Marcio Ricardo Rosemberg

SRAP - A New Authentication Protocol for Semantic Web Applications

Dissertação de Mestrado

Dissertation presented to the Programa de Pós-Graduação em Informática of the Departamento de Informática, PUC-Rio as partial fulfillment of the requirements for the degree of Mestre em Informática.

Advisor: Prof. Marcus Vinícius Soledade Poggi de Aragão
Co-Advisor: Prof. Daniel Schwabe

Rio de Janeiro
June 2014

Marcio Ricardo Rosemberg

SRAP – A New Authentication Protocol for Semantic Web Applications

Dissertation presented to the Programa de Pós-Graduação em Informática of the Departamento de Informática, PUC-Rio as partial fulfillment of the requirements for the degree of Mestre em Informática. Approved by the following commission.

Prof. Marcus Vinícius Soledade Poggi de Aragão
Advisor
Departamento de Informática – PUC-Rio

Prof. Daniel Schwabe
Co-Advisor
Departamento de Informática – PUC-Rio

Prof. Ricardo Dahab
UNICAMP

Prof. Sérgio Lifschitz
Departamento de Informática – PUC-Rio

Prof. José Eugenio Leal
Coordinator of the Centro Técnico Científico – PUC-Rio

Rio de Janeiro, June 16th, 2014

Marcio Ricardo Rosemberg

BSc. Electrical Engineering (Universidade Santa Úrsula) – 1990. Partner and founder of SYSNET Sistemas e Redes – 1994, a company that specializes in networking, network security, and development of custom software. The author has been asked several times to provide consulting as an expert for judges on several lawsuits.

Rosemberg, Marcio Ricardo

SRAP – a new authentication protocol for semantic Web applications / Marcio Ricardo Rosemberg; advisor: Marcus Vinicius Soledade Poggi de Aragão; co-advisor: Daniel Schwabe. – 2014.
103 f : il. (color.) ; 30 cm

Dissertação (mestrado)–Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática, 2014.
Inclui bibliografia

1. Informática – Teses. 2. Web semântica. 3. Autenticação. 4. Criptografia. 5. Controle de acesso. I. Aragão, Marcus Vinicius Soledade Poggi de. II. Schwabe, Daniel. III. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. IV. Título.

CDD: 004

This work is dedicated to my grandmother, Fany Bass Rosemberg Z’L.
Thank you for everything, Faigel.

Acknowledgments

To my advisor, Professor Marcus Poggi and to my co-advisor Professor Daniel Schwabe for their support, patience and teachings.

To Professor Ricardo Dahab for his expert collaboration and for his acceptance to be a member of the jury.

To Professor Sérgio Lifschitz for his teachings and encouragement.

To the other professors of the Departamento de Informática of PUC-Rio.

To my parents Neide e Roberto for their love and support.

To my wife Adriana for her love and incentive.

To my sister Sandra for her caring.

To my daughter Nicole, the joy of my life, who inspires and give me strength.

To my colleagues from SYSNET, especially Mr. Leandro Barreto.

To all my friends in PUC-Rio

To all other people who helped me directly or indirectly

Abstract

Rosemberg, Marcio Ricadro; de Aragão , Marcus Vinicius Soledade Poggi (advisor); Schwabe, Daniel (co-advisor). **SRAP - A New Authentication Protocol for Semantic Web Applications.** Rio de Janeiro, 2014. 103p. MSc. Dissertation – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Usually, Linked Data makes Semantic Web Applications query much more information for processing than traditional Web applications. Since not all information is public, some form of authentication may be imposed on the user. Querying data from multiple data sources might require many authentication prompts. Such time consuming operations, added to the extra amount of time a Semantic Web application needs to process the data it collects might be frustrating to the users and should be minimized. The purpose of this thesis is to analyze and compare several Semantic Web authentication techniques available, leading to the proposal of a faster and more secure authentication protocol for Semantic Web Applications.

Keywords

Semantic Web; Authentication; Cryptography; Access Control.

Resumo

Rosemberg, Marcio Ricardo; de Aragão, Marcus Vinicius Soledade; Schwabe, Daniel. **SRAP – Um Novo Protocolo para Autenticação em Aplicações Voltadas para Web Semântica.** Rio de Janeiro, 2014. 103p. Dissertação de Mestrado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Normalmente, aplicações semânticas utilizam o conceito de "*linked data*", onde é possível obter dados de diversas fontes e em múltiplos formatos. Desta forma, as aplicações semânticas processam muito mais dados do que as aplicações tradicionais. Uma vez que nem todas as informações são públicas, alguma forma de autenticação será imposta ao usuário. Consultar dados de múltiplas fontes pode requerer muitos pedidos de autenticação, normalmente através de uma combinação de conta de usuário e senha. Tais operações consomem tempo e, considerando-se o tempo extra que uma aplicação semântica leva para processar os dados coletados, pode tornar a experiência frustrante e incômoda para os usuários, devendo ser minimizado, sempre que possível. O propósito desta dissertação é o de analisar e comparar as técnicas de autenticação disponíveis para as aplicações semânticas e propor um protocolo mais rápido e mais seguro para autenticação em aplicações semânticas.

Palavras-chave

Web Semântica; Autenticação; Criptografia; Controle de Acesso.

Contents

List of Figures

List of Tables

1
Introduction

The traditional Web based applications focus on dissemination of information as their paradigm. The Semantic Web focuses on dissemination of knowledge instead. In order to achieve such a goal, Semantic Web applications rely on machine readable data descriptions based on vocabularies, ontologies and Linked Data [1].

Machine readable vocabularies give computers the ability to interpret the information, filtering it based on the semantic specified by the user through domain models.

Linked Data [2] gives a Semantic Web application the ability to query data from multiple sources in a form that is transparent to the user. Linked data works *in tandem* with machine-readable vocabularies.

A central concept in the Semantic Web is the Uniform Resource Identifier (URI) [3]. URIs allows the unique identification of objects, object properties, location of files and other resources. Anything can be represented by a URI. Besides, URIs give us the ability to represent and store data in the form of a graph. Graphs can be explored and navigated. Irrelevant information can be filtered in the navigation process.

Semantic Web uses semi structured data [4] to present information. One of the advantages of using semi structured data and URIs is that it allows us to integrate multiple sources of data, since schema data is also encoded as data.

Since not all published information is public, some form of authentication is required. Because Linked Data queries information from multiple sources, at first, multiple authentications may be imposed on the user. Such an imposition would frustrate the user adding even more time to the processing of his/hers requests.

1.1.
The problem

Authentication is necessary whenever the information or knowledge that is being disclosed is sensitive. Before a server or a resource provider presents sensitive information, such as users' personal information, or if a user puts

his/hers credit card information in a web form, the endpoints involved need to be confident they are really communicating with who they are supposed to be.

Authentication provides identity confirmation and servers as a basis to establish trust. Since information is a major asset to any business or organization, it is paramount to ensure integrity and safety of the data such an organization receives and stores. Authentication helps an organization to identify whom they are dealing with and that the information received is trustworthy. The same is true on the side of the end user. The end user wants to be sure he/she can trust the web site or the server, before revealing classified or sensitive information. This is true with any type of application, including Semantic Web ones.

Since Semantic Web applications may query information from many independent sources and users may have different identities for different resources, the entire authentication process might consume a long time.

The application must be capable of correctly authenticating itself with the right identity for the right resource provider automatically whenever possible.

The financial cost to deploy and maintain an authentication protocol must not exceed the cost of data loss.

Most importantly: the user's credentials must not be copied or tampered by attackers during the authentication process.

1.2. The State of the Art

The Transport Layer Security (TLS) [I] protocol is currently the state of the art in authentication and confidentiality. It is mature, widely used by social networks, webmail providers, financial organizations and government institutions. TLS supports digital certificates for client authentication although it is seldom used, mostly because of the monetary cost to acquire digital certificates.

The WebID [II] protocol, which uses TLS as part of its solution, allows self-issued client certificates and uses URIs to uniquely identify users but the computational cost is even higher than TLS and it is not as safe as TLS.

I Transport Layer Security – RFC 5246

II Former Foaf + SSL – W3C

The idea of using URIs to identify users has some advantages. Besides uniquely identifying a user, it also provides an address where it is possible to fetch user data that can be used to prove the user's identity.

For authentication purposes, combining semi structured data already present in Semantic Web and in digital certificates is also a good idea, because we can extend vocabularies. We can put more data for authentication purposes, such as an encrypted biometric template; new encryption algorithms may be written and added to the supported encryption algorithm set; specialized required hardware for authentication, such as a smartcard reader, can be integrated with the authentication process by adding more properties to existing vocabularies. If necessary, an organization can create its own authentication scheme different from any standard. In addition, the extended vocabularies can provide the basis for more semantic authentication policies, such as "authorize anyone who knows person X and has been admitted before date D".

1.3. Motivation

Most frequently, the authentication process involves a username (user account) / password combination and many times a single user has several different user accounts and passwords, which require the user to memorize (and later forget), write on a piece of paper or store such sensitive information on a "presumed" safe media.

In order to circumvent this problem, Single Sign On, which is the property of a user to present his credentials only once and gain access to all systems or information he or she has adequate (See 3.10), techniques and protocols, such as OpenID (See 5.1) and OAuth (See 5.2), were been developed, to diminish the need to type user accounts frequently. However, they are vulnerable to social engineering attacks and they do not completely solve the problem of the need for multiple identities.

Digital certificates provide an easier and safer way for authentication. If the user has multiple certificates installed, a simple prompt to choose one of the certificates for authentication would be enough to complete the process.

The problem with digital certificates is that they are expensive for an end user to obtain and maintain, they have a limited lifetime and the computational costs

are high, which becomes an issue for mobile devices. Because of these problems, digital certificates for client authentication are not often used.

Digital certificates use the X.509 standard. The X.509 standard, like Semantic Web metadata, is composed of semi structured data [5]. Semi structured data helps the authentication process, because it gives options, such as a collection of supported authentication protocols, minimum encryption key size, the digital signature algorithm used to validate the certificate, the certificate valid time frame the parameters of the public key (each protocol has a different set of parameters) and more.

The motivation for this work is the proposal of a new authentication protocol with the following requirements:

- A protocol that could be as difficult to break as TLS
- A protocol with little dependencies on Certificate Authorities (CAs) (Parties should be able to issue self-signed certificates without compromising security)
- A protocol with active trusted third party participation in order to make active attacks more difficult to perform, meaning that the trusted third party is actively contacted in order to authenticate the parties.
- A protocol capable of providing Single Sign On even in non-federated networks, eliminating multiple user / password prompts.
- A protocol faster (less computational cost) than TLS
- A protocol with low financial cost to deploy

1.4. Goals and Contributions

The purpose of this work is the analysis and comparison of Semantic Web authentication techniques available and a proposal of a fast and reliable technique for Semantic Web.

2
Foundations

2.1.
Information Security Fundamentals

According to International Standards (ISO/IEC 17799:2005) the key concepts in information security [6] are:

2.1.1.
Confidentiality

Property preventing the disclosure of information to unauthorized individuals or systems. Semantic applications query data from multiple sources and different formats. Besides, Linked Data exploration may direct the application to sensitive data, not available to the general public. Such data may require the application to provide a secure communications channel or to receive data encrypted at the source. In the latter case, the application would only be able to make use of the information with the proper decryption key

2.1.2.
Integrity

Property that maintains and assures the accuracy and consistency of the information over its entire life cycle (birth, maintenance and destruction). The integrity property must guarantee that the information has not been modified while in transit from the server to the application and vice-versa. The network environment between servers and applications must provide the means to ensure data integrity while in transit. In the Semantic Web domain, Integrity also helps to enforce Trust. How can an application trust data whose integrity cannot be verified?

2.1.3. Availability

This property guarantees that the information in always available to the authorized users. In the Semantic Web domain, availability is paramount. Because of linked data exploration, large volumes of data come from different locations and resources. If availability is not guaranteed, the application will have slow response times or it won't be able to retrieve all the data required, resulting in frustration for the users. Semantic Web applications usually cache large amounts of remote data, the same way a proxy server does, to improve availability. Some applications download entire databases to a local repository to improve availability and performance.

2.1.4. Authenticity

Complementing the Integrity property, authenticity must provide a way to check if the information is genuine. It is also important for authenticity to validate that both parties involved are who they claim to be. This is accomplished by encryption, digital signature algorithms which form a digital certificate. In the Semantic Web domain, where authenticity check may be required on multiple sources, authenticity must be done in a very quick way. If it takes too long to check the authenticity of the parties and the information, the availability property would be compromised. Authenticity is also connected to trust. How can applications trust information whose genuineness cannot be verified?

2.1.5. Non-repudiation

In law, non-repudiation implies one's intention to fulfill his or hers obligations in a contract. It also implies that one party of a transaction cannot deny having received a transaction request nor can the other party deny having sent a transaction request [7].

It is important to note that while technology, such as cryptographic systems, can assist in non-repudiation efforts; the concept is, at its core, a legal concept transcending the realm of technology. It is not, for instance, sufficient to show that

the message matches a digital signature signed with the sender's private key, and thus only the sender could have sent the message and nobody else could have altered it in transit. The alleged sender could in return demonstrate that the digital signature algorithm is vulnerable or flawed, or allege and prove that his signing key has been compromised. The fault for these violations may or may not lie with the sender himself, and such assertions may or may not relieve the sender of liability, but the assertion would invalidate the claim that the signature necessarily proves authenticity and integrity and thus prevents repudiation.

2.2. Access Control

Access control is the selective restriction of access to resources. The act of accessing may mean consuming, inspecting, or using. Permission to access a resource is called authorization [8]. Access Control is the combination of a set of permissions, usually called Access Control Lists (ACL) with the act of authentication. The most secure systems are the ones that authenticate users enforcing the triad [9]: **What you know**, **What you have** and **Who you are**.

Access Control Lists are implemented based on the following strategies [10]: Discretionary Access Control (DAC), Mandatory Access Control (MAC) and **Role-Based Access Control** (RBAC)

3
Cryptography and Attacks on Authentication

The word Cryptography comes from the Greek words kryptós (hidden) and gráphein (to write) [11]. Cryptography is used to protect sensitive or secret data in a way that un-authorized people or computerized systems are unable to understand or make use of the data. In Computer Science, Cryptography works in 3 steps [12]:

Encryption: the process to cipher the original message. The message could be plain text, an image, a stream of bits, voice data or any form binary data. The Ciphered message C is obtained by applying the function E on the original message M.

$$C = E(M)$$

- Transmission of the ciphered message
- Decryption: the process to decipher the ciphered message back to the original message. The original message M is obtaining by applying the function D on the ciphered message C

$$M = D(C)$$

Figure 1 - Encrypting and Decrypting

Cryptography performs major roles in information security. It helps to enforce Confidentiality, Integrity, Authenticity and Non-Repudiation.

The Mechanics of Cryptography involves a sender (Alice), a receiver (Bob) and sometimes a trusted third party (Trent) [13]. There are cryptographic algorithms that involve more parties.

It is important to define two other characters involved in attacks, meaning unauthorized people trying to gain access to private, sensitive or secret

information. These characters are: The eavesdropper (Eve) and the malicious active attacker (Mallory). We must always assume that the Eve has the ability to monitor any messages transmitted by Alice and Bob. Eve is the passive attacker.

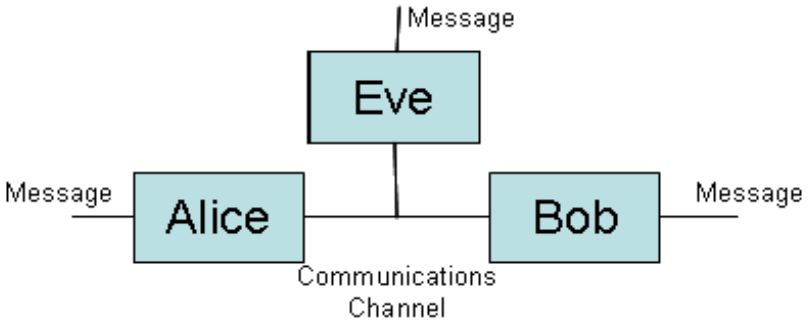

Figure 2 – The eavesdropper

Active attacks, on the other hand, can have much more diverse objectives. The attacker could be interested in obtaining information, degrading system performance, corrupting existing information, or gaining unauthorized access to resources. Active attacks are much more serious, especially in protocols in which the different parties don't necessarily trust one another. The attacker does not have to be a complete outsider. She/he could be a legitimate system user or the system administrator. There could even be many active attackers working together. Here, the role of the malicious active attacker will be played by Mallory. It is also possible that the attacker could be one of the parties involved in the protocol. He may lie during the protocol or not follow the protocol at all. This type of attacker is called a cheater. Passive cheaters follow the protocol, but try to obtain more information than the protocol intends them to. Active cheaters disrupt the protocol in progress in an attempt to cheat [14].

In order to ensure communications privacy, active attacks must be mitigated. One of the most difficult types of attack to mitigate is the Man in the middle attack. In this type of attack, Mallory disrupt the communications channel, positioning himself between Alice and Bob in a way they fail to notice him and think they still have a direct link between themselves.

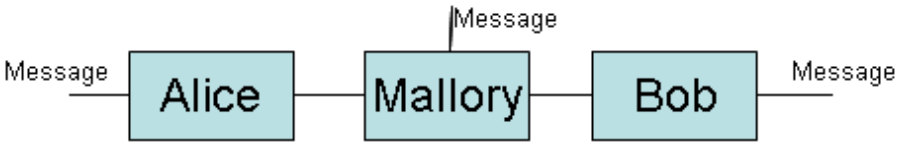

Figure 3 – Man in the middle attack

3.1.
Algorithms and Keys

Since Encryption and Decryption are functions, they are based on an algorithm. If the security is based on the algorithm, then the algorithm must be kept secret at all costs. If it leaks, everybody that uses the algorithm needs to change it. The solution to this problem is the use of public but strong cryptographic algorithms that use one or more keys to encrypt and decrypt messages. If the keys are compromised, the parties involved just need to change the keys, maintaining the algorithm. When an algorithm uses keys, the encryption and decryption functions are expressed $C=E_K(M)$ and $M=D_K(C)$, respectively, such that $D_K(E_K(M))=M$ holds [15]. Security is based in the size and complexity of the key (the longer and the more complex the better) and the complexity of the algorithm (usually, the more complex the better). Complexity of the algorithm increases the difficulty to write another algorithm capable of decrypting the ciphered message or capable of deducing the encryption key. Complexity of the key increases the difficulty to guess the key in a brute force attack.´

3.2.
Symmetric Algorithms

Algorithms that use the same key to encrypt and decrypt messages are called symmetric algorithms. Alice and Bob must agree on a single encryption and decryption key which would be use by both [16].

The problem with symmetric algorithms is how Alice and Bob negotiate a session key (a symmetric key used in one communications session) in an unsecured channel. Unless they agree to meet in person and negotiate the key, there's always the possibility Eve listens to the key negotiation and renders the encryption process useless. On the other hand, Mallory can do much worst. Mallory can negotiate a session key with Alice and another with Bob. Then, he can decrypt Alice's message, forge another message and send it to Bob. Bob thinks he received an authentic message from Alice and Alice doesn't know Bob received a false message.

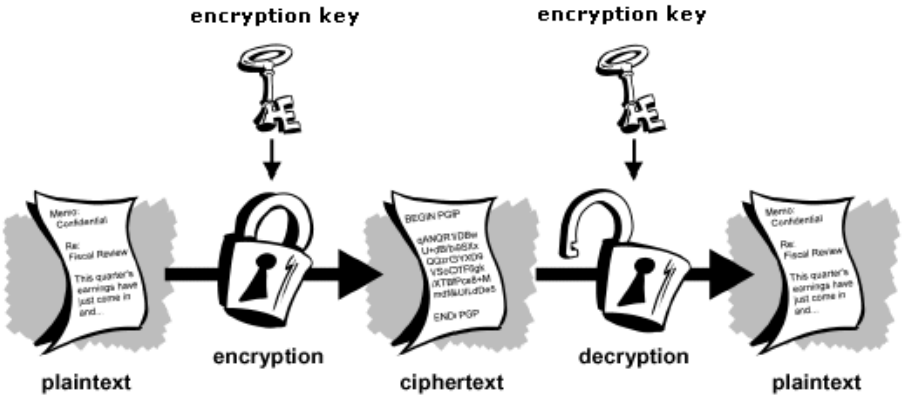

Figure 4 – Symmetric Encryption

3.3. Asymmetric Algorithms

Algorithms that use different encryption and decryption keys such that even in possession of one of the keys one cannot calculate the second in a reasonable amount of time are called asymmetric algorithms. One of the keys is the public key that can be widely distributed. The other key is the private key known only by its owner. Messages encrypted with the public key (Puk) can only be decrypted by the private key (PrK) and vice-versa.

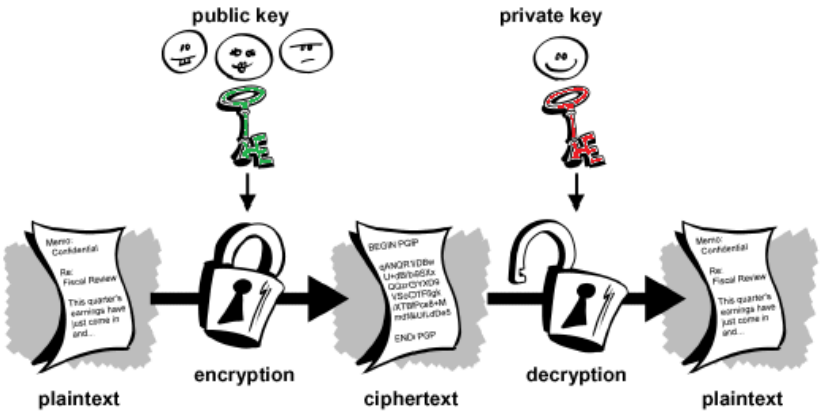

Figure 5 – Asymmetric Encryption

3.4. Digital Signatures

A digital signature is an algorithm designed to validate the authenticity of a digital message. A valid digital signature gives the recipient reason to believe the

message the have not been modified while in transit, enforcing Integrity. If the digital signature is bound to an unique person or organization the recipient has reason to believe the message was created by a known sender, such that the sender cannot deny having sent the message, enforcing both non-repudiation and authentication. One way hash algorithms is a good way to provide integrity. If Alice sends a message to Bob with a SHA-128 hash attached to the message and the message is tampered while in transit Bob will calculate the SHA-128 hash of the received message and it will not match the SHA-128 hash supplied by Alice. However, a one-way hash algorithm does not enforce non-repudiation or authentication. If Alice generates a SHA-128 hash and encrypts the SHA-128 hash with her private key, Alice provides authentication and non repudiation, because the hash can only be decrypted with Alice's public key and compared with a new hash computed, using the received message. If they don't match, either the message lost integrity during transmission or it was tampered by an attacker. However, if they match indeed, the message is authentic and non-repudiation is assured, because the hash signed with Alice's private key can only be correctly verified by using Alice's public key [17].

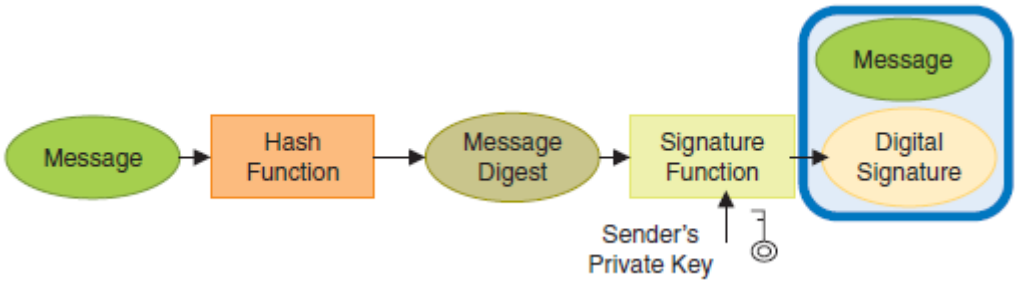

Figure 6 – Creating a digital signature

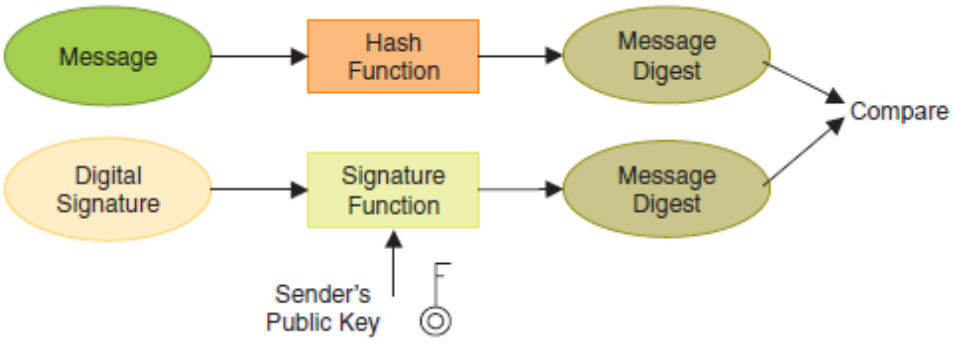

Figure 7 – Verifying a digital signature

3.5.
The Diffie-Hellman Key Exchange Cryptosystem

The Diffie-Hellman Key Exchange Cryptosystem (DHKX) [III] allows two parties that have no prior knowledge of each other to jointly establish a shared secret key over an insecure communications channel. The shared secret key may then be used to encrypt messages between the two parties, using a symmetric key encryption algorithm.

The cryptosystem is based on the discrete logarithm problem [18]. The two parties agree on a large prime number **p** and a generator **g**. RFC-2409 enumerates safe prime numbers and generators that can be used for the DHKX protocol.

Each participant chooses a number (most times a random number) **X** less than **p** to be the private key. If the participants are Alice and Bob, **Xa** is Alice's private key and **Xb** is Bob's.

Alice calculates her public key $Ya = g^{(Xa)} \bmod p$ (1)

Bob calculates his public key $Yb = g^{(Xb)} \bmod p$ (2)

They transmit their public keys to each other.

Alice calculates the session key $Ka = Yb^{(Xa)} \bmod p$

Bob calculates his public key $Kb = Ya^{(Xb)} \bmod p$

The keys Ka and Kb match because what Alice and Bob are really calculating is $K = g^{(XaXb)} \bmod p$, but without revealing their private keys to each other.

The proof:

$K = Yb^{(Xa)} \bmod p$

$K = (g^{(Xb)} \bmod p)^{(Xa)} \bmod p$ replacing Yb with (2)

$K = g^{(XaXb)} \bmod p$ by the rules of modular arithmetic

$K = (g^{(Xa)} \bmod p)^{(Xb)} \bmod p$

$K = Ya^{(Xb)} \bmod p$ replacing (1) with Ya

The security of the Diffie-Hellman key exchange lies in the fact that, while it is relatively easy to calculate exponentials modulo a prime, it is very difficult to

III DIFFIE, Whitfield; HELLMAN Martin. 1976

calculate discrete logarithms. For large primes, the latter task is considered infeasible [19].

3.6. The RSA Cryptosystem

The RSA cryptosystem publish by Ron Rivest, Adi Shamir and Len Adleman in 1978 that can be used for confidentiality, authenticity and non-repudiation [20] [21].

Alice's public and private keys are generated as follows:

First Alice picks up two prime numbers **p** and **q**.

She calculates **n** = **p** x **q** and ϕ**(n)** = (**p**-1)x(**q**-1)

She, then chooses a number **e** relatively prime to ϕ**(n)**

The public key is **n** and **e**

The private key **d** is calculated such that **de** $\equiv$ 1 mod ϕ**(n)**.

After the calculations, p and q must be safely discarded.

To encrypt a message M, M is encrypted in blocks, with each block having a binary value less than **n**.

The ciphered message C is calculated using the public key. $C = M^{\mathbf{e}} \bmod \mathbf{n}$

The original message M is calculated using the private key. $M = C^{\mathbf{d}} \bmod \mathbf{n}$

To digitally sign a message, guaranteeing authentication and non-repudiation, Alice uses her private key to encrypt a hash H such that H = h(M) where h is a hash function.

The digital signature of M is $DS = H^{\mathbf{d}} \bmod \mathbf{n}$ (encryption with the private key)

The original hash can only be restored using the public key. $H = DS^{\mathbf{e}} \bmod \mathbf{n}$.

In order to verify that the message is authentic, that it has not been modified during transmission (integrity) and that Alice cannot say she didn't send it (non-repudiation), the receiver generates the hash of the received message and compares it with the hash decrypted from the digital signature, using Alice's public key. If the hashes match, the message is authentic.

The security of the RSA cryptosystem lies in the fact that it is very difficult to factor a large compound integer number in its prime factors (The Integer Factorization Problem).

3.7. Attacks on Authentication Protocols

Authentication protocols should be resilient to the following attacks [22]:

3.7.1. Eavesdropping

Eavesdropping is perhaps the most basic attack on a protocol. Nearly all protocols address eavesdropping by using encryption. It is obvious that encryption must be used to protect confidential information such as session keys. In certain protocols there may be other information that also needs to be protected. An interesting example is that protocols for key establishment in mobile communications usually demand that the identity of the mobile station remain confidential. Eavesdropping is sometimes distinguished as being a passive attack since it does not require the adversary to disturb the communications of legitimate principals. The other attacks we consider all require the adversary to be active. It should be remembered that many sophisticated attacks include eavesdropping of protocol runs as an essential part.

3.7.2. Modification

If any protocol message field is not redundant then modification of it is a potential attack. Use of cryptographic integrity mechanisms is therefore pervasive in protocols for authentication and key establishment.

Whole messages, as well as individual message fields, are vulnerable to modification. Many attacks do not alter any known message field at all, but split and re-assemble fields from different messages. This means the integrity measures must cover all parts of the message that must be kept together; encryption of these fields is not enough.

3.7.3. Replay

Replay attacks include any situation where the adversary interferes with a protocol run by insertion of a message, or part of a message, that has been sent previously in any protocol run. We may regard replay as another fundamental type of attack which is often used in combination with other attack elements. Just as almost all protocols address eavesdropping and modification attacks by using cryptography, almost all protocols include elements to address possible replay attacks.

It is possible for the replayed message in an attack to have been originally part of a protocol run that happened in the past. Alternatively the replayed material may be from a protocol run that takes place at the same time as the attacking run.

3.7.4. Preplay

Preplay might be regarded as a natural extension of replay, although it is not clear that this is really an attack that can be useful on its own. It differs from Replay, because the attacker prepares the attack in advance, carrying out a false authentication process with the initiating party while pretending to be the destination party. Phishing Attacks [34] are a form of Preplay Attack.

3.7.5. Reflection

Reflection is really an important special case of replay. A typical scenario is where two principals engage in a shared key protocol and one simply returns a challenge that is intended for itself. This attack may only be possible if parallel

runs of the same protocol are allowed but this is often a realistic assumption. For example, if one principal is an Internet host, it may accept sessions from multiple principals while using the same identity and set of cryptographic keys. The possibility of instigating several protocol runs simultaneously is another common and realistic strategy for the adversary.

3.7.6. Denial of Service

In a denial of service attack (often contracted to DoS attack) the adversary prevents legitimate users from completing the protocol. Denial of service attacks in practice take place against servers who are required to interact with many clients. Attacks can be divided into those that aim to use up the computational resources of the server (resource depletion attacks) and those that aim to exhaust the number of allowed connections to the server (connection depletion attacks).

As a matter of principle it seems that it is impossible to prevent denial of service attacks completely. Any attempt to establish a connection must either result in allocation of a connection or use some computational work to establish that the attempt is invalid. Nevertheless there are certain measures that may be taken to reduce the impact of denial of service attacks and some protocols are much more vulnerable to this sort of attack than others, so it is important not to ignore this issue.

3.7.7. Typing Attacks

When a protocol is written on the page its elements are clearly distinct. But in practice a principal receiving a message, whether encrypted or not, simply sees a string of bits which have to be interpreted. Typing attacks exploit this by making a recipient misinterpret a message, accepting one protocol element as another one (that is, a message element of a different type). For example, an element which was intended as a principal identifier could be accepted as a key. Such an attack typically works through a replay of a previous message.

3.7.8. Cryptanalysis

Cryptographic algorithms used in protocols are often treated abstractly and considered immune to cryptanalysis. However, there are some exceptions that should be mentioned. The most important exception is when it is known that a key is weak and is (relatively) easy to guess once sufficient evidence in available. This 'evidence' will normally be a pair of values, one of which is a function of the key; examples are a plaintext value and the corresponding cipher text, or a plaintext value and it's MAC [IV].

The most common example of use of a weak key is when the key is formed from a password that needs to be remembered by a human. In this situation the effective key length can be estimated from the set of values that are practically used as passwords, and is certainly much smaller than would be acceptable as the key length of any modern cryptosystem. A number of protocols have been designed specifically to hide the evidence needed to guess at weak keys.

3.7.9. Certificate Manipulation

In public key protocols the certificate of a principal acts as an off-line assurance from a trusted authority that the principal's public key really does belong to that principal. Other principals who make use of a certificate are trusting that the authority has correctly identified the owner of the public key at the time that the certificate was issued. However, it is not necessarily expected that the authority is provided with evidence that the corresponding private key is actually held by the principal claiming ownership of the key pair. This leads to potential attacks in which the adversary gains a certificate that a public key is its own, even though it does not know the corresponding private key. By choosing the public key to be a function of an existing public key some undesirable consequences may arise.

IV Message Authentication Code

3.7.10. Protocol Interaction

Most long-term keys are intended to be used for a single protocol. However, it could be the case that keys are used in multiple protocols. This could be due to careless design, but may be deliberate in cases where devices with small storage capability are used for multiple applications (smart cards are the obvious example).

It is easy to see that protocols designed independently may interact badly. For example, a protocol that uses decryption to prove possession of an authenticating key may be used by an adversary to decrypt messages from another protocol if the same key is used. Kelsey et al. [V] give several examples of how things can go wrong, and discuss the chosen protocol attack in which a new protocol is designed by the adversary to attack an existing protocol. Apart from limiting keys to be used in unique protocols, one method to prevent such attacks is to include the protocol details (such as unique identifier and version number) in an authenticated part of the protocol messages.

3.8. Eliminating the Eavesdropper, providing authentication and Non-repudiation

Although the Diffie-Hellman Key Exchange eliminates the eavesdropper, it does not provide authentication and non-repudiation, since the private keys are randomly selected on each authentication. One solution to completely eliminate Eve's efforts to gain access to Alice's messages to Bob and providing authentication and Non-repudiation using the RSA cryptosystem is specified below:

- Alice sends her public key to Bob and asks for his.

[V] John Kelsey, Bruce Schneier, and David Wagner. Protocol interactions and the chosen protocol attack. In B. Christianson et al., editors, Security Protocols - 5th International Workshop, pages 91-104. Springer-Verlag, 1998. Lecture Notes in Computer Science Volume 1361.

- Bob acknowledges Alice's request by sending his public key

- Alice generates a one-way hash of the message she intends to send to Bob and signs it with her private key, creating the digital signature.

- Alice encrypts the both the message and the digital signature with Bob's public key, creating the ciphered message, and sends the entire content to Bob.

- Although Eve has Alice's and Bob's public keys and is capable of copying the ciphered message, she does not know Bob's private key, hence she cannot decrypt the ciphered message.

- Bob receives the ciphered message and decrypts it with his private key. He, then, verifies the digital signature, using Alice's public key and the hash of the received message. Because the digital signature can only be verified using Alice's public key, the solution guarantees, in respect to Alice and Eve's perspective, authentication, integrity, confidentiality and non-repudiation. It is up to Bob to provide availability.

However, from Mallory's perspective, the scheme presented will work only if Alice and Bob have previous knowledge of each others public keys, because Mallory has the ability to intercept Alice's and Bob's public keys and send his public key to Alice and to Bob. By doing so, he can decrypt Alice's ciphered message with his private key, tamper with the message, generate a new hash, sign the tampered message with his private key, create a new ciphered message with Bob's public key and finally send the ciphered message to Bob. If Bob does not have any means of detecting that Alice's public key has "changed", he will think he has received a genuine message form Alice. In order to mitigate Mallory's efforts, a third party, trusted by both Alice and Bob (Trent) must be brought in to validate Alice's and Bob's identities. Trent is referred to as a Certification Authority in the Public Key Infrastructure (PKI) concept.

3.9.
Public Key Infrastructure (PKI)

Public-key infrastructure is a cryptographic technique that enables users to securely communicate on an insecure public network, and reliably verify the identity of a user via digital signatures. [23] It is based on the use of asymmetric encryption algorithms.

A public key certificate (also known as a digital certificate or identity certificate) is an electronic document, using the X.509 standard that uses a digital signature to bind a public key with identity information such as the name of a person or an organization, their address, and so forth. The certificate can be used to verify that a public key belongs to an individual. [24]

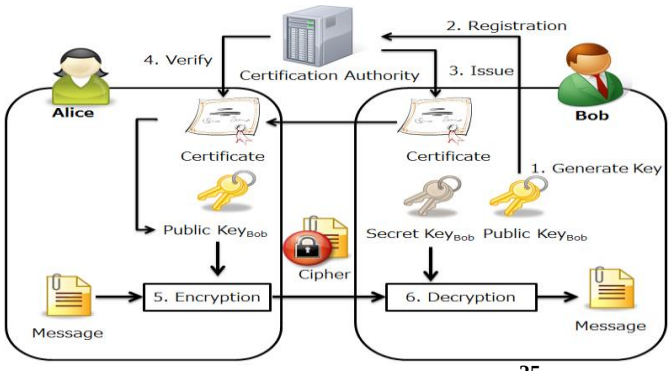

Figure 8 – Public Key Infrastructure [25]

The steps defined in the figure above are:

1. Public and private keys generation. Bob sends his personal data (name, country, email and other relevant information) to the CA. The CA uses an algorithm to generate the key pair.

2. Bob's information and the public key are registered in a certificate database

3. The CA digitally signs the certificate with its self issued private key. A digital certificate containing Bob's information, Bob's public key and the CA digital signature is issued. Bob receives the certificate and his private key. The private key must be kept secret at all costs, while the

certificate can be widely distributed. In the example, Bob sends his certificate to Alice.

4. Alice verifies the authenticity of Bob's certificate with the CA public key extracted from the CA digital certificate which must be installed on Alice's computer prior to the verification process of Bob's certificate.

5. Alice uses Bob's public key (contained in the certificate) to encrypt the message

6. Bob uses his private key to decrypt Alice's encrypted message.

A PKI consists of [26]:

- A CA that both issues and verifies the digital certificates.
- A registration authority (RA) which verifies the identity of users requesting information from the CA
- A central directory—i.e. a secure location in which to store and index keys.
- A certificate management system
- A certificate policy

It is important to mention that a CA issues certificates for itself. By installing the CA digital certificate on Alice and Bob computers they expressively declare that they trust the CA. In a PKI scenario, Alice would exchange her digital certificates which will be verified by the CA. Once she encrypts a message with Bob's verified public key, only Bob would be able to decrypt de ciphered message.

Now, Mallory has a big problem, because it would be very difficult for him to pose as an authentic CA and RA to both Alice and Bob. Difficult, but not

impossible. Once Mallory deploys his private PKI, he could deceive Alice and Bob by employing a strategy such as this:

He must find a way to have his private certification authority certificate installed on both Alice's and Bob's computers. By doing so he will appear as a trusted CA to Alice and Bob. If he succeeds, he can generate a false digital certificate to Alice and another to Bob; send Alice's false certificate to Bob and Bob's false certificate to Alice. Since he is positioned between them both, Alice will think she has a secure channel to Bob and vice-versa.

He should also find a way to fool Alice's and Bob's DNS Servers (DNS Spoofing) or put an entry in Alice's and Bob's HOST files, so their machines can resolve Mallory's fake full qualified domain name to Mallory's IP Address.

It is important to mention that when we install an operating system on a computer, we also install several root certification authority certificates. The same happens we install a browser that uses its own set of root certification authority certificates. If Mallory makes his own distribution of an operating system version and people download and install them on their computers or even mobile devices, they will be vulnerable to Mallory (See 5.3.1).

3.10. Single Sign On

As stated in the introduction, Single Sign On (SSO) [27] is the ability of a user to present his credentials only once and gain access to all systems or information he or she has adequate permissions without being prompted to log in again at each of these resources.

SSO shares the use of centralized authentication servers which other applications and systems use for authentication purposes. Once authenticated, the client application holds the user's credentials and presents them to any resource server that requires authentication. The resource server validates the user's credentials with the authentication servers. The first and only authentication can be a user account / password prompt, a digital certificate and a proof of possession of the corresponding private key or a biometric template.

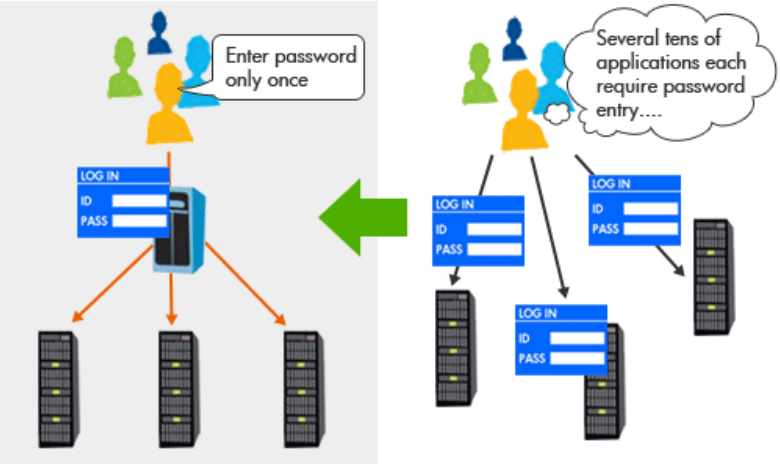

Figure 9 – Single Sign On

4
Semantic Web Concepts and Technologies

To achieve the goal of presenting structure data from non structured sources, a semantic web application makes use of several concepts and technologies, from which we highlight [28]:

- The Resource Description Framework (RDF)
- The RDF Schema (RDFS)
- The Web Ontology Language (OWL)
- The SPARQL Language
- The Linked Data Concept

4.1.
RDF

The Resource Description Framework (RDF) is a general-purpose language for representing information about resources in the Web [29]. It is particularly intended for representing metadata about Web resources, but it can also be used to represent information about objects that can be identified on the Web, even when they cannot be directly retrieved from the Web. To some extent, RDF is a lightweight ontology language designed to support interoperability between applications that exchange machine-understandable information on the Web.

The concepts of URI, URI reference, namespace, and qualified name are fundamental for structuring the Semantic Web as a distributed, federated information space, because they provide an addressing scheme that is stable, distributed, and effective.

A resource is anything that has an identity, be it a retrievable digital entity (such as an electronic document, an image, or a service), a physical entity (such as a book) or a collection of other resources.

A Uniform Resource Identifier (URI) is a character string that, uniquely, identifies an abstract or physical resource on the Web.

Examples of URIs following different URI schemes are:

- A URI following the FTP scheme for File Transfer Protocol services: ftp://ftp.mysite.com/files/foobar.txt
- A URI following the HTTP scheme for Hypertext Transfer Protocol services: http://www.mysite.com/pub/foobar.html
- A URI following the MAILTO scheme for e-mail addresses: mailto:em@w3.org

A URI reference (URIref) denotes the common usage of a URI, with an optional fragment identifier attached to it and preceded by the character "#". However, the URI that results from such a reference includes only the URI after removing the fragment identifier.

Examples of URIrefs are:

- A URIref identifying an individual: http://www.w3.org/People/EM/contact#me
- A URIref identifying a class (or type) http://www.w3.org/2000/10/swap/pim/contact#Person
- A URIref identifying a property: http://www.w3.org/2000/10/swap/pim/contact#mailbox
- A URIref identifying a property value: http://www.example.org/staffem/85741

An absolute URIref identifies a resource independently of the context in which the URIref appears. A relative URIref is a URIref with some prefix omitted; hence, information from the context in which the URIref appears is

required to fill in the omitted prefix. In particular, a relative URIref consisting of just a fragment identifier is equivalent to the URIref of the document in which it appears, with the fragment identifier appended to it. For example, the relative URIref #PrivateDoc, appearing in a document identified by the URIref http://www.cat.com/schema is considered equivalent to the URIref: http://www.cat.com/schema#PrivateDoc.

An XML namespace, or simply a namespace, is a collection of names. A namespace is identified by an URIref.

Names from namespaces may appear as qualified names (QNames) of the form P:L, containing a single colon ":", that separates the name into a namespace prefix P and a local part L. The namespace prefix must be associated with a namespace URIref N in a namespace declaration. We say that the qualified name represents the absolute URIref constructed by concatenating N and L.

An example of a namespace is RDF http://www.w3.org/1999/02/22-rdf-syntax-ns# rdf, where the namespace is RDF, the URIref is http://www.w3.org/1999/02/22-rdf-syntax-ns# and de prefix is rdf.

The QName rdf:description has namespace prefix rdf and local part description. It expands to the URIref: http://www.w3.org/1999/02/22-rdf-syntax-ns#description

An RDF statement (or simply a statement) is a triple (S, P, O), where S is a URIref, called the subject of the statement, P is a URIref, called the property (also called the predicate) of the statement, that denotes a binary relationship, and O is either a URIref or a literal, called the object of the statement; if O is a literal, then O is also called the value of the property P.

4.1.1. RDF as a Graph

The RDF triples notation translates RDF statements directly into character strings. More precisely, the RDF triple for an RDF statement (S, P, O) is a string of one of the two forms:

<S> <P> <O> . if O is an absolute or relative URIref

<S> <P> "O" . if O is a literal

The RDF triples notation for a set R of RDF statements is simply the concatenation of the RDF triples that represent each RDF statement in R, in any order. The RDF graphs notation translates a set of RDF statements into a graph,

with nodes representing subjects or objects, and arcs representing properties. More precisely, the RDF graph for a set R of RDF statements is a labeled graph where:

The set of nodes of the graph is constructed as follows:

- For each URIref U that occurs as subject or as object of an RDF statement in R, there is a node in the graph labeled with U ;
- For each literal L that occurs as object of an RDF statement in R, there is a node in the graph labeled with L ;

These are the only nodes in the graph. The set of arcs of the graph is constructed as follows:

- For each RDF statement (S, P, O) in R, there is an arc directed from the nodelabeled with S to the node labeled with O, and the arc is labeled with P ;

These are the only arcs in the graph.

Only absolute URIrefs are allowed to label nodes and arcs in RDF graphs. Furthermore, when drawing RDF graphs, nodes labeled with URIrefs are shown as ellipses, whereas nodes labeled with literals are shown as boxes.

The following figure shows a set of RDF statements and their corresponding graph [30].

Stmt	Element	Value (Absolute URIref or Literal)	Value (QName)
S1	Subject	http://www.cat.com/docs#R20301	
	Property	http://purl.org/dc/elements/1.1/creator	dc:creator
	Value	http://www.cat.com/auth#R051156	
S2	Subject	http://www.cat.com/docs#R20301	
	Property	http://purl.org/dc/elements/1.1/title	dc:title
	Value	Karin Homepage	
S3	Subject	http://www.cat.com/docs#R20301	
	Property	http://purl.org/dc/elements/1.1/date	dc:date
	Value	2021-01-20	

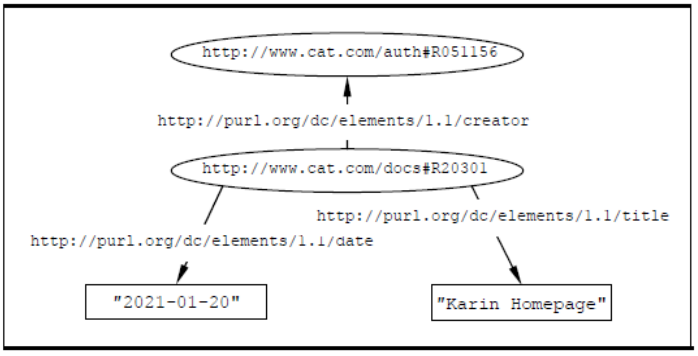

Figure 10 – RDF statements and their corresponding graph

4.2. SPARQL Language

SPARQL is a recursive acronym for SPARQL **P**rotocol **A**nd **R**DF **Q**uery **L**anguage.

SPQRQL is an RDF query language for databases, able to retrieve and manipulate data stored in RDF format.

In the case of queries that read data from the database, the SPARQL language specifies four different query variations for different purposes [31].

- SELECT query: Used to extract raw values from a SPARQL endpoint, the results are returned in a table format.
- CONSTRUCT query: Used to extract information from the SPARQL endpoint and transform the results into valid RDF.
- ASK query: Used to provide a simple True/False result for a query on a SPARQL endpoint.
- DESCRIBE query: Used to extract an RDF graph from the SPARQL endpoint, the contents of which is left to the endpoint to decide based on what the maintainer deems as useful information.

Each of these query forms takes a WHERE block to restrict the query although in the case of the DESCRIBE query the WHERE is optional.

Below, an example of a SPARQL query that retrieves all names and emails from a dataset, using the FOAF (Friend Of A Friend) ontology:

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?email
WHERE {
  ?person a foaf:Person.
  ?person foaf:name ?name.
  ?person foaf:mbox ?email.
}
```

4.3.
Linked Data

As mentioned in the introduction section, Linked Data is the ability of retrieving data form multiple sources, with possibly different formats, in a form that is transparent to the user.

Tim Berners-Lee outlined four principles of linked data in his *Design Issues: Linked Data note* [32]:

- Use URIs to denote things.
- Use HTTP URIs so that these things can be referred to and looked up ("dereferenced") by people and user agents.
- When someone looks up a URI, provide useful information, using the standards (RDF*, SPARQL).
- Include links to other URIs, so that they can discover more things.

5
Authentication Techniques

There are several products and techniques available for authentication, some of which are specific to the Semantic Web domain. This work focuses on analyzing the following products and techniques: OpenID, OAuth 2.0, TLS and WebID (formerly known as FOAF+SSL).

5.1.
OpenID

OpenID is a decentralized mechanism for single sign-on. A user does not need a password for every resource he/she needs to authenticate. An OpenID is basically a URI assigned to a unique user. The user can claim ownership of the URI and can prove that claim. Actually, a XRI (extensible resource identifier) is used to store the user's URI and additional information when required [33].

OpenID is a standard for authentication. When a server asks for the user's credentials for authentication, the user's sends his/hers URI. The URI will be validated by the server at a verification authority, the same way a digital certificate is validated by a CA. The validating server sends a XRI to the verification authority, which returns another XRI with the result of the validation.

Unlike Microsoft's Passport, no company or group owns the standard. Hence, the standard can be implemented without asking for permission. The user has the choice to pick up his/hers own OpenID provider.

The problem with OpenID is that it is vulnerable to phishing attacks. OpenID establishes a shared secret encryption key, using Diffie-Hellman key exchange protocol. Hence, Eve is not able to retrieve the URI and Mallory efforts to break a secure communications channel are difficult, although it is feasible to disrupt the protocol (See 3.4).

A classic phishing attack using e-mail occurs as follows [34]:

Step 1. The phisher sends the potential victim an e-mail that appears to be from the person's bank or other organization that would have the victim's personal information on the user. The phisher carefully uses the colors, graphics, logos and wording of the existing company.

Step 2. The potential victim reads the e-mail and takes the bait by providing the phisher with personal information by either responding to the e-mail or clicking on a legitimate-looking link and providing the information via a form on a website that appears to be from the bank or organization in question.

Step 3. This fake website or e-mail sends the victim's personal information directly to the phisher.

If the user goes to a fake website under the pretext of revalidating his/hers OpenID, the user will give away his/hers URI. Figure 6 illustrates a phishing attack.

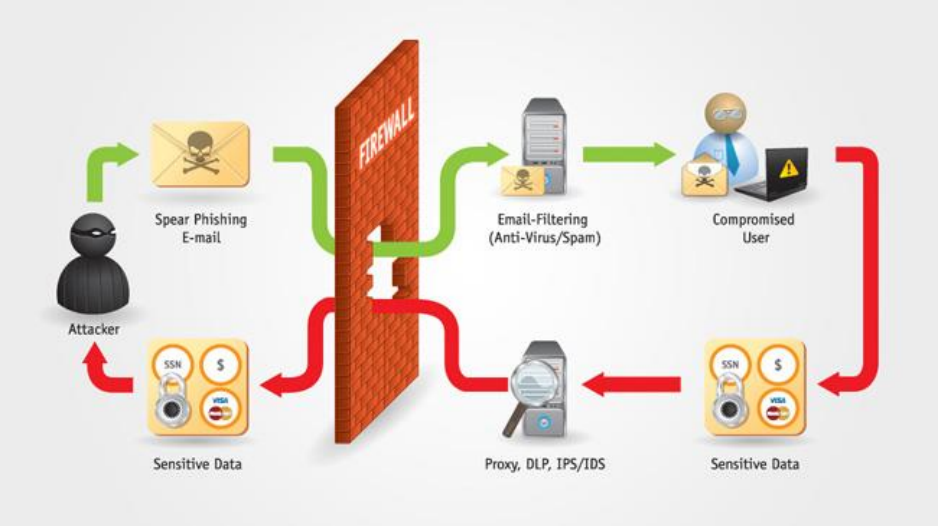

Figure 11 – Phishing Attack

On the other hand, OpenID is as safer as typing an email address and a password under an unsecure communications channel. Probably more, since the URI does not disclose password information and travels encrypted.

In regard to a Semantic Web application, OpenID is a good solution. It is light and the payload to authenticate a user is only a URI which travels encrypted by symmetric key. Unfortunately, it lacks authenticity and non-repudiation. As established, the URI may be stolen by a phishing attack. It will authenticate the real user, instead of the attacker who succeeded in stealing the real user's URI. Therefore, the real user might have just cause repudiate the misuse of his/hers URI.

5.2. OAuth 2.0

OAuth is an open protocol to allow secure authorization in a simple and standard method from web, mobile and desktop applications.

The OAuth 2.0 authorization framework enables a third-party application to obtain limited access to an HTTP service, either on behalf of a resource owner by orchestrating an approval interaction between the resource owner and the HTTP service, or by allowing the third-party application to obtain access on its own behalf [35].

OAuth implements RBAC with four defined roles [36]:

Resource owner: An entity capable of granting access to a protected resource. When the resource owner is a person, it is referred to as an end-user.

Resource server: The server hosting the protected resources, capable of accepting and responding to protected resource requests using access tokens.

Client: An application making requests on protected resource on behalf of its owner and with the owner's authorization. The term "client" does not imply any particular implementation characteristics (e.g., whether the application executes on a server, a desktop, or other devices).

Authorization server: The server issuing access tokens to the client after successfully authenticating the resource owner and obtaining authorization.

The primary objective of OAuth is to issue an authorization to an application to grant access to a resource in behalf of a user without the need of the user's password.

The protocol flow is illustrated by the figure bellow

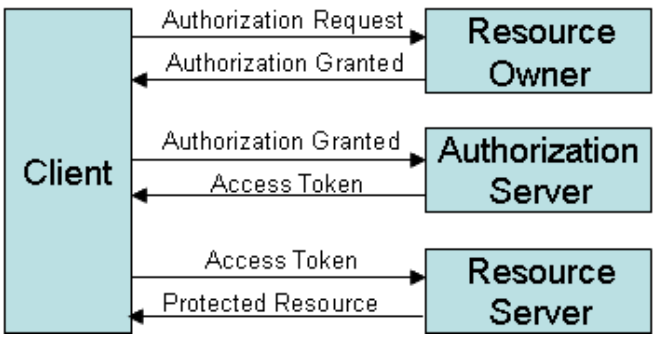

Figure 12 – Oauth 2.0 protocol flow

The Authorization Server issues temporary credentials to a protected resource, making OAuth more secure then Open ID. It is far more difficult for Mallory to disrupt the communications between the four parties involved.

Many social networks like Facebook, Google+, Microsoft Live use OAuth 2.0

The problem with OAuth is that it is, like OpenID, vulnerable to phishing attacks. Once the resource owner authorizes an access to a protected resource, it will remain authorized for the client, until the user revokes the authorization. Like OpenID, OAuth lacks a strong mechanism to ensure authenticity and non-repudiation.

In the Semantic Web domain, OAuth is slightly slower than Open ID, but is safer. Figure 13 shows a comparison between OpenID and OAuth.

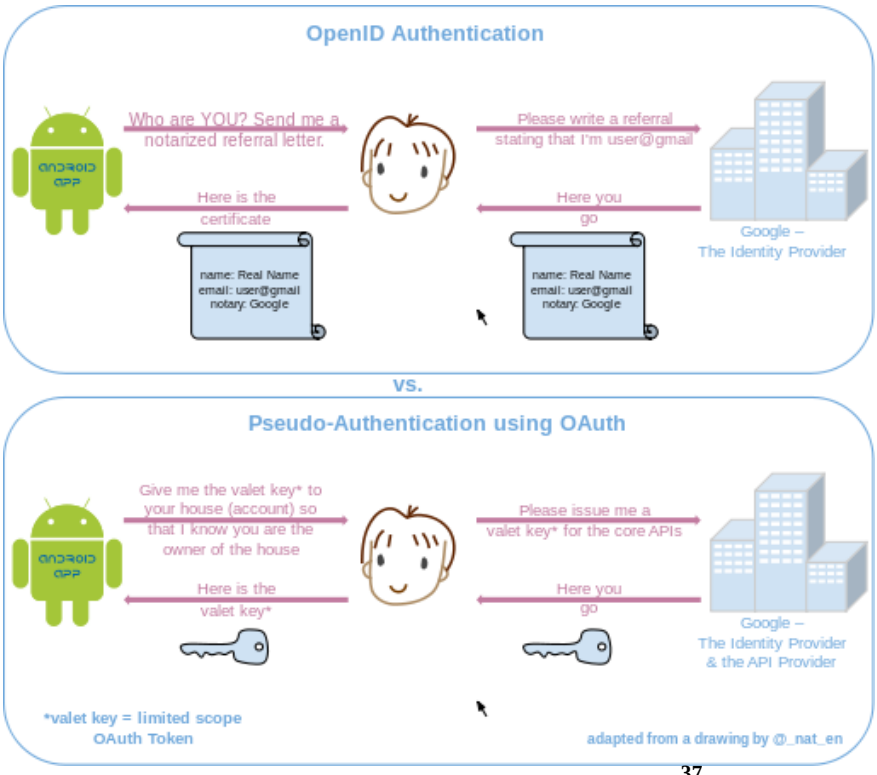

Figure 13 – OpenId and OAuth comparison [37]

Both protocols rely in an Identity Provider. The difference is that OpenID issues a certificate for every application. OAuth, however, issues a limited scope token for a single specific application instead.

5.3.
TLS (Transport Layer Security)

TLS is the pinnacle of digital authentication and confidentiality. It uses PKI, CAs and digital certificates to guarantee authenticity, confidentiality, non-repudiation and integrity. The certificates are digitally signed by a CA private key and can only be verified by the CA public key. A forged certificate would fail the verification process. Before sending any information to Bob, Alice verifies with the issuing CA public key that Bob's certificate is authentic. After verifying Bob's authenticity, Alice uses Bob's public key contained in the certificate to negotiate a session key. Only Bob can decrypt Alice's message with his private key.

TLS accepts the use of client certificates, making unnecessary to use user accounts and passwords. If Bob wants to authenticate Alice, she sends her certificate to Bob and a proof of possession of the private key. Bob verifies the authenticity of Alice's certificate and Alice's proof of possession of the corresponding private key. Only Alice can use her private key to sign a message, which is verified by Bob using Alice's public key [38] [39].

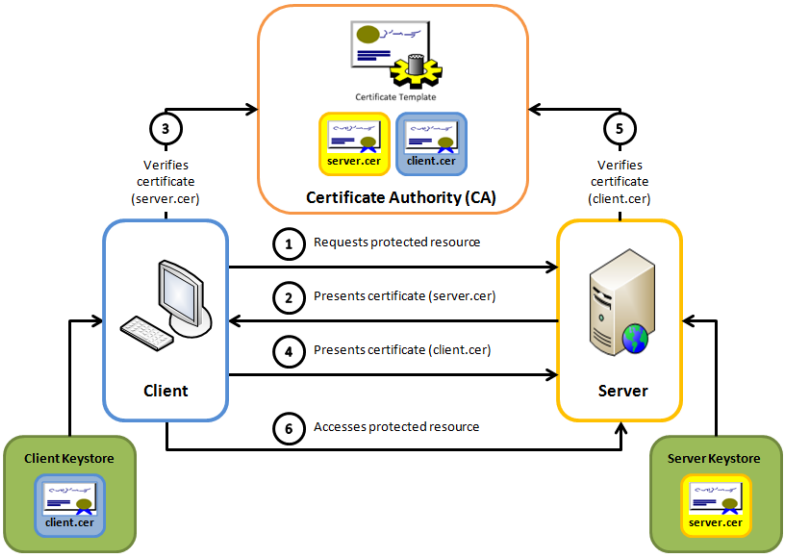

Figure 14 – Mutual TLS Authentication using client and server certificates

Web Browsers uses TLS to establish a secure channel and the HTTP protocol message exchange is encrypted with the symmetric session negotiated by TLS. This communications protocol is known as HTTPS.

To protect the root CA private key, a root CA only issues certificate for herself and for intermediate CAs. If a root CA private key is compromised, the entire certificate chain is compromised. However, if an intermediate CA private key is compromised, only a branch of the certificate chain is compromised.

Although client certificates can be used with TLS, they are seldom used, since digital certificates are expensive for users and must be renewed every year or every two years, depending on the issuing CA policy.

TLS is a very mature protocol the evolved from Secure Sockets Layer (SSL).

The problems we face with TLS are its flexibility and its implementations.

In most implementations, TLS authenticates only servers, leaving the client authentication to a user account / password prompt after the secure channel has been established.

The flexibility of TLS allows certificate chain errors to be ignored, further weakening the protocol.

Up to 2011, many libraries did not correctly check the certificate chain [40] [41]. Hence, it was possible for a user with the possession of a legitimate certificate to create another certificate, even if he or she was not an intermediate CA. The X.509 certificate has a field which specifies if the certificate was issued for an end user or for an intermediate CA. In the latter case, it also specifies the maximum number of certificates the intermediate CA is allowed to issue for other intermediate CAs.

Recently, libraries developed by the Open SSL Foundation (www.openssl.org), were shipped with the Heartbleed [42] bug. The bug is considered the most significant threat ever, allowing any ordinary user to make requests that might download recently used passwords and, in many cases, the private key of the server's digital certificate. More than half a million sites, including Facebook's, Google's and Bruce Schneier's own site [ref] were vulnerable to the bug.

Basically, the attacker would send a request to change the session password, with a 10 bytes payload, but with a 64K value in length field. The server does not check if the length field matches the actual payload length but copied and returned 64KB of memory instead. Multiple requests, would copy different 64KB memory blocks, hence the attacker would be able to copy user accounts, passwords, session keys (that could be used on a session hijack attack or to eavesdrop and

decrypt the client-server connection) and ultimately the private key of the server certificate stored in memory.

Another possible vulnerability of every PKI infrastructure is government pressure or intrusion on a CA.

Depending on the legislation of a country, a CA could be forced by a court order to revoke a digital certificate and issue another, in which the private key would be copied and delivered to law enforcement or security services institutions. Also, a CA could be invaded by hackers, by national security agencies or by foreign espionage agencies in order to copy the private key of every certificate issued by a particular compromised CA from the time of the successful invasion until the discovery of the invasion. This has happened already, at least once, on the DigiNotar[43] scandal.

5.3.1. Disrupting TLS

Because TLS uses the PKI, it is vulnerable to false root CA certificates installed on a client or server computer. The TLS protocol can be configured to ignore client certificates, accept client certificates (not mandatory) or to request client certificates (mandatory). According to RFC 5246 [VI], authentication can even be suppressed. In this mode, only the eavesdropper is mitigated.

Usually, TLS is used to authenticate the server only, because of the high financial cost to acquire digital certificates signed by trusted CAs., so typical users don't want to incur in this cost.

An organization can deploy its own PKI infrastructure (See 3.9) and issue private certificates for servers and clients, but unless the root certification authority is installed on every computer (client and server) of the organization, which is precisely what Mallory will try to do, the certificate chain verification will fail. The figures bellow shows how Mallory could disrupt TLS in a server only authentication and in a two way authentication.

VI Section 1 - Introduction

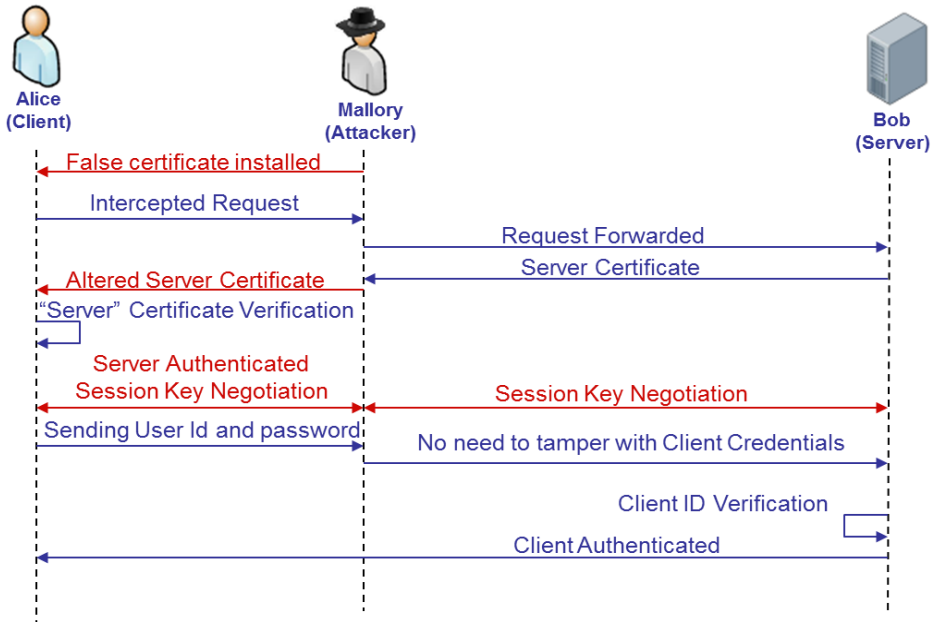

Figure 15 – Disrupting TLS in a one way authentication

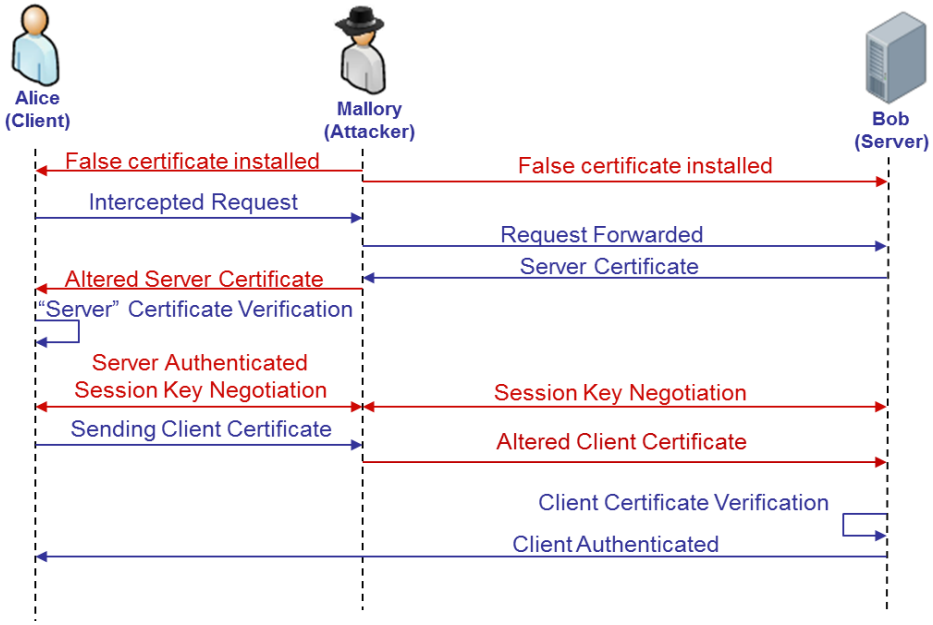

Figure 16 – Disrupting TLS in a two way authentication

It would depend on how the server uses the certificate for Mallory to be successful in disrupting a two way TLS authentication. If the server only matches the Subject structure of the X.509 certificate, the CN (Common Name) field of the Subject structure, Malory succeeds. However, if the PuK of the certificate is also matched, Mallory will not succeed. Many libraries only match the subject field, because the certificate has an expiration date. A renewed certificate would have a different PuK and, therefore, would require a new registration process. A renewed

certificate used by a system that does not match the certificate PuK with the user's credentials previously registered, would not require a new registration process.

We should notice that the Trusted Third Party (The CA) is not actively involved in the authentication process. In other words, Alice and Bob do not communicate with the CA to validate their certificates. The CA participates in issuing the certificates and signing them with its private key only.

Periodically, Alice and Bob must download certificates revocation lists (CRL). Certificates are revoked for a variety of reasons and once a certificate is revoked, it must not be accepted anymore. If Mallory is capable of performing a successful Distributed Denial of Service (DDoS) attack on a particular CA, he will halt the CRL traffic. Therefore, revoked certificates might be accepted or, since the status of the certificate cannot be verified, all authentications of a particular CA might also halt. CRL are being replaced by the Online Certificate Status Protocol (OCSP), which is lighter and more efficient the CRL, defined in RFC 6960. Therefore, the network bandwidth is used more efficiently even though it does not eliminate the risk of a DDoS attack.

HTTPS, where the TLS is most widely used, is vulnerable to 3 other types of attack [44]:

- Packet Injection Attack (could be mitigated by the application)
- Trace Attack (if server supports and enables TRACE command)
- SSLstrip Attack (downgrade from https to http attack)

5.4. WebID (formerly known as FOAF+SSL)

The WebID protocol mixes a private PKI and the FOAF ontology for authentication. Users create their own certificates, which are self-signed.

It works as follows: a user's public encryption key is stored in both a certificate and a remote RDF file stored on a Web server. Additionally, the certificate is created with the URI for the location of the remote RDF file stored in the certificate's Subject Alternative Name field. When a user sends a WebID certificate to a server, the server identifies the remote URI, pulls down the public

key from that URI, and compares it with the public key in the user's certificate. If the keys match, the user is authenticated as the entity associated with the (public key, remote URI) pair described in the certificate. The server can then decide whether or not to authorize a user based on a set of rules (e.g. an access control list) [45] [46].

By using self-signed certificates, WebID provides a trust-based user network that does not rely on any central authority; hence WebID authenticates a user in a single connection. The sequence diagram of figure 10 illustrates a successful WebID authentication:

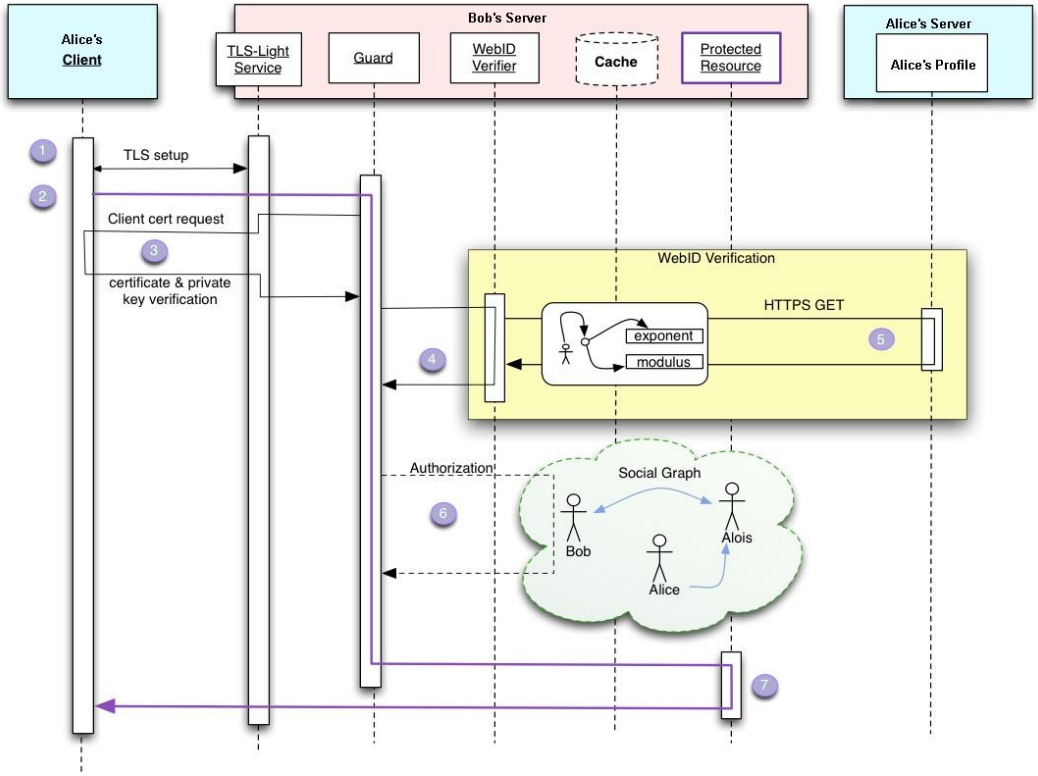

Figure 17 – WebID Authentication

The steps of the authentication process are [47]:

1) Alice's Client must open a TLS (Transport Layer Security) connection with the server which authenticates itself using well known TLS mechanisms. This may be done as the first part of an HTTPS connection (HTTP-TLS).

2) Once the TLS connection has been set up, the application protocol exchange can start. If the protocol is HTTP then the client can request an HTTP GET, PUT, POST, DELETE, or action on a resource. The Guard can then

intercept that request and by checking some access control rules determine if the client needs authentication. We will consider the case here where the client does need to be authenticated.

3) The Guard must request the client to authenticate itself using public key cryptography by signing a token with its private key and have the Client send its Certificate. This has been carefully defined in the TLS protocol and can be summarized by the following steps:

3.1) The guard requests of the TLS agent that it make a Certificate Request to the client. The TLS layer does this. Because the WebID protocol does not rely on Certificate Authorities to verify the contents of the Certificate, the TLS Agent can ask for any Certificate from the Client. More details in the step Requesting the Client Certificate (5.4.1)

3.2) The Client asks Alice to choose a certificate if the choice has not been automated. We will assume that Alice chooses a WebID Certificate and sends it to the client.

3.3) The TLS Agent must verify that the client is indeed in possession of the private key. What is important here is that the TLS Agent does not need to know the Issuer of the Certificate, or to have any trust relation with the Issuer. Indeed if the TLS Layer could verify the signature of the Issuer and trusted the statements it signed, then step 4 and 5 would not be needed - other than perhaps as a way to verify that the key was still valid.

3.4) The WebID Certificate is then passed on to the Guard with the proviso that the WebIDs still needs to be verified.

4) The Guard then must ask the Verification Agent to verify that the WebIDs do identify the agent who knows the given public key.

5) The WebID is verified by looking up the definition of the URL at its canonical location. This can be done by dereferencing it. The Verification Agent must extract the public key and all the URI entries contained in the Subject

Alternative Name extension of the WebID Certificate. A WebID Certificate may contain multiple URI entries which are considered claimed WebIDs at this point, since they have not been verified. The Verification Agent may verify as many or as few WebIDs it has time for. It may do it in parallel and asynchronously. However that is done, a claimed WebID can only be considered verified if the following steps have been accomplished successfully:

5.1) If the WebID Verifier does not have an up-to-date version of the WebID profile in the cache, then it must dereference the WebID using the canonical method for dereferencing a URL of that scheme. For an https://... WebID this would be done using the HTTP-TLS protocol.

5.2) The returned representation is then transformed into an RDF graph as specified in Processing the WebID Profile step (5.4.2).

5.3) That graph is then queried as explained in the Verifying the WebID Claim step (5.4.3). If the query succeeds, then that WebID is verified.

6) With the set of verified WebIDs the Guard can then check its access control rules using information from the web and other information available to it, to verify if the referent of the WebID is indeed allowed access to the protected resource. The exact nature of those Access Control Rules is left for another specification. Suffice it to say that it can be something as simple as a lookup in a table.

7) If access is granted, then the guard can pass on the request to the protected resource, which can then interact unimpeded with the client.

5.4.1. Requesting Client Certificates

TLS allows the server to request a Certificate from the Client using the CertificateRequest message [section 7.4.4] of TLS v1.1 [RFC5246]. Since WebID TLS authentication does not rely on CA's signing the certificate to verify the WebID Claims made therein, the Server does not need to restrict the certificate it

receives by the CA's they were signed by. It can therefore leave the certificate_authorities field blank in the request.

If the Client does not send a certificate, because either it does not have one or it does not wish to send one, other authentication procedures can be pursued at the application layer with protocols such as OpenID, OAuth, BrowserID, etc...

As far as possible it is important for the server to request the client certificate in WANT mode, not in NEED mode. If the request is made in NEED mode then connections will be broken off if the client does not send a certificate. This will break the connection at the application protocol layer, and so will lead to a very bad user experience. The server should therefore avoid doing this unless it can be confident that the client has a certificate - which it may be because the client advertised that in some other way to the server.

5.4.2. Processing the WebID Profile

The Verification Agent needs to fetch the document, if it does not have a valid one in cache. The Verification Agent must be able to process documents in RDF/XML (RDF-SYNTAX-GRAMMAR) and RDFa in XHTML (XHTML-RDFA). The result of this processing should be a graph of RDF relations that is queryable.

It is suggested that the Verification Agent should set the Accept-Header to request application/rdf+xml with a higher priority than text/html and application/xhtml+xml. The reason is that it is quite likely that many sites will produce non marked up HTML and leave the graph to the pure rdf formats.

If the Guard wishes to have the most up-to-date Profile document for an HTTPS URL, it can use the HTTP cache control headers to get the latest versions.

5.4.3. Verifying the WebID Claim

To check a WebID claim one has to find if the graph returned by the profile relates the WebID to the Certificate Public Key with the cert:key relation. In other words one has to check if those statements are present in the graph.

Verifying the WebID Claim with SPARQL:

Testing for patterns in graphs is what the SPARQL query language is designed to do (RDF-SPARQL-QUERY). We will first look at how to use this as it is also the simplest method, and then what some other programmatic options may be.

Below is the SPARQL Query Template which should be used for an RSA public key. It contains three variables ?webid, ?mod and ?exp that need to be replaced by the appropriate values:

```
PREFIX : <http://www.w3.org/ns/auth/cert#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
ASK {
   ?webid :key [
      :modulus ?mod;
      :exponent ?exp;
   ] .
 }
```

An ASK query simply returns true or false. If it returns true, then the key was found in the graph with the proper relation and the claim is verified.

Verifying the WebID claim without SPARQL:

If the RDF library does datatype normalization of all literals before loading them, then the most efficient way to execute this would be to start by searching for all triples whose subjects have relation cert:modulus to the literal which in our example was "cb24ed..."^^xsd:hexBinary. One would then iterate through all the subjects of the relations that satisfied that condition, which would most likely never number more than one, and from there filter out all those that were the object of the cert:modulus relation of the WebID - in the example Alice:me. Finally one would verify that one of the keys that had satisfied those relations also had the cert:exponent relation to the number which in the example above is "65537"^^xsd:integer.

For triples stores that do not normalize literals on loading a graph, the normalization will need to be done after the query results and before matching those with the values from the Certificate. Because one could not rely on the modulus having been normalized, one would have to start with the WebID - Alice:me and find all it's cert:key relations to objects - which we know to be keys - and then iterate through each of those keys' modulus and exponent, and verify if

the normalised version of the value of those relation is equal to the numbers found in the certificate. If one such key is found then the answer is true, otherwise the answer will be false [48].

5.4.4. WebID Access Control

When using WebID, Access Control Lists (ACLs) can also be expressed with semi structured data, employing the ACL vocabulary. Resources, users and group of users (a group can be a role) are referenced as URIs and one or more acl properties define the type of access granted to the resource [49].

Examples:

```
@prefix acl: <http://www.w3.org/ns/auth/acl#>.
@prefix foaf: <http://xmlns.com/foaf/0.1/>.

[acl:accessTo <public_resource>; acl:mode acl:Read; acl:agentClass foaf:Agent].
[acl:accessTo <protected_resource>; acl:mode acl:Read, acl:Write;  acl:agent
<user#i>].
```

Servers are required to recognize the class foaf:Agent as the class of all agents. This indicates that the given access is public. In some cases this will mean that authentication is therefore not required, and may be skipped. When a resource is being written, however, it may be necessary to associate the change with some kind of ID for accountability purposes [50].

```
[acl:accessTo <sensitive_file>; acl:mode acl:Read; acl:agentClass
<http://my.example.net/groups/friends#groupfr>].
[acl:accessTo <sensitive_file>; acl:mode acl:Read, acl:Write; acl:agentClass
<groups/family#groupfa>].
```

In the example above, the sensitive file may be read by the friends group but may be read and written by the family group.

```
<#groupfr> is rdf:type of  </user/Bob>, </user/Alice>, </user/charlie>.
<#groupfa> is rdf:type of  <../people/don>, <../people/eloise>.
```

Example of a group or a role definition

The main advantage of using the semi structured data to build ACLs is that we do not require an LPAD server nor do we need to specify ACLs on a file server. Likewise, we do not need to create multiple users, roles and rights on tables or columns on a database server. We do not have to build ACLs on every server.

Depending on the level of security required, we may build ACLs on a RDF file or semantic database server, navigate the graph, verify the type of access granted to a user for the resource servers, get the information from the resource servers with a single super user account common to all and return the processed knowledge to the client application.

5.4.5. WebId Analysis

In comparison to other Semantic Web authentication techniques, WebID is much safer than OpenID and OAuth. The client certificate ensures authenticity and a way for non repudiation. Phishing attacks are more difficult for Mallory to perpetrate, since TLS requires the server to send its certificate, which can be verified by Alice. The challenge to verify if the client has the possession of the certificate private key mitigates the theft of client certificates. Even if the certificate is stolen it cannot be used without its private key. In fact, WebID does two authentications: the first, occurring during TLS setup, verifies if the client has the private key of the certificate and the second is done by matching the certificate public key with the client's RDF file.

Client certificates are cached for a brief time. As a result, authentications of cached client certificates can be faster.

At this time, we see two potential problems with WebID:

1) If Mallory is capable of modifying Alice's RDF file, replacing Alice's public key with his own public key from a previously created WebID certificate, Bob's Server may authenticate Mallory as if he was Alice.

2) Since anyone may create a WebID certificate, Mallory does not even need a private PKI and whatever harm Mallory does, it will be logged with Alice's URI, not Mallory's, if he is able to modify Alice's RDF file.

6
Secure RDF Authentication Protocol (SRAP)

We introduce SRAP as an authentication protocol designed to minimize the use of widely trusted CAs, diminishing the financial cost of the certificate issued by CAs, particularly for client authentication. It uses self-signed certificates for clients and servers, uses the concept of the Web of Trust (WOT) [VII] and is capable of detecting if previously used public keys have changed, indicating a possible successful earlier intrusion.

Like WebID, SRAP uses semantic web ontologies into its conception. The URI is also used for unique user identification. But, whereas WebID uses a public RDF file for client authentication, SRAP uses an RDFK (encrypted and hidden) file. The location of the RDFK file is not revealed, unless an authentication is requested. The client reveals his/hers RDFK file only after a successful server authentication.

The use of a URI to uniquely identify and authenticate a user is a better choice than the use of the Common Name (CN) field of the X.509 Subject structure. The latter contains an email address, which cannot be verified during the authentication process, while the former points to a file that can be downloaded and can be used to verify both the client and the server's identities.

6.1.
Architecture of the SRAP protocol

The SRAP protocol consists in five distinctive steps:

1. Establish a presumed secure communications channel (eliminate any eavesdropper)
2. Verify the server identity (authenticate sever)
3. Verify the client identity (authenticate client)
4. Renegotiate the session key (eliminate MITM)

[VII] ZIMMERMANN, Phil. 1991

5. Certificate storage

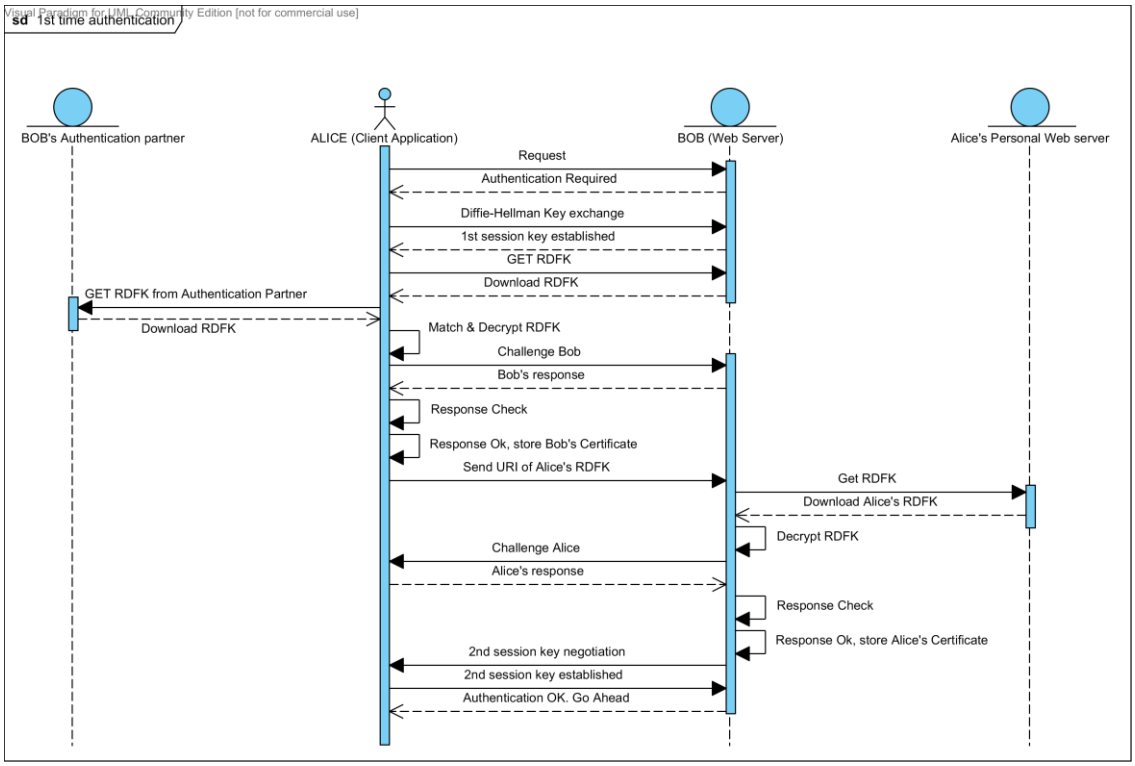

Figure 18 – SRAP Architecture

6.1.1. Step 1: Establishment of a presumed secure communications channel

As described in section 3.5, the Diffie-Hellman Key Exchange protocol allows two parties that have no prior knowledge of each other to jointly establish a shared secret key over an insecure communications channel. Any eavesdropper (Eve) is eliminated by this step [51].

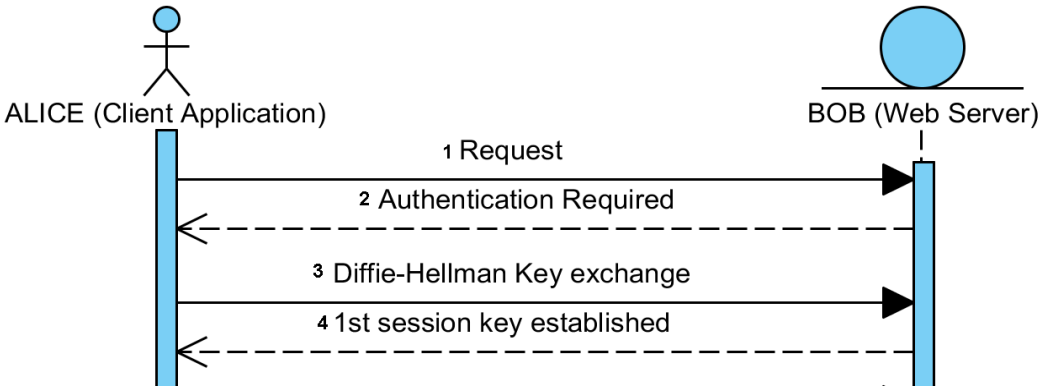

Figure 19 – Step 1 – Diffie-Hellman key exchange

Step 2: Server identity verification

Alice downloads Bob's encrypted RDF file (RDFK) containing information only for authentication purposes. The encryption uses a symmetric key algorithm (e.g.: AES-256) and a hash (e.g.: SHA-256) of the entire RDF-XML file plus the random generated symmetric key. The hash and key are encrypted with Bob's Private Key (PrK), which generates a digital signature. The verification and decryption of the symmetric key is possible only with Bob's Public key (PuK). The use of the private key to encrypt the hash and the symmetric encryption key is intended to provide non-repudiation.

Since any one could find a way to download Bob's RFDK file, Alice must verify the authenticity of Bob's RFDK. The RFDK should either contain a certificate signed by a trusted CA or, in case Bob already has a web of trust with other servers, his RFDK file must contain a list of his authentication partners. At least one of Bob's authentication partners must have a certificate signed by a CA. The rest of the partners may have self-signed certificates.

By replicating his RDFK file to other servers, Bob increases the chance Alice have already authenticated herself with one of Bob's partners. Alice downloads the RFDK file from two different locations, matches them and then proceeds with the challenge. If Alice has not authenticated herself with any of Bob's partners that use self-signed certificates, Alice will only accept a partner which uses a certificate signed by a CA. This partner is the authentication partner of last resort.

The challenge is the proof that Bob has the private key of the presented certificate. Alice generates a 256 bits long random string and a hash of the generated string, encrypts them with Bob's PuK and sends the ciphered message to Bob. He then uses his PrK to decrypt the message, calculate the hash of the decrypted random string and match it with the hash sent by Alice. Finally he returns another hash (e.g.: MD5) of the random string to Alice. Alice calculates the secondary hash and if it matches the one Bob sent her, Bob is authenticated.

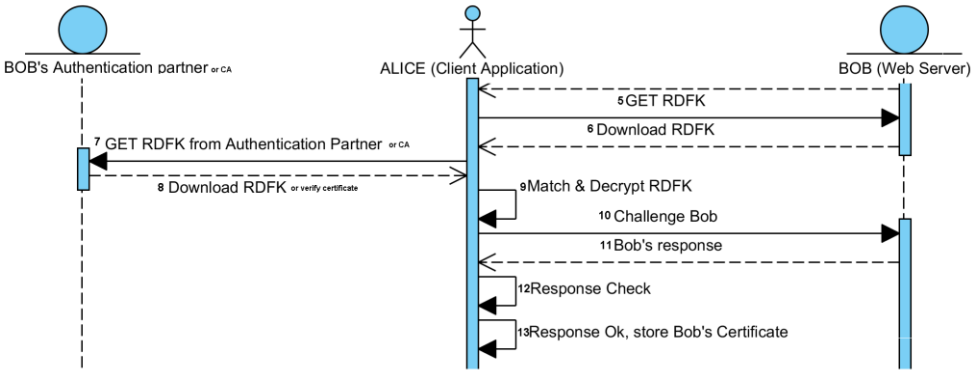

Figure 20 – Step 2 – Server authentication

The authentication partner of last resort has a different sequence as shown in the figure bellow.

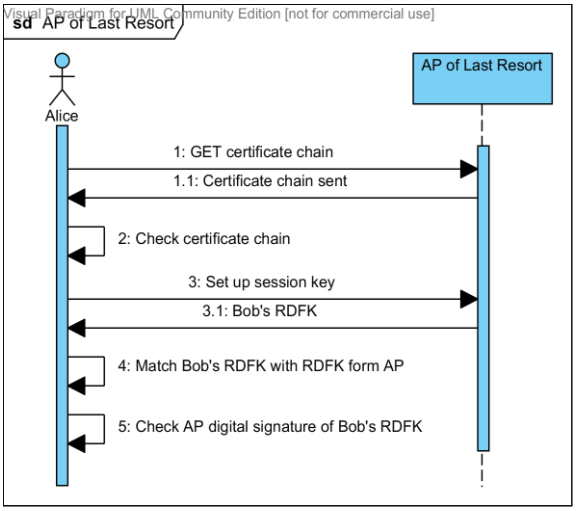

Figure 21 – Authentication Partner of Last Resort (APLR)

The client application must ask for the APLR certificate chain, verify the entire certificate chain, setup a session encryption key, ask for Bob's RDFK file, match Bob's RDFK with the RDFK received from the APLR and, finally, check a second digital signature, meaning that the APLR must use its PrK to digitally sign Bob's RDFK.

Step 3: Client identity verification

Alice sends the URI of her RDFK file. The 256 bits long random string used to challenge Bob is used as an auxiliary symmetric key to encrypt her URI and her self-signed certificate, before they are sent to Bob, which will be encrypted once

again with the already established session key. This is necessary, because Mallory could have positioned himself between Alice and Bob. Since he does not have Bob's PrK, he cannot know the secondary symmetric key.

Bob will decrypt Alice's URI and her digital certificate. He will also download Alice's RDFK file. He will use her PuK to recover the symmetric key used to encrypt the RDFK file and then he will challenge her to verify that she possesses the PrK of the certificate she sent and that matched the data stored in the RDFK file. The challenge is similar to the one she previously used to challenge Bob. If the response of the challenge is Ok, Bob can store Alice's certificate in a repository. Each certificate is about 1 KB in size. It can be persisted or cached in main memory. Bob may use any cache management strategy he wants to purge infrequently used certificates from clients.

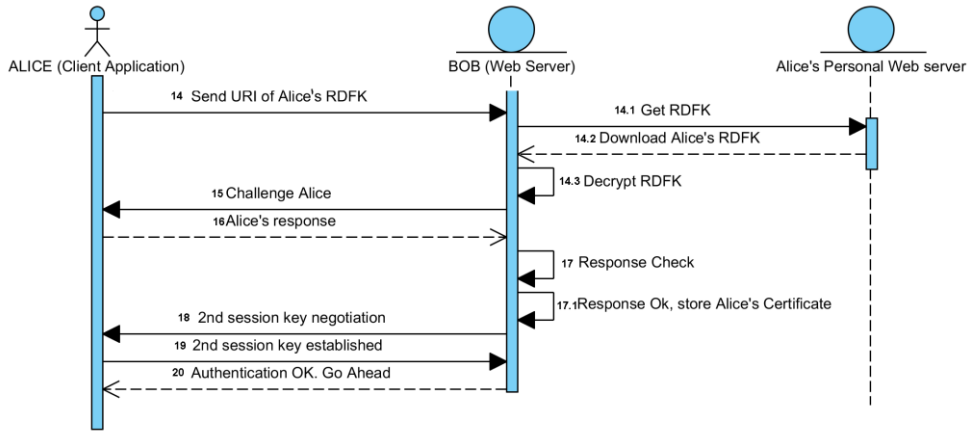

Figure 22 – Steps 3 and 4 – Client Authentication and session key renegotiation

Step 4: Session key renegotiation

In order to completely eliminate the MITM, if the authentication is successful, Alice and Bob reset the DH session key by encrypting Ya with Bob's PuK and Yb with Alice's Puk. Ya and Yb are the public information that Alice and Bob transmit to each other in order to calculate a common session key. Since Mallory does not have access to Alice's and Bob's PrK, he can no longer maintain a "secure" connection with Alice and Bob.

An alternative way would be to use each other's PuK to negotiate half of the new session encryption key each, so that both endpoints participate in the generation process of the session encryption key.

For security reasons, the session key and encryption protocol should be renegotiated periodically or when a certain amount of data have been exchanged by the parties (whichever occurs first) in order to make as difficult as possible for a cryptanalyst to be able to "guess" the session encryption key.

Step 5: Certificate storage

If both parties store each other's verified and authentic certificates, they can achieve faster authentications in the future by reducing the number of network operations and the number of asymmetric cryptography computations, since they are at least a hundred times slower than a hash calculation or a symmetric key encryption or decryption operation.

By caching certificates, Mallory is forced to devise a way to attack the server's or client's cache. He will try to poison the server's cache or create false entries in the client's certificate repository. Therefore, the cache must be protected at all costs. Compromising the cache would be as bad as compromising the private key.

However, because the cache resides inside some computer (either the client's or the server's), it will not be easy for Mallory to achieve his goals. He will need a "Trojan horse" to open a connection to his computer from Alice's computer, download and execute a worm with root or administrative rights in order to tamper with the cached certificate repository. If he is able to do that, he would also able to steal the private key of Alice's certificate, and therefore none of the proposed protocols in the literature would be any safer.

6.1.2. Second and subsequent authentications with cached certificates

If both Alice and Bob have each other's certificates cached, the authentication is much faster and secure. Since Alice knows Bob's PuK and Bob knows Alice's PuK, all they have to do is to challenge each other to verify they really are who they claim to be. No RDFK file download is required. The figure below shows the authentication process.

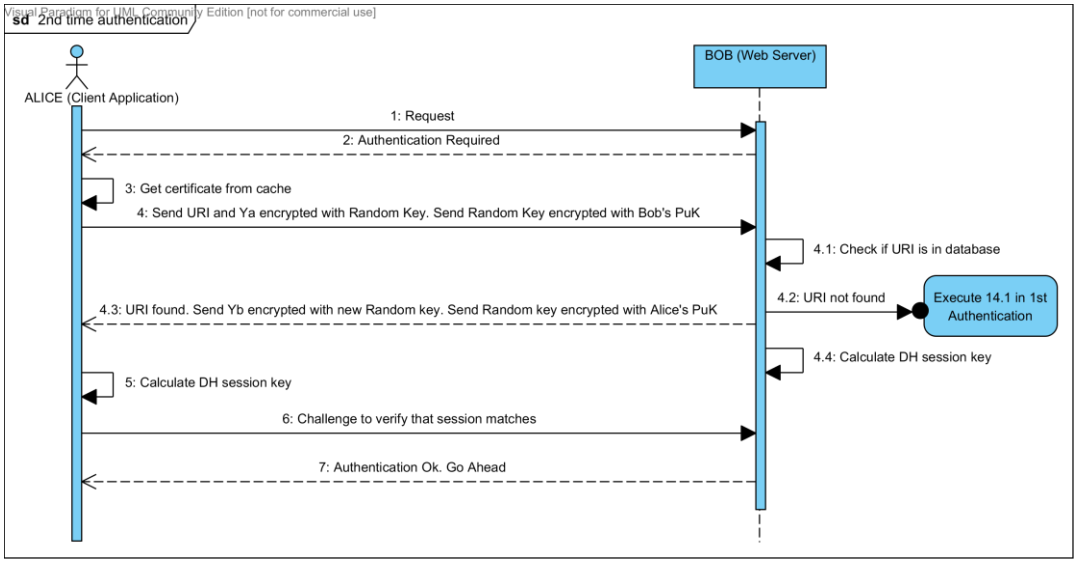

Figure 23 – Second and subsequent times authentication with enhanced security

Even if Alice's certificate is no longer in Bob's cache, the authentication is faster. Only Alice's RDFK needs to be downloaded. They start by establishing a secure channel using the Diffie-Hellman key exchange protocol, and then challenge each other. If, for some reason the challenge fails, they will know they were deceived somehow by Mallory, because although Mallory is able to deceive Alice and Bob, he does not have access to Alice's and Bob's private keys. Even if he is able to produce a false certificate with Alice's or Bob's URI and hack every server containing Alice's or Bob's RDFK file, he cannot mimic Alice's or Bob's key pair. Hence, when Alice and Bob try to establish communications with each other, using their real certificates, they will notice the PuKs are different from the ones they have in storage. (See 3.8)

This is a feature none of the other authentication protocols studied in this work have been able to provide until now.

The Diffie-Hellman parameters are slightly different from the first time authentication. If each PrK is less than the safe prime used for the calculations the Xa should be Alice's PrK and Xb should be Bob's PrK. Otherwise, the Xa or Xb generated on the first time authentication should be stored with the certificate's cache. A final challenge is only necessary to ensure that both endpoints have the same session key.

6.1.3.
Second and subsequent authentications with fast negotiation option

In order to speed up the authentication process, minimizing even more network and asymmetric key operations, SRAP has the option of fast negotiation, if the server agrees.

With Fast Negotiation, the Diffie-Hellman key exchange protocol is not used. Alice sends her URI and Half of the PRE_SESSION key, encrypted with Bob's PuK. Bob decrypts the message, gets Alice's client certificate from cache, generates and encrypts the other half of the PRE_SESSION key with Alice's PuK. Alice decrypts the other half of the PRE_SESSION key. The entire session key is the hash of the concatenation of both halves of the PRE_SESSION key (Alice's first and Bob's second). The only way the session encryption key is identical for both sides is if both sides have valid certificates previously cached. The challenge is simply the hash of Alice's URI, encrypted with the session symmetric key. Since Bob already knows Alice's URI, ha can calculate the hash and match with the decrypted hash transmitted by Alice.

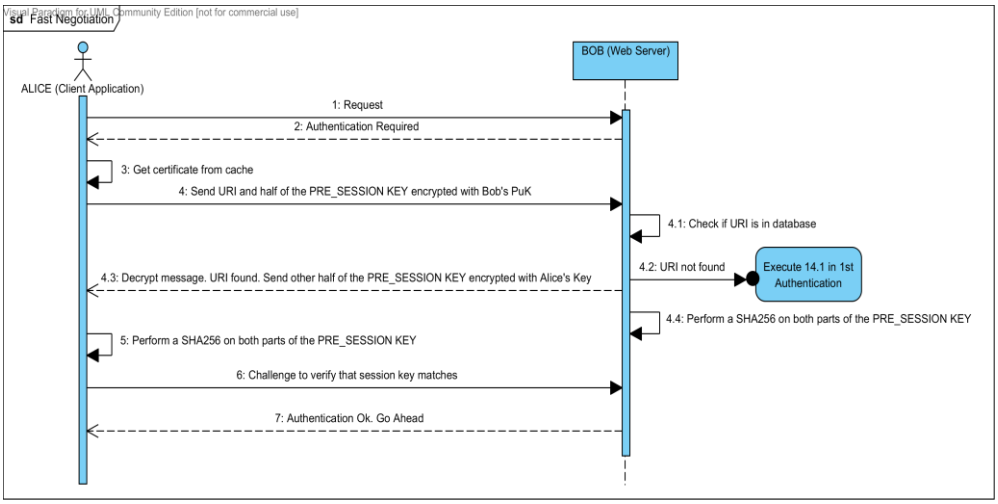

Figure 24 – SRAP with Fast Negotiation

6.2.
RDFK Details

An RDFK file must have encrypted information about the URI of the server (Server RDFK) or the client profile (Client RFDK), the public key of the self-signed certificate, the authentication partner list and the digital signature field

encrypted with the private key of the certificate. The RDFK stored in the authentication partner of last resort, must have an additional signature, signed with the private key of the certificate of the authentication partner of last resort, which must be issued by a trusted certificate authority.

The digital signature field contains two 32 bytes (256 bits) strings: the hash of the RDFK file and the password used to encrypt the RDFK file. The entire field must be signed with the RSA private key.

The digital signature of the authentication partner of last resort must only return true or false for verification purposes.

Since the RSA and CERT ontologies do not have all the properties required to implement SRAP, an SRAP ontology, which extends the CERT ontology, must be specified. The following properties are needed:

- SRAPEncryptedURI: points to another property that contains the server's or the client profile encrypted URI.
- encryptedAES256String: contains a string encrypted with the AES256 – CBC symmetric key algorithm
- CBCInitializationVector: contains a 16bytes hexadecimal number with the initialization vector for the AES-CBC algorithm.
- encryptedBlowfish256String: contains a string encrypted with the Blowfish symmetric key algorithm with a 256 bits key.
- encryptedSerpent256String: contains a string encrypted with the Serpent 256 symmetric key algorithm.
- SRAPEncryptedAuthenticationPartnersList: points to another property that contains the server's encrypted authentication partners list.
- SRAPDigitalSignature: points to a SRAPEncryptedSignature property that contains the encrypted hash and password of the RDFK file.
- SRAPEncryptedSignature: contains a 512 bit hexadecimal number containing 256 bits a SHA-256 hash and 256 bits, representing the

256 bit long password for the RDFK file. Both fields must be encrypted with a RSA private key to ensure non-repudiation.

- SRAPPartnerOfLastResortDigitalSignature: points to another property that contains the digital signature of the authentication partner of last resort.

- SRAPRSASignature: contains a hexadecimal number corresponding to the digital signature of the authentication partner of last resort, using the RSA algorithm

An example of a RDFK file is shown below:

```
@prefix : <http://www.w3.org/ns/auth/cert#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix rdfs: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix srap: <http://srap.com/SRAPSchema#> .

bob:me a srap:EncrpytedSrapResource;
  foaf:name "Bob";
  :key [ a :RSAPublicKey;
    rdfs:label "RDFK example";
    :modulus
"00cb24ed85d64d794b69c701c186acc059501e856000f661c93204d8380e07191c5c8b368d2ac32a428acb970398664368dc2a867320220f755
e99ca2eecdae62e8d15fb58e1b76ae59cb7ace8838394d59e7250b449176e51a494951a1c366c6217d8768d682dde78dd4d55e613f8839cf275d
4c8403743e7862601f3c49a6366e12bb8f498262c3c77de19bce40b32f89ae62c3780f5b6275be337e2b3153ae2ba72a9975ae71ab7246494970
66b660fcf774b7543d980952d2e8586200eda4158b014e75465d91ecf93efc7ac170c11fc7246fc6ded79c37780000ac4e079f671fd4f207ad77
0809e0e2d7b0ef5493befe73544d8e1be3dddb52455c61391a1"^^xsd:hexBinary;
    :exponent 65537 ;
  ] ;
  :uri [ a :SRAPEncryptedURI;
    :encryptedAES256String "ansjeu2nmdp03738wmd90nd0-ajnd-3hjde0-se8u39dnc-dfnf9dejnf-r9"^^xsd:string;
  ] ;
  :aplist [ a :SRAPEncryptedAuthenticationPartnersList;
    :encryptedAES256String "mdc904904jfrp9asncsd-0wefn___
(38dmnd8dfjduidndndndndn9884hfr9bcf9w9824nf9h076ae59cb7ace8838394d59e7250b449176e51a494951a1c366c6217d8768d682dde78d
d4d55e613f8839cf275d4c8403743e7862601f3c49a6366e12bb8f49804904jf076ae5987sjde-fnc92jf`p-jfj404jf-ajaw=-444gjkgjf-499
3hdhdhyfjfjfjf93jh39frj393983iu3jfrfjfps-4jg3lkhjt04j32je939f395jfmfjf83jfjf3ijnfjr-49trjfkfmfkf-=2kdmksa-wkfjgh3-56
38304ifjfjfjnfjonmplppoeuendundxnvmv,ejf94nf9bdpg93nslw03745wbcdnvpjhf444444rtjfjfjf-jnfkfjdf=21jrjf4jrfjfif04o4j36g
h493hfr93hr943hr93933fmrwe-,msdfabsdf0342jfhf012jismcvbvgmjdfpeidnsaudenf93nf9snd03983!2333@@344mfhfnf"^^xsd:string;
  ] ;
  :ds [ a :SRAPDigitalSignature;
    :SRAPEncryptedSignature
"4c8403743e7862601f3c49a6366e12bb8f498262c3c77de19bce40b32f89ae62366e12bb8f498262c3c77de19bce40b32f89ae62c3780f5b627
5be337e2b3001"^^xsd:hexBinary

  ] ;
  :aplrds [ a :SRAPPartnerOfLastResortDigitalSignature;
    :SRAPDSASignature "rfjfps-4jg3lkhjt04j32je939f395jfmfjf8rfjfps-4jg3lkhjt04j32je939f3"^^xsd:hexBinary
  ] .
```

Figure 25 – Example of a RDFK file for a server

6.3. SRAP Vulnerability Analysis

The first step of SRAP is to establish a secure communications channel between Alice and Bob. This step eliminates Eve but Mallory could establish a secure communications channel with Alice and another with Bob, without Alice and Bob noticing. Hence, this step does not protect Alice and Bob from Mallory. That's why Bob's and Alice's identities must be verified and a second session key must be negotiated in a way Mallory cannot interfere.

The second step of SRAP is the verification of Bob's (the Server) identity. In order to deceive Alice (the client), Mallory must make Alice think he is Bob and Bob's Authentication partners (Trent) or he must change Bob's RDFK stored in

Trent's server and make Alice think he is Bob. This is as difficult as it is to exploit the TLS vulnerability. If Alice has already authenticated herself with one of Bob's authentication partners, it would be very difficult for Mallory to deceive Alice, because she already would have the partner's certificate cached, whether it is self-signed or CA signed. Alice will establish a secure connection with a partner's verified PuK, which Mallory cannot break.

Mallory has only one chance to deceive Alice. This chance happens on the first time Alice wants to authenticate herself with Bob. Once Alice successfully authenticates herself with Bob, she will cache Bob's validated certificate and will use it to challenge Bob. Since Mallory does not have Bob's PrK, there is no way he can respond to Alice's challenge. Mallory must find a way to flush Alice's certificate storage or hack it. Not an easy thing to do, especially if the storage is encrypted. With WebID and TLS, it is possible for Mallory to deceive Alice and Bob without them finding out they were deceived. (See 5.3.1). The same is not true with SRAP. If Alice is deceived by Mallory and stores Mallory's certificate instead of Bob's, when she indeed connects herself with Bob, Bob will respond his PrK, which does not match the PuK used in Alice's challenge. Alice will find out she was deceived.

The third step of SRAP is the verification of Alice's (the Client) identity. An RDF file is a public document, but an RDFK is not. The location of a RDFK file is not revealed unless authentication is required. Although Alice's Puk is indeed public, it is also not distributed or revealed, unless authentication is required. So, unlike WebID, it would be difficult for Mallory to deceive Bob without deceiving Alice first. If he is able to deceive Alice, he still needs to hack Alice's personal Web server and change Alice's RDFK file with his own. He cannot alter Alice's RDFK because he does not have Alice's PrK to sign the RDFK file. He must replace the entire file, unlike WebID, where he needed only to change the modulus field (the public key) of the file. If he uses his own URI, instead of Alice's, to authenticate with Bob then Bob will not recognize him as Alice.

Like WebID, SRAP current specification uses RSA public and private keys only. TLS, however, can use El-Gamal [52], and Elliptic Curve Cryptography (ECC) [53] algorithms as well. ECC keys are smaller and ECC algorithms faster than RSA equivalents, but RSA is the only cryptosystem that allows encryption with the PrK (digital signing) and decryption with the PuK (signature

verification). It is necessary to decrypt the key stored in the encryptedAES256String, encryptedBlowfish256String or encryptedSerpent256String properties. Such key is used to decrypt the RDFK URI, and AP list. Because the key is encrypted with an RSA PrK, it can only be recovered with the corresponding RSA PuK. That is why he cannot modify Alice's RDFK file.

6.3.1. SRAP Resilience Against Eavesdropping Attacks

The Diffie-Hellman key exchange eliminates any eavesdropper on the first authentication. On the second and subsequent authentication using Fast Negotiation, no information is sent unencrypted. Eavesdropping passively does not achieve any goal for the attacker.

6.3.2. SRAP Resilience Against Modification Attacks

Modification attacks have the best chance of beating most protocols. In the case of SRAP, to be successful, a modification attack must first be able to update the client's RDFK file as mentioned in section 6.3. A modification attack that does not alter the client's RDFK file would authenticate the attacker as himself, not as the client.

6.3.3. SRAP Resilience Against Replay, Preplay and Reflection Attacks

Since the challenges are chosen randomly by the challenger, encrypted with the challenged PuK and the challenged party is required to use his/hers PrK to answer the challenger correctly, replay attacks will not work.

6.3.4.
SRAP Resilience Against DoS and DDoS Attacks

In our understanding, it is not up to any authentication protocol to provide resilience against DoS or DDoS attacks. It is up to network engineers and network designers.

6.3.5.
SRAP Resilience Against Typing Attacks

At this time, it is unclear if typing attacks could be successful against SRAP. RDFKs are digitally signed and clearly defines the encryption algorithm used to encrypt the RDFK file. Any attempt to tamper with the RDKF file, while in transit, would fail.

The second key negotiation would be done using both the client's and the server's PuKs, requiring both PrKs to decrypt the session key negotiation. It is possible to tamper with the Diffie-Hellman key negotiation, but hardly the second key.

The second session key is not a long term session key, and should be changed periodically, making typing attacks even more difficult to succeed.

6.3.6.
SRAP Resilience Against Cryptanalysis Attacks

Since no information ever travels unencrypted, there are no weak keys in SRAP and the second session key is periodically modified, Cryptanalysis have a low chance of success against SRAP.

6.3.7.
SRAP Resilience Against Certificate Manipulation

In order for certificate manipulation attacks to work, proof of possession of the PrKs should not be an issue. SRAP strongly demands the proof of possession of both client and server PrKs. For the APLR, unlike TLS, SRAP cannot be configured to ignore certificate chain errors.

6.3.8.
SRAP Resilience Against Protocol Interaction

SRAP is much less flexible than WebID and TLS. Both WebID and TLS allows cipher suite changes. SRAP does not. Once a cipher suite has been negotiated, it cannot be changed in the same session. At this time, we see an implementation error as the only possible way for Protocol Interaction Attacks to work.

6.4.
SRAP Performance

Since SRAP has not been implemented yet, an analytic study of the computational costs cannot be made at this time, but they can be estimated on statistical data, considering the each type of operation involved.

A similar work has been done with KERBEROS [VIII] by HARBITTER and MENASCE [54], although in their work, the authors did not consider the network cost.

When comparing TLS, WebID and SRAP, the following operations parameters are significant:

- Network Operations: the packet travel time between hosts. In our analysis, we estimate this based on the average round trip time (RTT) of several web sites and then divided by 2, so we can estimate the "average" or "typical" time expended in milliseconds for a network packet to travel from one host to another (See attachment 1).

- RSA Encryption/Decryption Operations: using Chilkats' [IX] commercial libraries, we implemented the RSA encryption/decryption operations and calculated the CPU time in milliseconds each operation requires.

VIII http://web.mit.edu/kerberos/

IX Chilkat Software Inc. is a developer of components and libraries for developers worldwide. http://www.chilkatsoft.com

- Diffie-Hellman Key Exchange: at first, we implemented in JAVA, using the big integer class, but Chilkat's libraries were twice as fast as JAVA, running in Visual Basic 6 (See attachment 2).

- Symmetric Encryption/Decription Operations: calculated using Chilkat's commercial libraries.

- Hash Operations: also calculated using Chilkat's libraries.

From the experiment, using a Dell® Optiplex® 780 with an Intel® Core2 Duo® CPU @ 2.93GHz and DDR3-1333 RAM (1066MHz FSB), running a 32 bits Microsoft® Windows® 7, we determined the following time values:

Network Operations (**No**): varies from 5 to 50ms

RSA Private Key Operations (2048 bits) (**RSApr**): 31ms. The PrK is used for decryption and for digital signature generation.

RSA Public Key Operations (2048 bits) (**RSApu**): 1.7ms. The PuK is used for encryption and for digital signature verification.

Diffie-Hellman Key Exchange (2048 bits) (**DH-KX**): 63ms for the entire process.

Symmetric Key Operations (AES 256 bits) (**SKo**): 0.028ms

Hash Operations (SHA 256) (**Ho**): 0.020ms

RSA encryption and digital signature verification is significantly faster than decryption and digital signature generation, because the public key exponent (commonly 65537) is much smaller than the private key exponent.

The calculated and measured times served as homogenization factors for the total computational cost. The Diffie-Hellman key exchange takes 7 steps and the average time of 9ms was used for each step.

The following tables show the breakdown of each protocol in terms of basic steps, showing for each step, the source (from) and destination (to) in which the

network packets are transmitted and the quantity of relevant operations to complete the step.

When the source is the client and the destination is the server, we mean that the client initializes the step and the server receives some data produced by the client.

When the source is the server and the destination is the client, we mean that on an already established TCP connection, the server is the initiating side of the step and the client receives the data produced by the server.

When the source and destination are the same, we mean that there are relevant operations other than network operations, executed on either the client or the server side.

When the source or the destination is an Authentication Partner, it is signaled as AP.

When the source or the destination is the Client Personal Web Server, it is signaled as CPWS.

By analyzing the sequence diagram of TLS and referring to RFC 5246 (See Figure 26), we decomposed the execution of the authentication as a sequence of steps, as follows:

TLS with both client and server authentication computational costs

Step	From	To	No	RSApr	RSApu	DH-KX	Ho
TCP Handshake	Client	Server	3	0	0	0	0
Client Hello	Client	Server	1	0	0	0	0
Server Hello	Server	Client	1	0	0	0	0
Server Certificate Chain	Server	Client	2	0	0	0	0
Server Key-Exchange (RSA)	Server	Client	0	0	0	0	0
Client Certificate Request	Server	Client	1	0	0	0	0
Server Hello Done	Server	Client	1	0	0	0	0
Server Certificate Chain Verification	Client	Client	0	0	2	0	2
Client Certificate Chain	Client	Server	2	0	0	0	0
Client Key Exchange	Client	Server	1	0	1	0	1
Server PRE_MASTER_SECRET Decryption	Server	Server	0	1	0	0	1
Certificate Verify	Client	Server	1	1	0	0	1
Client Certificate Chain Verification	Server	Server	0	0	3	0	2
Master_Secret Computation	Server	Server	0	0	0	0	1
Master_Secret Computation	Client	Client	0	0	0	0	1
Finished	Client	Server	1	0	0	0	0
Finished	Server	Client	1	0	0	0	0
Total			**15**	**2**	**6**	**0**	**9**

Table 1 – Operations for TLS mutual authentication with client and server certificates

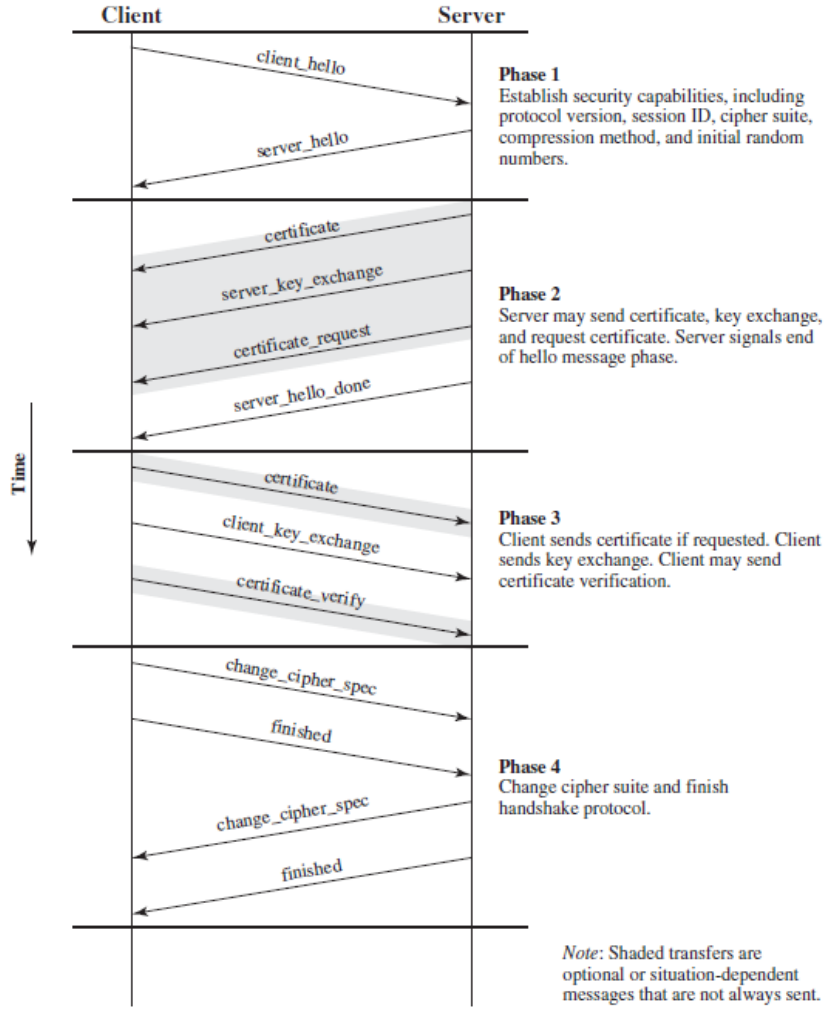

Figure 26 – TLS Sequence [55]

The same table was created for WebID and SRAP, in all possible scenarios.

WebID Computational Costs, using client certificates GET RDF via HTTP

Step	From	To	No	RSApr	RSApu	DH-KX	Ho	SKo
TCP Handshake	Client	Server	3	0	0	0	0	0
Client Hello	Client	Server	1	0	0	0	0	0
Server Hello	Server	Client	1	0	0	0	0	0
Server Certificate Chain	Server	Client	2	0	0	0	0	0
Server Key-Exchange (RSA)	Server	Client	0	0	0	0	0	0
Server Hello Done	Server	Client	1	0	0	0	0	0
Server Certificate Chain Verification	Client	Client	0	0	2	0	2	0
Client Key Exchange	Client	Server	1	0	1	0	1	0
Server PRE_MASTER_SECRET Decryption	Server	Server	0	1	0	0	1	0
Master_Secret Computation	Server	Server	0	0	0	0	0	0
Master_Secret Computation	Client	Client	0	0	0	0	0	0
Finished	Client	Server	1	0	0	0	0	0
Finished	Server	Client	1	0	0	0	0	0
SubTotal TLS Light			**11**	**1**	**3**	**0**	**4**	**0**
Client URI	Client	Server	1	0	0	0	0	2
Client Certificate Request	Server	Client	1	0	0	0	0	2
Client Certificate e PrK possetion proof	Client	Server	1	1	0	0	1	2
Certificate Verify	Client	Server	1	0	1	0	2	2
RDF Download via HTTP	Server	CPWS	6	0	0	0	0	0
Key Match	Server	Server	0	0	0	0	0	0
Authorized	Server	Client	1	0	0	0	0	2
SubTotal RDF Download and verification			**11**	**1**	**1**	**0**	**3**	**10**
Total Cost			**22**	**2**	**4**	**0**	**7**	**10**

Table 2 – Operations for WebID fetching client RDF via HTTP

WebID Computational Costs, using client certificates GET RDF via HTTPS

Step	From	To	No	RSApr	RSApu	DH-KX	Ho	SKo
TCP Handshake	Client	Server	3	0	0	0	0	0
Client Hello	Client	Server	1	0	0	0	0	0
Server Hello	Server	Client	1	0	0	0	0	0
Server Certificate Chain	Server	Client	2	0	0	0	0	0
Server Key-Exchange (RSA)	Server	Client	0	0	0	0	0	0
Server Hello Done	Server	Client	1	0	0	0	0	0
Server Certificate Chain Verification	Client	Client	0	0	2	0	2	0
Client Key Exchange	Client	Server	1	0	1	0	1	0
Server PRE_MASTER_SECRET Decryption	Server	Server	0	1	0	0	1	0
Master_Secret Computation	Server	Server	0	0	0	0	0	0
Master_Secret Computation	Client	Client	0	0	0	0	0	0
Finished	Client	Server	1	0	0	0	0	0
Finished	Server	Client	1	0	0	0	0	0
SubTotal TLS Light			**11**	**1**	**3**	**0**	**4**	**0**
Client URI	Client	Server	1	0	0	0	0	2
Client Certificate Request	Server	Client	1	0	0	0	0	2
Client Certificate e PrK possetion proof	Client	Server	1	0	1	0	1	2
Certificate Verify	Client	Server	1	1	0	0	2	2
RDF Download via HTTPS	Server	CPWS	11	1	3	0	4	2
Key Match	Server	Server	0	0	0	0	0	0
Authorized	Server	Client	1	0	0	0	0	2
SubTotal RDF Download and verification			**16**	**2**	**4**	**0**	**7**	**12**
Total Cost			**27**	**3**	**7**	**0**	**11**	**12**

Table 3 – Operations for WebID fetching client RDF via HTTPS

WebID Computational Costs, using client certificates GET RDF via HTTP

Step	From	To	No	RSApr	RSApu	DH-KX	Ho	SKo
TCP Handshake	Client	Server	3	0	0	0	0	0
Client Hello	Client	Server	1	0	0	0	0	0
Server Hello	Server	Client	1	0	0	0	0	0
Server Certificate Chain	Server	Client	2	0	0	0	0	0
Server Key-Exchange (RSA)	Server	Client	0	0	0	0	0	0
Server Hello Done	Server	Client	1	0	0	0	0	0
Server Certificate Chain Verification	Client	Client	0	0	2	0	2	0
Client Key Exchange	Client	Server	1	0	1	0	1	0
Server PRE_MASTER_SECRET Decryption	Server	Server	0	1	0	0	1	0
Master_Secret Computation	Server	Server	0	0	0	0	0	0
Master_Secret Computation	Client	Client	0	0	0	0	0	0
Finished	Client	Server	1	0	0	0	0	0
Finished	Server	Client	1	0	0	0	0	0
SubTotal TLS Light			**11**	**1**	**3**	**0**	**4**	**0**
Client URI	Client	Server	1	0	0	0	0	2
Client Certificate Request	Server	Client	1	0	0	0	0	2
Client Certificate e PrK possetion proof	Client	Server	1	0	1	0	1	2
Certificate Verify	Client	Server	1	1	0	0	2	2
Key Match	Server	Server	0	0	0	0	0	0
Authorized	Server	Client	1	0	0	0	0	2
SubTotal RDF Download and verification			**5**	**1**	**1**	**0**	**3**	**10**
Total Cost			**16**	**2**	**4**	**0**	**7**	**10**

Table 4 – Operations for WebID best case scenario

SRAP Computational Costs, 1st time authentication using AP of Last Resort

Step	From	To	No	RSApr	RSApu	DH-KX	Ho	SKo
Phase1: Diffie-Hellman Key exchange								
TCP Handshake	Client	Server	3	0	0	0	0	0
Client Hello	Client	Server	1	0	0	3	1	0
Server Hello	Server	Client	1	0	0	2	2	0
Calculate K	Client	Client	2	0	0	1	1	0
Calculate K	Server	Server	0	0	0	1	1	0
SubTotal Phase1			**7**	**0**	**0**	**7**	**5**	**0**
Phase2: Server Authentication								
Request ID	Client	Server	1	0	0	0	0	1
Send Server Certificate, RDFK & AP List	Server	Client	2	0	0	0	1	3
Server Verify	Client	Client	0	0	1	0	2	2
APLR Connect and DH-KX setup	Client	APLR	7	0	0	7	5	0
GET Certificate Chain	Client	APLR	1	0	0	0	0	1
SEND Certificate Chain	APLR	Client	2	0	0	0	0	1
Verify Certificate Chain	Client	Client	0	0	2	0	2	1
Setup Session Key	Client	APLR	1	1	1	0	0	1
GET RDFK	Client	APLR	1	0	0	0	0	1
RDFK Download	APLR	Client	2	0	0	0	0	1
APLR Verify	Client	Client	0	0	2	0	0	1
Challenge Server	Client	Server	1	0	1	0	1	1
Server Response	Server	Client	1	1	0	0	2	1
Response Check	Client	Client	0	0	0	0	1	0
SubTotal Phase2			**19**	**2**	**7**	**7**	**14**	**15**
Phase3: Client Authentication								
Client URI + Certificate	Client	Server	1	0	0	0	0	4
HTTP GET URI	Client	CPWS	4	0	0	0	0	0
RDFK Download via HTTP	CPWS	Client	1	0	0	0	0	0
Challenge Client	Server	Client	1	0	1	0	1	2
Client Response	Client	Server	1	1	0	0	2	2
Response Check	Server	Server	0	0	0	0	1	0
Rengotiate Session Key	Client	Server	1	0	1	0	1	1
Renegotiate Session Key	Server	Server	1	1	0	0	1	1
Go Ahead	Server	Client	1	0	0	0	0	1
SubTotal RDF Phase3			**11**	**2**	**2**	**0**	**6**	**11**
Total Cost			**37**	**4**	**9**	**14**	**25**	**26**

Table 5 – Operations for SRAP 1st time authentication, using AP of last resort

SRAP Computational Costs, 1st time authentication using Trusted AP

Step	From	To	No	RSApr	RSApu	DH-KX	Ho	SKo
Phase1: Diffie-Hellman Key exchange								
TCP Handshake	Client	Server	3	0	0	0	0	0
Client Hello	Client	Server	1	0	0	3	1	0
Server Hello	Server	Client	1	0	0	2	2	0
Calculate K	Client	Client	2	0	0	1	1	0
Calculate K	Server	Server	0	0	0	1	1	0
SubTotal Phase1			**7**	**0**	**0**	**7**	**5**	**0**
Phase2: Server Authentication								
Request ID	Client	Server	1	0	0	0	0	1
Send Server Certificate, RDFK & AP List	Server	Client	2	0	0	0	1	3
Server Verify	Client	Client	0	0	2	0	2	2
TCP Handshake	Client	AP	3	0	0	0	0	0
Session setup	Client	AP	1	0	1	0	1	0
AP Response	AP	Client	1	1	0	0	2	1
GET RDFK	Client	AP	1	0	0	0	1	1
RDFK Download	AP	Client	2	0	0	0	2	1
AP Verify	Client	Client	0	0	2	0	3	2
Challenge Server	Client	Server	1	0	1	0	1	2
Server Response	Server	Client	1	0	1	0	2	2
Response Check	Client	Client	0	0	0	0	1	0
SubTotal Phase2			**13**	**1**	**7**	**0**	**16**	**15**
Phase3: Client Authentication								
Client URI + Certificate	Client	Server	1	0	0	0	0	4
HTTP GET URI	Client	CPWS	4	0	0	0	0	0
RDFK Download via HTTP	CPWS	Client	1	0	0	0	0	0
Challenge Client	Server	Client	1	0	1	0	1	2
Client Response	Client	Server	1	1	0	0	2	2
Response Check	Server	Server	0	0	0	0	1	0
Rengotiate Session Key	Client	Server	1	0	1	0	1	1
Renegotiate Session Key	Server	Server	1	1	0	0	1	1
Go Ahead	Server	Client	1	0	0	0	0	1
SubTotal RDF Phase3			**11**	**2**	**2**	**0**	**6**	**11**
Total Cost			**31**	**3**	**9**	**7**	**27**	**26**

Table 6 – SRAP 1st time authentication using a trusted AP

SRAP Computational Costs, client certificate removed from cache

Step	From	To	No	RSApr	RSApu	DH-KX	Ho	SKo
Phase1: Diffie-Hellman Key exchange								
TCP Handshake	Client	Server	3	0	0	0	0	0
Client Challenge	Client	Server	1	0	1	3	1	1
Calculate K	Server	Server	0	1	0	1	1	0
Server Response. Certificate not in cache	Server	Client	1	0	0	2	2	2
Calculate K	Client	Client	0	0	0	1	1	0
SubTotal Phase1			**5**	**1**	**1**	**6**	**4**	**3**
Phase2: Client Authentication								
Client URI + Certificate	Client	Server	1	0	0	0	0	4
HTTP GET URI	Client	CPWS	4	0	0	0	0	0
RDFK Download via HTTP	CPWS	Client	1	0	0	0	0	0
Challenge Client	Server	Client	1	0	1	0	1	2
Client Response	Client	Server	1	1	0	0	2	2
Response Check	Server	Server	0	0	0	0	1	0
Reset Session Key	Server	Client	1	0	1	2	2	1
Calculate K	Client	Client	0	1	0	1	1	0
Verify matching session keys	Client	Server	1	0	0	0	1	1
Authentication Ok, go ahead	Server	Client	1	0	0	0	1	1
SubTotal RDF Phase2			**11**	**2**	**2**	**3**	**9**	**11**
Total Cost			**16**	**3**	**3**	**9**	**13**	**14**

Table 7 – SRAP authentication with client certificate removed from server cache

SRAP Computational Costs, certificate in cache Enhanced Security

Step	From	To	No	RSApr	RSApu	DH-KX	Ho	SKo
Single Phase								
TCP Handshake	Client	Server	3	0	0	0	0	0
Client Challenge	Client	Server	1	0	1	2	1	1
Calculate K	Server	Server	0	1	0	1	1	0
Server Challenge	Server	Client	1	0	1	1	2	2
Calculate K	Client	Client	0	1	0	1	1	1
Verify matching session keys	Client	Server	1	0	0	0	1	1
Authentication Ok, go ahead	Server	Client	1	0	0	0	1	1
Total Cost			**7**	**2**	**2**	**5**	**7**	**6**

Table 8 – SRAP 2nd and subsequent authentications, certificate in server cache with enhanced security option

SRAP Computational Costs, client certificate in cache - Fast Negotiation

Step	From	To	No	RSApr	RSApu	DH-KX	Ho	SKo
Single Phase								
TCP Handshake	Client	Server	3	0	0	0	0	0
Client Challenge	Client	Server	1	0	1	0	1	1
Server Verification	Server	Server	0	1	0	0	2	0
Server Challenge	Server	Client	1	0	1	0	2	2
Client Verification	Client	Client	0	1	0	0	2	1
Verify matching session keys	Client	Server	1	0	0	0	1	1
Authentication Ok, go ahead	Server	Client	1	0	0	0	1	1
Total Cost			**7**	**2**	**2**	**0**	**9**	**6**

Table 9 – SRAP 2nd and subsequent authentications, certificate in server cache with fast negotiation option

TLS, WebID & SRAP Computational Costs Summary

Protocol	No	RSApr	RSApu	DH-KX	Ho	SKo
TLS Full	15	2	6	0	9	0
TLS Full with 2 Intermediate CA	17	2	8	0	11	0
WebID RDF Download via HTTP	22	2	4	0	7	10
WebID RDF Download via HTTPS	27	3	7	0	11	12
WebID Client certificate in cache	16	2	4	0	7	10
SRAP 1st time auth AP of Last Resort	37	4	9	14	25	26
SRAP 1st time auth Trusted AP	31	3	9	7	27	26
SRAP 2nd+ time auth client cert not in cache	16	3	3	9	13	14
SRAP 2nd+ time auth Enhanced Security	7	2	2	5	7	6
SRAP 2nd+ time auth Fast Negotiation	7	2	2	0	9	6
Homogenization Factors ms	21	31	1,7	8,9	0,028	0,02

Protocol	No	RSApr	RSApu	DH-KX	Ho	SKo	Total Cost
TLS Full	315	62	10,2	0	0,252	0	**387**
TLS Full with 2 Intermediate CA	357	62	13,6	0	0,308	0	433
WebID RDF Download via HTTP	462	62	6,8	0	0,196	0,2	531
WebID RDF Download via HTTPS	567	93	11,9	0	0,308	0,24	672
WebID Client certificate in cache	336	62	6,8	0	0,196	0,2	**405**
SRAP 1st time auth AP of Last Resort	777	124	15,3	125	0,7	0,52	1.042
SRAP 1st time auth Trusted AP	651	93	15,3	62	0,756	0,52	823
SRAP 2nd+ time auth client cert not in cache	336	93	5,1	80	0,364	0,28	515
SRAP 2nd+ time auth Enhanced Security	147	62	3,4	45	0,196	0,12	**257**
SRAP 2nd+ time auth Fast Negotiation	147	62	3,4	0	0,252	0,12	**213**

Table 10 – Protocol Summary Table

As seen in Table 5, when using SRAP for the first time in the worst-case scenario (authenticate the server with the partner of last resort), SRAP has the worst performance. Nevertheless, as expected, it has the best performance, if the certificates are cached (Table 9). On the first authentication turn, there is a tradeoff: we sacrifice CPU and Network (and battery on a mobile device) to be sure we are communicating with the server we were supposed to be, and not Mallory. But, if we already have the certificates cached, SRAP can be twice as fast as TLS and WebID. Since network latency varies, the chart from Figure 27 shows the performance of the protocols according to different network latency values.

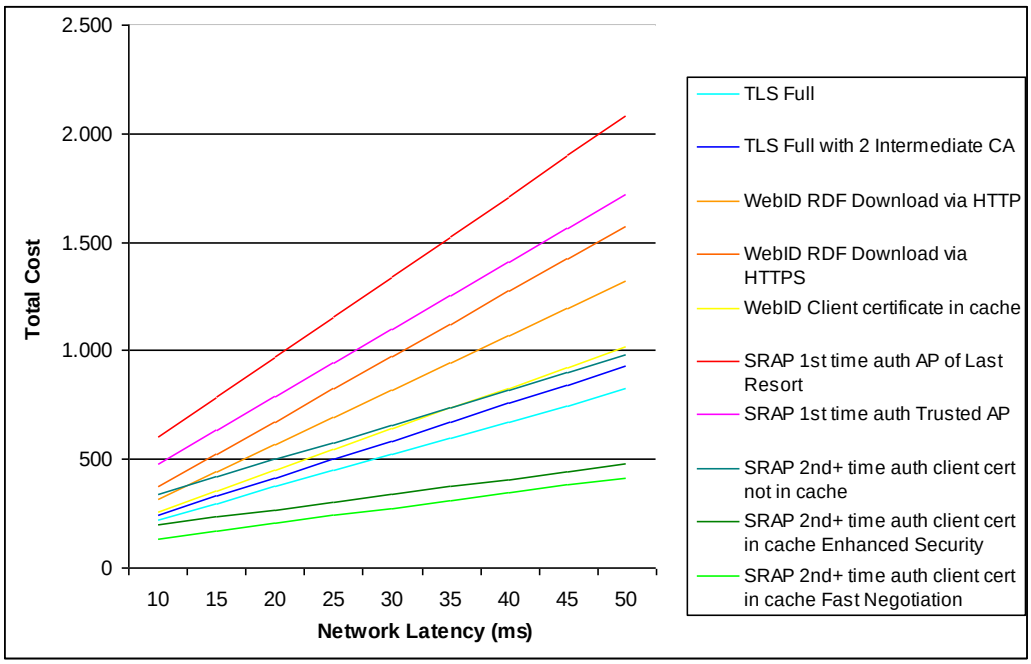

Figure 27 – Protocol performance comparison with multiple network latency times

Besides the first computer, we gather results from two other desktop computers for comparison:

2) An AMD® Athlon 64 Dual Core 4400+ @ 2,3GHz CPU with 1GB DDR2 800 RAM, running a 32 bits Microsoft® Windows XP® operating system.

For this computer, we have got the following values for the asymmetric encryption and decryption operations:

RSA Private Key Operations (2048 bits) (**RSApr**): 55ms.

RSA Public Key Operations (2048 bits) (**RSApu**): 2.8ms.

Diffie-Hellman Key Exchange (2048 bits) (**DH-KX**): 109ms for the entire process.

3) An Intel® Core I7 2600 @ 3,4GHz CPU with 12GB DDR3 1333 RAM, running a 64 bits Microsoft® Windows 7®

For this computer, we have got the following values for the asymmetric encryption and decryption operations:

RSA Private Key Operations (2048 bits) (**RSApr**): 22ms.

RSA Public Key Operations (2048 bits) (**RSApu**): 1.1ms.

Diffie-Hellman Key Exchange (2048 bits) (**DH-KX**): 39ms for the entire process.

Symmetric key and hashing operations, as seen in Table 10, do not influence the total cost significantly.

Although the hardware of second computer is outdated for today's standards, it gives us the equivalent performance of a mobile phone or tablet. However, the third computer is considered a top of the line model for today's standards.

Using the timings as homogenization factors, we have the following performance tables for computers 2 and 3 respectively.

TLS, WebID & SRAP Computational Costs Summary

Protocol	No	RSApr	RSApu	DH-KX	Ho	SKo
TLS Full	15	2	6	0	9	0
TLS Full with 2 Intermediate CA	17	2	8	0	11	0
WebID RDF Download via HTTP	22	2	4	0	7	10
WebID RDF Download via HTTPS	27	3	7	0	11	12
WebID Client certificate in cache	16	2	4	0	7	10
SRAP 1st time auth AP of Last Resort	37	4	9	14	25	26
SRAP 1st time auth Trusted AP	31	3	9	7	27	24
SRAP 2nd+ time auth client cert not in cache	16	3	3	9	13	12
SRAP 2nd+ time auth Enhanced Security	7	2	2	5	7	6
SRAP 2nd+ time auth Fast Negotiation	7	2	2	0	9	6
Homogenization Factors ms	21	55	2,8	15,57	0,028	0,02

Protocol	No	RSApr	RSApu	DH-KX	Ho	SKo	Total Cost
TLS Full	315	110	16,8	0	0,252	0	**442**
TLS Full with 2 Intermediate CA	357	110	22,4	0	0,308	0	490
WebID RDF Download via HTTP	462	110	11,2	0	0,196	0,2	584
WebID RDF Download via HTTPS	567	165	19,6	0	0,308	0,24	752
WebID Client certificate in cache	336	110	11,2	0	0,196	0,2	**458**
SRAP 1st time auth AP of Last Resort	777	220	25,2	218	0,7	0,52	1.241
SRAP 1st time auth Trusted AP	651	165	25,2	109	0,756	0,48	951
SRAP 2nd+ time auth client cert not in cache	336	165	8,4	140	0,364	0,24	650
SRAP 2nd+ time auth Enhanced Security	147	110	5,6	78	0,196	0,12	**341**
SRAP 2nd+ time auth Fast Negotiation	147	110	5,6	0	0,252	0,12	**263**

Table 11 – Protocol Summary for computer 2

TLS, WebID & SRAP Computational Costs Summary

Protocol	No	RSApr	RSApu	DH-KX	Ho	SKo
TLS Full	15	2	6	0	9	0
TLS Full with 2 Intermediate CA	17	2	8	0	11	0
WebID RDF Download via HTTP	22	2	4	0	7	10
WebID RDF Download via HTTPS	27	3	7	0	11	12
WebID Client certificate in cache	16	2	4	0	7	10
SRAP 1st time auth AP of Last Resort	37	4	9	14	25	26
SRAP 1st time auth Trusted AP	31	3	9	7	27	24
SRAP 2nd+ time auth client cert not in cache	16	3	3	9	13	12
SRAP 2nd+ time auth Enhanced Security	7	2	2	5	7	6
SRAP 2nd+ time auth Fast Negotiation	7	2	2	0	9	6
Homogenization Factors ms	21	22	1,1	5,57	0,028	0,02

Protocol	No	RSApr	RSApu	DH-KX	Ho	SKo	Total Cost
TLS Full	315	44	6,6	0	0,252	0	**366**
TLS Full with 2 Intermediate CA	357	44	8,8	0	0,308	0	410
WebID RDF Download via HTTP	462	44	4,4	0	0,196	0,2	511
WebID RDF Download via HTTPS	567	66	7,7	0	0,308	0,24	641
WebID Client certificate in cache	336	44	4,4	0	0,196	0,2	**385**
SRAP 1st time auth AP of Last Resort	777	88	9,9	78	0,7	0,52	954
SRAP 1st time auth Trusted AP	651	66	9,9	39	0,756	0,48	767
SRAP 2nd+ time auth client cert not in cache	336	66	3,3	50	0,364	0,24	456
SRAP 2nd+ time auth Enhanced Security	147	44	2,2	28	0,196	0,12	**221**
SRAP 2nd+ time auth Fast Negotiation	147	44	2,2	0	0,252	0,12	**194**

Table 12 – Protocol Summary for Computer 3

6.5. SRAP Cost Effectiveness

From the chart on Figure 27, we are able to determine the cost effectiveness of SRAP in relation to TLS. The chart shows very clearly that the greater the network latency, the more effective SRAP is in relation to TLS, after the first authentication. Hence, the higher the network speed, the more effective TLS is.

The cost effectiveness of SRAP must take the highest cost of the first authentication (which is always higher than TLS), add the cost of the subsequent authentications (which is always lower than TLS) and determine the breakeven point where the constant TLS accumulated cost is greater than SRAP.

The simple equation bellow shows us the breakeven point at which SRAP is more cost effective than TLS.

$TLS \times n \leq SRAP_{(bc)} \times (n-1) + SRAP_{(wc)}$

From which we determine n

$$n > \frac{(SRAP(wc) - SRAP(bc))}{(TLS - SRAP(bc))}$$

Where *n* is the nth time authentication turn, TLS is the constant cost for each TLS authentication, $SRAP_{(wc)}$ is the first time authentication turn cost for a worst case scenario and $SRAP_{(bc)}$ is the second and subsequent authentication turns for SRAP in a best case scenario, where client and server certificates are cached.

On a 5ms latency network, the threshold for SRAP with enhanced security to beat TLS is reached. Below that, it would cost more than TLS. It takes 123 authentication turns for SRAP with enhanced security to beat TLS, while it takes 7 turns for SRAP with Fast Negotiation to achieve the same goal.

On a 20ms latency network, SRAP beats TLS with nine authentication turns when enhanced security is used, but it takes only five turns to beat TLS when fast negotiation is used.

On a 40ms or worse, latency network, SRAP beats TLS after five authentications regardless of whether enhanced security or fast negotiation is used. (See Attachment 1 – Network Operations Report).

Typical latencies for social networks are 43ms. For on-line storage sites such as *github* or *google* drive, the average latency is 53ms. For webmail sites such as Yahoo or I-Cloud, the average latency is 25ms. For e-commerce sites such as PayPal, BestBuy or Mercado Livre, the average latency is 57ms. Unfortunately, we have been unable to test governments and financial institutions sites because the ones we tried are either are hosted in a cloud or do not respond to ICMP packets.

The following charts better illustrate SRAP cost effectiveness for various network latencies for each of the computers.

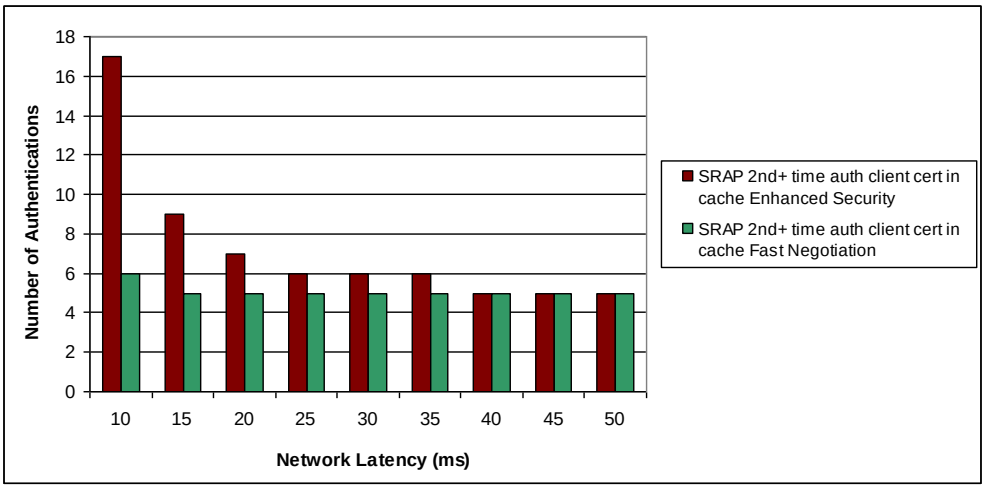

Figure 28 – SRAP cost effectiveness for computer 1

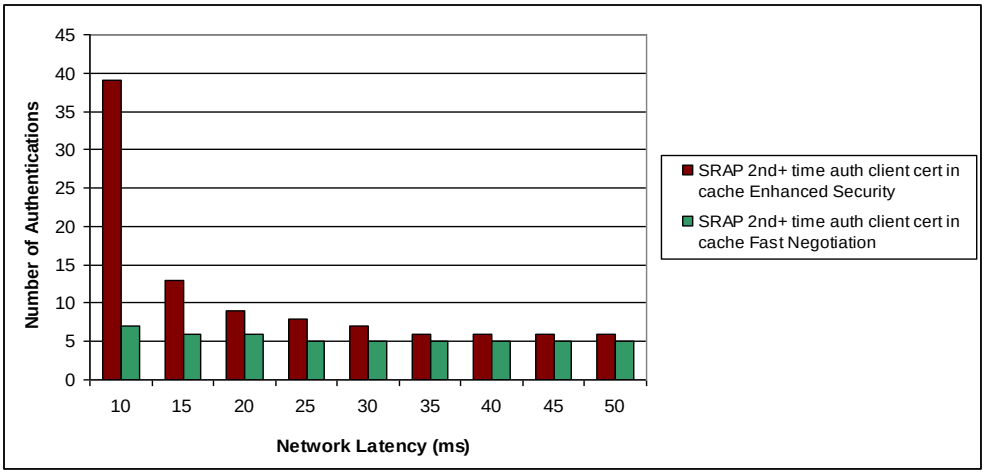

Figure 29 – SRAP cost effectiveness for computer 2

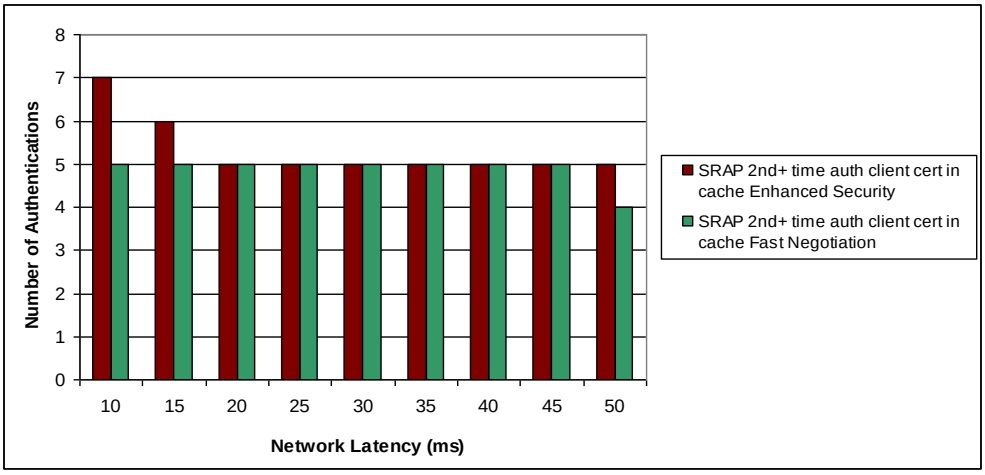

Figure 30 – SRAP cost effectiveness for computer 3

With these numbers, it is fair to assess that SRAP can be more cost effective than TLS in one day of use with Fast Negotiation. However, with Enhanced Security, SRAP may not be cost effective when the network latency drops below 10ms.

We can also deduce, from Table 10, Table 11 and Table 12, that it is very difficult for WebID to outperform TLS. Only with better than 5ms latency networks or in a LAN, WebID would be cost effective in relation to TLS.

6.6. SRAP Advantages

Even though SRAP was originally specified for semantic web applications, its use is not restricted to the semantic web domain.

Mail servers, creating a web of trust, can act as authentication partners for one another. Once they authenticate themselves and cache their certificates, they can exchange messages in a far more secure way. If a client authenticates itself using SRAP, using its mail server, the mail server may use the users' public key and a one-time symmetric key password to envelope its messages in a way only it can decrypt its messages with the user's private key.

Mobile users can take the advantage of the SRAP protocol, particularly to save battery, because of the fewer network operations required. Even if the first authentication actually consumes more battery than TLS or WebID, the subsequent authentications would be faster and would consume far less battery than the other protocols would. As shown in session 6.4, the slower the network, the more SRAP supersedes the other protocols.

Browsers can incorporate SRAP to their security suite of protocols, as an option to replace TLS on authentication and session encryption key establishment for HTTPS.

A corporation could deploy SRAP to authenticate customers, employers and telecommuters, using its servers as authentication partners. The corporation may generate its own self-signed certificates for its customers, employers and telecommuters, minimizing costs with certificates and enhancing security.

A cloud computing provider could deploy SRAP on its numerous servers and make them work as authentication partners. Like corporations, cloud computing providers may generate self-signed certificate for its users.

Social Networks may use SRAP instead of OAuth. They can also generate self-signed certificates for their users and any website that wishes to authenticate its users. Using a social network, it is only necessary to deploy SRAP and use the Social Network as an authentication partner or authentication partner of last resort, eliminating the phishing attack OAuth is vulnerable to.

Distributed Databases, especially mutidatabases[X] located in different cloud computing providers, can take a great advantage using SRAP. Considering the fact the cloud administration is out of the customers' control, authentication to the customers' databases is paramount. With multiple RDBMS (Relational Database Management Systems), running in different physical locations, each and every RDBMS can be an SRAP authentication partner. Therefore the web of trust is built by the servers themselves. Using SRAP, servers can authenticate to servers to process distributed transactions and client applications can authenticate to servers to submit transactions and retrieve results.

Governments can use their departments' public servers as authentication partners for each other, and thus creating a web of trust among themselves.

Intelligence and military uses for SRAP are also possible, especially because SRAP is able to tell us if we have been compromised or not. With this information, any targeted unit may decide whether or not to change compromised identities or to use the compromised identity to plant disinformation in the attacker.

[X] Autonomous, distributed and heterogeneous databases

7
Conclusions and Future Works

The contributions of this work are:

(i) The analysis and comparisons of the most common Semantic Web authentication techniques and (ii) the proposal of a new, safer and more cost effective authentication technique. It is hoped that with further study, a more thorough specification and with the implementation of the SRAP that the proposed technique become a standard for authentication.

Further studies need to be done on how deploy a web of trust. Authentication partners RDFK files may change from time to time and will need notification and synchronization, almost the same way a network with multiple routers using link state routing protocols does.

A software agent must be specified to query CRLs from trusted certification authorities, off-line in relation to the authentication process. Therefore, the trusted certificate chain will always be up to date, when an authentication using a partner of last resort is required. The same is also valid for compromised private keys of self-signed certificates.

Tools and frameworks to generate X.509 self-signed certificates, RDFK files and to implement the SRAP protocol need to be developed.

For mobile networks, an implementation of SRAP that uses UDP instead of TCP would be better suited.

For maximum security, SRAP could be configured to connect to at least five authentication partners and require that at least three of them vouch for the server the client is trying to connect for the first time. Likewise, the client could have a secondary personal web server, or a secondary URI for his/her RDFK file. This would mitigate even more Mallory's efforts to gain access to protected resources.

Attachment 1 – Network Operations Report

Source: www.site24x7.com

Group 1 – Social Networks

Facebook

Domain **Name** :
www.facebook.com

Location	Status	IP	acket P (%)Loss	(ms) Min RTT	(ms) Max RTT	(ms) Avg RTT	Response (Time (ms
California		173.252.112.23	0.0	75.155	76.974	75.787	75
New York		31.13.73.145	0.0	29.576	29.724	29.658	29
Toronto, Canada		66.220.152.19	0.0	129.932	148.387	140.557	140
London		31.13.64.65	0.0	7.579	7.618	7.596	7
France		69.171.242.27	0.0	85.55	85.977	85.785	85
Rio de Janeiro, Brazil		31.13.73.97	0.0	141.991	142.304	142.122	142
Munich, Germany		31.13.81.49	0.0	5.207	5.282	5.232	5
Russia		31.13.81.65	0.0	45.084	45.102	45.093	45
Tokyo, Japan		69.171.234.25	0.0	103.305	104.028	103.608	103
Sao Paulo, Brazil		173.252.112.23	0.0	131.988	132.135	132.041	132
Israel		31.13.64.17	0.0	82.066	88.662	86.344	86
Sydney, Australia		173.252.73.52	0.0	183.24	183.421	183.331	183

Average RTT: 111ms. Average TT: 55.5ms

Twitter

Domain **Name** :
www.twitter.com

Location	Status	IP	Packet (%)Loss	(ms) Min RTT	(ms) Max RTT	(ms) Avg RTT	Response (Time (ms
California		199.59.149.198	0.0	6.005	6.456	6.289	6
New York		199.16.156.198	0.0	18.618	18.746	18.668	18
Toronto, Canada		199.16.156.70	0.0	69.557	96.706	81.088	81
London		199.16.156.6	0.0	105.323	105.374	105.34	105
Rio de Janeiro, Brazil		199.16.156.102	0.0	146.25	146.364	146.295	146
Sao Paulo, Brazil		199.16.156.6	0.0	156.683	156.869	156.806	156
France		199.16.156.198	0.0	105.018	105.293	105.137	105
Munich, Germany		199.16.156.70	0.0	106.981	107.007	106.997	106
Russia		199.16.156.38	0.0	151.988	152.976	152.324	152
Tokyo, Japan		199.59.150.39	0.0	117.29	119.904	118.201	118
Sydney, Australia		199.59.150.39	0.0	224.659	224.829	224.748	224
Israel		199.16.156.70	0.0	185.925	186.146	186.016	186

Average RTT: 116.92ms. Average TT: 58.46ms

Google

Domain **Name** :
accounts.google.com

Location	Status	IP	Packet (%)sLos	(ms) Min RTT	(ms) Max RTT	(ms) Avg RTT	Response (Time (ms
California		74.125.20.84	0.0	33.977	36.398	35.406	35
Toronto, Canada		74.125.142.84	0.0	23.81	25.379	24.487	24
New York		74.125.29.84	0.0	13.27	13.277	13.273	13
London		173.194.67.84	0.0	7.475	7.559	7.505	7
France		173.194.67.84	0.0	13.825	13.878	13.85	13
Sao Paulo, Brazil		74.125.29.84	0.0	136.386	136.716	136.586	136
Rio de Janeiro, Brazil		74.125.196.84	0.0	143.767	144.387	144.022	144
Munich, Germany		173.194.70.84	0.0	5.968	6.156	6.044	6
Tokyo, Japan		173.194.72.84	0.0	37.645	40.762	39.705	39
Russia		173.194.71.84	0.0	44.336	44.623	44.521	44
Israel		173.194.70.84	0.0	69.554	75.462	71.55	71
Sydney, Australia		74.125.23.84	0.0	133.195	134.271	133.558	133

Average RTT: 55.42ms. Average TT: 27.71ms

Average Group1 TT: 43.06ms

Group 2: On-Line Storage

GitHub

Domain **Name** :
www.github.com

Location	Status	IP	Packet (%)Loss	(ms) Min RTT	(ms) Max RTT	(ms) Avg RTT	Response (Time (ms
California		192.30.252.128	0.0	65.969	71.494	69.557	69
Toronto, Canada		192.30.252.130	0.0	19.608	19.798	19.705	19
New York		192.30.252.131	0.0	6.604	43.266	29.074	29
London		192.30.252.130	0.0	85.812	89.616	87.309	87
Sao Paulo, Brazil		192.30.252.128	0.0	139.527	139.92	139.676	139
ranceF		192.30.252.129	0.0	115.409	121.377	119.374	119
Rio de Janeiro, Brazil		192.30.252.131	0.0	129.746	129.884	129.801	129
Munich, Germany		192.30.252.129	0.0	98.179	98.352	98.251	98
Russia		192.30.252.131	0.0	144.265	148.048	146.74	146
panTokyo, Ja		192.30.252.130	0.0	223.76	239.644	233.391	233
Sydney, Australia		192.30.252.129	0.0	216.389	218.905	218.053	218
Israel		192.30.252.131	0.0	145.852	148.876	147.499	147

Average RTT: 119.42ms Average TT: 59.71ms

DropBox

Domain **Name** :
www.dropbox.com

Location	Status	IP	Packet (%)Loss	(ms) Min RTT	(ms) Max RTT	(ms) Avg RTT	Response (Time (ms
California		108.160.166.148	0.0	3.657	4.173	3.923	3
New York		108.160.165.12	0.0	75.004	78.422	76.289	76
Toronto, Canada		108.160.165.20	0.0	73.393	73.568	73.456	73
London		108.160.166.142	0.0	158.955	163.918	161.3	161
France		108.160.165.20	0.0	152.482	152.674	152.587	152
Sao Paulo, Brazil		108.160.165.12	0.0	209.684	209.801	209.73	209
Rio de Janeiro, Brazil		108.160.165.12	0.0	202.078	208.523	205.085	205
Munich, Germany		108.160.166.142	0.0	157.298	158.437	157.755	157
Russia		108.160.165.147	0.0	202.337	207.784	204.927	204
Israel		108.160.166.20	0.0	212.008	213.156	212.396	212
Tokyo, Japan		108.160.165.20	0.0	128.394	149.614	135.471	135
Sydney, Australia		108.160.166.13	0.0	158.712	158.846	158.76	158

Average RTT: 145.42ms. Average TT: 72.71ms

Group2 Average TT (including Google Drive): 53.38ms

Group 3: Webmail

Yahoo

Domain Name : www.yahoo.com

Location	Status	IP	Packet (%)ssLo	(ms) Min RTT	(ms) Max RTT	(ms) Avg RTT	Response (Time (ms
California		206.190.36.45	0.0	38.712	72.478	52.061	52
Toronto, Canada		98.139.180.149	0.0	31.434	42.419	37.606	37
New York		98.139.180.149	0.0	11.518	14.024	12.495	12
London		87.248.112.181	0.0	14.967	18.511	17.328	17
Sao Paulo, Brazil		200.152.175.146	0.0	0.606	0.728	0.679	1
Rio de Janeiro, Brazil		98.139.180.149	0.0	196.39	199.933	197.982	197
France		87.248.122.122	0.0	17.475	24.173	19.727	19
Munich, Germany		87.248.112.181	0.0	25.798	26.475	26.099	26
Tokyo, Japan		111.67.226.84	0.0	35.371	35.517	35.455	35
Russia		98.139.180.149	0.0	143.235	155.329	150.846	150
Sydney, Australia		203.84.216.121	0.0	0.578	0.631	0.613	1
Israel		98.139.180.149	0.0	190.977	291.339	224.832	224

Average RTT: 64.25ms. Average TT: 32.12ms

I-Cloud

Domain Name : www.icloud.com

Location	Status	IP	Packet Loss(%)	Min RTT (ms)	Max RTT (ms)	Avg RTT (ms)	Response Ti(me (ms
California		23.193.54.46	0.0	1.548	2.089	1.863	1
Toronto, Canada		173.222.186.46	0.0	0.499	0.59	0.543	1
New York		173.222.214.46	0.0	20.548	22.794	21.297	21
London		95.100.162.46	0.0	9.011	16.745	14.079	14
France		2.20.250.46	0.0	11.686	11.757	11.713	11
Sao Paulo, Brazil		173.222.158.46	0.0	7.804	8.356	8.024	8
de Janeiro, Rio Brazil		184.25.166.46	0.0	130.54	130.638	130.577	130
Munich, Germany		172.227.14.46	0.0	5.414	5.518	5.458	5
Russia		23.61.242.46	0.0	79.563	86.479	82.972	82
Tokyo, Japan		23.2.38.46	0.0	2.222	2.359	2.305	2
Israel		2.17.254.46	0.0	87.486	88.222	87.894	87
Sydney, Australia		173.223.174.46	0.0	0.344	0.46	0.407	1

Average RTT: 30.25ms. Average TT: 15.12ms

Group3 Average TT (including G-Mail): 24.99ms

Group 4: E-commerce sites

PayPal

Domain **Name** :
www.paypal.com

Location	Status	PI	Packet (%)Loss	(ms) Min RTT	(ms) Max RTT	(ms) Avg RTT	Response (Time (ms
California		184.31.146.234	0.0	3.447	3.796	3.669	3
New York		23.4.34.234	0.0	21.401	24.125	22.438	22
Toronto, Canada		23.9.98.234	0.0	0.574	0.609	0.589	1
London		23.77.98.234	0.0	11.141	11.162	11.149	11
Rio de Janeiro, Brazil		23.200.2.234	0.0	130.737	130.924	130.818	130
France		23.52.226.234	0.0	11.496	11.577	11.53	11
Munich, Germany		172.227.98.234	0.0	5.418	5.825	5.554	5
Sao Paulo, Brazil		23.38.178.234	0.0	6.819	9.52	7.758	7
Russia		23.54.2.234	0.0	80.553	84.451	83.142	83
Tokyo, Japan		23.51.82.234	0.0	2.179	2.337	2.266	2
Israel		23.37.242.234	0.0	87.439	87.676	87.528	87
Sydney, Australia		23.37.130.234	0.0	0.26	0.428	0.331	1

Average RTT: 30.25ms. Average TT: 15.12ms

BestBuy

Domain **Name** :
www.bestbuy.com

Location	Status	IP	Packet (%)Loss	(ms) Min RTT	(ms) Max RTT	(ms) Avg RTT	Response (Time (ms
California		63.80.4.161	0.0	2.102	2.699	2.429	2
Toronto, Canada		184.84.243.48	0.0	0.571	0.64	0.604	1
New York		165.254.158.82	0.0	20.345	21.814	20.968	20
London		80.150.193.56	0.0	2.581	2.599	2.588	2
France		90.84.59.99	0.0	1.595	2.345	1.997	1
Rio de Janeiro, Brazil		23.45.65.58	0.0	114.354	114.541	114.462	114
Sao Paulo, Brazil		72.246.216.25	0.0	0.494	0.621	0.577	1
Munich, Germany		77.109.171.107	0.0	6.039	7.768	6.64	6
Russia		23.3.90.88	0.0	75.193	82.782	79.024	79
Tokyo, Japan		125.56.200.32	0.0	2.187	2.28	2.248	2
Sydney, Australia		184.84.223.155	0.0	0.327	0.424	0.382	1
Israel		2.16.219.11	0.0	69.86	70.487	70.208	70

Average RTT: 24.92ms. Average TT: 12.46ms

Mercado Livre

Domain Name :
www.mercadolivre.com.br

Location	Status	IP	Packet (%)Loss	(ms) Min RTT	(ms) Max RTT	(ms) Avg RTT	Response (Time (ms
California		216.33.196.79	0.0	73.38	74.801	74.131	74
New York		216.33.196.79	0.0	7.103	7.833	7.356	7
Toronto, Canada		216.33.196.79	0.0	19.723	20.072	19.95	19
London		216.33.196.79	0.0	77.342	78.358	77.69	77
Rio de Janeiro, Brazil		216.33.196.79	0.0	145.762	146.185	145.913	145
Sao Paulo, Brazil		216.33.197.79	0.0	151.899	152.437	152.257	152
Munich, Germany		216.33.197.79	0.0	104.42	105.66	104.859	104
Russia		216.33.196.79	0.0	142.677	147.18	144.533	144
Israel		216.33.196.79	0.0	147.276	148.182	147.681	147
panTokyo, Ja		216.33.197.79	0.0	174.955	206.31	189.848	189
France		216.33.197.79	0.0	107.977	122.232	117.243	117
Sydney, Australia		216.33.197.79	0.0	220.207	221.184	220.854	220

Average RTT: 116.25ms. Average TT: 58.12ms

Group 4 Average TT: 57.14ms

Group 4: Financial Institutions (either inside a cloud or do not respond to ICMP)
Group 5: Government Sites (either inside a cloud or do not respond to ICMP)

Attachment 2 – Source Codes of the Experiments

Diffie-Hellman Experiment

```
'  Bob will choose to use the 2nd of our 8 pre-chosen safe primes.
'  It is the Prime for the 2nd Oakley Group (RFC 2409) --
'  2048-bit MODP Group.  Generator is 2.
'  2^2048 - 2^1984 - 1 + 2^64 * { [2^1918 pi] + 124476 }
dhBob.UseKnownPrime 4
Dim p As String
Dim g As Long
'  Bob will now send P and G to Alice.
p = dhBob.p
g = dhBob.g

Dim eBob As String
Dim kBob As String
Dim kAlice As String
Dim eAlice As String
Dim dtStart As Long
Dim dtEnd As Long
Dim result As Long

dtStart = GetTickCount
'  Alice calls SetPG to set P and G.  SetPG checks
'  the values to make sure it's a safe prime and will
'  return 0 if not.
   success = dhAlice.SetPG(p, g)
   If (success <> 1) Then
       MsgBox "P is not a safe prime"
       Exit Sub
   End If

' Alice and Bob generate each a random number (RN) less than p
' and calculates g^RN mod p. The entite proccess is done by the CreateE method
   eBob = dhBob.CreateE(256) '256 bytes equals 2048 bits
   eAlice = dhAlice.CreateE(256)

'  The "E" values are sent over the insecure channel.
'  Bob sends his "E" to Alice, and Alice sends her "E" to Bob.
'  Each side computes the shared secret by calling FindK.
'  "K" is the shared-secret.

'  Bob computes the shared secret from Alice's "E":
   kBob = dhBob.FindK(eAlice)

'  Alice computes the shared secret from Bob's "E":
   kAlice = dhAlice.FindK(eBob)
dtEnd = GetTickCount
result = dtEnd - dtStart
Text1.Text = Text1.Text & "Bob's shared secret:" & vbCrLf
Text1.Text = Text1.Text & kBob & vbCrLf
Text1.Text = Text1.Text & "Alice's shared secret (should be equal to Bob's)" & vbCrLf
Text1.Text = Text1.Text & kAlice & vbCrLf
Text1.Text = Text1.Text & "Elapsed Time " & Str(result)
```

RSA Encryption and Decryption Experiment

```
success = rsa.GenerateKey(2048)
If (success <> 1) Then
    MsgBox rsa.LastErrorText
    Exit Sub
End If

'  Keys are exported in XML format:
Dim publicKey As String
publicKey = rsa.ExportPublicKey()
Dim privateKey As String
privateKey = rsa.ExportPrivateKey()
Text1.Text = publicKey & vbCrLf & vbCrLf & privateKey & vbCrLf & vbCrLf

Dim plainText As String
plainText = "Encrypting and decrypting should be easy!"

'  Start with a new RSA object to demonstrate that all we
'  need are the keys previously exported:
Dim rsaEncryptor As New ChilkatRsa

'  Encrypted output is always binary.  In this case, we want
'  to encode the encrypted bytes in a printable string.
'  Our choices are "hex", "base64", "url", "quoted-printable".
rsaEncryptor.EncodingMode = "hex"

Dim dtStart, DtEnd, Result As Long
dtStart = GetTickCount

'  We'll encrypt with the public key and decrypt with the private
'  key.  It's also possible to do the reverse.
rsaEncryptor.ImportPublicKey publicKey

Dim usePrivateKey As Long
usePrivateKey = 0
Dim encryptedStr As String
Dim cnt As Integer
For cnt = 1 To 1000
    encryptedStr = rsaEncryptor.EncryptStringENC(plainText, usePrivateKey)
Next

DtEnd = GetTickCount
Result1 = (DtEnd - dtStart) / 1000
dtStart = GetTickCount

'  Now decrypt:
Dim rsaDecryptor As New ChilkatRsa

rsaDecryptor.EncodingMode = "hex"
rsaDecryptor.ImportPrivateKey privateKey

usePrivateKey = 1
Dim decryptedStr As String
decryptedStr = rsaDecryptor.DecryptStringENC(encryptedStr, usePrivateKey)

DtEnd = GetTickCount
Result2 = DtEnd - dtStart
Text1.Text = Text1.Text & encryptedStr & vbCrLf
Text1.Text = Text1.Text & decryptedStr & vbCrLf
Text1.Text = Text1.Text & "Elapsed time Encryption : " & Str(Result1) & vbCrLf
Text1.Text = Text1.Text & "Elapsed time Decryption : " & Str(Result2) & vbCrLf

Text1.Refresh
```

AES 256 Encryption and Decryption Experiment

```
Dim password As String
password = "secretPassPhrase1234567890!@#$%&"

crypt.CryptAlgorithm = "aes"
crypt.CipherMode = "cbc"
crypt.KeyLength = 256

'  Generate a binary secret key from a password string
'  of any length.  For 256-bit encryption, GenEncodedSecretKey
'  generates the MD5 hash of the password and returns it
'  in the encoded form requested.  The 2nd param can be
'  "hex", "base64", "url", "quoted-printable", etc.
Dim hexKey As String
hexKey = crypt.GenEncodedSecretKey(password, "hex")
crypt.SetEncodedKey hexKey, "hex"

crypt.EncodingMode = "base64"
Dim text As String
text = "The quick brown fox jumped over the lazy dog."

'  Encrypt a string and return the binary encrypted data
'  in a base-64 encoded string.
Dim encText As String
Dim decryptedText As String

result = 0
i = 0
dtStart = GetTickCount
While (i < 1000) ' repeat the experiment 1000 times
      encText = crypt.EncryptStringENC(text)
      '  Decrypt and show the original string:
      decryptedText = crypt.DecryptStringENC(encText)
      i = i + 1
Wend
dtEnd = GetTickCount
result = dtEnd - dtStart
Text1.text = encText & vbCrLf & vbCrLf
Text1.text = Text1.text & decryptedText & vbCrLf & vbCrLf
Dim tempo As Double
tempo = result / i * 500 ' Two operations per loop
Text1.text = Text1.text & "Average operation time: " & Str(tempo) & vbCrLf
End Sub
```

SHA-256 Hash Experiment

```
Dim dtStart As Long
Dim dtEnd As Long
Dim result As Long
Dim i As Long
Dim hash As String
crypt.HashAlgorithm = "sha256"
Text = "The quick brown fox jumped over the lazy dog."

'  Generate a SHA-256 hash string
'  in a base-64 encoded string.
Dim encText As String
Dim decryptedText As String

result = 0
i = 0
dtStart = GetTickCount
While (i < 1000)
       hash = crypt.HashFileENC(Text)
       i = i + 1
Wend
dtEnd = GetTickCount
result = dtEnd - dtStart
Text1.Text = Text
Text1.Text = Text1.Text & "SHA256:" & vbCrLf
Text1.Text = Text1.Text & hash & vbCrLf

Dim tempo As Double
tempo = result / i * 1000 ' average time in microseconds of the hash operation
Text1.Text = Text1.Text & "Averave time of hash operation: " & Str(tempo) & vbCrLf
```

8
Bibliographic References

[1] BERNERS-LEE, Tim; HENDLER, James; LASSILA Ora. **The Semantic Web, Scientific American**, May 2001, p. 29-37.

[2] HEATH Tom; BIZER Christian**. Linked Data: Evolving the Web into a Global Data Space (1st edition)** (2011). Synthesis Lectures on the Semantic Web: Theory and Technology, 1:1, 1-136. Morgan & Claypool.

[3] BERNERS-LEE, Tim; CONNOLLY, Dan; **Naming and Addressing: URIs, URLs, ...**
Accessed on 05/07/2014
Available at: http://www.w3.org/Addressing/

[4] ABITEBOUL, S.; BUNEMAN, P.; SUCIU, D. (2000), **Data on the Web: from relations to semistructured data and XML.** Morgan Kaufmann Publishers Inc. , San Francisco, CA, USA .

[5] IETF. **RFC1422. Certificate-Based Key Management**
Accessed on 28/03/2014
Available at: http://tools.ietf.org/html/rfc1422

[6] RUSSEL, Deborah; GANGEMI, G. T. Sr. **Computer Security Basics.** O'Reilly & Associates, 1991. p. 9-11

[7] STALLINGS, William. **Network and Internetwork Security Principles and Practice.** New Jersey: Prentice-Hall, 1995. p. 10

[8] STALLINGS, William. **Network and Internetwork Security Principles and Practice.** New Jersey: Prentice-Hall, 1995. p. 11

[9] VACCA John R. Public key infrastructure: building trusted applications and Web services. CRC Press LLC (2004) p. 10-11

[10] HAJR,Layla; BA-HMAID Najlaa; Supervised by: Dr. TAHA,Yousri **Database Security** Accessed on 06/20/2013.
Available at
http://faculty.ksu.edu.sa/Taha/IS533FemalesSeminarMaterials/DataBase%20Security.ppt

[11] ASKDEFINE. **Etymology of the word cryptography**
Accessed on 06/18/2013.
Available at http://cryptography.askdefine.com/

[12] RUSSEL, Deborah; GANGEMI, G. T. Sr. **Computer Security Basics.** O'Reilly & Associates, 1991. p. 169-171

[13] SCHNEIER, Bruce. **Applied Cryptography 2nd edition.** John Wiley & Sons, 1996. p. 33

[14] SCHNEIER, Bruce. **Applied Cryptography 2nd edition.** John Wiley & Sons, 1996. p. 35

[15] SCHNEIER, Bruce. **Applied Cryptography 2nd edition.** John Wiley & Sons, 1996. p. 15-16

[16] STALLINGS, William. **Cryptography and Network Security Principles and Practice Fifth Edition.** New York: Prentice-Hall, 2011. p. 33-35

[17] SUBRAMANYA, S.R.; YI Byung K. **Digital signatures**. IEEE March/April 2006
Accessed on 11/06/2013
Available at
http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=1649003&queryText%3DS.R.+SUBRAMANYA+AND+BYUNG+K.+YI

[18] STALLINGS, William. **Cryptography and Network Security Principles and Practice Fifth Edition.** New York: Prentice-Hall, 2011. p. 257-262

[19] CAMENISCH, Jan Leonhard. **Group Signature Schemes and Payment Systems Based on the Discrete Logarithm Problem**. PhD. Dissertation. 1998. Swiss Federal Institute Of Technology Zürich. p. 14-15

[20] DAHAB, R.; LÓPEZ-HERNÁNDEZ, J.C; **Técnicas criptográficas modernas: algoritmos e protocolos.** Instituto de Computação – UNICAMP 2007 pp. 30-31

[21] STALLINGS, William. **Cryptography and Network Security Principles and Practice Fifth Edition.** New York: Prentice-Hall, 2011. p. 278-282

[22] BOYD, Colin; MATHURIA, Anish. **Protocols for Authentication and Key Establishment**. Springer-Verlag Berlin Heidelberg, 2003. p 23-31

[23] ADAMS, Carlisle; LLOYD, Steve. **Understanding PKI: concepts, standards, and deployment considerations**. Addison-Wesley Professional, 2003. pp. 11–15.

[24] MACPHEE, Allan. **Understanding Digital Certificates and Wireless Transport Layer Security (WTLS).** 2001**.**
Accessed on 10/03/2013.
Available at http://www.entrust.net/ssl-resources/pdf/understanding_wtls.pdf

[25] UGESI. **Public Key Infrastructure**
Accessed on 06/21/2013.
Available at http://www.cipher.risk.tsukuba.ac.jp/?page_id=609&lang=EN

[26] VACCA John R. **Public key infrastructure: building trusted applications and Web services**. CRC Press LLC (2004) p. 57-64

[27] WANG, Rui; CHEN, Shuo; WANG, XiaoFeng. **Signing Me onto Your Accounts through Facebook and Google: a Traffic-Guided Security Study of Commercially Deployed Single-Sign-On Web Services**
Accessed on: 02/20/2014
Available at: http://research.microsoft.com/pubs/160659/websso-final.pdf

[28] SILVA, Fernando de Freitas. **Uma nova abordagem de mineração de repositórios de software utilizando ferramentas da Web Semântica**. MsC. Thesis. 2013. Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática, Rio de Janeiro. Topic 1.1

[29] BREITMAN, K; CASANOVA, M.A.; TRUSZKOWSKI, **W. Semantic Web: Concepts, Technologies and Applications. Springer**. p. 57-64

[30] BREITMAN, K; CASANOVA, M.A.; TRUSZKOWSKI, **W. Semantic Web: Concepts, Technologies and Applications. Springer**. p. 65-66

[31] SILVA, Fernando de Freitas. **Uma nova abordagem de mineração de repositórios de software utilizando ferramentas da Web Semântica**. MsC. Thesis. 2013. Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática, Rio de Janeiro. Topic 1.1.3

[32] BERNERS-LEE, Tim. **Linked Data—Design Issues**
Accessed on 02/26/2014
Available at http://www.w3.org/DesignIssues/LinkedData.html

[33] RECORDON, David; WILLISON, Simon. **OpenID Bootcamp Tutorial**
Accessed on 06/27/2013.
Available at http://www.slideshare.net/daveman692/openid-bootcamp-tutorial

[34] WIKIA. **Phishing**
Accessed on 07/01/2013.
Available at http://itlaw.wikia.com/wiki/Phishing

[35] LEIBA, Barry; **OAuth Web Authorization Protocol**
Accessed on 07/03/2013.
Available at http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6123701

[36] IETF. **RFC6749. The OAuth 2.0 Authorization Framework**.
Accessed on 07/01/2013.
Available at http://tools.ietf.org/html/rfc6749

[37] WIKIPEDIA; **Oauth VS OpenId**
Accessed on 07/10/2013
Available at
http://en.wikipedia.org/wiki/File:OpenIDvs.PseudoAuthenticationusingOAuth.svg

[38] STALLINGS, William. **Cryptography and Network Security Principles and Practice Fifth Edition.** New Jersey: Prentice-Hall, 2011. p. 485-520.

[39] IETF. **RFC5246. The Transport Layer Security (TLS) Protocol version 1.2**
Accessed on 12/10/2013
Available at: http://tools.ietf.org/html/rfc5246

[40] MARLINSPIKE, Moxie. New Tricks For Defeating SSL In Practice
Accessed on 06/05/2014
Available at: https://www.blackhat.com/presentations/bh-dc-09/Marlinspike/BlackHat-DC-09-Marlinspike-Defeating-SSL.pdf

[41] GREGORIEV, M; IYENGAR, S; JANA, S; ANUBHAI, R; BONEH, D; SHMATIKOV, V. **The Most Dangerous Code in the World: Validating SSL Certificates in Non-Browser Software**
Accessed on 06/05/2014
Available at: http://www.cs.utexas.edu/~shmat/shmat_ccs12.pdf

[42] SCHNEIER, Bruce. **Heratbleed**
Accessed on 04/10/2014
Available at https://www.schneier.com/blog/archives/2014/04/heartbleed.html

[43] SCHWARTZ, Mathew J. **Stolen Digital Certificates Compromised CIA, MI6, Tor**
Accessed on 04/10/2014
Available at http://www.darkreading.com/attacks-and-breaches/stolen-digital-certificates-compromised-cia-mi6-tor/d/d-id/1099964?

[44] ZOLLER, **Thierry. TLS / SSLv3 renegotiation vulnerability explained**.
Accessed on 10/04/2013.
Available at http://www.g-sec.lu/practicaltls.pdf

[45] HOLLENBACH, James; PRESBREY Joe; BERNERS-LEE Tim. **Using RDF Metadata To Enable Access Control on the Social Semantic Web**
Accessed on 07/04/2013.
Available at http://dig.csail.mit.edu/2009/Papers/ISWC/rdf-access-control/paper.pdf

[46] W3C. **Foaf+ssl**
Accessed on 07/04/2013.
Available at http://www.w3.org/wiki/Foaf%2Bssl

[47] W3C. **WebID 1.0 (Web Identification and Discovery)**
Accessed on 07/04/2013.
Available at http://www.w3.org/2005/Incubator/webid/spec/

[48] STORY, Henry; HARBULOT, Bruno; JACOBI, Ian; JONES, Mike. **FOAF+SSL: RESTful Authentication for the Social Web**
Accessed on 02/27/2014
Available at http://bblfish.net/tmp/2009/05/spot2009_submission_15.pdf

[49] W3C. **WebAccessControl**
Accessed on 04/02/2014
Available at http://www.w3.org/wiki/WebAccessControl

[50] DE ARAÚJO BELCHIOR, Marion; **Modelo de Controle de Acesso no Projeto de Aplicações na Web Semântica.** MsC. Thesis. 2013. Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática, Rio de Janeiro.

[51] KAPOOR, Bhushan; PANDYA, Pramod; SHERIF, Joseph S. **A security pillar of privacy, integrity and authenticity of data communication**. P.43
Acessed on 10/06/2013
Available at http://www.emeraldinsight.com/0368-492X.htm

[52] DAHAB, R.; LÓPEZ-HERNÁNDEZ, J.C; **Técnicas criptográficas modernas: algoritmos e protocolos.** Instituto de Computação – UNICAMP 2007 p.32

[53] DAHAB, R.; LÓPEZ-HERNÁNDEZ, J.C; **Técnicas criptográficas modernas: algoritmos e protocolos.** Instituto de Computação – UNICAMP 2007 p.32

[54] HARBITTER, A; MENASCÉ, D.A. **A Methodology for Analyzing the Performance of Authentication Protocols**. ACM Trans. Inf. Syst. Secur. 5, 4 (November 2002), 458-491. DOI=10.1145/581271.581275
Accessed on 08/01/2014
Available at http://doi.acm.org/10.1145/581271.581275

[55] STALLINGS, William. **Cryptography And Network Security Principles And Practice Fifth Edition.** New Jersey: Prentice-Hall, 2011. p 496.