Federal University of Amazonas
Computing Institute
Informatic Post-Graduation Program

An Architecture to Resilient and Highly Available Identity Providers Based on OpenID Standard

Hugo Assis Cunha

Manaus
September 2014

Hugo Assis Cunha

An Architecture to Resilient and Highly Available Identity Providers Based on OpenID Standard

Document presented to the Informatic Post-graduation Program of the Computing Institute from Federal University of Amazonas as partial requisite to obtaining the Master in Informatic degree.

Advisor: Ph. D. Eduardo Luzeiro Feitosa

Manaus
September 2014

Cunha, Hugo Assis
C972a An Architecture to Resilient and Highly Available Identity Providers Based on OpenID Standard / Hugo Assis Cunha. 2014
103 f.: il. color; 31 cm.

Orientador: Eduardo Luzeiro Feitosa
Dissertação (Mestrado em Informática) - Universidade Federal do Amazonas.

1. OpenID. 2. Tolerância a Faltas e Intrusões. 3. Sistemas Resilientes. 4. Replicação de Máquinas de Estado. 5. Infra-estruturas de Autenticação e Autorização. I. Feitosa, Eduardo Luzeiro II. Universidade Federal do Amazonas III. Título

FOLHA DE APROVAÇÃO

"Uma Arquitetura para Provedores de Identidade Resilientes e Altamente Disponíveis Baseados no Padrão OpenID"

HUGO ASSIS CUNHA

Dissertação defendida e aprovada pela banca examinadora constituída pelos Professores:

PROF. EDUARDO LUZEIRO FEITOSA – PRESIDENTE

PROF. EDUARDO JAMES PEREIRA SOUTO – MEMBRO

PROF. JOSÉ LUIZ DE SOUZA PIO – MEMBRO

Manaus, 26 de setembro de 2014.

This dissertation is dedicated to God, my zealous keeper, the precious women in my life: Yasmin and Eliza, and my parents who always sought my adequate formation.

I may not did the best, but did everything to make the best was done. I may not be the man I want to be; I may not be the man I ought to be; I may not be the man I could be; I may not be the man I truly can be; but praise God, I am not the man I once was. (Marthin Luther King)

Acknowledgements

I am proud to release my gratitude to my advisor Eduardo Feitosa, my friend and PhD student Kaio Barbosa and professor Eduardo Souto for all kind of advices, ideas and teachings. I thank to Allan Kardec for the development tips and helps with the prototype, my friends PC, Ricardo, Tux and Flávio who boosted and helped to overcome the subject's difficulties of the graduation and post-graduation. Furthermore, I thank to SecFuNet project which sustained my researches and collaborated to make it possible. Finally, I thank to my family for the support and specially my daughter Eliza, who gave me strengths to finish one more step of my formation.

Resumo

Quando se trata de sistemas e serviços de autenticação seguros, há duas abordagens principais: a primeira procura estabelecer defesas para todo e qualquer tipo de ataque. Na verdade, a maioria dos serviços atuais utilizam esta abordagem, a qual sabe-se que é infactível e falha. Nossa proposta utiliza a segunda abordagem, a qual procura se defender de alguns ataques, porém assume que eventualmente o sistema pode sofrer uma intrusão ou falha e ao invés de tentar evitar, o sistema simplesmente as tolera através de mecanismos inteligentes que permitem manter o sistema atuando de maneira confiável e correta. Este trabalho apresenta uma arquitetura resiliente para serviços de autenticação baseados em OpenID com uso de protocolos de tolerância a faltas e intrusões, bem como um protótipo funcional da arquitetura. Por meio dos diversos testes realizados foi possível verificar que o sistema apresenta um desempenho melhor que um serviço de autenticação do OpenID padrão, ainda com muito mais resiliência, alta disponibilidade, proteção a dados sensíveis e tolerância a faltas e intrusões. Tudo isso sem perder a compatibilidade com os clientes OpenID atuais.

Palavras-chave: OpenID, Tolerância a Faltas e Intrusões, Sistemas Resilientes, Replicação de Máquinas de Estado, Infra-estruturas de Autenticação e Autorização.

Abstract

Secure authentication services and systems typically are based on two main approaches: the first one seeks to defend itself of all kind of attack. Actually, the major current services use this approach, which is known for present failures as well as being completely infeasible. Our proposal uses the second approach, which seeks to defend itself of some specific attacks, and assumes that eventually the system may suffer an intrusion or fault. Hence, the system does not try avoiding the problems, but tolerate them by using intelligent mechanisms which allow the system keep executing in a trustworthy and safe state. This research presents a resilient architecture to authentication services based on OpenID by the use of fault and intrusion tolerance protocols, as well as a functional prototype. Through the several performed tests, it was possible to note that our system presents a better performance than a standard OpenID service, but with additional resilience, high availability, protection of the sensitive data, beyond fault and intrusion tolerance, always keeping the compatibility with the current OpenID clients.

Keywords: OpenID, Fault and Intrusion Tolerance, Resilient Systems, State Machine Replication, Authentication and Authorization Infra-structures.

List of Figures

List of Tables

Contents

Chapter 1

Introduction

Nowadays the extensive use of virtualization, the emergence of new concepts as Software Defined Networks (SDN) and the outsourcing of network features (e.g. IaaS [1], SaaS [2], PaaS [3]) can be pointed out as a new computing trend, which is to provide infrastructure as a service. However, as any service, these infrastructures need to control the access to their resources in order to avoid fails and attacks. To achieve this goal, they make use of identification, authentication, and authorization services, which, in this context, are responsible for one of the major challenges of future virtualized networking environments: ensure higher degrees of security and dependability.

Due this fact, Authentication and Authorization Infrastructures (AAI) are becoming critical to any infrastructure which needs features as elasticity, multi-tenacy, availability, escalability and resilience. For instance, at the application level, users are allowed to transparently access different services (e.g. Google, Facebook, Twitter, among others) using a single credential set or session. Typically, these infrastructures rely on Identity Providers (IdP) such as Shibboleth [6], OpenID [7], Persona [8], MyID [9], and others, to identify and authenticate the user.

Although these AAIs have been growing in importance, their resilience and reliability still presents open issues [10], in specially related to the availability and reliability of these services as described in [4, 10, 11, 12].

In this context, a new model of authentication based on OpenID standard is proposed. This model is capable to provide an authentication service intrusion and fault tolerant, resilient and highly available.

---

[1] Infrastructure as a Service
[2] Software as a Service
[3] Platform as a Service

1.1 Motivation

This work is a direct result from researches performed to SecFuNet project [13]. The **Sec**ure **Fu**ture **Net**works project intends to provide solutions to future networks accordingly to the computing trends, designing a cloud security framework, besides introducing authorization and authentication features for this kind of environment. Thus, the project - which encompasses universities from European Union and Brazil - presents to the scientific community several solutions addressed to security of future networks, among them, the resilient and highly available OpenID architecture as presented here. Therefore, some text blocks, images and charts presented on this document are contained on SecFuNet's technical reports, specially on deliverables 5.1 [4] and 5.2 [5].

Future networks have been designed and deployed to integrate legacy and upcoming technologies into a heterogeneous and highly dynamic resource pool. The convergence between wireless and wired networks to a single and highly performance IP network, the opening up of service providers which allow third-party and other developers to join and create new services and the emergence of cloud computing are some examples of the possibilities offered by future networks. To fulfill these goals future networks are based on virtualization and management services (e.g. authentication, authorization, monitoring, control) to allow operation of multiple distinguished networked services, tailored to different needs, on the same physical infrastructure. However, these services suffer of the same problem: the lack of resilience.

In general, resilience can be defined as the ability of system/person/organization to recover/defy/resist from any shock, insult or disturbance [14]. According to the authors, the term 'resilience' is used in many different fields. For instance, [15] used the term 'resilience' in computer database systems as the ability to return to a previous state after the ocurrence of some event or action which may have changed that state. Often the terms related to the concept of resilience are privacy, security and integrity. [16] and [17] defined resilience as the persistence of service delivery that can justifiably be trusted, when facing *changes*[4]. Several authors provide a conceptual definition of resilience but they do not quantify the resilience of computer systems. [14] propose a manner to quantify resilience using dependability attributes of systems such as availability, performance, and so on. This document does not evaluate the resilience degree of the proposed intrusion and fault tolerant Identity Provider. However, as presented in the last section of this work, the evaluation is pointed as an important future work for the resilient Identity Provider's research.

Thus, Resilience for virtual networks is closely tied with important challenges,

[4] Changes may refer to unexpected failures, intrusion or accidents.

such as those described in [18]. First, the demanding of sharing physical resources. Since virtual networks have to share the same substrate of resources (e.g. memory, processor, forwarding tables, links), failures on the physical infrastructure can affect the availability of several services. Second, the abstraction of the network is another challenge. The design of resilient mechanisms becomes hard due to certain limitations on the knowledge about the underlying structure. Third, network control and management services (e.g. open identification systems, authentication, authorization, and accounting services, monitoring systems, control plane services) are not ready to support critical failures (e.g. complete virtual networks disruption) in these new environments.

For instance, resilient authentication and authorization services face the real need of distributing the authentication and authorization process, aiming to achieve more customers and provide a more robust platform to the service providers. Therefore, the employment of resilience for these services depends on requirements and questions [19] such as:

- How to distribute the authentication and authorization process?
- How robust and resilient must the solution be? Should it tolerate intrusions? How critical is the system availability? The service needs to tolerate both logical failures (e.g. operating system crash, message corruption, software bugs) and physical failures (e.g. connectivity and energy disruptions)?
- For how long, without going down a second, should the system be up and running? What are the required liveness guarantees (e.g. avoid locks, absence of starvation, avoid service disruptions)?

Resilience of future networks still is an open issue. In practical terms, the majority of the authentication services and identity providers do not consider some properties or security features and dependability, how can be observed in [20, 21, 22]. In some cases, only SSL connections and simple replication schemes (primary-master) are used to tolerate stopping failures [10].

Although all the issues, currently the World Wide Web holds several users accessing numerous online resources and uncountable web services which promise to solve all kinds of problems. However, these services try to protect the information access, requiring specific credentials from its users. This way, the users are obliged to own and manage a set of credentials everyday to use their services adequately. Furthermore, [23] and [24] present this characteristic as a security flaw. Surely in the future, the internet will have even more web services, users, attacks and vulnerabilities. And [25] asserts that the use of distinctive credentials require an efficient and safe identity management. Thereby, nothing better that identity providers to offer their identification and authentication services to

web applications that need user credentials, reducing redundant credentials and logins through the single sign-on scheme.

Despite of the increasing importance of Identity Providers and single sign-on solving the problem of use multiple accounts, there are still many vulnerabilities which include single-point-of-failure [26], phishing [27], development flaws [28] and open problems regarding the availability and reliability of the authentication and authorization services as previously cited in [4, 10, 11, 12]. As a consequence, these services are potential targets of attacks and intrusions, which could indeed lead to critical data leakage, abnormal behavior or deny access to services and virtualized infrastructures. Furthermore, recent reports show a growing trend in digital attacks and data breach incidents [29], as well as advanced persistent threats [30, 31] are becoming one of the top priorities of security specialists.

Actually, most of IdP services do not completely address security and dependability properties such as confidentiality, integrity and availability. This can be observed on the services' online documentation and deployment recommendations [21, 22]. For instance, some OpenID implementations provide basic mechanisms to improve reliability and robustness, such as SSL communications and simple replication schemes to avoid eavesdropping and tolerate crash faults, respectively.

Considering the above-mentioned context, it seems to be clear that there is still room to develop more resilient and reliable solutions, capable of deal with more frequent and advanced attacks. Therefore, the work's main goal is demonstrating how one can design and implement more reliable and highly available authentication and authorization infrastructures based on OpenID framework through building blocks and a robust architecture.

This work is part of SecFuNet [13] project that intends to design solutions to secure future networks accordingly to the computing trends, acting on designing a cloud security framework, besides introducing authorization and authentication features for this kind of environment.

1.2 Objectives

The main objective of this work is to propose and evaluate an authentication model based on OpenID standard, which provides high availability, resilience, and furthermore, be capable to tolerate byzantine faults. Three steps were identified as crucial to achieve this goal:

- Identify mechanisms which can provide resilience, intrusion and fault tolerance as well as high availability to the authentication model

- Identify libraries, components and related which allow execute sensible operations in a securely way and ensure the resilience of the authentication model
- Develop a functional prototype which allows to evaluate the proposed authentication model

1.3 Organization of this Document

The document has been structured in order to introduce the concepts involved in OpenID Identity Providers and present a satisfactory solution. The remainder of this document is organized as follows. Chapter 2 presents some basic concepts as OpenID, distributed systems, state machine replication, intrusion and fault tolerance and related work. Following, Chapter 3 presents the main characteristics of the resilient systems and their elements like main building blocks and deployment configurations. Chapter 4 is responsible for presenting the architecture of our proposal. It presents the functional model applied to this work, the architectural configurations and the types of faults tolerated. Next, Chapter 5 brings information and details about the implementation of the functional prototype such as libraries, tools and assumptions. Then Chapter 6 presents all the performed tests (availability, performance, latency) and respective results, besides a further discussion. Lastly, Chapter 7 presents the final remarks, difficulties, contributions of the study and finally, the future works.

Chapter 2

Basic Concepts

This chapter presents some important basic concepts to base the study presented in this work. The concepts ensure the good understanding of this dissertation and its proposal. Here it will be cited some themes like OpenID standard, fault tolerance, single sign-on, state machine replication, authorization and authentication systems.

2.1 Authorization and Authentication Services

With the rising of the number of services on the web, and consequently the number of attacks to these services, the authorization and authentication systems have been became essential to ensure more security for users. Many services handle with sensible information and consequently must avoid exposure of user's information. However, keeping safe all user information and transactions is a very hard task and a big responsibility. For this reason, many service providers prefer to transfer this role to specialized organizations and services.

These authentication and authorization services aim to use several mechanisms to reach the goal, employing different mechanisms of authentication and authorization services like the physical control access, identity providers and many others. Whatever be its type, all service providers must keep the infrastructure and data free of attackers. Usually, it is achieved by authentication and authorization mechanisms.

By definition, authentication is used to establish someone's identity and authorization is used to grant or deny access to some feature or resource after the authentication. Traditionally, this kind of service is very system-centric. However, many authentication and authorization services had became more user-centric and it allows identity providers to issue digital identities instead of username and password. It gives to the user control over which security token are sent to

a web service for authentication [32]. Thereby, different types of systems have been developed to provide this kind of authentication. The OpenID is one of the most famous efforts in the open source arena.

2.1.1 OpenID

In open and broad environments, providing authorization and authentication services is not a simple task. Many variables must be considered, among them, scalability, performance and security. By definition, the respective project intends to provide infrastructure as a service through virtualization and cloud infrastructures. Due the big rising of the cloud computing, the capacity of access many services using just one set of credentials has becoming an emergent authentication mechanism. Therefore, the OpenID presents itself as a great candidate to provide a resilient, secure and scalable authentication service in the cloud and federated environments.

The OpenID uses some roles on its authentication standard in order to make more clear the understanding of the process [1]:

- The **User** - is a real user or person who is trying to authenticate against a Relying Party with his/her digital identity to enjoy some service/feature;
- The **Identifier** - is the URL that identifies the digital identity of the User;
- The **Consumer or Relying Party (RP)** - is an entity accepting an assertion from an OpenID provider. It is the actual website where the user logins using OpenID. It is called **consumer** because it *consumes* the OpenID credentials provided by the Identity Provider;
- The **OpenID Provider (OP) or Identity Provider (IdP)** - is responsible for authenticating the User against a Relying Party. The OpenID URI (Uniform Resource Identifier) points to the IdP. It is the host where the user's credentials are stored. Sometimes it is also called as *OpenID Server*.
- The **User Agent** - is simply the user's browser which he/she interacts with.

Thus, OpenID is an open standard which provides a way to prove that an end user controls an Identifier for authentication and authorization purposes [33]. This standard permits that a user has only one credentials set and be able to authenticate yourself in several web services using your identity provider. This eliminates the need of multiple identification and authentication credentials (e.g. one per service), making it easier to provide Single Sign-On (SSO) functionality [34].

OpenID is decentralized, it means that no central authority must approve or register Relying Parties or OpenID Providers. Basically, Relying Parties do not have to trust the provider, because they trust the user. The user can change his/her identity provider at any time. Nevertheless, trust on OpenID providers can be achieved using identity provider credentials (e.g. server certificates) to validate the provider's identity within federation protocols. OpenID uses only standard HTTP(S) requests and responses, so it does not require any special capabilities of the User Agent or other client software. In other words, this authentication standard is designed to provide a base service to enable portable, user-centric digital identity in a free and decentralized manner. An end user can freely choose which Identity provider to use and authenticate and prove your identity [33]. An identity is represented by a specific URL composed by the provider domain name plus the user identity parameters (e.g. unique user name within the specific provider). Hence, this forms a globally distinguishable identification scheme.

Figure 2.1 presents in details all the necessary steps to perform a standard authentication in the OpenID. Full arrows represent messages in the client-server direction and dashed arrows represent the opposite. The process starts when a user requests a service to the Relying Party (RP). Next, the RP asks to the user's identification URL (step 2). In the step 3, the user informs the identifier which is normalized by the RP as an Extensible Resource Identifier (XRI) following the syntax version 2 [35] or as an URL accordingly to the [36], depending on the information inserted by the user.

After normalization, the RP performs the Discovery (step 4) trough the Yadis protocol [37]. This step can be done in 3 ways: using a XRI identifier or an URL identifier. Both return an XRDS (eXtensible Resource Descriptor Sequence) document [38]. If the Yadis protocol fails or the returned document is invalid, the RP performs an HTML based discovery as third attempt [39].

When the Identity Provider receives a Discovery request, it responds accordingly (step 5). In the majority of the cases, a XRDS document is returned with information valuable to the Service Provider, e.g. the list of OpenID Server's endpoint URLs, supported version of the OpenID protocol, whether server supports attribute exchanging or not, among others.

After receiving the document, the RP extracts some information from the XRDS and (optionally) requests an Association (step 6). The SP tries an association with each of the endpoints URL until one works successfully. The association aims to establish a shared secret between the Consumer and the Identity Provider [32] to add some security to the OpenID communication. In the Association request the Service Provider sends the endpoint URL used in the request, the Diffie-Hellman data (public key, modulus prime number and generator num-

ber) if used, among others. In the sequence of the process, the server responds sending its public key and then establishes the association with the Consumer keeping the association handle (unique association identifier).

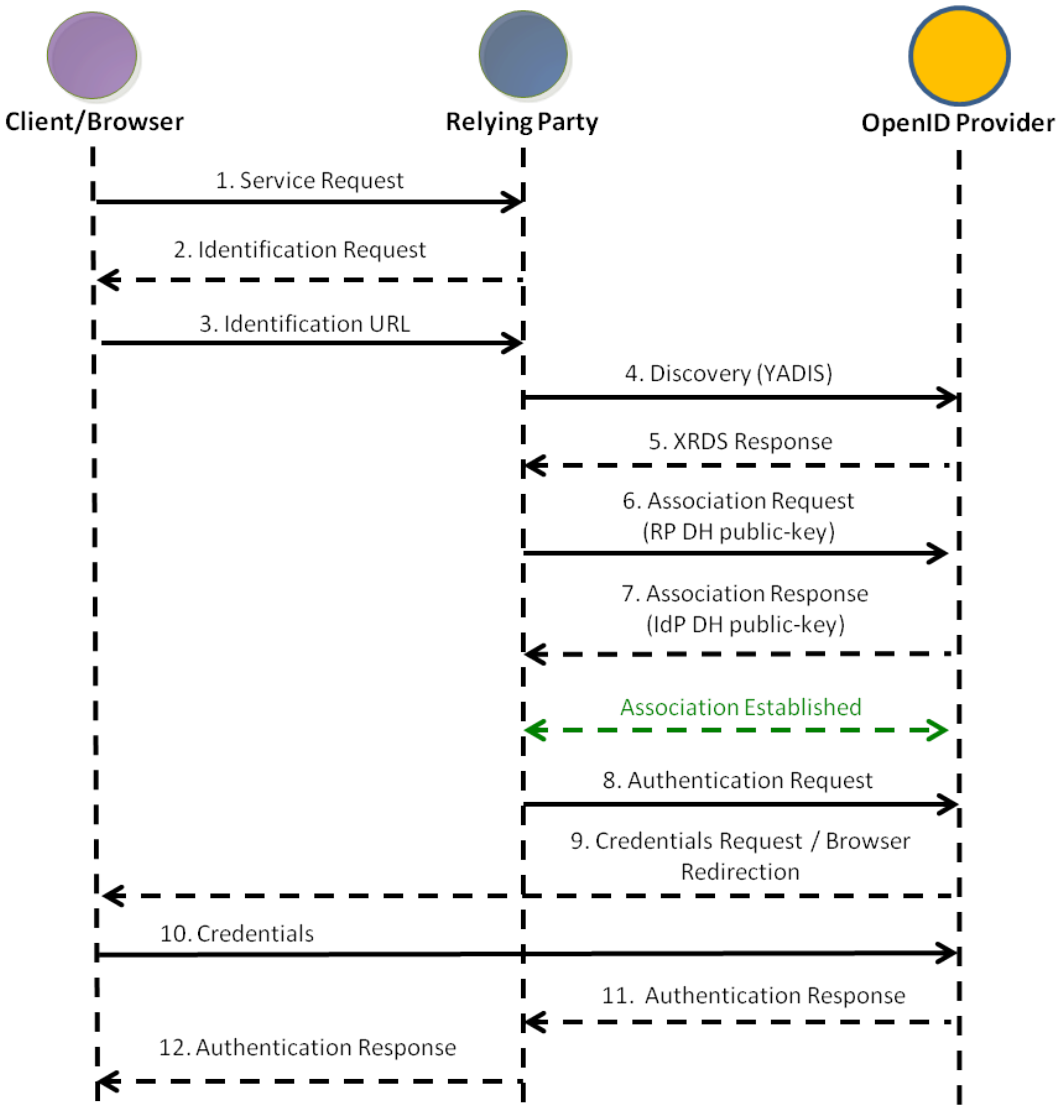

Figure 2.1: Necessary steps to a standard OpenID authentication.

In step 8, RP sends an authentication request to the OpenID Server, which redirects the user's browser to a form (step 9) in order to get user's credentials and performing the real authentication. Then, user informs your credential (step 10) and the server sends back (step 11) an assertion (positive or negative) to Service

Provider along with the nonce[1], the association handle and other information, all used in the signature of the authentication response. Lastly, the RP verifies the authentication response, its signature and its nonce expiration to deliver to the user the permission to access the desired service or not (step 12).

Table 2.1 presents all the OpenID parameters sent in the flow described in Figure 2.1. The following parameters are sorted by request type. Some parameters are more critical than other because they are optional on requisitions allowing breaches to parameter forging and so on. More details are presented in OpenID specification document [39].

Table 2.1: OpenID standard requests and respective parameters.

	Parameter name	Mandatory	Stored value
Association Request	openid.ns	No	OpenID version number being used for a particular message
	openid.mode	Yes	Type of the traveling message
	openid.assoc_type	Yes	Algorithm used for signing the message
	openid.session_type	Yes	Type of encryption of MAC key
	openid.dh_modulus	No	Prime number of Diffie-Hellman agreement
	openid.dh_gen	No	Generator number of Diffie-Hellman agreement
	openid.dh_consumer_public	No	Consumer's Diffie-Hellman public key
Association Response	openid.ns	SAA[2]	SAA
	openid.assoc_handle	Yes	Unique identifier of the association and the key used for encryption/decryption of the respective association and related
	openid.session_type	SAA	SAA
	openid.assoc_type	SAA	SAA

[1]In information security environments, Nonce is often a random or pseudo-random number used only once. It is issued in an authentication to ensure that old requests/packages cannot be reused in replay attacks.

[2]SAA - same as above

Table 2.1: OpenID standard requests and respective parameters.

	Parameter name	Mandatory	Stored value
	openid.expires_in	Yes	Lifetime of the respective association
	openid.mac_key	No	The non-encrypted but base-64 encoded MAC key in the case of "openid.session_type" was "no_encryption"
	openid.server_public	No	The Identity Provider's Diffie-Hellman public key
	openid.enc_mac_key	No	The Encrypted MAC key
Authentication Request	openid.ns	SAA	SAA
	openid.mode	SAA	SAA
	openid.claimed_id	No	The user's claimed identifier, which is not yet verified
	openid.identity	No	OpenID Provider local identifier of the user. If it is not specified, it must receive the claimed_id value
	openid.assoc_handle	No	SAA
	openid.return_to	Yes	URL that OpenID server will use to send the response back to the Consumer"
	openid.realm	No	URL which can be used by OpenID servers to identify a Consumer in an unique way and it may contain wildcards like "*"
Authentication Response	openid.ns	SAA	SAA
	openid.mode	Yes	The type of authentication response, indicating a successful or not authentication
	openid.op_endpoint	Yes	The OpenID server URL
	openid.claimed_id	SAA	SAA
	openid.identity	SAA	SAA

Table 2.1: OpenID standard requests and respective parameters.

	Parameter name	Mandatory	Stored value
	openid.assoc_handle	Yes	The association handle used to sign the message. It uses to be the same of the handle sent in the authentication request
	openid.return_to	SAA	SAA
	openid.response_nonce	Yes	A timestamp in UTC format and additional ASCII characters used to avoid replay attacks and must be unique for each message
	openid.invalidate_handle	No	Used to show if the handle attached with the request was valid or not
	openid.signed	Yes	The list of parameters that are signed
	openid.sig	Yes	The message signature which is base-64 encoded

OpenID Issues

Although OpenID standard have been developed to avoid the password fatigue [3] problem [40], like any other service, it has some vulnerabilities. Some of them are related to the protocol itself, and others are linked to bad implementation of Relying Parties or OpenID Servers.

In [1], the authors show that the simple misuse of the HTTP and HTTPS can create vulnerable points in the OpenID authentication process. For example, if the OpenID Provider or the Relying Party is addressed via HTTP, they simply redirect the request to HTTPS equivalent and proceed with the protocol flow. This vulnerability opens breaches which attackers performing parameter injection and parameter forgery attacks can use. The first one permits an attacker invalidates an user's authentication injecting some parameters not solicited by the RP in the original request and therefore making authentication's signature not match in the Relying Party. But, if the attacker replaces a non-required parameter for any other, the RP will not detect that the packet was violated and

[3] In information security, password fatigue is a security problem generated by excessive amount of passwords to remember or handle. It is very common due number of services available on the web.

modified (another flaw). The second one is that an attacker can modify any parameters (with some few exceptions) he/she wants when working in combination with the parameter injection attack. Other authors also present the OpenID as vulnerable to parameter forgery attacks like [1] and [28].

This last attack is possible because the "semi-effective" HTTPS redirection as cited previously. However, these attacks only work in combination with another vulnerability of OpenID protocol: the Man-in-the-Middle (MITM) attack. Figure 2.2 presents an example of attack combining the cited vulnerabilities. In the example, Relying Party mounts an authentication request containing some attributes on the optional field (e.g. nickname, date of birth and other) and required field (e.g. email). Next, RP sends the request to Identity Provider. However, an attacker (MITM) intercepts the message and removes optional and required fields from the request and forward it to IdP. The Identity Provider receives the authentication request, process it and answers back. Nevertheless since the adversary removed the required field (e-mail attribute) from the original request, IdP's response will no contain any e-mail information on the packet's signature. Thus, as presented in the figure, the attacker can easily inject any parameter in the required field of the IdP's response and RP will not perceive the forged parameter. Additionally, the packet will seem intact once the signature is perfectly certifiable. At this point, the adversary sends back the forged response to RP and the attack is accomplished. In other words, the attacker could easily inject any parameter value in the OpenID conversation as exemplified by Figure 2.2.

Besides the previous attacks, the OpenID standard is vulnerable to phishing attacks because an attacker can easily make a website that looks like an original Service Provider and redirect the user to a malicious Identity Provider. OpenID also has others vulnerabilities as pointed by [24] as Cross-site Request Forgery attacks (CSRF), impersonation attacks, DoS and replay attacks. For instance, [41] demonstrated how a fast network attacker could sniff an authentication response and reset the user's TCP connection to masquerade as that user (impersonation). The authors also demonstrated that an MITM attacker between the RP and IdP could perform two distinct DH key exchanges with each party to sign authentication assertions on behalf of the OpenID Provider. [42] presents the login CSRF, in which an attacker logs the victim into a site as the attacker by using the victim's browser to issue a forged cross-site login request embedded with the attacker's user name and password. The authors showed that OpenID protocol is also vulnerable to Swapping Session attacks. A lot of possible vulnerabilities are presented in the OpenID specification itself, like the replay attack vulnerability which can be exploited by the lack of assertion nonce checking by Relying Parties and Denial-of-Service attacks used to exhaust the computational resources of

Figure 2.2: Man-in-the-middle attack example [1]

RPs and OPs, beyond the previous cited attacks. Finally, all these vulnerabilities and a little bit more is summarized by [43] in their OpenID review website.

Although all these vulnerabilities, [24] presents some techniques to avoid CSRF attacks based on [42], [44] and [45] studies. But even using all the cited techniques, [24] show that all of them have flaws and can be bypassed. This among other reasons boosts the study of the intrusion and fault tolerance, once this approach basically does not tries to prevent intrusions and fault, but tolerate them.

2.2 Intrusion and Fault Tolerance

The classical security-related works and solutions have preached, with few exceptions, intrusion prevention, attack prevention, or intrusion detection, in most cases, without systematic forms of processing the intrusion symptoms. In other words, the classical security is not prepared to handle successful malicious attacks or successful intrusions [2]. But as well known, there is no system completely safe

or that can be fully protected to prevent intrusions. There is no silver bullet when the subject is systems security. It means that does not matter what we do to protect or keep untouchable a system, it always will have a chance of be invaded or compromised. The main idea is add as much defense as possible to make the system too hard or painful to invade that the attacker gives up of overrunning it. But, even using such defenses, it may not be sufficient.

Thereupon, there is a different approach which has been gaining much attention of the system security researchers. This approach is called Intrusion Tolerance (IT). That is, the notion of handling (react, counteract, recover, mask) a wide set of faults encompassing intentional and malicious faults, which may lead to failure of the system security properties if nothing is done to counter their effect on the system state [2].

This paradigm assumes and accepts that systems may have vulnerabilities, intrusions or failures. And because of this, they need special mechanisms to tolerate them. In this context, it is necessary to know the main types of failures, as well as its differences:

- Byzantine failures - in the presence of these failures, the component can exhibit arbitrary and malicious behavior, perhaps involving collusion with other faulty components [46];

- Fail-stop failures - in this class of failures, the component changes to a state that permits other components to detect that a failure has occurred and then stops [47].

So, based on these concepts the intrusion and fault tolerant systems assume that attacks on components or sub-systems can happen and some will be successful. Thus, instead of trying to prevent every single intrusion or failure, these are allowed, but tolerated. It means the system triggers mechanisms that prevent the intrusion from generating a system security failure. Thereby, the mechanisms ensure that the overall system nevertheless remains secure and operational, with a quantifiable probability.

Once we want to tolerate intrusions and faults, it is necessary to understand the two underlying causes of an intrusion: the **V**ulnerability and the **A**ttack. The first one is defined as a fault in a computing system that can be exploited with malicious intention. The second one is a malicious intentional fault attempted, with the intent of exploiting vulnerability in that system. Both lead to an event of a successful attack on a vulnerability, called **I**ntrusion. The relationship between these three kinds of fault defines the AVI composite fault model. This model describes the mechanism of intrusion precisely and provides constructive guidance to build in dependability against malicious faults, through

the combined introduction of several techniques like attack prevention, vulnerability prevention, intrusion prevention and intrusion tolerance followed by its respective removal techniques. Figure 2.3 presents how the AVI work together to reach a system failure and how they can be avoided:

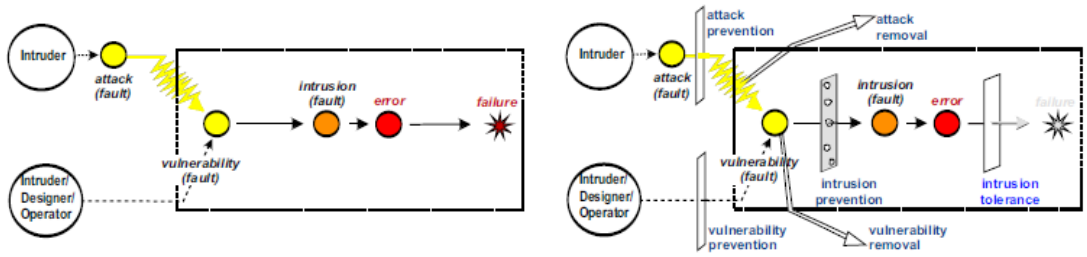

Figure 2.3: Attack-Vulnerability-Intrusion (AVI) model [2].

The AVI model is basis to determine the trust and trustworthiness degree between the elements of an intrusion tolerant architecture. Beyond the model, the intrusion tolerant architectures commonly use frameworks and mechanisms to provide fault tolerance. They are important points to be considered when building intrusion tolerant systems. In accordance with [2] the main frameworks and concerns about an intrusion tolerant system are:

1. Secure and fault-tolerant communication - it concerns the body of protocols ensuring intrusion tolerant communication. Basically, it is related to secure channels and secure envelopes. There are several techniques designed to assist this framework and the correct choice depends on the class of failures of the communication network components;

2. Use of software-based intrusion tolerance - it means tolerating software designed faults by design diversity. Thereupon, software-based fault tolerance by replication may be extremely effective at handling software faults. It is easier achieving high reliability of a replica set than individual replicas;

3. Use of hardware-based intrusion tolerance - it means to use fail-controlled components, i.e. components that are prevented from producing certain classes of errors failures. This framework contributes higher levels of trustworthiness and as a consequence achieving more efficient fault-tolerant systems;

4. Auditing - it means logging the system actions and events. It is a crucial framework in security because it allows a posteriori diagnosis of problems and their possible causes.

5. Intrusion detection - it concerns all kinds of attempts to detect the presence or the likelihood of an intrusion or error after an intrusion. It can addresses detection of erroneous states in a system computation e.g., modified files, OS penetration, among others;

6. Processing of the errors deriving from intrusions - essentially the typical error processing mechanisms used in fault tolerance are: (i) error detection, (ii) error recovery and (iii) error masking. Error detection is related to detecting the error after an intrusion is activated and it aims at confining the error to avoid propagation acting through error recovery and/or fault treatment mechanisms. Error recovery tries to recovering from the error once it is detected, ensuring the correct service despite the error. Error masking consists in masking the error through mechanisms like redundancy in order to provide the correct service without a noticeable glitch like systematic voting of operations; byzantine agreement and interactive consistency, among others.

Another very important aspect to the fault tolerance and the security fields is the distribution. Indeed, the fault tolerance and the distribution go hand in hand [2]. One distributes to achieve resilience to common mode faults, and one embeds fault tolerance in a distributed system to resist the higher fault probabilities coming from distribution. Based on the distribution, this study uses a technique known as state machine replication which will be further explained below. Although all the frameworks, strategies and techniques used to building a fault and intrusion tolerant system, its effectiveness will be determined by the good balance between the prevention and tolerance mechanisms used to avoid the failures. Following are introduced important concepts to design intrusion and fault tolerant systems like fault models, state machine replication, proactive/reactive recovery and diversity. Moreover, BFT-SMaRt will be presented since it is a fast state machine replication library used in the prototype.

2.2.1 Fault Models

Service failures can be characterized in consistent and inconsistent failures [48]. Consistent failures are perceived in the same way by all system's users (e.g. system crash, causing a temporary downtime). Whereas inconsistent failures are perceived differently by some or all users (e.g. a service with different outputs values for the same input query). The later is also known as Byzantine failure [48].

Crash and Byzantine fault models differ in the assumed types of failures. The former will generate consistent failures, which are easy to detect and perceive

by users or other processes. Byzantine faults [49] lead to inconsistent system failures, being much harder to detect, where distinct processes can have different views of the system. For instance, system intrusions can be treated as Byzantine failures [2]. An intrusion can maliciously tweak system components or resources in many ways, leading it to abnormal and unpredictable behaviors. This gives an idea of the extension of arbitrary faults.

Concepts like crash-only software have been proposed [50]. The idea is that this kind of programs will crash safely and recover quickly. In some sense, it borrows ideas from fail-stop processors [51], where a processor fails in a well-defined failure mode. Both crash-only software and fail-stop processors are supposed to stop to execute (i.e., crash safety) when a failure occurs. Further, the recovery process should be fast, using recovery protocols and restore operations from a well-defined state.

Byzantine or arbitrary fault model [49] implies in stronger and more complex mechanisms for fault detection, masking or correction. One of the approaches commonly used to mask Byzantine faults is state machine replication [52, 53]. As all replicas start on the same point (same initial state) and execute all instructions in the same order, one single faulty replica can easily be masked by the remaining replicas through a majority voting on the output. An user of the service will not even know (or suspect) when there is an abnormal replica in the system. However, techniques such as state machine replication come with an extra cost due to the protocols (e.g. consensus, total ordering, leader change) required for its operation. Thus, when designing a resilient system architecture, a good approach is to analyze and identify which services and components of the system need to tolerate Byzantine failures and which do not, i.e., on some components it could be sufficient to support crash faults or only some easy to detect arbitrary faults (e.g. message corruption).

In the context of resilient network services we can assume distinct fault models for different components of the service model or architecture. Some components can be designed assuming crash fault model, while others can mask arbitrary faults. As an example, in a network authentication service infrastructure, using secure end-to-end authentication, we may need to tolerate Byzantine faults only on the back-end service, where the authentication process is executed. All intermediate or stateless elements and components (e.g. network access servers, service gateways) can be designed to tolerate crash faults and detect simple arbitrary behaviors such as packet integrity violation. Thus, assuming there are some faulty elements, a client can try to authenticate many times, using different intermediate elements each time, until reaching the back-end service where the authentication will actually happen.

2.2.2 State Machine Replication

Nowadays, one of the major concerns about services provided over the Internet is related to their availability. Replication is a well known way to increase the availability of a service: if a service can be accessed through different independent paths, then the probability of a client being able to use it increases. The idea is to replicate a service on multiple servers so that it remains available despite the failure of one or more servers. Nevertheless, the use of replication has some costs like to guarantee the correct coordination and consistency between the replicas. But, when dealing with an unpredictable and insecure environment like Internet (the most common workplace of OpenID), the coordination correctness should be assured under the worst possible operation conditions: absence of timing guarantees and possibility of Byzantine faults triggered by malicious adversaries [54].

So, the state machine replication is a general method for implementing fault-tolerant services in distributed systems. Many protocols that involve replication of data or software - be it for masking failures or simply to facilitate cooperation without centralized control - can be derived using the state machine approach [46]. This approach achieves strong consistency by regulating how client commands must be propagated to and executed by the replicas [46, 55]. The command propagation can be decomposed into two requirements: (i) every non faulty replica must receive every command and (ii) no two replicas can disagree on order of received and executed commands [56]. Command execution must be deterministic: if two replicas execute the same sequence of commands in the same order, they must reach the same state and produce the same result.

Almost every computer program can be modeled as a state machine [46]. As everyone knows, state machine are composed by a set of states, each one with its transitions which determine the accepted inputs and possible outputs. However, because of these characteristics the state machines must present deterministic behavior. Thus, whether a system is designed based on the state machine model, it must have a set of states and its respective transitions, in other words, the same input generate always the same output. Having a system designed under this model means that all its decisions and behavior are well known, i.e. if the system presents an unexpected behavior, it means something goes wrong deriving from a malicious intrusion or something similar. Therefore it is very useful apply the state machine approach to service replication. Although state machine replication improves service availability, from a performance perspective it has two limitations. First, it introduces some overhead in service response time with respect to a client-server implementation. Second, service's throughput is determined by throughput of a single replica [56]. Thus, if the demand augments it cannot be absorbed by adding replicas to the group. So, in accordance with the

said previously, if we have a system based on state machine replication, all the replicas must receive the same sequence of messages and reaches the same state and produces the same result generating an important need of synchronization and ordering.

Furthermore, as previously explained, there is a significant difference between byzantine failures and fail-stop failures. This is important to fault-tolerant state machines, once the number of faulty tolerated replicas will be defined according to the fault model applied to the solution. Remembering that the key for implementing a fault-tolerant state machine is to ensure the replica coordination. And the more replicas, more categorical must be its control and coordination. In accordance with [46], the coordination is decomposed into two other requirements:

- Agreement - every non faulty state machine replica receives every request
- Order - every non faulty state machine replica processes the requests it receives in the same relative order

Both requirements are very important issues to state machine replication systems because the first governs the behavior of a client in interacting with state machine replicas, and the second governs the behavior of a state machine replica with respect to requests from various clients [46].

2.2.3 Proactive and Reactive Recovery

A system cannot be considered resilient if it is not capable of recovering to a correct state after a component failure. Hence, secure and dependable systems need self-healing mechanisms.

Proactive and reactive recovery techniques can help to extend the system liveness [57, 58]. On abnormal or adversary circumstances, a combination of proactive and reactive recovery can bring the system back to a healthy state, replacing compromised components. With proactive recovery all replicated components are periodically replaced with fresh new instances. On the other side, reactive recovery acts on components detected as faulty, replacing them in an on demand fashion.

Notwithstanding, proactive and reactive recovery have their effectiveness increased when combined with diversity of system components. Failures caused by non-malicious or malicious faults are likely to happen again if the same component is deployed again by the recovery mechanisms. Thus, when a component fails, its replacement should be a different version, improving the overall system robustness.

2.2.4 Diversity

Diversity has been proposed and used as one important mechanism to improve the robustness of systems designed to be secure and dependable [59, 60, 61, 62, 63]. The basic principle is to avoid common faults (e.g. software bugs or vulnerabilities). As an example, a specific operating system will have the same vulnerabilities in all instances where it is in use. To avoid this problem, each instance (a.k.a. replica) could use a different operating system. Indeed, off-the-shelf operating systems, from different families, have only a few intersecting vulnerabilities [63, 64]. Similarly, diversity in database management systems can also be very effective to avoid common faults and bugs [65]. Consequently, diverse operating systems, database systems, and other tools can difficult an attack exploring a specific vulnerability on those systems, because it might be present only in some instances of the system, but not anymore in most of them.

2.2.5 BFT-SMaRt

The BFT-SMaRt library was created by [3] and is an open-source implementation for state machine replication. The library development started at the beginning of 2007 as BFT total order multicast protocol for the replication layer of the DepSpace coordination service [66]. At 2009 was improved it became a complete BFT replication library with features like checkpoints and state transfer. And finally during a project (2010-2013) the authors improved even more in terms of functionality and robustness. Today, the library uses several mechanisms of state machine replication like fault tolerance, agreement, state transfer and reliable and authenticated channels for communication

BFT-SmaRt can be used to implement experimental dependable services and robust codebases for development of new protocols and replication techniques. This library is a complete base for state machine replication once it has all the necessary components to a fault and intrusion tolerant system based on SMR.

The library was designed taking into account some basic principles important to the authors. At first, they developed the library with a tunable fault model. For example, by default, BFT-SMaRt tolerates non-malicious Byzantine faults and also supports the use of cryptographic signatures to tolerate malicious Byzantine faults, or the use of a simplified protocol, similar to Paxos [67] in order to tolerate only crashes and message corruptions. Another important principle is the simplicity. The authors prefer to provide a reliable and correct library than a promising solution in terms of performance but too complex to ensure the correctness. Furthermore, BFT-SMaRt is a fast framework which implements the Mod-SMaRt protocol [68], a modular SMR protocol that uses a well defined consensus module in its core. The Mod-SMaRt protocol keeps a clear separation

between the protocols and system elements, unlike some systems like PBFT, which uses a monolithic protocol where the consensus algorithm is embedded inside of the SMR. Moreover, BFT-SMaRt makes use of all hardware threads supported by the replicas, especially when more computing power is needed (signatures, cryptography, etc.). This makes the BFT-SMaRt a very good tool when working with state machine replication solutions. All these features and singular properties are united in a simple and extensible API that can be used by programmers to implement deterministic services.

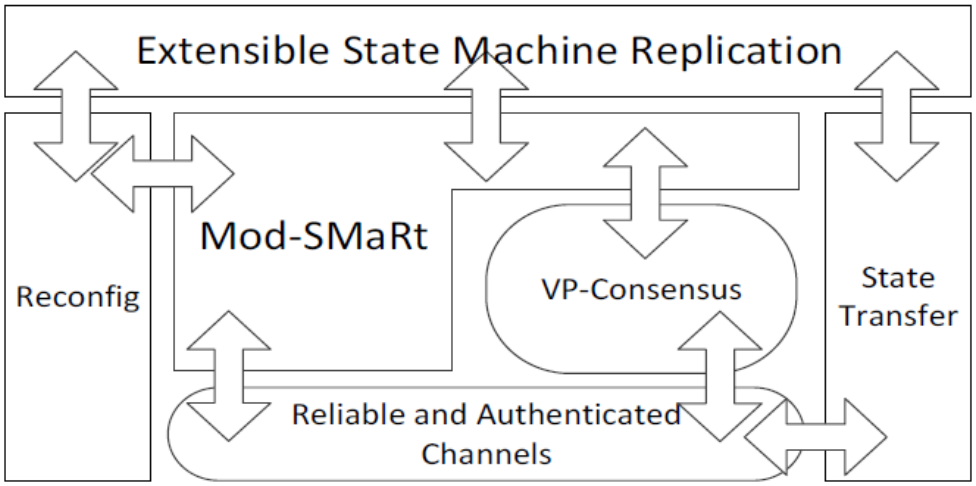

Figure 2.4: BFT-SMaRt library architecture [3].

Figure 2.4 illustrates how the BFT-SMaRt is divided and all its modularity that encapsulates the SMR complexity and the protocols used to assure the fault and intrusion tolerance, the correct communication and coordination between the replicas, as well as the consistency of the state machine replication.

The first of the protocols used by the library is the responsible for the total order multicast. The feature is achieved using the Mod-SMaRt protocol [68] which works together with the Byzantine **consensus** algorithm to provide the correct cast of messages between replicas. When everything is fine, the system executes in normal phase presenting a message pattern in accordance with Figure 2.5. Accordingly to the figure, when a client sends information to the distributed system, its messages are replicated to all system's replicas. Following, the leader sends to the other replicas a set of (not ordered) authenticated requests for processing (propose step). The other replicas receive the proposal and verify if the sender is the current leader, and if the proposed value is valid. At this point, all replicas weakly accept the proposal sending a *write* message to itself and other replicas (write step). If any replica receives more than $\frac{n+f}{2}$ *write* messages for the same

value, it strongly accepts this value and sends an *accept* message to other replicas. Finally, if some replica receives more than $\frac{n+f}{2}$ *accept* messages for the same value, this value is used as decision for consensus. All the cited flow is better explained in [3].

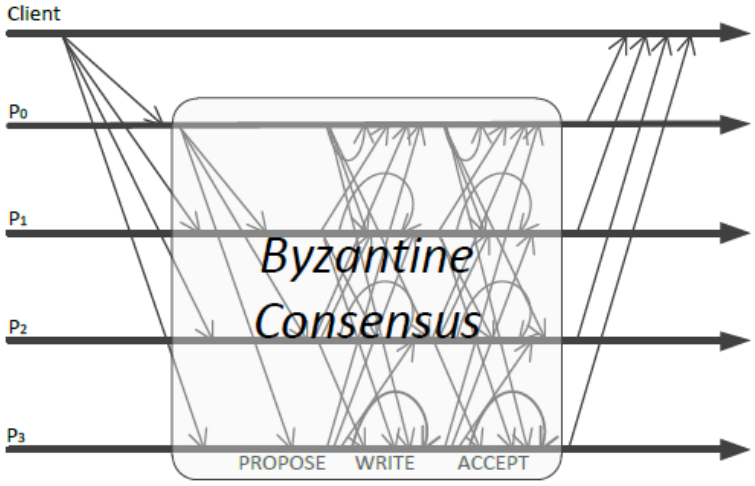

Figure 2.5: BFT-SMaRt multicast message pattern [3].

State transfer is another core protocol in BFT-SMaRt. This specific one is very important to make the replicas able to be repaired and reintegrated in the system without restarting the whole service. The protocol can be activated in four specific situations described below:

1. A replica crashes but it is later restarted
2. Replica detects that it is slower than the others
3. A synchronization phase is triggered
4. A replica is added to system while it is running

The last core protocol used in BFT-SMaRt is a great differential from the previous BFT SMR systems. Unfortunately, they assume a static system that cannot grow or shrink over time. On the other hand, BFT-SMaRt allows replicas be added or removed without stopping the system and updating the number of failures tolerated in the system at runtime.

BFT-SMaRt was developed in Java for several reasons advocated by the authors. Among them we have that Java is a type-safe language with large utility API, no direct memory access, security manager, portability, etc. It makes the implementation of secure software more feasible, say the authors. However, Java

is also well known for cripple the performance of the systems implemented on it. But, despite of using Java, BFT-SMaRt library shows one of the highest performances when compared with other SMR systems like PBFT [52], UpRight [69] and JPaxos [70]. The tests were made using two BFT-SMaRt setups. The first one used for byzantine fault tolerance and another one for crash fault tolerance in order to show how the library behaves in both scenarios. Fortunately, BFT-SMaRt presented the best performance even over C-based systems as presented in Table 2.2.

Table 2.2: Test results of BFT-SMaRt library [3].

System	*Throughput*	*Clients*	*Throughput 200*
BFT-SMaRt	83801	1000	66665
PBFT	78765	100	65603
UpRight	5160	600	3355
CFT-SMaRt	90909	600	83834
JPaxos	62847	800	45407

2.3 Related Work

Despite the existence of different solutions and components that can be used to improve the security and dependability of Authentication and Authorization Infrastructure (AAI) services, such as advanced replication techniques, virtualization, proactive and reactive recovery techniques and secure components, there are no methodologies, functional architectures or a set of system design artifacts that are capable of demonstrating how different elements can be orchestrated to build highly available and reliable systems. Existing approaches and solutions are designed for specific scenarios or a particular system. One example is to use TPMs to create trustworthy identity management systems [71]. While the solution allows one to create trustworthy mechanisms for issuing tickets, it is not designed for high availability or fault and intrusion tolerance.

On the other hand, many authors have researched more user-centric and adequate approaches to work well on high demand scenarios. SSO is one of the existent approaches to provide high availability services. Although this approach being a pretty old solution to the necessity of log in several accounts over the Internet, it was not very used some years ago. As we can see, big enterprises as Google and Facebook have adopted and make available their SSO capabilities. Now, many websites allow users log into their domains by Google's and Facebook's identity capabilities.

Among the several SSO solutions, the OpenID still presents a relevant investigation about security schemes for authentications. For example, [72] proposed an authentication scheme which uses two kinds of password, a fixed one and another temporary one, respectively. [73] proposes a strong authentication for OpenID, based on SSL smart cards with no passwords. [74] analyses the OpenID protocol and propose some improvements on the authentication scheme.

Although all these papers propose some improvements on the OpenID authentication schemes, they suggest difficult changes to adopt. Furthermore, the majority of these incremented authentication schemes solves security problems on the client side and forgets the server side, which keeps it always vulnerable to intrusion attacks and byzantine faults. Furthermore, inside the OpenId authentication context, the client side probably will be the last participant to obey these security adjustments, simply because they are users. However, some studies on the literature aim to make the OpenID server more fault tolerant, but in a little proportion.

[75] proposed a fault tolerant OpenID infrastructure and authentication using P2P networks, where each node of the infrastructure executes its own OpenID server. Although the authors propose a fault tolerant solution with high availability, clearly, it has a serious reliability flaw, once the attacker can use any of the OpenID vulnerabilities and compromise the authentication system as a whole. Of course availability is a very important issue on this field of study, but it is not the ultimate solution.

Another known solution uses TPM (Trusted Platform Module) to improve the OpenID identity security [71]. The authors use the TPM to replace the login/password pair and verify the integrity of the signed credentials and avoid phishing attacks. But, even the TPM offering some intrusion tolerance level, singly TPMs are vulnerable to TOCTOU[4] attacks [76], Cuckoo [77] and DDoS. Moreover they are not scalable to environments that have high demand of service request as OpenID [78].

Despite many OpenID researches have been done, the majority of them are focused in resolve vulnerabilities addressed to the protocol, to implementations flaws, or punctual security adjustments to make the OpenID transactions safer. However, to our knowledge, fault- and intrusion-tolerant Identity Providers have not being deeply investigated yet. Moreover, trustworthiness assessments of different elements of an OpenID infrastructure (e.g., clients and servers) have also not yet been addressed by existing solutions.

Unlike the many researches related to the standard OpenID vulnerabilities, when talking about intrusion and fault tolerance, to our knowledge, there are only two related work that address this kind of problem, a RADIUS-related and

[4]Time-of-check Time-of-use

another OpenID study. Despite of the fact that the latter proposes an intrusion-tolerant IdP as well, our approach provides further properties such as higher availability and arbitrary fault tolerance on the entire OpenID service.

The first, a resilient RADIUS [79] uses traditional state machine replication techniques for tolerating arbitrary faults. Additionally, the solution proposes software-based (isolated) trusted components to store all session keys and tokens of the services. These components are used to ensure the confidentiality of the system in case of intrusions.

The second is an intrusion-tolerant OpenID IdP [80] called OpenID-VR. However, it only tolerates intrusions regarding the OpenID protocol. The solution uses a state machine replication approach based on a shared memory provided by the hypervisor. In other words, virtual machines share this memory for communication purposes. Additionally, the agreement service, required for state machine replication protocols, is simply considered as a secure component because it is implemented at the hypervisor level. In other words, the hypervisor is assumed to be a trusted computing base. The main assumption of OpenID-VR is that the shared memory can significantly reduce the overhead imposed by message passing protocols, as it is the case of the resilient RADIUS and our proposal of a resilient OpenID service. Besides, OpenID-VR uses a single authentication service, based on secure processors (e.g., smart cards), for authenticating users. Therefore, the hypervisor, the agreement service and the authentication server are single points of failure of the system architecture.

Differently from OpenID-VR (Virtual Replication), our solution OpenID-PR (Physical Replication) supports from 1 up to $3f_R + 1$ secure elements, where f_R represents the threshold of simultaneous faults tolerated by the service without compromising its correctness and operation. Therefore, our solution provides higher availability when compared to OpenID-VR, resisting to different types of logical and physical faults, and supporting any type of attacks on the identity provider. Lastly, OpenID-PR supports multi-data center environments, being capable of taking advantage of the respective defense mechanisms of the infrastructures itself, such as solution for mitigating high proportion denial of service attacks [81].

For example, CloudFlare is a cloud provider which has already shown how the resources available on a multi-data center infrastructure can be used to mitigate the impact of large scale DDoS attacks [81]. By using techniques such as IP anycast [82], CloudFlare was able to support a DDoS attack of 20Gbps, with a peak of 120Gbps, for more than one week.

Chapter 3

Resilient Systems and Identity Management

This Chapter presents the main characteristics of resilient systems in general. It will be presented a generic resilient systems architecture which was used to base the resilient OpenID Provider. It describes the main resilient systems' requisites as building blocks, possible deployment configurations, among other characteristics. Moreover, it provides an overview and introduces how an OpenID service can be designed to provide more resilience.

3.1 Resilient Systems

Resilient network services can be designed by employing the concepts of hybrid distributed systems. They combine homogeneous distributed systems with smaller and proof-tampered components. These tiny and specialized components allow building systems with assured timely and security properties. Nevertheless, techniques like state machine replication are required for assuring correctness of stateful and critical parts of the system.

3.1.1 Generic Functional Model

First, the essential functional components for building resilient service infrastructures are introduced. Figure 3.1 shows a simplified and flat representation of the generic functional model. The four main elements are: (a) client; (b) target service; (c) service gateway; and (d) back-end service. In addition, the fifth component is a secure element, which can be used in conjunction with any of the other elements. Its purpose is to provide additional support for ensuring properties like confidentiality, integrity, and timing, when ever required.

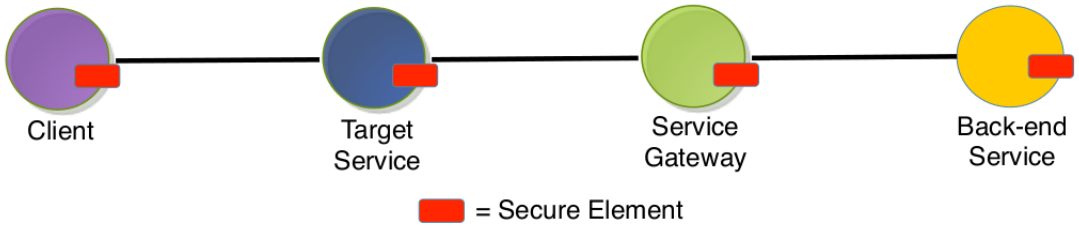

Figure 3.1: Generic functional model overview [4].

A client can be seen as user trying to get access to the network or as an OpenFlow switch willing to decide what to do with a new network flow, for instance. It represents a generic element, capable of representing different things depending on the target service infrastructure.

Target services are generic elements as well. In a typical network infrastructure it can represent components like wireless routers or Ethernet access switches. Taking as example a software-defined network control plane, a target service could be an OpenFlow switch. Yet in an OpenID use case it could be almost anything, ranging from a management application used to migrate virtual networks to an access control subsystem of an online shopping site.

Service gateway is a special purpose component. Its primary functionality is to provide connection between the target service and the back-end service. As a consequence it might need to understand different protocols, from both sides, acting similarly to a network gateway. A second attribution of this component is to mask the replication protocols and mechanisms used to deploy resilient and trustworthy back-end services.

Last, but not least, the back-end service represents a critical service of the infrastructure, with higher properties of security and dependability. These services might need to resist to arbitrary faults. Furthermore, assure correct and safe operation of components despite the presence of intrusions. AAA services, OpenID providers, monitoring systems and OpenFlow controllers are examples of highly critical systems for network infrastructures. Any failure on those services can have a huge impact on the network operation, with potential negative impact on users and business.

Finally, the secure component can add extra properties, such as data confidentiality, integrity checks, and timing assurances to specific parts of the other elements. As an example, the user keys can be stored inside a smart card. Similarly, the CA and server keys can also be securely stored inside secure elements. Further, all crypto operations can be safely executed by these trusted components, without compromising critical material even in the presence of an intrusion.

Figure 3.2 presents a more detailed overview of the functional components

in resilient services architecture. It can be observed that one of the ideas is to provide different fault thresholds (i.e. independent fault limits between different types of components) and security and dependability levels on different parts of the functional model. For each element, the definition of the fault model depends on the service properties or specific requirements.

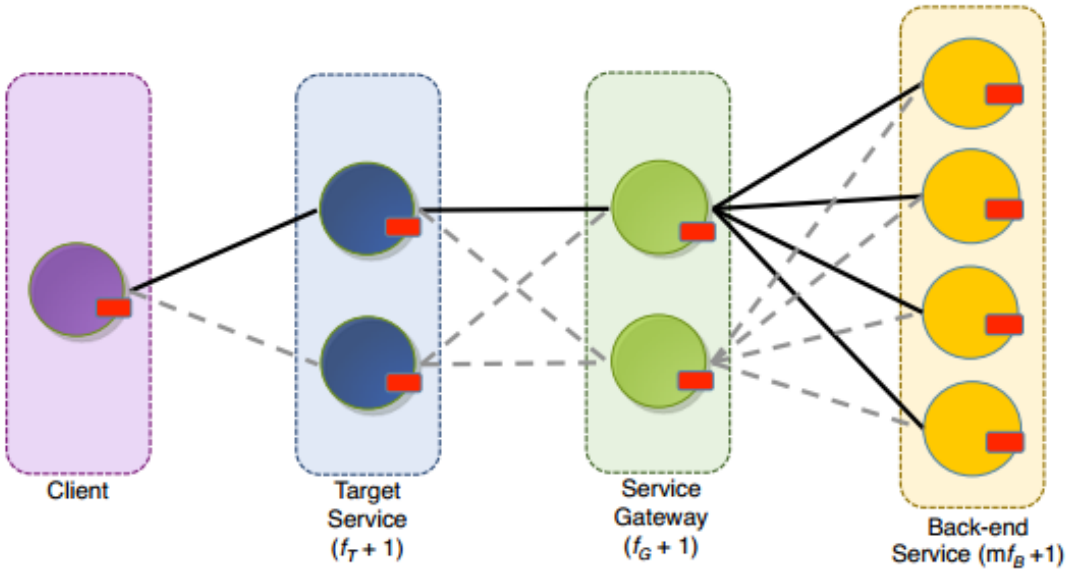

Figure 3.2: Generic architectural components overview [4].

Starting, the first element is a client. It can use a list of target service replicas to ensure dependability properties and a secure component to assure sensitive data confidentiality, for instance. Both target service and service gateway elements are assumed to tolerate crash faults and some simple to detect arbitrary faults (e.g. message integrity or authenticity), tolerating up to f_T and f_G faults, respectively. In principle, these components do not have state and can be easily replicated as needed.

Clients can connect to any target service, while a target service can connect to any of the available service gateways. The access to one of the replicas can be based on simple service lists, like it happens in AAA protocols, or round-robin for load balancing. However, in the functional model there is no strict need of load balancing, since the main goal is to provide fault tolerance. Thus, it is enough assume that components are configured with at least a simple list of replicas. Furthermore, each individual component can have a different list, varying in order or size. Nevertheless, once a replica, with which the component is connecting to, is failing to attend the requests, the next one of the list will be tried. This process goes on until end of the list. Then, it starts over again, depending on the specific

protocol timeouts and other parameters of the respective service.

In normal situations for target services and service gateways, there are at least two ways to detect a problematic replica. First, through the lack of response, using a timeout. Second, by analyzing the received responses. Once an element receives corrupted responses from a certain replica, it will try the next one. However, in some cases only timeout can be used for back compatibility reasons. On other cases it can be possible to have both approaches to detect faulty replicas.

Figure 3.3 illustrates the detection mechanisms between components of the functional model. Among clients and target services and service gateways there are detection alternatives based on timeouts or corrupted messages. Between service gateways and back-end services stronger mechanisms for arbitrary error masking are used, such as state machine replication. This means that any abnormal behavior (e.g. delays, malformed responses, correct but malicious messages) of a replica Bz will not affect or delay the system operation. For instance, malformed or divergent responses from a corrupted replica will be simply discharged. It is assumed that the majority of replicas (i.e. all replicas except f) are correct and working as expected.

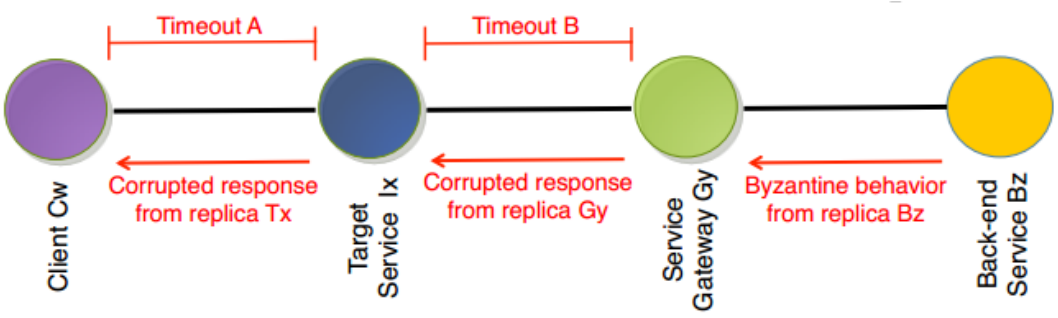

Figure 3.3: Generic fault model [4].

It is assumed that Byzantine fault tolerant protocols are used on the back-end or critical services. A gateway service will receive the responses from all back-end replicas and decide which one is the correct response that should be forwarded to the target service. To achieve this goal he back-end service requires $mf + 1$ replicas, where m refers to the specific BFT algorithm in use (e.g. 2, 3). The normal case is $m = 3$, leading to $3f + 1$ replicas to tolerate f_B faults. However, improved algorithms using trusted timely computing base components can make $m = 2$, requiring only $2f + 1$ replicas. Furthermore, other solutions such as proposed in [83] can be used to reduce the number of active replicas to only $1f + 1$.

3.1.2 Main Building Blocks

The main building blocks represent technologies and components that make it possible to conceive resilient and trustworthy network services based on the proposed architecture and functional model. Next, the five elementary units are briefly introduced.

1. **Virtual Machines**. Virtual machines represent the first building block. Nowadays, they are widely used because of their inherently simplicity and flexibility. Virtual machines easy to create, deploy, maintain, and migrate. Furthermore, the technology currently available is capable of assuring properties like administrative domain isolation, allowing multiple virtual machines, from different domains, to share the same physical infrastructure without major problems. Lastly, the virtual networks envisioned in the project are supported by extensive use of virtualization. All networks elements are virtualized by leveraging technologies such as Xen and OpenVSwitch [84].

2. **Trusted computing base**. By adopting virtualization to deploy services, the hypervisor becomes an obligatory part of the environment. It is necessary to interface between virtual machines and the underlying hardware, providing control mechanisms to ensure properties such as isolation and fairness in the resource consumption race. The hypervisor is assumed to be a trusted computing base (TCB) in the context of the functional model proposed for network services. Nevertheless, it is not requisite to trust the whole virtual machine monitor. Secure microkernel approaches [85], self-protection capabilities [86], and trusted local operations, such as start and stop virtual machines, can be assumed and ensured in a reasonable and safe way. Additionally, technologies such as secure elements can be used to implement extra verification procedures (e.g. attested hypervisor and virtual machine boot).

3. **Secure elements**. These components are small and reliable pieces of software or hardware (or a combination of both) capable of assuring critical properties or functionalities such as integrity control and data confidentiality. They can be used in different parts of the architecture. For instance, in an OpenID-based authentication solution both end user and OpenID server can trust their sensible security material (e.g. certificate, keys) and functions (e.g. all operations that need access critical data) to a trusted component. Thus, on the server-side, a compromised server will not compromise the confidentiality of the server certificate or crypto keys.

4. **Replication blocks**. Replication protocols represent one of the major building blocks of most resilient services. State machine replication and quorum systems are common approaches to mask arbitrary faults. Replicas can allow the system tolerate up to f simultaneous failures without compromising the service operation. These protocols, when combined with other building blocks and techniques like diversity and proactive-reactive recovery, represent a strong solution in the design and deployment of secure and dependable network services.

5. **Secure end-to-end communication**. Secure end-to-end communication is necessary to achieve confidentiality and privacy of user data. Protocols such as TLS and TTLS can be used to provide reliable channels, mutual authentication and server authenticity verification. These functionalities can be helpful to avoid attacks like man-in-the-middle and eavesdropping.

3.1.3 Hybrid Distributed Systems

Neither only heterogeneous nor only homogeneous distributed systems are, in a standalone way, the answer to all problems. Taking as an example an homogeneous asynchronous distributed system, there is no way to assure that consensus protocol will finish if single process is allowed to crash [87]. On the other hand, a synchronous system has too strong assumptions for a hostile environment, where arbitrary faults can happen or be exploited by malicious users. An attacker could try to compromise the timing assumptions (e.g. timeouts) of the system.

Besides the time facet, there is also the security facet. Generic trusted computing base models are not realistic on the design and development of secure systems. Secure elements (e.g. smart cards, TPMs), or trusted computing base (e.g. tiny and secure kernel systems), in practical terms, can only be verified and certified for small size components and small sets of well designed and proved functionalities. The interface and operation of such components need to be proved as tamper-proof.

The hybrid distributed systems model [88], named wormhole, proposes a different approach. Instead of having a homogeneous system, the system is designed as a composition of domains. In a hybrid model, at least two domains are needed, one with weak assumptions (e.g. asynchronous) and another one with stronger assumptions (e.g. partially synchronous). While the asynchronous domain represents the majority of components and functionalities of the system, the partially synchronous domain keeps only tiny and trusted local or distributed components (crash fault-tolerant). In other words, the wormhole concept proposes a subsystem with privileged properties that can be used to ensure critical operations in a secure and predictable way [89]. Nevertheless, the system's weaker assump-

tions and abstractions are not affected or disrupted by the smaller and trusted subsystem.

The wormhole model is of special interest in fault- and intrusion- tolerant architectures. An intrusion-tolerant system has to remain secure and operational despite some (under a measurable and predefined threshold) of its components or sub-systems are compromised [2]. To achieve these goals solutions such as a trusted timely computing base, through a distributed security kernels [90], can be conceived and used to provide trusted low-level operations. Thus, even under attack or with a compromised sub-set of processes (a malicious user inside a sub-part of the system), the system will still maintain its properties such as predictability and confidentiality.

A wormhole can itself be conceived as an intrusion-tolerant distributed system [91]. Techniques such as replication, diversity, obfuscation and proactive-reactive recovery can be used to build resilient and trustworthy wormholes.

In the proposed architecture of components for resilient network services, secure components and trusted computing base are two elements directly related with the wormhole model. Thus, the functional model and building blocks fit in the concept of hybrid distributed system model, where small parts of the system act in a predictable and secure way.

3.1.4 Deployment Configurations

The deployment configurations of resilient network services can be classified in three: (1) one single physical machine, (2) multiple physical machines in a single domain, and (3) multiple physical machines spread across multiple domains. One of the differences reside on the feasible resilience mechanisms, such as replication protocols. Whereas solutions based on shared memory (e.g. IT-VM) can be used within a single machine, message passing protocols (e.g. BFT-SMaRt) are required when multiple machines are used.

A second issue related with deployment configurations is the system availability. A service on a single physical machine, or single domain, will be inevitable affected by incidents in that domain. Events such as Internet connection disruption and power failures will affect the system availability. In some cases the local incidents can affect the system integrity as well. Corrupted disks due to repetitive power failures can compromise even the system recovery procedures. In scenarios where this kind of events can eventually happen, deploy the system over multiple physical machines spread across multiple domains can avoid such negative effects on the system operations (availability, recovery, integrity, etc.). An example of how to achieve high availability, while assuring recovery and integrity properties, is to run each replica of the service in virtual machines deployed across different cloud provider platforms.

1. **One physical machine**. Virtual machines can be used with a single hypervisor. This configuration can help to mask the effect of arbitrary behavior of some replicas. Good replicas are able to provide the service functionality, despite the presence of failed replicas. Needless to say that stronger attacks on the virtual machines, such as those trying to explore cross-channel vulnerabilities or resource exhaustion, can compromise QoS levels of the service operation. Further, physical (e.g. power failures, network failures) and logical problems (e.g. network misconfigurations) can also compromise the service availability.

2. **Multiple physical machines and a single domain**. Virtual machines of the service can be deployed on different physical machines when using replication protocols that use communication instead of shared memory. This is one step further towards more available and resilient services, when compared to the previous solution. Multiple servers allow the system to support arbitrary failures of the physical machines and infrastructure environment as well. However, it is yet a single domain. Attacks or failures (e.g. Internet link or local network disruption, energy failures) can still compromise the service availability and integrity.

3. **Multiple physical machines and multiple domains**. With these configurations it is possible to have independence of local domain failures. Using virtual machines deployed over multiple domains makes the system more robust, since it will be capable of tolerating a more broadly variety of failures. Service replicas could be deployed in different clouds infrastructures, for instance. However, the replicas do not need to be evenly distributed across a predefined number of physical machines [92]. Consequently, both in physical (e.g. network connection, power failures) and logical (e.g. misconfigurations, DoS attacks, resource exhaustion attacks) means the system will be more robust and reliable. Furthermore, it will be supported by the diversity of solutions (e.g. hardware, hypervisor, operating systems) and infrastructure protection mechanisms (e.g. intrusion prevention, intrusion detection, mitigation of DDoS attacks) of the cloud providers. In practical terms, it has already been shown how cloud providers can be capable of tolerating big threats such as huge DDoS attacks, without disrupting the customer's services [81, 93].

Figure 3.4 illustrates the performance and availability trade-offs of the deployment configurations. Additionally, it introduces a third trade-off, the susceptibility of depletion attacks. More physical machines and more domains represent potentially higher service availability, since it is much harder to attack or compromise multiple domains at once. On the other hand, less physical machines

naturally reduces the networking communication requirements, potentially improving the system performance.

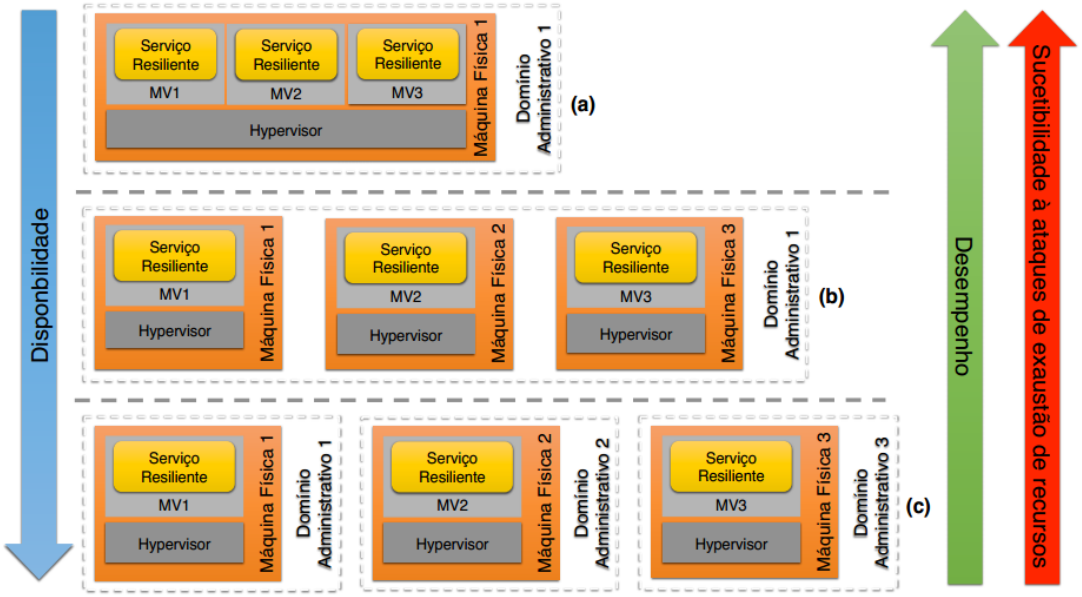

Figure 3.4: Deployment configurations. (a) One physical machine. (b) Multiple physical machines and a single administrative domain. (c) Multiple physical machines and multiple domains [4].

.

Most of the computational effort of replication and state machine replication algorithms is spent on communication, i.e., messages exchange among replicas. While virtual machines running on the same physical machine can increase the system performance through shared memory, the solution is also more susceptible to depletion attacks. In this case, both availability and performance can have a significant impact (degradation) with resource exhaustion attacks. A deployment using multiple physical machines, distributed across multiple domains, makes it much harder to degrade the service quality through depletion attacks. An attacker would have to get access to all domains and find the physical machines where the respective virtual machines are deployed. This is a tricky task and can become almost infeasible if proactive recovery and rejuvenation techniques are used to recover replicas in different locations. Diversity of location is one of the features that solutions such as those envisioned by DiverseAgent [94] and FITCH can provide [95].

Lastly, the best deployment configuration will depend on the service requirements. If performance is the main concern, one physical machine with several

virtual machines could be enough. However, in an environment with stronger requirements (e.g. availability, integrity) multiple physical machines and multiple domains could be the most suitable configuration.

3.2 Relisient Identity Management

The proposed identity management makes use of OpenID protocol [7] for establishing trust relationships among users, Identity Providers (IdPs) and Service Providers (SPs). On one hand, the OpenID protocol coupled with the use of password authentication is vulnerable to phishing attacks, in which the user is taken to a malicious SP to enter his password on a fake form that mimics the IdP [96, 97]. To eliminate vulnerabilities such as phishing attacks, this identity provider could be heavily dependent on secure hardware and/or software components and secure end-to-end mutual authentication, such as provided by EAP-TLS. In such cases, the user authentication can be done directly between the user smart card and a grid of secure processors of the IdP, using end-to-end EAP-TLS communications, for instance. Figure 3.5 gives an overview of the proposed identity provider solution. As indicated, clients and IdPs have secure elements that are used to safely carry out strong mutual authentications.

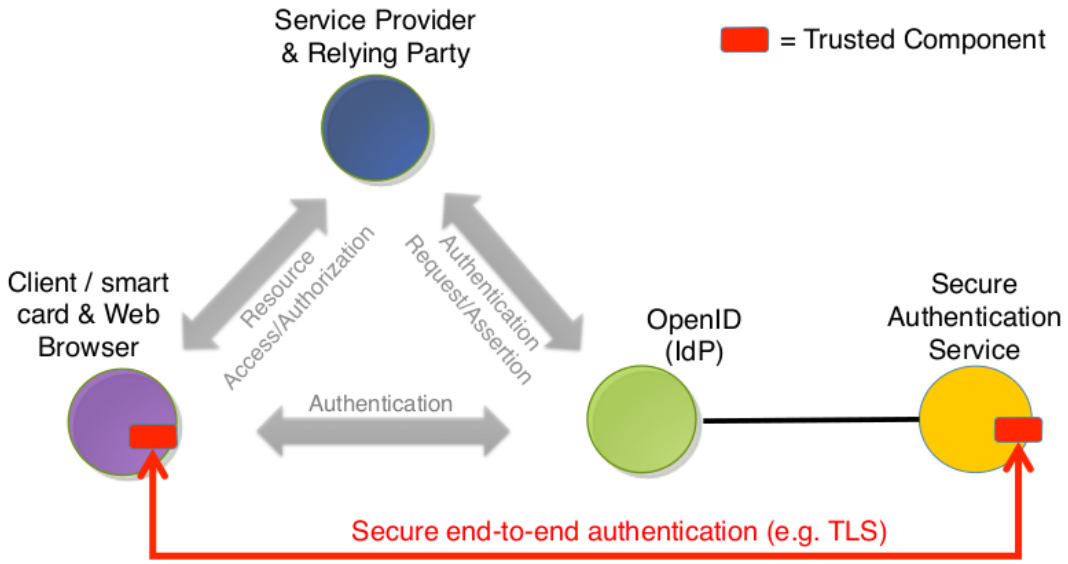

Figure 3.5: Trustworthy Identity Provider overview [5].

Additionally, the proposed work suggests new mechanisms to allow users to have more control over their attributes. The user attributes are no longer stored

on IdPs' databases, but in the user's personal smart cards. Therefore, a user can decide which of his identities or profiles (i.e., sets of attributes) will be given to the IdP and/or SP. The profile chosen will be available to the service provider during the current authentication session. In this way users can have greater control over the distribution of their attributes (user-centric attributes control). Both the authentication assertion and user's attributes are sent to service providers with attribute-based access control [7]. Further discussion and technical details can be found in previous deliverables of the project [98, 99].

The proposed identity provider model defines abstractions for integrating protocols and mechanisms of intrusion tolerance and also to keep backward compatibility with OpenID-based IdPs. Figure 3.6 extends the model presented in Figure 3.5 by introducing the resilient and trustworthy IdP. Through the two images we can have an overview of the resilient OpenID and how it works. The main elements of the extended model are: (a) a client with his web browser and user certificate in a secure component; (b) the relying party (a service provider in OpenID terms); (c) an OpenID server which does not handle user accounts (delegated to the backend); (d) authentication backend service, which is responsible of keeping the users' IDs and sensitive information in trusted components; and (e) the IdP formed by the OpenID server and the authentication service. As can be observed, it is assumed that there are secure elements both in the client-side (e.g. smart card, TPM) and in server-side (e.g. grid of smart cards, secure and isolated software component on a trusted computing base).

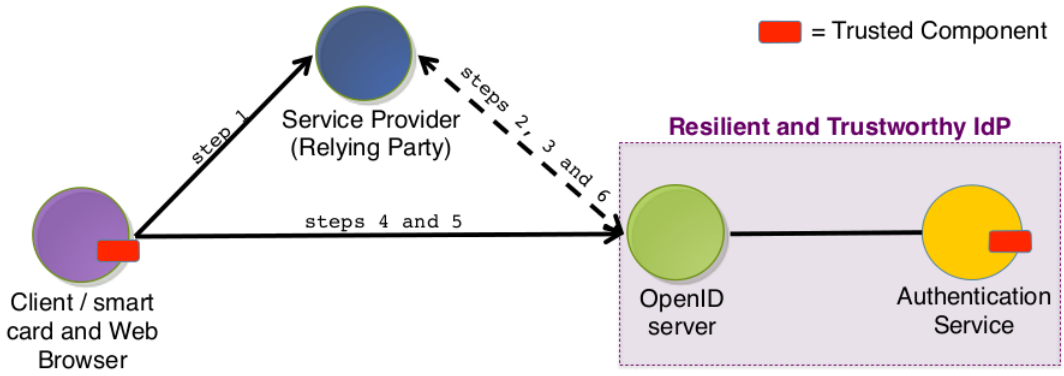

Figure 3.6: Extended trustworthy Identity Provider overview [5].

The resilience of the identity provider can be ensured through state machine replication protocols, recovery mechanisms and techniques to improve the diversity of system components. Despite using different mechanisms to tolerate faults and intrusions in the IdP, the user authentication follows the standard OpenID protocol.

Therefore, after studying several papers related to resilience and identity management, we were capable to summarize the generic aspects of a resilient identity management service. And at last, we could map this generic resilient service to an OpenID -based service. Basically, the resilient system must apply all the important concepts described in the beginning of this Chapter by using protocols designed to tolerate faults and intrusions. The literature showed that assuring the secure manipulation of sensitive data is very critical as well. However, it can be easily reached through the use of trusted components (software/hardware) as presented by the section.

Chapter 4

Proposed Resilient OpenID Architecture

There are essentially two different approaches when creating secure and resilient systems. First, we assume that we can build robust and secure enough systems for our purpose. However, it is well known that a system is as secure as its weakest link. Moreover, a system can be considered as a secured "island" until it gets compromised. Therefore, the second approach is to assume that eventually the system will fail or be intruded. With this approach in mind, we can design and deploy highly available and reliable systems by leveraging mechanisms and techniques that allow it to operate under adversary circumstances, such as non-intentional failures and attacks.

Our resilient OpenID solution is based on the second approach. In other words, we do not aim to fix all security vulnerabilities of existing identification and authentication services. Actually, we want to provide advanced techniques and resources from security and dependability properties to build fault, and intrusion-tolerant systems capable of ensuring critical properties such as integrity, confidentiality and availability.

This chapter presents all the details of the proposed OpenID Provider like its architecture, configurations and protocols, functional model, fault model, among other information. Lastly, the chapter presents the designing of trusted components for resilient OpenID servers.

4.1 Functional Model

Our proposal is based on all the system artifacts and generic functional architecture previously cited. Its functional model, along with the different technologies and protocols, allows us to design and deploy fault and intrusion-tolerant identi-

fication and authentication services, in this case an OpenID provider.

Figure 4.1 illustrates a simplified representation of the four main functional elements: (a) user browser and smartcard; (b) service and relying party; (c) IdP gateway; and (d) IdP and authentication service, i.e., the replicated OpenID service with trusted components to ensure the integrity, availability and confidentiality of sensitive data, such as cryptographic keys. This is the typical functional architecture of computing environments where identification and authentication solutions are deployed as separated services. Furthermore, the secure component can also be used in conjunction with any of the other elements, such as the relying party and the gateway. In such cases, its purpose could be to provide additional support for ensuring properties like confidentiality, integrity, and timing, when ever required.

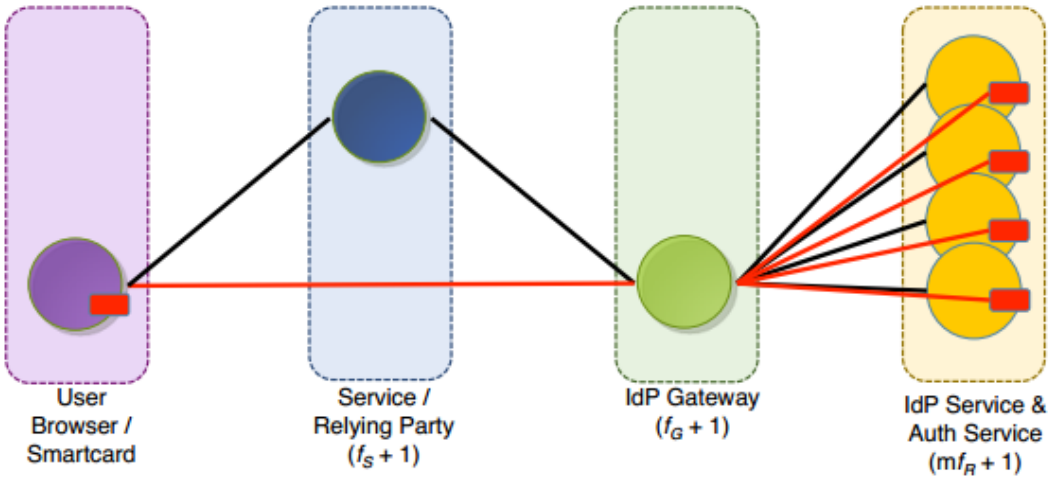

Figure 4.1: Main functional elements [5].

As already explained in this document, a client is a user trying to access an online service. This service is supported by a relying party that redirects the user for his own identification to an authentication provider. It is any service OpenID-complaint, i.e., the service can range from typical Web systems to specialized access control systems in network infrastructures.

An IdP gateway provides seamless connection between the service and/or client and the identity provider's service, i.e., the OpenID server and the authentication service inside the secure elements. Therefore, the gateway is a very simple system with two essential functions. First, it handles multiple protocols from both sides, acting similarly to a network gateway. The second attribution is to mask the replication protocols and mechanisms used to deploy resilient back-end services, providing transparent backward compatibility with existing

infrastructures which rely on OpenID providers. Maybe this is the most important role of the IdP gateway, because it makes our solution more flexible and easy to adopt.

Lastly, but not least, the IdP service (OpenID Server) is considered the infrastructure's most critical element, requiring higher levels of security and dependability properties. Such services can be part of the local domain or provided by third parties as an on-demand services, for instance. It is assumed that these back-end services must tolerate different types of faults, such as those caused by unexpected behavior or attacks, and correctly work in case of intrusions.

4.2 Architectural Configurations and Protocols

Figures 4.2 and 4.3 represent our first two envisioned configurations of the resilient OpenID architecture. As can be observed, the essential elements $f_S + 1$ relying parties, $f_G + 1$ gateways, $mf_R + 1$ replicas and secure elements. A client can use any of the available relying parties. Similarly, relying parties can use any of the available gateways. Yet, a gateway relies on at least $mf_R + 1 - f_R$ OpenID replicas. Furthermore, replicas can rely on a single, centralized, secure element, or multiple secure elements. In the second case each replica has only access to "its own" secure element, which can be running locally on the same infrastructure or remotely, on a separated system.

Regarding the main differences between these two architectural configurations, while the first can offer an improved performance if all OpenID replicas and the trust component are running on the same physical hardware, as virtual machines, the second configuration allows the IdP owner to place replicas and respective trusted components on different physical machines and/or distinct domains.

The configuration shown in Figure 4.3 is capable of providing higher levels of assurance for characteristics such as availability and integrity of the service. Furthermore, it is less susceptible to depletion attacks, as those that we describe in [4].

On the second configuration (Figure 4.3) we have one secure element per replica. This means that OpenID replicas and secure element can be distributed across different physical machine or event administrative domains. This naturally increases the availability and robustness of the system because it will be able to tolerate a wider range of benign and malign faults such as energy and connectivity disruptions, disk failures and even attacks such as DDoS. For instance, if each replica is running on a different cloud infrastructure, the system will be able to take advantage of a diverse range of protection mechanisms. As an example, some cloud providers have already shown their capabilities of dealing with DDoS

attacks of great proportions [81, 93]. Furthermore, each OpenID replica has to communicate only with its local (or nearest) secure element. It is also worth mentioning that an increased number of replicas and replicated secure elements can be used to augment the system throughout, i.e., increase the number of authentications per unit of time.

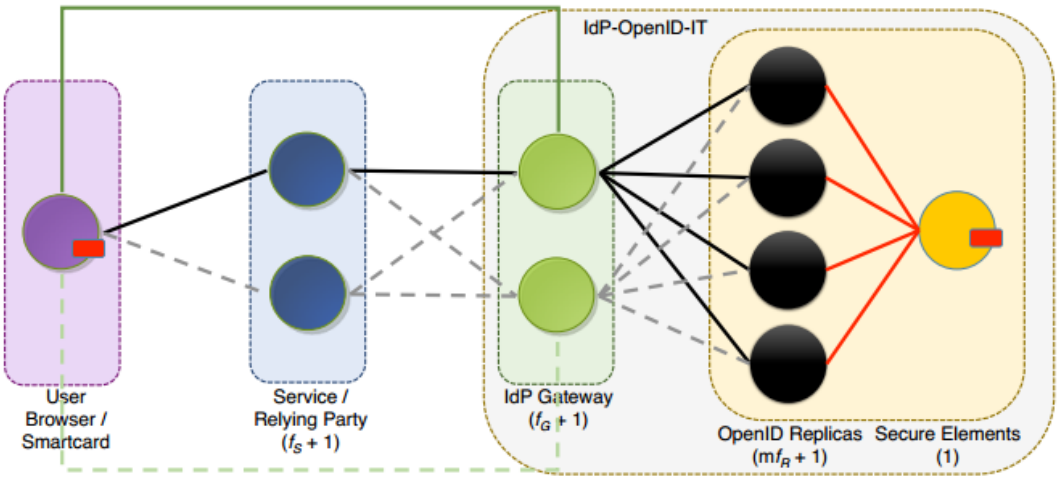

Figure 4.2: First OpenID configuration with a single centralized TC [5].

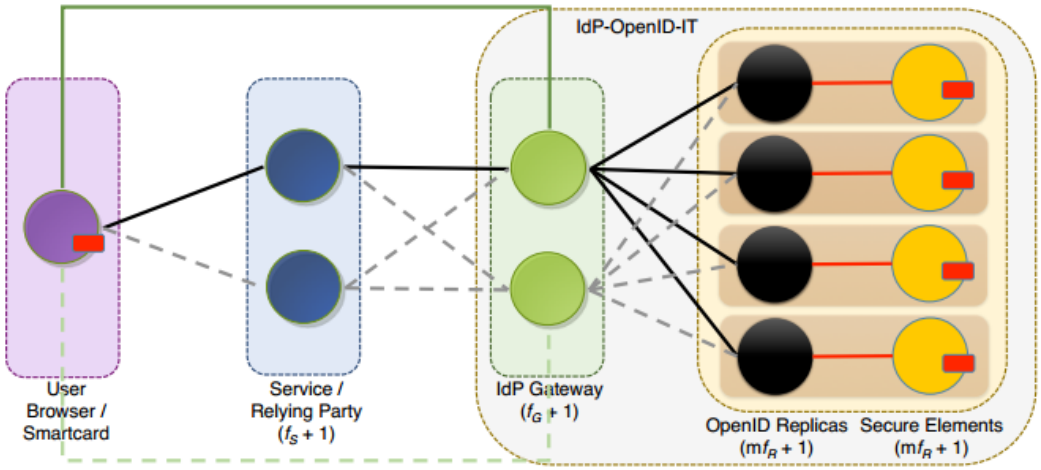

Figure 4.3: Second OpenID configuration with multiple TCs [5].

Our idea is to design and implement both architectural configurations and evaluate their benefits and/or drawbacks. More specifically, we intend to analyze the impact of having a single and/or replicated trusted component.

Lastly, Figure 4.4 shows the protocol layout of a typical OpenID deployment and our resilient OpenID. The protocol stack does not change, as can be observed. Essentially, we still have the traditional protocol layers, i.e., HTTP and SSL (or TLS for mutual authentication). The difference between them is the gateway element, which will simply encapsulate the HTTP/SSL packets in a Byzantine fault-tolerant (BFT) protocol. Consequently, a resilient OpenID identity provider can easily replace an existing, traditional, OpenID-based identity provider. A client, or relying party, will not notice any functional or operational difference between a non-resilient OpenID service and a resilient one. However, the latter improves the security and dependability of the IdP infrastructure, i.e., it is potentially more interesting for both, providers and users. It is important to make clear that Figure 4.4 simply illustrates the protocol stack, i.e. the appearing order of the protocols does not mean anything.

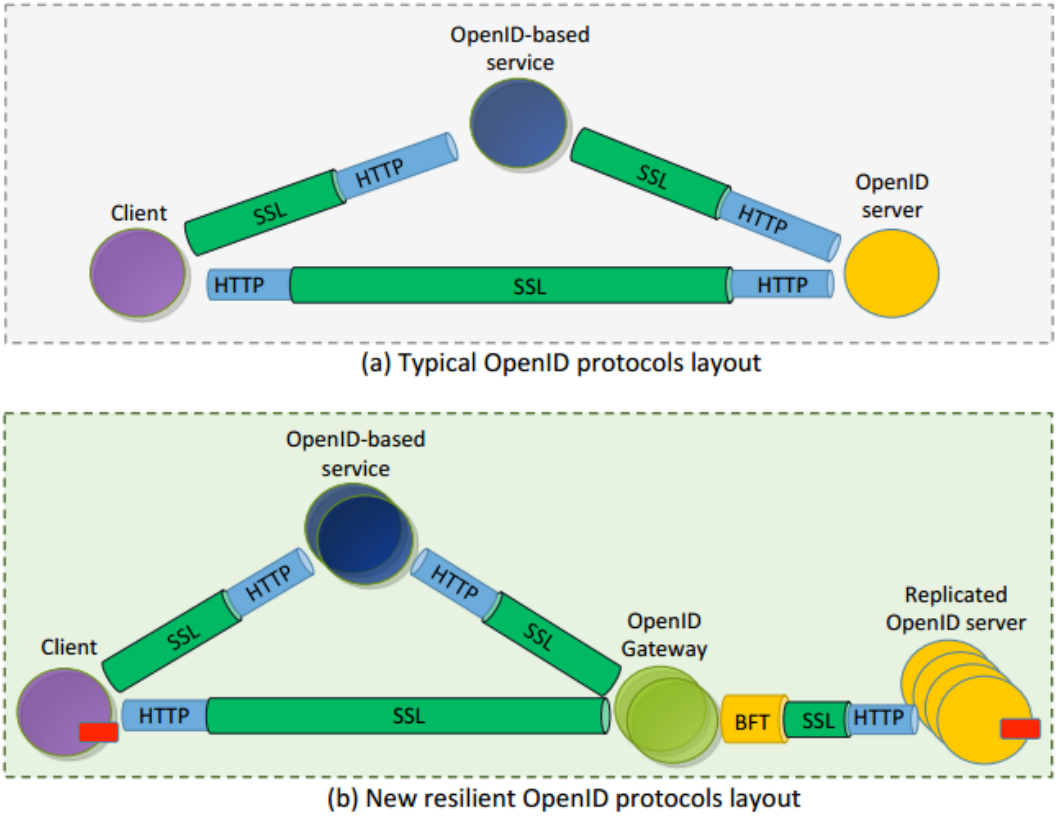

Figure 4.4: Proposed OpenID protocol stack [5].

4.3 Types of Faults and Threshold Values

The section presents the fault model of the proposed OpenID architecture and compare with the genereic fault model. Figure 4.5 illustrates the detection mechanisms between components of the resilient OpenID architecture. To mitigate failures, there are alternative detection mechanisms based on timeouts, corrupted messages, and malformed packets among clients and services (or relying parties), and OpenID gateways. Between OpenID gateways and OpenID Server replicas there are stronger mechanisms for tolerating arbitrary faults. In practical terms, arbitrary faults can be masked by state machine replication protocols. This means that any abnormal behavior (e.g. delay, malformed responses, and correct but malicious messages) of a replica Rx will not affect or delay the system operation. For instance, malformed or divergent responses from a corrupted replica will be simply deleted. It is assumed that the majority of replicas (i.e. all replicas except f_R) are correct and working as expected.

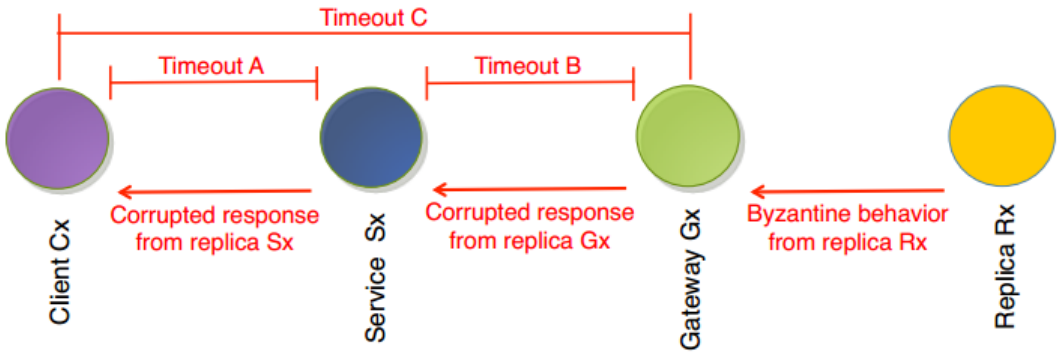

Figure 4.5: Proposed OpenID fault model [5].

As can be seen, the Figure 4.5 presents a timeout C between the client and gateway elements and the generic fault model on Figure 3.3 does not present the respective timeout. It happens because OpenID, in especial, presents moments that the gateway communicates directly with the client browser. Therefore, it is necessary the presence of a fault mechanism detection between the two elements to avoid blind points on the resilient system.

Table 4.1 summarizes the fault models and thresholds of the architecture's main elements. A relying party, OpenID gateway, and OpenID service can tolerate up to f_S, f_G and f_R simultaneous faults, respectively. Yet, the number of faults tolerated by the secure element depends on the specific architectural configuration. In the worst case, based on Figure 4.2, no faults are tolerated because there is only one single secure element. On the other hand, on the best case that we are considering, when the number of secure elements is equal to the number

of OpenID replicas, the number of faults tolerated is up to f_R, i.e., it is equal to the number of supported faulty replicas.

Table 4.1: Summary of fault models and respective thresholds

Component name	Fault model	Number of replicas	Fault threshold
Client	-	-	-
Relying party	Crash/Arbitrary(*)	$f_S + 1$	f_S
OpenID gateway	Crash/Arbitrary(*)	$f_G + 1$	f_G
OpenID service	Byzantine	$3f_R + 1$	f_R
Secure element	Crash	Up to $3f_R + 1$	Up to f_R

(*) These elements are capable of detecting some ("arbitrary") faults, such as malformed or corrupted packets.

4.4 System Model

In this section we introduce the system model and respective assumptions considered in our system design. We briefly describe the network/communication model, synchrony model, and fault model of the different elements of our architecture. This expresses the assumption and requirements of the system in order to be deployable and operational.

Network model. We assume that all service elements are visible to the clients and are able to connect to at least one gateway using standard TCP/IP. Furthermore, gateways are able to communicate with all replicas of the resilient OpenID service through state machine replication protocols. As a consequence, packets from the gateway to the replicas are encapsulated in these protocols. Moreover, it is also assumed that each replica is capable of communicating with at least one trusted component through a specific and/or standard interface, such as a shared memory channel, a secured IPSec connection, or a tightly controlled and isolated SSL/TCP/IP channel.

Synchrony model. We assume partial synchrony [100] to provide the minimum timing guarantees to ensure the complete execution of consensus protocols, which are required by Byzantine fault-tolerant protocols such as state machine replication.

Fault model. We assume arbitrary faults on the OpenID service. Therefore, we work with the standard case of $3f_R + 1$ replicas for tolerating up to f_R simultaneous faults without compromising the system's correct operation. Arbitrary fault tolerance is achieved through the BFT proxy module inside the gateway element, which expects at least $2f_R + 1$ equal answers from replicas before replying to the relying party or client. When the minimum quorum is not achieved for a

particular request, the gateway simply delete the replicas' replies and does not answer to the requester. In such cases, a protocol timeout will trigger a packet re-transmission on the client or service. Gateway and relying party are stateless and assumed to have a fail-stop behavior. Additionally, they are capable of detecting some protocol deviations as well, such as detect malformed or corrupted packets.

Furthermore, we assume an architecture comprised of $f_S + 1$ service relying party, $f_G + 1$ gateways and $3f_R + 1$ OpenID replicas, where f_S, f_G and f_R represent the maximum number of simultaneous faults tolerated by each element, respectively. Therefore, both clients and relying party elements can choose different elements (relying party and/or gateway) if the authentication fails or if the protocols' timeouts have expired and/or the maximum number of retries is exceeded, for instance.

User and server identification model. It is assumed that each user has a public key generated by a certificate authority (CA). Furthermore, all trusted components know the CA's public key (Pu_{CA}). This certificate is required to verify the user identity, based on the trusted third party CA.

Chapter 5

Functional Prototype Implementation

This Chapter presents the functional prototype and all its details. Firstly, it introduces the system design and main components. Furthermore, the chapter also presents the main interactions among the different elements and discuss the main deployment configurations used in our implementation.

5.1 System Implementation Overview

Figure 5.1 gives a high level overview of the implementation and building blocks of the main elements of the resilient OpenID service. We used the *openid4java* library [101] (version 0.9.8), which supports OpenID versions 1.0 and 2.0, to implement the replicated OpenID server. In our implementation we assume the OpenID version 2.0 as the default authentication scheme.

The active state machine replication of the system is achieved through the BFT-SMaRt library [3], which is freely and publicly available at Google Code [102]. This library provides a set of modules and communications protocols such as Mod-SMaRt, VP-Consensus and Reliable Channels for communicating among replicas (for more details, please refer Subsection 2.2.5). Both Mod-SMaRt and VP-Consensus use the reliable channels for communication purposes.

The secure element is an independent component of the system, required to safe guard confidentiality of sensitive information and used by the OpenID server for requesting the verification and generation of cryptographic material. These components, when deployed in a distributed fashion, i.e., one element per replica, can also leverage the functionalities offered by the Mod-SMaRt for exchanging verification data among the secure elements residing in different replicas. Furthermore, secure elements can be on the same physical machines of the OpenID

replicas, running in a different virtual machine, or in different physical machines. In other words, there is a minimal interface and communication subsystem between an OpenID replica and the secure element.

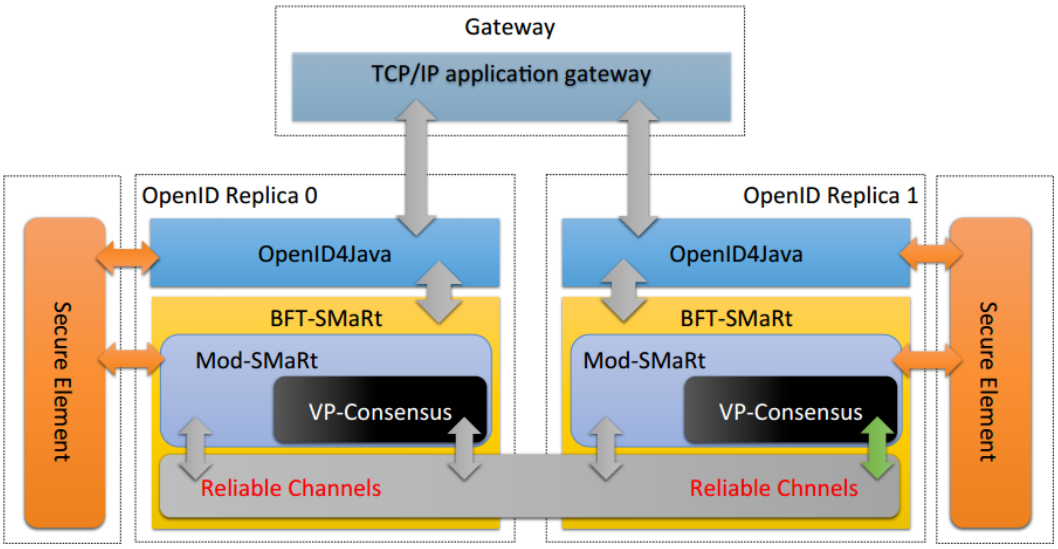

Figure 5.1: Overview of the replica internals [5].

5.1.1 Components

As we keep backward compatibility, components such as the client/browser and relying party can be used as they are, i.e., a normal browser and relying party can be used with our resilient OpenID. This is possible because of the new component introduced in the system's design, the OpenID gateway.

Client/Browser

In functional model concept, as we describe in more details in [4], a client is a generic element that can represents different things, such an applet running on the client's browser, an authentication component running on a virtual machine, and so forth. The only requirement is that is follows the standard definitions of the OpenID protocol version 2.0.

Relying Party

A relying party can be considered as a Web application that wants a proof that the end user owns a specific identifier [7]. Therefore, any application acting as a

relying party based on standard OpenID definition can use within our resilient OpenID system.

OpenID Gateway

The OpenID gateway is a new component introduced in the system design to keep backward compatibility. To this end, it needs to accept TCP/IP connections (with HTTP and/or HTTP over SSL) from OpenID clients and encapsulate the received packets in the BFT state machine replication protocol. This is necessary to forward the packets to all OpenID replicas. Similarly, packets coming from replicas are sent to the clients through the previously established TCP/IP connections. Therefore, the gateway is seen by the clients as a normal OpenID service. In other words, clients do not know that they are using a resilient OpenID service. In our model, the gateway tolerates up to f_G crash faults. Moreover, as the gateway only forwards (bi-directional) messages between clients/relying parties and replicas, it is a very simple element, which can be easily replicated and secured, making it unlikely to get compromised. Consequently, it is reasonable to assume that the security and trustworthiness of this element can be ensured to a certain level by leveraging security enhanced operating systems (e.g. SELinux [103] and REMUS [104]) and a well-defined, very tiny and verifiable packet forwarding/translating module.

Furthermore, we assume that the gateway is capable of providing network defense mechanisms, such as packet filtering. For instance, the well known and widely used GNU/Linux `iptables` can be employed limit the number of HTTP/HTTPS requests per unit of time (e.g. second) of the system, reducing the effect of potential denial of service attacks from relying parties and/or malicious users.

OpenID Replicas

The system replicas implement the standard OpenID version 2.0 using *openid4java* library [101] (version 0.9.8). The BFT-SMaRt library [3] is used to provide the required crash and Byzantine fault tolerant protocols.

Our current implementation supports only OpenID 2.0 over HTTP. Provide support for HTTPS and EAP-TLS (for strong mutual authentication) is planned as future work. Nevertheless, it is worth mentioning that we can leverage the EAP-TLS implementation of the resilient RADIUS (our resilient AAA service prototype), with the respective adaptations to the OpenID framework. Furthermore, EAP-TLS can be used in OpenID based system to provide strong mutual authentication using trusted components for the client (e.g. USB dongles) and the OpenID service (grid of smart cards) [105].

Trusted Component

In our first prototype, we assume that the trusted component can be executed on an isolated and secured special-crafted virtual machine, with the help of the isolation properties provided by modern hypervisors. Alternatively, in future versions, we can extend our prototype to use hardware-based secure elements such as grids of smart cards [106], as proposed in the context of the SecFuNet project [99, 107].

We implemented the trusted component using Java and the BouncyCastle [108] API. One of the challenges of the replicated trusted component was to overcome the determinism of the OpenID replicas. In other words, we need a trusted component that behaves deterministically among all replicas, if we are using one secure element per replica. To solve this problem we used a solution similar to our resilient AAA service, i.e., pseudo-random function (PRF), adapted from the TLS protocol, which outputs the same values in all replicas/trusted components. Further details of the solution can be found in the chapter describing the resilient RADIUS service.

5.1.2 Interactions between the System's Elements

Figure 5.2 shows the communication steps among the elements in the resilient OpenID system. As in the standard OpenID communication (Figure 2.1), full arrows represent messages in the client-server direction and dashed arrows represent the opposite. The communication start with the user requesting access to the service through the relying party (step 1). In step 2, the relying party presents with the end user a form that has a field for entering the user-specific identifier (identification URL in step 3). Following, in step 4 the relying party performs a discovery (YADIS [37]) to look up the necessary information for continuing the authentication process. The discovery request is forwarded by the gateway to the replicated OpenID service.

The OpenID service replies the relying party's request (step 5) with a XRDS document. This document contains, among other things, a list of endpoint URLs, with their respective priorities, which can be used by the relying party on the next steps.

Next, the relying party must choose one server (URL) and establish a protected association with it. Typically, the relying party attempts to connect to each server in the list according to the pre-defined priority. The first successful connection is used to carry out an association request (step 6). This kind of request contains data such as the respective endpoint URL and the relying party's Diffie-Hellman [109] data (public key, modulus prime number, generator number) to secure the association.

When the OpenID server receives the association request, it makes a request to the trusted component asking for the association handler and MAC key (step 7). Following, the OpenID server requests DH key-pair by sending the incoming data (modulus prime number and generator number) to the trusted component. After that, the trusted component generates and returns (step 8) all the information requested (association handler, MAC key and OpenID DH key-pair). The association response, containing the OpenID DH public key (among other information), it sent to the relying party (step 9), thus completing the association.

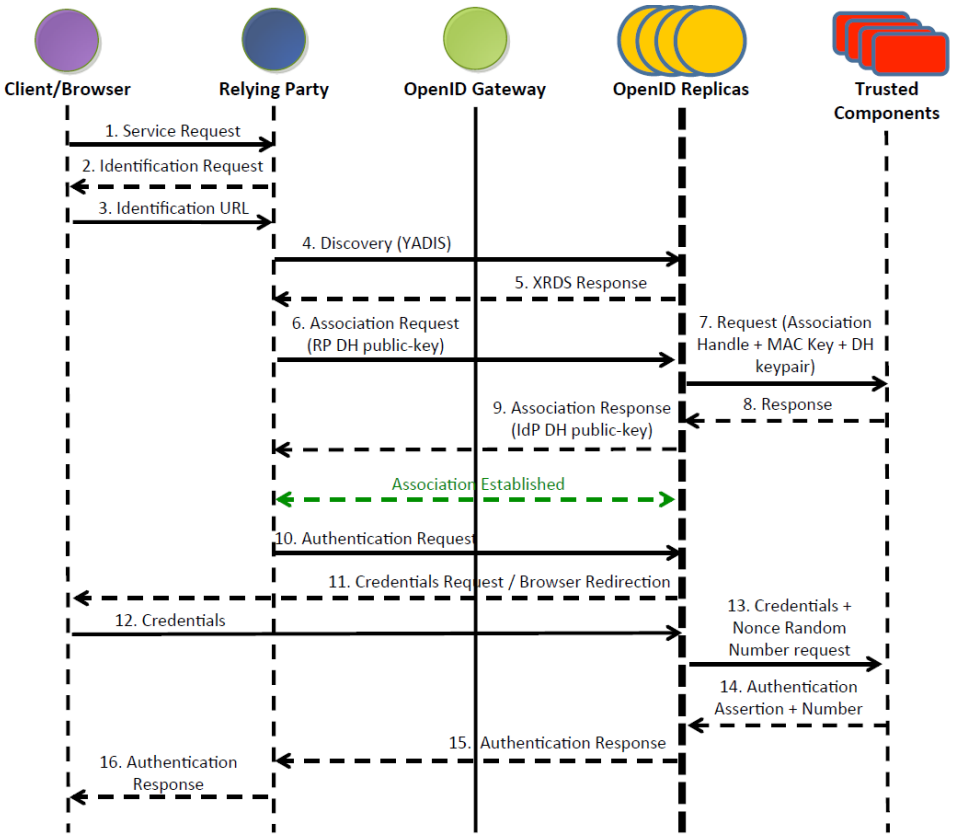

Figure 5.2: Overview of the proposed OpenID authentication flow [5].

As soon as the association between the relying party and OpenID server is established, the authentication takes place. In step 10, the relying party sends an authentication request to the OpenID server, which is forwarded to the client's browser in order to request the client's credentials (step 11). The client sends its credentials to the OpenID server (step 12). Following, the OpenID server requests the credentials' verification to the trusted component. Additionally, it also requests a nonce random number (step 13). The trusted component replies

with the authentication assertion and the generated nonce (step 14). After that, the authentication response is sent to the relying party (step 15), which performs the necessary actions. Lastly, the relying party sends an authentication response to the client, completing the authentication process (step 16).

5.1.3 Deployment Configurations

Figures 5.3 and 5.4 illustrate the two configurations possible with our prototype implementation. Similarly to the architectural configurations shown in Section 4.2, in the first deployment configuration there is only a single secure element, which is responsible of keeping the safety of sensitive data and critical crypto operations for all OpenID replicas. These replicas can be running on virtual machines controlled by the same hypervisor, for performance reasons, or in different physical machines for higher availability guarantees. However, when distributing the OpenID replicas over different physical machines, the system will naturally experience an increased overhead due to the communications among replicas and the single trusted component. Moreover, as the secure element represents a single point of failure, depending on the target environment requirements, there might be no reason for deploying the OpenID replicas in a different hypervisor since all replicas rely on a single trusted component for authenticating users. Once it fails, all replicas will be unable to proceed user authentications. Nonetheless, if the most critical point of the infrastructure is to ensure the OpenID framework operations, then it would make sense to distribute the OpenID replicas across different platforms or domains, even having a single trusted component to process the authentications. As discussed in Section 5.1.2, there are only two communications between the replicas and the trusted element. On the other hand, there are at least two requests which can be processed by the replicas without interacting with the trusted component.

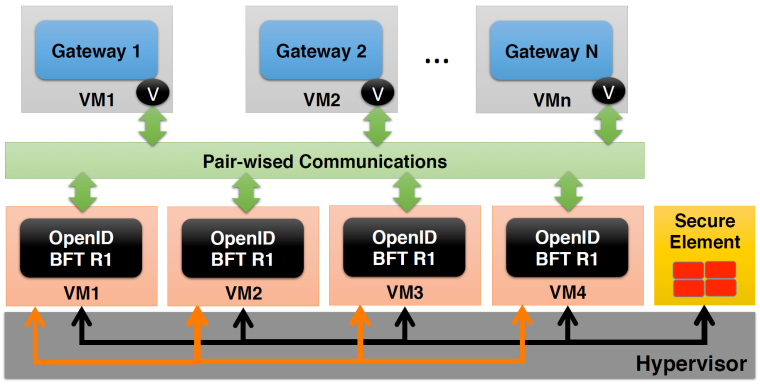

Figure 5.3: System design with one single secure element [5].

The second deployment configuration offers a more robust platform, as shown in Figure 5.4. In this case, each replica is deployed on a different physical machine, on a single or multiple domains. Each replica has access to its own trusted component, respectively. Therefore, up to f_R trusted components and replicas can fail (e.g. crash, be down for maintenance, out or reach due to communication problems, and so forth) without compromising the system operation. Furthermore, as both the gateway and the replicated OpenID have been implemented in Java, they can be easily deployed on diverse operating systems, where diversity increases the system robustness by avoiding common vulnerabilities [63]. In fact, we have tested our prototype in different systems, such as different Linux distributions (e.g. Debian, Ubuntu Server 12.04 LTS, Ubuntu Server 13.10 and Amazon Linux AMI), Windows and Mac OS X. Another interesting characteristic regarding diversity is the hypervisor independence. As the system relies on the message communication paradigm, it does not use shared memory resources of the hypervisor (e.g. shared memory provided by the hypervisor as a means of communication subsystem among virtual machines). Therefore, we can deploy our resilient OpenID service using different operating systems as well as diverse hypervisors, which increases even more the system diversity and, consequently, its robustness against common vulnerabilities both in operating systems and hypervisors. We have deployed out prototype on virtual machines running on different hypervisors, such as VirtualBox, Xen and Amazon's EC2 hypervisor.

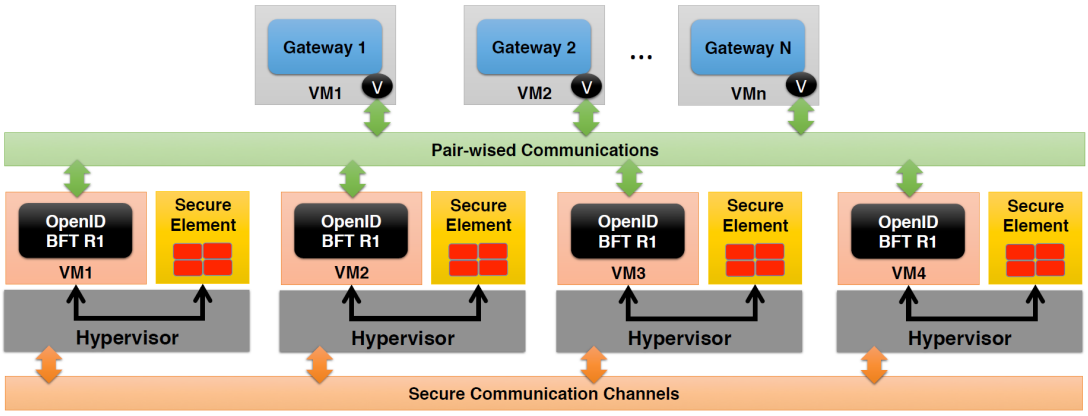

Figure 5.4: System design with multiple $(3f + 1)$ secure elements [5].

The Chapter presented the functional prototype and all technical details like architectural structure, components and how they communicate between themselves. It presented the deployment configurations of the proposed system as well as the advantages and disadvantages of the different configurations of the prototype.

Chapter 6

Discussion and Experimental Evaluation

This Chapter presents general discussion about some OpenID vulnerabilities and attacks. The goal here is to evaluate the proposed solution and our prototype implementation regarding different attacks and types of faults, verifying how the system behaves on the attacks' presence. Moreover, it is provided performance evaluations on three different scenarios, measuring the throughput and latency of the system's essential operations.

6.1 OpenID Attacks Analysis

As said previously on Subsection 2.1.1, OpenID standard presents several security issues, both in terms of specification and implementation [24, 74, 110, 111]. Therefore, we analyze on the following sections some of the current OpenID attacks and how they are mitigated (or what kind of problems they may cause) in our resilient OpenID system.

It is worth mentioning that phishing and CSRF attacks [24] in OpenID architectures are not covered by our solution. These attacks focus on the user/client and relying party, i.e., elements outside of our intended protection scope. More details about the real scope is presented in next sections like some known OpenID issues (MITM, DoS, replay), resilience, and intrusion and fault tolerance as well.

6.1.1 Man-in-the-middle Attacks

Problem: Man-in-the-middle attacks are characterized by the attacker interception communications in both directions. As an example, the relying party connects with the attacker's system. However, for it the attacker is an OpenID

provider because the attacker forwards the relying party requests to the OpenID provider, which will reply to the attacker. Subsequently, the attacker is going to send a reply to the caller, the relying party. Obviously, the attacker is intercepting all communications between the relying party and the OpenID provider.

In order to avoid man-in-the-middle attacks, the OpenID framework (version 2.0) specifies that associations between relying party and the identity provider must use a shared secret for preventing an attacker of tampering the signed fields. However, the problem is that the associations are negotiated over Diffie-Hellman, which can not by itself avoid interception attacks.

Solution: To effectively prepare the system against man-in-the-middle attacks, the use of protocols such as TLS, with certificates and mutual authentication, is imperative. A user and the OpenID server need to have certificates signed by a trusted authority that can be verified by each element (e.g. client and server) of the communication. As discussed in Chapter 4, our system design assumes protocols such as SSL and TLS to protect the system's communications. We highlight the move as a solution because even being a standard precaution, many 'security' systems do not take care of this simple detail.

6.1.2 DoS Attacks

Problem: Denial of Service (DoS) attacks have the goal to exhaust the system's resources, making it unavailable to the end users. In the OpenID framework, all elements can eventually be affected by DoS attacks. The relying party can be a target of DoS attacks if it allows the user choose the URL to identify himself. For instance, a user could maliciously insert an URL pointing to a movie file. Consequently, during the discovery process, the relying party would download the movie file of the input URL [43]. Furthermore, an OpenID server can also be the target of a DoS attack. For instance, an attacker can use a compromised relying party to generate a huge number of association, authentication or signature verification requests.

Solution: To prevent DoS attacks from the relying party to the OpenID provider, packet filters, such as `iptables`, can be used on the gateway to impose a limit on the number of requests per second coming from the relying parties. Furthermore, the OpenID service can ban requests based on the values *openid.realm* and *openid.return_to* in the protocol messages exchanged with the relying party [43]. Our resilient OpenID provides mechanisms based on timeouts, corrupted messages and malformed packets to detect attacks between clients, relying parties and OpenID gateways. Nevertheless, it is not sufficient to defend the system against DoS attacks, such as huge proportion DDoS attacks. Despite providing defense mechanisms in the gateway and replicated OpenID server, our system can also leverages the power of advanced detection and mitigation mechanisms provided

by use of cloud infrastructures, such as CloudFare [81, 93], a cloud provider that was able to tolerate a DDoS of 20Gbps (with a peak of 120Gpbs) for more than one week without compromising the clients' services. As we show and discuss in the evaluation section, our solution can be deployed in a multi-cloud and/or multi-data center environment.

6.1.3 Replay Attacks

Problem: Replay attacks try to eavesdrop information without authorization and use it to trick the receiver (e.g. by retransmitting the same packet) to execute unauthorized operations such as authentication of an unauthorized client/user. According to OpenID specifications, the number used once, also known as nonce, required in the authentication process does not have to be signed and verified. Therefore, it is up to the developer to decide whether the nonce is going to be signed and verified or not. If the nonce is not part of the signed information, an eavesdropper can intercept a successful authentication assertion (sent from the OpenID service to the relying party) and re-use it.
Solution: Our replicated OpenID server inserts the nonce in the signed information list, which is sent within the authentication response. Additionally, the *openid4java* client performs a check of the signature and keeps track of the non-expired nonce values, which were already used in positive assertions, and never accepts the same value more than once for the same OpenID endpoint URL. If the received nonce value is expired, it is rejected.

6.2 Tolerating Crash and Byzantine Faults

In this section, we discuss the behavior of the replicated OpenID in the presence of a faulty replica due crash or Byzantine fault. Additionally, we analyze the replicated OpenID with one faulty gateway. Furthermore, we analytically compare our solution with a single application OpenID server, the JOIDS [112], which represents the common case of available OpenID implementations.

- **Fail-stop**. Forcing a replica to crash is as simple as killing its respective process. While our system tolerates up to f_G gateway faults and f_R replica faults, the JOIDS does not tolerate faults. If the application crashes, the OpenID provider will be compromised. The resilient OpenID service keeps working correctly, without system delays, despite up to f_R simultaneously compromised replicas.

- **Byzantine**. Considering arbitrary faults, things get even worse. The JOIDS OpenID application does not tolerate arbitrary faults (e.g. bugs,

misconfigurations, message delays caused by an attacker, and so forth). Therefore, arbitrary faults can have different kinds of impacts in the system's operation, such as denying user authentication, granting of association assertions to un-authorized users, and so forth. On the other hand, our solution tolerates up to f_R arbitrary faults on the OpenID replicas. Furthermore, if we consider a scenario with arbitrary faults in up to f_G gateways, only relying parties and users relying on those f_G gateway will be affected. Relying parties and clients using the remaining gateways will not experience any abnormal behavior of the system.

6.3 Performance Analysis

In the following section we discuss some experimental results. Our main goal is to measure the system performance and behavior in different computing environments. Therefore, we use three distinct execution environments in order to observe and compare the system performance and potential limitations. Additionally, the section presents a brief comparision between the replicated OpenID and the standard one.

6.3.1 Execution Environments

We used three different environments to run our experiments. Two of them with virtual machines (VMs) running on the same physical infrastructure, one on the same server and one in the same data center. A third one is composed by multiple data centers spread from the east to the west coast of the US.

The **first test environment (UFAM-VMs)** has five virtual machines with 2GB of RAM, 2 vCPUs (vCPU - virtual CPU), one hard disk of 20GB (allocated on a disk image file) and a swap disk image with 1GB of space. All virtual machines run the GNU/Linux distribution Debian Wheezy 7.0. Furthermore, these VMs are supported by a physical machine with one Intel Xeon E5-2420 processor with 6 cores with hyper threading supporting up to 12 threads in parallel, 24GB (6x4GB) of RAM, one Intel I350 Gigabit Network card and a software-based RAID 1 disk with 500GB (2x500GB). This physical machine is allocated at UFAM. Virtualization is supported by the Xen 4.1 kernel module.

The **second test environment (Amazon-EC2)** uses elastic computing nodes (EC2) from Amazon AWS to run our system. We used five `m3.xlarge` instances [113], with 4 vCPUs, 13 ECUs, 15GB of RAM and 2 x 40GB SSD disks. All instances were allocated in Amazon's data center in N. Virginia (US East Zone). These nodes are interconnected through high speed gigabit network. The nodes were running Ubuntu Server version 13.10.

The **third test environment (Amazon-DCs)** uses a multi-cloud/multi-datacenter configuration. Again, we used five VMs. Two of them allocated in N. Virginia, two in N. California and one in Oregon. These five VMs have the same configuration of those used in Amazon-EC2 environment. However, what changes is the operating system. N. Virginia's VMs run Ubuntu Server version 13.10, while the VMs allocated in N. California run Amazon Linux AMI. Lastly, the Oregon's VM was running Ubuntu Server version 12.04 LTS.

Figure 6.1 illustrates the Amazon-DCs environment. Clearly, it is a cutting map and three different locations can be seen in the Figure. Specifically, it illustrates the places where our test machines from Amazon are located. As can be observed, one machine is in Oregon and other two are in Northern California, all of them on the west coast of United States. The fourth and fifth machine are in Northern Virginia, on the east coast of US. The Figure is capable to present some latency information about the whole test environment. Furthermore, as one would expect, the latency between east and west coasts (87.34ms and 78.36ms) is much higher than the latency between N. California and Oregon (32.10ms). The network latency measurements were taken using the `ping` tool to generate one hundred packets of 512 bytes. Therefore, the number represent the average of 100 ICMP messages. Moreover, we used the size of 512 bytes due to the fact that most of the OpenID's authentication messages, in our experimental setup, were of around this size, as further described in the following sections.

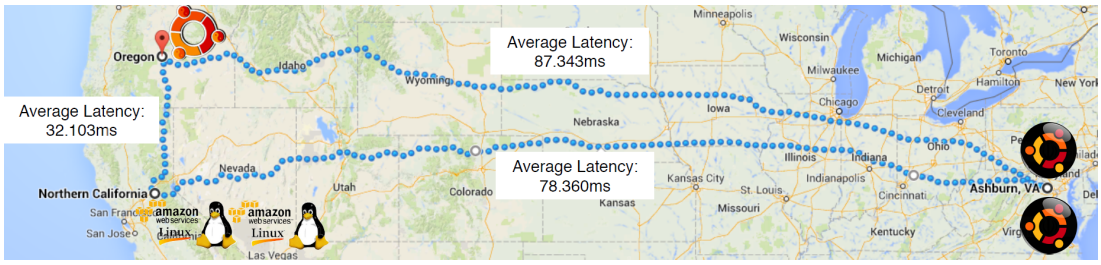

Figure 6.1: Multi-cloud environment overview [5].

All VMs from the three environments were running `openjdk-1.7`. The resilient OpenID prototype was developed using *openid4java* version 0.9.8 [101] and BFT-SMaRt version 0.8 [3].

6.3.2 System Authentication Throughput and Latency

One of our goals was to measure how many OpenID authentications our resilient service can support in different execution environments. Furthermore, we measured also the network and system latency to identify the impact of the network and the potential system bottlenecks or points of improvement.

Table 6.1 summarizes the main results of our executions in the three environments. We used a variable number of clients (20, 40, 80, 100) running with the gateway element. Each client executes 2.000 authentication requests. An authentication is composed by 5 different messages, in accordance to the OpenID 2.0 specification. Two of them are needed for discovery process, one for the association and the last two for authentication process. Therefore, each client sends 10.000 messages to the gateway and OpenID replicas. Due to time and resource allocation constraints, we executed each configuration (e.g. 20 clients) on each environment only five times. Thus, the system throughput number (authentications/s) shown in this sections represent an average of five executions. Consequently, some of the differences, in particular on the standard deviation results, could change with more executions. However, based on our tests and observations, the main throughput results would not significantly change.

The discovery process is a formal request sent by the client aiming to establish a connection with the OpenID service. This request is essentially an HTTP `HEAD` request. If the OpenID service returns an error, the client sends a subsequent HTTP `GET` request. Since our implementation is based on the `openid4java`, and the original implementation of the library only handles `GET` requests, we send but ignore the `HEAD` request in order to keep backward compatibility. In the second message, the client performs a `GET` request with the respective `Accept` header. As both discovery requests do not have content, we set their payload to 0 (zero) bytes in our experiments.

The next step is to send (by the client) an association request using HTTP `POST` containing a set of OpenID parameters as content, resulting in payload size of 343 bytes. The request includes the OpenID Authentication request's version (`openid.ns`); a value to identify the message type of the OpenID request (`openid.mode`); and the shared secret key, resulting from the Diffie-Hellman Key Exchange, to be used to sign subsequent messages between client and the OpenID service.

The forth message starts the authentication phase. It is another HTTP `POST` message with a payload of 506 bytes. The request's parameters include the identification provided by the user (`openid.claimed_id`), the return URL that will be used by the OpenID service to send the authentication result (`openid.return_to` and the handler of the previously established association (`openid.assoc_handle`).

During our experiments, we assumed that client has not yet been authenticated in the OpenID service. Consequently, the client needs to feed a web form using his credentials. In our prototype, this form has 537 bytes. Nevertheless, this size can vary accordingly to the respective relying party.

As can be observed in Table 6.1, the number of authentications per second varies from nearly 860 (with 20 clients) to 995 (with 80 clients) in the UFAM-VMs.

Table 6.1: Experimental results [5].

Environment	# of clients	# of auths	Exec time	# of auths/s
UFAM-VMs	20	40000	46.39s	867.73
	40	80000	87.94s	984.59
	80	160000	162.67s	995.12
	100	200000	209.41s	960.11
Amazon-EC2	20	40000	20.37s	1969.17
	40	80000	36.93s	2166.58
	80	160000	71.41s	2244.30
	100	200000	89.33s	2244.04
Amazon-DCs	20	40000	150.00s	26.66
	40	80000	157.71s	50.72
	80	160000	173.12s	92.42
	100	200000	175.39s	114.05

There is a drop in throughput with 100 clients. This is due to the execution environment limits. Too many simultaneous clients leads the system to a thrashing situation, i.e., the scheduling, concurrent I/O requests and memory consumption exceed the reasonable values of the system, leading to a lower performance. A similar behavior can be observed in the second environment, Amazon-EC2. In order to execute more clients (lets say 80 or more) simultaneously, more virtual machines are required. Whatever misunderstanding or suspicion related to the number of authentications per second or the total authentications on the table, please refer to Table 6.2 which presents the standard deviation of the values.

Another interesting performance result is the difference between the UFAM-VMs and Amazon-EC2. They are similar computing environments, except that the VMs of Amazon-EC2 have more computing power, which makes a huge difference on the overall system performance. Taking as an example 80 clients, the Amazon-EC2 supports 2.25x more authentications per second.

Table 6.2: Standard deviation from authentication per second [5].

Environment	20	40	80	100
UFAM-VMs	79.7382	261.6494	111.1233	118.7884
Amazon-EC2	120.8838	44.5214	106.01878	121.5597
Amazon-DCs	0.3752	0.6322	1.0592	1.9577

Lastly, as one would expect, the Amazon-DCs environment presents the lowest performance. One of the main reasons for this significant drop in performance

in the network latency, as can be observed in Table 6.3. The average network latency between data centers (e.g. `nvirginia-ncalifornia`) is 94.40x higher than the worst network latency case (`ec2h1-ec2h2`) of the Amazon-EC2 environment. However, it is worth mentioning that we have a growing trend in the number of authentications per second. As the network latency is high, more simultaneous clients can potentially explore better the network I/O and bandwidth. Differently from the other two environments, the major limitation of the Amazon-DCs environment is the proportionally huge network latency. Nevertheless, we considered the results achieved in the inter-data center setup reasonably good. For 20 clients we achieve a throughput of 26 authentications per second, while around 114 authentication/s for 100 clients. An authentication service with such number of authentications per second can be considered of medium scale. You need at least thousands of users in the system to reach more than one hundred authentications/s.

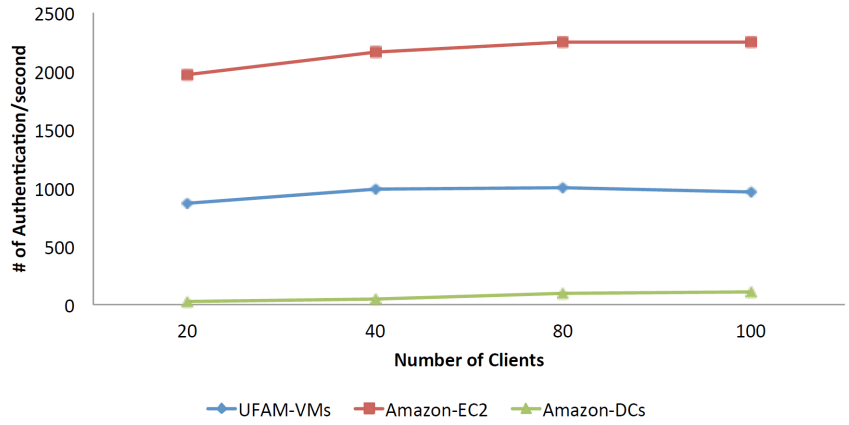

Figure 6.2: Total number of authentications per second [5].

The system throughput on the UFAM-VMs and Amazon-EC2 environments provides evidence that the virtual machine's and environment's configurations (number of vCPU, memory, disks, network efficiency) has a significant impact on the number of authentication per second. Furthermore, despite having a higher throughput in the UFAM-VMs' environment when compared to the Amazon-DCs environment, as show in Table 6.2, the standard deviation from the latter is more stable and lower than the former. This indicates that the OpenID replicas running on the Amazon-DCs' environment can handle a more constant (and potentially growing) number of authentications per second with an increased (e.g. 200) number of clients. On the other hand, despite having a variable and high standard deviation, by running the replicas in the Amazon-EC2 environment the system is capable of handling much more authentications per seconds, as can be

observed in the graph of Figure 6.2.

Table 6.3: Latency: min, max, average and standard deviation [5].

Environment	Hosts	min	avg	max	std dev
UFAM-VMs	host0-host1	0.089	**0.092**	0.258	**0.021**
	host0-host2	0.056	**0.059**	0.204	**0.014**
	host0-host3	0.090	**0.092**	0.257	**0.020**
	host0-host4	0.056	**0.059**	0.217	**0.015**
	host1-host2	0.090	**0.105**	0.319	**0.028**
Amazon-EC2	ec2h0-ec2h1	0.143	**0.176**	0.283	**0.027**
	ec2h0-ec2h2	0.160	**0.191**	0.298	**0.029**
	ec2h0-ec2h3	0.157	**0.188**	0.299	**0.038**
	ec2h0-ec2h4	0.180	**0.213**	0.354	**0.031**
	ec2h1-ec2h2	0.148	**0.173**	0.830	**0.068**
Amazon-DCs	oregon-nvirginia	86.325	**86.707**	87.343	**0.312**
	oregon-ncalifornia	31.391	**31.703**	32.103	**0.244**
	nvirginia-ncalifornia	77.728	**78.027**	78.360	**0.225**

Table 6.3 shows the latency results for all three environments. The measurements were carried out using the `ping` program generating 100 ICMP requests of 512 bytes between each pair of hosts. The Table contains the standard output (results) of the GNU/Linux `ping` command. `Host0`, `ec2h0` and `oregon` represent the hosts running the simulated clients and gateway elements of the system. Therefore, the measures are done from the gateway host to the four OpenID replicas. Moreover, another measurement between replicas' hosts (e.g. `host1-host2` for UFAM-VMs) has been done. This last measurement gives an idea of the network latency between the OpenID replicas.

As expected, the UFAM-VMs' virtual machines have a low latency and the lowest standard deviation. This happens because all virtual machines are running in the same physical machine. Nevertheless, the network latency varies significantly between different virtual machines. If observing the average values, it goes from 0.059ms up to 0.090ms, which represents a variation of nearly 1.52x.

On the other hand, the average latency between hosts varies less in the Amazon-EC2 environment. Nevertheless, the latency is higher than in the UFAM-VMs environment. Furthermore, the standard deviation is also higher in the Amazon-EC2 setup.

Lastly, the network latency variation is much higher in the WAN connections, as can be seen in Amazon-DCs' latency results. It goes from 32.103ms between Oregon and N. California up to 87.343ms between Oregon and N. Virginia, which represents a difference of 2.72x. This latency variation is almost unpredictable

and can significantly vary from data center to data center connection. Therefore, one should also take care before choosing the clouds and/or data centers to deploy the nodes of a distributed system.

Table 6.4 presents a brief comparison between the replicated OpenID and an original one. The original OpenId was subjected to the same set of tests performed to the replicated one. As expected, the original OpenID provides a higher throughput of authentications. It happens due the overhead of messages exchanged by the state machine replication protocol existent in the proposed OpenID. Many messages are exchanged in order to ensure the coordination and synchronization between replicas, keep the determinism, perform consensus, leader election and other controls performed by the replication library BFT-SMaRt. Moreover, it is worth to remember that the replicated solution adds a new element in the flow, the gateway, increasing even more the number of exchanged messages, and consequently reducing the system performance. In other words, we present an OpenID solution with less performance, but in return we provide resilience, high availability and fault and intrusion tolerance.

Table 6.4: OpenID Type Comparison

OpenID Type	**# of clients**	**# of auths**	**Exec time**	**# of auths/s**
Replicated OpenID	20	40000	46.39s	867.73
	40	80000	87.94s	984.59
	80	160000	162.67s	995.12
	100	200000	209.41s	960.11
Standard OpenID	20	40000	28.82s	1388.19
	40	80000	54.29s	1474.01
	80	160000	108.93s	1468.97
	100	200000	135.25s	1478.94

6.3.3 Latency Measurements between System Elements

We have also done some latency measurements between system elements. The main idea was to identify potential bottlenecks and/or places where the system can be further improved. For instance, one of the goals is to observe the latency overhead of centralized and distributed secure components.

Between the gateway and the OpenID replicas we have the following latency (average of 9000 authentications) 7.16ms (UFAM-VMs), 6.22ms (Amazon-EC2) and 857.93ms (Amazon-DCs). As expected, the latency follows the throughput trend. Furthermore, 7.16ms and 6.22 are relatively low values considering that we have five OpenID messages per authentication plus the delay of the state

machine replication protocol among the system replicas. However, on the inter-data center environment the value is significantly high. One way to reduce the latency overhead could be to place the replicas in the data centers with the lowest latency (e.g. two replicas in N. California and two in Oregon). This would certainly significantly reduce the latency of the Byzantine fault tolerant protocols.

Lastly, the latency between the replicas and the trusted components is of 10.51ms (UFAM-VMs) and 10.50ms (Amazon-EC2) for the system configuration with a single trusted component. When using replicated secure elements, considering one agreement round among them per replica request, the latency goes up to 20.58ms (UFAM-VMs) and 20.23ms (Amazon-EC2). Consequently, one could think that the system is limited to nearly 100 authentications per second with a centralized trusted component and around 50 authentications/s when using a replicated secure element (one per replica). This would be true if the trusted component is a single-threaded application or hardware-device that sequentializes all incoming requests. Therefore, one of the main potential bottlenecks of the system is, of course, the trusted component. While a hardware-based trusted component could impose a significant limit to the system, a secure element on a trusted computing base (e.g. secure hypervisor), running within the hypervisor or in an isolated virtual machine, can significantly boost the system's performance by exploring mechanisms for processing requests concurrently and/or in parallel.867,73-984,59

6.3.4 Attacks on replicas and gateways

One of our goals was to evaluate the system behavior under constant crashes and/or attacks.

Constantly crashing up to f_R replicas. To evaluate the system performance under harsh circumstances, such as continuous crashes, we used the UFAM-VMs environment with 20 and 40 clients. We implemented a script that kills and restarts one of the replica every 10s during the system execution. With periodically less replicas (up to f_R replicas, which in our test represents 1 replica) in the system, we experienced a slightly higher throughput, going from 867.73 to 1009.86 with 20 clients and from 984.59 to 1145.98 with 40 clients. This is explained by the fact that less replicas on the system ($3f_R + 1 - f_R$) generates a lower overhead in communications for state machine replication protocols (consensus, ordering, and so forth). Consequently, our system did not experience any kind of problems, or performance degradations, with constant crashes of up to f_R replicas.

DoS attack on up to f_R replicas. Similarly, with a DoS attack on up to f_R replicas, we experienced a slightly increase in the system throughput. We used

the `hping3` command to generate a constant DoS attack, using TCP SYN and ACK flags, on the TCP port of one replica. This replica started to slow down and/or not receive all messages from the gateway due to the attack. Therefore, the remaining replicas considered it as compromised and kept the system in operation relying on the $3f_R+1-f_R$ correct replicas, in our case 3 replicas. Again, the system experienced a slightly higher increase in performance, going from 867.73 to 956.46 authentications/s with 20 clients, and from 984.59 to 1005.54 authentications per second with 40 clients. The increase in throughput is lower than with crash faults because the replica is still up and working throughout the execution of the experiment, eventually sending replies to the remaining $3f_R + 1 - f_R$ replicas, despite the DoS attack. Nevertheless, the overall system performance keeps still over the normal case, without any faults.

Constantly crashing up to f_G gateways. We ran also an experiment to observe the behavior of the system with faulty gateways. To this purpose, we used 10 clients sending (each of them) 100 authentication requests per second, two gateways and four OpenID replicas. We created a script to kill and restart one of the gateways every 5 seconds. In the end, we observed that a faulty gateway, crashing every 5 seconds, causes an overall drop of approximately 27% on the system throughput (number of authentications/s). This is due the fact that clients and relying parties have to re-send the requests not answered by the faulty gateway and/or try to connect to the second gateway, which causes an additional overhead on the authentication process. However, we believe that the results can be improved by applying some optimizations (on the communications behavior) on the relying party and clients, for instance.

Chapter 7

Conclusion

This Chapter finalizes this master thesis and presents the final remarks of the work. Moreover, it presents the main difficulties found, main contributions to the scientific community as well as the future work.

7.1 Contributions

The section presents the main contributions of this work:

1. The design and implementation of a resilient OpenID-compliant server for services that rely on OpenID-based identity providers
2. Implementation of a fault- and intrusion-tolerant OpenID service with backward compatibility with existing systems, i.e., its deployment should not require any modification in a typical OpenID-based architecture
3. Design, implementation and evaluation of a resilient and trustworthy OpenID-based identity provider based on virtual replication and end-to-end TLS authentication
4. Experimental evaluation of the system performance and behavior in different environments, such as multiple virtual machines in a single physical machine and multiple virtual machines running in different clouds and/or data centers

Beyond the enumerated contributions, we can highlight the conference publications achieved as well:

1. Kreutz, D., Feitosa, E., Malichevskyy, O., Barbosa, K., **Cunha, H.** "A Functional Model for Identification and Authentication Services Tolerant

to Faults and Intrusions", in XIII Symposium on Information Security and Computer Systems.

2. ??

7.2 Research difficulties

Despite we have used some libraries to facilitate the job like BFT-SMaRt [3], OpenID4Java [101] and Bouncy Castle [108], implementing an OpenID fault- and intrusion tolerant service is not an easy job. This section presents to the reader the main difficulties found while developing and designing the resilient OpenID service.

Java provides a large set of methods and classes capable of encapsulate all the HTTP conversation through the web containers called Servlets. However, these facilities could not be used because BFT-SMaRt library handles all the communication by vector of bytes, forcing us to perform treatment of HTTP requests and responses some layers below application layer. In other words, it was necessary mount and read manually the HTTP packages in all the three points of the prototype (relying party, gateway and server replicas). This task is not too hard by itself, but it took us some days re-doing something which is already done by Java methods and classes.

Moreover, the task of keeping the solution's determinism was a hard job. OpenID standard was not designed to tolerate arbitrary faults and work with state machine replication. It means that as soon as we configured the OpenID standard implementation to use BFT-SMaRt we faced different behaviors on each OpenID service replica when performing some tasks. As explained on Subsection 2.2.2 of this work, state machine replication services must operate coordinated and in a deterministic way, processing the requests on the same order and responding equally. However, OpenID authentication flow needs to performs some random number generation and Diffie-Hellman key generation. The system should be capable of generate the same random numbers and keys on all the OpenID replicas in order to keep the determinism on the system. The difficulties related to this problem are:

(a) Association handle

(b) Mackey seed (32 bytes length)

(c) Diffie-Hellman keypair

(d) Nonce

The first two items above were solved passing the responsability to the Trusted Component. In the first system configuration (see Section 4.2), the TC could generate the information in a random way and return to all replicas the same generated value. Otherwise, if system is accordingly to the second deployment configuration, TC uses a pseudo-random number generator which receives two fixed seeds in order to generate securely the bytes to all replicas. Diffie-Hellman keypair (on the replicated Trusted Component form) is not generated by the pseudo-random number generator because even providing the same seed, Bouncy Castle library does not repeat DH keys. To this specific situation, we created a large vector of DH keypairs and access them through a pseudo-randomized index. Additionally, nonce is not generated using the pseudo-random generator solely, it uses a timestamp information which is provided by the BFT-SMaRt library and sent by the replica leader to all other replicas for synchronizing purposes. To do that, we use a modified version of the BFT-SMaRt library, because in the original build, the replica leader does not keep the timestamp information for itself.

Beyond the difficulties described, we had to insert all the solutions keeping the OpenID4Java [101] architecture (object types, dependencies and the classes' way of functioning) to avoid compatibility problems with the remaining OpenID4Java code.

7.3 Final Remarks

In this work we proposed a system architecture for developing and deploying resilient identity providers based on OpenID standard keeping the compatibility with the existent OpenID-based identification and authorization infrastructures. It was presented all the basic concepts necessary to understand the proposal and its characteristics. Beyond the concepts, we described the necessary elements to design resilient systems and provide fault- and intrusion-tolerance on network services. Moreover, we described and analyzed our results on developing this OpenID-based infrastructure using state machine replication protocols and trusted elements, among other techniques.

A prototype of the system design was implemented as a proof of concept. The implementation is based on Java and it uses the OpenID4Java [101], BFT-SMaRt [3] and Bouncy Castle [108] libraries. Using this prototype we showed that is possible add some security to OpenID authentications as well as provide more resilience and availability.

As presented in Section 4.2, the resilient OpenID system follows a pshysical state machine replication and therefore, it allows replicas being distributed across multiple physical machines and/or administrative domains. Using the de-

veloped prototype, we have demonstrated how the system is capable of masking up to f_R arbitrary faults on the OpenID service replicas. Furthermore, we have introduced a new component, the OpenID gateway, which performs the communication between the Relying Party and the service replicas keeping the backward compatibility.

Furthermore, we evaluate and discussed how our system design is capable of avoiding or mitigating different attacks such as reply and DoS. The experimental results demonstrate also the system performance on threee different environments, multiple virtual machines on a single physical machine, multiple virtual machines on a single data center and multiple virtual machines spread across multiple data centers. Through the respective results we could achieve a high-value throughput, in specially in the Amazon-EC2 environment of more than two thousand OpenID authentication per second. This is a significant achievement considering that we are relying on heavy protocols such as those required for state machine replication and for tolerating faults and intrusions.

We also discussed how the network latency and the latency between different elements can affect the overall system performance. As indicated by our results the two most significant impacts are caused by the network and system latency. For instance, WAN latency, significantly reduces the system throughput. Similarly, trusted components can have also a significant impact on the system performance.

Lastly, we present the throughput comparison between our proposal and a real pure OpenID server. Despite our proposal is slower than an OpenID pure server, it is easily minimized by the other benefits brought by the resilient OpenID proposal. The capacity of provide a higher available, resilient and fault- and intrusion tolerant service worths less authentications per second.

7.4 Future Work

Despite the promising results, there is still room for improvements and further investigation. The main ones are:

1. Use of the most recent version of BFT-SMaRt which has several performance and durability optimizations;

2. Use optimized pools of thread on the gateway;

3. Use multiple gateways on performance tests, since the replicas are capable of processing more than 70k raw messages per second [3];

4. Use more powerful computing nodes such as m3.2xlarge [113], which nearly double the computing power of the nodes used in Amazon-EC2 environment;

5. Send requests in batches between the gateway and replicas;

6. Implement a more powerful version of the Secure Element to process requests faster using multiple threads. And also transfer all the sensitive data responsability to the Secure Element such as handle the association information and user data (passwords, among others);

7. Analyse the scalability of OpenID as a service;

8. Adapt the new version of OpenID Connect [114] to the resilient architecture presented here;

9. Evaluate the resilience degree of the proposed Identity Provider using metrics and techniques present in the literature like [14].

After all, "*The research can not stop.*" (Ruiter Caldas and Kaio Barbosa).

Bibliography

[1] P. Sovis, F. Kohlar, and J. Schwenk, "Security analysis of openid.," in *Sicherheit*, pp. 329–340, 2010.

[2] P. E. Veríssimo, N. F. Neves, and M. P. Correia, "Intrusion-tolerant architectures: Concepts and design," in *Architecting Dependable Systems* (R. de Lemos, C. Gacek, and A. Romanovsky, eds.), vol. 2677 of *Lecture Notes in Computer Science*, pp. 3–36, Springer Berlin Heidelberg, 2003.

[3] A. Bessani, J. Sousa, and E. Alchieri, "State machine replication for the masses with bft-smart," *DI/FCUL, Tech. Rep*, 2013.

[4] D. Kreutz, H. Niedermayer, E. Feitosa, J. da Silva Fraga, and O. Malichevskyy, "Architecture components for resilient networks," tech. rep., SecFuNet Consortium, 2013.

[5] H. Niedermayer, D. Kreutz, E. Feitosa, O. Malichevskyy, A. Bessani, J. Fraga, H. A. Cunha, and H. Kinkelin, "Trustworthy and resilient authentication service architectures," tech. rep., SecFuNet Consortium, 2014.

[6] S. Consortium, "Shibboleth identity provider." `https://shibboleth.net/`, 2014. Acessed on: 2014-9-10.

[7] D. Recordon and D. Reed, "Openid 2.0: A platform for user-centric identity management," in *Proceedings of the Second ACM Workshop on Digital Identity Management*, DIM '06, (New York, NY, USA), pp. 11–16, ACM, 2006.

[8] M. D. Network, "Persona identity provider." `https://www.mozilla.org/en-US/persona/`, 2014. Acessed on: 2014-9-10.

[9] I. G. Plc., "Myid." `http://www.intercede.com/`, 2014. Acessed on: 2014-9-10.

[10] D. Kreutz, E. Feitosa, O. Malichevskyy, K. R. S. Barbosa, and H. Cunha, "A functional model for identification and authentication services tolerant to faults and intrusions," in *Proceedings of XIII Symposium on Information Security and Computer Systems*, SBC, 2013.

[11] D. Kreutz, O. Malichevskyy, E. Feitosa, K. R. S. Barbosa, and H. Cunha, "System design artifacts for resilient identification and authentication infrastructures," in *ICNS 2014, The Tenth International Conference on Networking and Services*, pp. 41–47, 2014.

[12] D. Kreutz, F. Ramos, and P. Verissimo, "Towards secure and dependable software-defined networks," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pp. 55–60, ACM, 2013.

[13] E. C. F. P. 7 and CNPq, "Secure future networks research project." `http://www.secfunet.eu/`, 2014. Acessed on: 2014-9-14.

[14] K. S. Trivedi, D. S. Kim, and R. Ghosh, "Resilience in computer systems and networks," in *Proceedings of the 2009 International Conference on Computer-Aided Design*, ICCAD '09, (New York, NY, USA), pp. 74–77, ACM, 2009.

[15] P. A. Dearnley, "An investigation into database resilience," *The Computer Journal*, vol. 19, no. 2, pp. 117–121, 1976.

[16] J.-C. Laprie, "Resilience for the scalability of dependability," in *Network Computing and Applications, Fourth IEEE International Symposium on*, pp. 5–6, July 2005.

[17] L. Strigini, "Resilience assessment and dependability benchmarking: challenges of prediction," in *DSN Workshop on Resilience Assessment and Dependability Benchmarking*, 2008.

[18] I. B. Barla, D. A. Schupke, and G. Carle, "Analysis of resilience in virtual networks," in *11th Wurzburg Workshop on IP: Joint ITG and Euro-NF Workshop Visions of Future Generation Networks (August 2011)*, 2011.

[19] D. Forsberg, "Secure distributed aaa with domain and user reputation," in *World of Wireless, Mobile and Multimedia Networks, 2007. WoWMoM 2007. IEEE International Symposium on a*, pp. 1–6, IEEE, 2007.

[20] J. N. Inc., "Steel belted radius carrier 7.0 administration and configuration guide," tech. rep., 2010.

[21] C. Messina, "Openid community wiki." http://wiki.openid.net/w/page/12995165/FrontPage, 2014. Acessed on: 2014-9-10.

[22] Clamshell, "Clamshell: An openid server." http://wiki.guruj.net/Clamshell!Home, 2014. Acessed on: 2014-9-10.

[23] D. Florencio and C. Herley, "A large-scale study of web password habits," in *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, (New York, NY, USA), pp. 657–666, ACM, 2007.

[24] S.-T. Sun, K. Hawkey, and K. Beznosov, "Systematically breaking and fixing openid security: Formal analysis, semi-automated empirical evaluation, and practical countermeasures," *Computers & Security*, vol. 31, no. 4, pp. 465 – 483, 2012.

[25] S. Gaw and E. W. Felten, "Password management strategies for online accounts," in *Proceedings of the Second Symposium on Usable Privacy and Security*, SOUPS '06, (New York, NY, USA), pp. 44–55, ACM, 2006.

[26] S.-T. Sun, E. Pospisil, I. Muslukhov, N. Dindar, K. Hawkey, and K. Beznosov, "What makes users refuse web single sign-on?: An empirical investigation of openid," in *Proceedings of the Seventh Symposium on Usable Privacy and Security*, SOUPS '11, (New York, NY, USA), pp. 4:1–4:20, ACM, 2011.

[27] J.-H. You and M.-S. Jun, "A mechanism to prevent rp phishing in openid system," in *Computer and Information Science (ICIS), 2010 IEEE/ACIS 9th International Conference on*, pp. 876–880, Aug 2010.

[28] *Signing Me onto Your Accounts through Facebook and Google: a Traffic-Guided Security Study of Commercially Deployed Single-Sign-On Web Services*, IEEE Computer Society, May 2012.

[29] V. R. Team, "Data breach investigations report," tech. rep., Verizon, 2013. http://www.verizonenterprise.com/DBIR/2013/.

[30] C. Tankard, "Advanced persistent threats and how to monitor and deter them," *Network Security*, vol. 2011, no. 8, pp. 16 – 19, 2011.

[31] Stuxnet.net, "All about stuxnet." http://www.stuxnet.net/, 2014. Acessed on: 2014-9-11.

[32] C. Rafeeq Ur Rehman, *The OpenID Book*. Conformix Books, a division of Conformix Technologies Inc., 2007. http://www.conformix.com.

[33] O. Foundation, "Openid foundation." `http://openid.net/foundation/`, 2014. Acessed on: 2014-9-11.

[34] J. De Clercq, "Single sign-on architectures," in *Infrastructure Security* (G. Davida, Y. Frankel, and O. Rees, eds.), vol. 2437 of *Lecture Notes in Computer Science*, pp. 40–58, Springer Berlin Heidelberg, 2002.

[35] D. Reed and D. McAlpin, "Extensible resource identifier syntax 2.0, oasis committee specification, oasis xri technical committee," 2005.

[36] L. Masinter, T. Berners-Lee, and R. T. Fielding, "Uniform resource identifier (uri): Generic syntax," 2005. `https://tools.ietf.org/html/rfc3986`.

[37] J. Miller, "Yadis specification 1.0," *Notes*, vol. 6, p. 3, 2006.

[38] G. Wachob, D. Reed, L. Chasen, W. Tan, and S. Churchill, "Extensible resource identifier (xri) resolution v2. 0," *OASIS, March*, 2005.

[39] O. Foundation, "Openid authentication 2.0 - final," 2007.

[40] A. Jøsang and S. Pope, "User centric identity management," in *AusCERT Asia Pacific Information Technology Security Conference*, p. 77, Citeseer, 2005.

[41] E. Tsyrklevich and V. Tsyrklevich, "Single sign-on for the internet: a security story," *July and August*, vol. 340, 2007.

[42] A. Barth, C. Jackson, and J. C. Mitchell, "Robust defenses for cross-site request forgery," in *Proceedings of the 15th ACM Conference on Computer and Communications Security*, CCS '08, (New York, NY, USA), pp. 75–88, ACM, 2008.

[43] A. Jain and J. Hodges, "Openid review." `https://sites.google.com/site/openidreview/issues`, 2009. Acessed on: 2014-9-11.

[44] B. Adida, "Sessionlock: Securing web sessions against eavesdropping," in *Proceedings of the 17th International Conference on World Wide Web*, WWW '08, (New York, NY, USA), pp. 517–524, ACM, 2008.

[45] K. Singh, H. Wang, A. Moshchuk, C. Jackson, and W. Lee, "Httpi for practical end-to-end web content integrity," tech. rep., Microsoft technical report, 2011.

[46] F. B. Schneider, "Implementing fault-tolerant services using the state machine approach: A tutorial," *ACM Comput. Surv.*, vol. 22, pp. 299–319, Dec. 1990.

[47] F. B. Schneider, D. Gries, and R. D. Schlichting, "Fault-tolerant broadcasts," *Science of Computer Programming*, vol. 4, no. 1, pp. 1 – 15, 1984.

[48] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *Dependable and Secure Computing, IEEE Transactions on*, vol. 1, pp. 11–33, Jan 2004.

[49] L. Lamport, R. Shostak, and M. Pease, "The byzantine generals problem," *ACM Trans. Program. Lang. Syst.*, vol. 4, pp. 382–401, July 1982.

[50] G. Candea and A. Fox, "Crash-only software.," in *HotOS*, vol. 3, pp. 67–72, 2003.

[51] R. D. Schlichting and F. B. Schneider, "Fail-stop processors: An approach to designing fault-tolerant computing systems," *ACM Trans. Comput. Syst.*, vol. 1, pp. 222–238, Aug. 1983.

[52] M. Castro and B. Liskov, "Practical byzantine fault tolerance and proactive recovery," *ACM Trans. Comput. Syst.*, vol. 20, pp. 398–461, Nov. 2002.

[53] A. Haeberlen, P. Kouznetsov, and P. Druschel, "The case for byzantine fault detection.," in *HotDep*, 2006.

[54] P. Sousa, N. Neves, and P. Verissimo, "Resilient state machine replication," in *Dependable Computing, 2005. Proceedings. 11th Pacific Rim International Symposium on*, p. 5 pp., Dec 2005.

[55] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *Commun. ACM*, vol. 21, pp. 558–565, July 1978.

[56] P. Marandi, M. Primi, and F. Pedone, "High performance state-machine replication," in *Dependable Systems Networks (DSN), 2011 IEEE/IFIP 41st International Conference on*, pp. 454–465, June 2011.

[57] P. Sousa, A. Bessani, M. Correia, N. Neves, and P. Verissimo, "Highly available intrusion-tolerant services with proactive-reactive recovery," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 21, pp. 452–465, April 2010.

[58] P. Sousa, A. Bessani, M. Correia, N. Neves, and P. Verissimo, "Resilient intrusion tolerance through proactive and reactive recovery," in *Dependable Computing, 2007. PRDC 2007. 13th Pacific Rim International Symposium on*, pp. 373–380, Dec 2007.

[59] A. Avizienis, "The n-version approach to fault-tolerant software," *Software Engineering, IEEE Transactions on*, vol. SE-11, pp. 1491–1501, Dec 1985.

[60] S. Forrest, A. Somayaji, and D. Ackley, "Building diverse computer systems," in *Operating Systems, 1997., The Sixth Workshop on Hot Topics in*, pp. 67–72, May 1997.

[61] B. Cox, D. Evans, A. Filipi, J. Rowanhill, W. Hu, J. Davidson, J. Knight, A. Nguyen-Tuong, and J. Hiser, "N-variant systems: A secretless framework for security through diversity," in *Proceedings of the 15th Conference on USENIX Security Symposium - Volume 15*, USENIX-SS'06, (Berkeley, CA, USA), USENIX Association, 2006.

[62] S. Neti, A. Somayaji, and M. E. Locasto, "Software diversity: Security, entropy and game theory," in *Proceedings of the 7th USENIX Conference on Hot Topics in Security*, HotSec'12, (Berkeley, CA, USA), pp. 5–5, USENIX Association, 2012.

[63] M. Garcia, A. Bessani, I. Gashi, N. Neves, and R. Obelheiro, "Analysis of operating system diversity for intrusion tolerance," *Software: Practice and Experience*, vol. 44, no. 6, pp. 735–770, 2014.

[64] M. Garcia, A. Bessani, I. Gashi, N. Neves, and R. Obelheiro, "Os diversity for intrusion tolerance: Myth or reality?," in *Dependable Systems Networks (DSN), 2011 IEEE/IFIP 41st International Conference on*, pp. 383–394, June 2011.

[65] I. Gashi, P. Popov, and L. Strigini, "Fault tolerance via diversity for off-the-shelf products: A study with sql database servers," *Dependable and Secure Computing, IEEE Transactions on*, vol. 4, pp. 280–294, Oct 2007.

[66] A. N. Bessani, E. P. Alchieri, M. Correia, and J. S. Fraga, "Depspace: A byzantine fault-tolerant coordination service," in *Proceedings of the 3rd ACM SIGOPS/EuroSys European Conference on Computer Systems 2008*, Eurosys '08, (New York, NY, USA), pp. 163–176, ACM, 2008.

[67] L. Lamport, "The part-time parliament," *ACM Trans. Comput. Syst.*, vol. 16, pp. 133–169, May 1998.

[68] J. Sousa and A. Bessani, "From byzantine consensus to bft state machine replication: A latency-optimal transformation," in *Dependable Computing Conference (EDCC), 2012 Ninth European*, pp. 37–48, May 2012.

[69] A. Clement, M. Kapritsos, S. Lee, Y. Wang, L. Alvisi, M. Dahlin, and T. Riche, "Upright cluster services," in *Proceedings of the ACM SIGOPS 22Nd Symposium on Operating Systems Principles*, SOSP '09, (New York, NY, USA), pp. 277–290, ACM, 2009.

[70] N. Santos and A. Schiper, "Achieving high-throughput state machine replication in multi-core systems," in *Distributed Computing Systems (ICDCS), 2013 IEEE 33rd International Conference on*, pp. 266–275, July 2013.

[71] A. Leicher, A. Schmidt, Y. Shah, and I. Cha, "Trusted computing enhanced openid," in *Internet Technology and Secured Transactions (ICITST), 2010 International Conference for*, pp. 1–8, Nov 2010.

[72] Q. Feng, K.-K. Tseng, J.-S. Pan, P. Cheng, and C. Chen, "New anti-phishing method with two types of passwords in openid system," in *Genetic and Evolutionary Computing (ICGEC), 2011 Fifth International Conference on*, pp. 69–72, Aug 2011.

[73] P. Urien, "An openid provider based on ssl smart cards," in *Consumer Communications and Networking Conference (CCNC), 2010 7th IEEE*, pp. 1–2, Jan 2010.

[74] B. van Delft and M. Oostdijk, "A security analysis of openid," in *Policies and Research in Identity Management* (E. de Leeuw, S. Fischer-Hübner, and L. Fritsch, eds.), vol. 343 of *IFIP Advances in Information and Communication Technology*, pp. 73–84, Springer Berlin Heidelberg, 2010.

[75] Y.-W. Kao, C.-T. Tsai, W.-H. Hung, S.-M. Yuan, and H.-T. Chiao, "A cross-platform p2p based blog system," in *Proceedings of the 7th International Conference on Advances in Mobile Computing and Multimedia*, MoMM '09, (New York, NY, USA), pp. 510–513, ACM, 2009.

[76] A. K. Kanuparthi, M. Zahran, and R. Karri, "Architecture support for dynamic integrity checking," *Information Forensics and Security, IEEE Transactions on*, vol. 7, no. 1, pp. 321–332, 2012.

[77] B. Parno, "Bootstrapping trust in a" trusted" platform.," in *HotSec*, 2008.

[78] T. Moyer, K. Butler, J. Schiffman, P. McDaniel, and T. Jaeger, "Scalable web content attestation," *Computers, IEEE Transactions on*, vol. 61, pp. 686–699, May 2012.

[79] O. Malichevskyy, D. Kreutz, M. Pasin, and A. Bessani, "O vigia dos vigias: um serviço radius resiliente," in *INForum*, 2012.

[80] L. Barreto, F. Siqueira, J. Fraga, and E. Feitosa, "An intrusion tolerant identity management infrastructure for cloud computing services," in *Web Services (ICWS), 2013 IEEE 20th International Conference on*, pp. 155–162, June 2013.

[81] M. Prince, "The ddos that almost broke the internet," *CloudFlare blog, March*, vol. 27, p. 2013, 2013.

[82] E. Basturk, R. Engel, R. Haas, V. Peris, and D. Saha, "Using network layer anycast for load distribution in the internet," in *TECH. REP., IBM T.J. WATSON RESEARCH CENTER*, 1997.

[83] T. Distler and H. P. Reiser, "Spare: Replicas on hold," in *Proceedings of the 18th Network and Distributed System Security Symposium*, pp. 407–420, 2011.

[84] B. Pfaff, J. Pettit, K. Amidon, M. Casado, T. Koponen, and S. Shenker, "Extending networking into the virtualization layer.," in *Hotnets*, 2009.

[85] G. Heiser, K. Elphinstone, I. Kuz, G. Klein, and S. M. Petters, "Towards trustworthy computing systems: Taking microkernels to the next level," *SIGOPS Oper. Syst. Rev.*, vol. 41, pp. 3–11, July 2007.

[86] Z. Wang and X. Jiang, "Hypersafe: A lightweight approach to provide lifetime hypervisor control-flow integrity," in *Security and Privacy (SP), 2010 IEEE Symposium on*, pp. 380–395, May 2010.

[87] M. J. Fischer, N. A. Lynch, and M. S. Paterson, "Impossibility of distributed consensus with one faulty process," *J. ACM*, vol. 32, pp. 374–382, Apr. 1985.

[88] P. E. Veríssimo, "Travelling through wormholes: A new look at distributed systems models," *SIGACT News*, vol. 37, pp. 66–81, Mar. 2006.

[89] P. Veríssimo, "Uncertainty and predictability: Can they be reconciled?," in *Future Directions in Distributed Computing* (A. Schiper, A. Shvartsman, H. Weatherspoon, and B. Zhao, eds.), vol. 2584 of *Lecture Notes in Computer Science*, pp. 108–113, Springer Berlin Heidelberg, 2003.

[90] M. Correia, P. Veríssimo, and N. Neves, "The design of a cots real-time distributed security kernel," in *Dependable Computing EDCC-4* (A. Bondavalli

and P. Thevenod-Fosse, eds.), vol. 2485 of *Lecture Notes in Computer Science*, pp. 234–252, Springer Berlin Heidelberg, 2002.

[91] P. Verissimo, N. Neves, C. Cachin, J. Poritz, D. Powell, Y. Deswarte, R. Stroud, and I. Welch, "Intrusion-tolerant middleware: the road to automatic security," *Security Privacy, IEEE*, vol. 4, pp. 54–62, July 2006.

[92] F. Araujo, R. Barbosa, and A. Casimiro, "Replication for dependability on virtualized cloud environments," in *Proceedings of the 10th International Workshop on Middleware for Grids, Clouds and e-Science*, MGC '12, (New York, NY, USA), pp. 2:1–2:6, ACM, 2012.

[93] M. Prince, "Ceasefires don't end cyberwars," *GI506*, 2012.

[94] V. V. Cogo, "Diversity in automatic cloud computing resource selection," Master's thesis, Science Faculty, Lisbon University, 2011.

[95] V. Cogo, A. Nogueira, J. a. Sousa, M. Pasin, H. Reiser, and A. Bessani, "Fitch: Supporting adaptive replicated services in the cloud," in *Distributed Applications and Interoperable Systems* (J. Dowling and F. Taïani, eds.), vol. 7891 of *Lecture Notes in Computer Science*, pp. 15–28, Springer Berlin Heidelberg, 2013.

[96] K. Cameron, R. Posch, and K. Rannenberg, "Appendix d. proposal for a common identity framework: A user-centric identity metasystem," 2009.

[97] H. J. Lee, I. Jeun, K. Chun, and J. Song, "A new anti-phishing method in openid," in *Emerging Security Information, Systems and Technologies, 2008. SECURWARE '08. Second International Conference on*, pp. 243–247, Aug 2008.

[98] D. Boger, L. Barreto, J. Fraga, H. Aissaoui, and P. Urien, "D3.2 identity management system development," tech. rep., SecFuNet Consortium, Aug 2013. `http://www.secfunet.eu`.

[99] SecFuNet, "D2.1 infrastructure of the authentication server," tech. rep., SecFuNet Consortium, 2012. `http://www.secfunet.eu`.

[100] C. Dwork, N. Lynch, and L. Stockmeyer, "Consensus in the presence of partial synchrony," *J. ACM*, vol. 35, pp. 288–323, Apr. 1988.

[101] OpenID4Java, "Openid 2.0 java libraries." `https://code.google.com/p/openid4java/`, 2014. Acessed on: 2014-9-13.

[102] A. N. Bessani and M. Santos, "Bft-smart-high-performance byzantine-fault-tolerant state machine replication," 2011. https://code.google.com/p/bft-smart/.

[103] P. Loscocco and S. Smalley, "Meeting critical security objectives with security-enhanced linux," in *Proceedings of the 2001 Ottawa Linux symposium*, pp. 115–134, 2001.

[104] M. Bernaschi, E. Gabrielli, and L. V. Mancini, "Remus: A security-enhanced operating system," *ACM Trans. Inf. Syst. Secur.*, vol. 5, pp. 36–61, Feb. 2002.

[105] P. Urien, E. Marie, and C. Kiennert, "A new convergent identity system based on eap-tls smart cards," in *Network and Information Systems Security (SAR-SSI), 2011 Conference on*, pp. 1–6, May 2011.

[106] P. Urien, E. Marie, and C. Kiennert, "An innovative solution for cloud computing authentication: Grids of eap-tls smart cards," in *Digital Telecommunications (ICDT), 2010 Fifth International Conference on*, pp. 22–27, June 2010.

[107] ITA and UECE, "Virtual network architecture and secure microcontroller user cases and first choices," tech. rep., SecFuNet Consortium, 2012. http://www.secfunet.eu.

[108] T. L. of the Bouncy Castle Inc., "The legion of the bouncy castle." https://www.bouncycastle.org/, feb 2014. Acessed on: 2014-9-13.

[109] E. Rescorla, "Diffie-hellman key agreement method," 1999.

[110] M. Urueña, A. Muñoz, and D. Larrabeiti, "Analysis of privacy vulnerabilities in single sign-on mechanisms for multimedia websites," *Multimedia Tools and Applications*, vol. 68, no. 1, pp. 159–176, 2014.

[111] S. Feld and N. Pohlmann, "Security analysis of openid, followed by a reference implementation of an npa-based openid provider," in *ISSE 2010 Securing Electronic Business Processes* (N. Pohlmann, H. Reimer, and W. Schneider, eds.), pp. 13–25, Vieweg+Teubner, 2011.

[112] O. S. Community, "Joids(java openid server). a multi-domain, multi-user openid provider." https://code.google.com/p/openid-server/, 2014. Acessed on: 2014-9-13.

[113] I. Amazon Web Services, "Amazon ec2 pricing." http://aws.amazon.com/pt/ec2/pricing/, 2014. Acessed on: 2014-9-13.

[114] N. Sakimura (NRI), J. Bradley (Ping Identity), M. Jones (Microsoft), B. d. Medeiros (Google), and C. Mortimore (Salesforce), "Openid connect core 1.0," 2007.