Towards Secure and Dependable Authentication and Authorization Infrastructures

Diego Kreutz, Alysson Bessani
Faculdade de Ciências, Universidade de Lisboa, Portugal
kreutz@lasige.di.fc.ul.pt, bessani@di.fc.ul.pt

Eduardo Feitosa, Hugo Cunha
Computing Institute, Federal University of Amazonas
{efeitosa,hugo.cunha}@icomp.ufam.edu.br

***Abstract*—We propose a resilience architecture for improving the security and dependability of authentication and authorization infrastructures, in particular the ones based on RADIUS and OpenID. This architecture employs intrusion-tolerant replication, trusted components and untrusted gateways to provide survivable services ensuring compatibility with standard protocols. The architecture was instantiated in two prototypes, one implementing RADIUS and another implementing OpenID. These prototypes were evaluated in fault-free executions, under faults, under attack, and in diverse computing environments. The results show that, beyond being more secure and dependable, our prototypes are capable of achieving the performance requirements of enterprise environments, such as IT infrastructures with more than 400k users.**

***Keywords*-authentication and authorization services, security, dependability, intrusion tolerance, RADIUS, OpenID.**

I. INTRODUCTION

Despite their widespread use, Authentication and Authorization Infrastructures (AAIs) such as RADIUS-based network access controllers does not implement security mechanisms able to tolerate advanced persistent threads [1], [2], large scale distributed denial of service [3], [4] or even to protect authentication information confidentiality in case of intrusions. Beyond that, current available and widely deployed services such as RADIUS and OpenID have several vulnerabilities regarding their security and dependability [5]. Given the importance AAIs in modern networked systems, these limitations can be considered as one of top threats in future network environments [5].

In this work we tackle the problem of increasing the resilience of AAIs, which represent key services for ensuring the security and reliability of most networked environments. More specifically, we describe a novel architecture and baseline building blocks for designing and deploying more secure and dependable authentication and authorization infrastructures without sacrificing the system scalability. This architecture is based on the use of intrusion-tolerant replication [6], [7] (for ensuring correct operation even if some of the system components are compromised), well-defined Trusted Components (TCs – for protecting the confidentiality of key material even in case of intrusions) and untrusted gateways (for maintaining compatibility with current clients and protocols).

The proposed architecture can be applied in services such as RADIUS, OpenID, Diameter, TACACS+, and so forth, since all of them have a conceptually similar architecture. In fact, the essential elements are basically the same, changing mostly the stack of protocols and other specific properties and functions. In this paper we instantiate the architecture for two authentication services, RADIUS and OpenID. Their design and implementation is depicted and evaluated, showing that it is possible to significantly improve the resilience and security of AAIs, without imposing significant penalties on its scalability. Furthermore, our results indicate that more secure and dependable AAIs can be developed and deployed to support the demand of IT infrastructures with more than 400k users. Lastly, but not less important, the system design also keeps backward compatibility with corresponding existing technologies and protocols.

The main contributions of this paper are: (a) a step-by-step design and discussion of a resilient architecture for authentication and authorization infrastructures using as reference two use cases, RADIUS and OpenID; (b) a novel trusted component for ensuring the confidentiality of sensitive data stored in replicated systems subject to malicious faults; (c) experimental evaluation in three different environments, showing that it is feasible to take advantage of different physical infrastructure (e.g., data centers, clouds) for deploying and increasing the robustness of systems against first class attacks such as large scale DDoS attacks.

II. PROBLEM STATEMENT

A. Weaknesses of Current Authentication Systems

Figures 1 and 2 illustrate the traditional architecture of the RADIUS and OpenID services, respectively. As can be seen in the figures, the architectures are quite similar since in both cases there are clients/supplicants, services (NAS and SP/Relying Party), authentication servers and backends.

In a typical RADIUS architecture, the network access server (NAS) and authentication, authorization and accounting (AAA) servers share a secret that is used to ensure communication confidentiality and integrity. On receiving the supplicant's credentials, the AAA server can send it to a distinguished back-end to be validated, which can be also another AAA server on a federation, as it is the case of *eduroam* [8].

In an OpenID infrastructure, relying parties act as a bridge

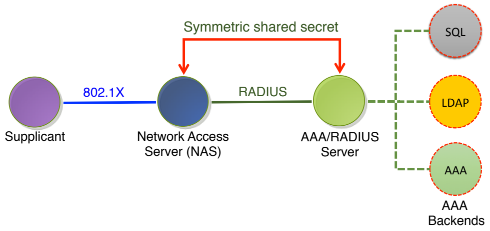

Figure 1. Current RADIUS architecture.

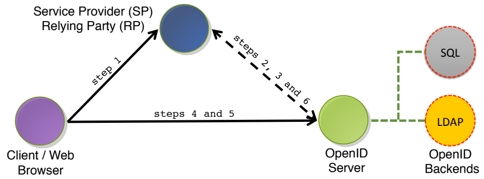

Figure 2. Current OpenID architecture.

between the online service or application (e.g., Facebook, e-commerce sites) and the OpenID service. In terms of functionality, relying parties allow service providers to use external OpenID-based authentication services.

Regarding fault tolerance, one of the ways to improve it in RADIUS services is by using multiple instances of FreeRADIUS and a cluster of MySQL servers [9]. Similarly, to improve the availability of OpenID servers a replicated LDAP backend can used to provide session data and configuration distribution among multiple servers [10]. However, these solutions have two limitations. First, they augment the complexity of the service infrastructure. To tolerate one single fault it is necessary to instantiate two RADIUS servers and two backends (e.g., MySQL servers). Both have to be configured for high availability. It is worth emphasizing that these services are independently configured, which poses more risk to configuration errors, logical problems, and so forth. This is indeed a concerning affair since configuration problems are one of the major issues in critical IT infrastructures [11]. Second, even though with replicas, the services are capable of tolerating only fail-stop behavior, such as server crashes or network outages (e.g., connection disruption with one server). Any abnormal/arbitrary behavior (e.g., a glitch in a component, a defective operating system) or intrusion can potentially compromise the correct operation of the system.

Another issue worth mentioning is the fact that those services are commonly deployed in a single physical infrastructure. This is still a wide spread practice in most enterprise environments, with some forefront exceptions such as cloud providers, which have several data centers in different locations. For instance, in an interconnected and interdependent world, an power outage in one AAI could impact many uses in different locations. This is the case of *eduroam* [8], where an power outage in one university will simply deny thousands of users to access resources in different locations. Therefore, it is of paramount importance to design AAIs that can be easily deployed across multiple physical infrastructures, such as data centers and clouds.

This scenario is becoming critical once AAIs are being deployed for ensuring the security (e.g., access control) of large-scale distributed virtual networks, controlling not only users but also virtual routers and virtual machines. Therefore, their security and dependability is crucial for ensuring different security, availability and correct operation properties of future networks and IT infrastructures [12], [13].

B. Limitations of Intrusion-Tolerant BFT Systems

The state machine replication (SMR) approach [14] is a widely used paradigm in the implementation of Byzantine fault-tolerant (BFT) protocols [15]. From the client perspective, the replicas of the service work as a single logically-centralized service, with the replicas executing all submitted operations in a coordinated way. For implementing SMR-based services, three requirements need to be satisfied: (i) all correct replicas start on the same initial state; (ii) all correct replicas produce the same output and resulting state if receiving the same input on the same state; (iii) all correct replicas process the same sequence of commands.

The safety of the system is related with the fact that all correct replicas must execute the same sequence of operations, while the liveness requires that all correct clients operations are eventually executed. These requirements and properties allows the implementation of fault-tolerant and strongly-consistent services as long as the SMR machinery is available. Programming libraries such as BFT-SMaRt [16] and PBFT [15] encapsulate such machinery.

Byzantine fault-tolerant protocols are a required building block for implementing intrusion-tolerant systems [6]. However, these protocols by themselves are not enough [7]. A system is intrusion-tolerant if it is capable of ensuring its security properties under adversarial circumstances, such as an intruder with access to some of its components. Therefore, other techniques, complementary to BFT, such as proactive-reactive recovery, diversity and confidential operation are required to rejuvenate or replace components of the system, protect against parallel attacks on common vulnerabilities, and ensure sensitive data confidentiality despite of a malicious user or attacker getting access to some of the components of the system.

C. Ensuring the Confidentiality of Sensitive Data

Ensure the confidentiality of sensitive data is one of the cornerstones of intrusion-tolerant systems. In AAIs using protocols such as SSL and TLS for authentication purposes, special attention has to be given to certificates, RSA private keys and TLS master secrets. For instance, in common deployments anyone with access to the RADIUS or OpenID services would have access to the server keys. Both a malicious sysadmin and an intruder have easy access to sensitive data that, if leaked or used for malign purposes, can compromise the system operation, reliability or even affect the enterprise business.

Similarly, in an academic environment, any compromised RADIUS server on an federated RADIUS architecture (e.g. *eduroam* [8]) is a major security issue. A malicious user can easily take advantage of a compromised RADIUS server of any institution integrating the federation to get access to infrastructure resources (e.g., networks, services) in different institutions in several countries. These examples show how critical is has become to protect sensitive data, in particular if we look at the increasing number of threats and advanced persistent threats in the cyber space [17].

III. OUR SOLUTION

A. Overview

Figure 3 presents the high-level system's architecture, which extends our previously defined funcional model [18] and is composed by five types of components: clients, services, gateways, AAI replicas and AAI secure components. The architecture considers one TC per AAI replica. The gateways are used in the system to keep backward compatibility, essentially hiding the replication techniques used by BFT protocols. Replicas of services and gateways can be used by clients and/or services to reach the AAI through different paths.

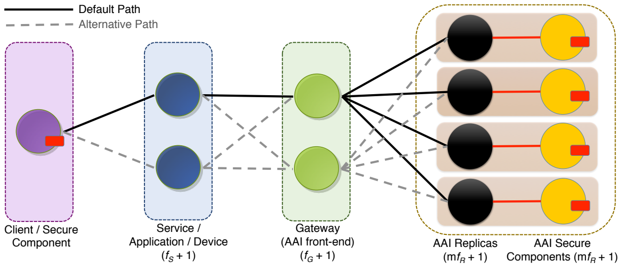
Default Path
Alternative Path
Client / Secure Component
Service / Application / Device ($f_S + 1$)
Gateway (AAI front-end) ($f_G + 1$)
AAI Replicas ($mf_R + 1$)
AAI Secure Components ($mf_R + 1$)

Figure 3. Overview of proposed architecture for authentication services.

B. System Model

Network model. We assume best effort networks (e.g., TCP/IP, UDP/IP) between clients, services and gateways. The client is able to connect to any service using standard protocols such as IEEE 802.1X for WiFi and RADIUS, and HTTP/HTTPS for OpenID. Additionally, gateways are able to communicate with all replicas of the AAI through the BFT protocols. Hence, the packets received by the gateway (from the service and/or client) are encapsulated in these protocols and sent to the replicas of the authentication infrastructure.

Synchrony model. We assume partial synchrony, i.e., the system can behave asynchronously for some time, until it becomes synchronous, when there will be some unknown processing and communication time bounds. This is an essential requirement to ensure the termination of consensus protocols, a fundamental building block for implementing SMR.

Fault model. The architecture comprises an unbounded number of clients, f_S+1 services, f_G+1 gateways and $3f_R+1$ authentication replicas, where f_S, f_G and f_R represent the maximum number of simultaneous faults tolerated by each type of element. Additionally, an unbounded number of clients can be faulty.

Each replica of the AAI has also access to at least one TC, which is required to securely store and processing sensitive cryptographic material. This component is used for securely storing shared and private keys. Replicas have a minimal interface to request cryptographic operations on the keys stored in the TCs. These components are assumed to fail only by crash. All other system components can be subject to Byzantine (arbitrary) faults.

Trusted third party certificate authority (CA). We assume that all users and authentication servers have a valid certificate issued by a trusted third party CA. We rely on a public key infrastructure (PKI) as the anchor of trust, which is something that is becoming of common use in enterprise environments [19]. Additionally, all TCs know the CA's public key (Pu_{CA}). This certificate is required to verify the user's identity, based on the CA's digital signature.

C. Adversary Model

We assume that an attacker can: (i) simultaneously compromise up to f_S services, f_G gateways, and f_R replicas, respectively; (ii) intercept and generate messages; and (iii) drop messages in transit. Moreover, our system design consider a threat model where the attacker can have control of the communication channels among the system elements. An attacker is able to eavesdrop packets in transit, drop, replay and insert packets into the communication channels [20]. However, we assume that the attacker is not able to break any cryptographic mechanisms without obtaining the appropriate cryptographic keys. Furthermore, the attacker is not able to subvert the trusted component.

We also assume that an attacker can have control of the network for some time, but cannot control the whole network for the whole time. This is a reasonable assumption since all elements of the system, including replicas, can be running on different physical infrastructures and, consequently, it is highly unlikely for an attacker to get control over communication channels or resources on distinct places.

D. Building Blocks

Byzantine fault tolerance (BFT). Intrusion-tolerant systems must be capable of tolerating both accidental and malicious faults. There are different extensions that can be made on

BFT SMR systems to tolerate intrusions, such as adding and managing diversity, ensuring confidential operations of sensitive data, proactive-reactive recovery mechanisms, and mechanisms for ensuring a graceful system degradation [7]. Ensuring the confidentiality of sensitive data and operations is another critical aspect for tolerating intrusions.

As one of the essential building blocks of our resilient AAIs, we assume the use of BFT-SMaRt [16], [21], a publicly-available high-performance BFT SMR library that implements a protocol similar to PBFT [15].

The protocols used in BFT-SMaRt assume a system model with partial synchrony and $3f_R + 1$ replicas for tolerating up to f_R Byzantine failures.

Secret Sharing. Secret sharing schemes are methods that can be used to enhance the protection of confidential data. Such schemes allow the creation of n shares of a secret in such a way that at least t of them are needed to rebuild the secret [22]. Furthermore, no information about the secret is revealed with less than t shares.

In our system design, a share can be attributed to each TC for generating a valid signature (e.g., HMAC, digital signature). Therefore, a collusion of up to $f_R < t$ replicas will not be able to forge or fake authentication requests.

Trusted Component. We propose a simple TC that can be implemented with technologies such as Intel SGX [23]. We store user data, shared keys required by the service, and session data outside the TC, but ciphered or verifiable (e.g., MAC, RSA signature) with the keys that are only known by the TC. Consequently, an intruder would not be able to modify or read any sensitive data stored in the replica, which is encrypted or signed by the key owner (the TC).

Figure 4 illustrates the proposed TC and its interaction with an authentication replica. As can be observed, a single TC can be used to provide the functions required by both RADIUS and OpenID-based authentication. The only thing that changes is the data table. In the case of RADIUS, the ID of the NAS is used as the key of the table, whereas for OpenID the association handler is used with the same purpose. In practice, there will be one table per service. Here, these tables have been merged due to space constraints.

This TC stores six critical informations: `ID`, K_{User}, Pr_S, Pu_{CA}, K_{NAS}, and K_{Assoc}. The tables with the users' IDs and permission, as well the NASes' IDs and shared secrets (for RADIUS) and association handler and key-pair (for OpenID), are stored in the replicas.

The TC's `ID` and Pr_S are used by the secret sharing algorithm to generate the share that will be sent to the gateway. In this case, each replica has to request the TC to execute the `Confidential` method, which will output a share which reveals no relevant information alone. As a practical example, the signature of a packet can be partially generated by each replica/TC. Each TC knows which part (share) of the packet's signature it has to generate. These "partial signatures" (shares of the secret, i.e., the packet's signature) are replied to the gateway, which waits for a sufficient amount of shares to build the correct packet signature. In fact, by using a secret sharing algorithm [22], the gateway have to wait for at least $3f_R + 1 - f_R$ "parts of the packet's signature" (shares), and only then assemble a valid signature to the NAS. Thus, if an attacker (controlling a gateway) eventually forwards messages of up to f_R compromised replicas, both the NAS and supplicant would detect it right away.

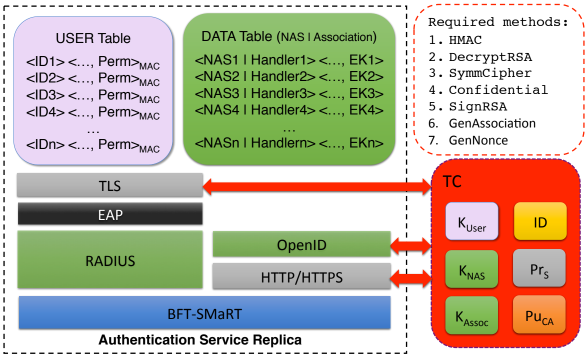

Figure 4. Trusted component (TC) for dependable authentication services.

The K_{User} is used to verify the user's `id` and permissions. Therefore, an attacker is not able to generate or modify a valid entry in the user table without knowing this secret. Only the TC can generate a valid MAC for the table's entries. Alternatively, a system administrator could also know the secret key in order to modify (`add`, `remove`, etc.) the users' table in the system. In this case, both the TC and the system administrator would be able to update the table.

Similarly, the K_{NAS} secret key is required to ensure the confidentiality of the shared keys stored in the NAS table. Consequently, only by knowing this secret key one would be able to encrypt and decrypt the NASes' shared keys. In the same way, the OpendID uses the K_{Assoc} key and a symmetric cipher (`SymmCipher`) to securely store the association information in the replica's file system.

The Pu_{CA} is required to verify the user's ID. We assume that valid users have a valid certificate generated using the private key of the CA. Therefore, by using the public key the TC is able to verify the user's identity.

The server private key (Pr_S) is used by the TLS/SSL for mutual authentication. This key is used to decrypt messages (`DecryptRSA`) encrypted using the server's public key.

Lastly, the `GenNonce` creates a nonce using a timestamp and a pseudo random number, also required by the OpenID 2.0 specification. This nonce is used to avoid replay attacks with the authentication response.

Table I summarizes the interface methods of the TC. As can be observed, there are only seven well-defined methods, with respective input and output parameters.

Method	Protocol	Input	Output
`DecryptRSA`	TLS	Packet to be verified.	Status of the signature verification.
`SignRSA`	TLS	Data to sign.	RSA signature using the key Pr_S.
`SymmCipher`	RADIUS&TLS	Protocol id and data.	Ciphered output of the input data.
`Confidential`	TLS	The packet data.	A confidential `share` of the data.
`HMAC`	RADIUS	`data` + encrypted shared key.	`HMACMD5` of the input data.
`GenAssoc`	OpenID	Public key and two big integers.	Association info + server's public key.
`GenNonce`	OpenID	Two big integers.	Pseudo random nonce.

Table I
INTERFACE FOR TLS, RADIUS AND OPENID

IV. INRUSION-TOLERANT AAIS

In this section we introduce our reference implementations (RADIUS and OpenID) for intrusion-tolerant AAIs.

A. *Resilient RADIUS Architecture*

One of the first challenges in designing a resilient RADIUS architecture is the backward compatibility, i.e., to create a more secure and dependable AAA service that can be employed in existing infrastructures without requiring any modification on them. Figure 5 illustrates our resilient AAA architecture, whereas Figure 1 shows the traditional RADIUS architecture. As can be observed, a new component, called gateway, is introduced to hide the AAA server replication. Its main function is to provide transparent communications between traditional NASes and the replicated AAA service. In this way, a NAS does not need to know that it is relying an authentication request to a resilient architecture, since the RADIUS server is represented by the gateway.

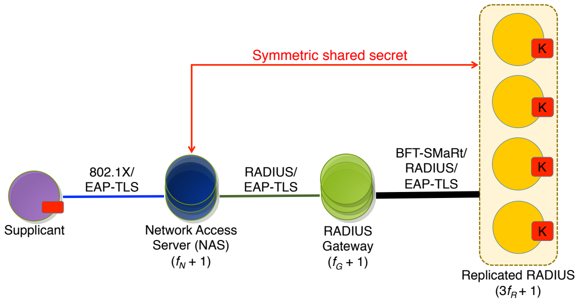

Figure 5. Resilient RADIUS Architecture.

Supplicant and NASes do not need to be modified in any way on our architecture, which means that any existing supplicant supporting EAP-TLS authentication can be used. Similarly, any NAS that supports RADIUS can take advantage of our fault- and intrusion-tolerant AAA service.

The replicated AAA server leverages the BFT SMR to tolerate faults and intrusions. Each replica of our system has its own trusted component (K) as well, which is responsible for ensuring the confidentiality of all sensitive data (e.g. server and CA keys).

Another important observation is that the resilient RADIUS architecture assumes that there are shared secrets between the NAS and authentication replicas. Consequently, communications between these two elements can be encrypted using the shared secret and cannot be simply unveiled in the gateway element. As the gateway does not know the shared secret, it cannot generate valid packets.

Resilient RADIUS communication pattern. Figure 6 describes the communication pattern between the different elements of the architecture. A supplicant uses the IEEE 802.1X protocol combined with the WPA-Enterprise mode and EAP-TLS authentication method. IEEE 802.1X is required to communicate with the network access server, while EAP-TLS is used to provide end-to-end secure authentication between the supplicant and the trusted components.

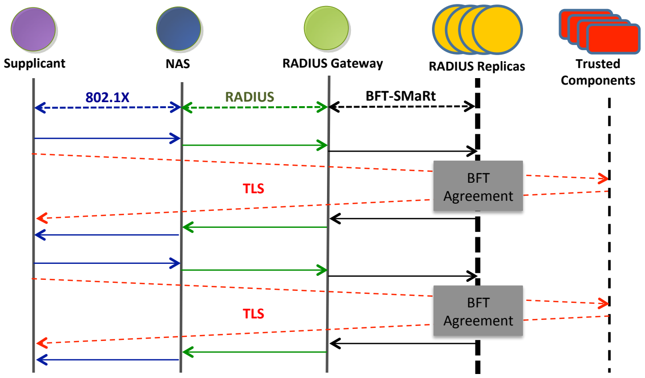

Figure 6. Resilient AAA communication pattens.

The BFT-SMaRt [21] library was chosen to provide the replication protocols for tolerating arbitrary faults on the AAA server. This library provides two interfaces, one for the client side, in our case the gateway, and another one for the server side, the replicated AAA server. The client interface allows the gateway to communicate with the system's replicas in a transparent way. Differently, the server side interface allows one to build a replicated service without having to implement protocols such as leader election and total ordering.

EAP-TLS allows a strong end-to-end authentication. After a successful authentication, the replicas send also a master key to the NAS. It is encrypted with the shared secret between NAS and AAA replicas. This master key is used to cipher subsequent communications between the supplicant and NAS.

The EAP-TLS handshake process requires a random number generation at the server side. This is not a problem on a standalone service. However, it is indeed a problem on a replicated service because of the required determinism of the replicas [14]. Therefore, an adapted deterministic random generator is required on the TCs.

B. *Resilient OpenID Architecture*

Figure 7 shows the resilient OpenID architecture, whereas Figure 2 illustrates a typical architecture. 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) the OpenID gateway, which provides backward compatibility and mask the replicated infrastructure; (d) an OpenID server which does not authenticate users; (e) authentication backend secure component (K), which is responsible for keeping the confidentiality of the users' IDs and other sensitive information, as well as doing the user authentication; and (f) the identity provider composed by the OpenID server and the trusted components. As can be observed, it is assumed that there are secure elements both in the client-side (e.g. smart card, TPM) and in the server-side (e.g. secure and isolated software component on a trusted computing base) [18].

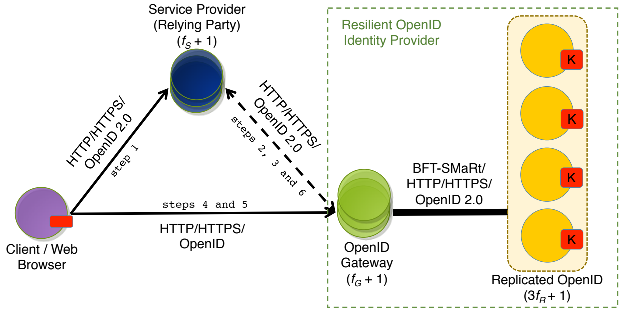

Figure 7. Resilient OpenID architecture.

The only stateful elements are the OpenID server and the TCs. Similarly to the RADIUS prototype, the OpenID gateway and relying parties are stateless elements. Thus, they can be easily replicated for increased availability and fault tolerance (e.g. fail-stop). Lastly, the user authentication follows the standard OpenID protocol.

Resilient OpenID communication pattern. Figure 8 shows the communication steps among the elements in the resilient OpenID system. Essentially, it follows the OpenID 2.0 specification. All requests send from clients and relying parties pass through the OpenID gateway to reach the service replicas. Moreover, steps 7, 8, 13 and 14 require interactions with the trusted components.

The first successful connection with a gateway is used for the association request (step 6). This type of request contains data such as the respective endpoint URL and the relying party's Diffie-Hellman data (public key, modulus prime number, generator number) to secure the association.

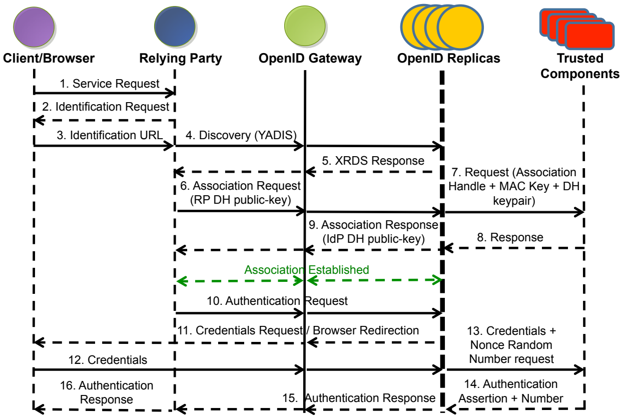

Figure 8. OpenID interactions among the system's components.

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). Next, the OpenID server requests a 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, is sent to the relying party (step 9), completing the association.

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). Next, the OpenID server requests the credentials' verification to the trusted component. Additionally, it also requests a 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, concluding the authentication process (step 16).

C. *Implementation*

In this section we briefly describe the gateway of each service and the trusted component. The remaining elements (clients/supplicants, NAS/relying party, and service replicas) follow the architecture and communication patterns described in the previous section.

RADIUS Gateway. The gateway has two interfaces, one to communicate with NAS elements through the RADIUS protocol, and a second one to interact with BFT-SMaRt. In simple terms, supplicant requests arrive at the gateway's NAS interface and are forwarded to the BFT-SMaRt interface.

Additionally, the gateway also contains complementary functions used to detect and avoid forwarding malformed packets. Each received packet, from both interfaces, is verified against a packet format template. Only well-formed packets are forwarded.

OpenID Gateway. To be backwards compatible with existing OpenID services, the gateways needs to accept TCP/IP connections (with HTTP and/or HTTP over SSL) from OpenID clients/relying parties and encapsulate the received packets in the BFT SMR protocol. Similarly, packets coming from replicas are sent to the clients through the previously established TCP/IP connections.

Trusted Component. In our first prototype, we assume that the trusted component can be executed in an isolated and secured special-crafted virtual machine, with the help of the isolation properties provided by modern hypervisors. We implemented the TC using Java and the BouncyCastle cryptographic library [24]. One of the challenges of implementing the trusted components in the OpenID replicas is ensuring deterministic operation, one of the requirements of the SMR approach. This is particularly important for the pseudo-random function (PRF) used in the TLS protocol. We solve this problem using a special seed to implement a progressive and deterministic pseudo-random number generator. This seed is calculated using `HMAC-MD5` of the server private key, known only by the trusted component, the client certificate and the previous seed. This progressive seed ensures a secure, yet deterministic, random number generation. If an attacker wishes to guess the next random number or the current seed, he will need to know the initial seed (stored inside the trusted component), the server's private key, and the exact sequence order of all client certificates used for the seed evolution.

V. EVALUATION

In this section we show an evaluation of both prototypes in fault-free executions and under faults (e.g., up to f_R replicas subject to crash and Byzantine behaviors) and attacks.

A. *Resilient RADIUS*

For RADIUS evaluation, both `FreeRADIUS` and our resilient RADIUS were analyzed considering the same faults, such as delay attack and components behaving in an arbitrary way. Throughout out tests the NAS elements were configured with a timeout of 3s and 3 retries for each authentication request. This is a common NAS configuration in real-world deployments.

Performance analysis. The testing environment was composed of machines with Quad-core Intel Xeon E5520 processors, 32 GB (8x4GB) of RAM, and Broadcom NetXtreme II BCM5716 Gigabit Ethernet network cards. All machines are interconnected through two Dell PowerConnect 5448 switches with 48 Gigabit Ethernet ports each. We used 7 machines for the replicated RADIUS and 3 machines for the `FreeRADIUS`. Our solution uses one client machine (supplicant + NAS), two gateways, and four RADIUS replicas, while the `FreeRADIUS` environment uses one client (supplicant + NAS) and two server replicas. This means that `FreeRADIUS` is able to tolerate up to one crash fault on the RADIUS server without compromising the system operation, while our solution is capable of tolerating one fault on the gateways and one fault on the RADIUS replicas.

Latency. We used 2 to 10 supplicants doing 10k EAP-TLS authentications each. In this test, we have observed that the latency remains nearly constant for both systems. However, the resilient RADIUS adds some extra latency when compared to the original AAA architecture. One resilient RADIUS authentication takes approximately 200ms, while in `FreeRADIUS` it takes around 100ms.

Throughput. We used 2 to 20 supplicants executing 10k authentications, where each authentication requires ten packets. To measure the throughput we wrote a C program to count the number of packets cached by the `tcpdump`.

For the resilient RADIUS, the throughput remains almost stable, a bit over 100 authentications/s. For `FreeRADIUS`, the throughput varies between 100 to 175 authentications/s due to the thread pool police employed in the system, which automatically increases the number of working threads (between 3 and 30, in our configuration) based on the number of authentication requests.

Fail-stop fault of one replica. Table II shows that clients are able to authenticate even in case of one crashed AAA replica. However, supplicants may experience up to 9s of delay with one crashed gateway. If the crashed gateway is the first on the list of available gateways in the NAS, it will try to authenticate the user on the crashed gateway for three times, with a timeout of 3s for each retry. Obviously, these configuration parameters can be changed. For instance, one could configure the NAS with a 500ms interval between each retry. In this case the user would experience an authentication delay of only 1.5s.

Attack	FreeRADIUS	RADIUS replica	RADIUS gateway
Fail-stop	9s of delay.	No authentication delay.	9s of delay.
Byzantine	Max delay of 9s.	No delay.	Up to 9s of delay.

Table II
ATTACK RESULTS FOR ONE FAULTY RADIUS SERVER.

Byzantine faults in one replica. To simulate simple arbitrary behavior, two types of faults were injected: malformed packets and communication delays. The results under this kind of fault are also presented in Table II.

First, the system replicas were modified to produce malformed packets. In this case, we introduced an invalid packet code. In the second case, a delay was introduced on the packets delivery. Essentially, replicas were configured to exceed the NAS timeout before sending the responde. Therefore, for each request sent by the NAS to the AAA server, a re-transmission would be triggered due to the replicas' delay.

In the delay attack users could still be successfully authenticated despite the existence of one faulty replica. However, during this attack supplicants may have to retry the authentication, causing extra load on the NAS. A high delay on the replicas' response can eventually compromise the NAS's operation since its resources can potentially be exhausted if there are too many clients trying to authenticate.

Considering the malformed packets attack, the replicated RADIUS is not affected with a single compromised replica since it does a majority voting. On the other hand, the `FreeRADIUS` NAS is forced to change to another replica when it receives a malformed response. Naturally, this adds some extra communication delays on the authentication process. While `FreeRADIUS` can reach a delay of 9s, the replicated RADIUS was able to keep authenticating supplicants without any noticeable delay. However, with one compromised gateway the replicated RADIUS will experience a similar delay of up to 9s.

Moreover, considering typical deployments of a `FreeRADIUS` server, a compromised replica can also leak the derived TLS master key. This is a relevant issue since it would allow an attacker to intercept the subsequent communications between supplicants and the NAS. Even worse, a Byzantine `FreeRADIUS` server may deny access to the legitimate user. On the other hand, a compromised replica of our resilient RADIUS service would not compromise the TLS master secrets or any other sensitive data because they are stored in an isolated trusted component.

B. *Resilient OpenID*

We used three different environments for evaluating the performance of our prototype, as summarized in Table III. In each environment we instantiated five computing units, one to execute the gateway and clients and the other four for running the replicas and the trusted components. In the first environment, Quinta-VMs (Quinta-VMsR and Quinta-VMsG), the VMs were running Ubuntu Server 12.04 LTS top of the KVM *hypervisor*. Quinta-VMsR represents the configuration used for the OpenID replicas, while Quinta-VMsG was used for running the gateway and the OpenID clients. These virtual machine were running on a single physical machine of Quinta-PHY. The second environment, Quinta-PHY, was composed of five physical machines running Ubuntu Server 12.04 LTS. On the third environment, Amazon-DCs, we used `m3.large` [25] computing node (running Ubuntu Server 14.04 LTS) on the Amazon's N. California and Oregon DCs.

Raw performance in fault-free executions: Table IV summarizes the main results of our executions in the three environments. We used a variable number of clients (10, 20, 40, 80, 100, 200) running together with the gateway. Each client executes 1k authentication requests, each of them comprised of five messages as described in the following paragraph. Additionally, we executed each configuration (e.g., 10 clients) on each environment 100x.

Environment	**vCPUs**	**ECUs**	**MEM**	**Disk**	**Network**
Quinta-VMsR	3	—	4GB	4GB (qcow2)	Gigabit Ethernet
Quinta-VMsG	6	—	8GB	4GB (qcow2)	Gigabit Ethernet
Quinta-PHY	16	—	32GB	146GB SCSI	Gigabit Ethernet
Amazon-DCs	2	6.5	7.5GB	1 x 32GB SSD	public WAN

Table III
ENVIRONMENTS AND CONFIGURATION OF THE VMS

The OpenID authentication process consists essentially in five HTTP messages. The first two messages, used by the discovery process, have no payload (only the HTTP method). The following 3 messages have 343B, 506B, and 537B, respectively. These messages follow the standard OpenID 2.0 specification. The size of these messages can vary accordingly to the respective relying party and user credentials.

Environment	**10**	**20**	**40**	**80**	**100**	**200**
Quinta-VMs	501	769	986	1077	1136	1424
Quinta-PHY	1489	2540	3487	4719	5011	5290
Amazon-DCs	62	111	210	401	489	704

Table IV
# AUTHS/S WITH 10, 20, 40, 80, 100 AND 200 CLIENTS

As can be observed in Table IV, the number of authentications/s varies from nearly 501 (with 10 clients) to 1424 (with 200 clients) in the Quinta-VMs. The throughput keeps increasing from 10 to 200 clients. However, as the number of clients grow, the gains in performance get smaller. A similar behavior occurs in the second environment. This is due to the execution environment limits. Too many simultaneous clients lead 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 scaling in performance.

Although the network latency is the major limitation of the Amazon-DCs environment, we considered the results achieved. With 10 clients we achieve a throughput of 62 authentications/s, while approximately 704 authentication/s with 200 clients. An authentication service with such number of authentications/s can be considered of medium scale. It is necessary at least thousands of users (e.g., 60k) in the system to reach more than one hundred authentications/s. In fact, by collecting and analyzing statistics of a real enterprise environment, we figured out that our results from the Quinta-PHY environment are enough to support a university-like environment with more than 400k users.

For measuring the network latency we used the `ping` tool to generate one thousand ICMP packets of 512B (which is approximately the average OpenID payload size). Both

Quinta-PHY (0.123ms) and Quinta-VMs (0.182ms) have a low latency. Considering the network latency and computing power, the Quinta-PHY environment reached a throughput 297% higher than Quinta-VMs for 10 clients and 371% higher for 200 clients. Yet, the network latency (in average 20.28ms) between Oregon and N. California data centers led the system to a throughput 8x lower (for 10 clients) and only 2x lower with 200 clients when compared to the Quinta-VMs environment. Obviously, this means that the number of clients compensates the network latency, i.e., with similar computing power capacity and a higher number of clients, the inter-DCs deployment will achieve a similar performance of the other environments.

Constantly crashing up to f_R replicas. To evaluate the system performance under harsh circumstances on the Quinta-VMs. We implemented a script that kills and restarts one of the replica every 1s, 2s, 4s, 8s, and 16s 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 throughput nearly the fault-free execution (FF_Exec) up to 40 clients and a slightly higher than the fault-free executions for more than 40 clients, as shown in Figure 9. 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 problem or significant performance degradation with constant crashes of up to f_R replicas in different time intervals. In average, considering crash faults, the lower performance is achieved by higher crash intervals (i.e., 8s and 16s). Differently from short intervals crashes, which keeps one replica nearly always slower than the others, the replica replacement occur less frequently and, as a consequence, it can affect one of the fast replicas, causing as a result a greater impact on the system performance.

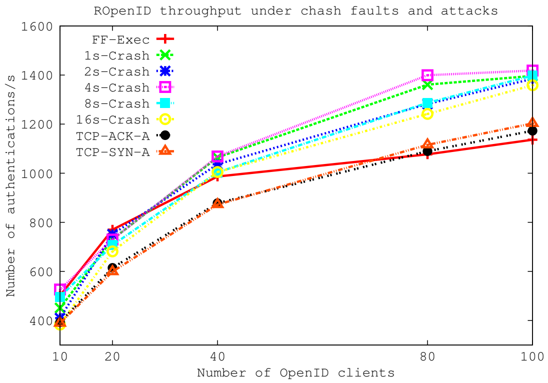

Figure 9. The effect of faults and attacks on the ROpenID

DoS attack on up to f_R replicas. With a DoS attack on up to f_R replicas, we experienced a slightly different throughput behavior on the system. 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. However, the TCP SYN and ACK attacks also affected the network performance. Therefore, a lower performance can be observed up to 80 clients. After that, the system throughput starts to increase. This can be explained by the behavior of the network. With more clients, more concurrent channels are opened with the replicas, taking advantage of the fairness behavior of networks. Another thing to observe is the fact that DoS attacks affect more the system than crash faults. This is an expected behavior since a faulty replicas (crashed) is temporarily disconnected from the system, while on the DoS attacks all replicas are still running. In spite of the fact that the replica under attack is slowing down its performance, it is still part of the system, i.e., all clients and replicas still sending and receiving messages to/from it.

VI. RELATED WORK

The limitations of BFT protocols related with the confidentiality of the stored service in case of adversary-controlled faulty replicas were discussed in Section II. Despite these limitations, several survivable security-related services have been designed by exploiting the capabilities of Byzantine fault tolerance. COCA is an intrusion-tolerant certificate authority that makes use of Byzantine quorum based protocols and threshold signatures for PKI provision despite intrusions in less than a third of its components [26]. CODEX is secret store system based on the same principles of COCA [27]. Both these systems support a limited set of operations that can be satisfied with Byzantine quorum protocols, which are simpler to implement than state machine replication [15]. DepSpace, by the other hand, integrate an SMR protocol with secret sharing for storing tuples with confidentiality guarantees [28]. A key limitation of DepSpace is that confidential tuples can only be stored and retrieved (similarly to CODEX), while in our architecture, we need to compute certain functions with the confidential data (keys) to be compatible with current authentication protocols. This explains our need for the trusted component on the service replicas, which can safely use the confidential data for executing such limited functionality.

To the best of our knowledge, fault- and intrusion-tolerant AAIs have not being deeply investigated yet. We could find only one similar work, which proposes intrusion tolerant OpenID providers [29], called here as OpenID-VR. It is based on virtual machines running on top of a single physical server using a shared memory provided by the hypervisor to communicate among them. The assumption is that shared memory can significantly reduce the overhead imposed by message passing protocols. The OpenID-VR considers also the agreement service, required for SMR, as a secure component because it is implemented at the hypervisor level, i.e., the hypervisor is assumed to be a trusted computing base.

Beyond that, the OpenID-VR has two more problems. First, the scalability of the system is limited to the single authentication service, based on secure processors (e.g., smart cards), for authenticating users. Thus, the authentication server (a grid of smart cards) can pose a significant overhead on the system performance, which can go over 2s of latency [30], significantly impacting the system throughput. Second, it only tolerates faults and intrusions regarding the OpenID protocol. Put another way, it is not capable of tolerating physical failures (e.g., energy blackouts, defects in hard drivers, connectivity losses, and so forth), logical failures (e.g., software failures, network configuration and/or connectivity problems, and so forth) and it also does not tolerate resource exhaustion attacks, such as those reported in [5]. Therefore, the hypervisor, the agreement service and the authentication server are single points of failure of the system architecture.

Differently from OpenID-VR, our resilient OpenID can use from 1 up to $3f_R+1$ secure elements. More interestingly, the latency of our prototype is similar (considering a similar setup, i.e., multiple virtual machines over a single hypervisor) to the OpenID-VR, despite the fact that our solution uses message passing communications instead of shared memory. Both systems have a authentication latency of 7ms.

VII. CONCLUSION

We have described an architecture for developing and deploying more resilient and secure AAIs. Among the key components and techniques are protocols to tolerate arbitrary faults, diversity of operating systems and hypervisors, secure components to ensure the confidentiality of sensitive data and operations, and a new element called gateway to keep backward compatibility with existing AAIs.

We discuss the design, implementation and evaluation of two prototypes as proof of concept (RADIUS and OpenID), following their respective protocol specifications. One of the challenges in implementing these services was regarding the determinism required by state machine replication protocols, which leads us to design an adapted version of the TLS PRF function suitable for replicated systems.

The results show how both prototypes can tolerate different kinds of faults and attacks, such as crashes, arbitrary behavior of some components and DoS attacks. Furthermore, the throughput and latency results indicate that our prototype implementations are able to meet the demand of real environments with hundreds of thousands users.

ACKNOWLEDGMENT

This work was partially supported by the EC through projects projects SecFuNet (FP7-ICT-STREP-288349) and SEGRID (FP7- 607109), by FCT through the LaSIGE StrategicProject (PEst-OE/EEI/UI0408/2014), and also by CNPq/Brazil, through grants 590047/2011-6 and 202104/2012-5.

REFERENCES

[1] C. Tankard, "Advanced Persistent threats and how to monitor and deter them," *Network Security*, 2011.

[2] "All about Stuxnet," 2013, http://stuxnet.net.

[3] M. Prince, "The DDoS that almost broke the internet," 2013, goo.gl/oeDrMY.

[4] M. Prince, "Ceasefires don't end cyberwars," 2012, goo.gl/Vkljbi.

[5] D. Kreutz, H. Niedermayer, E. Feitosa, J. da Silva Fraga, and O. Malichevskyy, "Architecture components for resilient networks," SecFuNet, Tech. Rep., 2013.

[6] 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," *IEEE Security & Privacy*, 2006.

[7] A. N. Bessani, "From byzantine fault tolerance to intrusion tolerance (a position paper)," in *IEEE/IFIP DSN-W*, 2011.

[8] GEANT & TERENA, "eduroam," 2012, goo.gl/hBh6l3.

[9] MySQL & Network RADIUS, "Deploying FreeRADIUS with the MySQL Cluster Database," 2009, goo.gl/GXuZHH.

[10] Gluu, "High availability clusters," 2014, goo.gl/oBZOEN.

[11] U. Hoelzle and L. A. Barroso, *The Datacenter As a Computer: An Introduction to the Design of Warehouse-Scale Machines*. Morgan and Claypool, 2009.

[12] D. Kreutz, F. M. Ramos, and P. Verissimo, "Towards secure and dependable software-defined networks," in *SIGCOMM HotSDN*, 2013.

[13] D. Kreutz, F. M. V. Ramos, P. Verissimo, C. Esteve Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-Defined Networking: A Comprehensive Survey," *ArXiv e-prints*, 2014, goo.gl/73x1ZV.

[14] F. Schneider, "The state machine approach: A tutorial," in *Fault-Tolerant Distributed Computing*. Springer, 1990.

[15] M. Castro and B. Liskov, "Practical Byzantine fault-tolerance and proactive recovery," *ACM Trans. on Comp. Sys.*, 2002.

[16] A. Bessani, J. Sousa, and E. Alchieri, "State machine replication for the masses with BFT-SMaRt," in *IEEE DSN*, 2014.

[17] Verizon RISK Team, "Data breach investigations report," Verizon, Tech. Rep., 2013, goo.gl/7mIBy.

[18] 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.

[19] J. Vacca, *Public Key Infrastructure: Building Trusted Applications and Web Services*. CRC Press, 2007.

[20] D. Dolev and A. C. Yao, "On the security of public key protocols," *Information Theory, IEEE Transactions on*, 1983.

[21] "BFT-SMaRt - High-performance Byzantine-Fault-Tolerant State Machine Replication," 2014, goo.gl/FU26dS.

[22] A. Shamir, "How to share a secret," *Commun. ACM*, 1979.

[23] Intel Corp., "Software guard extensions programming reference," 2013, goo.gl/tZpDxa.

[24] "The Legion of the Bouncy Castle," 2014, goo.gl/1NEPzX.

[25] Amazon Web Services, Inc., "Amazon EC2 Pricing," 2014, goo.gl/rJNrA6.

[26] L. Zhou, F. Schneider, and R. Van Rennesse, "COCA: A secure distributed online certification authority," *ACM Trans. on Computer Systems*, 2002.

[27] M. A. Marsh and F. B. Schneider, "CODEX: A robust and secure secret distribution system," *IEEE Trans. on Dependable and Secure Computing*, 2004.

[28] A. Bessani, E. Alchieri, M. Correia, and J. Fraga, "DepSpace: a Byzantine fault-tolerant coordination service," in *ACM EuroSys'08*, 2008.

[29] L. Barreto, F. Siqueira, J. Fraga, and E. Feitosa, "An intrusion tolerant identity management infrastructure for cloud computing services," in *IEE ICWS*, 2013.

[30] P. Urien and M. Dandjinou, "Introducing smartcard enabled radius server," in *Int. Symposium on CTS*, May 2006.