UNIVERSIDADE DE LISBOA
Faculdade de Ciências
Departamento de Informática

RESILIENT AUTHENTICATION SERVICE

Oleksandr Malichevskyy

DISSERTAÇÃO

MESTRADO EM ENGENHARIA INFORMÁTICA
Especialização em Sistemas de Informação

2013

UNIVERSIDADE DE LISBOA
Faculdade de Ciências
Departamento de Informática

RESILIENT AUTHENTICATION SERVICE

Oleksandr Malichevskyy

DISSERTAÇÃO

Projecto orientado pelo Prof. Doutor Alysson Neves Bessani
e co-orientado pelo Prof. Doutor Marcelo Pasin

MESTRADO EM ENGENHARIA INFORMÁTICA
Especialização em Sistemas de Informação

2013

Acknowledgments

Initially, I would like to thank my advisers, Professors Marcelo Pasin and Alysson Bessani, for all lessons given to me day after day.

I also thanks to my colleagues in the Navigators research group and the LaSIGE, and to all my professors from master and undergraduate courses by all the teaching on computer science and regarding life. Special thanks to Diego Kreutz that during this work was always presents to help and advice. So I can say he was my third supervisor.

Last but not least, I would like thank all my family by their unconditional support, patience and encouragement even I being physically away most of the time.

Finally, I would like to thank João Sousa for help in use of BFT-SMaRt library and Typhon.

This work is partially supported by the European Commission through the project SecFuNet (STREP 288349, FP7-ICT-2011-EU-Brazil) and FCT through multiyear programs (LaSIGE) and CMUPORTUGAL and CloudFIT (PTDC/EIACCO/108299/2008).

Dedico este trabalho aos Navigators.

Resumo

A grande maioria dos sistemas *on-line* dependem dos serviços básicos de autenticação e autorização. Estes são responsáveis por prover os recursos necessários para evitar acessos indevidos ou não autorizados a aplicações, dados ou redes. Para aceder aos recursos por norma os utilizadores usam um nome de utilizador e uma prova, que geralmente é uma senha, ou seja, uma informação teoricamente conhecida unicamente pelo respectivo utilizador.

Nos últimos anos o uso de redes sem fios sofreu um grande aumento. A maioria destes serviços necessitam de algum tipo de autenticação e mecanismos de autorização para dar ou não o acesso ao serviço e verificar os direitos e permissões dos utilizadores. Para isso o utilizador tem de se autenticar perante o serviço.

É comum os utilizadores terem um par de nome de utilizador/palavra chave para cada serviço que usam. Isso traz problemas de gestão tanto para os utilizadores, que têm de memorizar as suas credenciais, como para os administradores, que têm de gerir um grande número de utilizadores. O utilizador não só tem de memorizar as credenciais para os serviços que usa como também passa a ter varias identidades, uma vez que identidades não são partilhadas entre serviços. Para resolver o problema de múltiplas identidades apareceu o modelo de identidade federada. As contas de utilizadores são agregados num único provedor de identidade, a escolha de cada utilizador. Assim os utilizadores têm os seus dados num só local em que eles confiam e só tem de memorizar uma credencial. Isso permite usar as suas credenciais para acesso a vários serviços. Como exemplo podemos dar a rede sem fios *eduroam*.

Esta dissertação vai se focar nos serviços de autenticação para redes sem fios com grande numero de utilizadores. Com as identidades federadas os utilizadores podem aceder aos serviços de rede usando as suas credenciais a partir de qualquer local. No caso de serviço *eduroam*, um utilizador tendo credencias de uma instituição de ensino pode deslocar-se a outra instituição da federação em qualquer parte do mundo e ter acesso a rede usando as credenciais da sua instituição. Para isso os pontos de acesso usam um servidor de autenticação AAA (authenticação, autorizaçao e responsabilidade), que no caso de *eduroam* é RADIUS.

AAA é uma arquitectura que permite uso de protocolos de authenticação dos utilizadores nas redes de grande porte e é baseada em três componentes base, suplicante, NAS

(e.g., router Wi-Fi) e o servidor de autenticação. Quando suplicante quer aceder a rede, ele manda as suas credenciais ao NAS e este usa o servidor de autenticação para valida-las.

A longo da existencia de arquitectura AAA foi dado mais enfase a segurança dos protocolos de autenticação do que a resiliencia das componentes, tais como o NAS e o servidor de autenticação. No caso de falha do NAS o suplicante pode escolher outro e voltar tentar autenticar. Se o servidor de autenticação falhar, sofrer um ataque ou mesmo uma intrusão o atacante consegue negar acesso a rede aos utilizadores legitimos, ou roubar as credenciais dos mesmos e fazer um ataque a rede. No caso de uma federação, em que os utilizadores usam uma credencial para aceder a vários serviços, esse problema torna-se ainda mais grave, visto que o atacante consegue atacar não só um servidor de autenticação como toda a federação e os serviços prestados na rede da mesma.

O grande objectivo desta dissertação é desenvolver um servidor de autenticação para redes sem fios resiliente, tolerante a faltas e as intrusões. Para cumprir estes objectivos foi escolhido o protocolo RADIUS devido a seu alargado uso (e.g., *eduroam*, provedores de Internet) e a sua simplicidade. As garantias de tolerância a faltas e a intrusões foram conseguidas através do uso de replicação activa, com máquinas de estados em conjunto com uma componente segura. A replicação de um serviço, por norma, obriga a um mudança de cliente, neste caso seria o NAS, de modo a suportar a replicação. Duarante o desenho de arquitectura teve-se o cudado de evitar a mudança nas componetes mais proximas do supplicante, de modo a possibilitar a integração de novo serviço resiliente nas redes actuais.

O protocolo RADIUS suporta, na sua definição base, mecanismos de autenticação fracos baseados em nome de utilizador/password, porque foi projectado para redes com fios. Em redes sem fios, geralmente é mais fácil escutar a comunicação e, assim, roubar credenciais dos utilizadores. A solução para este problema foi a adição de suporte de métodos de autenticação EAP (Extensible Authentication Protocol). Com a utilização de EAP, podemos adicionar métodos de autenticação fortes a fim de conseguir as propriedades de segurança durante a autenticação. A principal razão para usar EAP é eliminar a necessidade de mudar as componentes intermédios da rede, tais como NAS. Precisamos mudar apenas o suplicante e o servidor de autenticação. Os pacotes EAP são transportados através dos componentes de rede do suplicante para o servidor de autenticação através de, por exemplo, o protocolo 802.1X entre suplicante e NAS e RADIUS entre NAS e servidor de autenticação.

O método de autenticação EAP escolhido foi EAP-TLS visto que é um padrão aberto e um dos mais robustos protocolos de autenticação. Permite uma autenticação fim-a-fim e a geração de chaves simétricas entre o suplicante e o servidor de autenticação de forma secreta. Apesar de ser um sistema de autenticação forte existe uma dificuldade em distribuição de credenciais. Ao contrario das credenciais baseadas em nome de utilizador/palavra chave, este metodo necessita de geração de um certificado para cada servidor

de autenticação e para cada utilizador do sistema.

O sistema desenhado e desenvolvido é composto por quatro componentes: suplicante (pede acesso a rede), NAS (no nosso caso é um *router* de rede sem fios), *gateway* (elimina a necessidade de alterarmos os clientes RADIUS existentes e funciona como cliente do nosso servidor de autenticação replicado) e servidor de autenticação RADIUS replicado (um serviço replicado tolerante a faltas bizantina e a intrusões). Para implementação do servidor de autenticação replicado e do seu cliente (*gateway*) foi usada biblioteca de replicação *BFT-SMaRt*. Cada servidor de autenticação tem a sua componete segura, que providencia a tolerancia a intrusão escondendo os dados senciveis do servidor, tais como seu certificado e chaves partilhadas com o NAS. Se o servidor necessitar de usar esses dados a componente segura providencia um interface que permite o servidor executar todas as operações necessarias que envolvem esses dados.

Para validar o desempenho do sistema foram feitos vários testes de latência e de debito comparando o prototipo concretizado a uma implementação bastante popular de *FreeRADIUS*. Notaram-se algumas diferenças em termos de desempenho de serviço de RADIUS replicado em relação ao *FreeRADIUS*. Os testes mostraram que o RADIUS replicado tem uma latência superior e o débito inferior ou de *FreeRADIUS*. Isso deve-se, em especial, pelo facto do primeiro ser um sistema replicado e necessitar uma maior troca de mensagens devido aos protocolos BFT e replicação de máquina de estados. Apesar do RADIUS replicado ser um sistema replicado, consegue mostrar uma latência razoável e aceitável em ambientes de redes locais.

Palavras-chave: tolerancia a faltas distribuidas, resiliência, serviços de autenticação, RADIUS, EAP-TLS.

Abstract

The increasing use of the wireless networks in the last years has created the demand for authentication and authorization for these networks. The basic model usually requires a user, to access the network, authenticate itself before the authentication server using its credentials. Authentication and authorization in networks with the large number of users is usually achieved using the WPA-Enterprise mode. WPA-Enterprise allows the use of the external authentication server to validate user credentials and determinate his rights.

Most common and widely used protocol for WPA-Enterprise is RADIUS, which follows AAA architecture. Normally RADIUS servers are running in a single machine and in a single process. If RADIUS server stops users are unable to authenticate and access the network. To solve this problem, most RADIUS servers are replicated for redundancy and load management. AAA architecture and RADIUS protocol fail completely in case of server Byzantine behavior, i.e., if a failure makes the system present arbitrary behavior. In case of intrusion on authentication server, the attacker is able to access user credentials and other sensible data, such as server certificates.

The major focus of this work is to develop a resilient, fault- and intrusion-tolerant authentication server for WPA-Enterprise wireless networks, without changing existent systems. To meet these objectives we implemented a replicated RADIUS-compliant protocol, which uses EAP-TLS as its authentication method. Fault and intrusion tolerance is ensured using state machine replication, together with a tamper-proof component used for storing cryptographic keys related with user credentials.

The service was evaluated and compared with a popular non-fault-tolerant solution, which is used in the eduroam network, FreeRADIUS. Initial results demonstrate the applicability of the proposed solution.

Keywords: Byzantine fault tolerance, intrusion tolerance, authentication, RADIUS, EAP-TLS.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this chapter we present the motivation for this thesis and most significant achievements. In the end of the chapter we describe contribution of this work and also present structure of the document.

1.1 Motivation

The last years have seen a significant increase in the use of wireless networks, not only in home environment, but also in business and educational infrastructures. For private usage we use a secret key to have access to the wireless network. This key not only protects access to the network, but also is used as a key to protect communication between user and wireless access point, because we use open air space. In business and education environments, with large number of users, we have a need to generate a secret key to every user that wish to access the network. If we use same shared key for every user it is possible to intersect the user communication by the malicious agent.

WPA-Enterprise, normally, requires some authentication server that will authenticate and authorise user access to the network. Authentication server can control network access in institutions with large number of access points and shared resources, like printers, internal sites, etc. With this each user must have a unique identity (e.g., username) and his identity prove (e.g., password). Before access the network user needs to authenticate himself before the authentication server. After authentication server validates user credentials, and authorise access to the network, it generates a unique secret key that will be used to protect user communication.

Among the last years it was given more emphasis to security of the protocols and authentication methods than the resilience of the components. Current protocols are only able to verify packet correctness and component availability, such as the access point and the authentication server. In case of Byzantine behaviour or an intrusion architecture components are unable to detect it.

Trust and resilience of the authentication servers is a critical issue in enterprise envi-

ronment. In case of federated identity, a single local attack to an authentication service could lead to the large-scale problems. First of all, the users could be denied to access any service of the federation. Second, the attacker could make bad use of all user's identities in the compromised domain. An example could be the attacker using user's credentials to access services or attack other service and identity providers in the federation.

In order to address this problems, we replicated the architecture components for tolerate Byzantine behaviour. An intrusion tolerance is achieved through trusted component and strong end-to-end authentication protocol. Additionally, to provide intrusion tolerance, a tamper-proof component is used, to store security-sensitive data. Trusted component provides interface that allows authentication server performs needed operations that involve stored data.

This thesis proposes the design of a new fault- and intrusion-tolerant architecture for WPA-Enterprise wireless networks, compatible with existent network architecture components. Compatibility with the current architecture is achieved through the introduction of a new component called "*Untrusted Gateway*". This new component is stateless and hides authentication server replication, passing packets from access point to replicated authentication server and back.

1.2 Objectives

The main objective of this work is to develop a resilient AAA server. In this work, a RADIUS service is replicated in order to render it fault- and intrusion-tolerant.

The specific objectives of this work are:

- Design a resilient architecture for WPA-Enterprise like environments;
- Design a resilient, fault- and intrusion-tolerant AAA server;
- Design a trusted component that can be used to store cryptographic material. This trusted component provide an interface with all necessary cryptographic methods without expose sensitive cryptographic material;
- Provide end-to-end security (EAP-TLS => modify EAP-TLS process to support replication), using server and client certificates;
- Avoid changes on existing environment. In general, clients of replicated services must be modified in order to communicate with several replicas. We use smart combination of components in order to keep RADIUS client as they are today.

1.3 Contributions

- Design and implementation of a fault- and intrusion-tolerant authentication service based on RADIUS protocol with EAP-TLS, which can be used in normal environments without requiring modifications on existing systems.
- Comparison of a prototype with a popular RADIUS implementation authentication service called FreeRADIUS.
- A preliminary version of the work in this thesis was published in the paper *"O vigia dos vigias: um serviço* RADIUS *resiliênte"*, that appeared on INFORUM' 12.
- Also, developed prototype going to be used as demonstration in SecFuNet - Security for Future Networks project `http://www.secfunet.eu/` and as a use case to resilient authentication service.

1.4 Structure of the document

This document is organized as follows:

- Chapter 2 - Background, describes existent authentication schemas, AAA architecture, AAA authentication protocols and most relevant related work. This chapter also discusses existent problems with current architectures and provides some theoretical background about techniques that can be used to overcome problems.
- Chapter 3 - System design, is described designed solution and how it can be integrated with existent AAA systems without changing them.
- Chapter 4 - Implementation, presents how the prototype was implemented. In this chapter is described also how each component was implemented.
- Chapter 5 - Evaluation, in which experimental results are presented. The tests and the evaluations that made to compare the prototype's performance with existing implementation of FreeRADIUS [1]. Also simulated attacks were performed in order to verify system properties and functionalities.
- Chapter 6 - Conclusion and future work, summarize the achievements of this work and suggests possibilities of future development.

[1] http://freeradius.org/

Chapter 2

Background

This chapter presents the theoretical background, protocols, most common threats and attacks to existent AAA architecture. In the end we also present the most significant related work.

2.1 Authentication Schemes

Authentication is the action of uniquely identifying a user to grant him (or her) access to the different types of services. The authentication process is started with the user providing a unique identity. Usually, this step is accomplished with the user typing a unique name and an access credential such as password, which is known only by the user. There are many different types of credentials, which can include things like certificates, one-time-passwords, fingerprints, smart cards and tokens.

Each authentication scheme has its own purposes, pros and cons, depending on the deployment environment. Existing authentication schemes can be divided in four architectures: *conventional*, *centralized*, *federated* and *user-centered*.

In *conventional* authentication scheme, a user creates his digital identity for each service provider with which user interacts. The user's unique identifier, also known as ID, is not shared among the applications, services and identity providers. Even in a single domain, you can have different user information bases. Thus, this approach tends to be discontinued by the service providers since it is not practical both for providers and users.

Centralized authentication schemes are composed of two basic components, the identity provider (IdP) and the service providers (SPs). All user IDs are kept within the centralized IdP and are used by SPs. In centralized authentication schemas all user credentials, during the authentication session, can be used all the services. Service providers have a relationship of trust with the identity providers. This architecture was created to solve the conventional authentication problems of having to manage multiple users. However, identity providers are still the single points of failure, and have total control and access to the user's private information.

The idea behind the *federated* authentication scheme is to distribute the identification and authentication process among multiple identity providers. In this sense, it extends the centralized authentication model. The federated schema is based on a trusted relationship among all identity and service providers. Each identity provider and service provider may belong to a different administrative domain, and users are allowed to choose their identity provider. Once registered within one identity provider, user is able to access any service in the federation.

A *user-centered* scheme is based on the autonomy of the end-user to control their personal identities and personal information. The essential debate about the loss of privacy revolves around the trust. The personal information should not be revealed to anyone, but only for those who need it for official business and for legal purposes [8]. The fact that the information is protected and available only to authorized people for legal purpose, is the essence of privacy.

2.2 Authentication, Authorization and Accounting

Authentication, authorization and accounting (AAA) [3, 49] is a set of network security services that provide a primary architectural framework, through which we can configure the access control to a router or to an access server. These protocols are defined considering three key elements, supplicant (client wanting to access the network), NAS [33, 44] (e.g., wireless router) and AAA server as presented on Figure 2.1.

Figure 2.1: Authentication, authorization and accounting infrastructures.

Authentication is a process in which the user identity is verified. This is typically done by providing evidence that the user holds a specific entity such as identifiers and credentials. Authentication is the way a user is identified prior to being allowed access to the network and its services.

Authorization is used to determine what actions an entity can perform after a successful authentication process. Authorization works by assigning a set of permissions that describe what the entity can perform.

Accounting provides methods for collecting data about the network or service usage. The collected data can be used for billing, reporting, traffic accounting, among other things. Additionally, accounting mechanisms can be used to register authentication and

authorization failures, including auditing features, for checking the correctness of procedures performed based on accounting data.

2.2.1 The AAA architecture

To better understand the AAA architecture, we will use, as an example, a user trying to access a wireless network using a Wi-Fi router as a NAS.

1. The supplicant associates with Wi-Fi router and try to access the network;
2. NAS have two logical entrances, one that is pass-through to the AAA server with no access control and other to the network with controlled access, for authenticated and authorized supplicants. NAS request supplicant for it credentials, and sends them to the AAA server;
3. If the AAA server accepts supplicant credentials and authorize him to access the network, the NAS will allow the supplicant to access the network. After the AAA server accepts user credentials, it generates a unique secret key that is used to ensure security of communication between the supplicant and the NAS. This secret key will be used to cipher all communication with NAS in the case of the Wi-Fi communication;
4. Then access is accounted.

As we can see from this example all the authentication part of the access control to wireless network is made in the AAA server. If the AAA server suffers an intrusion, an attacker can steal server cryptographic material and users data. Also, the attacker can create new user(s) or give higher permissions to some user that it controls.

Federated identity based on AAAs

Federation identity management is simplified with the use of the AAA architecture; because the most used AAA protocols have a proxy feature. Proxy feature allows AAA server to pass request to the other AAA server. If the local server does not know user, it can send users credentials to the server where the user has the account.

Let's look for a simple example of authentication using federated identity in an AAA architecture. When a supplicant wants to access the network it must first authenticate before the AAA server using his credentials with its identity, that can be something like *username@domain*. When local server looks for received supplicant credentials and in case of user identity is not from local server it sends a request to the respective *domain* server. After the server responsible for the user identity makes a decision about presented credentials he responds to the local server. Depending on the server decision, the local server allows or not the supplicant to access the network.

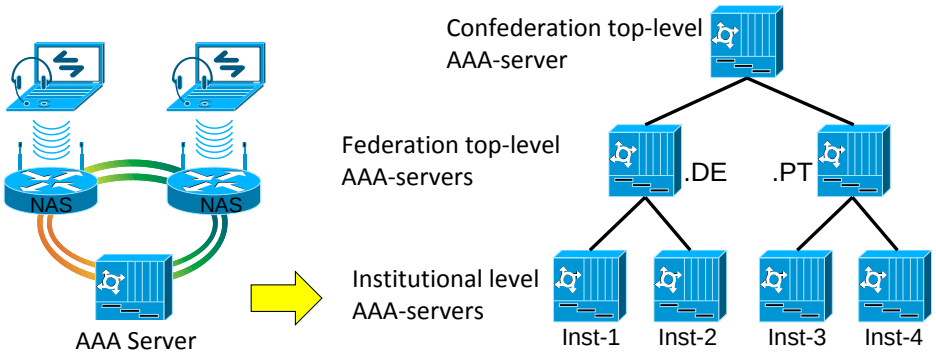

Figure 2.2: Federation identity management.

In a federated schema the authentication servers must have a trust relationship between them. This can be achieved using an hierarchical AAA architecture such as used in *eduroam*[1], as shown on Figure 2.2. In this architecture, if *Inst-1* wish to authenticate a user that have his identity in *Inst-4* it sends a request to his superior, *.DE* server and so on until reach the top-level server. The *.DE* server knows how to reach the *.PT* server, and the *.PT* server knows how to reach *Inst-4* server. When the *Inst-4* server decides about supplicant credentials, it sends a response back to the *Inst-1* using the inverse path.

Eduroam is primarily intended to provide the European academic community service mobility between university campuses. This project is working on Europe, America, Canada and Asia. Eduroam initiative aims to bring together all the different networks of higher education institutions, thus creating a global academic network. The interest in extending the eduroam to other communities outside the European borders has been increasingly evident. As a result of this growing interest eduroam was deployed in Australia, Japan and Latin America.

Access to the *eduroam* network requires a user to obtain a credentials on his academic institution. When users have his credentials (certificate or username/password) he can access *eduroam* any access point of the network all over the world.

2.2.2 AAA protocols

This section describes relevant and widely used AAA protocols. Some protocols are described in further detail because they are used in the development of experimental prototype.

[1]Education Roaming - `http://eduroam.org/`

RADIUS

The Remote Authentication Dial-In User Service, also known as RADIUS [53] is AAA protocol used to provide authentication in a different networks, such as dial-up, VPNs (Virtual Private Network) and wireless networks.

RADIUS uses a client-server model, where the client is the NAS. It receives connection requests from its own clients, called supplicants, passing along the corresponding AAA information to the RADIUS server. The RADIUS server is responsible for receiving connection requests, authenticating the user and send necessary information to NAS so it can grant access to the network to the user. In addition, a RADIUS server can also function as a proxy to other RADIUS servers.

In a network that uses RADIUS a supplicant that wishes to access it need to authenticate towards the authentication server. In case a RADIUS server can authenticate the user, it determines if the authenticated user is authorized to access the requested network, and his access is accounted. Transactions between the NAS and the RADIUS server are authenticated through the use of a shared secret, which is never sent over the network. The shared secret is used to provide integrity and authenticity of packets. Packet integrity can also be granted by a user-password which is ciphered using shared secret, to eliminate eavesdropping. When user password is present in the RADIUS packet, it is hidden using a method based on the RSA Message Digest Algorithm MD5 [54]. As request authenticator is random number, password is used to verify integrity of a packet.

The RADIUS server can support a variety of authentication methods. When using authentication based on username/password, RADIUS most basic configuration, it can deal with password authentication protocol (PAP) [32], challenge protocol (CHAP), and UNIX login. As RADIUS is used not only in wired networks, but also in wireless networks, we cannot use username/password authentication, because PAP sends user credentials in clear text and CHAP is a weak authentication protocol [13]. RADIUS also allows the use of protocol security extensions such as Extensible Authentication Protocol (EAP), which will be described later.

RADIUS can use different back-ends, examples are SQL database, Kerberos, LDAP, Active Directory or even other AAA server to store a user credentials. When a RADIUS server receives user credentials it send credentials to some back-end to validate them. RADIUS back-ends also can be used for authorization and to store accounting data.

RADIUS protocol make, for authentication and authorization, uses 4 types of packets `Access-Request`, `Access-Challenge`, `Access-Reject` and `Access-Accept` to provide authentication as presented on Figure 2.3.

`Access-Request` is used by the NAS to send to the RADIUS server supplicant credentials. If legacy authentication protocol such as PAP (Password Authentication Protocol) is used only `Access-Request` and `Access-Reject` or `Access-Accept` are needed.

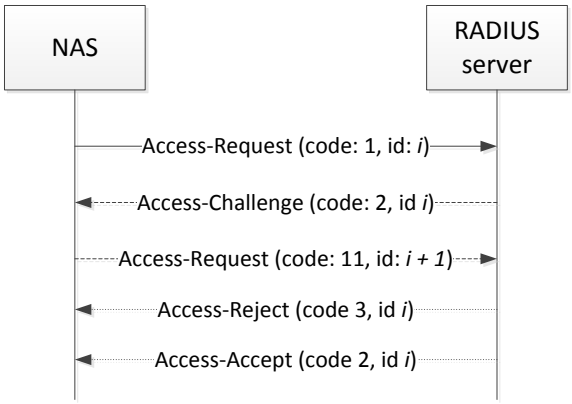

Figure 2.3: RADIUS authentication.

Figure 2.3 shows as that `Access-Reject` or `Access-Accept` packet identifier (*id*) must be the same, so NAS could make a much between request and response. When supplicant uses complex authentication methods like CHAP or EAP is used one more packet from RADIUS protocol called `Access-Challenge`. When this packet is sent `Identifier` field must match request `Identifier` field. This packet is used to challenge supplicant in case of CHAP or as a response packet from RADIUS server to NAS in case of EAP. Challenge is resent to supplicant in order to supplicant answer it. When NAS receives a response from supplicant creates new `Access-Request` message with supplicant answer using different `Identifier`.

In case of successful user authentication and authorization the `Access-Accept` packet is sent with list of configuration values for the NAS. These two messages, `Access-Accept` and `Access-Reject`, can transport some user-friendly message to supplicant, for example hello message or to supplicant understand why his authentication failed. When NAS receives `Access-Accept` it receives in the packet a key that will be used during communication between the supplicant and the NAS.

Packet

Figure 2.4 represents an overview of a RADIUS packet. It consists in five main fields: `Code`, `Identifier`, `Length`, `Authenticator` and `Attributes`. Each of them has a specific rule in the RADIUS protocol.

Code Size: one octet. Represents type of packet. Existing codes: 1 - Access-Request, 2 - Access-Accept, 3 - Access-Reject, 4 - Accounting-Request, 5 - Accounting-Response and 11 - Access-Challenge.

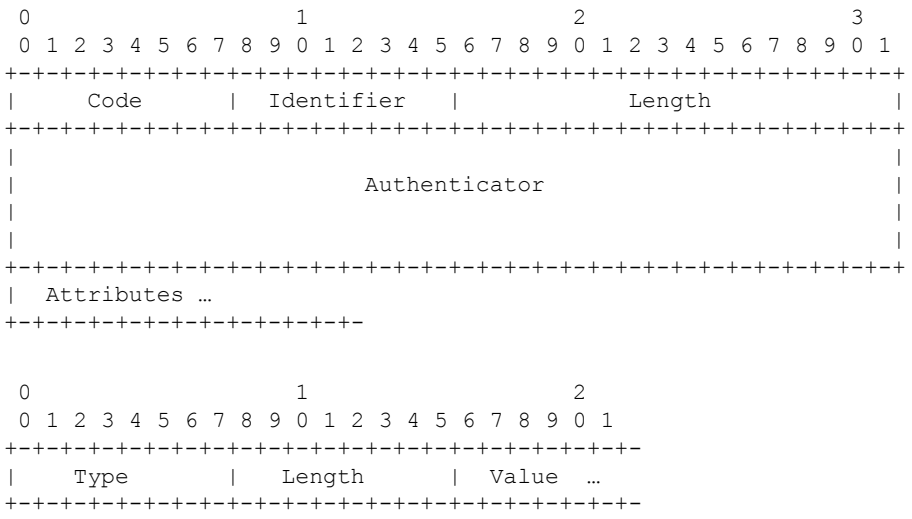

Figure 2.4: RADIUS packet.

Identifier Size: one octet. Uniquely identifies a packet. When NAS sends a request to RADIUS server it responds using the same `Identifier`. `Identifier` is different for each request/response message pair. If some NAS sends a packet with the same identifier, from same NAS IP and same NAS port in short amount of time this packet is discarded. This happens to avoid replay attack.

Length Size: two octets. Represent length of whole packet. As packet length have only two octets, packet max length is limited, in some cases, for example when EAP is used, packet fields must be fragmented to transport all needed data.

Authenticator Size: 16 octets. Is used to ensure integrity and authenticity of a packet, and user-password hiding, if present. In case of a client request, this authenticator is a random number called *Request Authenticator*. This value must be unique to each `identifier`. All response packets coming with *Response Authenticator*, which are generated using *Request Authenticator*, all packet data and shared secret between NAS and authentication server. That is, $ResponseAuth = HMACMD5(Code+Identifier+Length + RequestAuth + Attributes + Secret)$ where $+$ denotes concatenation.

As in the request packet `Authenticator` is a random number we can use `Message-Authenticator` attribute, which is similar to `Authenticator` field, but provide packet integrity and authenticity to request and response packets.

Attribute Is a field variable in length. It contains the list of attributes that are requires by the service. Each attribute is composed by `Type`, `Length` and `Value` fields.

- `Type` is one octet field that represents the type of an attribute;
- `Length` is one octet field that represents length of entire attribute, including `Type`, `Length` and `Value`;
- `Value` is variable in length and contains attribute data.

Some of attribute types are used by RADIUS service, but we can create a new one if it necessary. New RADIUS attribute must be recognized by NAS and RADIUS server. If type is unknown, RADIUS server or NAS may ignore it, or ignore a whole packet. End of attributes is defined by RADIUS packet `Length` field.

Diameter

After IETF RADIUS working group decided that it had concluded its work, a new IETF group called AAA working group started its work on finding the successor to RADIUS, which would become Diameter [1].

Diameter is a protocol based on RADIUS, which tries to fix the deficiencies present in its predecessor. This new protocol proposes some changes in RADIUS, and tries to absorb their best features to meet the recent demands for more efficient and resilient AAA services. Among the new features, we can cite TCP usage. Hop-to-hop and end-to-end packet protection, RADIUS has only hop-to-hop packet protection. Field size has incised, compared to RADIUS packet, so we can transport more data without fragment it. Diameter protects entire packet, leaving only a header unencrypted for hop-to-hop communication instead use a authenticator as RADIUS.

Diameter name comes from a play of its predecessor RADIUS, because diameter is twice the radius. The new protocol is not fully compatible with RADIUS, but has many similarities with regard to its functionality. Diameter uses UDP, TCP and STCP and requires more messages and, on the contrary of RADIUS, it responds to every message that are sent by the NAS. Instead of using the client-server model, as happen on RADIUS, Diameter uses the peer-to-peer communication model. Use of peer-to-peer model allows as, for example, to disconnect a connected supplicant, or to divide the three AAA functionalities of the protocol in three different servers. Every Diameter component is considered an independent application.

Authentication servers based on Diameter, dealing with NAS, are considered as a separate application named NASREQ. It means that NAS specification is described in a separate document. As all nodes have their own specification we can assume that not every Diameter node will support all the Diameter applications. So before Diameter applications establish communication, they need to verify their capabilities [1, 49].

TACACS+

Terminal Access Controller Access Control System Plus (TACACS+) [20] developed by Cisco Systems, is a network access technique in which users are connected with a centralized server that contains a unique database of authorized accounts. It is an authentication protocol used to communicate with remote authentication servers, usually on UNIX-based operating systems.

After authenticating the individual using the access server, it forwards the information to the server log data requested by the individual who uses it. TACACS+ allows a remote access server to communicate with an authentication server to verify that the user has access to the network. The NAS collects the username and password and then sends a query to a TACACS+ authentication server, sometimes called a TACACS+ daemon or simply TACACSD. Based on the answer to this query, the user access and released or not.

RADIUS only encrypts the password in the request packet while response packet contains only hash to validate packet integrity and authenticity. Other packet information can be captured by the third party. TACACS+ encrypt the entire body, but leaves a standard TACACS+ header similar what happens on the Diameter protocol.

TACACS+ separate all three AAA functionalities, so we can use every functionality in separate way. For example, it's possible to use Kerberos authentication and TACACS+ authorization and accounting. If during a session additional authorization checking is needed, the NAS can check with the TACACS+ server in order to determinate if the user can perform the specific command.

AAA protocols comparison

As we can see on Table 2.1, RADIUS is a protocol that requires lowest hardware specification. Compared with others, RADIUS protocol has some security problems, because his packet protection is only granted between hops. Plus only the password attribute is ciphered. RADIUS is most used nowadays, and we can conclude it because TACACS and TACACS+ are proprietary protocols and are mostly used by Cisco. In case of Diameter it is not yet commonly used.

Diameter was developed as a RADIUS successor. TACACS+, also as Diameter, define commands instead of "message types" specified in RADIUS. To improve RADIUS protocol EAP support was added, which extends RADIUS adding support for more secure authentication methods. In conclusion, Diameter and TACACS+ are better, more secure and resilient protocols when compared with RADIUS, however, the later continue to be the most used AAA protocol.

Characteristic	RADIUS	Diameter	TACACS+
Transport protocol	UDP	UDP, TCP, STCP	TCP
Authentication and Authorization	Combined	Separated	Separated
Encryption	Passwords only	Packet payload	Packet payload
Extension support	EAP	EAP	No
Multiprotocol Support	IP	IP	IP, Apple, NetBIOS, Novell, X.25
Hardware Requirements	Low HW specification	Low HW specification. More effort than RADIUS because of TCP usage.	Similar effort compared to Diameter. Low support out of Cisco products
Usage	Most used	Low usage	Used in Cisco products.
Security	Hop-to-Hop	Hop-to-Hop, End-to-End	End-to-End

Table 2.1: AAA protocols.

2.3 EAP

The Extensible Authentication Protocol (EAP) [2, 33, 44] is an authentication framework used primarily in point-to-point connections and wireless networks. It was initially developed for networks where Internet Protocol (IP) is not available, and subsequently was applied to wired and wireless networks. The main objective of the EAP protocol is to allow two parties to combine an authentication method to use before their authentication process.

EAP supports multiple authentication methods, and provides trading functions for authentication method combination. Among these methods, we can mention the EAP-MD5 [44], EAP-TLS [32] and EAP-TTLS [44]. The last one can support multiple authentication mechanisms, such as token cards, smart cards, certificates, one-time passwords or even PAP or CHAP, through encapsulation.

2.3.1 Authentication Methods

In this section we describe some of the most used EAP authentication methods.

EAP-MD5 (Message Digest) is a challenge-based EAP authentication type that provides a basic level of EAP support. MD5 stands for Message Digest 5 [63], and was developed by RSA. EAP-MD5 uses a 128-bit generated number string, or hash, to verify

the authenticity of a peer. This method uses a three-way handshake in order to authenticate the user.

EAP-MD5 uses MD5 as the hash function to hide the secret or password required for authentication. This is the first problem of this method, since MD5 considered insecure [6]. Moreover, user's password is stored in a way that enables the authentication server to get the original plain text password. Additionally, EAP-MD5 does not support mutual authentication. Finally, unlike other authentication methods described below such as EAP-TLS, EAP-MD5 does not provide the means to derive a session key immediately after authentication phase. Session key must be generated by the authentication server and can be compromised in case of malicious authentication server.

EAP-TLS (Transport Layer Security) provides one of the most secure certificate-based mutual authentication available for RADIUS [4, 6]. TLS (based on Netscape's SSL v3.*x*) requires a public key infrastructure (PKI) to authenticate both clients and servers.

The supplicant and the authentication server have their certificates to perform mutual authentication and can dynamically generate user-based and session-based keys [44] to secure subsequent communications between the supplicant and NAS. Authentication is performed end-to-end, between the supplicant and the authentication server, and a session key is generated using the TLS sub-protocol.

The EAP-TLS authentication method is used when we need strong user identity proof, for example when the user must request certificate in person or certificate can be stored on user identity smart card. Compared to username/password authentication, EAP-TLS hides user private data such as its private key, in the supplicant.

We can separate TLS hand-shake in five logical parts: *TLS start*, *server authentication*, *client authentication*, *specification change* and *connection start*.

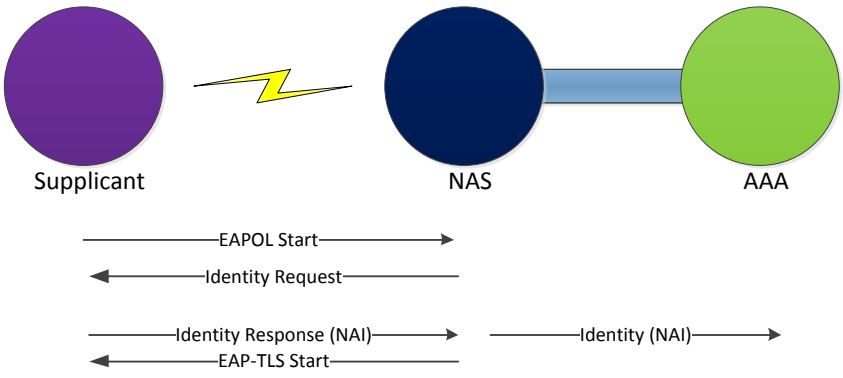

Figure 2.5: EAP-TLS Authentication (EAP-TLS Start)

Figure 2.5 presents beginning of supplicant authentication. Authentication server sends `EAP-TLS/Start` encapsulated in `Access-Challenge` packet to the NAS

containing `State` attribute that will be maintained until the end of the authentication process and will be used to uniquely identify the authentication process. `EAP-TLS/Start` packet indicates to supplicant that the EAP-TLS protocol was started.

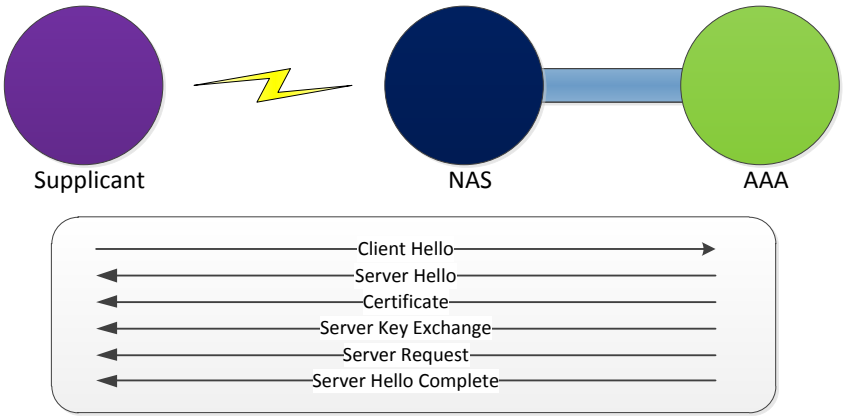

Figure 2.6: EAP-TLS Authentication (Server Authentication)

After supplicant receives confirmation that EAP-TLS method will be used, it sends the TLS `Client Hello` message described in RFC 4346. This message presented in Figure 2.6, is used to indicate beginning of the TLS connection containing:

- *client random* is used for a master secret generation;
- *suggested cipher suites* are cipher suites supported by the supplicant;
- *suggested compression methods* ordered by the client preferences, TLS 1.0 RFC define only NULL compression method as mandatory.

Messages that follow are for the server authentication, Figure 2.6. In first the server message, `Server Hello`, described in RFC 2246 [62] is sent:

- *server random*, as the client random is used to generate the master secret;
- *cipher suite* that will be used, server is mandatory. Server looks for the supplicant *suggested cipher suites* and cipher suites which server supports and choose the strongest one.

In the next message, `Certificate` presented on Figure 2.6, server sends it certificate. `Server Key Exchange` message transports values that are needed to generate the master key. In this implementation as RSA key exchange is used and this message goes empty, because all needed information for the key generation is generated by the supplicant. `Server Request` message is used to request a client certificate. And then server sends `Server-Complete` message to complete his authentication part.

After the server authentication, supplicant verifies server certificate and only then it continue with his authentication. As we use the RSA key generation, client generates premaster secret, which is 48 bytes random number, and then generates the master key.

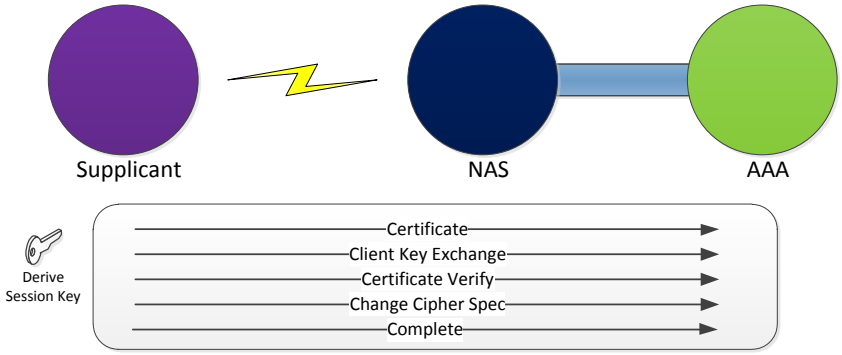

Figure 2.7: EAP-TLS Authentication (Client Authentication)

Similarly to the server authentication, client sends his certificate, as shown on Figure 2.7 first message. `Client Key Exchange` message contains the premaster secret ciphered with the server public key, so server can generate the master key. Next message indicates to the server that his certificate was successfully verified.

After client authentication it sends `Handshake Finish` message. This message ensures the integrity of the all exchanged messages and proves master key knowledge. After receive `Handshake Finish` the server sends his `Change Cipher Spec` and `Handshake Finish` message. `Handshake Finish` is defined in [39].

EAP-TTLS (Tunneled Transport Layer Security) was developed by Funk Software and Certicom, as an extension of EAP-TLS. This security method provides certificate-based authentication of client and network through an encrypted channel [4, 6] (or tunnel), as well as a means to derive dynamic, per user, per session, keys [44]. Unlike EAP-TLS, EAP-TTLS requires only server side certificates avoiding the use of PKI in the client side. That significantly reduces the overall implementation cost and eases network management and administration issues.

A client's identity is verified using a second authentication method, called inner authentication, through the symmetric encryption tunnel for the actual authentication used within the tunnel, which could be an EAP type (often MD5) or a legacy method such as PAP or CHAP. EAP-TTLS allow mutual authentication in the first phase of the authentication. In case of the mutual authentication, second phase may be skipped.

PEAP Protected Extensible Authentication Protocol [4, 16, 44] provides a method for securely transport of the authentication data, including legacy password-based protocols,

via 802.1X wireless networks. Like the competing standard TTLS (Tunneled Transport Layer Security), PEAP authenticates the wireless LAN clients using only a certificate on the server side, thus simplifying the implementation and management of a secure wireless networks. The only limitation compared with the TTLS is that the inner authentication methods must be an EAP.

LEAP (Lightweight Extensible Authentication Protocol) [22, 44], also called Cisco LEAP, is an EAP authentication method used primarily in Cisco Aironet WLANs. It encrypts data transmissions using a dynamically generated WEP[2] keys. LEAP is a two-way challenge-response protocol and, it is modified version of MS-CHAP [44] to support the mutual authentication. Security of the protocol is based on the shared key between the supplicant and the authentication server.

Probably the biggest advantage of the EAP-LEAP is the low overhead associated with the particular EAP method. The biggest disadvantage of this method is that it is the propriety protocol, so we do not know how some parts of it are implemented. For example, session key is never sent through the network, and it is computed based on a some nonce value, but this procedure is not published.

EAP-FAST (Flexible Authentication via Secure Tunneling) [22, 44] was developed by the Cisco. It is a hybrid authentication method, like TTLS and PEAP. While the TTLS and PEAP require digital certificate for server authentication, the use of the server certificate is optional in EAP-FAST. The goal of the EAP-FAST is to provide a higher level of security than was achieved by EAP-LEAP, while still maintaining the low overhead associated with the EAP-LEAP.

Mutual authentication is achieved by a PAC (Protected Access Credential) which can be managed dynamically by the authentication server. The PAC can be provided (distributed one time) to the client, either manually or automatically. The security provided by the FAST basically depends on its implementation. If it is poorly implemented, the security level provided by FAST could be comparable to LEAP or even EAP-MD5.

Authentication method comparison The Table 2.2 presents comparison between the most used EAP authentication methods.

Despite being difficult to distribute as we can see on the Table 2.2, EAP-TLS is a strongest EAP authentication method. The biggest strength of this method lies in a user and a server authentication credentials. Both have a certificate with a public and private key, that are protected by the password. The password is only known by the holders of the certificates.

[2] WEP - wired equivalent privacy

EAP Types	MD5	TLS	TTLS	PEAP	LEAP	FAST
Client-side certificate	No	Yes	No	No	No	No (PAC)
Server-side certificate	No	Yes	No	Yes	No	No (PAC)
Key management	No	Yes	Yes	Yes	Yes	Yes
Rogue AP detection	No	No	No	No	Yes	Yes
Authentication attributes	One way	Mutual	Mutual	Mutual	Mutual	Mutual
Difficulty of distribution	Easy	Hard	Moderated	Moderated	Moderated	Moderated
Security	Bad	Very High	High	High	High	High

Table 2.2: EAP authentication methods.

2.3.2 EAP over RADIUS

The EAP over RADIUS [13] is not an EAP method, but is the passing of EAP packets of any type by the NAS to the RADIUS server for authentication and back. The EAP packets are sent between the NAS and the authentication server in the format of RADIUS attribute, called EAP-Message. The NAS becomes a device to pass EAP messages between the supplicant and the RADIUS server. The EAP messages are processed in the supplicant and in the RADIUS server, not on the NAS.

The advantage of using EAP over RADIUS is that EAP authentication methods do not have to be understood by the NAS, but only by the supplicant and by the RADIUS server, which allows to add the new authentication methods without changing the existent components. However, the NAS must support EAP negotiation as an authentication protocol and pass the EAP messages to the RADIUS server and back.

In a typical use, the NAS is configured to use EAP and to use the RADIUS as the authentication provider. When we try to connect, the supplicant negotiates the use of EAP with the NAS. The RADIUS server processes the EAP message and sends a RADIUS-formatted EAP message back to the NAS, which then forwards the EAP message to the supplicant. EAP packets are transported between supplicant and NAS using EAP over 802.1X protocol.

In the RADIUS protocol EAP support is provided by RADIUS protocol shuttling EAP packet in `Access-Request` and `Access-Challenge` packets using `EAP-Message` attribute. `Access-Request` packet is used to transport data from NAS to authentication server and `Access-Challenge` is used to transport data from authentication server to NAS.

To use EAP, supplicant must negotiate it usage with NAS, Figure 2.8. After EAP

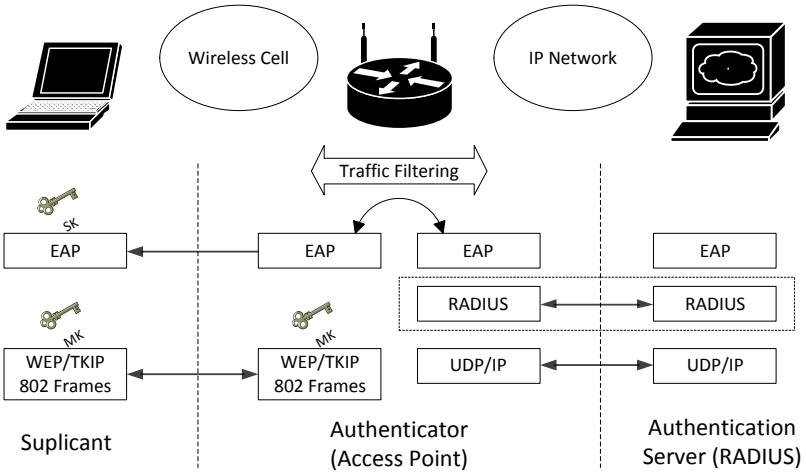

Figure 2.8: EAP Authentication

negotiation protocol continue as follows:

1. NAS send initial `EAP-Request/Identity` packet;

2. Supplicant responds with `EAP-Response/Identity` packet. This identity could be used to determinate witch EAP method to use with this supplicant and as unique identifier to accounting;

3. NAS sends `EAP-Request/Identity` packet received from supplicant to RADIUS server encapsulated in `Access-Request` RADIUS packet. After RADIUS server receives thus packet it creates `State` attribute, which is unique identifier for authentication process, and will be maintained until the end of the authentication;

4. RADIUS server can reject EAP authentication, if it don't support EAP, sending `Access-Reject`, otherwise it sends `EAP-Request` encapsulated in challenge RADIUS packet. In alternative if RADIUS server wants to change EAP authentication method it can send a request to change authentication method or simply reject supplicant access.

   In case of RADIUS server doesn't know witch EAP method to use, it choose the most basic (e.g., EAP-MD5). Supplicant or RADIUS server can change EAP method sending `Nak` packet.

5. if it's needed supplicant respond with `EAP-Response` and NAS encapsulate it in the `Access-Request` packet and sends it to RADIUS server. `Access-Request` and `Access-Challenge` are used as request/response packets until authentication is complete;

6. in case of RADIUS server accepts user credentials, it sends `EAP-Success`, otherwise `EAP-Failure` is sent;

7. some of EAP methods, like EAP-TLS, allows to supplicant and RADIUS server to derive session key. If supplicant used EAP method that do not permit session key derivation, like EAP-MD5, RADIUS server sends session key in `EAP-Success` packet containing `EAP-Key` packet to supplicant. NAS receives generated key in `Access-Accept` packet.

With use of EAP over RADIUS it's mandatory by RFC to use additional authenticator called `Message-Authenticator` and it's described in RFC 3579 [13], because RADIUS authenticator field only protect response messages and those that have password attribute in request messages.

2.4 Fault and intrusion tolerance

The Fault-tolerant protocols are critical in the design of reliable distributed systems. In a dynamic distributed system [12], where nodes can enter and leave the network at any time, the challenge of providing reliable services is even greater. Additionally, due to the wide range of participants and the common use of wireless communication, it is not possible to provide an overview of the processes of all the network. So, each node has partial knowledge of the composition of the system.

Security is a major concern in wireless networks, since the dynamics population of the nodes of the network facilitates the action of malicious nodes on the system. The Byzantine fault model handles malicious attacks through the explicit assumption of existence of corrupted processes, which can behave arbitrarily in an attempt to prevent the system to work properly. The system maintains its proper operation despite the malicious behavior of some of its processes.

During the existence of the AAA architecture, was given more emphasis to the security of authentication protocols and authentication methods than to the architecture resilience. Even with strong authentication protocols and authentication methods, we cannot ensure that component behavior is correct.

As was said before, the AAA architecture can tolerate fail-stops and congestion, replicating components for redundancy. But these replicated components are unable to detect arbitrary behavior of the architecture components. It means, if we have some faulty authentication server, because of an error, attack or intrusion, it can deny access to the legitimate user or allow access to a user who have no credentials on the server, or do not have rights to access this service.

2.4.1 Byzantine Fault Tolerance

The Byzantine fault model [42] assumes that processes can fail arbitrarily, even maliciously. This assumption is quite realistic in practice, given the security problems faced in distributed systems and computer networks in general. Situations that can make hosts behave arbitrarily include the action of successful attackers, the corruption of a program or unintended situations.

The first BFT model was presented in [43], which describes a solution for agreement problems in synchronous system functions in a reliable multicast. The following assumptions are made about inter-process communication:

- *Each correct message sent by a correct process to a correct process is delivered.* This leads to the assumption of reliable or secure routing layer;
- *The receiver knows the sender of the message.* The medium itself can meet this requirement, otherwise mechanisms can be used for message authentication [55];
- *The absence of a message can be detected by correct processes.* In partially synchronous systems [23], such service is not easy to implement, and in fact discussed below as one of the types of failures that must be addressed when you want to deal with Byzantine failures.

There is a work [40] that describes two ways of dealing with malicious processes: the redundancy of information and use of digital not-forged signatures. Both models seek to provide the correct processes a coherent picture of process messages. These techniques, however, do not guarantee that messages sent by faulty processes are consistent with the requirements. One way to handle this situation is adding additional information to the messages in the form of certificates that can be used to validate the content being transmitted.

2.4.2 State Machine Replication

Service replication is one of the most common ways for improving fault tolerance. Due to the necessity of agreement protocols, a state machine replicated services can with $3f + 1$ replicas tolerate up to f faults.

On one hand we have stateless services that are easier to replicate because they don't deal with data management avoiding consistency problems. On the other hand, we have the state-full services, which need to maintain the same state (information) and the same order of execution, where all operations have to be executed exactly in the same order. To ensure statefull systems functionality three properties must to be assured [17]:

- *Initial state.* All correct replicas start in the same state;

- *Determinism.* Two or more machines in the same state when executing the same instructions will reach the same state and this new state is valid;
- *Coordination.* All replicas that are considered correct receive and execute the same sequence of requests.

It is simple to ensure the first property. For the 2nd property, all valid replicas have to behave in a deterministic way, for example, we cannot generate a different random number in each replica.

To ensure the third property it is necessary for a client to send its messages to all the replicas in a same order, ensuring that all replicas receive same sequence of messages. Message order can be ensured using a leader replica, which order messages and send this order to other replicas. If leader replica do not send message order for some specific time, other replicas can suspect it and start leader election protocol.

State machine replication naturally imposes some performance limitations. There is an induced overhead due the total order broadcast among the service replicas. Thus, the performance of the system may not necessarily decrease with the addition of more replicas. The main target of the state machine replication is to increase the service availability and correctness. Beside that, recent work show the overhead is quite affordable [17].

2.5 Related work

In this section we present the most relevant related work.

2.5.1 Kerberos

The primary goal of Kerberos [41, 50] is to eliminate the transmission of unencrypted passwords over the network. It is a strong network authentication protocol for client-server applications based on secret-key cryptography. The protocol provides strong cryptography, so that a client can prove its identity to a server and server to the client using an open and insecure network connection. After client and server used Kerberos, to prove their identity, they can also encrypt all of their communications to assure privacy and data integrity.

Kerberos works without relying on operating system assertions or trusted host addresses, and without requiring physical security of all network hosts. Packets can be read, modified and inserted back to the network, or even be reused. Each entity (client or server) shares a key with Kerberos, which is used to encrypt data and to prove the identity.

Kerberos uses symmetric encryption and a trusted third party key distribution center or KDC component to authenticate and grant user access to a set of network services. The user authentication is done with the KDC instead of the service. The KDC gives a granting ticket to the user, which is valid for a limited period of time, called session. All

Kerberos-based services look for the user ticket and not for the login and password, as it happens on other traditional authentication schemes, but in some cases secret key between user and Kerberos can be generated based on username/password pair.

As explained in [50], Kerberos norm specifies two logical components, authentication service (AS) and ticket granting service (TGS). First allow entity authentication before Kerberos service. Second is intended to mediate authentication between two entities, after both authenticated before the Kerberos (AS). Note that although the AS and TGS are different logical components, does not mean that they are necessarily reside on the different machines. They are typically implemented within the same application.

EAP-Kerberos

An EAP-Kerberos [38] method is not supported nowadays, but was studied as an alternative solution to this work. The main reason this method is not used is because we need to modify existent equipment to support it.

As was said before EAP protocol is used over RADIUS to eliminate the need to modify NAS. We just need to modify end point, supplicant and authentication server, in some cases. This allows us to increase the lifetime of equipment. To use this specific EAP method we need not only modify supplicant and authentication server, but also modify NAS.

Authentication is proceeds with supplicant send KRB-AS-REQ containing client ID to the NAS. As we use EAP method NAS will create RADIUS packet containing EAP packet and send it to authentication server. If authentication is completed with success, authentication server sends KRB-AS-REP containing session key and service ticket. Than this message is resend to the supplicant.

To have network access supplicant send KRB-AS-REQ message, containing a ticket, to the NAS. After sending this message supplicant is authenticated and authorized. As we know Kerberos authentication server must have shared key with client and with the service. In this implementation client shared key is derived using client password and shared key with the service is a key that is used between NAS and authentication service.

Typhon

Usually, Kerberos is implemented using a centralized service, which resides in a single process within a server machine. If such process or machine fail, the authentication service becomes unavailable (fail-stop) or erratic [43] (arbitrary fail or Byzantine). The failure of the authentication service can force the whole system that depends on it to stop.

Typhon [11, 37] is an authentication and authorization service that follows the Kerberos v5 specification and yet is not a point of failure [19]. It increases the robustness of Kerberos safety properties.

The challenge of the Typhon service it is make more robust the four safety properties of Kerberos v5:

- *Authenticity* - those who use the system can prove its identity;
- *Integrity* - the system and communication data is protected against corruption;
- *Confidentiality* - system and communication data is protected from the not authorized access;
- *Availability* - the extent to which the system is being protected from attacks aimed at disrupting its execution.

An authenticity is already offered by Kerberos, since its purpose is to guarantee the authenticity of the users and services that use it. To ensure the other tree safety properties, Typhon was designed to use state machine replication technique combined with a local secure component.

State machine replication is used to provide arbitrary fault tolerance, while the secure components ensure that clients and services keys are kept secret even in the presence of intrusions. This way, secure components allow for confidentiality as they store all sensitive data (i.e., the keys). Also these delegate operations that process the data. The idea is to store sensitive data on the component and to perform operations on them without having them exposed to the rest of the system.

Usually when we replicate a service, it takes longer to perform an operation since the client must wait for responses from more than one server. When using state machine replication the servers themselves need to communicate to reach a consensus. Performance tests conducted to compare Typhon with other similar services revealed that a replicated service can have similar performance figure to other services of the same type.

2.5.2 Grid of smart cards as RADIUS back-end

AAA infrastructures are important in Telecom companies and *eduroam*, using RADIUS servers as basic security element. Normally critical information, such as server keys and cryptographic material, are stored in the server. Any system administrator could have access to them. Server keys may be easily stolen, changed or eavesdropped by those who hold the proper rights on the server. Therefore it is necessary to create an infrastructure that can be trusted even in the presence of malicious administrators.

One way of achieving more security within RADIUS servers is through the use of back-ends secure micro-controllers as authentication servers [68]. Such authentication servers can be implemented using smart cards, making all necessary processing [18]. The RADIUS server receives a request for network access, which is a packet EAP-TLS over RADIUS, removes RADIUS headers and sends the remaining EAP-TLS packet to the smart card, which plays the role of a proxy for subsequent communication.

A smart card [57] contains an embedded computer chip-either a memory or a microprocessor type-that stores and transacts data, used for phone calls, e-cash payment, access credentials, and other applications.

An authentication server that runs on a normal physical machine can authenticate multiple users simultaneously without a great effort, for example using EAP-TLS over RADIUS. But if the server is a smart card that has very limited processing power and very limited resources, performance became an issue. The price of smart cards is low, so it is possible to use several smart cards simultaneously in a grid [66]. Each smart card runs its own server for EAP-TLS with his own TLS certificate.

Chapter 3

System design

This chapter covers design of our system architecture and shows how designed system can be integrated with the existent AAA architecture. In the end of the chapter it is discussed how the new AAA architecture can tolerate faults and attacks.

3.1 Overview

AAA architecture is composed by the three base components, supplicant, NAS and authentication server. Usually, AAA server uses some back-end to verify supplicant credentials, rights, and to provide accounting, as presented on Figure 3.1. Normally both the NAS and the AAA server have some shared secret, that is never sent over the network, and it is used to ensure a packet authenticity and integrity. When the AAA server receives supplicant credentials, it sends them to a distinguished back-end to be validated.

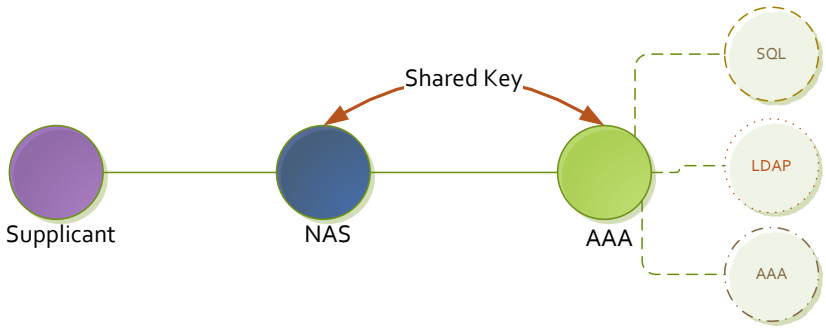

Figure 3.1: AAA architecture with back-ends

When designing our architecture we had special concern to not modify the existent network components closer to the supplicant user. On the Figure 3.2 it is presented a resilient and replicated AAA architecture. This architecture introduces new component called *Untrusted Gateway* or simply *gateway*, used to hide the AAA server replication.

On the original AAA architecture *gateway* is an AAA server and replicated AAA server is a back-end.

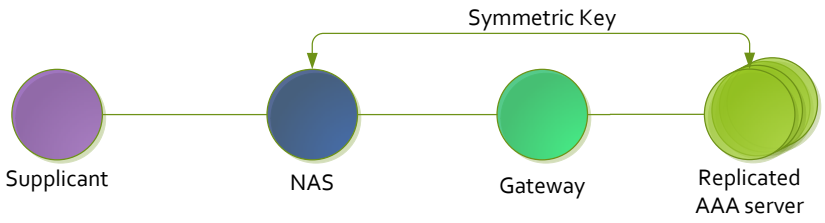

Figure 3.2: Resilient AAA architecture.

In AAA architecture NAS have some shared data with the AAA server to provide integrity and authenticity of the sent packets. All shared data in our implementation is shared between the NAS and the replicated AAA server. With this we can guarantee that the attacker cannot steal any sensible data from the *gateway*. It means, if a *gateway* suffer an intrusion attacker can only drop packets or try to modify them. As *gateway* has no knowledge about shared data between the NAS and replicated AAA server, it cannot generate legitimate packets. If attacker changes the packet it will be discarded.

3.2 Components

In this section we explain each of the components depicted in Figure 3.2.

Supplicant and NAS The supplicant and the NAS are two components of our architecture that will not suffer any modification. It means that every WPA-Enterprise-capable supplicant may access our network and any AAA-capable access point can use our authentication server.

Untrusted Gateway The *Untrusted Gateway* is a component that eliminates the need to change existing NAS and allows us to communicate with the replicated AAA service. This component only forwards packets from the NAS to the replicated AAA server and back.

Replicated AAA server Our server is a replicated fault- and intrusion-tolerant AAA server. Each replica of the server has its own trusted component containing all sensible server data. The trusted component, explained in section 4.2.4, provide function that use sensible server data and guarantee server intrusion tolerance.

3.3 System Model

This section will present system, communication and fault system model.

Network model We assume that all NAS are visible to the supplicants and are connected to at least one *gateway* using the AAA protocol. The supplicant is able to connect to the NAS using the Wi-Fi protocol and its Wi-Fi driver can provide the needed authentication mechanisms. The *gateway* is able to communicate with the authentication server replicas using a BFT protocol.

In this work will be used one of the most used AAA protocols, RADIUS, and one of the strongest authentication methods for the WPA-Enterprise networks, EAP-TLS. Authentication server replicas must be able to communicate between them using the TCP/IP communication model.

Synchrony model System has partial synchrony [24], that allows the execution of a consensus protocol, used for state machine replication.

Fault model The proposed authentication service tolerates Byzantine faults and intrusions as presented in Figure 3.3. In this case we use $3f + 1$ replicas for authentication server, tolerating f faults without affect system correct operation. Byzantine fault tolerance is achieved because the *gateway* expects at least $2f + 1$ equal answers from replicas before responding to the NAS. If such quorum of the requests is not achieved the *gateway* ignores replicas answer and do not respond to the NAS, which will provoke packet to be resent.

The NAS and *gateway* are stateless components and are assumed fail-stop errors and some Byzantine faults detectable by the respective protocols. In general we assume a basic architecture is composed by $f_N + 1$ NAS, $f_G + 1$ *gateways* and $3f + 1$ replicas of authentication server, where f_N, f_G and f are the number of tolerated faults on the different components.

Using the presented architecture in case the NAS fails, the supplicant can choose another one. A *gateway* is connected to the replicated authentication service. As we have $3f + 1$ replicas of AAA server, if f servers fail or become Byzantine the system will continue to work normally. If more than f replicas fail, the system stops.

We cannot detect NAS arbitrary behaviour. If the supplicant fails to authenticate, it can choose another NAS and try again.

NAS has a configured timeout for a packet response time and number of retries. If timeout expires and the number of retries is reached the NAS will choose another *gateway* from the list.

The authentication server is replicated using state machine replication technique and a BFT protocol is used between the *gateway* and replicas. Before replicas respond to

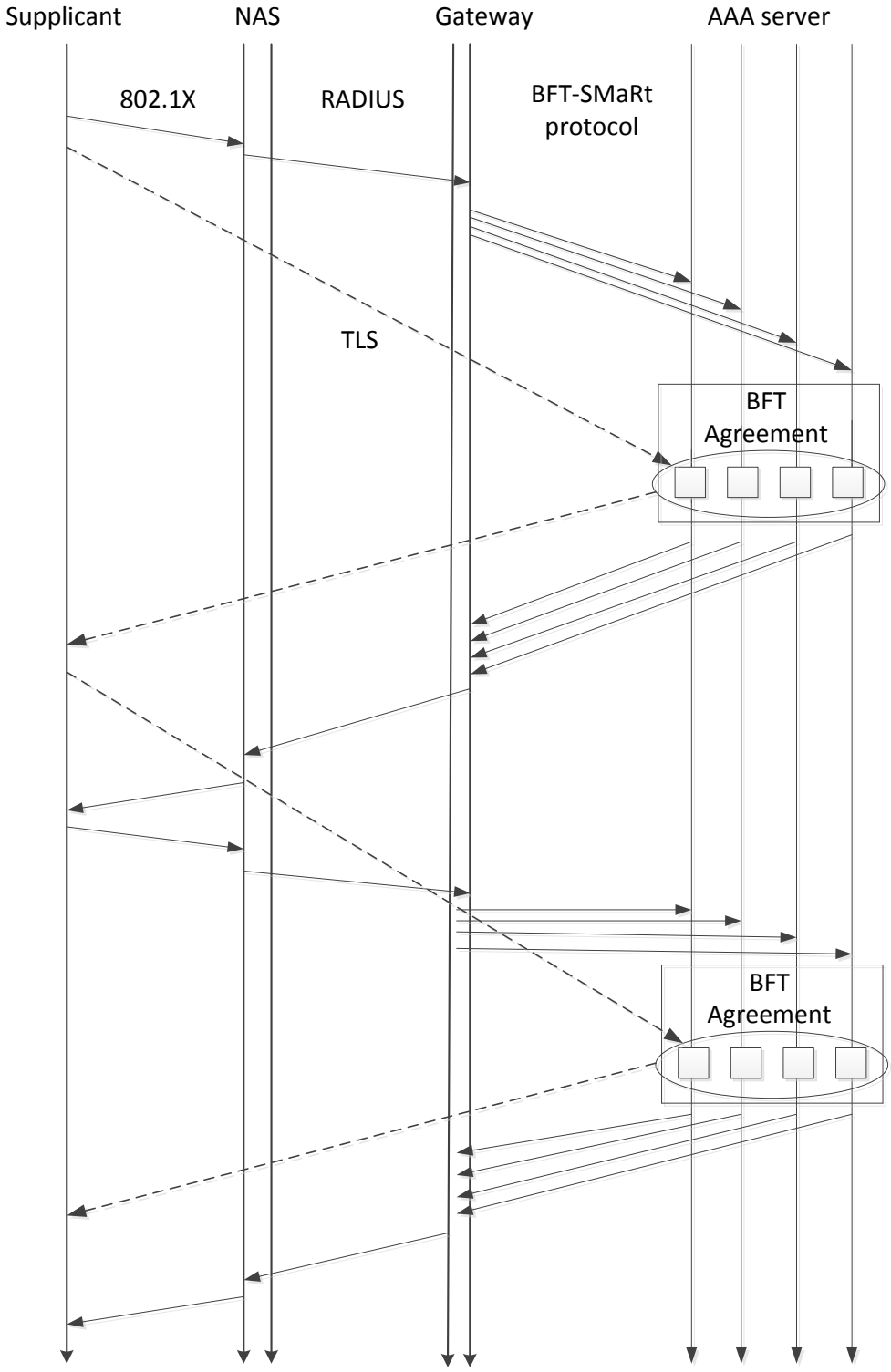

Figure 3.3: Resilient AAA architecture and communication.

the *gateway*, they must reach consensus to not only verify their correctness, but also to guarantee correct response.

During consensus phase replicas exchange series of messages between them to verify who is lying. In every phase of consensus each replica receives messages from all other replicas and compare all of them with its message. With BFT agreement we can guarantee that at lest $2f + 1$ replicas are correct, and client message was processed correctly.

3.4 Protocols

Communication between supplicant and NAS is provided using a 802.1X protocol with the WPA-Enterprise authentication mode. The NAS communicates with the `gateway` using the RADIUS protocol. The *gateway* communicates with the authentication server replicas using a BFT protocol.

Supplicant authentication is performed using the EAP-TLS protocol. EAP-TLS is shuttled between the supplicant and the NAS by a 802.1X protocol and by RADIUS between the NAS and the *gateway*. When the gateway receives a RADIUS packet, it sends it to the replicas and waits for replicas to answers. When replicas send their response to the *gateway*, it waits until it receives at least $2f + 1$ equal responses, and then sends a response back to the NAS. If the quorum of $2f + 1$ is not achieved, *gateway* does not respond to the NAS.

Chapter 4

Implementation

This chapter describes a prototype implementation of the resilient RADIUS architecture. First, we start with the system components and used protocols between them. After, implementation of the communication protocols used among the components are detailed.

4.1 System overview

Our prototype implementation of the architecture is composed by a supplicant, $f + 1$ NAS, $f + 1$ gateway, and $3f + 1$ AAA service replicas, as can be seen in Figure 4.1. Each service replica has a trusted component, used to store and secure sensitive cryptographic material. To simplify the system implementation, supplicant and NAS were implemented as a single component, but they can be used separately. This is only possible because we do not modify them.

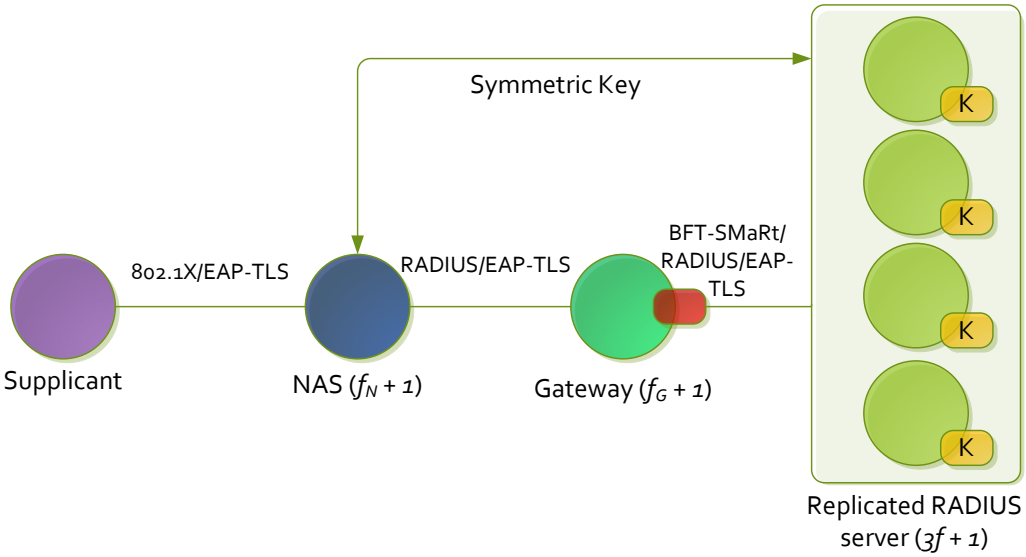

Figure 4.1: Architecture and protocol implementation.

A supplicant needs to use 802.1X protocol combined with WPA-Enterprise mode and EAP-TLS authentication method. 802.1X is required to communicate with the NAS, while EAP-TLS is used to provide end-to-end secure authentication and communication between the supplicant and the authentication service.

BFT-SMaRt [9] library was chosen to provide the BFT protocol between the gateway and replicated authentication servers. It provides two interfaces, one for the client side and another one for the server side. The client interface allows an application to communicate with a replicated service using the BFT protocol based on Paxos [42]. On the other hand, the server side interface allows one to build a replicated services without having to implement protocols such as leader election and total ordering. To mask Byzantine faults, the client interface verifies for a majority $(2f + 1)$ of correct (equal) responses from replicas. Requests with less than $2f + 1$ correct answers are simply discharged.

EAP-TLS allows a strong end-to-end authentication. After authentication, replicas send a master key to the NAS. It is ciphered with the shared secret used between NAS and AAA replicas. Master key is used to cipher communication between 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 a problem on a replicated service, when different replicas will execute the same instructions and have to give an equal answer (same response). Thus, a deterministic random generator is required among replicas. The implemented solution, using a pseudo-random number generator, will be explained later.

4.2 Components

4.2.1 Client

The client represents two system components, supplicant and NAS. It was developed as an authentication module and test tool for RADIUS servers with EAP-TLS support.

The client shares a secret with AAA replicas, as it happens in a normal RADIUS architecture, where NAS and AAA server have a shared secret to secure their communication. Further, the client's NAS part contains a list of authentication servers. In a traditional architecture each authentication server would be an independent AAA service or a simple backup-based replica. However, in our architecture the authentication server configured in the NAS is represented by the *gateway*. Several *gateways* can be used to provide availability, throughput and fault tolerance.

4.2.2 Untrusted Gateway

The gateway has two interfaces, one to communicate with NAS (RADIUS) components and a second one for active replication (to integrate with BFT-SMaRt). User requests arrive at the NAS interface and are forwarded to the BFT-SMaRt interface. *Gateway* is implemented in three layers AAA, packet commutator and BFT. AAA layer is responsible for communication with NAS and AAA packet interpretation. Packet commutator stores packet header data and is responsible to identify to which NAS response packet must be sent. BFT layer is used to communicate with the replicated AAA service.

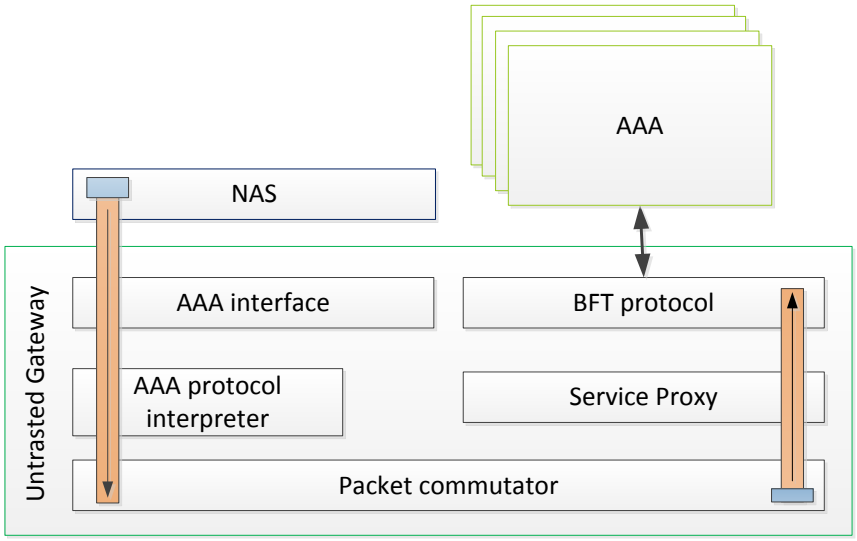

Figure 4.2: *Untrusted Gateway* communication.

Figure 4.2 illustrates the gateway's communication stack. Packets received from NAS are interpreted and forwarded to the service proxy. For each RADIUS incoming packet from NAS the gateway stores header data, including NAS IP address, NAS port and packet identifier for EAP-IDENTITY packet and STATE attribute for the rest of the packets. These information is used to know where to send back the authentication replica responses. Finally, RADIUS packets are sent to the replicas without any modification.

As soon as the gateway receives $2f + 1$ correct responses from AAA replicas, the stored packet information is retrieved to send it back to the respective NAS as the authentication service response. The gateway uses the packet `Identifier` or `State` attribute to recover the corresponding NAS (origin) information.

Inside the *gateway* there are also complementary functions, such as detect and avoid to forward malformed packets. Each received packet, from both interfaces, is verified against a structure template. All packets with correct structure will be forwarded. However, all malformed packets are dropped. Verification functionally comes for free with

packet interpretation, used to identify the NAS that will receive response from replicas.

The gateway packet commutator is implemented using RADIUS packet header and attributes such as CODE, IDENTIFIER fields and STATE attribute. Using CODE field we can identify where the packet must be sent, to NAS or replicas. Table 4.1 presents how *gateway* identifies to which side packet must be sent.

Code	Name	Side
1	Access-Request	AAA replicas
2	Access-Challenge	NAS
3	Access-Reject	NAS
4	Access-Accept	NAS

Table 4.1: Packet commutator - packet codes.

Gateway cannot use the `Identifier` alone to identify a packet because it has only two octets. This limits the range of possible values from 1 to 255. For example, if we have two NAS who sent sequentially a packet with the same `Identifier`, packet commutator will overwrite header data of first NAS. On the other hand, the `State` attribute (16 octets) is a unique value, generated by the authentication server, which can be used as an unique identifier for the subsequent packets of the authentication process. The `State` attribute is maintained until the end of the authentication process and identifies authentication process on the authentication server.

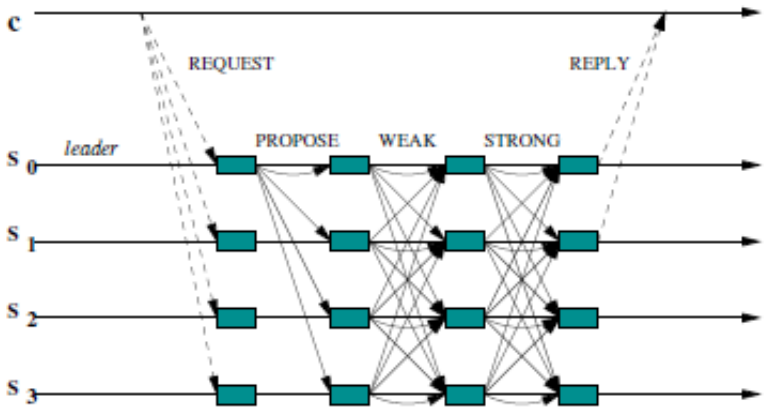

Figure 4.3: BFT-SMaRt communications [60]

Figure 4.3 show the communication between *gateway* and replicas using BFT-SMaRt library. BFT-SMaRt client (`c`) sends the same request to all replicas ($s_1, ..., s_4$). In this example we assume that first replica (s_1) is a leader. After BFT consensus (i.e., `propose`, `weak` and `strong` phases) each replica sends its reply to the client (i.e., *gateway*). The client proxy will look for $2f + 1$ correct answers. If found, the packet will be commuted (forwarded) to the corresponding requester NAS.

4.2.3 Replicated RADIUS server

The replicated RADIUS server implements RADIUS and EAP-TLS protocols. The service replication is implemented using BFT-SMaRt's server side interface.

Packets received from the gateway are processed in three different layers, RADIUS, EAP and TLS. Each layer is responsible to validate, process and create a respective response packet to each request.

The first verification is at the RADIUS layer. If the packet contains an EAP-Message attribute, then it is passed to the EAP layer. If the TLS method is supported, then the EAP layer will forward the EAP data to the next layer. Otherwise, the packet is dropped. Once on the TLS layer, the authentication will start.

The first EAP packet send by a client is the `EAP-Response/Identity`. When replicas receive this packet, they create a new `State` attribute and a new TLS stack. This stack is stored in a hash map using the `State` attribute as the key. The TLS stack is implemented using a light weight library called Bouncy Castle[1].

Usually, the RADIUS authentication server discovers the shared key to use with a specific NAS through its IP address. In our implementation, NAS's IP is hidden from the authentication server, since the gateway is the communication point with NAS components. Thus, we use RADIUS `NAS-Identifier` attribute to transport a NAS identifier to the replicas. For compatibility reasons, we put the NAS IP address in this attribute.

4.2.4 Trusted component

Some considerations are necessary for a trusted component implementation. The amount of code and interfaces must be small. It has to be easy to assure correctness or detect errors. In some cases, number of accesses to the trusted component has to be limited, because accessing component may be slow. For example, in our implementation each call to the trusted component will trigger a BFT protocol between them, which requires message exchange between them during BFT consensus.

The trusted component was designed and implemented to provide all, yet minimal, necessary cryptographic mechanisms for the execution of RADIUS and EAP-TLS protocols. Figure 4.4 shows the interaction with both protocols. First three messages client random, server random and premaster contain a necessary information for master secret generation. Client and server random message are sent in clear text and premaster is ciphered with server public key. When server receives a first message it stores client random number on he trusted component and then trusted component generates server random number. After supplicant receives server random number it will generate premaster and cipher using server public key. As premaster is ciphered server will give it to the trusted component. Trusted component have access to the server private key and can

[1] Official site - `http://www.bouncycastle.org/`

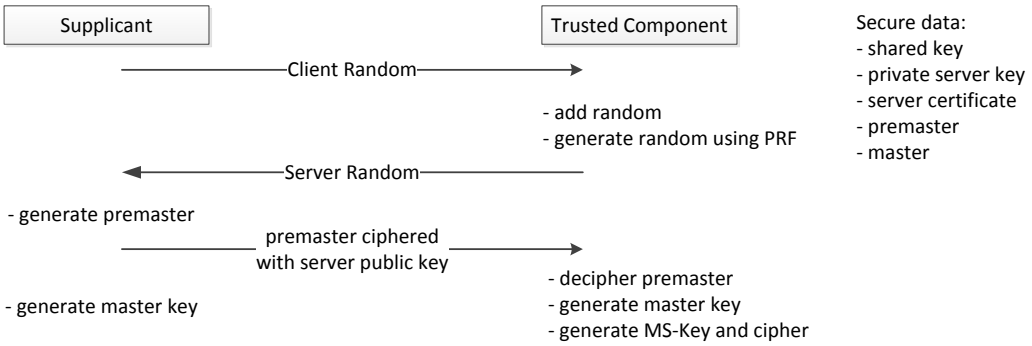

Figure 4.4: Trusted Component interaction during RADIUS/EAP-TLS authentication

decipher premaster and generates master key.

After TLS handshake is complete trusted component generate MS-Key that will be used to protect communication between the supplicant and NAS. MS-Key is ciphered using shared secret between the NAS and authentication server, in our case with trusted component.

Even with use of the state machine replication and BFT protocol we cannot ensure total intrusion-tolerance of the system. If attacker invades one of the authentication server replicas and one *gateway* he can ask trusted component to provide needed operations and generate legitimate packet. To overcame the BFT protocol attacker can modify *gateway* to only accept packets from the corrupted replica. Using this strategy attacker can deny a legitimate user to be authenticated.

Solution to the described problem is to confirm execution of every operation of the trusted component with other trusted components using BFT protocol. In other words, every time server asks trusted component to perform some operation, it will be preformed in at least $2f + 1$ trusted components of other replicas with the same final result. So, attacker can only use trusted component if he have at least $2f + 1$ replicas under his control.

Interface

Table 4.2 presents critical functions from the trusted component interface. *HMACMD5* function makes HMAC-MD5 of data using shared key between NAS and authentication server. *createResponseAuthenticator* used in the RADIUS protocol and creates response authenticator based on the packet data and the request authenticator. *generateSecureRandom* function is used when server needs to generate a random number. Generated random number can vary in size and will be the same in all replicas. Random number generation will be explained later. Size of the random number depends of the need, for example `State` attribute is 16 bytes long and server random, from TLS handshake,

is 48 bytes long. *updateSecret* reads received RADIUS packet and looks for the NAS shared secret based on the NAS IP address. RADIUS packet contains NAS IP on the `NAS-Identifier` attribute. *readPreMaster* decipher premaster key sent by the supplicant using server private key and stores it. *generateMaster* function generates TLS master secret using client random, server random and stored premaster.

Method	**Parameters**	**Return**
HMACMD5	data	HMACMD5 of data using current client shared key
createResponseAuthenticator	packet, packet length, attributes, request authenticator	Response authenticator based on request authenticator.
generateSecureRandom	size	Random number with size bytes.
updateSecret	RADIUS packet	Nothing.
readPreMaster	premaster	Nothing.
generateMaster	Nothing.	Nothing.

Table 4.2: Trusted component interface

Random number generation

One of the functions inside a trusted component is a pseudo random number generator. The random numbers are required to RADIUS (`State` attribute) and TLS (server random) protocols. Thus, the main challenge is to have a deterministic random number generation in a replicated system.

The solution was to use a PRF (pseudo-random number function) with a progressive seed. We used the same PRF of TLS protocol. The key was to define and setup a progressive and deterministic seed. This was accomplished by a HMAC-MD5 and the shared secret between NAS and authentication service. The seed evolution is defined as $seed = HMAC - MD5(secret, seed)$. This progressive seed ensures a secure random number generation. If attacker wishes to guess the next random number or the current seed, he will need to know the initial seed (hard-coded inside the trusted component), all shared keys so far used, and the exact sequence order of the shared keys used for the seed evolution.

4.3 Protocols

Here we explain how EAP-TLS, explained in section 2.3.1, was implemented and adapted for a replicated architecture. In our implementation, all replicas have the same cryptographic material, including the same shared keys with NAS, same certificates and same CA public key. The server sensitive data is stored and secured in the trusted component.

To generate client and server certificates we used *openssl*. Client and authentication server certificates were generated as follows:

1. Creating new CA certificate;
2. Generate client and server certificates;
3. The client and the server certificate are signed using same CA certificate generated in the point one;
4. Each client and server is configured with their respective certificates and CA certificate with CA public key.

This certificates are used as a credentials. During TLS handshake the authentication server and supplicant authenticate mutually using those certificates. CA certificate with the public key is used to verify server and supplicant certificate signature.

We implement only the mandatory TLS protocol, cipher suite and compression method. Mandatory TLS protocol version is TLS 1.0 and mandatory cipher suite, defined in RFC 5216, is TLS_RSA_WITH_3DES_EDE_CBC_SHA. Where:

- TLS_RSA means that we will use TLS with RSA key exchange;
- 3DES_EDE means 3DES in Encrypt-Decrypt-Encrypt (EDE) symmetric encryption mode;
- CBC_SHA means `CBC-mode` encryption and SHA hash algorithm.

The base TLS 1.0 RFC defines only `NULL` compression method (i.e., no compression).

Premaster is a 48 bytes secret, used to generate master secret, is generated by the supplicant because we use RSA premaster secret generation. Master secret is generated using client random, server random and premaster using pseudo-random function (PRF) defined in Algorithm 1. This function is also used for random number generation between replicas.

Algorithm 1 PRF definition

1: **function** PRF($secret, asciiLabel, seed, size$)
2: $label \leftarrow getBytes(asciiLabel)$ ▷ Transforms `asciiLabel` into byte array
3: $s1, s2 \leftarrow devide(secret)$ ▷ Divide secret in two half parts
4: $ls \leftarrow concat(label, seed)$
5: $buf \leftarrow HMAC\_MD5(s1, ls)$
6: $prf \leftarrow HMAC\_SHA1(s2, ls)$
7: **return** $XOR(buf, prf)$
8: **end function**

Algorithm 1 uses a secret, ASCII label and seed to generate pseudo-random numbers. This algorithm divide secret in two half and them using two hash functions, MD5

and SHA1, generates two parts of the random number. It perform two hashes because separately they are weak. And then, PRF XORes result of two hashes.

Chapter 5

Experimental Results

This chapter describes the environment and experimental results of the resilient architecture evaluation. We measured latency and throughput and evaluated the architecture behaviour under crash and Byzantine faults on different components. The results of our solution are compared with results from `FreeRADIUS`[1], one of the most widely used RADIUS server implementation.

5.1 Environment

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 (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 single crash-like fault on the RADIUS server without compromising the system operation.

The evaluation has two main goals: (1) evaluate and compare the system latency and throughput when tolerating Byzantine faults; and (2) analyse the system behaviour under Byzantine faults or attacks. Next sections present the experimental evaluation results for each goal.

5.2 Latency

To measure the system latency we used 2 to 10 supplicants doing 10.000 EAP-TLS authentications each. Resulting measurements represent an average of sets of 500 authentications. Authentications were separated in sets to verify system consistency during the

[1] `http://freeradius.org/`

whole performance test.

The authentication latency calculation required two values (timestamps) stored the each client: one when the authentication starts and a second when the accepts packet is received. The difference between those two timestamps represents the time required to achieve a successful authentication, i.e. authentication latency.

Figure 5.1 presents the latency results. As can be observed, the latency remains nearly constant for the both systems, up to 10 simultaneous clients. However, the replicated RADIUS service adds some extra latency cost to the original architecture. One resilient RADIUS authentication takes 0.2 seconds, while a normal one takes only around 0.1 seconds. Nevertheless, 0.2 seconds is still fast enough. A normal user will not even notice the difference between resilient and non-resilient authentications.

The main problem of the latency lies on the *gateway* and trusted component, because of it implementation. *Gateway* was implemented to process only one packet at a time. It send packet to the replicas and only when replicas respond, it process next NAS request. Other problem that affects latency is a trusted component. Each trusted component access will trigger the execution of the BFT consensus protocol between trusted components of the system. The BFT consensus is needed to provide intrusion tolerance to the system described in 4.2.4.

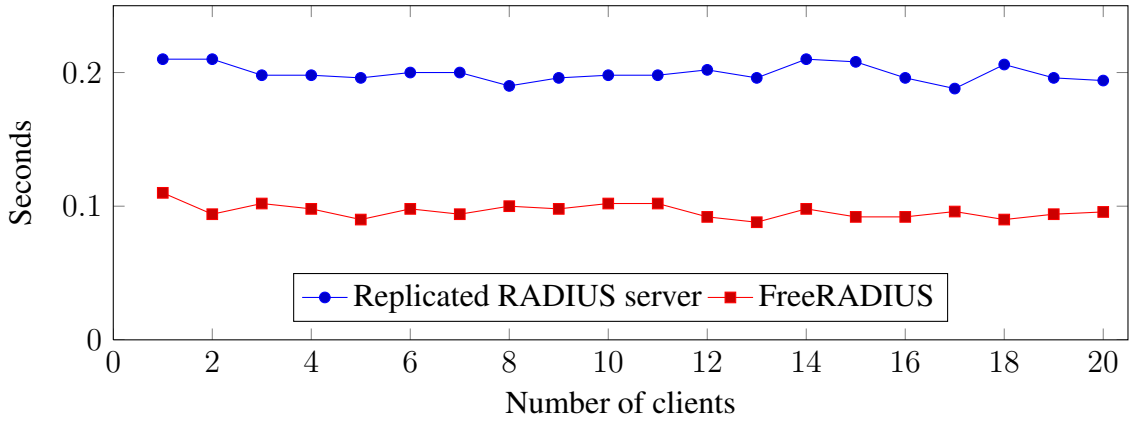

Figure 5.1: Authentication latency

5.3 Throughput

The throughput of the system was measured for 2 to 20 simultaneous supplicants. Each client was configured to execute 10.000 sequential authentications using same credentials. Thus, the average throughput is calculated based on the number of clients multiplied by ten thousand and divided by the time spent to process all authentications.

The throughput measurement was made using a program written in C, which measures the number of packets cached by the `tcpdump`. This program was made to simplify a

throughput measurement, because we have many clients which authenticate simultaneously and they are not synchronised. Each authentication have exactly ten packets, so we can calculate number of performed authentications.

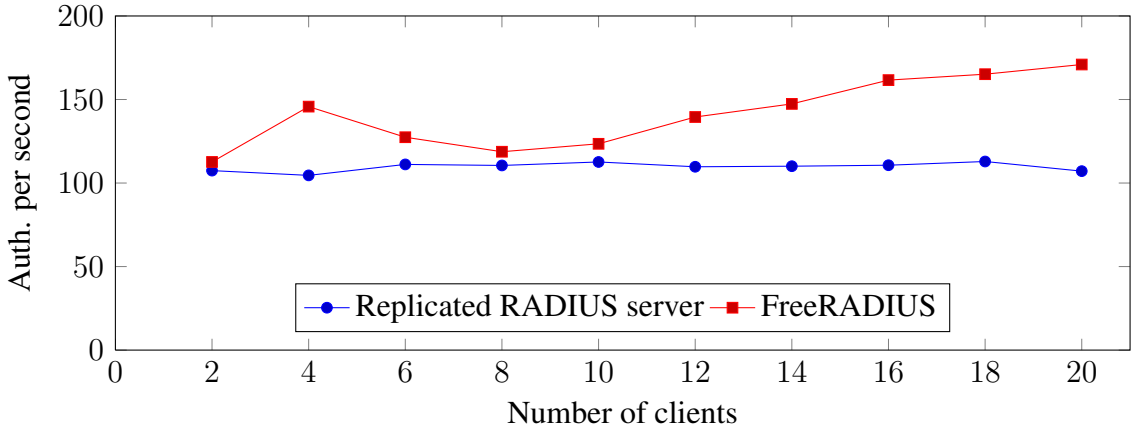

Figure 5.2: Throughput

Figure 5.2 shows the throughput results for both systems. The throughput remains almost stable for the replicated RADIUS. On the other hand, it varies for `FreeRADIUS`. This variation is due to the dynamic pool of threads, which automatically increases the number of working threads based on the number of authentication request. Thus, it causes a slightly decrease in performance from 4 to 10 simultaneous clients. After, by activating new threads, the system's performance goes up again. `FreeRADIUS` was configures with a minimum of 3 active threads and a maximum of 30 threads.

The constance of the replicated RADIUS can be attributed to the *gateway* throughput and server replication. *Gateway* receives packet from the NAS, stores needed data, sends it to the replicas and waits for replicas to respond. Only when replicas respond and *gateway* sends response to the NAS it will process next NAS request. A solution similar to the `FreeRADIUS` (thread pool) could be attempted on the *gateway*, since a possibility to maintain determinism in replicated RADIUS calls, which has not yet been solved. RADIUS server replication also decrease it throughput.

5.4 Attacks

The main idea of this section is to provide some insights and analysis of systems' behaviour under different conditions. We start with one and two faulty replicas, with both crash and Byzantine faults simulation. Further, we evaluate the replicated architecture with one faulty gateway.

Both `FreeRADIUS` and replicated RADIUS are submitted to the exact same tests, such as delay attack and a Byzantine components. To execute those tests the NAS was

configured with a timeout of 3 seconds and 3 times to retry for each authentication server. NAS configuration is a usual configuration used in the real world.

5.4.1 One faulty replica

Attack	FreeRADIUS	Replicated RADIUS
Fail-stop	Authenticated with 9 seconds delay.	Authenticated with no delay and with 9 seconds delay with faulty *gateway*.
Byzantine	Authenticated with max 9 seconds delay.	Authenticated with no delay. In case of faulty *gateway* max delay of 9 seconds.

Table 5.1: Attack results for one faulty replica

Fail-stop. To crush a replica we simply killed process of the authentication server. With only one crashed authentication replica, as presented on the Table 5.1, both `FreeRADIUS` and replicated RADIUS still provide user authentication through the working replicas. Nevertheless, a user may experience a 9 seconds delay due to NAS retry mechanism configuration.

During fail-stop of one replica supplicant will retry to connect. This will provoke NAS to send another request to the authentication server and increase number of retries of the NAS. When number of retries is achieved, NAS will choose another server.

Byzantine. The system replicas were changed to introduce Byzantine behavior. Two kinds of faults have been introduced, malformed packets and communication delays. Each kind of fault represents one attack test.

To induce Byzantine faults on the replicas we have changed their source code to produce errors on the packets (malformed packets) and to produce delays in the communication. For the malformed packet attack we changed packet codes variables for the invalid ones (e.g., for Access-Challenge we changed for code 99 instead of code 2). Also, we changed authenticator field calculation adding some zeros in the end of the packet before authenticator is calculated.

The delay attack consisted in observe system operation despite the presence of a faulty replica, delaying packet delivery. If the server response takes longer than NAS's timeout, NAS will re-send the authentication request. In both cases, `FreeRADIUS` and replicated RADIUS, users could be successfully authenticated with the presence of one faulty replica. During the delay attack the supplicant may retry to authenticate, causing the number of NAS attempts is reached more quickly.

To simulate the second kind of attack we used malformed packets with wrong `Code` and `Authenticator` fields. The replicated RADIUS works well with malformed packets because it does a majority voting. Thus, one faulty replica will be masked. In other words, the gateway will discharge up to f malformed packets (of a single authentication process), since the system is designed to tolerate f faults. On the other hand, the `FreeRADIUS` NAS is forced to change to the another replica when it receives a malformed response. This adds extra communication delays for an user authentication process.

In the case of `FreeRADIUS`, a compromised replica may also leak a derived TLS master key. This would allow an attacker to intersect the communication between a supplicant and the NAS. Even worse, a Byzantine `FreeRADIUS` server may deny access to the legitimate user. A compromised replica of replicated RADIUS would not compromise TLS master secrets or any other sensitive cryptographic material because a trusted component is used to assure the security of those kind of material.

If *gateway* is Byzantine, it can only discard packets that pass through. In case gateway tries to change a packet, NAS and authentication server will notice it, because to generate legitimate packet *gateway* must know shared secret between NAS and authentication server.

5.4.2 Two faulty replicas

Attack	**FreeRADIUS**	**Replicated RADIUS**
Fail-stop	Fail to authenticate with 9 seconds delay.	Fail to authenticate with no delay and with 18 seconds delay with faulty *gateway*.
Byzantine	Fail to authenticate with small delay.	Fail to authenticate with no delay. Fail to authenticate with 18 seconds delay with faulty *gateway*.

Table 5.2: Attack results for two faulty replicas

Fail-stop. With two faulty replicas both systems stopped to work, Table 5.2. `FreeRADIUS` had only two replicas, which means that all available replicas have failed. Despite having four replicas, replicated RADIUS was ready to tolerate only one fault ($f = 1$). Consequently, two faults exceeded the system fault threshold.

Byzantine. Under delay attack, as presented on Table 5.2, with delays lower than the NAS timeout, both systems could still successfully authenticate users. However, if the communication delay is greater than the NAS timeout, authentications will fail.

Two replicas generating malformed packets will cause a system failure, since there are not enough correct resources to proceed one complete and successful authentication. Even if the NAS tries another replica (or gateway), the system will still not be able to finish one correct authentication process.

With malformed packets, the difference resides on the overall system behavior. In the *FreeRADIUS* architecture malformed packets will be sent to NAS. It means that only the NAS will detect the packet formation problem. On the replicated RADIUS architecture this detection is accomplished much earlier, right away on the *gateway*. This reduces the ”try another replica” delay of the NAS. Nevertheless, user will still not authenticate with two faulty replicas, unless f is set to 2 (i.e., system is supporting up to two faults).

Chapter 6

Conclusion and future work

The continues growth of enterprise wireless networks brought more attention to authentication services, such as those provides by AAA servers. One AAA service, such as RADIUS, is usually locally bounded. This means that a user needs one par of id and credentials for each domain. To reduce the multiple identity hassle solution like federated identity providers have been developed. Besides providing inter-domain authentication, making users' life easier, federated identity providers brought a new kind of security treats. One single compromised authentication server can now be used to get access to resources of multiple domains.

RADIUS is an AAA protocol widely used in enterprise networks. One example of a federated RADIUS architecture is `eduroam`. It provides easy user mobility among several universities and institutions in different countries. Thus, any compromised RADIUS service (e.g., RADIUS server of university A) could be used to get access to the network and services of all other federated institutions. An university B is not capable to identify valid or invalid users of university A, for instance. University B trusts on users authenticated by university A.

To avoid the problems that could be caused by compromised local AAA services, we presented a replicated and resilient RADIUS architecture. The architecture is capable of masking up to f Byzantine faults. This means that events such as an attack can compromise up to f system replicas without compromising the overall system liveness and safety. The main building blocks to guarantee these properties are a state machine replication library and a trusted component.

A prototype of the architecture was implemented as a proof of concept. The implementation is based on Java and makes use of BFT-SMaRt [9] library. It follow the standard RADIUS specification, without requiring any system modification for deploying it in real world RADIUS-based identity provider infrastructures. This compatibility is accomplished through a new component called *gateway*. It is located between the NAS and the traditional RADIUS authentication service.

Further, user authentication is ensured and secured by an end-to-end EAP-TLS secure

channel. Thus, eavesdrop at any point between the end user and the authentication service is not possible without breaking the TLS security. Nevertheless, implement a replicated TLS process was one of the main challenges of the work. The replicated TLS authentication was achieved through a combination of a pseudo random generator, cryptographic keys and synchronized seeds among replicas.

Finally, the prototype was evaluated and compared to `FreeRADIUS`, one of the most popular RADIUS implementations. Both authentication latency and throughput are slightly higher in the replicated RADIUS architecture. However, they do not affect the system operations and end users will not even perceive the difference. For instance, the authentication latency is around 200 ms, which is not enough to cause a significant delay noticeable by the end user. Thus, the results demonstrate the applicability and feasibility of the proposed solution. Nevertheless, there is still room for improvements in terms of performance and intrusion tolerance. Latency and throughput can be improved by introducing multi-thread mechanisms in the architecture. Intrusion tolerance can be improved with the support of diversity and proactive recovery mechanisms.

Bibliography

[1] Diameter Base Protocol. 2003.

[2] B Aboba, L Blunk, J Vollbrecht, J Carlson, and H Levkowetz. RFC 3748: Extensible Authentication Protocol (EAP). *Request for Comments*, 2004.

[3] B Aboba and J Wood. RFC 3539: Authentication, Authorization and Accounting (AAA) Transport Profile. *Request for Comments*, 2003.

[4] Monis Akhlaq, Baber Aslam, Muzammil A. Khan, John Mellor, M. Noman Jafri, and Irfan Awan. Performance evaluation of IEEE 802.1x authentication methods and recommended usage. *WTOC*, 7:133–143, March 2008.

[5] Ai Al-Alawi and Ma Al-Amer. Young generation attitudes and awareness towards the implementation of smart card in Bahrain: an exploratory study. *Journal of Computer Science*, 2(5):441–446, 2006.

[6] Khidir M. Ali and Ali Al-Khlifa. A Comparative Study of Authentication Methods for Wi-Fi Networks. In *Proceedings of the 2011 Third International Conference on Computational Intelligence, Communication Systems and Networks*, CICSYN '11, pages 190–194, Washington, DC, USA, 2011. IEEE Computer Society.

[7] Jon Allen and Jeff Wilson. Securing a wireless network. In *Proceedings of the 30th annual ACM SIGUCCS conference on User services*, SIGUCCS '02, pages 213–215, New York, NY, USA, 2002. ACM.

[8] Smart Card Alliance. Identity Management Systems Smart Cards and Privacy: Frequently Asked Questions. `http://www.smartcardalliance.org/resources/pdf/identity_faq.pdf`.

[9] João Sousa Alysson Bessani and Marcel Santos. BFT-SMaRt - High-performance Byzantine Fault-Tolerant State Machine Replication. `http://code.google.com/p/bft-smart/`.

[10] S. Ansari, S.G. Rajeev, and H.S. Chandrashekar. Packet sniffing: a brief introduction. *Potentials, IEEE*, 21(5):17 – 19, 2002/jan 2002.

[11] Jo ao Sousa. Typhon: Um servico de autenticacão e autorizacão tolerante a intrusões. Master's thesis, University of Lisbon, 2010.

[12] Algirdas Avižienis. Design of fault-tolerant computers. In *Proceedings of the November 14-16, 1967, fall joint computer conference*, AFIPS '67 (Fall), pages 733–743, New York, NY, USA, 1967. ACM.

[13] P. Calhoun B. Aboba, Microsoft and Airespace. RFC 3579: RADIUS (Remote Authentication Dial In User Service) Support For Extensible Authentication Protocol (EAP). *Request for Comments*, 2003.

[14] W. Simpson B. Lloyd, L&A and Daydreamer. Rfc 1334 - ppp authentication protocols. 1992, 1992.

[15] G. R. Blakley. Safeguarding cryptographic keys. *Managing Requirements Knowledge, International Workshop on*, 0:313–317, 1979.

[16] Steven K. Brawn, R. Mark Koan, and Kelly Caye. Staying secure in an insecure world: 802.1x secure wireless computer connectivity for students, faculty, and staff to the campus network. In *Proceedings of the 32nd annual ACM SIGUCCS fall conference*, SIGUCCS '04, pages 273–277, New York, NY, USA, 2004. ACM.

[17] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance and proactive recovery. *ACM Trans. Comput. Syst.*, 20:398–461, November 2002.

[18] Serge Chaumette, Pascal Grange, Achraf Karray, Damien Sauveron, and Pierre Vignéras. Secure distributed computing on a Java Card; Grid.

[19] Miguel Correia, Paulo Veríssimo, and Nuno Ferreira Neves. The design of a COTS real-time distributed security kernel. In *In Proceedings of the Fourth European Dependable Computing Conference*, pages 234–252, 2002.

[20] Lol Grant D. Carrel and Cisco Systems. The TACACS+ Protocol. *Network Working Group*, 1997(01/10), 1997.

[21] R. Hurst D. Simon, B. Aboba and Microsoft Corporation. RFC 5216: The EAP-TLS Authentication Protocol. *Request for Comments*, 2008.

[22] Ram Dantu, Gabriel Clothier, and Anuj Atri. EAP methods for wireless networks. *Comput. Stand. Interfaces*, 29:289–301, March 2007.

[23] Xavier Défago, André Schiper, and Péter Urbán. Total order broadcast and multicast algorithms: Taxonomy and survey. *ACM Comput. Surv.*, 36(4):372–421, December 2004.

[24] Cyntia Dwork, Nancy A. Lynch, and Larry Stockmeyer. Consensus in the Presence of Partial Synchrony. *J. ACM*, 35(2):288–322, 1988.

[25] Pierre-Alain Fouque, Gaëtan Leurent, and Phong Q. Nguyen. Full key-recovery attacks on HMAC/NMAC-MD4 and NMAC-MD5. In *Proceedings of the 27th annual international cryptology conference on Advances in cryptology*, CRYPTO'07, pages 13–30, Berlin, Heidelberg, 2007. Springer-Verlag.

[26] Y Frankel and Y G Desmedt. Parallel reliable threshold multisignature. *Security*, 1992.

[27] C Gahlin. Secure ad hoc networking. Master's thesis, University of Umeå, 2004.

[28] S.B. Guthery. Java card: Internet computing on a smart card. *Internet Computing, IEEE*, 1(1):57 –59, jan/feb 1997.

[29] Peter Gutmann. Everything you Never Wanted to Know about PKI but were Forced to Find Out What is Public Key Infrastructure Certificate History (ctd) Original paper on public-key encryption proposed the. *Computer*, 2005.

[30] J. Salowey H. Haverinen, Nokia and Cisco. EAP SIM Authentication, draft-haverinenpppext-eap-sim-12. *Request for Comments*, 2004(01/10), 2003.

[31] H. Haverinen and J. Salowey. EAP AKA Authentication, draft-arkko-pppexteap-aka-12. *Request for Comments*, 2004(01/10), 2004.

[32] Lianfen Huang, Ying Huang, Zhibin Gao, Jianan Lin, and Xueyuan Jiang. Performance of Authentication Protocols in LTE Environments. *2009 International Conference on Computational Intelligence and Security*, 2:293–297, 2009.

[33] Antonio Izquierdo, Nada Golmie, Katrin Hoeper, and Lidong Chen. Using the EAP framework for fast media independent handover authentication. In *Proceedings of the 4th Annual International Conference on Wireless Internet*, WICON '08, pages 42:1–42:8, ICST, Brussels, Belgium, Belgium, 2008. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[34] H. Haverinen J. Arkko, Ericsson and Nokia. RFC 4187 - Remote Authentication Dial In User Service (RADIUS). *Request for Comments*, 2006(01/10), 2006.

[35] P. Krumviede J. Klensin, R. Catoe and MCI. RFC 2195: IMAP/POP AUTHorize Extension for Simple Challenge/Response. *Request for Comments*, 1997.

[36] K Jiejun and Z Petros. Providing robust and ubiquitous security support for mobile ad-hoc networks. *Proceedings Ninth International Conference on Network Protocols ICNP 2001*, pages 251–260, 2001.

[37] Alysson Bessani João Sousa and Paulo Sousa. Typhon : Um servico de autenticacão e autorização tolerante a intrusões. *Security*, 2011.

[38] Mohamed Kâafar, Lamia Benazzouz, Farouk Kamoun, and Davor Males. A kerberos-based authentication architecture for wireless lans. In Nikolas Mitrou, Kimon Kontovasilis, George Rouskas, Ilias Iliadis, and Lazaros Merakos, editors, *NETWORKING 2004. Networking Technologies, Services, and Protocols; Performance of Computer and Communication Networks; Mobile and Wireless Communications*, volume 3042 of *Lecture Notes in Computer Science*, pages 1344–1353. Springer Berlin / Heidelberg, 2004.

[39] C. Kaufman, R. Perlman, and M. Speciner. *Network security: private communication in a public world*. Prentice Hall series in computer networking and distributed systems. Prentice Hall PTR, 2002.

[40] Kim Potter Kihlstrom, Louise E. Moser, and P. M . Melliar-Smith. Byzantine Fault Detectors for Solving Consensus. *The Computer Journal*, 46:2003, 2003.

[41] John T Kohl, B Clifford Neuman, and Theodore Y Ts'o. The Evolution of the Kerberos Authentication Service. *Society*, 1991.

[42] Leslie Lamport. Fast paxos. *Distributed Computing*, 19(2):79–103, 2006.

[43] Leslie Lamport, Robert Shostak, and Marshall Pease. The Byzantine Generals Problem. *ACM Trans. Program. Lang. Syst.*, 4:382–401, July 1982.

[44] David Q. Liu and Mark Coslow. Extensible authentication protocols for IEEE standards 802.11 and 802.16. In *Proceedings of the International Conference on Mobile Technology, Applications, and Systems*, Mobility '08, pages 47:1–47:9, New York, NY, USA, 2008. ACM.

[45] Yue Ma and Xiuying Cao. HOW TO USE EAP-TLS AUTHENTICATION IN PWLAN ENVIRONMENT. In *Neural Networks and Signal Processing*. IEEE, 2003.

[46] Ueli M. Maurer and Pierre E. Schmid. A Calculus for Secure Channel Establishment in Open Networks. In *Proceedings of the Third European Symposium on Research in Computer Security*, ESORICS '94, pages 175–192, London, UK, UK, 1994. Springer-Verlag.

[47] Microsoft. Choosing EAP-TLS or MS-CHAP v2 for User-Level Authentication. `http://technet.microsoft.com/en-us/library/cc739638(WS.10).aspx`, March 2003.

[48] Hassnaa Moustafa, Gilles Bourdon, and Yvon Gourhant. Authentication, authorization and accounting (AAA) in hybrid ad hoc hotspot's environments. In *Proceedings of the 4th international workshop on Wireless mobile applications and services on WLAN hotspots*, WMASH '06, pages 37–46, New York, NY, USA, 2006. ACM.

[49] Madjid Nakhjiri and Mahsa Nakhjiri. *AAA and Network Security for Mobile Access: Radius, Diameter, EAP, PKI and IP Mobility*. John Wiley and Sons, Ltd, 2006.

[50] B.C. Neuman and T. Ts'o. Kerberos: an authentication service for computer networks. *Communications Magazine, IEEE*, 32(9):33 –38, sep 1994.

[51] N.F. Neves, M. Correia, and P. Verissimo. Solving vector consensus with a wormhole. *Parallel and Distributed Systems, IEEE Transactions on*, 16(12):1120 – 1131, dec. 2005.

[52] W. Rankl and W. Effing. *Smart card handbook*. Wiley, 2003.

[53] C Rigney, S Willens, A Rubens, and W Simpson. RFC 2865 - Remote Authentication Dial In User Service (RADIUS). *Request for Comments*, 2007(01/10), 2000.

[54] R. Rivest. RFC 1321: The MD5 Message-Digest Algorithm. *Request for Comments*, 1992.

[55] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21:120–126, February 1978.

[56] Augustin P. Sarr, Philippe Elbaz-Vincent, and Jean-Claude Bajard. A secure and efficient authenticated Diffie-Hellman protocol. In *Proceedings of the 6th European conference on Public key infrastructures, services and applications*, EuroPKI'09, pages 83–98, Berlin, Heidelberg, 2010. Springer-Verlag.

[57] Robbie Schaefer, Wolfgang Mueller, Andres Marán López, and Daniel Díaz Sánchez. Using smart cards for secure and device independent user interfaces. In *Proceedings of the 4th international conference on mobile technology, applications, and systems and the 1st international symposium on Computer human interaction in mobile technology*, Mobility '07, pages 743–750, New York, NY, USA, 2007. ACM.

[58] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[59] William Allen Simpson. PPP Challenge Handshake Authentication Protocol (CHAP). Internet RFC 1994, August 1996.

[60] Joao Sousa and Alysson Bessani. From Byzantine Consensus to BFT State Machine Replication: A Latency-Optimal Transformation. In *Dependable Computing Conference (EDCC), 2012 Ninth European*, pages 37 –48, may 2012.

[61] Martin Sutter. Realization of a Vision: Authentication and Authorization Infrastructure for the Swiss Higher Education Community. EDUCAUSE Annual Conferences.

[62] Certicom T. Dierks and C. Allen. RFC 2246: The TLS Protocol Version 1.0. *Request for Comments*, 1999.

[63] Chad Teat and Svetlana Peltsverger. The security of cryptographic hashes. In *Proceedings of the 49th Annual Southeast Regional Conference*, ACM-SE '11, pages 103–108, New York, NY, USA, 2011. ACM.

[64] Michael Treaster. A Survey of Fault-Tolerance and Fault-Recovery Techniques in Parallel Systems. *CoRR*, abs/cs/0501002, 2005.

[65] Yuh-Min Tseng. USIM-based EAP-TLS authentication protocol for wireless local area networks. *Computer Standards Interfaces*, 31(1):128–136, 2009.

[66] P. Urien and M. Dandjinou. Introducing Smartcard Enabled RADIUS Server. In *Collaborative Technologies and Systems, 2006. CTS 2006. International Symposium on*, pages 74 – 80, may 2006.

[67] Pascal Urien, Estelle Marie, and Christophe Kiennert. An Innovative Solution for Cloud Computing Authentication: Grids of EAP-TLS Smart Cards. *2010 Fifth International Conference on Digital Telecommunications*, pages 22–27, 2010.

[68] Pascal Urien and Guy Pujolle. TEAPM, Trusted EAP modules. In *e-Smart2006*, September 2005.

[69] Pascal Urien and Guy Pujolle. Security and privacy for the next wireless generation. *Int. J. Netw. Manag.*, 18:129–145, March 2008.

[70] Eun-Jun Yoon, Wan-Soo Lee, and Kee-Young Yoo. Secure PAP-Based RADIUS Protocol in Wireless Networks. In De-Shuang Huang, Laurent Heutte, and Marco Loog, editors, *Advanced Intelligent Computing Theories and Applications. With Aspects of Contemporary Intelligent Computing Techniques*, volume 2 of *Communications in Computer and Information Science*, pages 689–694. Springer Berlin Heidelberg, 2007.