Password-Based Authentication with Zero-Knowledge Proof of Quadratic Residuosity

Jakob Povsic

2020

Contents

Abstract

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

Chapter 1

Introduction

1.1 Introduction

Authentication is a core component of computer security and an indispensable part of our modern digital lives. In this thesis we design an authentication protocol using Zero-Knowledge Proofs (ZKPs), an interesting cryptographic phenomenon that reveal nothing more than their validity of its proofs. Our protocol enables network authentication using a username and password. To create a secure password authentication protocol we have to be aware of the common pitfalls and how modern security systems handle them. We use the Extensible Authentication Protocol (EAP) as the framework on top of which we design our authentication protocol.

Chapter 2

Methodologies and Tools

2.1 Authentication

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

In computer science authentication is used for establishing access between restricted system resources and users through digital identities. Government and international institutions have developed guidelines for managing digital identities and authentication processes [23] .

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

2.1.1 Authentication Process Components

Authentications is a method used in information security to manage access between restricted system resources and an external user or system wishing to access them.

As defined in RFC-4949 [35], authentication is *the process of verifying a claim that a system entity or system 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*.

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/object can be implicit from the user context. 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.

2.1.2 Authentication Factors

As described in [17] authentication systems can rely on three distinct ”factors”.

- **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 [34, 18], secure authentication is based on two or more authentication factors. This is also referred to as *multi-factor authentication*.

2.2 Password Authentication

Passwords are one of the most common and oldest forms of end user authentication, being first used in computers at MIT in the mid-60s [28].

We need to understand the high level model of authentication model of password authentication, and solutions to overcome their weaknesses.

2.2.1 Authentication Model

Password based authentication is a simple authentication model, based on a shared secret between a user and a system. The secret (password) is often used in a combination with a user ID. The password itself is usually a set of characters or words memorised by the user, and inputted via a keyboard.

To authenticate, the user exchanges the password with the system via a keyboard interface, and the system authenticates the user according to passwords validity.

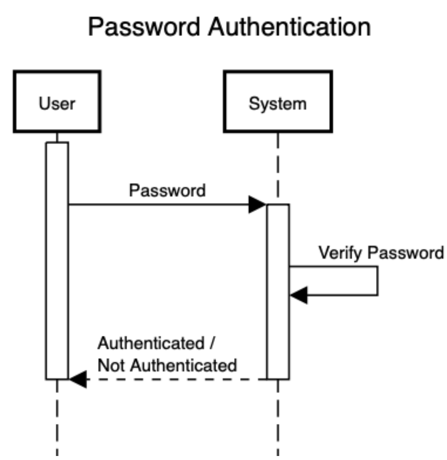

Figure 2.1: Password Authentication Model

2.2.2 Security

Security of any authentication system is defined by is security against many attack vectors, these can be split into two groups in relation to our authentication method. General attack vectors exist in any authentication system, regardless of the specific authentication mechanism, these include (but not limited to) insecure network communication, compromised host systems, phishing, etc. For example, an authentication system would be insecure if an attacker could eavesdrop to the communication between the user and the system, or a malicious program on the clients system could intercept the password before being sent, a user could compromised his password by mistakingly sending it to a phishing website.

We are going to assume participating systems and network are secure agains general threats and are going to focus on attack vectors specific to a common password authentication system implementation.

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 according to an internal mechanism and responds.

Vulnerabilities

The simplicity that makes passwords practical for users is what makes them especially vulnerable for systems that use them.

Because passwords are supposed to be memorised and the proliferation of different websites requiring them, users tend to pick password that are easier to remember and reuse passwords across different websites [15]. Many websites also don't properly handle and store passwords or are insecure in some other ways, enabling attackers to steal users plain-text passwords when a security breach happens.

Attacks can be according to NIST [23] classified as *online* or *offline*, based on wether the attacker is directly interacting with an authentication system.

Online Attacks A form of an *active attack*, where an attacker is attempting bypass authentication by directly interacting with the system. These attacks are usually very *noisy*, making it easy for an authentication system to detect an attack is happening, and prevent it. This makes online attacks much less effective than offline ones.

For example, with a simple brute-force method an attacker can try random passwords to authenticate into a single user account. An authentication system can detect an attack is happening because of the high number of failed authentication attempts, or some other security triggers. If the system knows it's under attack it can react to it making it ineffective.

Real life methods of online attacks rely on operating under the radar of detection, for example, by trying out a small number of passwords on each user. Such popular methods are *password spraying* and *credential stuffing*, 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 common passwords. *Credential stuffing* is taking a leaked user credential, for example a username and password combination found in a data breach, and using them 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 other websites.

Offline Attacks A form of a passive attack 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 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 factors, that influence the time required to guess the password.

Security Practices

To protect ourselves against vulnerabilities of passwords, we can use practices that improve our security. We can split the practices into technical tools for handling and storing passwords, and behavioural incentives to improve the password strength.

Password Handling and Storage Describes how passwords are stored at rest and used in the verification process.

A naive system might store the passwords or password-equivalent data in plain text and compare them for verification. While simple, this system is insecure as user credentials directly are exposed with any unauthorised access.

A common approach today is to use methods of *password hashing* to derive a password digest that is then stored in the database. When verifying the password is hashes again, and the digests are compared.

Using pure hashing functions like SHA family is discouraged because they designed to run fast and can be accelerated with ASIC chips, making them vulnerable to pre-computed hash tables. A better solution are password-hashing functions. Algorithms utilising hash functions designed with the purpose of being both time-consuming and memory-hard, examples of such tools are Argon2 [5], Scrypt [30] and Balloon [6]. Using an extra value called *salt* [25] prevents attacks with pre-computed hash tables. Because salt is stored alongside password hashes, systems sometimes also utilise a third value called *pepper*, which is the same for all passwords, but stored in a different place from the salt.

Authentication with Password-Hashing Functions In traditional password authentication systems, the password p is hashed with a salt s, the password hash p_H and salt s are stored, while the password p can be discarded.

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

When authenticating the user, the system hashes the submitted password p' with the stored salt s, and compares the hash p'_H with the stored hash p_H to know if the password is correct.

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

Password Strength Measure of information entropy and the difficulty of the password being guesses or brute-forced. Re-using passwords greatly undermines password strength and is what attacks like credential stuffing rely on. Have I Been Pwned [27] catalogs 613,584,248 passwords recovered from data breaches, while CrackStation [26] lists a collection of 1,493,677,782 words used for password cracking.

Systems sometime enforce or encourage users to choose stronger passwords by requiring uncommon characters like digits, symbols or different capitalisations.

2.3 Extensible Authentication Protocol

Our protocol should be implemented using the EAP framework, lets overview how the protocol works and how we were to implemented a new EAP method.

2.3.1 Overview

Extensible Authentication Protocol [1] (EAP) is a general purpose authentication framework, designed for network access authentication, where IP might not be available. It runs directly over the data link layer such as PPP [37] and IEEE 802.

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. In the protocol the peer is authenticating with the authenticator.

2.3.2 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.

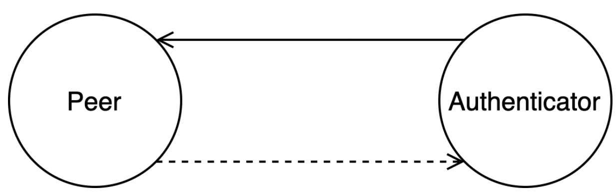

Figure 2.2: Peer and Authenticator Communication

EAP Message Structure

Messages are composed of fields, each field length is multiple of an octet of bits. Each field type has a special purpose in EAP.

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

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

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

Failure *Code 4*. Sent by the authenticator, if the peer cannot be authenticated. The peer doesn't have to 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
Type Field	...	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. In IANA [33] 49 authentication methods have been assigned type numbers. The original RFC [1] already assigned 3 authentication protocols.

MD5-Challenge *Type 4*. An EAP implementation of the [36] PPP-CHAP protocol.

One-Time Password *Type 5*. An EAP implementation of the [24] one-time password system.

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

Some other notable examples are EAP-TLS [36], EAP-PSK [4]. EAP SRP-SHA1 [12] is especially interesting as it uses a zero-knowledge protocols to verify the peers secret.

2.3.3 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 [14].

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, 13, 14]

2.4 Zero-Knowledge Proofs

In our protocol we wish to use zero-knowledge proofs as a core tool to verify users password.

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

2.4.1 Introduction

Zero-Knowledge Proofs (ZKPs) are a concept in the field of cryptography, and can be used to prove the validity of mathematical statements. What makes ZKPs particularly interesting is that it can achieve that without revealing any information about why a statement is true.

In mathematics, traditional 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 being *false* is next to zero. 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 [**?**] in 1985. They proposed a proof system as a two-party protocol between a *prover* and a *verifier*. It relies on the computational difficulty of the quadratic residuosity problem.

The Strange Cave of Ali Baba

To help our understanding we will explore the [31] 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"*.

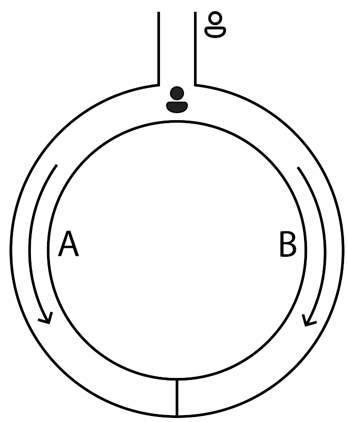

Figure 2.3: The Strange Cave of Ali Baba

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.

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 do this they come up with a scheme. Victor stands in front of cave and faces away from the entrance, to not see Peggy. 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 didn't 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 50%. If Victor were to repeat the process n time, her chances of fooling him become arbitrarily small (2^{-n}).

With this process Victor can be convinced that Peggy really knows the secret with a very chance $(1-2^{-n})$.

Further more any third party observing the interaction cannot be convinced of the validity of the proof because it 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 digital identity spaces.

The cryptocurrency Zcash uses a *non-interactive zero-knowledge protocol* zk-SNARK [7] to prove the validity of transactions, without revealing anything about the recipients nor the amount sent.

The cryptocurrency Monero uses a ZKP protocol Bulletproofs [8], to achieve anonymous transactions.

Idemix [11] an anonymous credential system for interaction between digital identities relies on CL-signatures [10] to prove ownership of a credential offline, without the issuing organisation. Idemix has been implemented in the open-source Hyperledger Indy project.

2.4.2 Interactive Proof Systems

An interactive proof system is a proof system in which a *prover* attempts to convince a *verifier* that a statement is true. The prover and the verifier exchange messages until the verifier is convinced of the proof or not.

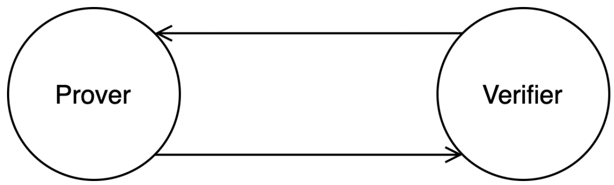

Figure 2.4: Interactive Proof System

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 witness y for statement x.

$V(y) = 1$: verifier V, analyses witness y and outputs 1 for valid witness or 0.

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 Knowledge Complexity

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 what the verifier *sees* is indistinguishable from what can be easily *simulated* on public inputs. The term *knowledge complexity* quantifies the degrees of indistinguishability of different languages and proof constructions.

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"* without provers secret information. For example, if we return to our analogy in the introduction. Victor wants to "cheat" and 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 Larry and Monica decide to record a similar scheme without knowing the secret. Larry records himself calling the tunnels and Monica emerging randomly, sometimes she emerges from the correct one other times she doesn't. Larry later edits the video to only show the times Monica correctly emerged from the tunnel, as if she knew the secret. Assuming Larry's video editing skills are good, the videos Larry and Victor recorded are indistinguishable, both videos feature someone calling tunnels and a person emerging. While one video records a valid proof and the other one doesn't, there is no information in them from which we could learn that.

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 $U = \{U(x)\}$ and $V = \{V(x)\}$ 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" data $M(x)$ that is indistinguishable from some data $U(x)$.

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 level of knowledge complexity 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. Because the verifier is formalised as a probabilistic Turing machine, the term *view* is used to describe data available to the verifier, as in what the verifier sees on its tapes. The verifiers view doesn't contain only data of the current interaction with the prover, but might contain data from past interactions with the prover (in the case of a cheating verifier).

Strengths of Zero-Knowledge

There are three levels of zero-knowledge, defined by the strength of indistinguishability. We will defined the levels of 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. Two factors determine the strength of indistinguishability, the time available to the judge and the size of the sample available to study. $V(x)$ represents the verifiers view and $S(x)$ the generated data by the simulator S.

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 Languages with Zero-Knowledge Interactive Proof Systems

We have explored in abstract terms what defines interactive proof systems and their knowledge complexity. But what are concrete examples of zero-knowledge proof systems and what can have a zero-knowledge proof system. The determining factor of wether a zero-knowledge proof system exists or not is the *problem* or *language* the proof is for. The underlying language also defines the applicability of the protocol, simpler ZKPs are used to prove knowledge of a secret, while advanced ZKPs are used to prove signatures over hidden values [8, 10, 7], set membership or range proofs [9].

The core of our protocol is based on the ZKP of quadratic residuosity as presented in [22]. We dive deep into how and why the protocol works, by exploring the mathematical foundation of quadratic residues, what is the quadratic residuosity problem and its cryptographic applications.

We also look at examples other languages with zero-knowledge proof systems and more broadly at complexity classes of languages with zero-knowledge proof systems.

2.5.1 ZKP of Quadratic Residuosity Problem

The original paper [22] on ZKPs, presented a zero-knowledge proof protocol for the *Quadratic Residuosity Problem*. Quadratic residuosity problem has a *perfect* zero-knowledge proof system. Let's explore the mathematical fundamentals, used them to describe the quadratic residuosity problem and finally see how this is used to create a ZKP system. The quadratic residuosity problem is much older than ZKPs, it was first described by Gauss in 1801 [19].

Quadratic Residues

[3] Quadratic residues come from modular arithmetic, a branch of number theory.

For $a, n \in \mathbb{Z}$, $n > 0$, $\gcd(a, n) = 1$. a is a *quadratic residue* if $\exists x : x^2 \equiv a \pmod{n}$, otherwise a 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, a is a quadratic residue mod n, if and only if.

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

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

If p is an odd prime then,

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

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

For example

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 $\left(\dfrac{a}{m}\right)$, 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 $\left(\frac{a}{n}\right) = 1$, a is a quadratic residue only if a is a quadratic residue of every prime factor of $n = p_1 p_2 \cdots p_n$.

Prime Factorization

[3] The *Fundamental Theorem of Arithmetic* states that for each integer $n > 1$, exist primes $p_1 \leq p_2 \leq \cdots \leq p_r$, such that $n = p_1 p_2 \cdots p_r$.

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

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. It is in class *NP*, but is not proven to be NP-complete. The hardest instance of this problem are when factoring the product of two prime numbers (*semiprimes*). The difficulty of this problem is a core building block in modern asymmetric cryptography like RSA [32].

Quadratic Residuosity Problem

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

The *law of quadratic reciprocity* enables us to efficiently compute the Jacobi Symbol $\left(\frac{a}{n}\right)$.
However if the computed $\left(\frac{a}{n}\right) = 1$, it does not necessarily tell if a is a quadratic residue modulo n or not, a is only a quadratic residue if a is a quadratic residue of both modulo p and q. To calculate this we would have to know the primes p and q by factoring n. However since n is a semiprime, we know this is an exceptionally difficult task.

Zero-Knowledge Proof of Quadratic Residuosity

In the original paper [22] on zero-knowledge proofs, the problem of quadratic residuosity was used to construct a zero-knowledge proof system. The protocol is an interactive proof system in which a *prover* attempts to convince a *verifier* that an integer x is a quadratic residue modulo n. The prover attempts to proving the knowledge w, where $w^2 \equiv x \pmod{n}$.

The bottom table demonstrates the steps in the protocol, the number on the left side of each row determines the *step*. The middle space displays what information is exchanged between two parties and the direction of the exchange. Space of individual parties display computations done by each party. This notation is used in all further protocol examples.

n	Semiprime, where Jacobi $\left(\frac{x}{n}\right) = 1$
x	Public input, where $x = w^2 \pmod{n}$
w	Provers private input

	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 number 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 functions 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 verifies the proof, by checking if $z^2 = yx^b \pmod{n}$, this is possible since,

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

In each round the prover has $\frac{1}{2}$ probability of cheating, by attempting to guess the value of random bit b, to improve the strength of the proof the verification is repeated m times, for a confidence of $1 - 2^{-m}$.

Parallel Composition Zero-knowledge proof of quadratic residuosity, can be alternatively be composed in *parallel* instead of sequentially. Parallel composition is very interesting because it reduces the number of interactions between the prover and the verifier, in practical applications this improves the speed of the protocol as we are less affected by communication inefficiencies.

Only languages in *BPP* have 3-round interactive zero-knowledge proofs [20]. However the quadratic residuosity problem is not believed to be in BPP, so its parallel 3-round proof system, is assumed to have a weaker notion of zero-knowledge. Our protocol design uses a sequential proof

The QRP is not believed to be in BPP, so a parallel composition of QRP has weaker notion of zero-knowledge.

2.5.2 Computational Complexity Classes

Alongside specific problems with zero-knowledge proof systems, existence of ZKPs can be related to computational complexity classes.

This knowledge is not necessary for understanding our authentication protocol, but offers an interesting background of zero-knowledge proofs.

NP (Non-deterministic Polynomial Time)

NP is a class of problems solvable by a non-deterministic Turing machine in polynomial time. Or rather proof of any language in NP can be verified by a deterministic Turing machine in polynomial time.

Article [21] proved that every language in NP has a zero-knowledge proof system, by creating 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 [29]. According to Cook's theorem [16] SAT or its 3 literal instance 3-SAT is *NP-Complete*, and any language in $L \in NP$ can be reduced to and instance of 3-SAT. Furthermore because polynomial reductions are *transient*, any language $L \in NP$ can be reduced to an instance of 3-COL.

Bounded-Error Probabilistic Polynomial Time Languages

BBP is a class of problems that can be verified by a probabilistic Turing machine in polynomial time.

Trivially every language in BPP has a ZKP system, where the prover sends nothing to the verifier, the verifier checks the proof of $x \in L$ and outputs a the verdict.

Chapter 3

Results

3.1 Protocol Design

The main goal of our authentication protocol was to enable password authentication using zero-knowledge proof, based on the quadratic residuosity problem. The computations used to assert the zero-knowledge proof present a vulnerability when used with passwords. We extend the protocol with password-hashing tools to secure the data derived from insecure passwords. The integration of password hashing is not as trivial as it might seem because of the underlying zero-knowledge protocol, we can overcome mathematical limitations imposed by the ZKP protocol by introducing a setup phase to derive the parameters used in the ZKP protocol.

In this section we will refer to the §2.5.1 ZKP protocol of quadratic residuosity as the *original protocol* and our new protocol as the *extended protocol*.

3.1.1 Vulnerabilities

Our use case is for password authentication, which features unique vulnerabilities, resulting from properties of passwords themselves, we've explored this topic in §2.2.2. In particular the original protocol is vulnerable to offline attacks with pre-computed tables. This vulnerability is caused the equation used to derive the quadratic residue x, which we later prove as a quadratic residue with the knowledge of password w.

$$x = w^2 \pmod{n}$$

Additionally, intuitively the computation of this equation is relatively inexpensive when compared to special password-hashing function like *Argon2* [5] .

3.1.2 Limitations

The solution seems to be a password-hashing function, as we've described in §2.2.2. However we cannot simply hash the data that would be otherwise held by the verifier. Let's have a look at how the verifier verifies the proof. On the last step the verifier asserts that,

$$z^2 = yx^b$$

if we were to hash the sensitive value x,

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

to verify the proof, we would need an inverted function H^{-1}, where $H(H^{-1}(H(x))) = H(x)$,

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

but since we know that H is a one-way function, the probability of a polynomial algorithm H^{-1} to successfully compute a *pseudo-inverse* is negligibly small, for all positive integers c.

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

Additionally, even if hypothetically the function H^{-1} would succeed with probability of 1, because the function H is not injective, the *pesudo-inverse* $x' = H^{-1}(H(x))$, might not be equal to x.

3.1.3 Solution

To overcome our limitation, we will use a two-phase approach. Our extended protocol is constructed from two phases, first the *setup phase* and the *verification phase*. The purpose of the setup phase is to derive the parameters used in the verification phase of the protocol. The users password p is hashed to compute the provers private input w.

$$w = H(p,s)$$
$$x = w^2 \pmod n$$

The protocol is no longer vulnerable to offline attack with a pre-computed table, since to calculate any value x a unique salt s is required.

We are assuming the prover knows the salt s, in practice this value can be passed to the user by the verifier.

3.1.4 Protocol

The setup phase runs exactly once in the protocols, while the verification phase is repeated m times for a confidence of $1 - 2^{-m}$.

		Prover		Verifier
Setup Phase	1	$w = H(P, s)$		
Verification Phase	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 $z^2 \equiv yx^b \pmod{n}$

3.2 EAP Method

We want to encapsulate our extended zero-knowledge authentication protocol defined in §3.1 within the EAP framework §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.

3.2.1 Terminology

In terminology for describing interactive proof systems §2.4.2, we name participating parties as the *prover* and the *verifier*. The prover provides the proof or the witness and the verifier asserts its validity.

In EAP §2.3 the protocol runs between the *peer* and the *authenticator*, where the authenticator authenticates the peer based on the result of an EAP method execution.

Our extended protocol builds on the original zero-knowledge protocol §2.5.1 which is an interactive proof system. When mapping the extended protocol to an EAP framework, the *prover* becomes the *peer* and the *verifier* becomes the *authenticator*. The zero-knowledge proof provided by the peer (prover) is used to assert the validity of authentication by the authenticator (verifier).

3.2.2 Overview

To define an EAP method we need to break down our extended protocol to EAP messages representing interactions between between the prover and the verifier. Each EAP messages defines its data format, the sender and recipient processes and local state changes.

The symbols n, s, w, y, z have the same definition as in the extended protocol. Our extended protocol assumes the modulus n and salt s are known by the prover, however the EAP method needs to facilitate the discovery of this data by the peer.

The EAP method consists of two message pairs, the *setup message* pair is sent once and the *verification message* pair is sent m times. The first pair is used to facilitate the *setup phase* of the extended protocol and the second pair to facilitate the *verification phase*. It should be noted that both message pairs are not one-to-one match to the communication of the extended protocol. Doing a one-to-one match would crate three message pairs with empty spaces in some response and request data. In order to save on space we've managed to compress the extended protocol into two message pairs, by interlacing some data transfer of the *verification* phase in the response of the *setup* phase and the response of the preceding verification phase, we'll explore how exactly we achieve this when defining the data formats of each message.

The authenticator might optionally use the *identity* type message the query the identity of the peer, this might be useful to locate the unique salt belonging to the peer.

To end the method, the authenticator sends a *success* message after successfully authenticating the peer, or a *failure* message otherwise.

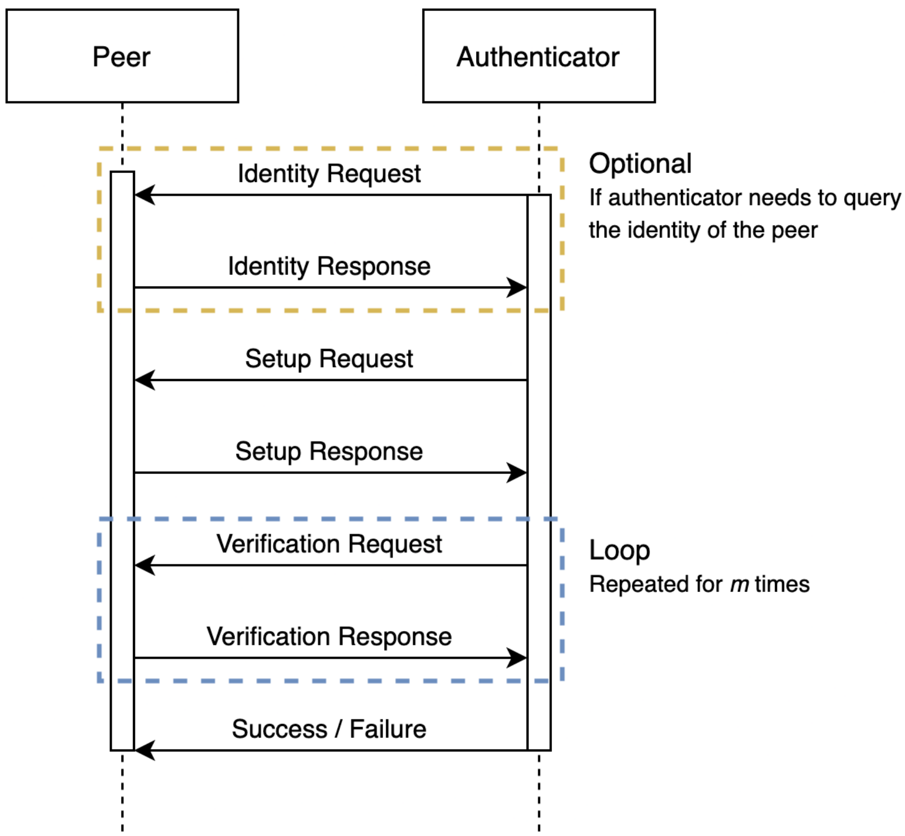

Figure 3.1: EAP Method Execution

3.2.3 EAP Message Format

Each EAP method is identified by the *type* field of the EAP message, our method is represented by the type 84. The message pair of the EAP method is identified by the *sub-type* field.

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

3.2.4 Setup Message Pair

Request

Setup request provides to the peer the salt s and semiprime modulus n.

Data Format

1	$4 \leq k \leq 255$	$64 \leq j$
Salt Length	Salt	Semiprime 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 max 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 header. Should be at least 64 octets (512 bits).

Request Handling When a request is received, the peer computes the private input w using the password p the salt s with the pre-determined hashing function H.

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

Private input w should be stored stored in memory by the peer. Next the peer should generate a random integer u from field Z_n^*, and store it in memory. The peer should compute the control value y.

$$y = u^2 \pmod{n}$$

The control value y is sent in the *setup response* data.

In order to locate the unique salt in a system with multiple peer identities, the authenticator can optionally use the *identity* method (type 1) to query the identity of the peer.

Response

Data Format

k
Control Value y

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

Response Handler The authenticator should store the y control value locally to be used when verifying the proof.

3.2.5 Verification Message Pair

This message pair is exchanged repeatedly until the authentication is concluded. The message exchange is iterated for m times to reach the confidence of $1 - 2^{-m}$.

To make our method more efficient, we reduce the number of exchanged messages between the parties, by interlacing some data between iterations. This means that on round i, the response contains data required for round $i+1$.

Request

The authenticator generates random bit b stores it locally, and sends it to the peer.

Data Format

1
Random Bit b

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

Request Handling The peer computes the witness $z = uw^b \pmod{n}$, according to the bit b received in the request.

Additionally the peer generates the control value y for the next $(i+1)$ round of the verification phase, it generates a random integer u_{i+1} from field Z_n^*, and store it in memory and computes the control value y_{i+1}.

$$y_{i+1} = u_{i+1}^2 \pmod{n}$$

Response

The control value z and the witness value y_{i+1} should be sent in the response data.

Data Format

1	k	j
Witness Length	Witness z	Control Value y_{i+1}

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

Witness Value z computed by the peer, used by the authenticator to assert the proof.

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 $z^2 \equiv yx^b \pmod{n}$. If the assertion fails the a *failure* message must be sent to the peer, otherwise a *success* message must be sent if the peer was successfully verified m times. If that is not the case, the y_{i+1} is stored by the authenticator and a new random bit b is send to the peer in the message request.

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

[5] 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.

[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] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential trans-

actions and more. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 315–334. IEEE, 2018.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

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

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

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

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

[21] 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.

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

[23] 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.

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

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

[26] Taylor Hornby. Crackstation's password cracking dictionary. 2019. .

[27] Troy Hunt. Have i been pwned. *Last retrieved*, 2021.

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

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

[30] 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.

[31] Jean-Jacques Quisquater, Myriam Quisquater, Muriel Quisquater, Michal Quisquater, Louis Guillou, Marie Guillou, Gad Guillou, Anna Guillou, Gwenol Guillou, Soazig Guillou, and Thomas Berson. How to explain zero-knowledge protocols to your children. pages 628–631, 08 1989.

[32] 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.

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

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

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

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

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