Zero-Knowledge Authentication

Jakob Povšič

July 25, 2021

Contents

Abstract

We design an authentication system to authenticate users over a network with a username and a password. The system uses the zero-knowledge proof (ZKP) of quadratic residuosity as a password verification mechanism. It is designed on top of the Extensible Authentication Protocol (EAP) framework as an EAP method. The ZKP verification yields interesting security properties that make the system favourable to be used over insecure networks.

Chapter 1

Introduction

1.1 Motivation

Our lives are becoming more digital everyday, and with big tech companies whose business models rely on accessing user data, privacy is becoming more important every day. It seems that to participate in digital spaces we have to sacrifice some privacy, however technologies like zero-knowledge proofs could help us retain it. Zero-knowledge proofs are a fascinating cryptographic phenomenon for proving mathematical statements, without revealing *why* they are true. This has incredibly interesting real world applications.

Cryptocurrencies like Zcash [31] are using zero-knowledge proofs to validate transactions on their networks while keeping transaction senders, recipients anonymous and amounts opaque.

The Self Sovereign Identity space is using zero-knowledge proofs and blockchain technologies to build a decentralised and privacy preserving digital identity infrastructure. Zero-knowledge proofs enable asking complex questions about sensitive user data in a completely privacy preserving manner [13]. For example, proving you are over 18 without revealing your date of birth, or that you hold a certain amount of funds in your bank account without disclosing your financial statements.

Advancements like this hint that we will look at this time and our attitude to personal data handling, as before we started washing our hands.

1.2 Focus of the Thesis

Our work will be focused on building a password authentication system using zero-knowledge proofs as a method of verifying the password. When creating a password authentication system we have to protect ourselves from vulnerabilities of passwords, however the integration of methods to improve security is not as straight forward as in regular password authentication systems, because of the underlying zero-knowledge proof. Our system is defined as an authentication method on top of the extensible authentication protocol (EAP).

1.3 Structure of the Thesis

This thesis is composed of three chapters. In §1 we explain the motivation behind the thesis, the focus of the thesis and its structure. In §2 we explore the two main topics of *authentication* and *zero-knowledge proofs*. In §3 we present the architecture of our authentication system and the extensible authentication protocol method definition.

Chapter 2

Methodologies and Tools

Authentication and zero-knowledge proofs. Authentication splits into a primer into authentication, password authentication and EAP. ZKP - Introduction, Example, Definitions, Our Choice of ZKP

In this chapter we will dive deep into *authentication* and *zero-knowledge proofs.*

In this chapter we will focus on what is *authentication* and *password authentication*

We will explore password auth

FUCK THIS COME BACK LATER!!!

2.1 Authentication

Authentication is the process of proving a claim or an assertion. Today it is most commonly used in information security [44], however methods of authentication are not limited to computer science and are also used in fields of archeology, anthropology and others [36].

In computer science authentication is commonly used for establishing access rights between restricted system resources and users through digital identities. Government and international institutions have developed guidelines for managing digital identities and authentication processes [28] .

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

Authentication Process Components

Authentication [44] is the process of 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 users identity (e.g at login), however assertions can be made and verified 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 a restricted system resources of an external users.

When designing an authentication system it's important to understand all components of the authentication process.

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 an identifier can be determined by an IP address the user wants to authenticate from, or a system might only have a single identity that can authenticate.

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

Our system will verify the users password with a zero-knowledge proof.

Authentication Factors

Authentication systems can rely on three groups of factors [21].

- **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 [22, 43], is a system using two or more factors. This is also referred to 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 [34].

We need to understand the high level model of password authentication, its risks and tools to mitigate it.

Authentication Model

Password based authentication is a simple model, based on a shared secret between the user and the system. The secret is referred to as a password, because the secret is usually a set of characters or words memorised by the user, inputted via a keyboard. The password is often used in a combination with a username.

To authenticate the user exchanges the password with the system, and the system authenticates the user if the password is correct.

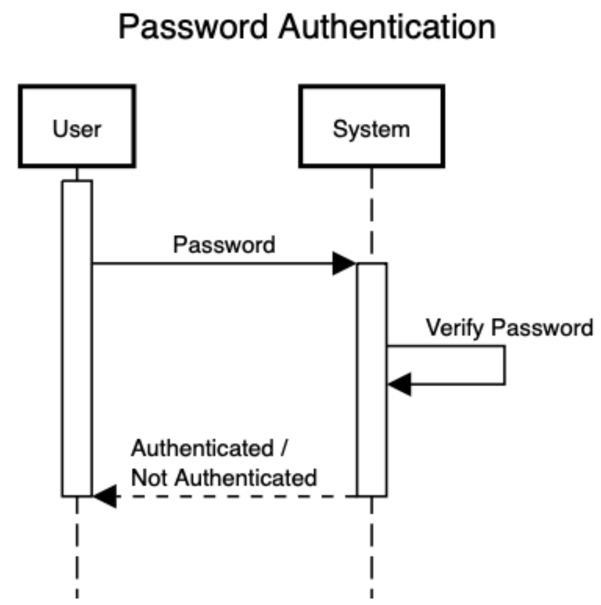

Figure 2.1: Password Authentication Model

2.2.1 Security Vulnerabilities

In a common password authentication system implementation 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 is what makes them a vulnerability for systems that rely on them.

Because passwords are supposed to be memorised and the proliferation of websites requiring them, users tend to pick password that are easier to remember and reuse passwords across different websites [19]. Many websites also don't properly handle passwords, enabling attackers to access plain--text passwords when a security breach happens. The industry is aiming to improve password security with the adoption of password managers and initiatives like FIDO [4] working to retire passwords altogether.

Attacks can be according to NIST [28] classified as *online* or *offline*, based on wether the attacker is directly interacting with an authentication system.

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 it and react. For example, locking an account after 5 failed authentication attempts. For this reason, most online attacks are not very effective.

Effective online attacks work by operating under the radar of detection, for example, by trying out a small number of passwords on each user. Popular methods are *password spraying* and *credential stuffing* [29], 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 small number of commonly used passwords and attempting to authenticate with a large number of accounts, the attacker is assuming that in a large sample of accounts some will be using 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 file, eavesdropping an authentication protocol or a system penetration.

Password cracking [7] is method of extracting user credentials from data used by the authentication system to verify users credentials. The success of password cracking is generally determined by two parameters, 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 different things an authentication system can incorporate to improve its security. An authentication system can adopt techniques for preventing active attacks and improving password strength independently of the underlying password verification methods. We are going to be focusing on methods for handling passwords on the data layer, where we protect ourselves against offline attacks. The form in which passwords exits on the data layer is also constrained by the ZKP protocol used for password verification.

Key-Stretching Protecting passwords on the data layer is of critical importance. *Key-stretching* [32] 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, the output called a *password hash* p_H and the salt are stored in persistent memory while the plain text password is discarded.

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

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 $p'_H \overset{?}{=} p_H$, if it matches the password is correct.

Key-stretching [7] is traditionally done with hash iteration functions (PBKDF2, Bcrypt) [33,38], these algorithms are CPU intensive, however are vulnerable to attackers with special purpose hardware (ASIC), so a better choice are memory-hard algorithms (Argon2, Scrypt, Balloon) [6,8,37].

2.3 Extensible Authentication Protocol

Our authentication system is defined as a method in the extensible authentication protocol (EAP) framework.

Extensible authentication protocol [1] (EAP) is a general purpose authentication framework, designed for network access authentication. It runs directly over the data link layer such as PPP [46] and IEEE 802 [48]. 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 the each end of a link. The terms *peer* and *authenticator* are EAP terminology.

2.3.1 Messages

The peer and the authenticator communicate by exchanging *EAP messages*. 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.

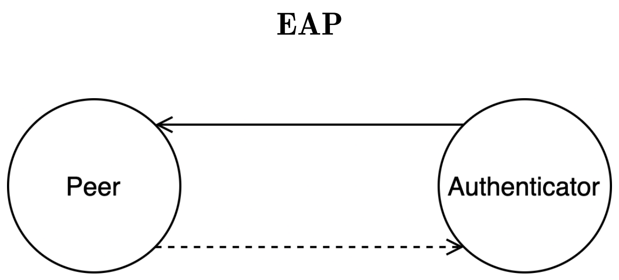

Figure 2.2: Peer and Authenticator Communication

Message Structure

Messages are composed of fields, each field length is multiple of an octet of bits.

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

Each field has a special purpose.

Code Field The code field determines who the packet is intended for and how or even should the recipient respond.

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 The identifier field is used 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 response message, but relies on the authenticator to re-transmit a request message after some time if the matching response is lost.

Length Field The 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 is entirely dependent on the authentication method used.

***Type* and *Type Data* Field** The *type* field determines how the message should be processed and how to interpret the *type data* field. Most message types represent authentication methods, except 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 in cases where that *type* is not supported by the peer.

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 [42] assigns type numbers to 49 different authentication methods. The original RFC [1] already assigned 3 authentication protocols.

MD5-Challenge *Type 4.* An EAP implementation of the PPP-CHAP protocol [45].

One-Time Password *Type 5.* An EAP implementation of the one-time password system [30].

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

Some other notable examples are EAP-TLS [45], EAP-PSK [5]. EAP SRP-SHA1 [16] is especially interesting as it uses a ZKP system to verify the peers 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 [18].

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 [2, 17, 18]

2.3.3 MD5-Challenge EAP method

Let us look at an example of an EAP authentication process and examine 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 [47]. The method is denoted by the message type 4.

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 [40].

Let us examine the individual steps in EAP authentication with this method. The steps is visualised in the sequence diagram 2.3. At each step we will describe what is happening and note 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 by 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. $(Code = 2, Type = 4)$.

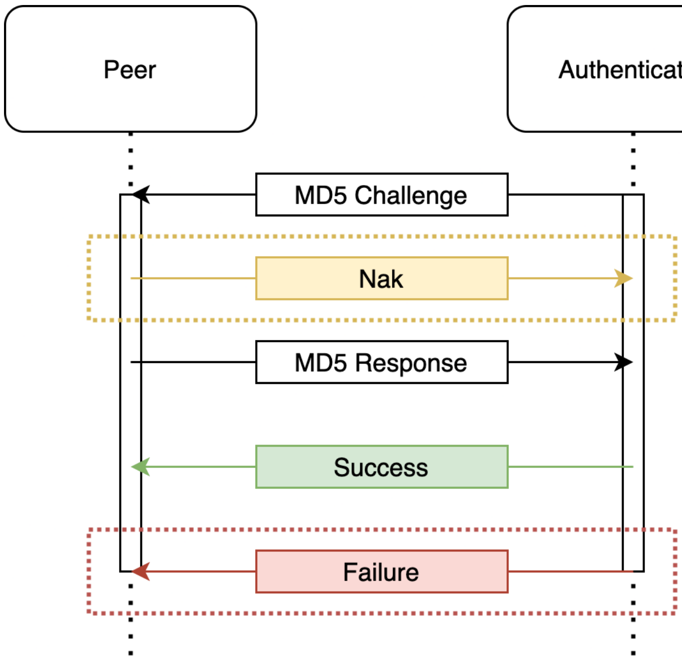

Figure 2.3: EAP (MD5-Challenge)

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 zero-knowledge proofs 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. And what is *knowledge complexity*, or how to quantify the knowledge exchanged in an interactive proof system and finally what makes an interactive proof systems zero-knowledge.

2.4.1 Basics

Zero-Knowledge Proofs (ZKPs) are a concept in the field of cryptography used for proving the validity of mathematical statements. What makes them particularly interesting is that ZKPs can prove a statement without revealing any information about why a statement is true, hence the term *zero-knowledge*.

In mathematics, theorem proofs 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 rather 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 [27] 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 [39] *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".

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 can only be opened with a secret passphrase.

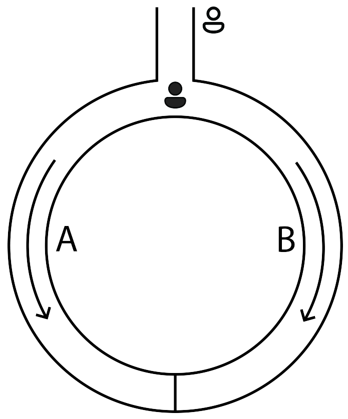

Figure 2.4: The Strange Cave of Ali Baba

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

To accomplish this they come up with a scheme. Victor stands in front of cave and faces away from the entrance, to not see Peggy as she enters the cave, and goes into one of the tunnels 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 secret can pass 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 random, Peggy's chance of picking the correct tunnel is $\frac{1}{2}$. If Victor was to repeat the process n time, her chances of Peggy fooling him become arbitrarily small $(\frac{1}{2^n})$.

With this process Victor can be convinced that Peggy knows the secret with an arbitrarily high probability of $(1 - \frac{1}{2^n})$.

Further more 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

Most commonly ZKPs were used in authentication and identification systems, as a way to prove knowledge of a secret. Recently however there have been a number of new applications in the cryptocurrency and decentralised identity systems.

The cryptocurrency Zcash uses a *non-interactive zero-knowledge protocol* zk-SNARK [9] to prove the validity of transactions, without revealing anything about the recipients nor the amount sent.

Idemix [15] an anonymous credential system for interaction between digital identities relies on CL-signatures [14] to prove validity of a credential offline, without the issuing organisation. Idemix has been implemented in the open-source Hyperledger projects.

ZKPs can be also used to prove that value satisfy complex constraints like range proofs [12].

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.

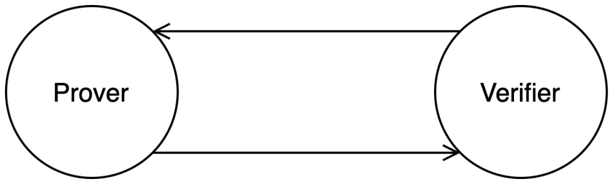

Figure 2.5: Interactive Proof System

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, $L = \{x|P(x)\}$. 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 $L_A = \{x|3 < x < 7\} = \{4,5,6\}$. 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

Zero-knowledge proof systems prove the membership of x in language L, without revealing any 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 with known information. For example, if we return to our analogy in the introduction. Victor wants to record what he sees to later analyse, or to prove to someone else that Peggy knows the secret. Victor manages to record 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 Victors 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 set 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 an interactive proof system, in which no meaningful information can be learned, besides the validity of the proof.

Zero-knowledge is a description of an interactive proof systems, at which we cannot extract any meaningful information from the data available to the verifier.

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* 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 Zero-Knowledge Proof 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 even be proven in zero-knowledge?

Whether a statement can be proven 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 [9, 11, 12, 14].

The ZKP system [27] 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 [27] is for the *quadratic residuosity problem*. The quadratic residuosity problem is much older than ZKPs, it was first described by Gauss in 1801 [23]. 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 [3] comes from modular arithmetic.

Definition 2.5.1 (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 and only if.

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

Legendre Symbol $\left(\dfrac{x}{p}\right)$ 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 $\left(\dfrac{x}{m}\right)$, to allow the case where m is any odd number.

If $m = p_1 p_2 \cdots p_n$, where p_i are odd primes, then

$$\left(\frac{n}{m}\right) = \left(\frac{n}{p_1}\right)\left(\frac{n}{p_2}\right)\cdots\left(\frac{n}{p_n}\right)$$

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 $n = p_1 p_2 \cdots p_n$.

Prime Factorization

The *fundamental theorem of arithmetic* [3] 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 [10]. 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 [41].

Quadratic Residuosity Problem

Definition 2.5.2 (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 us to efficiently compute the Jacobi Symbol $\left(\frac{x}{n}\right)$. However if $\left(\frac{x}{n}\right) = 1$, it does not necessarily 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 easy to prove when you know its root w.

Zero-Knowledge Proof Protocol

To prove $x \pmod{n}$ is a quadratic residue in zero-knowledge we need to prove the existence of the root w, where $w^2 \equiv x \pmod{n}$, without revealing w to the verifier. Let us examine how the protocol [27] defined by Goldwasser, Micali, and Rackoff achieves that.

The bottom table 2.1 is a notation presenting an execution of a process, we will be using this notation in all further examples. 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 done by each party, the column between parties displays the information exchanged and direction of the exchange.

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

Table 2.1: Protocol

	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 $z^2 = yx^b \pmod{n}$

The prover begins by picking a random integer u from 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? Existence of ZKPs can be more broadly examined with classes of problems, this is a vast topic, so we will look only at some points. This knowledge is not necessary for understanding the focus of our work, but offers an interesting background of zero-knowledge proofs.

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 [26] 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 [35]. According to Cook's theorem [20] 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 §2.2.1 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 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 as a password verification method we can treat the root w as the password $w = p$ known by the user. If the user assumes the role of prover and the authentication system the role of the verifier, when both follow the ZKP protocol, the user will prove that x is a quadratic residue modulo n, to do this however the user needs to prove the knowledge of the password w. With this, the system can assert that the password is valid.

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 §2.2.1. An attacker with access to x could crack the password w in an offline attack with pre-computed tables. As mentioned in §2.2.1, we need to use a key-stretching method to ensure adequate security.

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 [24]

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

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 w, 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. Further more 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 the proof verification process. How does this change the authentication process we've described in §3.1.1?

The process begins with the user deriving the root w from the password w and salt s.

	User		Authentication System
1	$w = H(P, s)$		

Once the user derives the root w, he can authenticate, by following ZKP protocol with the system as mentioned in §2.5.1 Earlier we argued that the ZKP works as a password verification method 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. This part of the process is repeated m times for a confidence of $1 - 2^{-m}$

1	$u \leftarrow_R \mathbb{Z}_n$		
	$y = u^2$	$\xrightarrow{y}$	
2		$\xleftarrow{b}$	$b \leftarrow_R \{0, 1\}$
3	$z = uw^b \pmod{n}$	$\xrightarrow{z}$	assert $z^2 \equiv yx^b \pmod{n}$

3.1.3 Considerations

Parallel ZKP Composition If we look at the steps that occur in our ZKP verification process, we can notice many 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. Can we improve this by running the steps in parallel instead of sequentially? If look closely at the data, we can see that at any given step there are no dependencies to data from the previous step, meaning that technically nothing is preventing us from running this in parallel.

What we are proposing is theoretically called a 3-round interactive zero--knowledge proof. The existence of these proofs is limited only to a class of problems *BPP* [25]. Unfortunately the quadratic residuosity problem is not believed to be in this class, so a parallel proof is assumed to have a weaker notion of zero-knowledge.

For this purpose we've decided to use 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 consists of messages exchanged between the *peer* and the *authenticator*, their data formats and how processes for handling them.

Terminology In this section we will be using 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 of two message pairs, the *setup* message pair and the *verification* message pair. The message pairs are not an identical mapping to the exchanges as described in §2.5.1, but are slightly interlaced to optimise the number of messages sent for a successful authentication.

Our authentication system was designed for systems with multiple users, for this reason the authenticator needs to start the authentication process by querying the identity of the peer with the *identity* (type 1) EAP message. The *setup* message pair is used for discovery of ZKP parameters by the peer, and to provide the values for the first proof verification round to the authenticator. This message pair is exchange only once. The *verification* message pair 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 pair is exchange m times. The protocol ends with the authenticator sending either a *success* or a *failure* message.

Authentication steps

Let us examine the steps involved in authentication with our EAP method. We are going to be referencing messages displayed in the sequence diagram in 3.1. The purpose of this section is to visually aid the understanding of the authentication process, meaning that some nuances are left our for the purpose of simplification, a detailed definition of individual messages is 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.

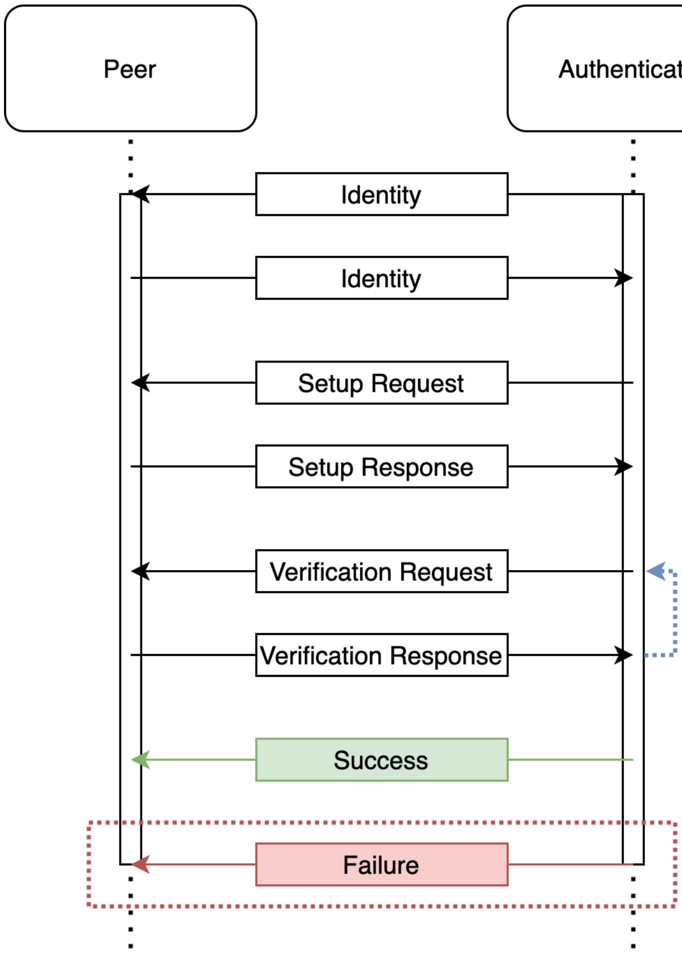

Figure 3.1: EAP Method Execution

Identity This message is used to query the identity of the peer. In a system with multiple users this is required to identify the peer authenticating, but more specifically the authenticator needs to find the correct salt s and quadratic residue x.

Setup The peer needs both the salt s and the modulus n to compute the proof, however he only know the password p. Once the peer identifies himself, the authenticator needs to send him the salt s and modulus n in the *setup request* message.

Verification With this message pair the peer and the authenticator exchange data to compute and verify the ZKP. The authenticator send the random bit b and the peer responds with the proof z. The peer also sends the control value y_{i+1} for the next verification round $i+1$, this is done as an optimisation to improve the speed of the process.

Success/Failure After each *verification* message, the authenticator verifies the proof, and once it was successfully verified for m rounds, the authenticator sends the *success* message. However at anytime if the proof isn't valid the authenticator must send a *failure* message.

Method *Type* and *Sub-Type*

Our EAP method is identified by the *type* number 84. The message pairs of the EAP method are identified by the *sub-type* field within the *type data*.

- Setup (Sub Type 1)
- Verification (Sub Type 2)

Key-Stretching Method

Our 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, without the need for discovery.

3.2.1 Setup Message Pair

In our architecture 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. The response message additionally contains the control value y_1, used to verify the proof in the fist verification round. The data is already included in this message to improve the method performance.

We are assuming that in a system with multiple users, the authenticator has identified the peer with the *identity* message.

Request

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

Data Format

Length (Octets)	...	1	$4 \leq k \leq 255$	$64 \leq j$
Field Type	...	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.

Semiprime 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 secret w using the password p the salt s with the pre-determined hashing function H.

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

Secret value 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 compute the control value $y = u^2 \pmod{n}$. The control value y is included in the response message.

Response

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.

Data Format

Length (Octets)	...	k
Field Type	...	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 Handler The authenticator should store the y_1 control value locally to be used when verifying the proof z_1.

3.2.2 Verification Message Pair

The purpose of this message pair is to exchange data required to compute and verify the proof. It is continuously exchanged until the authentication is concluded. 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

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

Data Format

Length (Octets)	...	1
Field Type	...	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

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

Data Format

Length (Octets)	...	1	k	j
Field Type	...	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.

Bibliography

[1] Bernard Aboba, Larry Blunk, John Vollbrecht, James Carlson, Henrik Levkowetz, et al. Extensible authentication protocol (EAP). 2004.

[2] Bernard Aboba and Pat Calhoun. Radius (remote authentication dial in user service) support for extensible authentication protocol (EAP). Technical report, RFC 3579, September, 2003.

[3] George E Andrews. *Number theory.* Courier Corporation, 1994.

[4] Dirk Balfanz, Brad Hill, and Jeff Hodges. Fido uaf protocol specification v1. 0. 2013.

[5] Florent Bersani and Hannes Tschofenig. The eap-psk protocol: A pre-shared key extensible authentication protocol (eap) method. Technical report, RFC 4764, January, 2007.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

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

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] J Carlson, B Aboba, and H Haverinen. Eap srp-sha1 authentication protocol (draft-ietf-pppext-eap-srp-03. txt). *Network Working Group, Internet Draft*, 135:136--137.

[17] Jyh-Cheng Chen and Yu-Ping Wang. Extensible authentication protocol (eap) and ieee 802.1 x: tutorial and empirical experience. *IEEE communications magazine*, 43(12):supl--26, 2005.

[18] Paul Congdon, Bernard Aboba, Andrew Smith, Glen Zorn, and John Roese. Ieee 802.1 x remote authentication dial in user service (radius) usage guidelines. *RFC*, 3580:1--30, 2003.

[19] 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.

[20] Stephen A Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, pages 151--158, 1971.

[21] Federal Financial Institutions Examination Council. Authentication in an internet banking environment. *FFIEC gencies (August 2001 Guidance)*, 2005.

[22] ECB ECB. Recommendations for the security of internet payments. Technical report, Tech. Rep. January, 2013.

[23] Carl Friedrich Gauss. *Disquisitiones arithmeticae.* 1801.

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

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

[26] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to prove all np-statements in zero-knowledge, and a methodology of cryptographic protocol design. volume 263, pages 171--185, 08 1986.

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

[28] 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.

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

[30] Neil Haller, Craig Metz, Phil Nesser, and Mike Straw. A one-time password system. *Network Working Group Request for Comments*, 2289, 1998.

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

[32] Taylor Hornby. Salted password hashing-doing it right. *Code Project: For those who code*, 2016. .

[33] Burt Kaliski. Pkcs# 5: Password-based cryptography specification version 2.0. *RFC 2898*, 2000.

[34] Robert McMillan. The world's first computer password? it was useless too. 2012. https://www.wired.com/2012/01/computer-password/.

[35] Lalla Mouatadid. Introduction to complexity theory: 3-colouring is np-complete.

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

[37] Colin Percival and Simon Josefsson. The scrypt password-based key derivation function. *IETF Draft URL: http://tools. ietf. org/html/josefsson-scrypt-kdf-00. txt (accessed: 30.11. 2012)*, 2016.

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

[39] Jean-Jacques Quisquater, Myriam Quisquater, Muriel Quisquater, Michaël Quisquater, Louis Guillou, Marie Guillou, Gaïd Guillou, Anna Guillou, Gwenolé Guillou, Soazig Guillou, and Thomas Berson. How to explain zero-knowledge protocols to your children. pages 628--631, 08 1989.

[40] Ronald Rivest and S Dusse. The md5 message-digest algorithm, 1992.

[41] 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.

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

[43] R Schaeffer. National information assurance (ia) glossary. *CNSS Secretariat, NSA, Ft. Meade*, 2010.

[44] Robert Shirey. Internet security glossary, version 2. Technical report, RFC 4949, August, 2007.

[45] Dan Simon, Bernard Aboba, Ryan Hurst, et al. The eap-tls authentication protocol. *RFC 5216*, 2008.

[46] William Simpson. *RFC1661: the point-to-point protocol (PPP)*. RFC Editor, 1994.

[47] William Simpson et al. Ppp challenge handshake authentication protocol (chap), 1996.

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