Zero-Knowledge Authentication

Jakob Povšič

November 25, 2021

Abstract

In this thesis we focus on designing an authentication system to authenticate users over a network with a username and a password. The system uses the zero-knowledge proof (ZKP) system as a password verification mechanism. The ZKP protocol used is based on the quadratic residuosity problem. The authentication system is defined as a method in the extensible authentication protocol (EAP). Using a ZKP system yields interesting security properties that make the system favourable to be used over insecure networks.

Contents

Chapter 1

Introduction

Today privacy is a necessary sacrifice we have to make in order to take part in the digital world, imperative to our modern life. Every day, more digital systems gain access to our personal information. While this practice is often a necessary evil, many companies seek to exploit this position. Zero-knowledge proofs (ZKPs) have the potential to change how our data exists in the digital space. ZKP systems are an intriguing cryptographic phenomenon for proving mathematical statements without revealing *why* they are true.

Cryptocurrencies like Zcash [29] use ZKPs to confirm transactions while keeping the sender and the recipient anonymous and the transaction amount opaque. The self-sovereign identity space [48] uses ZKPs as an essential part in a decentralized and privacy-preserving digital identity infrastructure. ZKPs enable a system to verify the properties [12] of sensitive data without seeing it and risking misuse. With these tools, we can prove legal age, financial solvency, or our nationality, while revealing no sensitive information.

Advances like ZKPs hint of a future where we will look at our current personal data practice as feudal and undignified.

The focus of this thesis will be to define a simple use for ZKPs. We will design an authentication system using a ZKP as a password verification method, as an authentication method in the extensible authentication protocol (EAP). Moreover, in the system design we also have to consider standard methods for protecting against inherent vulnerabilities of password based systems.

1.1 Structure of the thesis.

This thesis is composed of three chapters. In §2 we explore the two key topics of authentication and ZKPs. In §3 we present the architecture of our authentication system and the extensible authentication protocol method definition. Chapter §4 concludes the thesis and gives some possible ways for future work.

Chapter 2

Methodologies and Tools

In this chapter, we will explore the concepts and components behind our authentication system. To design a password authentication system, we must first understand what an authentication system is, the mechanism of password based authentication, and how to avoid its vulnerabilities. We will also look into the EAP framework, into which we will build our authentication system. Our system will use a ZKP as a password verification mechanism, so we also examine what ZKPs are, how they work, and the mathematics behind why do they work.

2.1 Authentication

Authentication is the process of proving a claim or an assertion. Today, the term is most commonly used in information security [42], however, we can find the principles of authentication in fields of archeology, anthropology and others [35].

In computer science, we commonly use authentication for establishing access rights between protected system resources and users through digital identities. Government and international institutions have developed guidelines for managing digital identities and authentication processes [26].

While systems can authenticate both humans and other computer systems, we are focusing on authentication of a human end-user.

2.1.1 Authentication Process Components

Authentication [42] is verifying a claim that an entity or a resource has a certain attribute value. This is a broad definition, and it most frequently applies to the verification of a user's identity (e.g. at login), however we can make and verify claims about any subject or object. The process of authentication is done in two parts, *identification* and *verification*. A common application of authentication is to manage access of an external user to protected system resources.

Identification. Presenting an identifier to the authentication system that establishes the entity being authenticated. This is commonly a username or an email address. The identifier needs to be unique for the entity it identifies.

The process of identification is not necessarily externally visible, as the identity of the subject can be implicit in the environment. For example, we can determine an identifier from the user's IP address. Or a system might only have a single user that needs to be authenticated.

Verification. Presenting or generating authentication information that can verify the claim. Commonly used authentication information are passwords, one time tokens, digital signatures, etc.

Our system will verify the user's password using a ZKP.

2.1.2 Authentication Factors

Authentication systems can rely on three groups of factors [3].

Knowledge factors Something the user **knows** (e.g. password, security question, PIN)

Ownership factors Something the user **owns** (e.g. ID card, security tokens, mobile devices)

Inherence factors Something the user **is** or **does** (e.g. static biometrics - fingerprints, retina, face. dynamic biometrics - voice patterns, typing rhythm)

Strong authentication. As defined by governments and financial institutions [20, 27], strong authentication is a system using two or more factors. We also referred to this as *multi-factor authentication*. Our system will focus only on the user possessing a password (*knowledge factor*), while the relying party can use additional authentication factors to improve security.

2.2 Password Authentication

Passwords are one of the most common and oldest forms of user authentication, being first used in computers at MIT in the mid-60s [33].

Let us examine the high-level model of password authentication, its risks and tools to mitigate them.

2.2.1 Authentication Model

Password authentication is a simple model (Figure 2.1) based on a shared secret between the user and the system. The secret is usually a set of characters or words memorised by the user, inputted via a keyboard. We often used the password in a combination with a username. To authenticate, the user simply needs to prove to the system his knowledge of the password.

2.2.2 Security Vulnerabilities and Attacks

In a common password authentication system used on the web, the user sends a plain-text password over a secure HTTPS connection, the server verifies it and responds. The simplicity that makes passwords practical for users makes them vulnerable.

Authors [18] have shown that users pick passwords that are easier to remember and reuse the same passwords across different websites. Many websites also don't properly handle passwords, permitting attackers to access plain-text passwords when a security breach happens. These vulnerabilities can result in mild inconveniences to serious offences like identity theft. The industry is aiming to improve password security with the adoption of password managers and initiatives like FIDO [17] working to retire passwords altogether.

The National Institute of Standards and Technology [26] classifies attacks as *online* or *offline*, based on how the attacker is interacting with the system.

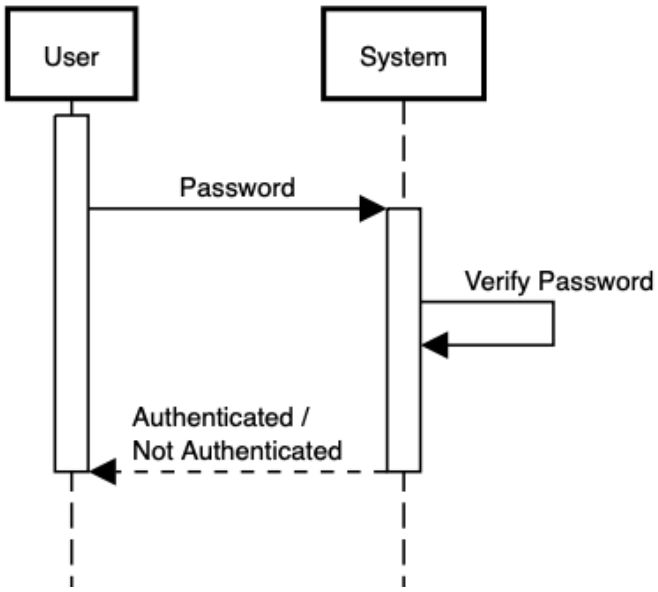

Figure 2.1: Password Authentication Model

Online Attacks

An attack where an attacker is directly interacting with the authentication system. These attacks are usually very *noisy*, making it easy for an authentication system to detect and react. For example, locking an account after a few failed authentication attempts. For this reason, brute-force online attacks are not effective.

Effective online attacks use the strategy of appearing as normal users, thus remaining under the radar of detection. Popular methods are *password spraying* and *credential stuffing* [28], both of which utilise information from data breaches, like lists of most commonly used passwords, or username and password combinations. Password spraying is taking a few commonly used passwords and attempting to authenticate with many accounts. The attacker is assuming that in a large sample of users some will use these weak passwords. Credential stuffing is taking a compromised user credential, for example, a username and password combination found in a data breach, and using it to authenticate into multiple websites. The attacker is assuming that if a person is using a set of credentials on one website, they are potentially reusing them on others.

Offline Attacks

Are attacks performed in a system controlled by the attacker. For example, an attacker might analyse data on his personal computer to extract sensitive information. The data is obtained by either theft of a file, eavesdropping on an authentication protocol or a system penetration.

Password cracking [5] is a method of extracting passwords from data used by the authentication system for password verification. Two parameters determine the chance of success when password cracking, the time required to check a single password and number of guesses required or the strength of the underlying password.

Security Practices

There are many practices an authentication system can incorporate to improve its security. The authentication system uses the persistently stored sensitive data for password verification. This data could be used by an attacker in an offline attack, so we need to protect them somehow. Often the password verification method imposes constraints on how we can transform the sensitive password verification data. Later, we will examine which constraints our ZKP system imposes. We are going to focus on methods for improving the security of the data layer against offline attacks. A production ready system should adopt many other security practices, however this is outside the scope of this thesis.

Key-Stretching Protecting password verification data in persistent storage is of critical importance. *Key-stretching* [30] also called *password hashing* is the industry standard method of improving security of low entropy secrets like passwords.

With this approach the password p is *stretched* or *hashed* using a function H and a high entropy value called a *salt* s, $H(p,s) = p_H$. The output called a *password hash* p_H and the salt are persistently stored while the plain text password is discarded. When verifying the password p', it is stretched again $H(p',s) = p'_H$ with the stored salt s and the output hash p'_H is compared with the stored password hash $p'_H \stackrel{?}{=} p_H$, if it matches the password is correct.

Key-stretching [5] is traditionally done with CPU intensive hash iteration functions (PBKDF2, Bcrypt) [32, 37]. Recently, however memory-hard algorithms (Argon2, Scrypt, Balloon) [4, 6, 36] are becoming a standard choice,

because they protect against using special purpose hardware (ASIC) for calculating hashes.

2.3 Extensible Authentication Protocol

We will define our authentication system as a method in the extensible authentication protocol (EAP) framework.

Extensible authentication protocol [49] (EAP) is a general purpose authentication framework designed for network access authentication. It runs directly over the data link layer, such as PPP [44] and IEEE 802 [47]. EAP defines a set of messages that support negotiation and execution of a variety of authentication protocols. EAP is a two-party protocol between a *peer* and an *authenticator* at each end of a link.

2.3.1 Messages

The peer and the authenticator communicate by exchanging *EAP messages* (Figure 2.2). The protocol starts with the authenticator sending a message to the peer. They keep exchanging messages until the authenticator can either authenticate the peer or not.

Messages are exchanged in a lock-step manner, where an authenticator sends a message and the peer responds to it. The authenticator dictates the order of messages, meaning it can send a message at any point of communication, as opposed to the peer, which can only respond to messages from the authenticator. Any messages from the peer not in a response to the authenticator are discarded.

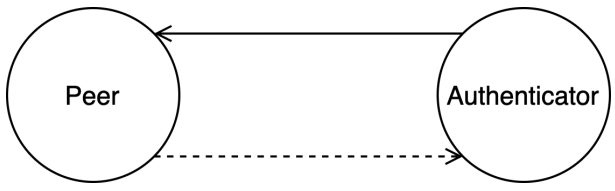

Figure 2.2: EAP Peer and Authenticator Communication

Length (Octets)	1	1	2	1	$n \leq 2^{16}$
Field	Code	Identifier	Length	Type	Type Data

Table 2.1: EAP Message Structure

Message Structure

Messages are composed of fields (Table 2.1), each field length is multiple of an octet of bits. Each field has a special purpose:

Code Field Determines who the packet is intended for and how or even should the recipient respond. The following code values are defined:

Request *Code 1.* Messages sent by the authenticator to the peer. Response is always expected.

Response *Code 2.* Messages sent by the peer to the authenticator as a reply to a *request* message.

Success *Code 3.* Sent by the authenticator, when the peer is successfully authenticated. The peer doesn't respond to the message.

Failure *Code 4.* Sent by the authenticator, when the peer cannot be authenticated. The peer doesn't respond to the message.

Identifier Field Used to create a session between the peer and the authenticator. The authenticator uses the field to match request and response messages, each response message needs to have the same identifier as the request. The authenticator will discard response messages that don't have a matching identifier with the current request. The peer does not re-transmit a response message, but relies on the authenticator to re-transmit a request message after some time if the matching response is lost.

Length Field Determines the total size of the EAP message. Because EAP provides support for generic authentication methods, the final length of the messages is variable. The length of the *type data* field entirely depends on the authentication method used.

***Type* and *Type Data* Field** Determines how the message should be processed and how to interpret the *type data* field. Most message types represent authentication methods, except for four special purpose types. The *type* used is determined by the authenticator when sending the request message. The response message from a peer needs to be of the same type as the request, except where that type is not supported by the peer. The following types are defined:

Identity *Type 1.* Used to query the identity of the peer. The type is often used as an initial message from the authenticator the peer, however its use is entirely optional and EAP methods should rely on method-specific identity queries.

Notification *Type 2.* Used to convey an informative message to the peer, by the authenticator. Usage of this type is entirely optional.

Nak *Type 3.* Used only as a response to a request, where the desired type is not available. The peer includes desired authentication methods, indicated by their type number. This type is also referred to as Legacy Nak, when compared to *Expanded Nak* (sub-type of the Expanded Type).

Expanded Type *Type 254.* Used to expand the space of possible message types beyond the original 256 possible types. The expanded type *data field* is composed from a *Vendor-ID* field, *Vendor-Type* and the type data.

Length (octets)		3	4	n
Field Type	...	Vendor-ID	Vendor-Type	Vendor-Type Data

A peer can respond to an unsupported request type with an *expanded nak*, if he desires to use an EAP method supported with the expanded type.

Experimental *Type 255.* This type is used for experimenting with new EAP Types and has not fixed format.

Authentication Methods The remaining types correspond to different authentication methods. IANA [41] assigns type numbers to 49 different authentication methods. Authors of the original RFC [49] already assigned 3 authentication protocols:

MD5-Challenge *Type 4.* An EAP implementation of the PPP-CHAP protocol [43].

One-Time Password *Type 5.* An EAP implementation of the one-time password system [31].

Generic Token Card *Type 6.* This type facilitates various challenge/response *token card* implementations.

Some other notable examples are EAP-TLS [43], EAP-PSK [2]. EAP SRP-SHA1 [15] is especially interesting as it uses a ZKP system to verify the peer's secret, similar to our own EAP method.

2.3.2 Pass-Through Behaviour

An authenticator can act as a *Pass-Through Authenticator*, by using the authentication services of a *backend authentication server*. In this mode of operation, the authenticator is relaying the EAP messages between the peer and the backend authentication server. For example, in IEEE 802.1x the authenticator communicates with a RADIUS server [46].

IEEE 802.1x Is a port based network access control standard for LAN and WLAN. It is part of the IEEE 802.11 group of network protocols. IEEE 802.1x defines an encapsulation of EAP for use over IEEE 802 as EAPOL or "EAP over LANs". EAPOL is used in widely adopted wireless network security standards WPA2. In WPA2-Enterprise, EAPOL is used for communication between the supplicant and the authenticator.

With WPA2-Enterprise, the authenticator functions in a pass-through mode and uses a RADIUS server to authenticate the supplicant. EAP packets between the authenticator and the authentications server (RADIUS) are encapsulated as RADIUS messages [10, 16, 46].

2.3.3 MD5-Challenge EAP Method

Let us look at an example of an EAP authentication process and examine the messages exchanged between the peer and the authenticator. This EAP instance uses the MD5-Challenge authentication method. MD5-Challenge is an EAP method analogous to PPP CHAP [45]. The message Type 4 denotes the method.

PPP CHAP. The PPP *challenge handshake authentication protocol* is an authentication model based on a shared secret between the peer and the authenticator. The authenticator authenticates the user by first sending him a random challenge c, which the user concatenates with the secret s and hashes with a hashing function $d = h(c|s)$. The hash digest d is returned in the response and the authenticator compares the received hash with the locally computed hash, if they match the peer is authenticated.

Aside from a slightly different message format, the MD5-Challenge authentication process is functionally the same as PPP CHAP, with the difference that while PPP CHAP is hashing algorithm agnostic, MD5-Challenge specifics the use of the MD5 hashing algorithm [39].

Let us examine the individual steps in EAP authentication with this method. The steps are visualised in the sequence diagram 2.3. At each step we will describe what is happening and note the contents of the EAP messages being exchanged, we are omitting the *identifier*, *length* and *type-data* fields of the messages, as their contents are dynamically determined when the protocol is running.

MD5-Challenge Message is sent to the peer with a random challenge c. ($Code = 1, Type = 4$).

Nak In the case that MD5-Challenge method is unacceptable to the peer, he should respond with a *Nak* message ($Code = 2, Type = 3$), the *type data* of the message can contain the number indicating the preferred EAP method.

MD5-Response The peer computes the hash digest d as described before with the MD5 hashing algorithm and sends it in a response. ($Code = 2, Type = 4$).

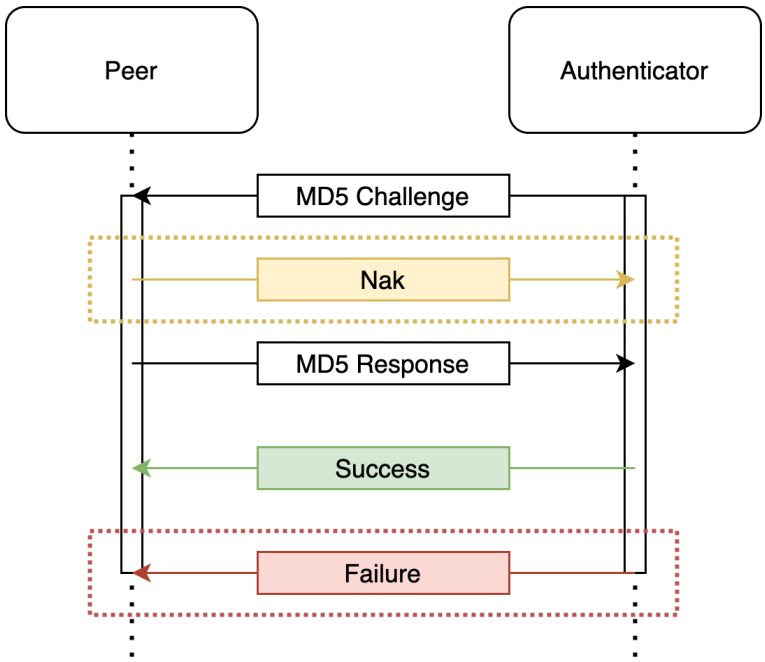

Figure 2.3: EAP (MD5-Challenge)

Success The authenticator sends this message if the digest d was successfully verified. The peer was successfully authenticated, and the protocol stops. ($Code = 3$).

Failure If the digest d isn't valid, the authenticator sends the *failure* message indicating that the peer could not be authenticated. ($Code = 4$).

2.4 Zero-Knowledge Proofs

In our authentication system, we wish to use a ZKP as a password verification method.

In this section we explore what ZKPs are on a high level, look at a practical analogy of how they work and also how they are used in real life. Next we look at what are *interactive proof systems*, the parent of ZKP systems. How to quantify the knowledge exchanged in an interactive proof system and finally what makes an interactive proof system zero-knowledge.

2.4.1 Basics

ZKPs are a concept for proving the validity of mathematical statements. What makes them particularly interesting is that ZKPs can prove a statement revealing no information about why a statement is true, hence the term *zero-knowledge*.

In mathematics, theorems are logical arguments that establish truth through inference rules of a deductive system based on axioms and other proven theorems. ZKPs are probabilistic, meaning they *convince* the verifier of the validity with a negligible margin of error. We use the term convince, because ZKPs are not absolute truth, but the chance of a false statement convincing a verifier is arbitrarily small. The difference in definition is subtle, but we will see what that means in practice further on.

ZKPs were first described by Goldwasser, Micali and Rackoff in [25] in 1985. They proposed a proof system as a two-party protocol between a *prover* and a *verifier*.

To help our understanding, we will explore the strange cave of Ali Baba, a famous analogy for a zero-knowledge protocol from a publication called "How to explain zero-knowledge protocols to your children" [38].

The Strange Cave of Ali Baba

Ali Baba's cave has a single entrance that splits into two tunnels that meet in the middle, where there is a door that only opens with a secret passphrase.

Peggy (or Prover) wants to prove to Victor (or Verifier) that she knows the passphrase, but she doesn't want to reveal it nor does she want to reveal her knowledge of it to anyone else besides Victor.

To accomplish this, they come up with a scheme. Victor stands in front of the cave and faces away from the entrance, to not see Peggy as she enters the cave, and goes into one tunnel at random. Victor looks at the entrance, so he can see both tunnels, and signals Peggy which tunnel to come out from. Peggy, knowing the passphrase, can go through the door in the middle and emerge from the tunnel requested.

If Peggy did not know the secret, she could fool Victor, only by entering the correct tunnel by chance. But since Victor is choosing the tunnel at

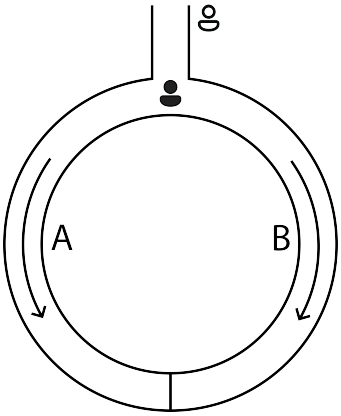

Figure 2.4: The Strange Cave of Ali Baba

random, Peggy's chance of picking the correct tunnel is $\frac{1}{2}$. If Victor was to repeat the process m time, her chances of Peggy fooling him become arbitrarily small ($\frac{1}{2^m}$).

This way Peggy can convince Victor that she knows the secret with an arbitrarily high probability of $(1 - \frac{1}{2^m})$.

Any third party observing the interaction cannot be convinced of the validity of the proof because they cannot be assured that the interaction was truly random. For example, Victor could have told Peggy his questions in advance, so Peggy would produce a convincing looking proof.

Applications

There have been several applications in the blockchains and decentralised identity systems. The cryptocurrency Zcash uses a *non-interactive zero--knowledge protocol* zk-SNARK [7] to prove the validity of transactions, revealing nothing about the recipients or the amount sent. Alternativley, *Idemix* [14] an anonymous credential system for interaction between digital identities relies on CL-signatures [13] to prove validity of a credential offline, without the issuing organisation. Idemix has been implemented in the open-source Hyperledger projects.

ZKPs can also prove that values satisfy complex constraints like range proofs [11].

2.4.2 Interactive Proof Systems

An interactive proof system is a proof system where a *prover* attempts to convince a *verifier* that a statement is true. The prover and the verifier interact with each other by exchanging data until the verifier is convinced or not.

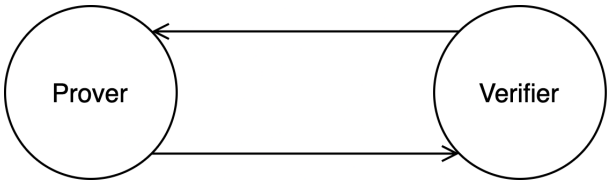

Figure 2.5: Interactive Proof System

The prover is a computationally unbounded polynomial time Turing machine and the verifier is a probabilistic polynomial time Turing machine. An interactive proof system is defined by properties *completeness* and *soundness*.

Notation

$\Pr[A]$: probability of event A happening.

$P(x) = y$: prover P, outputs a proof y for statement x.

$V(y) = 1$: verifier V, verifies proof y and outputs 1 or 0.

L: Language L is a set of *words* or values x, with a property P, $L = \{x \mid P(x)\}$. Often used to described a set of values that are a solutions to a mathematical problem. For example, a problem, find x larger than 3 and lesser than 7, has the associated language L_A is $L_A = \{x \mid 3 < x < 7\} = \{4,5,6\}$. Proving membership of x in L is equivalent to proving x is a solution to a problem P that defines the language L.

Completeness Any honest prover can convince the verifier with overwhelming probability.
For $x \in L$ and each $k \in \mathbb{N}$ and sufficiently large n;

$$\Pr[x \in L; P(x) = y; V(y) = 1] \geq 1 - \frac{1}{n^k}$$

Soundness Any verifier following the protocol will reject a cheating prover with overwhelming probability.
For $x \notin L$ and each $k \in \mathbb{N}$ and sufficiently large n;

$$\Pr[x \notin L; P(x) = y; V(y) = 0] \geq 1 - \frac{1}{n^k}$$

2.4.3 Zero-Knowledge

ZKP systems prove the membership of x in language L, revealing no additional knowledge (e.g. why is $x \in L$).
The essence of zero-knowledge is the idea that the data the verifier has (from current and past interactions with the prover) is indistinguishable from data that can be simulated from public information. For example, if we return to our analogy in the introduction. Victor wants to record what he sees to analyse, or to prove to someone else that Peggy knows the secret. Victor records which tunnels he calls and from which Peggy emerges, he doesn't record which tunnel Peggy goes into as he is facing away. Later on Bill and Monica decide to record a similar scheme without knowing the secret. Bill records himself calling the tunnels and Monica emerging randomly. Sometimes she emerges from the correct one, other times she doesn't. Bill later edits the video to only show the times Monica correctly emerged from the tunnel, as if she knew the secret. Assuming Bill's video editing skills are good, the videos Bill and Victor recorded are indistinguishable, both videos feature someone calling tunnels and a person correctly emerging. While Victor's video recorded a genuine proof, there is no information in the video from where we could prove that. The only one who can be truly convinced is Victor, because he trusts that his own choices of tunnels to call were truly random.

Indistinguishability

Indistinguishability describes the (in)ability of distinguishing between two sets of data. The *data* we are comparing is formalised as a random variable.

Let U and V be two families of random variables and $x \in L$. We are given a random sample x from either distribution U or V, we study the sample

to learn which distribution was the origin of x. U and V are said to be *indistinguishable* when our studying of x is no better than guessing randomly.

Approximability

The notion of approximability described the degree to which a process M could *generate* a distribution $M(x)$ that is indistinguishable from some distribution $U(x)$.

Formally, a random variable $U(x)$ is *approximable* if there exists a probabilistic Turing machine M, such that for $x \in L$, $M(x)$ is *indistinguishable* from $U(x)$.

Definition of Zero-Knowledge

Zero-knowledge is a type of interactive proof system in which we can't learn any meaningful information, besides the validity of the proof.

An interactive proof system is *zero-knowledge* if $V(x)$ data available to the verifier is *approximable* by $S(x)$ data that can be generated by a *simulator* S from public information. This also accounts for additional data that might be available to a cheating verifier, for example past interactions with the prover.

Strengths of Zero-Knowledge

There are three levels of zero-knowledge, defined by the strength of indistinguishability. We have defined *indistinguishability* as the ability of a *judge* to distinguish between random variables $V(x)$ and $S(x)$, by attempting to determine the origin of a sample x, taken randomly from either distribution. The strength of indistinguishability is determined by two parameters, the available *time* to analyse and the *size* of the sample.

$V(x)$ represents the verifiers view and $S(x)$ the generated data by the simulator S. Or if we return to our earlier analogy, $V(x)$ represents Victors interaction with Peggy, and $S(x)$ a fabricated recording of an interaction between Bill and Monica.

Perfect Zero-Knowledge $V(x)$ and $S(x)$ are **equal** when they remain indistinguishable even when given arbitrary time and an unbounded sample size.

Statistical Zero-Knowledge Two random variables are **statistically indistinguishable** when they remain indistinguishable given arbitrary time and a polynomial sized sample.

Let $L \subset \{0,1\}^*$ be a language. Two polynomial sized families of random variables V and S are *statistically indistinguishable* when,

$$\sum_{\alpha \in \{0,1\}^*} |P[V(x) = \alpha] - P[S(x) = \alpha]| < |x|^{-c}$$

for all constants $c > 0$ and all sufficiently long $x \in L$.

Computational Zero-Knowledge $V(x)$ and $S(x)$ are **computationally indistinguishable** when they remain indistinguishable given polynomial time and a polynomial sized sample.

Let $L \subset \{0,1\}^*$ be a language. Two polynomial sized families of random variables V and S are *statistically indistinguishable* for all poly-sized families of circuits C when,

$$|P[V,C,x] - P[S,C,x]| < |x|^{-c}$$

for all constants $c > 0$ and all sufficiently long $x \in L$.

2.5 Appropriate Problems for ZKP Systems

We have explored what defines an interactive proof system and what makes it zero-knowledge, but what are concrete examples of ZKP systems, and which statements can we even prove in zero-knowledge?

Whether we can prove a statement in zero-knowledge depends on the underlying mathematical problem. The problem also determines the ZKPs practical applications, simpler ZKPs are used to prove knowledge of a secret, while advanced ZKPs are used to prove signatures over committed values, set membership or range proofs [7, 9, 11, 13].

The ZKP system [25] used in our authentication system is based on the *quadratic residuosity problem*. We dive deep into how this ZKP system works, by exploring the mathematical foundation of quadratic residues, the quadratic residuosity problem, and the construction of the ZKP system.

We also look at examples of other problems and more broadly at classes of problems with ZKP systems.

2.5.1 ZKP System for the Quadratic Residuosity Problem

The first ZKP system defined [25] is for the *quadratic residuosity problem.* The quadratic residuosity problem is much older than ZKPs. It was first described by Gauss in 1801 [21]. The problem emerges when computing quadratic residues with a modulo that is a product of two unknown prime numbers. The properties of the problem enable it to be used as a *trapdoor* function. To define the problem, we must define the concept of quadratic residues and prime factorization.

Quadratic Residues

The concept [1] comes from modular arithmetic.

Definition (Quadratic residues)**.** *For* $x,n \in \mathbb{Z},\ n > 0,\ \gcd(x,n) = 1.$ *x is a quadratic residue if* $\exists w : w^2 \equiv x \pmod{n}$*, otherwise x is a quadratic non-residue.*

For example, 3 is a quadratic residue mod 11, because $6^2 = 36 \equiv 3 \pmod{11}$.

Generally, when n is an odd prime, x is a quadratic residue mod n if,

$$x^{\frac{n-1}{2}} \equiv 1 \pmod{n}.$$

Legendre Symbol $\left(\frac{x}{p}\right)$ is a convenient notation for computation of quadratic residues, and is defined as a function of x and p.

If p is an odd prime then,

$$\left(\frac{x}{p}\right) = \begin{cases} 1 & x \text{ is a quadratic residue modulo } p \\ -1 & x \text{ is a quadratic non-residue modulo } p \\ 0 & gcd(x,p) \neq 1 \end{cases}$$

$$\left(\frac{x}{p}\right) \equiv x^{\frac{p-1}{2}} \pmod{n} \quad \text{and} \quad \left(\frac{x}{p}\right) \in \{-1,0,1\}$$

Using the same example as before,

3 is a quadratic residue modulo 11

$$\left(\frac{3}{11}\right) \equiv 3^{\frac{11-1}{2}} = 243 \equiv 1 \pmod{11}$$

6 is a quadratic non-residue modulo 11

$$\left(\frac{6}{11}\right) \equiv 6^{\frac{11-1}{2}} = 7776 \equiv -1 \pmod{11}$$

Jacobi Symbol A generalised definition of the Legendre symbol $\left(\frac{x}{n}\right)$, to allow the case where n is any odd number.

If $n = p_1 p_2 \cdots p_r$, where p_i are odd primes, then

$$\left(\frac{x}{n}\right) = \left(\frac{x}{p_1}\right)\left(\frac{x}{p_2}\right)\cdots\left(\frac{x}{p_r}\right)$$

Unlike the Legendre symbol, if $\left(\frac{x}{n}\right) = 1$, x is a quadratic residue only if x is a quadratic residue of every prime factor of $n = p_1 p_2 \cdots p_r$.

Prime Factorization

The *fundamental theorem of arithmetic* [1] states that for each integer $n > 1$, exist primes $p_1 \leq p_2 \leq \cdots \leq p_r$, such that $n = p_1 p_2 \cdots p_r$.

For example,

$1995 = 3 \cdot 5 \cdot 7 \cdot 19$	$1996 = 2^2 \cdot 499$
$1997 = 1997$	$1998 = 2 \cdot 3^3 \cdot 37$

Prime factorization is the decomposition of an integer n to its prime factors $p_1 p_2 \cdots p_r$. The problem is considered *hard*, because currently no polynomial time algorithm exists for solving it [8]. It is in class *NP*, but is not proven to be *NP-complete*. The hardest instance of this problem is factoring the product of two prime numbers (*semiprimes*). The difficulty of this problem is a core building block in modern asymmetric cryptography like RSA [40].

Quadratic Residuosity Problem

Definition (Quadratic Residuosity Problem)**.** *Given an integer x, a semiprime modulus $n = pq$, where p and q are unknown different primes, and a Jacobi symbol value $\left(\frac{x}{n}\right) = 1$. Determine if x is a quadratic residue modulo n or not.*

As mentioned before, the problem emerges when computing the quadratic residue with a modulo that is a product of two unknown primes. The *law of quadratic reciprocity* enables efficient computation of the Jacobi symbol value $\left(\frac{x}{n}\right)$. However, if $\left(\frac{x}{n}\right) = 1$, it does not tell if x is a quadratic residue modulo n or not, x is only a quadratic residue if x is a quadratic residue of both modulo p and q ($\left(\frac{x}{p}\right) = \left(\frac{x}{q}\right) = 1$). To calculate this, we would have to know the primes p and q by factoring n. Since n is a product of two prime numbers, factoring it is computationally hard.

The only efficient way to prove $x \pmod n$ is a quadratic residue, is with the root w. The problem acts as a *trapdoor* function, where it's hard to prove if $x \pmod n$ is a quadratic residue solely from x and n, while it is easy to prove when you know w.

Zero-Knowledge Proof Protocol

To prove $x \pmod n$ is a quadratic residue in zero-knowledge we need to prove the existence of w, where $w^2 \equiv x \pmod n$, without revealing w to the verifier. Let us examine how the protocol [25] defined by Goldwasser, Micali, and Rackoff achieves that.

The Table 2.5.1 contains a notation presenting an execution of the ZKP protocol for the quadratic residuosity problem. The table displays consecutive steps in a process, the number on the left side of each row determines the step. The columns under each party displays computations by that party, the column between parties displays the information exchanged and direction of the exchange. This notation will be used in all further examples.

n Semiprime, where Jacobi $\left(\frac{x}{n}\right) = 1$
x Residue, where $w^2 \equiv x \pmod n$
w Root

	Prover		Verifier
1	$u \leftarrow_R \mathbb{Z}_n^*; y = u^2 \pmod n$	$\xrightarrow{y}$	
2		$\xleftarrow{b}$	$b \leftarrow_R \{0,1\}$
3	$z = uw^b \pmod n$	$\xrightarrow{z}$	verify $z^2 = yx^b \pmod n$

Table 2.2: ZKP Protocol for the Quadratic Residuosity Problem

The prover begins by picking up a random integer u from the field $\mathbb{Z}_n$, computing $y = u^2 \pmod{n}$ and sending y to the verifier. The verifier picks a random bit b and sends it to the prover. This random bit acts as the *split in the tunnel* of our earlier cave analogy. The prover computes the value z based on b and sends it over. The verifier checks the proof by asserting $z^2 \equiv yx^b \pmod{n}$, this is possible since

$$z^2 \equiv yx^b \pmod{n}$$
$$(uw^b)^2 \equiv u^2(w^2)^b \pmod{n}$$
$$u^2w^{2b} \equiv u^2w^{2b} \pmod{n}.$$

For each round a cheating prover has a $\frac{1}{2}$ probability of succeeding by correctly guessing the value of the random bit b, to improve the confidence of the proof this is repeated m times.

2.5.2 Computational Complexity Classes

We've looked at a ZKP protocol for a specific problem, but what other problems have ZKPs? We can broadly examine the existence of ZKPs with classes of problems. This is a vast topic, so we will look only at some points. This knowledge is unnecessary for understanding the focus of our work, but offers an interesting background of ZKPs.

Non-deterministic Polynomial Time (NP). Class of problems solvable by a poly-time non-deterministic Turing machine, their proofs can be verified by a poly-time deterministic Turing machine.

Authors [24] proved that every language in NP has a ZKP system, by defining a ZKP protocol for the Graph 3-Colouring problem (3-COL). *Minimum colouring problem* is a problem in graph theory, of what is the minimal k *proper* colouring of a graph, where no adjacent vertices are of the same colour. An instance of $(k = 3)$ colouring (3-COL) is proven to be *NP-Hard* because a polynomial reduction exists from *Boolean-Satisfiability problem* (3-SAT) to 3-COL [34]. According to Cook's theorem [19] SAT or its 3 literal instance 3-SAT is *NP-Complete*, and any language in $L \in NP$ can be reduced to an instance of 3-SAT. Furthermore because polynomial reductions are *transient*, any language $L \in NP$ can be reduced to an instance of 3-COL.

Chapter 3

System Architecture and EAP Method Definition

In this chapter, we will focus on the architecture of our authentication system and the definition of the EAP method. In the first section, we will examine how to utilise the ZKP protocol described in §2.5.1 as a password verification method. We need to be aware of the pitfalls of password authentication described in §2.2.2 and support necessary security practices. In the second section, we will define our authentication system as an EAP method in the EAP authentication framework described in §2.3 by defining the messages and processes necessary for execution of our authentication system.

3.1 System Architecture

In this section we will define the architecture of our authentication system, to do so we need to combine the model of password authentication we've examined in §2.2 and the ZKP system for quadratic residuosity problem from §2.5.1.

3.1.1 Password Verification

The purpose of password verification is to assert that the user authenticating knows the correct password p. How can we do this with the ZKP protocol? The ZKP protocol proves that x is a quadratic residue modulo n, by proving the knowledge of the root w, where $w^2 \equiv x \pmod{n}$.

To use this protocol for password verification, we can use the password p as the root w. The user and the authentication system can both follow the ZKP protocol, and the user will inevitably prove that he knows the password w, by proving that x is a quadratic residue modulo n. With this, the system can assert that the user knows the correct password.

Vulnerability

To verify the proof provided by the user, the system needs to know the quadratic residue x. Because the root $w = p$ is a password, this introduces a vulnerability as mentioned in §2.2.2. An attacker with access to x could crack the password w in an offline attack with pre-computed tables. As mentioned in §2.2.2, we need to use a key-stretching method to ensure adequate security against such offline attacks.

Theoretical Constraints of Key-Stretching Vulnerable Data

A common usage of a key-stretching method is to transform the vulnerable data stored in the authentication system. However, this approach doesn't work in our case. Let us explore how the authentication system verifies the proof, and why using key-stretching directly over stored data is an issue.

We assume the system can verify the proof and use key-stretching methods directly over the vulnerable data. However, we will see why this is not possible.

Proof Verification with Key-Stretched Data On the last step of the protocol the system verifies that

$$z^2 \equiv yx^b \pmod{n}.$$

If we stretch the vulnerable value x with a function H and a salt s

$$H(x,s) = x_H,$$

we can then verify the proof with an inverse function H^{-1}

$$z^2 = yH^{-1}(x_H,s)^b.$$

This is possible assuming a polynomial algorithm H^{-1} exists, however, since key-stretching methods are based on hashing functions which are one-way functions, we know that the probability of a polynomial algorithm H^{-1} to

successfully compute a *pseudo-inverse* is negligibly small, for all positive integers c [22]

$$\Pr[H(H^{-1}(H(x))) = H(x)] < |x|^{-c}.$$

Even if given unbounded time and resources, the *pseudo-inverse* $x' = H^{-1}(H(x))$ might not be equal to $x' \neq x$. The set $x' \in I_x$ are all values that map into $H(x) = H(x')$, and since H is not injective we know that $|I_x| \geq 1$. Meaning that the probability that $x' = x$ is

$$\Pr[H^{-1}(H(x)) = x] = \frac{1}{|I_x|}.$$

Key-Stretching the Root w

We've seen that key-stretching the vulnerable value x prevents us from verifying the ZKP. However, by increasing the entropy of the root w, we can eliminate the vulnerability and ensure adequate security.

Instead of treating the password p as the root w, we can instead derive the root w from the password p, by stretching it with a function H and a salt s

$$w = H(p, s),$$

and use the output as the root w. This way we've ensured the same level of protection as if we stretched the data stored in the system. Because we didn't change the value x, we can verify the proof without being affected by issues mentioned in §3.1.1.

3.1.2 Secure Authentication Process using ZKPs

By key-stretching the password to derive the root w, we've figured out how to secure our system while respecting the constraints imposed by the proof verification process. How does this change the authentication process we've described in §3.1.1?

In Table 3.1, variables in with an $_i$ subscript (e.g. y_i) are unique to each iteration i.

The process (Table 3.1) will now begin with the user computing the root w from the password p and salt s. Once the user computes the root w, he can authenticate by following ZKP protocol with the system, as mentioned in §2.5.1 Earlier we argued the ZKP works as a password verification method

	User		Authentication System
1	$w = H(p,s)$		
2	$u_i \leftarrow_R \mathbb{Z}_n$		
	$y_i = u_i^2$	$\xrightarrow{y_i}$	
3		$\xleftarrow{b_i}$	$b_i \leftarrow_R \{0,1\}$
4	$z_i = u_i w^{b_i} \pmod n$	$\xrightarrow{z_i}$	check $z_i^2 \equiv y_i x^{b_i} \pmod n$

Table 3.1: Secure Authentication Process using ZKPs

because $p = w$, this argument isn't true anymore. However, even though $w \neq p$, the user can only derive w knowing the password p, so when the user proves the knowledge of w, it can only be so because they know p as well. We repeat this part of the process m times for a confidence of $1 - 2^{-m}$.

3.1.3 Considerations

Performance. If we look at the steps that occur in our ZKP verification process (Table 3.1), we can notice iterations of data exchanges between the user and the system. In a real world environment, this can cause the authentication process to be slow because of network inefficiencies when transmitting data between the user and the authentication system. However since iterations are mutually independent, we can execute them in parallel (Table 3.2).

	Peer		Authenticator
1	$w = H(p,s)$		
2	$u_1, ..., u_m \leftarrow_R \mathbb{Z}_n$		
	$y_1, ..., y_m = u_1^2, ..., u_m^2 \pmod n$	$\xrightarrow{y_1,...,y_m}$	
3		$\xleftarrow{b_1,...,b_m}$	$b_1, ..., b_m \leftarrow_R \{0,1\}$
4	$z_1, ..., z_m = u_1 w^{b_1}, ..., u_m w^{b_m} \pmod n$	$\xrightarrow{z_1,...,z_m}$	check $z_1^2, ..., z_m^2 \equiv y_1 x^{b_1}, ..., y_m x^{b_m} \pmod n$

Table 3.2: Parallel ZKP Construction

What we are proposing is theoretically called a 3-round interactive ZKP. The existence of these proofs is limited only to a class of problems *BPP* [23]. Unfortunately, the quadratic residuosity problem is not believed to be in this class, so we assume a parallel proof to have a weaker notion of zero-knowledge.

For this purpose, we've used a sequential execution for our authentication process.

3.2 EAP Method Definition

We want to encapsulate our extended zero-knowledge authentication system defined in §3.1 as an EAP method in the EAP framework we've explored in §2.3. To achieve this, we must define a new EAP method, which comprises of the messages exchanged between the *peer* and the *authenticator*, their data formats and the processes for handling them.

Terminology. In this section we will use EAP terminology as described in §2.3, which uses different names to describe parties involved, as the ones used in our system architecture in §3.1 or as in the ZKP protocol in §2.5.1. The two parties in EAP are called the peer and the authenticator, where the peer is authenticating with the authenticator. To draw parallels between our system architecture, where we use the names *user* and *authentication system*, the peer is the user, and the authenticator is the authentication system. In the ZKP protocol names *prover* and *verifier* are used, the peer is the prover, and the authenticator is the verifier.

To define an EAP method, we need to break down our authentication system described in §3.1 to EAP messages representing interactions between the user and the authentication system. Each message defines its data format, the sender and recipient processes and local state changes. Our EAP method defines two messages, the *setup* message and the *verification* message.

We designed our authentication system for multiple users. For this reason, the authenticator needs to start the authentication process with the *identification phase* by querying the identity of the peer with the *identity* (Type 1) EAP message as described in §2.3.1. In the *setup phase* the peer uses the *setup* message for discovery of ZKP parameters, and to provide the values for the first proof verification round to the authenticator. This message is exchanged only once. In the *verification phase* the *verification* message is used to exchange data required for a single round of proof verification. Since proof verification as described in §2.5.1 requires m iterations, this message is exchanged m times. The protocol ends with the authenticator sending either a *success* or a *failure* message.

Authentication process overview

Let us examine the EAP messages (Figure 3.1) of the authentication process described in Table 3.3. The mapping between EAP messages and the steps in Table 3.3 is *not one-to-one*, as we merged some steps to reduce the number of message exchanges to speed up the whole process. In this section we present a simplified authentication process to present the general idea, while a detailed description is given in the next section.

The mathematical variables have the same meaning as in the system architecture described in §3.1. This applies to all further sections.

	Peer		Authenticator
1		$\xrightarrow{I}$	
2	$w = H(p,s)$	$\xleftarrow{s,n}$	
3	$u_i \leftarrow_R \mathbb{Z}_n$		
	$y_i = u_i^2 \pmod{n}$	$\xrightarrow{y_i}$	
4		$\xleftarrow{b_i}$	$b_i \leftarrow_R \{0,1\}$
5	$z_i = u_i w^{b_i} \pmod{n}$	$\xrightarrow{z_i}$	check $z_i^2 \equiv y_i x^{b_i} \pmod{n}$

Table 3.3: Improved ZKP Authentication with EAP

The authentication process (Figure 3.1) consists of multiple phases:

Identification Phase Used to establish the identity of the peer. The authenticator sends an *identity request* message, and the peer responds with an identifier I, which is used the locate the salt s and quadratic residue value x. (Table 3.3, Step 1.)

Setup Phase Used to exchange necessary values for the *verification phase*. The authenticator sends the salt s and modulus n in the *setup request* message, which the peer uses to derive the root w from the password p. The peer responds with y_1 which will be used in the first round of the verification phase. (Table 3.3, Steps 2. and 3.)

Verification Phase In this phase both parties continuously exchange data for m rounds to construct and verify the proof. In a given round i the authenticator sends the random bit b_i and the peer responds with the partial proof z_i. The peer also sends the y_{i+1} for the next verification round $i+1$, this is done as a performance optimization (Table 3.3, Steps

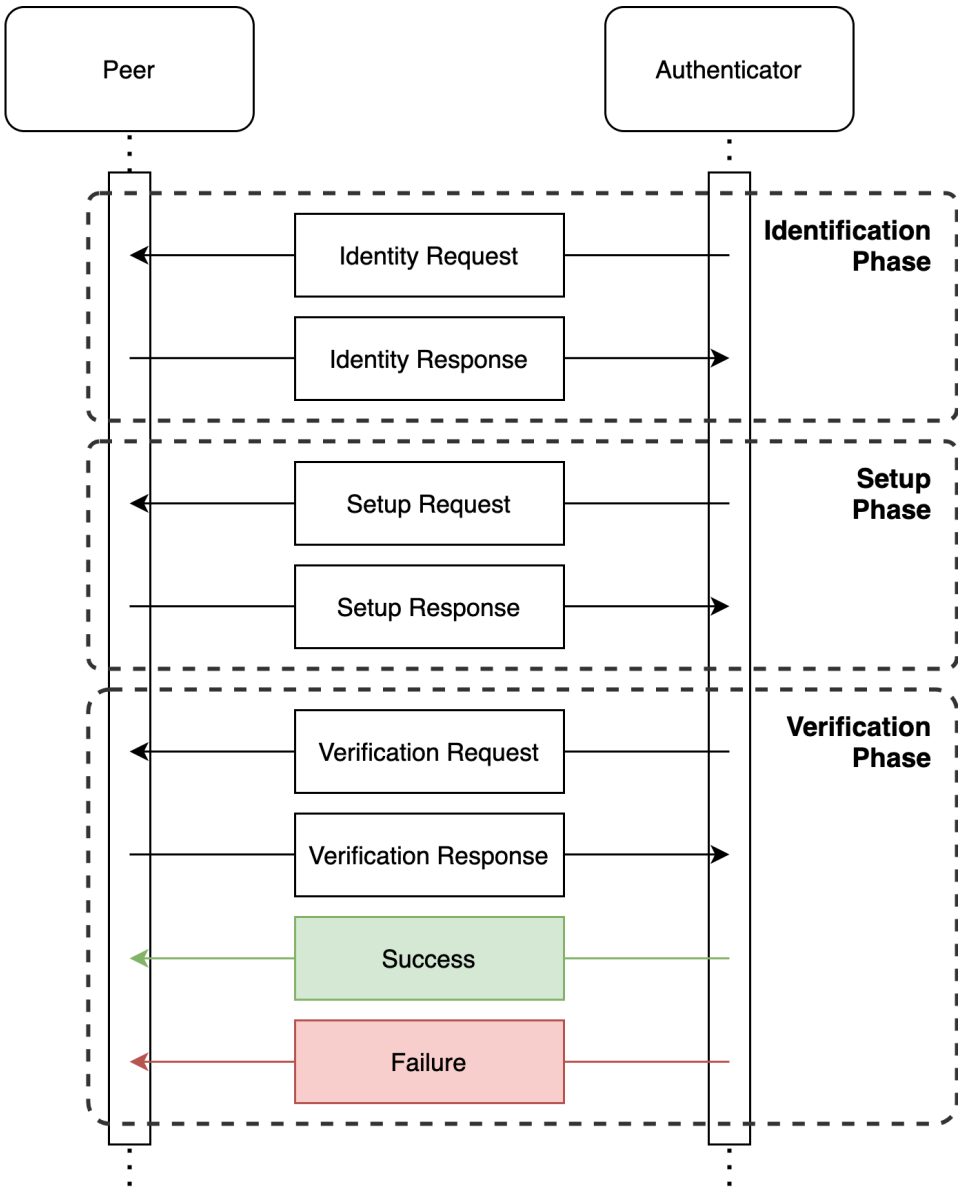

Figure 3.1: EAP Method Execution

4., 5. and 3. again). After receiving each response the authenticator verifies the partial proof, once he has done so for m successfully, he sends a *success* message. If the partial proof isn't valid, the authenticator must send a *failure* message.

Message Format

The EAP method *type* as described in §2.3.1 determines how the peer and authenticator should interpret the message *type-data*. We've decided to use the number 84 as our message *type*.

Additionally our authentication method is using a secondary field, called *sub type* to identify the phases of the authentication process. This field is defined within the *type-data* field and is meant to be interpreted only by our authentication method. We have defined 2 sub types, that correspond with the phases of our authentication process:

Sub Type 1 : Setup Message (Figure 3.1 - Setup Phase)

Sub Type 2 : Verification Message (Figure 3.1 - Verification Phase)

The *Identification Phase* in Figure 3.1 doesn't have a corresponding sub type as the *Identity* message as described in §2.3.1 is a pre-defined EAP message type.

Note also, that the architecture does not specify a key-stretching method, and neither does this EAP method. We assume that in a practical implementation the method would be pre-defined and implicitly known by both parties.

3.2.1 Setup Phase

Previously we assumed the modulus n and salt s are known by the user, however, the EAP method needs to facilitate the discovery of this data. Our system is designed to support multiple users, so before this phase the authenticator needs to identify the peer with the *identity* message. The response message additionally contains the control value y_1, used to verify the proof in the first verification round. The data is already included in this message to improve the method performance.

Request Message

Message is used to deliver the salt s and semiprime modulus n to the peer.

Type-Data Format

Length (Octets)	1	$4 \leq k \leq 255$	$64 \leq j$
Field	Salt Length	Salt	Modulus

Salt Length A single octet for the length of the *s*alt field in octets.

Salt A random salt value, should be from 4 octets to 255 octets long. The max length is determined by the largest number able to be encoded in the *salt length* field.

Modulus Fills the rest of the message to the length specified by the *length* field in the EAP message. Should be at least 64 octets (512 bits).

Request Handling When a request is received, the peer computes the root w using the password p the salt s with the pre-determined hashing function H. Root w should be stored stored in memory by the peer. Next the peer should pick a random integer u from field Z_n^*, and store it in memory and then computes the control value $y = u^2 \pmod{n}$. The control value y is included in the response message.

Response Message

The response contains the control value y_1 to the authenticator to be used in the first round of the verification process. The subscript of a variable y_i denotes in which verification round i is the variable used.

Type-Data Format

Length (Octets)	k
Field	Control Value y_1

Control Value Computed by the peer, where $y_1 = u_1^2 \pmod{n}$ and $u_1 \leftarrow_R \mathbb{Z}_n^*$.

Response Handling The authenticator should store the y_1 control value locally to be used when verifying the proof z_1.

3.2.2 Verification Phase

This message pair exchanges data required to compute and verify the proof. They continuously exchange it until the authenticator concludes the authentication. After m successful rounds, when the authenticator reaches a confidence of $1-2^{-m}$ in the proof, the authentication is successful. To make our method more efficient, we reduce the number of exchanged messages between the parties by interlacing some data between iterations. On the round i, the response contains data required for the round $i+1$.

Request Message

In the round i, the authenticator generates random bit b_i stores it locally, and sends it to the peer.

Type-Data Format

Length (Octets)	1
Field	Random Bit b_i

Random Bit A single-bit b_i, at the right-most place. 1 octet long.

Request Handling The peer computes the proof $z_i = u_i w^{b_i} \pmod{n}$, with the bit b_i received in the request.

Additionally the peer generates the control value y_{i+1} for the next $(i+1)$ verification round. The peer picks a random integer u_{i+1} from field Z_n^* and stores it in memory. The control value is computed as $y_{i+1} = u_{i+1}^2 \pmod{n}$ and sent in the response.

Response Message

The response transmits the proof z_i and the control value y_{i+1} to the authenticator, who verifies the proof and decides on how to proceed.

Type-Data Format

Length (Octets)	1	k	j
Field	Proof Length	Proof z_i	Control Value y_{i+1}

Proof Length A field one octet in length. Determines the length of the Proof field in octets.

Proof Value z_i computed by the peer, verified by the authenticator.

Control Value Value y_{i+1}, required to verify the proof of the $(i+1)$-th round.

Response Handling The authenticator should verify the proof by asserting that $z_i^2 \equiv y_i x^{b_i} \pmod{n}$. If the verification fails, the a *failure* message must be sent to the peer, otherwise a *success* message must be sent if the verification was successful for m rounds. If that is not the case, the y_{i+1} is stored by the authenticator and a new verification message request is sent.

Chapter 4

Conclusions and Future Work

The aim of this thesis was to study the use of ZKPs as an authentication mechanism. In section §3.1, we have presented the architecture of an authentication system, which uses a ZKP protocol as the password verification method. We have described how the ZKP protocol can prove the knowledge of the user's password. The architecture also supports key-stretching for protection against password vulnerabilities discussed in §2.2.2. In section §3.2, we have encapsulated our authentication system within EAP by defining a specification for a new EAP method. The specification contains definitions for EAP messages and their handling procedures.

We have been successful in our goal of studying and using the ZKP protocol. However, upon observation, the system performance is not on par with today's industry standards. The iterative nature of the underlying ZKP protocol accumulates communication latencies, slowing down the system.

Future work.

- The EAP method specification presented in this work can be implemented and tested in a real-world environment.
- The ZKP protocol used in this work is a first generation protocol. Today there are many newer protocols that have solved many shortcomings of the older generation ZKPs. Using a newer generation ZKP protocol can improve the performance of the authentication system.
- The ZKP protocol we've examined is iterative, which can cause worse performance. We've discussed an alternative proof construction in

§3.1.3, which we aren't pursuing in this thesis because of assumed weaker strength of zero-knowledge. However, in a real-world application, the performance improvements might justify the theoretical shortcomings.

Bibliography

[1] G.E. Andrews. *Number Theory*. Dover Books on Mathematics. Dover Publications, 1994.

[2] Florent Bersani and Hannes Tschofenig. The EAP-PSK Protocol: A Pre-Shared Key Extensible Authentication Protocol (EAP) Method. RFC 4764, January 2007.

[3] Kane Baxter Bignell. Authentication in an internet banking environment. In *International Conference on Internet Surveillance and Protection (ICISP'06)*, pages 23--23. IEEE, 2006.

[4] Alex Biryukov, Daniel Dinu, and Dmitry Khovratovich. Argon2: new generation of memory-hard functions for password hashing and other applications. In *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 292--302. IEEE, 2016.

[5] Jeremiah Blocki, Benjamin Harsha, and Samson Zhou. On the economics of offline password cracking. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 853--871. IEEE, 2018.

[6] Dan Boneh, Henry Corrigan-Gibbs, and Stuart Schechter. Balloon hashing: A memory-hard function providing provable protection against sequential attacks. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 220--248. Springer, 2016.

[7] Sean Bowe, Ariel Gabizon, and Matthew D Green. A multi-party protocol for constructing the public parameters of the Pinocchio zk-SNARK. In *International Conference on Financial Cryptography and Data Security*, pages 64--77. Springer, 2018.

[8] Johannes A. Buchmann. *Factoring*, pages 171--183. Springer US, New York, NY, 2001.

[9] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 315--334. IEEE, 2018.

[10] Pat R. Calhoun and Dr. Bernard D. Aboba. RADIUS (Remote Authentication Dial In User Service) Support For Extensible Authentication Protocol (EAP). RFC 3579, September 2003.

[11] Jan Camenisch, Rafik Chaabouni, et al. Efficient protocols for set membership and range proofs. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 234--252. Springer, 2008.

[12] Jan Camenisch, Rafik Chaabouni, and Abhi Shelat. Efficient protocols for set membership and range proofs. In Josef Pieprzyk, editor, *Advances in Cryptology - ASIACRYPT 2008*, pages 234--252, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[13] Jan Camenisch and Anna Lysyanskaya. An efficient system for non--transferable anonymous credentials with optional anonymity revocation. In *International conference on the theory and applications of cryptographic techniques*, pages 93--118. Springer, 2001.

[14] Jan Camenisch and Els Van Herreweghen. Design and implementation of the idemix anonymous credential system. In *Proceedings of the 9th ACM conference on Computer and communications security*, pages 21--30, 2002.

[15] James D. Carlson, Dr. Bernard D. Aboba, and Henry Haverinen. PPP EAP SRP-SHA1 Authentication Protocol. Internet-Draft draft-ietf-pppext-eap-srp-03, Internet Engineering Task Force, July 2001. Work in Progress.

[16] Jyh-Cheng Chen and Yu Ping Wang. Extensible Authentication Protocol (EAP) and IEEE 802.1x: Tutorial and Empirical Experience. *IEEE Communications Magazine*, 43(12):S26--S32, January 2005.

[17] Sang-Rae Cho, DS Choi, SH Jin, and HH Lee. Passwordless authentication technology-FIDO. *Electronics and Telecommunications Trends*, 29(4):101--109, 2014.

[18] Art Conklin, Glenn Dietrich, and Diane Walz. Password-based authentication: a system perspective. In *37th Annual Hawaii International Conference on System Sciences, 2004. Proceedings of the*, pages 10--pp. IEEE, 2004.

[19] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, STOC '71, page 151–158, New York, NY, USA, 1971. Association for Computing Machinery.

[20] ECB. Recommendations for the security of internet payments. Technical report, ECB, 2013.

[21] Carl Friedrich Gauss and Arthur A. Clarke. *Disquisitiones Arithmeticae*. Yale University Press, 1965.

[22] Oded Goldreich. *Foundations of cryptography: Volume 1, basic tools*. Cambridge university press, 2007.

[23] Oded Goldreich and Hugo Krawczyk. On the composition of zero-knowledge proof systems. *SIAM Journal on Computing*, 25(1):169--192, 1996.

[24] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to prove all np statements in zero-knowledge and a methodology of cryptographic protocol design (extended abstract). In Andrew M. Odlyzko, editor, *Advances in Cryptology --- CRYPTO' 86*, pages 171--185, Berlin, Heidelberg, 1987. Springer Berlin Heidelberg.

[25] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on computing*, 18(1):186--208, 1989.

[26] Paul A Grassi, Michael E Garcia, and James L Fenton. NIST Special Publication 800-63-3 Digital Identity Guidelines. *National Institute of Standards and Technology, Los Altos, CA*, 2017.

[27] CNSS Glossary Working Group et al. National information assurance (IA) glossary. *Committee on National Security Systems Instruction 4009*, 2006.

[28] Morey J Haber. Attack vectors. In *Privileged Attack Vectors*, pages 65--85. Springer, 2020.

[29] Daira Hopwood, Sean Bowe, Taylor Hornby, and Nathan Wilcox. Zcash protocol specification. *GitHub: San Francisco, CA, USA*, 2016.

[30] Taylor Hornby. Salted password hashing-doing it right, 2016.

[31] Philip J. Nesser II, Mike Straw, Craig Metz, and Neil M. Haller. A One-Time Password System. RFC 2289, February 1998.

[32] Burt Kaliski. PKCS #5: Password-Based Cryptography Specification Version 2.0. RFC 2898, September 2000.

[33] Robert McMillan. The world's first computer password? It was useless too, 2012.

[34] Cristopher Moore and Stephan Mertens. *The nature of computation*. OUP Oxford, 2011.

[35] Nancy Odegaard and Vicki Cassman. *Authentication and Conservation in Archaeological Science*, pages 702--711. Springer New York, New York, NY, 2014.

[36] Colin Percival and Simon Josefsson. The scrypt Password-Based Key Derivation Function. *IETF Draft: The scrypt Password-Based Key Derivation Function*, 2012.

[37] Niels Provos and David Mazieres. Bcrypt algorithm. In *USENIX*, 1999.

[38] Jean-Jacques Quisquater, Myriam Quisquater, Muriel Quisquater, Michaël Quisquater, Louis Guillou, Marie Annick Guillou, Gaïd Guillou, Anna Guillou, Gwenolé Guillou, and Soazig Guillou. How to Explain Zero-Knowledge Protocols to Your Children. In Gilles Brassard, editor, *Advances in Cryptology --- CRYPTO' 89 Proceedings*, pages 628--631, New York, NY, 1990. Springer New York.

[39] Ronald L. Rivest. The MD5 Message-Digest Algorithm. RFC 1321, April 1992.

[40] Ronald L Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120--126, 1978.

[41] Joseph Salowey. Extensible authentication protocol (EAP) registry, method types. *IANA Extensible Authentication Protocol (EAP) Registry*, 2004.

[42] Robert W. Shirey. Internet Security Glossary, Version 2. RFC 4949, August 2007.

[43] Daniel Simon, Ryan Hurst, and Dr. Bernard D. Aboba. The EAP-TLS Authentication Protocol. RFC 5216, March 2008.

[44] William A. Simpson. The Point-to-Point Protocol (PPP). RFC 1661, July 1994.

[45] William A. Simpson. PPP Challenge Handshake Authentication Protocol (CHAP). RFC 1994, August 1996.

[46] Andrew H. Smith, Glen Zorn, John Roese, Dr. Bernard D. Aboba, and Paul Congdon. IEEE 802.1X Remote Authentication Dial In User Service (RADIUS) Usage Guidelines. RFC 3580, September 2003.

[47] William Stallings. *A Tutorial on the IEEE 802 Local Network Standard*, page 1–30. Computer Science Press, Inc., USA, 1986.

[48] Andrew Tobin and Drummond Reed. The inevitable rise of self-sovereign identity. *The Sovrin Foundation*, 29(2016), 2016.

[49] John Vollbrecht, James D. Carlson, Larry Blunk, Dr. Bernard D. Aboba, and Henrik Levkowetz. Extensible Authentication Protocol (EAP). RFC 3748, June 2004.