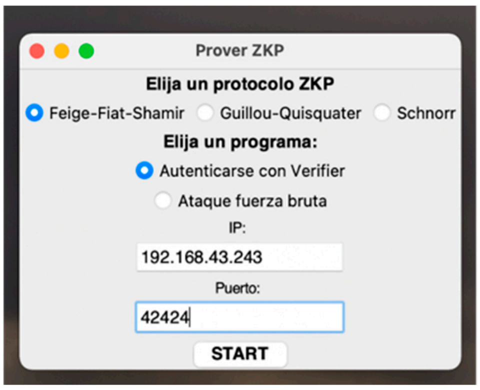
The Save button stores the data entered in the system after checking its correctness. These parameters are shared with the Prover that establishes a connection to calculate their public/private keys. Once a Tester connects to the Verifier socket, the execution of the verification protocol begins. The chat window shows the steps performed by the Verifier.

5.3.2. Prover Interface

The Prover interface is activated with the python main.py p command. It allows the user to choose the ZKP protocol to be used, as well as the program to run:

- Authenticate with Verifier: The Prover generates public/private keys according to the chosen ZKP method and executes the authentication protocol with the global parameters shared by the Verifier.
- Brute-force attack: The Prover acts as an impostor, generating a public key and trying to get the Verifier to authenticate it without using the private key, but rather by predicting the challenge that it will send and preparing the verification variables accordingly.

In both cases, for a correct TCP connection it is necessary to enter the Verifier's IP address and listening port. This screen is shown in Figure 4.



Elija un protocolo ZKP

[✓ Feige-Fiat-Shamir] Guillou-Quisquater Schnorr

Elija un programa:

[✓ Autenticarse con Verifier]

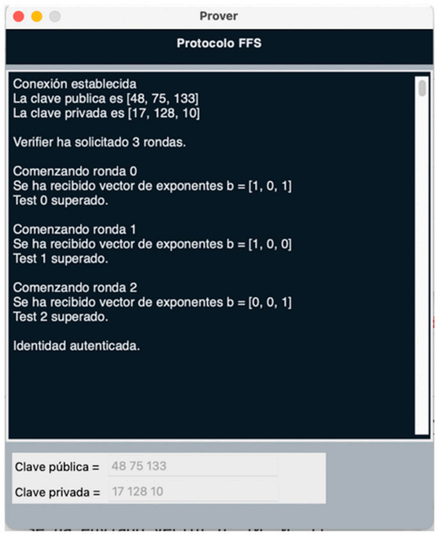
Ataque fuerza bruta

IP: 192.168.43.243

Puerto: 42424

Figure 4. Prover Home Screen.

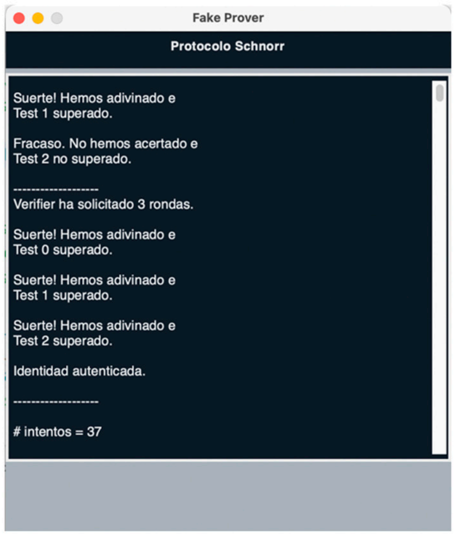
When in Authenticate with Verifier mode, the Prover screen shows the automatically generated public and private keys and each round of the protocol, as shown in Figure 5a.



```
Conexión establecida
La clave publica es [48, 75, 133]
La clave privada es [17, 128, 10]

Verifier ha solicitado 3 rondas.

Comenzando ronda 0
Se ha recibido vector de exponentes b = [1, 0, 1]
Test 0 superado.

Comenzando ronda 1
Se ha recibido vector de exponentes b = [1, 0, 0]
Test 1 superado.

Comenzando ronda 2
Se ha recibido vector de exponentes b = [0, 0, 1]
Test 2 superado.

Identidad autenticada.
```

Clave pública = 48 75 133
Clave privada = 17 128 10

(**a**)



```
Suerte! Hemos adivinado e
Test 1 superado.

Fracaso. No hemos acertado e
Test 2 no superado.

------------------
Verifier ha solicitado 3 rondas.

Suerte! Hemos adivinado e
Test 0 superado.

Suerte! Hemos adivinado e
Test 1 superado.

Suerte! Hemos adivinado e
Test 2 superado.

Identidad autenticada.

------------------

# intentos = 37
```

(**b**)

Figure 5. (**a**) Prover screen (**b**) FakeProver screen.

When in Brute-Force Attack mode (Figure 5b), a message is displayed for each round indicating whether the Verifier challenge has been successfully passed. If all rounds have been passed and, therefore, it has been possible to authenticate successfully, the number of attempts that have been necessary is displayed. This allows the user to compare the security of the protocol for different values of the parameters that can be configured from the Verifier screen, as well as make comparisons between the three protocols.

6. Security Tests and Evaluation

In this section, the tests carried out to verify the correct operation of the application are presented, as well as the analysis of the attacks on the implemented ZKP algorithms. Specifically, the following points are elaborated on:

- Evaluation of the correctness of the ZKP protocols;
- User input verification;
- Brute-force attack analysis;
- MITM attack analysis.

Finally, the results of the tests, lessons learned, and the conclusions drawn are summarized.

6.1. Experimental Setup

The computer used for the experiment was a Mac Air (M1, 2020) running on macOS Big Sur Version 11.7.1. For the programming language, Python 3.10.4 was used to implement the ZKP protocols and simulate the brute-force attacks.

To ensure that the experiment was conducted in a controlled environment, the computer was not connected to the Internet during the experiment. The experiment code was run from the local disk.

6.2. Evaluation of the Correctness of ZKP Protocols

The implementation of the application presented a series of challenges that conditioned certain design-related decisions.

Firstly, displaying the messages exchanged between the tester and the verifier, while essential to the application, has an impact on its performance. If the number of protocol repetitions requested by the verifier is low, the number of messages on the screen is manageable; but it can become infeasible, especially in the case of brute-force attacks. Showing the information exchanged between both actors allows the protocol to be transparent to the user, which is the main objective of the application, so the sacrifice in efficiency is assumed in exchange for providing more information.

6.3. User Input Verification

There are two sources of input external to the application: the configuration of the connection port (and IP address in the case of the Prover) and the parameter configuration. Allowing the user to establish the global parameters on which the protocol calculations are made forces a series of restrictions to be established. Specifically, for each of the protocols:

- Feige-Fiat-Shamir:
    - p, q must be two prime integers;
- Guillou Quisquater:
    - p, q must be two prime integers;
- Schnorr:
    - p, q must be two prime integers;
    - q must be divisor of p $-$ 1;
    - $a \neq 1$ such that $a^q = 1 \ mod \ p$.

The rest of the variables are calculated from these or randomly in each iteration of the protocol. If the entered parameters do not meet these restrictions, they should be discarded.

6.4. Brute-Force Attack Analysis

The application's brute-force attack functionality intends to emulate the scenario where a cybercriminal tries to impersonate a user without knowing their private information; that is, it is authenticated against the Verifier without knowing the private key. To perform this, it bases its calculations and the information sent to the Verifier on an assumption of what challenge the Verifier will send.

For each of the protocols, the probability that the imposter succeeds in guessing the Verifier's challenge is:

Feige-Fiat-Shamir: 1 in 2^{kt}, where k is the length of the key and t is the number of protocol rounds;

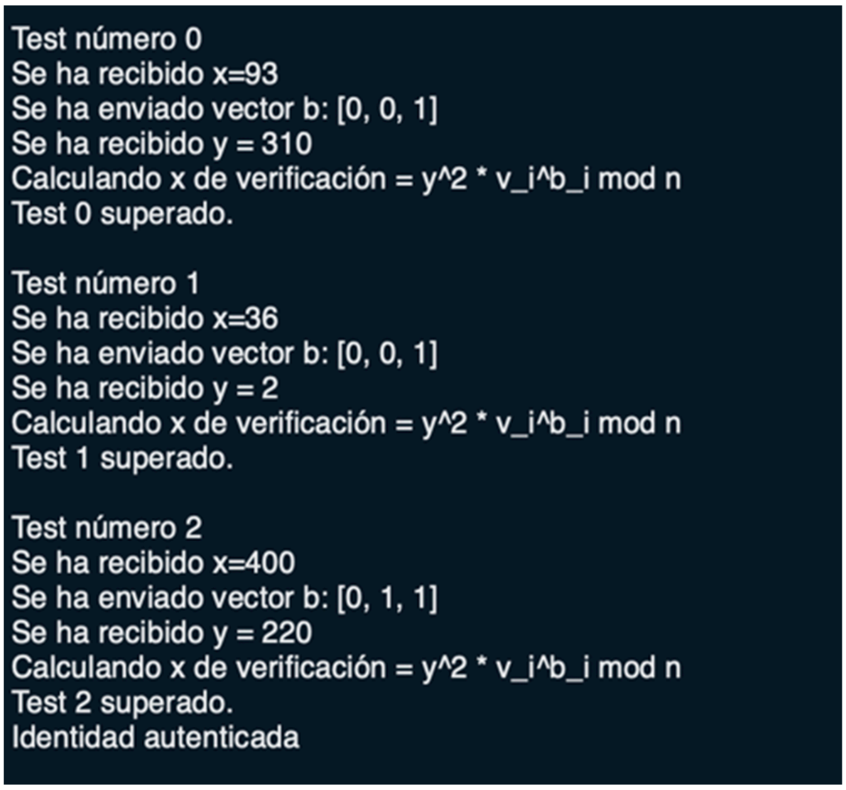
Guillou-Quisquater: 1 in v^t, where v is a defined parameter and t is the number of rounds of the protocol;

Schnorr: 1 in 2^{it}, where i is a defined parameter and t is the number of rounds of the protocol.

The tests carried out revealed a security factor that had not been considered.

During the execution of the Feige-Fiat-Shamir protocol with parameters $p = 5$, $q = 3$, $t = 3$, $k = 3$, the brute-force attack succeeded in an average of less than 10 attempts. Increasing the values to $p = 17$, $q = 31$ increased the mean number of trials to 230; and for $p = 173$, $q = 233$, the mean was 633. As explained in Section 2.2.1, the security of the FFS protocol resides in the value of the variables k and t, the key length and number of protocol iterations, respectively; however, these tests showed that the choice of prime numbers is also relevant.

One of the main bases to guarantee the security of computational cryptography is the choice of "large" prime numbers, i.e., with enough bits to make the calculations computationally expensive. In this case, the choice turns out to be just as relevant, since the choice of a small p, q increases the probability that the hypothetical attacker will be able to authenticate himself without even correctly guessing the verifier's challenge. Figure 6 shows an example of a trial where a FakeProver successfully verified their identity using the FFS protocol.



```
Test número 0
Se ha recibido x=93
Se ha enviado vector b: [0, 0, 1]
Se ha recibido y = 310
Calculando x de verificación = y^2 * v_i^b_i mod n
Test 0 superado.

Test número 1
Se ha recibido x=36
Se ha enviado vector b: [0, 0, 1]
Se ha recibido y = 2
Calculando x de verificación = y^2 * v_i^b_i mod n
Test 1 superado.

Test número 2
Se ha recibido x=400
Se ha enviado vector b: [0, 1, 1]
Se ha recibido y = 220
Calculando x de verificación = y^2 * v_i^b_i mod n
Test 2 superado.
Identidad autenticada
```

Figure 6. FakeProver test—FFS.

The general parameters values were $p = 17, q = 31, t = 3, k = 3$. The public key used was $v = [38\ 256\ 9]$. The impostor guessed the Verifier's challenge in the second and third rounds, but not so in the first one. The challenge vector was $b = [0\ 0\ 1]$. The FakeProver's guess was $b_{fake} = [0\ 1\ 1]$. Using the other values received, $x = 93, y = 310$, the Verifier's calculations are valid both for the true vector b and the guessed b_{fake}:

$$x = y^2 * \prod_1^3 v_i^{bi}\ mod\ 527 = 310^2 * 9\ mod\ 527 = 186 * 9\ mod\ 527 = 93$$

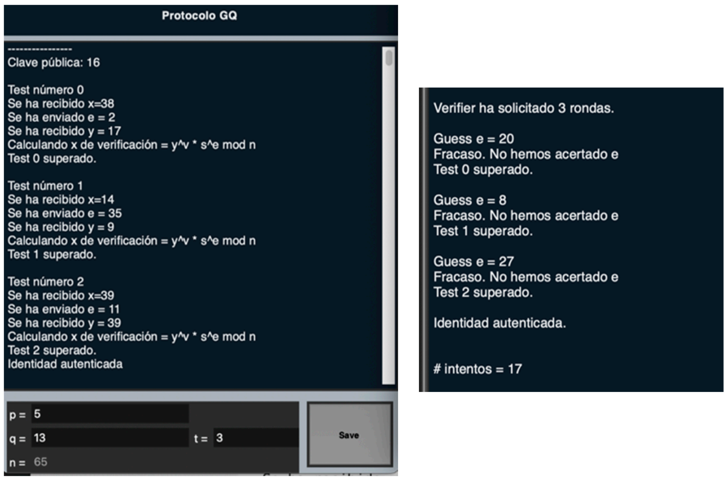
$$x' = y^2 * \prod_1^3 v_i^{b_{fake}i}\ mod\ 527 = 310^2 * 256 * 9\ mod\ 527 = 186 * 256 * 9\ mod\ 527 = 93$$

As $x = x'$, the impostor's authentication succeeds.

Another instance where the FakeProver's probability of succeeding increases is that where the private key has a repeated element. If the private key was $v = [315\ 497\ 497]$ and the Verifier's challenge was $b = [0\ 0\ 1]$, the FakeProver would pass the test without any trouble with the vector $b_{fake} = [0\ 1\ 0]$.

Finally, one last successful deception detected in the tests is that of the public key containing a one. Let the parameters be $p = 5, q = 3$, the private key $s = [11\ 4\ 14]$ has a matching public key equal to $v = [1\ 1\ 1]$, since $11^2\ mod\ 15 = 4^2\ mod\ 15 = 14^2\ mod\ 15 = 1$, $s_i^{-2} = 1 \forall i$. This fact invalidates the purpose of the challenge vector b because the equality $x = x'$ will hold regardless of the value of y chosen by the FakeProver.

In the case of the Guillou-Quisquater protocol, something similar happens. For values $n = 65$ $(p = 5, q = 13)$ and $t = 3$, the FakeVerifier is capable of authentication without guessing the challenge e. Figure 7 shows an example where the public key is $s = 16$ and $v = 47$.



```
Protocolo GQ

---------------
Clave pública: 16

Test número 0
Se ha recibido x=38
Se ha enviado e = 2
Se ha recibido y = 17
Calculando x de verificación = y^v * s^e mod n
Test 0 superado.

Test número 1
Se ha recibido x=14
Se ha enviado e = 35
Se ha recibido y = 9
Calculando x de verificación = y^v * s^e mod n
Test 1 superado.

Test número 2
Se ha recibido x=39
Se ha enviado e = 11
Se ha recibido y = 39
Calculando x de verificación = y^v * s^e mod n
Test 2 superado.
Identidad autenticada

p = 5
q = 13          t = 3
n = 65
```

```
Verifier ha solicitado 3 rondas.

Guess e = 20
Fracaso. No hemos acertado e
Test 0 superado.

Guess e = 8
Fracaso. No hemos acertado e
Test 1 superado.

Guess e = 27
Fracaso. No hemos acertado e
Test 2 superado.

Identidad autenticada.


# intentos = 17
```

Figure 7. FakeProver test—GQ.

In the first two rounds, the Verifier sends challenges $e = 2$. The FakeProver's prediction is $e_{fake} = 20$. After defining $y = 17$ randomly, the FakeProver computes $x = y^v \cdot J^{e_{fake}} = 17^{47} \cdot 16^{20}$ mod 65 = 38. Thus, the Verifier receives $x = 38$ and checks that $x' = y^v \cdot J^e = 17^{47} \cdot 16^2 = 38$. As $x = x'$, this round of the protocol is passed successfully. This error occurs due to the fact that $16^2 \equiv 16^{20} \; mod \; 65 = 61$; these flukes will be less likely the bigger the value of the modulus n.

Table 9 shows the average number of times needed for the different parameter settings for each of the protocols. As expected, the number of tries necessary to break their security increases as the number of trials requested by the verifier increases. However, it is important to note that the values assigned for the parameters in this experiment are much smaller than the ones that would be used in real life, as the computing power available to the user is not enough to support such numbers. Therefore, while these results provide a useful first practical experience with ZKP, they should not be taken as indicative of the performance of these protocols in real-world scenarios. Nonetheless, this experiment serves as a good starting point for understanding the strengths and weaknesses of these protocols and their potential applications in various fields.

Table 9. Performance measurement tables. (**a**) FFS, (**b**) GQ, (**c**) Schnorr.

(a)

t	Number of Tries (*p* = 3, q = 5)	Number of Tries (*p* = 17, q = 31)	Number of Tries (*p* = 233, q = 173)
1	4	7	59
2	5	39	469
3	10	230	633
4	17	1355	3365
5	69	7989	15,840

(b)

t	Number of Tries (*p* = 3, q = 5)	Number of Tries (*p* = 17, q = 31)	Number of Tries (*p* = 233, q = 173)
1	3	2	4
2	3	6	17
3	10	14	247
4	16	223	4449
5	160	3564	71,209

(c)

t	Number of Tries (*p* = 11, q = 5, a = 3)	Number of Tries (*p* = 29, q = 7, a = 11)	Number of Tries (*p* = 373, q = 31, a = 8)
1	3	48	137
2	3	282	314
3	10	399	693
4	16	4	5548
5	160	5	34,560

6.5. Man-in-the-Middle Attack

Although this type of attack has not been implemented in the application, it is interesting to study how a Man-in-the-Middle attack would affect the ZKP algorithms studied.

A diagram showcasing the flow of this attack is presented in Figure 8. If a third person, let us call her Eve, were to gain access to the communication channel between the Prover who is trying to authenticate their identity and the Verifier who is providing the challenges, she would be able to impersonate the Prover and authenticate as such.

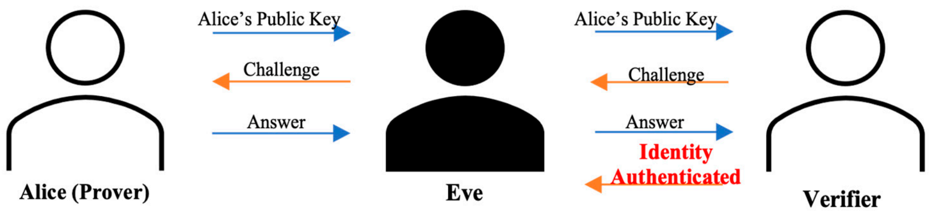


Figure 8. MITM Attack.

To perform this, Eve would simply have to intercept the messages exchanged in the communication, posing as the Verifier in front of Alice and repeating the challenges proposed, while she uses Alice's answers to authenticate herself in front of the Verifier.

This vulnerability exists for all three ZKP algorithms—Feige-Fiat-Shamir, Guillou-Quisquater, and Schnorr. ZKP tests are not designed to protect against MITM attacks, only to protect against information leakage. To prevent them, the following measures can be established:

- Strong synchronization: Decrease the waiting time allowed for the responses of the Prover so that there is no time for a retransmission of the messages.
- Communication security: Like any protocol vulnerable to this attack, the first line of defense would consist of not allowing a third party to intercept the communication.

7. Discussion

The experiments carried out on the application have proven quite illustrative of the characteristics and operations of the three implemented ZKP algorithms. Focusing on the role of the global parameters, which in a real case would be defined by a trusted third party outside the verifier and prover, it has been possible to study the effectiveness of brute-force attacks. Thus, some aspects of the implementation of the protocols that would be naive for a real application case have been identified. However, they do not interfere with the academic objective of this work and may even prove instructive:

- Allowing an unlimited number of connection attempts: The security of the authentication system is exposed in brute-force attacks because it allows the malicious user to make as many connection attempts as necessary to bypass the protocol. A real Verifier can protect itself, or at least make the attacker's job significantly more difficult, by refusing connections from an address for a set amount of time, similar to IPS protection against SYN Flood attacks on TCP servers. One may also block authentication for the specific user whose public key is being used after a certain number of attempts.
- Using small numbers as global parameters: The parameters with which the protocols work to compute the calculations must be large enough so that random matches do not occur in the calculations within modular arithmetic, since this increases the probability of a successful attack.
- Insecure value of the key: In the case of the Feige-Fiat-Shamir protocol, preventing repetition among the elements of the vector that constitute the public key is essential. In any of the protocols, a public key with value one would be equally insecure. The key generator should be implemented to assign key values inside a high and wide enough range.

Other types of attacks such as MITM have been theoretically analyzed. ZKP tests are not designed to protect against MITM attacks, only to protect against information leakage.

The number of input sources is small enough so that the implementation of the application is not overly complicated, even when adding security checks. The interaction with the user is fluid and allows the user to be able to carry out their own analysis of the execution of the protocols since all of the necessary information is shown.

The main objective of the application is to allow the interested student to carry out first-hand experiments in order to achieve a greater understanding of the ZKP algorithms in a pleasant and practical way. In this sense, the expectations for this work have been met.

8. Conclusions

Studying the operation of the Feige-Fiat-Shamir, Guillou-Quisquater, and Schnorr protocols, the three ZKP protocols have been successfully implemented with the necessary checks to ensure that the calculations carried out in each case are the ones indicated and that the code is as simple as possible. The primary focus of this paper was to accomplish the following goals:

- Creating an application that manages the Verifier and Tester roles: The developed application has two main interfaces, one server and one client, which act as the Verifier and Prover, respectively. The Prover connects to the port where the Verifier is waiting for a connection, starting at the moment the ZKP authentication protocol is chosen.
- Testing the security of the protocols against brute-force attacks: The Prover interface includes a brute-force authentication option, where it will try to authenticate not with the private key, but under the assumption that it is unknown by trial and error.
- Allowing the application to be executable: The application has been developed in Python code, which allows it to be executed on different operating systems.

The tests carried out on the application have not only allowed us to verify its correctness but have also shown that it can be a useful tool for studying the chosen ZKP algorithms; it allows the user to follow its execution step-by-step and to analyze the factors that make them vulnerable to brute-force attacks in a practical way.

In conclusion, our paper has introduced the concept of Zero-Knowledge Proof (ZKP) protocols and their use in authentication in public-private key encryption systems. ZKP protocols ensure that sensitive information such as passwords and private keys are not vulnerable to interception or attack. This is especially relevant in applications such as online banking, e-commerce, and secure communication where the security of the underlying encryption system is paramount. As such, the study and implementation of ZKP protocols have become increasingly important in ensuring the privacy and security of online transactions. We have implemented and evaluated three ZKP protocols—Feige-Fiat-Shamir, Guillou-Quisquater, and Schnorr—using an application that checks their security against brute-force attacks. By providing a clear explanation of the math behind these protocols and offering practical examples of their application, we hope to make the study and implementation of ZKP protocols more accessible to a wider audience, contributing to the development of more secure online transactions.

9. Future Work

This work was motivated by the problem of Online Identity Management today. The concept of Self-Sovereign Identity has been studied: its origin as an alternative to the Centralized Identity system and its evolution thanks to the development of new technologies and mathematical tools. It is in this context that ZKP methodologies find a natural and extremely useful application.

The implemented ZKP protocols were chosen based on their simplicity and clarity, optimal for a first approach to this concept for people interested in studying the subject:

- Implementation of more complex ZKP protocols, such as those based on elliptic curve cryptography [14], which would allow extending the use of the application to more advanced levels of teaching and study.

- Improvement of the implementation of the defined protocols to improve their efficiency and thus be able to work assigning parameter values closer to real-life scenarios.
- Studying other types of attacks: the implemented brute-force attacks show the security of these protocols against the computational capacity of current machines. It would be interesting to study attacks that expose other types of vulnerabilities, such as the proposed Man-in-the-Middle attacks.

Likewise, the transition of the application to a blockchain environment would allow the practical connection of the ZKP algorithms with the theory developed at the beginning, bringing the developed tool closer to a more realistic simulation of a real and practical application of these protocols.

In summary, future work on the ZKP application includes implementing more complex protocols, improving efficiency, studying different types of attacks, and transitioning the application to a blockchain environment. These enhancements aim to make the tool more practical, comprehensive, and relevant to real-world scenarios.

Author Contributions: Methodology, J.R.B.H., J.B.H. and J.A.S.M.; Validation, J.R.B.H., J.B.H. and J.A.S.M.; Formal analysis, J.R.B.H., J.B.H. and J.A.S.M.; Investigation, C.V.M.; Writing–original draft, C.V.M.; Writing–review & editing, C.V.M., J.R.B.H., J.B.H. and J.A.S.M. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: No available data.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Bernard, T.S.; Hsu, T.; Periroth, N.; Lieber, R. Equifax Says Cyberattack May Have Affected 143 Million in the U.S. *The New York Times*, 7 September 2017. Available online: https://www.nytimes.com/2017/09/07/business/equifax-cyberattack.html (accessed on 15 April 2023).
2. Rosenberg, M.; Confessore, N.; Cadwalladr, C. How Trump Consultants Exploited the Facebook Data of Millions. *The New York Times*, 17 March 2018. Available online: https://www.nytimes.com/2018/03/17/us/politics/cambridge-analytica-trump-campaign.html (accessed on 15 April 2023).
3. AEPD. Cifrado y Privacidad IV: Pruebas de Conocimiento Cero. 2020. Available online: https://www.aepd.es/es/prensa-y-comunicacion/blog/cifrado-privacidad-iv-pruebas-conocimiento-cero (accessed on 2 March 2023).
4. Zcash. What Are zk-SNARKs? 2022. Available online: https://z.cash/technology/zksnarks/ (accessed on 2 March 2023).
5. Preukschat, A.; Reed, D. *Self-Sovereign Identity Decentralized Digital Identity and Verifiable Credentials*; Manning Publications Co.: Shelter Island, NY, USA, 2021; ISBN-13: 978-1617296598; ISBN-10: 1617296597.
6. Goldwasser, S.; Micali, S.; Rackoff, C. The knowledge complexity of interactive proof systems. *SIAM J. Comput.* **1989**, *18*, 186–208. [CrossRef]
7. Trappe, W.; Washington, L.C. *Introduction to Cryptography with Coding Theory*, 3rd ed.; Pearson: London, UK, 2020; ISBN-13: 9780135260166.
8. Guillou, L.C.; Quisquater, J.J. A Practical Zero-Knowledge Protocol Fitted to Security Microprocessor Minimizing Both Transmission and Memory. In *Advances in Cryptology, Proceedings of the EUROCRYPT 1988 (EUROCRYPT '88), Davos, Switzerland, 25–27 May 1988*; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 1988; Volume 330. [CrossRef]
9. Schneier, B. *Applied Cryptography: Protocols, Algorithms and Source Code in C*, 2nd ed.; John Wiley & Sons: Hoboken, NJ, USA, 2015; ISBN 978-1-119-09672-6.
10. Fedrecheski, G.; Rabaey, J.; Costa, L.; Calcina Ccori, P.; Pereira, W.; Zuffo, M. Self-Sovereign Identity for IoT environments: A perspective. In Proceedings of the 2020 Global Internet of Things Summit (GIoTS), Dublin, Ireland, 3 June 2020; pp. 1–6.
11. Rabaey, J.M. The swarm at the edge of the cloud—A new perspective on wireless. In Proceedings of the 2011 Symposium on VLSI Circuits—Digest of Technical Papers, Kyoto, Japan, 15–17 June 2011; pp. 6–8.
12. Costa, L.C.P.; Rabaey, J.; Wolisz, A.; Rosan, M.; Zuffo, M.K. Swarm os control plane: An architecture proposal for heterogeneous and organic networks. *IEEE Trans. Consum. Electron.* **2015**, *61*, 454–462. [CrossRef]
13. Archer, A. NoKnow Zero-Knowledge Proof Implementation in Pure Python. 2019. Available online: https://github.com/GoodiesHQ/noknow-python (accessed on 2 March 2023).

14. Chatzigiannakis, I.; Pyrgelis, A.; Spirakis, P.G.; Stamatiou, Y.C. Elliptic Curve Based Zero Knowledge Proofs and their Applicability on Resource Constrained Devices. In Proceedings of the 2011 IEEE Eighth International Conference on Mobile Ad-Hoc and Sensor Systems, Valencia, Spain, 17–22 October 2011; pp. 715–720. [CrossRef]
15. Yadav, V.K.; Yadav, R.K.; Chaurasia, B.K.; Verma, S.; Venkatesan, S. MITM Attack on Modification of Diffie-Hellman Key Exchange Algorithm. In Proceedings of the International Conference on Communication, Networks and Computing, Gwalior, India, 29–31 December 2020; Springer: Singapore, 2020; pp. 144–155. [CrossRef]
16. Kayathri Devi, D.; Akilan, S.S. Comparison of ZKP based Authentication Mechanisms for securing the web server. *Int. J. Eng. Technol. IJET* **2018**, *10*, 1243–1247. [CrossRef]
17. Raffo, D. Digital Certificates and the Feige-Fiat-Shamir Zero-Knowledge Protocol. Master's Thesis, Université de Marne la Vallée, Marne la Vallée, France, 2002. Available online: https://dr0.ch/papers/dc-and-ffs.pdf (accessed on 2 March 2023).
18. García, E.H.J.V. *Gestión Soberana de Identidades Descentralizadas con Blockchain*; Universidad Oberta de Cataluña: Barcelona, Spain, 2019.
19. Alamillo, I. *El Uso de los Sistemas de Identidad Auto-Soberana en el Sector Público Español y de la Unión Europea*; Blockchain Intelligence: Vancouver, BC, Canada, 2019; Available online: https://blockchainintelligence.es/wp-content/uploads/2019/03/Art%C3%ADculo_El-uso-de-los-sistemas-de-identidad-auto-soberana-en-el-sector-p%C3%BAblico-espa%C3%B1ol-y-en-la-Uni%C3%B3n-Europea.pdf (accessed on 2 March 2023).
20. Ley 39/2015, de 1 de Octubre, del Procedimiento Administrativo Común de las Administraciones Públicas. Available online: https://www.boe.es/buscar/act.php?id=BOE-A-2015-10565 (accessed on 2 March 2023).
21. Directiva 1999/93/CE del Parlamento Europeo y del Consejo, de 13 de Diciembre de 1999, por la que se Establece un Marco Comunitario para la Firma Electrónica. Available online: https://www.boe.es/buscar/doc.php?id=DOUE-L-2000-80059 (accessed on 2 March 2023).
22. Real Decreto 311/2022, de 3 de Mayo, por el que se Regula el Esquema Nacional de Seguridad. Available online: https://www.boe.es/diario_boe/txt.php?id=BOE-A-2022-7191 (accessed on 2 March 2023).
23. Bartolomeu, P.C.; Vieira, E.; Hosseini, S.M.; Ferreira, J. Self-Sovereign Identity: Use-cases, Technologies, and Challenges for Industrial IoT. In Proceedings of the 2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), Zaragoza, Spain, 10–13 September 2019; pp. 1173–1180. [CrossRef]
24. Weng, C.X.; Gao, R.Q.; Bao, Y.; Liu, W.B.; Xie, Y.M.; Yin, H.-L.; Lu, Y.S.; Chen, Z.B. Beating the fault-tolerance bound and security loopholes for Byzantine agreement with a quantum solution. *arXiv* **2022**, arXiv:2206.09159.
25. Yin, H.-L.; Fu, Y.; Li, C.-L.; Weng, C.-X.; Li, B.-H.; Gu, J.; Lu, Y.-S.; Huang, S.; Chen, Z.-B. Experimental Quantum Secure Network with Digital Signatures and Encryption. *Natl. Sci. Rev.* **2022**, preprint. [CrossRef]