Honors College Thesis

Submitted in partial fulfillment of the requirements for graduation from the Honors College at Adelphi University

"Zero Knowledge Proofs"

Arfan Rasheed

Kees Leune

Sean Bentley

Sixia Chen

May 2024

Abstract

In this paper, I introduced Zero Knowledge Proofs, or ZKPs, an emerging topic in cryptography. ZKPs can change the way authentication systems are used by removing server sided hashed passwords and usernames allowing increased privacy to users. My research was conducted through an incremental quasi-structured literature review. I introduced some examples of ZKPs found in literature. In this paper, I sought to answer the question of — what kind of challenges do zero knowledge proofs provide a better solution than traditional approaches in the context of web and mobile applications? Through this question, I created an experiment in which I tested and altered Slawomir's (Grzonkowski et al.) authentication method. Based on my findings, I then reviewed the current state of Zero Knowledge Proofs. I found that ZKPs, as of now, are not suitable for replacing traditional methods of user authentication as they still rely on shared secrets.

Contents

1 Introduction

Cryptography is a part of cyber security, in which it attempts to provide a defense for message interception. Some key principles are privacy, authentication, signatures, minimality, simultaneous exchange, and coordination (RIVEST, 1990). It is important that while a message is secure and gets to another person, we can ensure the message came from the intended source, information was not leaked, has only the intended information, and is secure even if leaked. This presents some challenges that make modern-day cryptography important to society, as secure communication and privacy are vital. Cryptography tends to rely on the concept of *shared secrets*, in which some key information is exchanged between two parties that allows for authentication later. For example, Alice and Bob (common names in cryptographic exchanges, can also be denoted as A and B) will have a secret password they can use to ensure the identity of one another. Zero Knowledge Proofs intend to circumvent this notion of shared secrets (RIVEST, 1990).

1.1 Research Question

Zero knowledge proofs are an emerging cryptography topic that can be used for message authentication applications and information privacy. A zero knowledge proof is applied through a confirmation system in which a prover can prove knowledge of some information to a verifier without telling the verifier that exact information. This allows message authentication where a party can show they have a message without revealing it to a corresponding one. From that I derived the research question: in the context of web and mobile applications, what kind of challenges do zero knowledge proofs provide a better solution than traditional approaches?

1.2 Methodology

My methodology will use an incremental quasi-structured literature review, in which I will develop sub-questions about zero knowledge proofs from my initial question. After this review, I will conduct a gap analysis in the current literature to formulate a hypothesis that can be tested. Following the hypothesis, I will develop and execute an experiment to test and confirm something previously proven in the literature review or test a new idea based on a gap in existing literature. After conducting my experiment, I will analyze my findings and conclude the experiment answering my initial question and other sub-questions created during my research process.

1.3 Structure of this Thesis

This thesis will follow a format starting with background on ZKPs. Then, I will define what a ZKP is in-depth and some important clauses they must adhere to. Following this will be the first implementations of ZKPs and some examples of methods in which ZKPs are used. Then, there will be a brief introduction of Non-interactive ZKPs which showcase some applications of ZKPs in today's world. After that, an overview of web authentication and my experiment regarding Slawomir, followed by the analysis of my findings, conclusion, and thoughts on future work.

2 Background

2.1 Cryptography

As denoted in Section 1, cryptography deals with secure communication between two parties with the knowledge of a possible third party working against them. An important part of cryptography is the authentication of a party. A party should be able to prove its identity to another party since information from an unauthenticated party cannot be trusted. This extends into the web, as generally before entering most websites, a user can create an account. This account contains a username and password that can be used to authenticate a user later, allowing them to access information that only their account would have.

Dwivedi et al. (2021) defines a series of cryptographic attacks that have to be considered when creating a cryptographic algorithm. Direct attacks are attacks that have physical access or proximity to a device. Active attacks are attacks that require interaction in an attempt to impersonate a user. Denial-of-service attacks are attacks that make a system unavailable. Passive attacks are attacks that rely on observing or tapping, which allows an attacker to gain information on a user. Brute force attacks can be split into three different types, targeted attacks, trawling attacks, and blind attacks. Targeted attacks are dictionary-based attacks for guessing passwords, where an attacker will enter a large amount of passwords in an attempt to gain access. Trawling attacks are attacks that take a specific password and guess usernames instead of passwords. Blind attacks are where a username and password are both guessed.

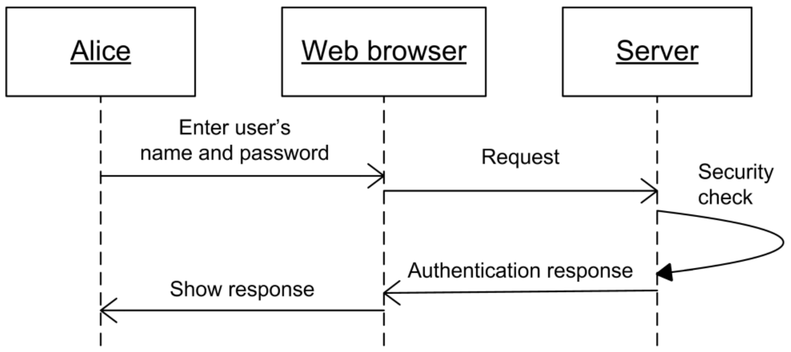


Figure 1: Classical Web Authentication (Grzonkowski et al., 2008)

2.2 Message Authentication

Web authentication can be achieved through many methods such as HTTP (Hypertext Transfer Protocol), HTTP digest, and most commonly HTTPS. HTTP gives servers credentials in readable form but not as plaintext. HTTP digest allows the server to impersonate the user but is not a serious problem for noncommercial businesses (Grzonkowski et al., 2008). HTTPS is in widespread use today and is more secure as it uses asymmetric cryptography. This means that public and private key is used for security. Asymmetric cryptography with digital certificates allows for a safer protocol overall (Grzonkowski et al., 2008).

The classical way to authenticate a user, indicated in Figure 1, is by entering a pre-established username and password and the browser authenticating it. Sometimes the verification step will be unsuccessful yet the browser will let the user have access to the site which can be an issue.

Some existing examples are presented by a simple challenge-response as implemented by Yahoo!, but this is ineffective, as it uses an insecure algorithm, MD5. Password-authenticated key exchange, or PAKE, uses short passwords memorized by the user (Grzonkowski et al., 2008). PAKE is safe against man-in-the-middle attacks and eavesdropping as it uses a shared password (Dwivedi et al., 2021). A subclass called encrypted key exchange, or EKE, combines asymmetric and symmetric encryption (Grzonkowski et al., 2008). Simplified Password-authenticated Exponential Key Exchange, or SPEKE, is used for commercial purposes. Secure Remote Password, or SRP, is another solution, however, it is vulnerable to dictionary attacks (Grzonkowski et al., 2008).

2.3 Zero Knowledge Proofs

Zero Knowledge proofs at their core are meant to produce a scenario in which no (malicious) verifier can get any extra information from the proof procedure, except the correctness of the statement. Zero knowledge proofs must satisfy two conditions, completeness and soundness (Wu, 2014). Completeness equates to, if a statement is correct, a verifier will always accept it, or in a proof system for a set S of possible actions for a user "for every $x \in S$, the verifier always accepts after interacting with the prover on common input x" (Mohr, 2007). Soundness, on the contrary, is that if a statement is incorrect, a verifier will always reject it, or "for some polynomial p, it holds that for every $x \notin S$ and every potential strategy P^*, the verifier rejects with probability at least $\frac{1}{p(|x|)}$ after interacting with $P*$ on common input x", where the strategy is a probabilistic polynomial-time strategy (Mohr, 2007).

A ZKP is complete when an honest prover always convinces an honest verifier. A ZKP is sound when a cheating prover can convince an honest verifier a proof is true with a very small probability. Another definition is "A strategy A is zero-knowledge on (inputs from) the set S if, for every feasible strategy B^* there exists a feasible computation C^* so that the following two probability ensembles

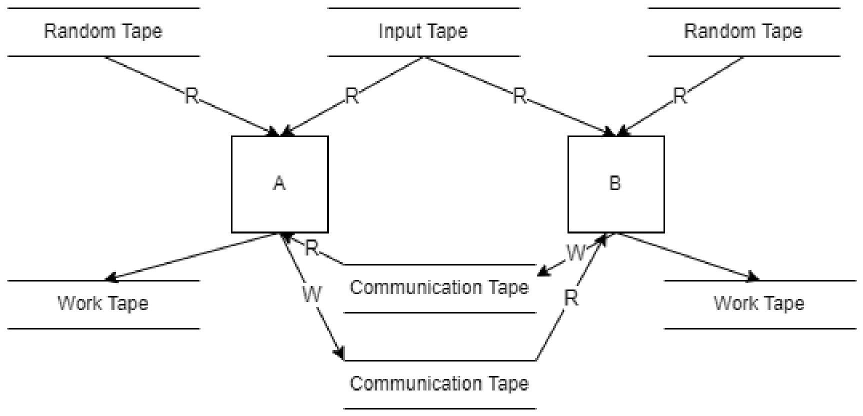


Figure 2: Turing Machines GMR (Goldwasser et al., 1985)

are computationally indistinguishable: the output of B^* after interacting with A on common input $x \in S$ [and] the output of C^* on input $x \in S$" where the computation C^* is derived from the strategy B^*(Mohr, 2007). The strategy in these problems deals with the prover's attempts to prove a statement, while the computation deals with the verifier's ability to confirm that statement to be true.

Dwivedi et al. also defines the basis of a ZKP protocol as a user makes a commitment where they choose a random value from a set that cannot be changed, a challenge where a query is sent to the user, and a verification where the user's response to the query is validated.

Zero knowledge proofs initially were defined in an article by Goldwasser et al. (1985) (GMR), which is referenced in many research papers on the topic, in which all participants are considered to have infinite computing power, and the object that they try to "know better" is not public input. For example, we can take a coin toss where not everyone knows which side the coin has landed, and those who do not know have to guess. Knowledge is a notion relative to a specific model of computation with specified computing resources and one

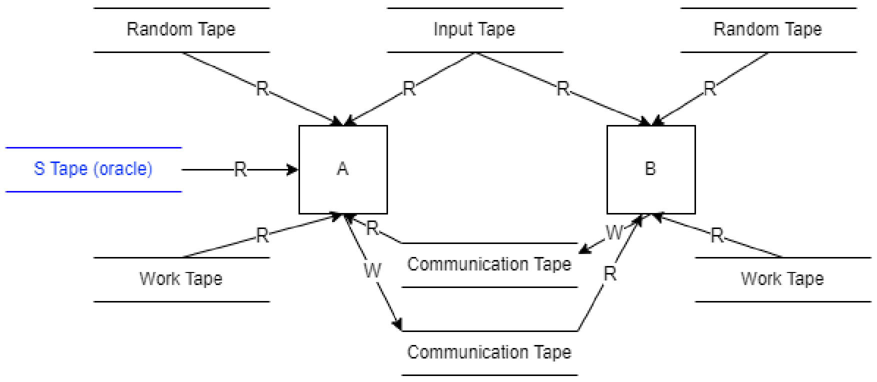


Figure 3: Turing Machines Feige, Fiat, and Shamir (Fiege et al., 1987)

studies and gains knowledge about available objects (Goldwasser et al., 1985). Their process uses two Turing machines taking turns computing using random tapes, work tapes, an input tape, and communication tapes as seen in Figure 2. The W refers to writing and the R refers to reading in the figure. The prover is exponential-time and the verifier is polynomial-time, in which A's read-only is B's write-only and vice versa. A is the prover and B is the verifier in Figure 2. These machines interact with one another through turns and send messages to each other determined by the random tape. As Turing machines are computational devices, A can prove to B knowledge of some information through a series of computations sent to one another. Essentially this shows the main differentiation between a ZKP and a traditional approach as ZKPs use repetition to prove an idea, while traditionally messages are sent once to ensure security. (Goldwasser et al., 1985)

The Feige, Fiat, and Shamir piece takes the original GMR take on ZKPs a bit further in which the prover and verifier are no longer infinitely powerful in terms of knowledge complexity. For their Turing machines, there are two communication tapes, two private work tapes, and two private random tapes, as seen in Figure 3. There is also a private oracle tape S, which is only read by

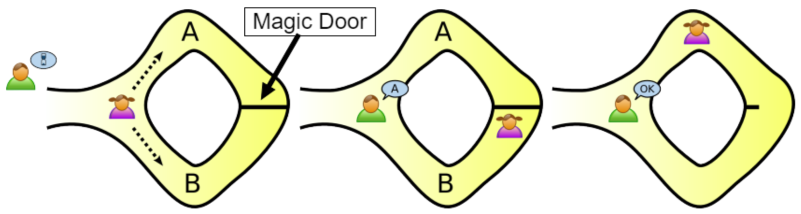


Figure 4: Peggy and Victor's cave (Mohr, 2007)

the prover. This tape is used when a clause is satisfied. For example, the clause could be a specific formula being satisfied or a valid 3-color graph coloring. In addition, there is more complexity (not shown in Figure 3) as there are crooked versions of A and B who are dishonest. Crooked A does not know the information but is trying to fake knowledge while Crooked B is trying to gain more knowledge about the information that is not necessary for the proof. Fiege et al. asserts that only a straight A can convince B to accept correct inputs while a crooked A cannot. This proof is later expanded to an identification scheme is a protocol that enables party A to prove his identity polynomially many times to party B without enabling B to misrepresent himself as A to someone else. This allows proof of identity without someone being able to steal identities which can be vital in increasing personal privacy (Fiege et al., 1987).

2.4 Examples

2.4.1 Peggy and Victor's Cave

As shown in Figure 4, the classic example of a Zero Knowledge Proofs is Peggy (commonly the prover in cryptographic examples) and Victor's (commonly the verifier in cryptographic examples) cave (Mohr, 2007). The cave has a magic door that opens upon entering a secret word. Peggy has discovered the word

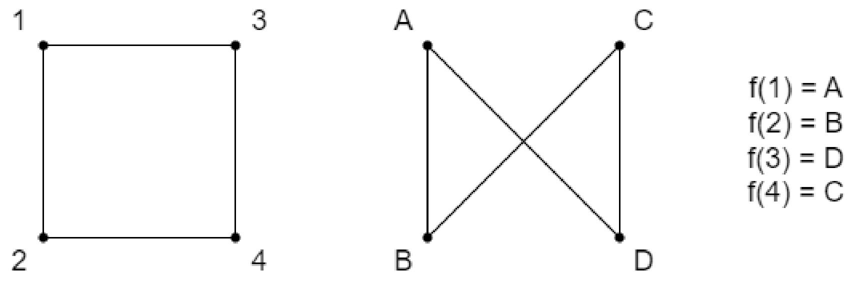


Figure 5: Graph Isomorphism (Mohr, 2007)

and Victor is willing to pay Peggy for the word but he has to be sure that Peggy does know the word and Peggy wants to show Victor she knows the word without revealing it. Peggy chooses a path to go down and Victor chooses her path of return, and with enough attempts, Victor can tell without doubt that Peggy knows the secret word, as if she chooses to go down path A and has to return path B, she must know the word. There are multiple trials to lower the probability she returns along the path that she had traveled. Victor's chances of overhearing her saying the secret word are also reduced by the fact that he does not know the path she has traveled. This example is the prime example of ZKPs when used to describe its basic functionality.

2.4.2 Graph Isomorphism

As shown in Figure 5 another example of ZKPs is through the use of graph isomorphism (Mohr, 2007). An isomorphic graph contains the same number of graph vertices mapped in the same way. The graph on the left has the same mapping of edges as the graph on the right. If we relabel the vertices as shown in the equations on the right of Figure 5, we will see that vertex 1 connects to vertex 2 and vertex 3 on both graphs. It is important to note that if G_1

is isomorphic to G_2, and G_1 is isomorphic to G_3, G_2 is also isomorphic to G_3. The prover will generate a graph H from two isomorphic graphs G_0 and G_1, where H is a random isomorphic copy of G_1. The verifier then sends the prover a random $\alpha \in_R \{0, 1\}$. If the α is 1, the prover sends π, the symmetric group of $|V|$ elements, to the verifier or π multiplied by the isomorphism. If the permutation is not an isomorphism between α and H, the verifier rejects it.

This example can be shown by trying to prove two graphs are isomorphic to one another. First, we assume that two graphs have an extremely large number of nodes to the point where it cannot be determined by eye that these graphs G_1 and G_2 are isomorphic and a prover is attempting to show a verifier that these graphs are isomorphic and that he knows their permutation without showing them how they are isomorphic. The prover can send a permutation of one of the graphs, or graph H, and save the permutation π. Then the verifier picks an integer, t, which can be a 1 or 2 with equal probability. The prover will send $\pi^{-1} : H \to G_1$ or $f \circ \pi^{-1} : H \to G_2$ yielding G_t. Since the permutation is isomorphic to the G_1, the verifier can then check if G_t is isomorphic to H, which will prove that the original graphs are isomorphic through multiple tests, as eventually a permutation of G_2 will be tested (Kun, 2016).

2.4.3 3-colored graph

Figure 6 is the graph three colorability problem (Mohr, 2007) which works for all languages in NP, nondeterministic polynomial time. NP is a set of decision problems that are verifiable by deterministic Turing machines and solvable by nondeterministic Turing machines. The way a three-colorability graph works is that no adjacent vertices of the graph are the same color. The verifier chooses an edge of a random 3-color graph with n locked boxes and sends it to the prover

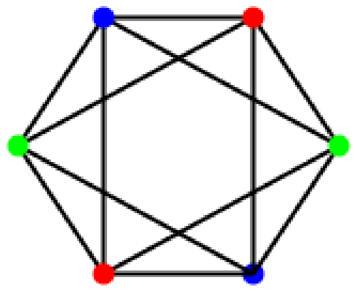
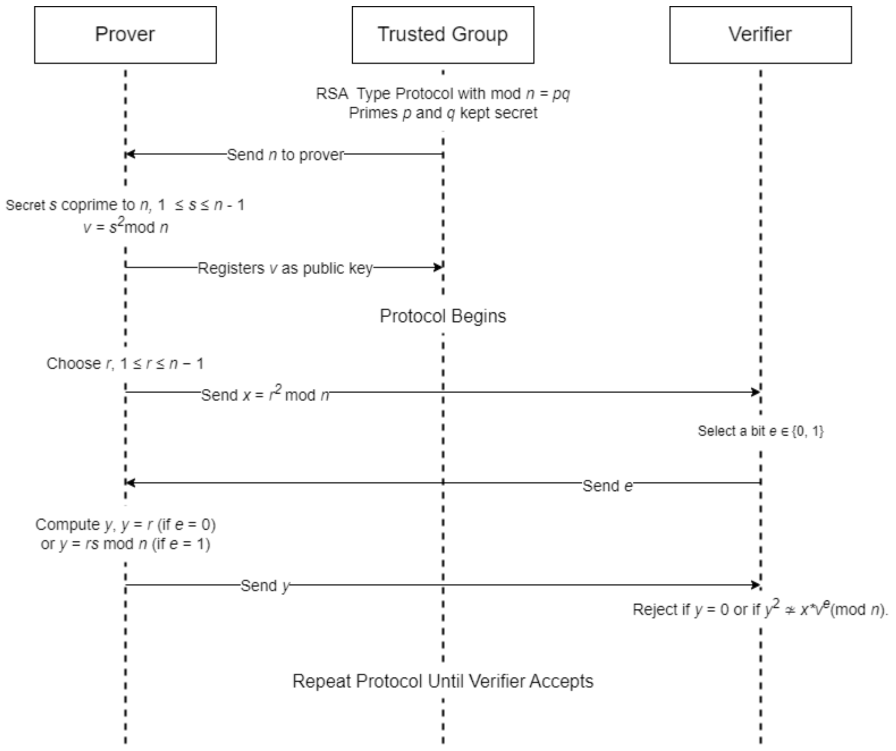


Figure 6: 3-colored graph (Mohr, 2007)

who sends back keys to the adjacent vertices. The keys must open adjacent vertices to the edge, and this process is repeated unless the keys do not match the boxes or contents violate the conditions which then the verifier will stop and reject the prover. This is done m^2 times with m being the number of edges in the graph.

A simpler way to put this protocol is simply through the use of a random color map. A prover will create a large 3-color graph for the verifier. They will then cover all vertices of this map and then give the verifier a choice of one edge to pick. Once an edge is selected, the two adjacent vertices will be revealed to show the verifier that they are two different colors. The prover will then recreate the 3-color graph with different colors but in the same orientation and cover the vertices. The verifier will once again challenge an edge to show that two vertices are two different colors. This process will repeat over and over until the odds that the color map is incorrect are negligible and the verifier is convinced. The verifier will never be guaranteed to know if the map was correct but will be confident in his decision and lack knowledge of the color map throughout the process, as the colors will be in different places throughout the process giving the verifier no ability to try to record and recreate the color graph (Green, 2017).



Prover
Trusted Group
Verifier
RSA Type Protocol with mod $n = pq$
Primes p and q kept secret
Send n to prover
Secret s coprime to n, $1 \leq s \leq n - 1$
$v = s^2 \bmod n$
Registers v as public key
Protocol Begins
Choose r, $1 \leq r \leq n - 1$
Send $x = r^2 \bmod n$
Select a bit $e \in \{0, 1\}$
Send e
Compute y, $y = r$ (if $e = 0$)
or $y = rs \bmod n$ (if $e = 1$)
Send y
Reject if $y = 0$ or if $y^2 \not\equiv x * v^e (\bmod n)$.
Repeat Protocol Until Verifier Accepts

Figure 7: Fiat-Shamir Protocol (Mohr, 2007)

2.4.4 Fiat-Shamir Identification Protocol

The Fiat-Shamir identification protocol (Mohr, 2007) is another example of zero knowledge proofs used for entity authentication as seen in Figure 7. Peggy can prove to Victor that she is Peggy and has a secret s without revealing the secret and proving she has it. A trusted middleman creates a similar scheme to an RSA protocol in which $n = pq$ and keeps the primes secret. An RSA protocol normally uses a public and private key in which a private key can decrypt messages encrypted by the public key. The prover will create a coprime to n, such that $1 \leq s \leq n - 1$ and create another number v, such that $v = s^2 \bmod n$ will be used by the middleman as her public key. The prover will then send a random number r, such that $1 \leq r \leq n-1$, and sends $x = r^2 \bmod (n)$ to the verifier. The verifier then sends a bit $e \in \{0, 1\}$ to the prover. The prover then can compute a y where $y = r$ when $e = 0$ and $y = rs \bmod n.$ when $e = 1$. The verifier will then reject if $y = 0$ or $y^2 \approx x * v^e \bmod n$ (Mohr, 2007).

2.5 Applications

Wu and Wang (2014), introduce Noninteractive Zero Knowledge (NIZK) which contains a message only from the prover to the verifier, forming essentially a one-way implementation of ZKPs. This allows for cryptographic advances in applications to Common Cryptographic Architecture (CCA, used for financial transactions), security encryption, anonymous authentication, and group and ring signatures.

Witness indistinguishability, a weaker version of ZKPs, is used for security in some applications but is not as strong as ZKPs in terms of confidentiality. Using a common reference string (CRS, the model generated by a trusted party), a prover and verifier can complete a ZKP, however, the prover and verifier will have to interact in the first place (Wu and Wang, 2014).

Dwivedi et al. introduces another authentication system based on ZKPs intended for the Internet of Things (network of devices and technology that connects devices to the internet), IoTs, and specifically used for healthcare but can be used in other fields. The registration protocol uses a UID (user ID, a chosen username or email address), a SID (server ID), or a QR code containing the UID and server domain name, and a URL given to the user to create a master password. x is created from a hash of the UID, SID, and master password. A public parameter $v = g^x$ is also created from the application, where g is an arbitrary variable created in the application that is unknown to the user. The user registers with v and their UID. To authenticate a user, they send in their UID which gets matched with the v parameter. This is then used in a computation with the master password as inputted to authenticate the user and then redirect them to the page. This is used for Zero Knowledge Nimble, ZKNimble, which is a block cipher of size 64. ZKNimble with permutation and substitution layers followed by a round function to produce keys. This protocol implements NIZKs to authenticate users (Dwivedi et al., 2021). Dwivedi et al. also introduces Schnorr's identification protocol, or SIP, which is secure against direct attacks and eavesdropping (Dwivedi et al., 2021).

Soewito and Marcellinus (2021) creates a similar authentication method to Dwivedi et al. (2021) that uses Advanced Encryption System (AES) instead of the ZKNimble procedure to create a privacy algorithm with ZKP as authentication and AES as data encryption. (Soewito and Marcellinus, 2021).

2.6 Background Summary

In this chapter, I introduced the basis of cryptography, message authentication, and ZKPs. I introduced three examples of ZKPs: Peggy and Victor's cave, graph isomorphism, 3-colored graphs, and the Fiat-Shamir Identification Protocol. I also introduced a key factor of ZKPs that differentiates them from traditional cryptography — the repetition of a process to increase the likelihood of it being true, rather than a single process used. In addition, I established some applications of ZKPs that are currently being used today. Next chapter, I explain my experiment dealing with Slawomir's algorithm in web authentication.

3 Experiment

In this section, I go through the algorithm devised by Slawomir. I also revised his algorithm to make it function properly, as in its current state it would not work in practice.

3.1 Analysis

Grzonkowski et al. (2008) proposes a new method of authentication using ZKPs, as shown in Figure 8, which uses a private key algorithm. The private key goes through a SHA1 hash algorithm and is altered through a permutation algorithm leaving it as some value of $a_k k!$ such that k is a natural number. For example, the number 159 can be written in this form as $(1)5! + (1)4! + (2)3! + (1)2! + (1)1!$. This can be done as a browser extension or a script, in which a script has to have server trust while a browser extension is browser dependent. There is also the precondition that the site is free of XSS, or cross-site scripting, vulnerabilities. The protocol is secure from graph isomorphism attacks as with a large square matrix brute force attacks are rendered less effective and the graphs are completely random. Dictionary attacks can be slowed down and detected using captchas as well (Grzonkowski et al., 2008).

Through the use of ZKPs, the browser can directly interact with a web server in which a user's password never has to leave the browser. The browser will use a user's public and private key pairs in combination with a user's password. The server and the browser will interact and use challenge graphs to determine if the user has inputted the correct password as shown in Figure 8. This, however, still leaves the username open, which makes this similar to a public key exchange

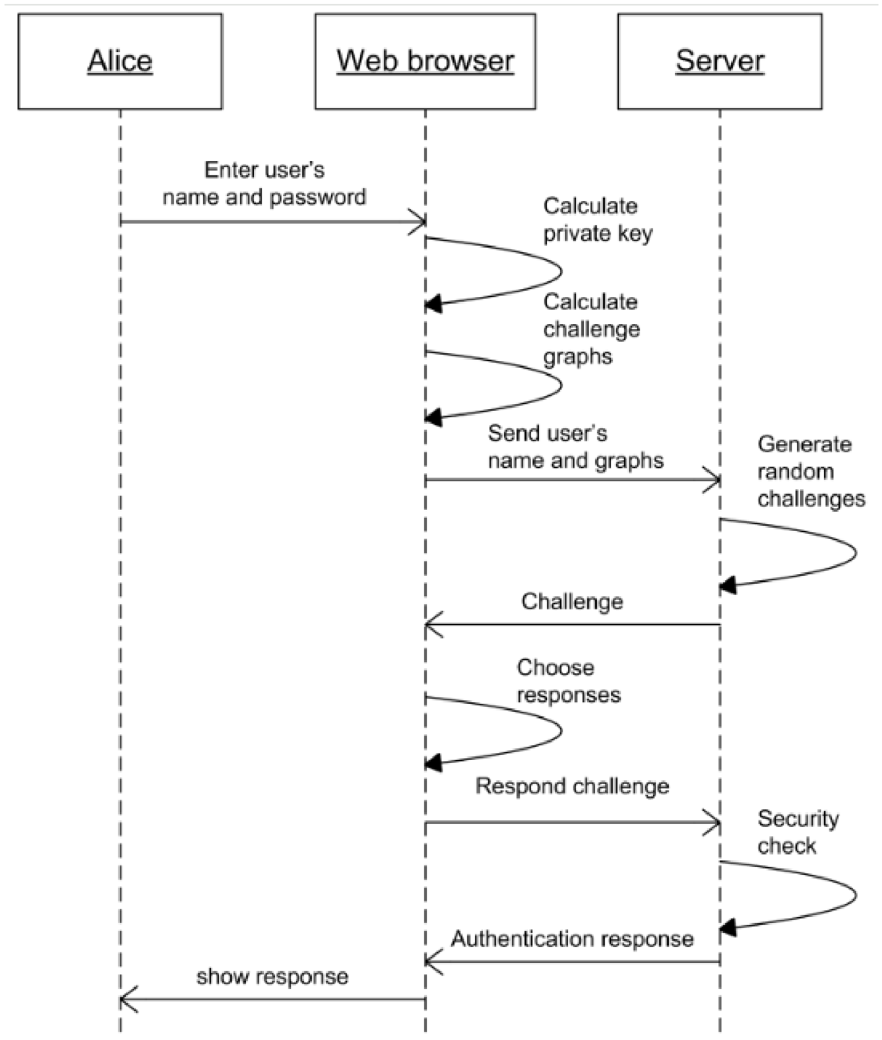


Figure 8: Slawomir's Authentication (Grzonkowski et al., 2008)

```
Convert(number)
var i := 0
var factor := 0
var a[] := newArray()
while number > 0 do
    factor := GreatestFactorial(number)
    a[i] := number/ factor
    number := number − a[i] ∗ factor
    i = i + 1
end while
return a
```

Figure 9: Slawomir's Convert 1

system (Grzonkowski et al., 2008).

The main algorithms and processes tested were that of Slawomir. Slawomir was attempting to use the graph isomorphism approach to prove a user's identity to the server with the browser creating graphs and the server challenging them. This method would work by the challenge graphs of the server proving that the graphs from the browser were indeed isomorphic and that the browser knew the permutation to create a second graph from the first, which is derived from the user's password therefore proving the user's identity.

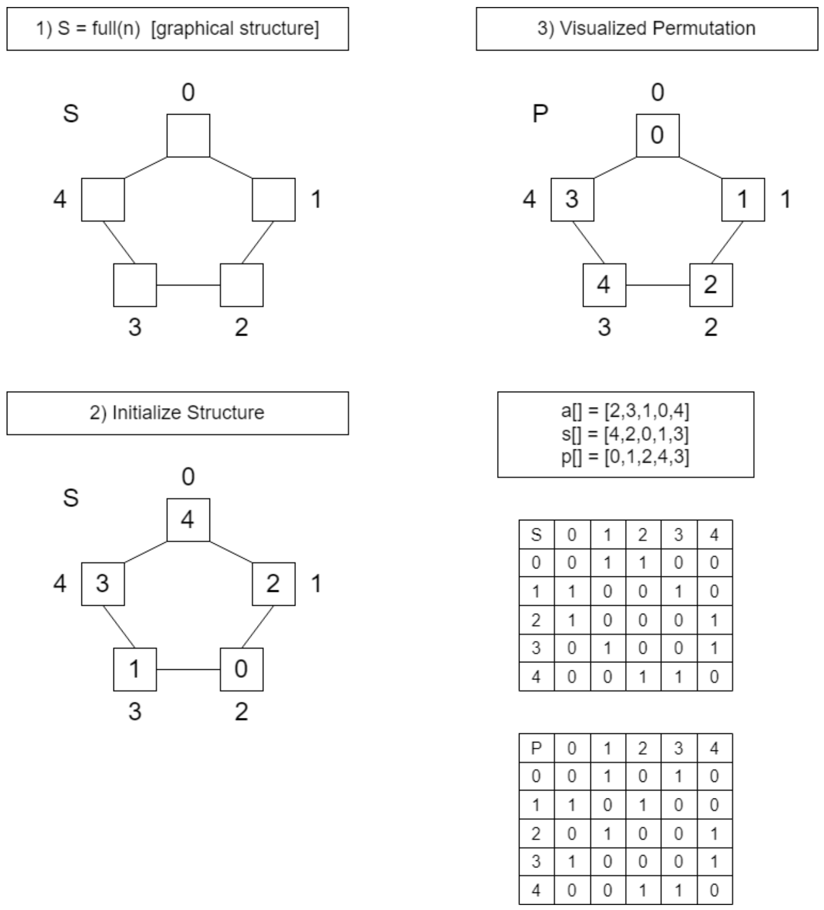
I attempted to follow Slawomir's algorithms and recreate this process as a secondary test. When reading through Slawomir's algorithm 1 (as seen in Figure 9), which used a converted secure hash from a user's password, I realized that this algorithm would not work. The first issue was with the `GreatestFactorial` method, which looked for the smallest factorial greater than a given number. From this method, I arrived at two distinct answers, that the number generated was the factorial, for instance, $3! = 6$ where the number would be 3 or the number would be 6. This is where the issue arises, as using the number 5 and running it through the algorithm, we have an issue if the method returns 3, as

```
2ndConvert(a[])
var n := lengtha
var s := full(n) {structure with integers 0,1,....(n−1)}
var a[] := newArray()
for i = 0 to n − 1 do
    s.insert(i) {Initializing our temporary structure}
end for
for i = 0 to k − 1 do
    permutation[i] := s.elementAt(a[i]) {getting
        an element at a certain position}
    s.remove(a[i]) {removing the taken element}
end for
return permutation
```

Figure 10: Slawomir's Convert 2

the number will end up as 0 by the second iteration. If we take if the method returns 6, we will end up with an infinite loop as the number does not lower and the integer division yields 0. Therefore the `GreatestFactorial` would not work as intended as it takes numbers larger than the number inputted rather than numbers smaller leading to an array that would not work. I attempted to correct this method `GreatestFactorial` to instead return the largest factorial less than a given number, which would then generate an array as I assume was intended by the algorithm.

Figure 10 was a much harder algorithm to follow, as `lengtha` would not work and should be denoted as `length(a)`. The structure `full(n)` was an ambiguous one which was interpreted as an empty graphical structure as shown in Figure 11. This structure was not defined well by Slawomir, as it says it is a structure with integers from 0 to $n-1$. `full(n)` was then filled with integers again ranging from 0 to $n-1$, which raised the confusion of whether the structure should be filled or not. After this, Slawomir then creates a new array, `a[]` which is fine other than the fact that it would wipe out the input of the array `a[]`,


1) S = full(n) [graphical structure]
S
0
1
2
3
4
2) Initialize Structure
S
0
4
1
2
2
0
3
1
4
3
3) Visualized Permutation
P
0
0
1
1
2
2
3
4
4
3

a[] = [2,3,1,0,4]
s[] = [4,2,0,1,3]
p[] = [0,1,2,4,3]

S	0	1	2	3	4
0	0	1	1	0	0
1	1	0	0	1	0
2	1	0	0	0	1
3	0	1	0	0	1
4	0	0	1	1	0

P	0	1	2	3	4
0	0	1	0	1	0
1	1	0	1	0	0
2	0	1	0	0	1
3	1	0	0	0	1
4	0	0	1	1	0

Figure 11: Slawomir's Scaffolding

```
def randomConvert(number: int):
    random.seed(number)
    a = [-1]*(41)
    counter = 0
    while(a.count(-1)>0):
        num = random.randint(0,41)
        if(a.count(num)<1):
            a[counter]=num
            counter+=1
    return a
```

Figure 12: Random Seed

later leading to the second for loop not working at all.

Figure 11 attempts to follow the process of the second algorithm by filling the structure and then creating a permutation structure based on the `a[]`. This is then shown in the form of an adjacency matrix. The first row and column represent the vertexes of the graph. The zeroes in the matrix represent the lack of an edge between the two vertices while the ones represent an edge. The glaring issue when trying to implement this as code is the indices. As the indices of the array get removed to prevent two of the same numbered points from existing in multiple positions, the array becomes too small leading to an index out-of-bounds error. For instance, if we take `a[]` from Figure 11, as we remove indices, the array becomes shorter and if we try to access an index that is no longer in the array, this will lead to an error.

3.2 Altered Algorithm

Through the process of using these two algorithms and attempting to implement them, it seemed that they were password-deterministic, which led to the idea of abandoning both of these algorithms and instead opting for a different route

```
def inversePermutation(a:dict):
    perm = dict()
    for key in a:
        perm[a[key]] = key
    return perm
```

Figure 13: Inverse Permutation

that used the password to generate a permutation array similarly to that of Slawomir, which was through the use of a random number generator. The password would be converted into a secure hash as before, however, this number would be entered into a seeded generator which would create a list of entries such that the entries were randomly assorted and unique, as denoted by Figure 12. This list would then be used as a permutation on a randomly generated graph to create an isomorphic graph which could be tested for isomorphism. Using the networkx library, these graphs can be tested for isomorphism, and permutations to these graphs can be applied to create new isomorphic graphs. This allows a simulation of a server creating random graphs and sending them back to the browser, which the browser can then send the inverse permutation back as noted in Figure 13. This is important, as an inverse permutation generated as noted in Section 2.4.2, can be used to prove the graphs are isomorphic and will not give a verifier the permutation to get from $G1$ to $G2$.

3.3 Experiment Summary

In this section, I explained Slawomir's algorithm and went through it step by step. I pointed out the critical errors that would prevent his algorithm from running and then altered it with a new method based on a random seed number generator. In the next section, I will conclude this paper and address future work.

4 Conclusion

Zero knowledge proofs fundamentally require zero knowledge and a glaring issue with the methodology used for the authentication of mobile and web applications is an initial secret. Unless a user can initially prove their identity, a server will have no identity to even check against in the first place resulting in a lack of access. A user must initially create their username password paired key to even start the process of authentication in the model described which leads to it being an overdone attempt at public key exchange. Although this can increase the complexity behind such a method, it fails at achieving the core concept of zero knowledge. Shared secrets remain prevalent in ZKPs, however, the idea of multiple trials to prove something is a new concept to cryptography. I concluded that in its current state, in addition to lacking documentation and articles that had a strong methodology, ZKPs are not a better method than classic web authentication.

4.1 Future Work

The random seed method, while applying Slawomir's algorithm in a method that could work, reached the same issue of initial identification. However, I thought of a new idea of generating the graph from the username of the user and using the password as the permutation. A user could be identified through their graph and permutation graph, while the permutation itself was still a secret. This process is not perfect, as it is unclear whether this method would be secure as the graphs would always stay the same. The same graphs create some possible issues from brute force methods to gain someone's username or password. Analysis of this method is something that can be looked into in the future to see if the trade-off of a less secure method in terms of graphs being the same would be worth the initial proof of identity to be implemented.

List of Figures

Bibliography

Dwivedi, A. D., Singh, R., Ghosh, U., Mukkamala, R. R., Tolba, A., and Said, O. (2021). Privacy preserving authentication system based on non-interactive zero knowledge proof suitable for internet of things. *Journal of Ambient Intelligence and Humanized Computing*, 13(10):4639–4649.

Fiege, U., Fiat, A., and Shamir, A. (1987). Zero knowledge proofs of identity. In *Proceedings of the nineteenth annual ACM conference on Theory of computing - STOC '87*, STOC '87. ACM Press.

Goldwasser, S., Micali, S., and Rackoff, C. (1985). The knowledge complexity of interactive proof-systems. In *Proceedings of the seventeenth annual ACM symposium on Theory of computing - STOC '85*, STOC '85. ACM Press.

Green, M. (2017). Zero knowledge proofs: An illustrated primer. *A Few Thoughts on Cryptographic Engineering.*

Grzonkowski, S., Zaremba, W., Zaremba, M., and McDaniel, B. (2008). Extending web applications with a lightweight zero knowledge proof authentication. In *Proceedings of the 5th international conference on Soft computing as transdisciplinary science and technology - CSTST '08*, CSTST '08. ACM Press.

Kun, J. (2016). Zero knowledge proofs - a primer. *Math $\cap$ Programming.*

Mohr, A. (2007). A survey of zero-knowledge proofs with applications to cryptography.

RIVEST, R. L. (1990). *Cryptography*, page 717–755. Elsevier.

Soewito, B. and Marcellinus, Y. (2021). Iot security system with modified zero knowledge proof algorithm for authentication. *Egyptian Informatics Journal.*

Wu, H. and Wang, F. (2014). A survey of noninteractive zero knowledge proof system and its applications. *The Scientific World Journal*, 2014:1–7.