Pervasive Secure Content Delivery Networks Implementation

2017

Hector Lugo-Cordero
University of Central Florida

PERVASIVE SECURE CONTENT DELIVERY NETWORKS IMPLEMENTATION

by

HECTOR M LUGO-CORDERO
B.S. University of Puerto Rico, Mayagüez Campus, 2006
M.S. University of Puerto Rico, Mayagüez Campus, 2009

A thesis submitted in partial fulfilment of the requirements
for the degree of Master of Science
in the Department of Computer Science
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Spring Term
2017

ABSTRACT

Over the years, communication networks have been shifting their focus from providing connectivity in a client/server model to providing a service or content. This shift has led to topic areas like Service-Oriented Architecture (SOA), Heterogeneous Wireless Mesh Networks, and Ubiquitous Computing. Furthermore, probably the broadest of these areas which embarks all is the Internet of Things (IoT). The IoT is defined as an Internet where all physical entities (e.g., vehicles, appliances, smart phones, smart homes, computers, etc.), which we interact daily are connected and exchanging data among themselves and users. The IoT has become a global goal for companies, researchers, and users alike due to its different implementation and functional benefits: performance efficiency, coverage, economic and health. Due to the variety of devices which connect to it, it is expected that the IoT is composed of multiple technologies interacting together, to deliver a service. This technologies interactions renders an important challenge that must be overcome: how to communicate these technologies effectively and securely? The answer to this question is vital for a successful deployment of IoT and achievement of all the potential benefits that the IoT promises.

This thesis proposes a SOA approach at the Network Layer to be able to integrate all technologies involved, in a transparent manner. The proposed set of solutions is composed of primarily the secure implementation of a unifying routing algorithm and a layered messaging model to standardize communication of all devices. Security is targeted to address the three main security concerns (i.e., confidentiality, integrity, and availability), with pervasive schemes that can be employed for any kind of device on the client, backbone, and server side. The implementation of such schemes is achieved by standard current security mechanisms (e.g., encryption), in combination with novel context and intelligent checks that detect compromised devices. Moreover, a decentralized content processing design is presented. In such design, content processing is handled at the client side, allowing server machines to serve more content, while being more reliable and capable of processing complete security checks on data and client integrity.

This thesis is dedicated first of all to God for giving me the most precious of all gifts, life. Secondly, I want to dedicate this thesis to my family for being always there for me and making me the person I am today.

To my advisor, Dr Ratan K. Guha, and my friends that have been there during my best days and my worst days when I needed them the most.

ACKNOWLEDGMENTS

I would like to thank my advisor, Dr. Ratan K. Guha for his support and guidance throughout my graduate studies at UCF, the development of the present thesis and PhD dissertation, as well as life and work advises.

Special thanks as well to my graduate committee Dr. Annie Wu and Dr. Kenneth Stanley, who helped with their teachings which became foundation toward this work.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1: IINTRODUCTION

Wireless technologies such as Wireless Sensor Networks (WSN), and Wireless Mesh Networks (WMN), have acquired great importance and progress during the last years. It has now become possible to deploy such networks to obtain data from an area, or provide internet access to mobile end users [1]. With the creation of Wireless Metropolitan Area Networks technologies (i.e. WiMAX and LTE), content distribution can be obtained even from long distances, in practically real-time.

With these wide range of technologies, many domains such as Ubiquitous Computing and Urban Sensing are able advance into providing content that was first not possible. Example of such content may include video, traffic updates, or precise route directions considering local information.

However, to achieve such goals, a unifying mechanism is needed with service provision as the main goal. Such services can be provided, independently of connection type between nodes. To accomplish such integration, the Service-Oriented Architecture (SOA) has was proposed [2].

Services in SOA are loosely coupled. SOA is composed of 1) a service provider or producer (i.e. server), to which clients subscribe and requests services from, 2) the service consumer (i.e. client), and 3) a message independent of the service, which can carry the control information and content of services. Traditionally SOA relies on web processing and XML messages, such as the Simple Object Access Protocol (SOAP), which may not be available to all nodes.

This thesis targets integration at the network layer, and leaves the application layer to provide a uniform interface using Single Page Applications (SPA) with REpresentational State Transfer (REST) services. With such an integration we can exploit their individual advantages into a global system which can be applied to multiple areas. An example of such area can be the Internet of Things (IoT), where content provision must be provided pervasively and securely. Failing to address security as a design principle of services in heterogeneous networks will prevent efficient reaction of network attacks. An occurrence of this could be seen on October 2016, where an unknown entity attacked a large Domain Name Service provider in the US, using a large amount of IoT devices to cause a Denial of Service attack. The enterprise was able to detect the presence of a DDos early on, but immediate action was not possible in the current network architecture.

In order to complete the proposed integration, a unifying architecture and routing algorithm is proposed, keeping security as part of its core design. In particular, we target the confidentiality of the data and anonymity of endpoints. These requirements are achieved via a enhancement implementation of the routing algorithm proposed in [3], which we are denoting as ESORA (Enhanced Service-Oriented Routing Algorithm).

We also target the integration of content services at the application layer, by providing a common interface for all technologies. The interface would takes advantage of the advance computational power that the network endpoints these days posses, to implement a thick client, which can maximize resilient content distribution using a transparent credential storage system.

The reminder of this thesis provides first some background on SORA and enhancements required for adapting ESORA. Such enhancements, presented in Chapter 2, include the addition of new routing metrics to achieve more resiliency and load balancing properties as well as a unifying architecture. Following such introduction, we elaborate on the third layer of the model, known as the technology layer. In such layer, all technologies integrate as one backbone via the Enhanced Service-Oriented Routing Algorithm (ESORA) in Chapter 3. ESORA acts as a unifying routing service, originally intended to provide SOA at the network layer, now focusing on content provision and routing resources properly. Afterwards, in Chapter 4, we introduce the front end located at the final layer of the messaging model. Such front end delivers content uniformly using a thick client approach known as Single Page Application (SPA). The front end uses SPA and enhances it with security approaches to detect anomalies in user patterns distributed and defend the integrity of the resources and client participants. Chapter 5 then closes the thesis with some final remarks on the proposed systems.

CHAPTER 2: SERVICE-ORIENTED ARCHITECTURE AT THE NETWORK LAYER

During recent years, there has been a common interest for integrating information processing into everyday tasks, with the goal of achieving better quality of life for everyone. Such integration, commonly referred to as the Internet of Things (IoT), requires a natural human-computer interaction. Hence, traditional approaches may not be appropriate, because of the need of a transparent device-to-device communication [4], which may include many types of network technologies.

To address the integration issue, a Versatile Service-Oriented Wireless Mesh Network (VESO-Mesh) has been proposed [5]. VESO-Mesh integrates network communication with Signal Processing tools, inside of its nodes, known as NETSIG (Networking + Signal Processing). VESO-Mesh aims to provide optimal services, based on the principle of locality of service and heterogeneous wireless mesh networks, under a unifying architecture.

This chapter covers the proposed unifying architecture, which employs a layered approach to provide content on Heterogeneous Wireless Mesh Networks (HWMN). At its core, the Service-Oriented Routing Algorithm (SORA), originally presented in [3] and [6], is adopted with some enhancements.

During the reminder of the chapter we elaborate on the design of the proposed unifying architecture, as means of establishing the required background to build up a secured Content Delivery Network (CDN). But first we introduce the concepts of Heterogeneous Networks and Service-Oriented Architecture as means to delivering pervasively the content. Afterwards, the latest version of SORA is explained. SORA resides at the technology layer of the unifying architecture; other layers of the architecture following SORA's explanation. Lastly, we present the advantages and disadvantages of SORA. Such disadvantages are later addressed in Chapter 3.

Heterogeneous Networks

This section presents the design and benefits of heterogeneous wireless mesh networks. The term heterogeneous refers to different technologies [7], which may be determine according to processing capabilities (e.g. Zigbee, WiFi, LTE/WiMAX, NETSIG [5], etc.). The idea of NETSIG arrives from the evolution of telephone circuits. It started with mobile phones as part of the phone switch network (0G), and then it moved to a separate but still analog network (1G). Afterwards, the second generation of phones arrived (2G) with a digital scheme known as GSM. During the third generation (3G), voice and data circuits where integrated into one, providing mobile users with easy access to the Internet, pictures, message, phone calls, etc. Finally, the fourth generation (4G) networks are now starting to rise, to provide high speed mobile access to all the 3G functionality, and add real-time video on demand.

Following the principle, we now consider integrating communications with signal processing. The approach can help monitoring and analysis of environmental events of interests (see Figure 2.1).

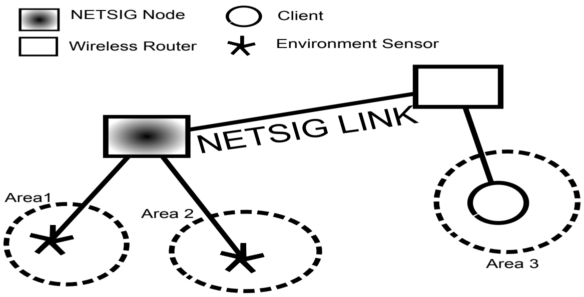


Figure 2.1: NETSIG Scenario

From home researchers may learn about species of interest, or possible events (e.g. earthquake). NETSIG allows to receive concrete information from different areas, whereas without signal processing data would have no clear meaning.

Non-IP networks may communicate through the VESO-Mesh via subscriptions to multi-cast address. For example, consider a Zigbee network (IEEE802.15.4) monitoring the temperature. Such a network can extend its range with a bridging node (e.g. IEEE802.11n), which maps the multicast address 239.7.15.83. Upon receiving a packet, the bridging node verifies if the destination address is 239.7.15; if so, the content of its payload should contain embedded the structure of the Zigbee packet.

Service-Oriented Architecture

Traditionally, the web backbone implementation is focused on providing efficient end-to-end connection. On occasion, this leaves the user with the opportunity to select which specific server will provide a request (i.e., mirror selection). This process many times leads to sub-optimal results, as physically close connections may not be the best choice due to network topology factors. Some factors that can contribute to this phenomenon can be the number of hops between source and destination and the quality of the link between them. Although routing algorithms will find the best connection between the source and destination, they will not explore other routes that provide the same service as the request that comes from the client is to connect to the specific server.

The Service-Oriented Architecture (SOA) takes a different approach, by establishing context of services and making the network able to choose better servers that can provide the same service. Such context is established via endpoints, which can be described using the Web Services Denifintion Language (WSDL). WSDLs describe the services, while the Simple Object Access Protocol (SOAP) describes the communication commonly using XML via the Hypertext Transfer Protototocol (HTTP) or the Simple Mail Transfer Protocol (SMTP) services. SOA is being extensively used in both academia [8] and enterprises such as Netflix.

In [3] and [6], SOA was targeted at the network layer via a cross-layer routing algorithm with load balancing effects, known as Service-Oriented Routing Algorithm (SORA). SORA gives advantage of providing applications and devices access to use the benefits of SOA without the need of connecting to WSDLs. This thesis elaborates on the the original design of SORA to achieve the same solution in a pervasive and secure environment, such that more coverage can be achieved for the IoT.

SORA is enhanced throughout the rest of this thesis at the network layer and web service layer (employing REST services with JSON), implementing security at each solution. In the following sections we describe the latest version of SORA.

A Unifying Routing Algorithm: SORA

The original work of the Service-Oriented Routing Algorithm (SORA) was presented in [3] and [6], upon which we extend its functionality to meet the security requirements. For completeness, this section presents the essentials of SORA. SORA uses a DNS-like (Domain Name Service like) approach to create a mapping of services to the IP Addresses of the servers which provide such services. When a server wishes to join the network, it sends a control packet to its gateway telling the services it can provide. A server may be seen as a host, or network, which provide a one or more services. Then, the mesh network creates a service hash to keep a history of the services available on the mesh. Such hash makes possible the integration of non TCP/UDP services (e.g. sensor/actuator applications). For example, if initially a server provides FTP which is a traditional service, the network assigns the traditional ports 20 and 21 for it, whereas non-traditional services such as Temperature-Sensor may get assigned unused ports after 1024. This hash is broadcast to the adjacent nodes, increasing the hop count by one, before sending. Broadcasting stops when the receiving node learns no new routes, and the received message has a higher hop count than the one already stored for each of the routes. An example of the SORA's original routing table can be seen in Table 2.1.

Table 2.1: Classic Routing Table Example

Service	**Interface**	**# Hops**
25	172.16.2.1	2
21	172.16.3.1	2
21	172.16.3.1	4
25	172.20.1.1	2

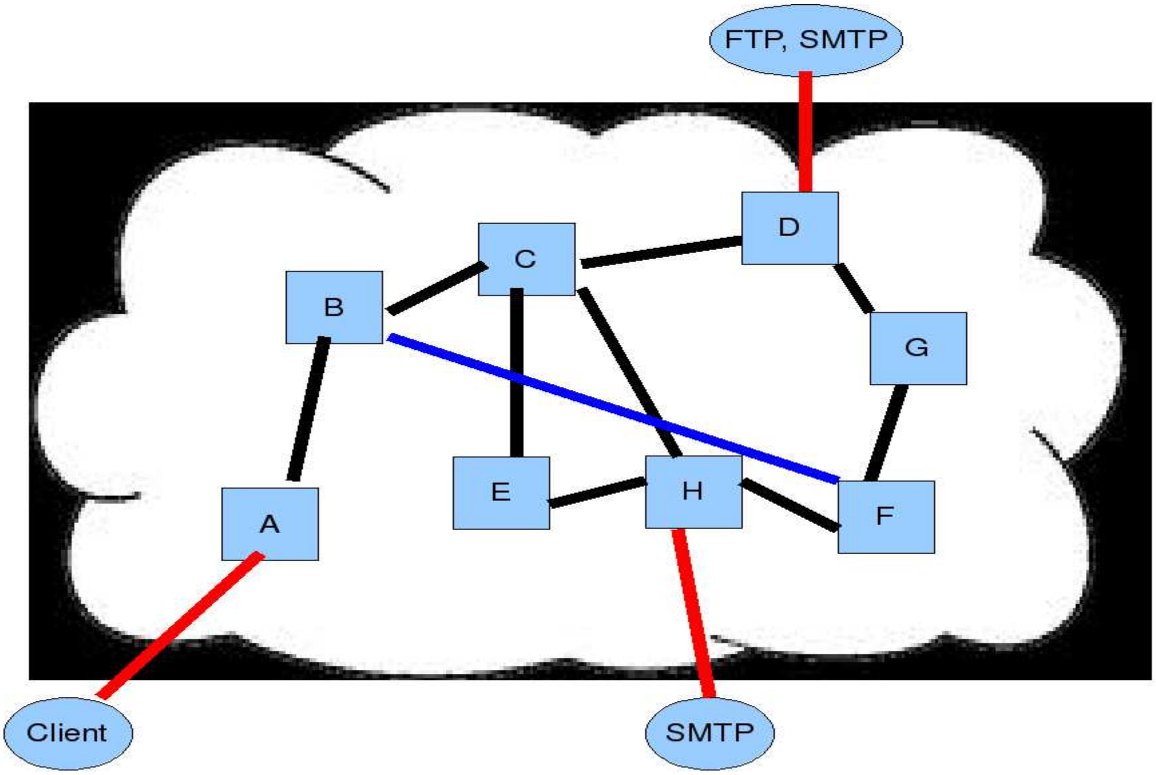


Figure 2.2: Example of an FTP request

The client wishes to download a file through an FTP request. This client needs to request such a service to the mesh (i.e., using its default gateway A). A then checks its routing table, finding that there is an FTP server 3 hops away from it (i.e., at D). Hence, A sends the request through a session with B, where B does the same check that A did. Then such request is transmitted to C and finally to D. At D the routing table specifies that the FTP server is connected to D's local network. At this moment the interface in the routing table is the IP Address of the server. To answer the request, the server only needs to transmit back to the session hop's source. The process is repeated at each node until it reaches the client.

At the moment of a request, a client specifies to its gateway the service of interest (see Figure 2.2). Afterwards the gateway verifies its routing table (see Table 2.1) and selects the server closest to it. In case of error (e.g., resource is not provided by selected server), the gateway searches on the alternative table for alternative servers. The alternative table is composed of the records on the routing table which are not the closest route. If no additional server is found, the error message is propagated back to the client. If the server provides the desired service, it sends it through the reverse path using the hop source of the received request (see Figure 2.3).

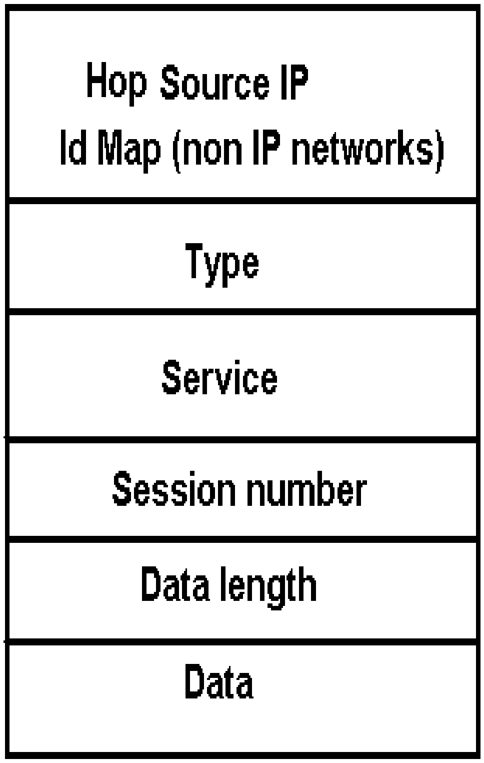


Figure 2.3: SORA's packet structure

In SORA only the hop's source address needs to be specified to return a request. For non IP networks, SORA creates a map from a technology to a multicast IP address; encapsulating the content of the packet structure with smaller maximum transmission unit inside of the data. The flow of the request to the server is decided upon the type and service fields. A session number is kept for encrypting and decrypting the data. The last two fields are the length of the data and the data itself.

SORA is ideal for service provision in heterogeneous networks, due of its flexibility with the service historical marking and service abstract access. However, a network using such a scheme is vulnerable to rogue servers and man-in-the-middle attacks. These issues are addressed in the next chapter, with a description of the proposed techniques to enhance SORA, which will be referred to as ESORA (Enhanced SORA). For the remainder of this chapter, we shall discuss the new non-security enhancements done to SORA to prepare it for ESORA, providing more reliability and adaptive behavior. These modification includes addition of routing metrics and link cost, as well as a unifying layered architecture to integrate different technologies.

Routing Metrics

In its original design, SORA only considered number of hops as routing metric to decide which is the best route to a specific service. In preparation for ESORA, we have considered network load and server reliability as metrics. Our ideal route should be closer in number of hops, with the least amount of network load, and highest chance of finding the content of interest (i.e., highest reliability).

Number of Hops

Ideally we want to minimize the number of hops, such that the request is handled faster (assuming no overload is on the server), and the network congestion is kept to a minimum. The number of hops is easily found upon service subscription, as the service broadcast hop count is incremented by one by every mesh router before sending.

Network Load

Finding up the network load has to be done in an efficient manner, i.e., avoiding flooding of information, as this process will be run continually. For such reason, we are limiting the network load to local knowledge only. Nodes running SORA keep track of the each individual route load inside the routing table. Every time a request is sent through a route, the load count is incremented; such counter is then decremented when a connection for the requested service is ended. The goal then becomes picking a route with the smallest load count.

Server Reliability

Once more our interest is to obtain a reliable, yet simple to compute metric. Just as the Network Load, the Server/Route Reliability is estimated locally as the percentage of requests that have been successfully handled by the server (i.e., no error or unsupported conditions). Such metric is very important because it allows us to use the best servers more often. However, this routing metric can also be dangerous as it can ruin a server's reputation with rouge clients asking for many unsatisfied requests. In order for this metric to work, we have added two more tables to SORA.

One of these tables is the content table, where the outcome of a request is logged, such that a route may get more or less importance for a given content. Such table is shown in Table 2.2, where a value of 0 means a server does not have the content, 1 means the server has the content. In the absence of a record in such table, meaning the server has never been asked for a specific content, a value of 0.5 is used instead. The last request column serves to know if a server has already been consulted recently for a resource, and in the case the outcome was a 1 it serves as a cache service also.

Table 2.2: Server Successful Requests Example

Server	**Service**	**Last Requests**	**Outcome**
172.16.3.1	21	profile.txt, contact.pag	1
172.16.3.5	80	index.html	1
172.16.3.5	21	favicon.ico	0

The second table (Table 2.3) is the number of unsuccessful request per client; clients with a very high value in such table are logged since they may indicate a malicious user. This table is reset every certain time ($1hour$ by default), such that blocked/logged clients can later retry to access the mesh. In the example shown, client IP $172.16.3.115$ will get block as a malicious user, as its number of failed requests is too high and account for most of its requests outcomes.

Table 2.3: Clients Reputation Example

Client	**Fail Count**	**Fail %**
172.16.3.115	159	78
172.16.3.34	2	3
172.16.3.59	5	10

Cost Metric Function

Now that the routing metrics have been defined, we need to combine these in such a manner that effective routing may be accomplish. A ranking function has been defined in terms of the number of hops h, the depth of the network δ, load factor λ, and the server reliability ρ. After some testing we arrived to Equation 2.1.

$$\rho_{server} = \begin{cases} 0.5 & \textit{no request made} \\ \frac{\sum outcome}{\#requests} & \textit{otherwise} \end{cases}$$

$$\lambda_{server} = \frac{\#pendings_{service}}{\sum_{i \epsilon servers} \#pendings_{i,service}}$$

$$\Delta = \frac{h}{\delta}$$

$$Rank(route) = \frac{\rho - \lambda}{\Delta} \tag{2.1}$$

This equation gives higher importance to servers with less requests pending and higher reliability. The number of hops is used simply as a metric to boost those routes which are logically closer. Furthermore, ranks with negative value reflect that the particular server is overloaded at the time (load is higher than reliability) and the route at the moment should not be selected. Thus, the route with the highest rank to handle the request is chosen as best. Alternatives are considered upon failure of the primary route. An example of the resulting routing table, with the new metrics, is shown in Table 2.4.

Table 2.4: New Routing Table Example

Service	**Interface**	**# Hops**	**# Pending**	**Reliability**	**Rank(realtime)**
25	172.16.2.1	2	1	0.4	0.4
21	172.16.3.1	2	6	0.9	0.3
25	172.20.1.1	2	4	0.7	-0.2
	Alternative	**Route**	**Section**		
21	172.16.3.1	4	2	0.5	0.25

The next section discusses the unifying architecture which will later be used for the content distribution system, part of ESORA, discussed in the next chapter.

A Unifying Architecture

The following section introduces the VESO-Mesh architecture, which incorporates components from Artificial Intelligence, Context-Awareness, Network Communication, and Green Energy Alternatives to create networks that can provide content anywhere. VESO-Mesh follows and combines the best practices from [9], [10], [11], [12], and [4], where the concepts of context, inference, sensing, etc. were described. The proposed layered model is shown in Figure 2.4. The individual layers are described in detailed in the following subsections, from bottom to top.

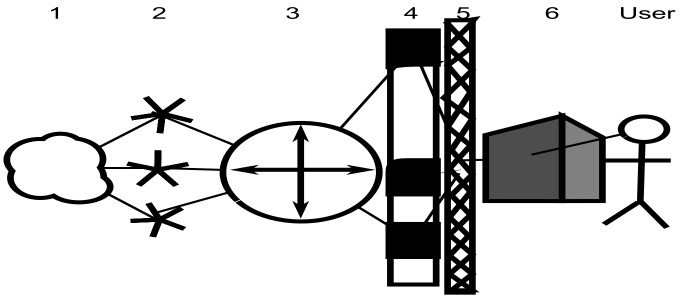


Figure 2.4: VESO-Mesh Layered Model Architecture

From left (physical world) to right (user) the layers of the architecture are presented: 1) Environment Layer, 2) Sensor Layer, 3) Technology Layer, 4) Service Layer, 5) Context Layer, and 6) Application Layer.

Environment Layer

As the first layer, the environment layer is composed of the physical world of interest. Such layer contains enumerated the physical variables to be monitored (e.g. temperature, humidity, pressure, illumination, etc.), which the VESO-Mesh must monitor and react accordingly. On this layer designers should concentrate also on energy sources (traditional and green), emergency sources such as battery blocks, and any other device required to power such alternative sources (e.g. inverters). A good design at the Environment Layer can help identify required components and increase the range extend the reach a heterogeneous wireless mesh network, to reach virtually any terrain.

Sensor Layer

The sensor layer acts as input/output device to access the environment layer. The specifications of such a layer should contain the number of sensors/actuators, along with their deployment configuration. Several factors such as: power consumption, maintainability, data rate, range, etc., must be considered when designing the sensor layer of a VESO-Mesh network. Furthermore, sensors and actuators may be further extend to any type of data acquisition device, including robots and security cameras. Processing the information monitored by the sensor layer is task of the Service Layer, which will be discussed in a moment. The interface defined to access the Sensor Layer is shown in Figure 2.5.

```
enum Action{
    READ,
    WRITE,
    UPDATE,
    SUSPEND,
    RESUME
};
Object reading();
boolean isActuator();
Object action(Action,Object[]);
String type();
```

Figure 2.5: Sensor Layer Interface

Sensors may be defined by implementing the $reading()$ interface. Actuators are a special type of sensor, which implement the $action()$ interface. Node status may be controlled through the $action()$ interface, passing the corresponding $Action$ (e.g. $READ$ senses, $WRITE$ uses actuator functionality). A human readable description is returned by the $type()$ interface.

Technology Layer

The task of the technology layer consists in providing the VESO-Mesh access to all devices and services (e.g., sensors, actuators, cameras, servers). Access should be transparent and with no range limit, hence support for multiple technologies (e.g., WiFi, Bluetooth, WiMAX, LTE, etc.) should be available. The different technologies should be bridged with the proper software and/or hardware. In [10], it was studied the alternative of wireless mesh routers as a solution to this problem. Wireless mesh routers are a good solution because it introduces very low infrastructure changes. Integration may be possible with service orientation at the routing layer using SORA (or ESORA once defined in the next chapter), which can be seen as analogous to the technology layer. The interface used for the technology layer is shown in Figure 2.6.

Moreover, the technology layer makes appropriate to include cognitive capabilities, because bridge nodes may have the responsibility of scanning the spectrum and configuring the different available technologies. Some of these mesh routers may be NETSIG nodes [5], containing signal processing capabilities to analyze the environment or process the sensor layer information. Such capabilities may be used for enhancing security concepts, such as privacy, and anonymity.

```
String[] verify();
boolean register(String,String);
subscribe(String,String);
Object access(String,String,Object[]);
```

Figure 2.6: Technology Layer Interface

The VESO-Mesh returns the list of available services through the $verify()$ interface. Servers and sensors may register their services by calling the $register()$ interface, while users may call $subscribe()$ for update on services, or $access()$ for one time access, giving the name of the service of interest to access.

Service Layer

A service is an abstraction of at least one sensor and/or actuators of the sensor layer. A service can also be a traditional service such FTP, in which case the flow goes to the traditional network upon arriving at the endpoint. The service layer provides users with a collection of service that can be performed by the network, without having to inform what type of technology is used for such service. This gives the user the idea it is a single and easy to use technology. Such a list is acquired by consulting the technology layer, via a call to the $verify()$ interface. The service layer also acts as a wrapper, providing users with human readable descriptions of the available services. For example, the $READ$ access of a temperature sensor would be *Get Temperature*, and in the same manner a multi-media server would have *Play Track*, rather than $WRITE$ which would be its access on the sensor layer. In a manner of speaking, the service layer is like the drivers in an operating system. The mapping is hence created and maintained at the technology layer, with elements of the sensor and service layers. The interface used for the service layer is shown in Figure 2.7.

```
enum Status{
    UNAVAILABLE,
    IDLE,
    BUSY,
    ERROR
};
String name();
Status status();
Object call(String ,Object[]);
String help();
```

Figure 2.7: Service Layer Interface

The status Enum returns the status of the services, which may be extracted from the technology layer. The $name()$ and $status()$ interfaces provide information about the service, while the help interface provides a list of possible commands for the service. For example, a thermostat could have *Get — Set Temperature* services, which would translate at the technology layer to the corresponding $READ$/$WRITE$ for its actuator. The interface $call()$ is used to access the service itself, which internally will call the $access()$ interface of the technology layer, to begin the mapping.

Context Layer

In the context layer the information is filtered, according to the context in which the network is in. A context is a group of activities associated to some factor. This activities could include social networking concepts. For example, in the case of a smart home, each context is represented by a user profile, in which for security reasons, children should not have access to appliances which may cause accidents such as the range in the kitchen. Adults should have full rights, as long as they are not intruders.

This context establishment can be done by access to speech or image data coming from the service layer. Thus, giving a high grade of selectivity through a natural interface. Other types of algorithms, such as learning user patterns or preferences and suggesting services based on these patterns/preferences are possible as well, due to the NETSIG capabilities.

The context layer hence, acts as a firewall to the application, where users will only view the functionality available to them, and hide other functionality. Any schedule of frequent tasks should be created and archived here. Just like the technology and service layer, the context layer also resides in the mesh NETSIG nodes.

Application Layer

The application layer is the final layer is in charge of creating front end graphical user interfaces, that facilitate the ease of network access. The applications are only able to see the content filtered from the context layer; i.e. only operations that users are able to select should appear.

The application layer is the only layer which resides in the end-user devices, allowing for a transparent integration with the TCP/IP technology stack. Chapter 4 provides a secure implementation for a SORA web layer, which takes advantage of the distributed nature of the network and the load balancing features of SORA.

SORA Security Needs

SORA when compared to other routing algorithms will find dynamically an optimal path from client to service, while preserving its load balancing nature. Such a path may change as the environment and topology changes; thus, SORA provides a compact and adaptive solution for the

IoT.

However, since SORA provides a SOA solution for delivering content at the network layer, the topology can be vulnerable as no security at each node is being assumed. Chapter 3 addresses this concern by implementing a novel and secure service discovery mechanism.

There are some solutions which currently exist for security at web services, such as CORS and OAuth. These solutions will be adopted and extended by SORA in Chapter 4.

CHAPTER 3: ENHANCING SORA WITH SECURITY

Over the past decades, wireless networks have been developed and deployed significantly, which enables more and more applications. For each application, different requirements (i.e. data rate, delay, storage, security, etc.) must be met. For such reason, researchers have dedicated to create different technologies and standards that meet optimally these requirements. These standards include: the IEEE802.11 (WiFi), the IEEE802.15 (including Bluetooth, Zigbee, and even Visual Light Communication), and IEEE802.16 (WiMAX). Such standards have been of most importance because of their low cost solutions and Wireless Mesh Networks (WMN) capabilities. WMN has made possible the integration of information processing and communication into every-day tasks; the deployment is easier and can have longer coverage range by using relay nodes.

Although WMNs are appropriate for certain type of applications, such integration cannot be achieved completely because the different technologies lack the capabilities to communicate among themselves. This communication is not possible because the traditional network architecture has focused on the connection of end points, rather than the service being provided. To address this issue, a Service Oriented Architecture (SOA) was proposed. The SOA relies on message containers to request and provide services throughout the network. Messages are independent of the service; services are loosely coupled to achieve higher re-usability. Service reuse makes SOA an attractive solution for the domains of: context networks, location-aware networks, and ubiquitous computing, which may lead to a different kind of solution called content-oriented networks. Content-Oriented Networks (CON) give importance to service provision, as well as location and context requirements.

In order for CON to be accepted as a valid solution, it must provide users with the necessary security. Failure to meet the security requirements will prevent wireless heterogeneous networks from ever reaching its peak. Security primarily concerns with the following aspects: (1) *confidentiality* to have only the requester and provider of a service access to the information, contained within the message, (2) *integrity* so that service content does not change as it travels from one end of the path to the other, (3) *availability* and (4) *authentication* to have services only accessible by legitimate users whenever they need to.

In [7], SOA networks requirements were analyzed for WiMAX technology; proposing that in order to have service orientation at the network layer which may leave to optimal content provision, a service control framework and unified routing algorithm must be implemented. In Chapter 2, we proposed a service control framework for Heterogeneous Wireless Mesh Networks, focusing on content provision under the name of VESO-Mesh. VESO-Mesh strongly relies on the existence of the unifying routing algorithm which has been designed with security being one of its primary criteria.

Based on such design guidelines mentioned in [7], we developed a Service-Oriented Routing Algorithm (SORA), also discussed in Chapter 2. In this chapter, we extend SORA for HWMN (Heterogeneous WMN) with three techniques to improve the confidentiality, integrity, and availability of services [13]. First we adopt a key management scheme from Heterogeneous Wireless Sensor Networks (HWSN), then implement a novel technique which we call *key shifting*, and finally develop node state-relations. The key management is used to provide encryption at the network layer, since MAC layer encryption does not protect confidentiality over a multi-hop scenario. Once the keys are distributed, the routing algorithm establishes secure sessions and enhances them by applying key shifting to reduce the probability of an attacker learning the information. Finally, the enhanced SORA (ESORA) uses the state-relations up trust among nodes, ensuring higher resilience, trust and integrity, and service availability.

The rest of the chapter is organized as follows. First we introduce a distribution system to securely address the computational requirements for a heterogeneous network. The security enhancements are then studied and explained. We then show the optimization problem of the key management configuration, and the results obtained from it and the different security enhancements presented. Finally, we conclude the chapter with some remarks.

Distribution System

In order to create effective routes for SORA, it is important to identify the provided content and preserve their integrity [14]. However, to accomplish such goals, content may not stay on their original locations, because some locations may not have high storage capacities (e.g. sensors), or may be compromised, loosing its data to some attacker.

To solve this issue, we introduce VESO-DRS (VESO Distributed Resource System). VESO-DRS is a distributed hash table approach (Figure 3.1). Such hash-table is composed of multiple keys, which help to access data, verify integrity, and validate updates (sub-section 3). The resource table may be accessed through an interface presented in sub-section 3.

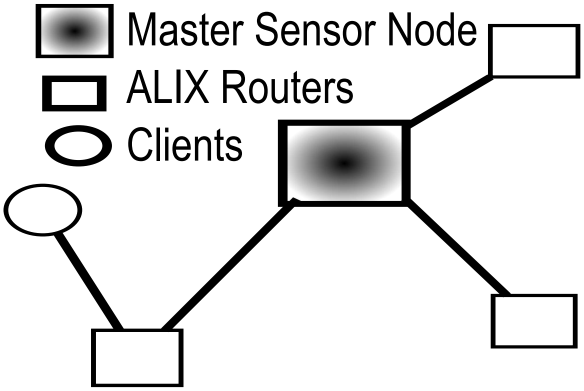


Figure 3.1: VESO-DRS Architecture

Resources are distributed across the mesh, stored only by those nodes which have the capability. Nodes without such capability, store their resources remotely. Resources may be queried unified, independently from location.

An additional indexing feature is added, to maintain privacy on sensitive information (sub-section 3). Sensitive information can be defined as: data of the resource which should only be accessed by some specific destination only (e.g. medical records, e-bills, military communication, etc.). As last component of VESO-DRS, a network wrapper is defined (sub-section 3), to transmit the VESO-DRS request to the different nodes in the mesh. Such a wrapper utilizes a compact approach, to be integrated in virtually any technology, and reduce the amount of information. In the next sub-sections, we discuss in greater detail each component of VESO-DRS.

VESO-DRS Interface

This sub-section presents the set of functions which form the VESO-DRS interface. The VESO-DRS interface is defined as follows:

1. **Join(TableName, AllocSize)** – used to join a node to the DRS specified by TableName. After this call returns true the requesting node will notify its neighbors that it has AllocSize

KB to store files locally.

2. **Leave(TableName) –** used to unsubscribe from the DRS. The content store must be passed to the DRS, unless the owner is the requesting node.

3. **Clear(Source <default=null>) –** clears all table or only the entries from Source.

4. **Remove(UpdateKey, ResourceName) –** removes from the DRS the resource given by its name. The UpdateKey is needed to ensure the user removing the DRS resource has rights to remove such resource.

5. **Put(ResourceName, Type, Content) –** adds to the DRS the content of the new resource; the resource file is inserted on the most appropriate node. This function returns false if the resource cannot be added, which is the case when the resource needs to be updated, otherwise True is returned.

6. **Update(UpdateKey, ResourceName, Content) –** updates the content of the ResourceName and UpdateKey specified; the update key serves to distinguish different resources with the same name, and validate authenticity of the update.

7. **Get(ResourceName) –** gets the resource from the DRS.

8. **Size() –** returns the number of KB stored in the node queried, the size is rounded to the nearest KB.

9. **Free() –** returns the number of KB free.

10. **Contains(ResourceName) –** checks if the system has a resource with the specified name.

11. **Share(ResourceName) –** asks the owner of the ResourceName to share its edit key with the requester.

The next sub-section describes the VERSO-DRS hash-table; such hash-table is used to query content from the mesh.

VESO-DRS Hashtable

To achieve complete functionality of the VESO-DRS, that is, resource integrity and easy access, VESO-DRS utilizes a hash table approach, in contrast to a file system approach while keeping the integrity of resources[15]. The structure of such table is shown in table 3.1. The value of the key K_1 is used as the main primary key of the table. To obtain K_1, the resource name is hashed using a hash function such as the MD5.

Table 3.1: VESO-DRS Indexing Table

K_1 **Hashed Resource Name**	K_2 **Hashed Content**	K_3 **Update Validation**	**Resource Content** **or Source Id**
md5	md5	long	bytes

The other two keys, K_2 and K_3, are used for preserving the integrity of the resource as a digital signature [16]. The value of K_2 is obtained as a hashed function of the content. When an attacker compromises a node or changes a resource, it needs to change correctly K_2, failing to do so will result in a detection of the corrupted resource [17], and the resource will be blocked to end users. Furthermore, any user or attacker who desires to make a valid update needs to know the value of K_3, which is a large integer randomly generated when the resource is added to the mesh.

Secure Indexing

There are times where data in a network must be kept private and viewable only to their destinations [18]. Failing to do so may result in learning about sensitive information such as medical records, military strategies,etc. Simple encryption however, provides the necessary privacy [19, 20], but makes it practically impossible to search the data within the network. This section discusses a scheme for storing encrypted data, and searching it to recover it.

Protecting and Recovering Information

To protect the data, the interface of VESO-DRS is enhanced with two additional functions, such functions allow the owner of a content to protect the data with a newly generated update key. The prototype for the two additional functions is defined as:

- **Protect(ResourceName, UpdateKey, AlgoName) –** Protects the resource ResourceName by encrypting it using UpdateKey as key for the algorithm named AlgoName. AlgoName may be any encryption algorithm supported by the mesh (e.g. 3-DES, AES, RSA, etc.). At the end, the function returns a newly generated updated key, which replaces the specified by the original put.
- **ProPut(ResourceName, Type, Content, AlgoName) –** adds to the DRS the content of the new resource; the resource file is inserted on the most appropriate node, and encrypted using the randomly generated update key and the specified algorithm. This function returns 0 if the resource cannot be added, or the update key otherwise.

When a get request is performed, data is returned to the requested encrypted by the selected algorithm and update key. Only users with knowledge of the algorithm and update key may access the information inside of the resource. If no such right exists, then requester may first ask to the owner to share its encryption parameters, using the mechanism described in later in 3.

Searching Information

Resources may contain encrypted or non encrypted information. To obtain the relevance of a resource with respect to the search of interest, the mesh needs to create the set $\hat{W}$ which is an encrypted search if the resource is encrypted, or plain-text if no encryption is used. The elements in $\hat{W}$ are the words in the search, encrypted as needed; each resource is then ranked according to Algorithm 1.

Line 2 of the algorithm states that for each search with keywords K, the first step is to transform all words into their stem form (i.e. singular, present tense, etc.), and remove the words which do not contribute to the search. Afterwards, the new keywords are stored in $\hat{W}$, and encrypted if necessary (line 3). In the next step (line 8), the algorithm creates a subset of matches M, which is created from the keywords that are found on each sentence.

Following this step, the algorithm computes a factor ι at lines 10–13, which is to give higher ranking to resources with keywords of greater importance. That is, users are most likely to specify as first keywords, the keywords that they find more relevant. For example, in the search 'bluetooth and zigbee wireless technologies', the user would most likely want to find information about bluetooth and zigbee, and not care much about the WiFi technology.

Then, the algorithm proceeds to find the distance between two adjacent keywords δ in M on lines 14–27. After all sentences are analyzed, the resource is ranked according to the expression in line 18, where the factor $\dfrac{|M|-1}{|\hat{W}|-1}$ serves as a penalty factor for those sentences which do not contain all the keywords. That is, if all the keywords are found in the sentence, then the factor becomes

1, and the sentence contribution just depends upon the distance between keywords (the closer the better contribution), and the importance of the keywords (most relevant in context should be as first keywords).

Algorithm 1 Resource Ranking Algorithm

```
Start with some user specified keywords $K$
$\hat{W} \leftarrow K$ in its stem form, without non informative words (e.g. conjunctions, prepositions articles, etc.)
if Resource Encrypted then
    $Encrypt(\hat{W})$
end if
for Each Resource $R$ do
    $Rank_R \leftarrow 0$
    for Each Sentence $S$ do
        Find match $M \subset \hat{W}$ in S
        if $M$ is not $null$ then
            $\iota \leftarrow 1$
            for Each keyword in $M$ do
                $\iota \leftarrow \iota \cdot (|\hat{W}| - indexofin(M_j, \hat{W}))$
            end for
            $\delta \leftarrow 0$
            for Each pair of consecutive words in $M$ do
                $\delta \leftarrow \delta + \frac{1}{|indexof(M_j) - indexof(M_{j+1})|}$
            end for
            $Rank_R \leftarrow Rank_R + \iota \cdot \delta \cdot \frac{|M|}{|\hat{W}|}$
        end if
    end for
end for
Sort resources in descending order with respect to $Rank_R$
```

Since sensitive information should be encrypted by the owner, the mesh needs to share limited information of these resources, such that authorized non-owners may obtain the resources. In the following sub-section we present the process on information sharing, by enhancing the interface previously described.

Sharing Information

As seen in the interface at sub-section 3, any user who wishes to get rights over a resource, may ask the owner for its update key. If data is encrypted, then users may also ask for the update key, when recovering some information not added by them. Hence, if the resource is encrypted, sharing requires that the owner gives the update key and encryption algorithm.

To protect this information, the network wrapper (sub-section 3) sends the request for the hashed resource name and a request id, the owner then checks its resources by hashing their name first and comparing against the requested resource name. Upon approval, the owner sends back the id that it received such that the requester may know to which resource it belongs, followed by the update key and encryption algorithm.

Network Wrapper

A network wrapper is needed to transfer the VESO-DRS requests across the mesh. Such a wrapper is shown in figure 3.2. The data of the request (e.g. resource name, update key, content, etc.) needs to be serialized into an array of bytes, before being inserted into the parameters field. Since most of the parameters are hash values, the wrapper's implementation knows a priory most of its length. For example, if an MD5 algorithm is used, the wrapper will know that the resource names will be fixed to 16 bytes; moreover, the update key and the hashed content for preserving integrity, are always of the same length as well. Hence, by subtracting the value of parameter length minus these fixed size parameters, the wrapper can find out the byte length of the content. This approach makes the wrapper compact enough, to be supported by many wireless technologies (e.g. wifi, bluetooth, zigbee).

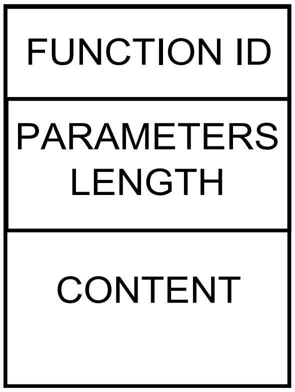


Figure 3.2: Wrapper Structure

A Function ID may take 1 byte, to identify which of the 10 functions of the interface is being accessed. Additionaly, two ID's are added for the REPLY from server, and CONTROL of the hashtable flooding. Moreover, only the data is needed, which is the only variable length field; its length is specified by the 2 bytes of parameters length.

In the following section, we describe the different security features which ESORA implements at the network topology.

Security at the Topology

A secure service-oriented network should posses at least 4 requirements [21]: acceptable delay, session keys, perfect forward secrecy, and anonymity. From [7], the control framework and unified routing algorithm can be added to these requirements. To achieve such requirements, ESORA adapts the following enhancements. First the algorithm employs a key management scheme to assign the keys accordingly to each node (sub-section 3). Then, these keys are used to create secure hop by hop sessions that allow confidentiality throughout the path (sub-section 3). The next step is to maintain periodic relationships among the nodes to monitor each other for detecting any compromised node (sub-section 3). The next subsections will discuss in detail how these three approaches are designed.

Key Management

There are many schemes for solving the key management problem, which could be considered when designing an HWMN. In this work, we adopt the scheme in [22], where Heterogeneous Wireless Sensor Networks (HWSN) consisting of 2 types of nodes was considered. We want to experiment how the concepts of heterogeneity in sensor networks can be adapted to mesh networks. In such approach, there are different classes of nodes ranging from 0 to N; the scheme states that classes with lower number have higher processing power, communication range, and storage space. Hence, class 0 is the most powerful, and as the number increases, the capabilities of nodes decrease. Each node stores K_c keys, where K_c is the number of keys in class c, from a key-pool of size S. These keys are pre-loaded into the nodes randomly without replacement, such that the K_c keys in one node are all different.

Nodes are able to communicate if at least they share K_{min} (i.e. $K_{shared} \geq K_{min}$) keys with their adjacent nodes, where K_{shared} is the number of keys shared by a node and its neighbors, and K_{min} is the minimal number of shared keys required between two nodes to communicate. The original work provides the entire mathematical model for the 2 class connection problem. Hence, here we define the probability of connectivity for a class 0 node connecting with a class c node. Such probability can be interpreted as the probability of having a secure bridge between technologies, and it can be defined as

$$p_{connectivity} = p(K_{shared} \geq K_{min}) = 1 - \sum_{i=0}^{K_{min}-1} p_{share[n]}(i) \tag{3.1}$$

where $p_{share[n]}$ can be defined as the probability that a class 0 node shares i different keys with the other n neighbors. The value of $p_{share[n]}$ can be calculated from equations 3.2. The conditional probability $h(i-j|j)$ states the probability of a node sharing an additional $(i-j)$ different keys, given that it already shares j different keys. Each node stores K_c keys from the total S available keys; the subscript c denotes the class number to which the node belongs (e.g. K_0 is the number of keys in class 0 nodes). The class number ranges from 0 to N, with N being the number of available technologies.

$$
\begin{aligned}
p_{share[n]}(i) &= \textstyle\sum_{j=0}^{i} p_{share[n-1]}(i)h(i-j|j), \textit{for}\, n > 1 \\
p_{share[1]}(i) &= \frac{\binom{S}{i}\binom{S-i}{K_0-i}\prod_{m=1}^{N}\binom{S-\sum_{h=1,h\neq m}^{N}K_h}{K_m-i}}{\prod_{m=0}^{N}\binom{S}{K_m}} \\
h(i-j|j) &= \frac{\binom{S-j}{i-j}\binom{S-i}{K_0-i}\prod_{m=1}^{N}\binom{S+i-\sum_{h=1,h\neq m}^{N}K_h}{K_m+j-i}}{\binom{S-j}{K_0-j}\prod_{m=1}^{N}\binom{S}{K_m}}
\end{aligned}
\tag{3.2}
$$

Note that if K_{min} is increased, it becomes harder for unauthorized users to have access to the network; however, the connectivity drops creating a trade-off that needs to be considered when designing the network. To improve this scenario, designers can add more nodes to the neighborhood, or increase the storage capacity of existing ones so that more keys can be stored. But under these circumstances, the cost of the network increases as well, along with a decrease in security because there are more nodes that can be compromised.

To adopt such a scheme, class 0 nodes are defined as the bridging nodes (i.e. nodes that have the capability to communicate all technologies). Class 0 nodes are the most powerful in terms of communication range, processing, and storage capabilities. Following class 0, there are N technologies. Class t ($1 \leq t \leq N$) can only communicate with nodes in the same technology t, or with a class 0 node. This assumption transforms our problem into the same 2 class problem seen in [22]; thus the same analysis is valid. Hence, the heterogeneity on an HWMN as proposed here is given by the technology types.

Depending on the importance of each factor, the network designer may neglect some of the trade-offs. Based on the configuration, the key-pool is selected and from it, the randomly chosen keys to be assigned at each node. The next section discusses how ESORA exploits the benefit of such key management scheme.

Key Shifting

ESORA encrypts information hop by hop throughout the network. To guarantee a secure path, ESORA utilizes a simple session handshake (see figure 3.3) per hop to obtain a common key for encryption. Such keys are previously distributed to each node as presented in sub-section 3. Thus, for each hop and session, a key is used as argument of the encryption algorithm.

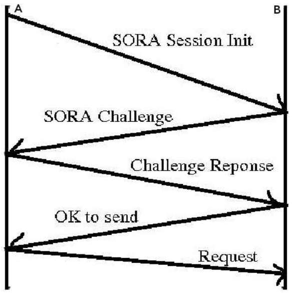


Figure 3.3: Simple Handshake, Session Key Establishment

When a source A wants to establish a session with an adjacent destination B, it sends a $SESSION\_INIT$. B sends a pre-defined message encrypted with one of its keys $K \epsilon K_B$. A tries to decrypt the message using the keys stored in K_A. Upon success (i.e. $K \epsilon K_A \cap K_B$) A sends a challenge response to B. Such a response is another pre-defined message encrypted with the key that A found. If the key that A chose is the same as B, B decodes the challenge response and sends the acknowledgment to A to begin the request.

Additional security is provided using a novel technique which we define as *key shifting*. Key shifting involves using an additional parameter to shift the session key K_s. Such a parameter is conveniently chosen as the number of hops to reach the destination server. Thus, the probability of an attacker finding the content of a message is given by equation 3.3, where δ is the depth of the mesh. For example, to find the selected key $K_{sel} = K_s + h$ the attacker needs to find out K_s (from session table) and h (from routing table). To illustrate the benefits of key shifting, even an algorithm with a probability of finding a key of 0.5, can be reduced by a factor for 0.33 (i.e. 0.167) with a small mesh of at most 3 hops. When receiving a response, nodes can decode the content using the session id to find out the required parameters.

$$p_{decode} = \frac{p_{algorithm}}{\delta} \tag{3.3}$$

Table 3.2: Example of Session Table

Session Id	**Session Key**	**Service**	**Client**	**Route Id**
(8*A + 4*B + 2*C + D)	**Session Id(128bits)**		**(A.B.C.D)**	
1447	3050	21	172.16.3.1	0 (Primary)
1448	3050	21	172.16.3.1	1 (Alternative table)

To implement encryption in ESORA, XTEA has been chosen. XTEA is a block extension of the Tiny Encryption Algorithm (TEA) demonstrated first in [23]. XTEA has a simple implementation and quick performance, allowing it to be implemented even in hardware. The suggested number of rounds for XTEA (i.e. 64) has been selected.

Table 3.2 presents an example of a session structure. The first column represents the mapping of the source hop, which is inserted into the packet to recover the client's identity later. If a created session id is already assigned, it is then increased by 1. Once the id is generated, it is inserted into the packet for answering the request later.

Additionally, to prevent timing attacks which have proven to be strong enough even against AES

[24], an extra random delay was added at the end of encryption. Hence any time relationship between two frames would have also included an additional phase shift of $K_{sel} + rand$, where K_{sel} is the selected key.

For adapting such a scheme, ESORA has two additional packet types: $SESSION\_START$ and $SESSION\_INVALID$. $SESSION\_START$ is sent by any kind of source node A that wishes to transmit a packet to a destination B. The moment a $SESSION\_START$ is received, the receiving node begins the security challenge as seen in figure 3.3. If the challenge is complete, both A and B will have created successfully a new session. If a session expires or is not completed, nodes are notified with the $SESSION\_INVALID$ packet type. The next section discusses the usage of node state-relations to increase the resilience, such that the HWMN is not lost if one node is compromised.

Node State Relations

ESORA creates node state-relations in order to monitor the nodes on the mesh frequently. Nodes form families, similar to the work presented in [25], with the difference that families are kept to a fixed size (i.e. up to 4 nodes), and the addition of a sleep policy. Moreover, nodes agree to follow some policies that allow each other to keep track of their well-being (i.e. non compromised state, low power, admin maintenance, etc.). The types of policies in ESORA are:

1. **Initial:** When new nodes are being authenticated on the network, they must associate to another node. Initially such a node doesn't exist, so the first node should pair temporarily with its default gateway. Once an extra node is added, the first couple is created. Subsequent nodes will become a child of such a couple, and later on get married to another node.

2. **Marriage:** Families of nodes are, at most composed, of three nodes. Such policy allows

easier management of the network, while avoiding flooding the network with monitor packages and reducing power consumption. To achieve this behavior, a family is broken when it reaches its fourth member, i.e. the newly added node will pair with the child of the selected family. After marriage, the child has two relationships to maintain, with at most 4 relatives (parents, couple, and child) to inform.

3. **Periodic:** Periodical updates are sent by each node to their relatives, to let know they are operating under normal conditions. Such updates are possible with Rijndel's algorithm, where the key scheduling allows the nodes know what value to expect from their relatives. If a member of the family fails to send such a message properly, the rest of the relatives will know that it has been compromised. Afterwards they can make the proper arrangements to notify the gateway of the event.

4. **Sleep:** At some point during their lifetime, nodes may decide to sleep. Such nodes can do so, only by continuing to monitoring their relatives, so that existent families are kept. A node must notify its relatives before going to sleep as part of the policy; the relatives will then agree on what to expect from the node if it decides to wake up. A node may decide to sleep if it is running low on power, or needs to be updated by an administrator. A node wakes up in the event that it needs a service from the mesh, or one of its relatives is compromised (i.e. periodic policy violation).

Optimal Key Distribution With PSO

The Particle Swarm Optimization (PSO) algorithm [26] has been chosen to find the best parameters, such that both connectivity and security are balanced. The PSO is ideal for such a task, due to its properties of moving the particles in a population (i.e. the parameters). A Java implementation of the PSO was implemented; PSO is a generational evolutionary algorithm which updates each particle in a population, according to their position (equation 3.4) and velocity (equation 3.5), where p_{id} indicates the particle current best value, and $pgbest_{id}$ denotes the best value among all particles. The value of x_{id} is the current particle (i.e. possible solution), and ω is the velocity's inertia weight, which decreases linearly from a maximum value (to explore search space), to a minimum value (to refine search).

$$x_{id} \leftarrow x_{id} + v_{id} \tag{3.4}$$

$$v_{id} \leftarrow \omega \cdot v_{id} + c_1 \cdot rand_1 \cdot (p_{id} - x_{id}) + c_2 \cdot rand_2 \cdot (pgbest_{id} - x_{id}) \tag{3.5}$$

The search space for the problem has $2 \cdot (N+1)$ dimensions; the 1 of the expression comes from the key-pool size parameter and N is for each pair of number of nodes in a class and their respective stored keys. The term $N + 1$ is multiplied by 2 to have the effect of K_{min} and K_i's parameters. Thus, the objective function for the be PSO can be expressed as equation 3.6, where the goal is to maximize the connectivity, while still having the highest K_{min} possible.

$$argmax_{(S_x, K_{minx}, N_{ix}, K_{ix})}\{P_{conn}(S_x, K_{minx}, N_{ix}, K_{ix})\} \tag{3.6}$$

In order to keep parameters in a practical range, minimum and maximum thresholds were set. That is, the key pool size was kept $S_x \leq S_{max}$, where S_{max} is the maximum key-pool size. In the same manner, the minimum keys required are kept $0 \leq K_{minx} \leq K_{mint}$, where $Kmint$ is a fixed threshold. Finally the number of nodes per class (N_{ix}) and the number of keys inside them (K_{ix})

are set such that equations 3.7 and 3.8 are satisfied, since higher class nodes have less capabilities.

$$N_i \leq N_{i+1}; i \geq 0 \tag{3.7}$$

$$K_i \geq K_{i+1}; i \geq 0 \tag{3.8}$$

Experimental Results AND Discussion

Resource Identification

This section presents the results obtained for the VESO-DRS system. Figure 3.4 denotes the process of write (put) and read (get) of the VESO-DRS. Through the interface data is access as if it were on local disk. That is, using service orientation, the user does not need to know about where the data is stored. In figure, storing a file on the table requires only to give the path of the file to be stored (i.e. content), and its name. Analogous, non-IP services such as a temperature sensor, may store its readings into a server to be analyzed, or when performing a get with the local coordinates, a near by temperature sensor may then provide the local temperature.

```
root@localhost# vesodrs put file1.txt

862590864108702 9248
root@localhost#
root@localhost# vesodrs get file1.txt

At: 10.43.42.2/file1.txt
Content: Hello World!
```

Figure 3.4: Example of Data Access

Data may be accessed from the network, as if it were stored locally on the machine, using the command line interface.

In figure 3.5, the process of detecting corrupted resources is shown. Since the update key is unknown, a compromised resource will have an invalid modification, and an violation in the checksum will be detected. At detection the mesh can stop the propagation of the resource to adjacent nodes; hence, preventing an attack to the network and possible compromise (e.g. botnet), independently of the technology type of the adjacent nodes. Further action on the compromised resource may be taken, such as deletion of the resource or move the resource to quarantine.

```
root@localhost# vesodrs get file1.txt

At: 10.43.42.2/file1.txt
Content: Hello World!
root@localhost#
root@localhost# vesodrs get file1.txt

At: 10.43.42.2/file1.txt
Error: Data has been compromised.
Check: a7c286cc198b7fb1c71cc0fcddbf807a
SCheck: b7886c8984a28557265d16b7ed16b87b
```

Figure 3.5: Example of Detection of Integrity

Invalid updates due to non-authorized modifications are detected with the checksum difference.

Moreover, the correct process for updating a resource is presented in figure 3.6. There are three components that must be available to finish the update: the resource name which will be hashed, the update key which is only known to users who are authorized to modify the resource, and the new content. A malicious user does not know the update key, and hence it cannot make a proper update, which in term will be detected as seen previously in figure 3.5.

```
root@localhost# vesodrs update file1.txt 12345
”This is only a test. File updated.”

Error: Invalid update key
root@localhost#
root@localhost# vesodrs update file1.txt
8625908641087029248 ”This is only a test.
File updated.”
root@localhost#
root@localhost# vesodrs get file1.txt

At: 10.43.42.2/file1.txt
Content: This is only a test. File updated.
```

Figure 3.6: Example of Resource Update

Valid updates are performed through the interface, by providing the proper update key as well.

In table 3.3, we present the ranking results for the phrase ’bluetooth and zigbee wireless technologies’, in 4 resources. Results suggest that the scheme is able to give a good idea of each resource relevance to the search, since the second resource ranked the highest, and the last resource ranked the lowest. Scaling the results to a complete document would be a matter of adding each individual sentence. For example, if we consider a resource having the first 3 phrases in table 3.3, the total ranking for that resource would be $1.5 + 11.11 + 1 = 13.61$.

Table 3.3: Ranking for ’bluetooth and zigbee wireless technologies’

Content	**Ranking**
Bluetooth provides higher throughput than zigbee	1.5
Bluetooth and zigbee are examples of small area wireless technologies	11.11
Wireless technologies have evolved to cover larger area	1
Technology advances have improved the cinema community	0.25

Security

During the first part of the experiment, we were interested to see how a change in the number of available keys with fixed topology, could affect the connectivity vs security trade-off. For such purpose, three configurations were used (see table 3.4). The experiments were run using one class 0 node (i.e. bridge node) with five adjacent class 1 nodes (e.g. Wi-Fi, Zigbee, Bluetooth).

Table 3.4: Configurations for First Experiment

Keypool Size	**Keys in class 1**	**Keys in class 2**
100	50	25
200	50	25
100	50	10

Figure 3.7 shows the relation of connectivity vs the minimum number of required keys. As expected, connectivity drops with the increase of K_{min}; however, the starting point of this decent depends upon the configuration.

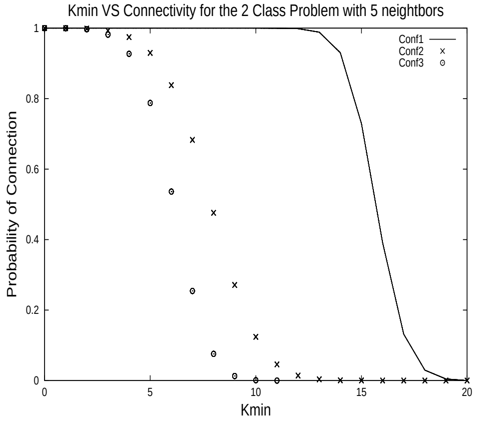


Figure 3.7: Effect of different configurations on connectivity and K_{min}

Among the 3 configurations, configuration 1 turned out to be the best. Thus, as the configuration changes the value of K_{min} can be increased, without causing a connectivity drop.

The second part of the experiment dealt with scaling the key management model; hence, the number of classes was increased gradually. The topology remained the same; that is, a class 0 node was surrounded by five nodes of each other class. Table 3.5 shows the case where more technologies are present. Depending on how many technologies are needed within ESORA to solve the domain, and how sensitive is the data being processed, users may select the proper configuration for the network, while achieving the desired connectivity.

Table 3.5: Key Distribution Performance

P_{conn}	K_{min}	S	K_0	K_1	K_2	K_3
1	0	N/A	N/A	N/A	N/A	N/A
0.9	15	100	50	25	N/A	N/A
0.95	10	100	50	25	20	N/A
0.9	20	100	30	25	20	10

Conclusions

In this chapter we design a secure service-oriented routing algorithm by extending the functionalities of SORA [3]. SORA can act as a unifying routing scheme to provide services in a HWMN. Three techniques were implemented: a key management scheme from HWSN, key shifting, and node relation monitoring. These techniques targeted providing confidential and reliable multi-hop path links.

With the implementation of ESORA, it becomes possible to deploy scalable and manageable HWMN, focusing in providing more content to clients. This approach leaves questions to be answered: how to provide error free environments, traffic control, and quality of service. We plan on following our research with error control and recovery scheme for ESORA.

CHAPTER 4: SECURING DISTRIBUTED SERVER ENDPOINTS

In previous chapters, we have discussed service provisioning from a network layer perspective. Security concerns were addressed such that a resilient backbone could be provided. This chapter addresses securing the endpoints of the provided services, i.e., the servers.

For the remainder of the chapter we shall talk about the proposed security model. Such a model relies on machine learning and statistical features, which may applied to any kind of service endpoint [27]. The model employs expiring token authorization to ease access to services, while keeping the service endpoints secure, as any token can be revoked if a client is detected to be compromised. Before a token is revoked the proposed model checks the authenticity of clients via question-like challenges which use user context information to be formulated.

In particular we cover in this chapter the use of web services due to their extensive presence in cyber space; however, the model can be use on any service type. Single Page Applications (SPA) and its benefits are presented afterwards as a foundation to our scheme. We then expose the security challenges that a SPA may have and how we address them. The later sections of the chapter describe the experiment we have used to test the feasibility of our approach, and the results that we have obtained respectively. Finally, the chapter is closed with some remarks and future directions for this research.

Web Services

Web services have come a long way since their first establishment. So far, the client-server architecture has proven to be successful in providing services. The popular paradigm of Client/Server web services has opened endless possibilities where information can be accessed from anywhere.

However, the traditional Client/Server paradigm demands strong servers specially under high demand systems. Such need open the server to possible denial of service attacks which may results in an extreme amount of losses depending on how much time a system's availability is affected. This scenario can be even more common on large enterprises, where the expected number of requests in seconds can be around thousands.

To address the demand needs, we propose a different approach to web services, a Single Page Application (SPA), with the goal of service provision. SPA technology relies more on the clients, providing each of them with all the resources they need to run the application, along with a JavaScript file that acts as a manual to tell clients how to run the application. With SPA technology, the server's load can be leveraged and distributed across its multiple clients; thus, increasing availability and resilience of the system as a whole.

However, such an approach demands a well designed security, given that to increase availability, the other two fundamental security principles (i.e., integrity and confidentiality) can be greatly sacrificed; this fact is primarily because the complete Application Programming Interface (API) is exposed to the client. For such a reason, a trust framework is built around REST services to achieve both signature and encryption, such that a system can have all main aspects from security.

The following sections cover the concepts of Single Page Applications and security measures employed with the proposed model.

Single Page Application

This section presents the paradigm under study. Traditionally web services have been delivered using the Client and Server model. Such a model consists of a client who queries for web pages from a server, typically using the HTTP protocol or equivalent. Under this traditional paradigm, the server is responsible for everything; meanwhile (see Fig. 4.1), the client side only sees the processed information. Such responsibilities on the server side take time and resources, which may be leveraged to the client side in order to support higher service loads.

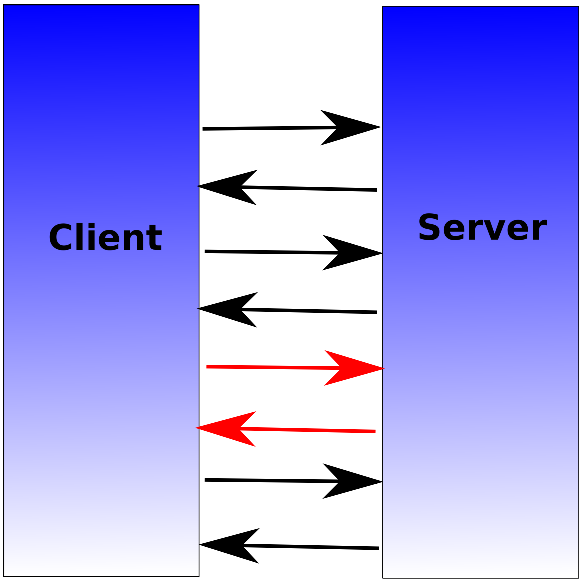


Figure 4.1: Traditional Client/Server Service Model

The black lines represent the navigation interaction of the user. As the user wishes to navigate the application, the server needs to send the different requested pages. Red arrows represent connections to obtain or modify data.

Alternatively, the Single Page Application paradigm relies more on the processing power which resides at the client side. Relying more on the client leaves the server with the sole responsibility of providing the service data. Moreover, the server can tolerate higher amount of clients as it does not need to handle non-server tasks (see Fig. 4.2). [28] describes SPA as: ”composed of individual components which can be updated/replaced independently”

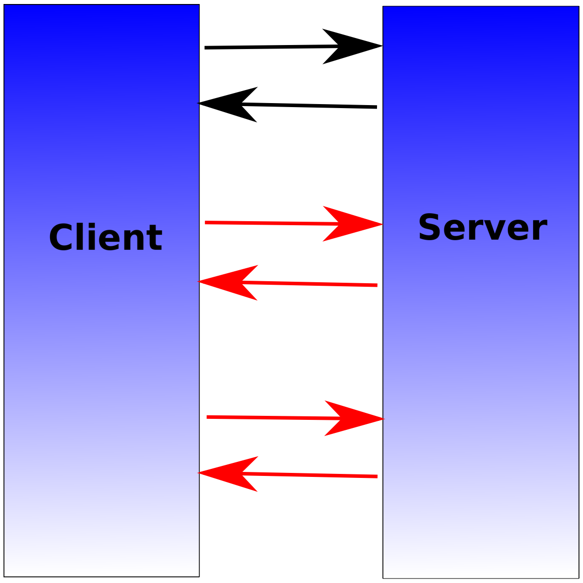


Figure 4.2: Single Page Application Service Model

The initial black line represents the query to obtain the index page, which the server replies with the complete application, and javascript, which acts as a manual. Subsequent red lines are the queries to obtain or modify data. Since the applicaton already has all required resources (aside from the services) loaded, then users can navigate through it as it was an offline application. Server concentrates more on providing content that will be shown on the application.

SPAs are convenient in terms of performance; however, when it comes to security, servers communicating with SPA need to be able to trust that the integrity of the application has not been compromised, and that its users are kept legitimate at all times. In the next section we elaborate on security measures appropriate for SPA, considering trust and privacy.

Security in a SPA

Security in Single Page Applications have to rely a lot on trust, as the client side is assigned with the application's API and free access to its services. This section covers approaches that, when combined, can provide a secure environment to SPA, such that it may be adapted to any scenario.

There has been successful attempts to provide a single sign on functionality, for web applications in the past, in particular for traditional client/server paradigm [29], and mobile devices [30].

First we adapt OAuth as a back service to our approach (subsection 4), taking advantage of its expiring tokens. When correct credentials are given to the OAuth provider, our service stores these credentials encrypted and distributed to use them in the future, when OAuth token is expired (subsection 4 . Upon token expiration, we check the legitimacy of the user by asking personalized reCaptchas, which are generated from user questions (subsection 4), these questions allows us to verify the user is still the same in the event that a malicious user gains access to the user credentials. Finally, our server gathers periodical anonymous and non-personal data from the usage of the SPAs, such that it can identify intruder behaviors and present the personalized reCaptcha before token expiration, upon suspicious user activity (subsection 4).

In the following subsections, we go into detail on how these separate components are implemented.

OAuth

OAuth is, according to the website [31]: ”An open protocol to allow secure authorization in a simple and standard method from web, mobile and desktop applications.” Simply put, OAuth provides tokens to applications so that they can act on behalf of users (see Fig. 4.3), without the need for their credentials. Moreover, OAuth allows for selective access of information for each token; i.e., one application may receive a token with certain rights, while another application may get another token with less rights. These tokens can also expire after a certain amount of time. OAuth is not the only type of token revocation system, in [32] a game model revocation system got proposed, which served in part as inspiration for enhancing OAuth. OAuth is still poses the advantage that it has many service providers.

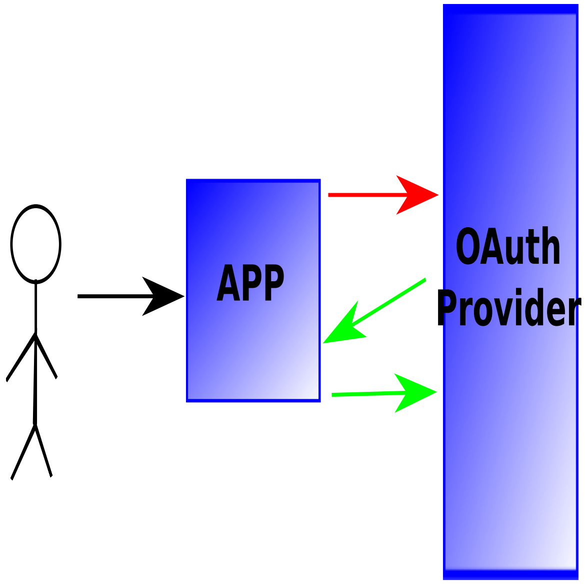


Figure 4.3: OAuth Authentication Model

OAuth's handshake is established by providing the user credentials to OAuth provider. Once the credentials are provided, a token is genereated and sent to the application that required authentication. Further authorization needs use the token in place of the user credentials.

At first glance, it may look as OAuth is the ultimate solution to solve the security requirements in SPAs. However, a malicious user can potentially gain access to the OAuth token, and thus, pose to be a legitimate user. [33] and [34] identified several attacks and medium which may be used for such attacks. Some of the attacks included token eavesdropping, token theft via XSS, impersonation, session swapping, and force login.

This is a problem exhibited in any web application, but it becomes larger when dealing with SPA as the whole application resides on the client side. OAuth tokens, still provide effective access control, for which part of our solution incorporates OAuth. Specifically, we chose OAuth 2.0; although it does not force SSL certificates at the client side it does have the token expiration feature which is a key component for this work. OAuth's functionality is enhanced by taking advantage of its expiring tokens, with secure and distributed storage of users credentials, and generated personalized reCaptchas.

Distributed Secure Credentials

In this subsection, we elaborate the design of securely and distributed storing the credentials of our clients inside of some nodes. Like in [35], the idea is distribute optimally the information, in this case the credentials, such that we can securely authorize the execution of API calls. This can by achieved by renewing expired tokens without the need to prompt the user credentials, as these or the application itself may have been compromised.

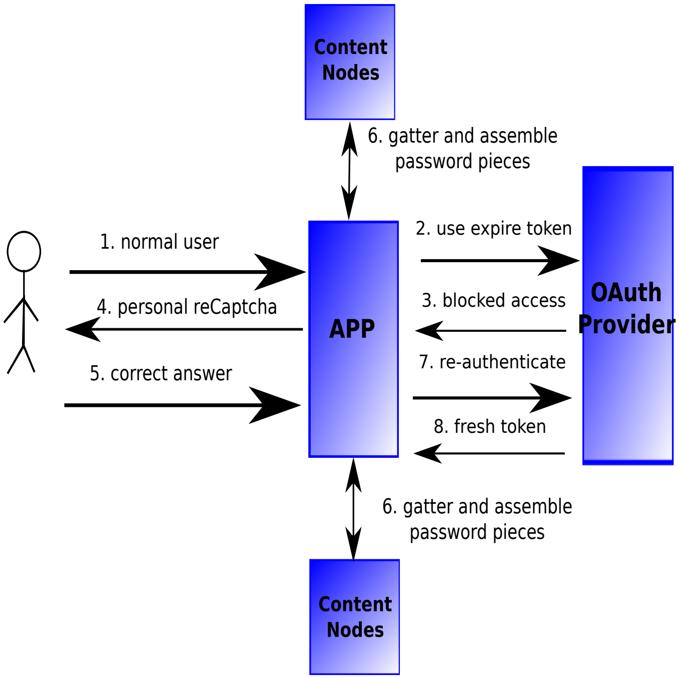


Figure 4.4: Token Renewal Process

This process is carried out when a token expired or if suspicious activity by a user is detected. The goal is to verify user's identity without the need for credentials.

The renewal process is shown in Fig. 4.4. The approach taken is to encrypt the password and distribute it across the content network nodes. For the secure distribution needed for step 6 we do the following:

1. Get the plain text password from the user
2. Mix the plain text password with known random characters at known locations to make the password of fixed length (e.g., 256 characters)
3. Encrypt the fixed length password created in the previous step
4. Slice the encrypted password into equal length segments
5. Distribute each segment to the content nodes
6. Nodes reply with a value which will be used to query later for the individual pieces of the password when decryption is needed

This approach achieves that no single node on the network knows anything about the password, aside from the central node, which only knows where the encrypted parts are stored. If either of the nodes gets compromised the only information that is seen is just part of an encrypted password. If the central node gets compromised, then only the location of the distributed password would be known, but schemes like the ones done in [36] for networks where service nodes join and leave, can be used to stop trusting this central node. However, no credentials can be found by just taking control of one node.

The next subsection discusses the personalized reCaptcha generation.

Trusting the Anonymous

In this subsection, we discuss the proposed scheme to trust the client apps. Such procedure must be done anonymously in order to preserve the client side privacy. The approach taken is from an application perspective; such an approach may be combined with network layer anonymous trust as shown in [37].

From the server's perspective, the goal is to ensure that the user is kept as the same across the token's lifetime. The reCaptcha technology has been created to test for human users; however it cannot test for legitimacy of the user.

Upon joining the app, we ask the user for questions that can generate personal reCaptchas (see Fig. 4.5). Such reCaptchas may be generated in a systematic manner, or using more natural approaches as the one proposed in [38]. Later, the OAuth token expiration feature can be used to periodically check if the user is still the same. Fig. 4.5 shows an example of a personalized reCaptcha, which includes preferences information to ask questions. These questions are easy to answer, but require knowledge that only the original user has. In this manner, we can re-authenticate the user, without having to ask for the credentials. Upon successful re-authentication we can use OAuth to regenerate the expired token. In order for the token renewal to work, we need to store the original credentials securely, so we can exchange it with the OAuth provider. We can securely store the credentials as discussed in the previous subsection. Therefore, a user's information contains $(username, personalized\_data, password\_nodes)$.

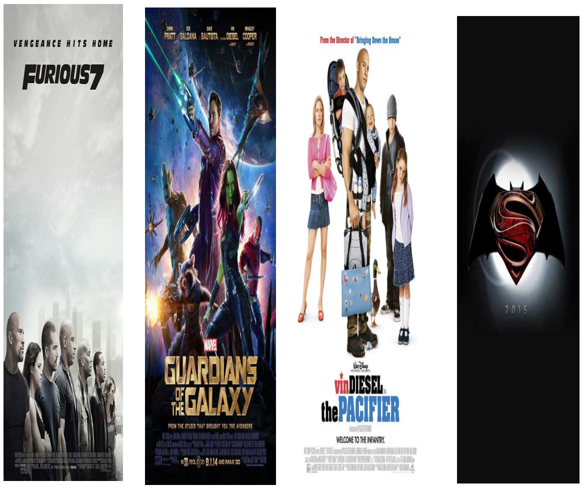


Figure 4.5: Personalized reCaptcha using movie posters

From user preferences many questions can be asked, the collection is only one of endless. Without knowing the context, we cannot be sure which the correct answer is. Possible questions, all with different answers: Which of these movies does contain not your favorite actor/actress? (Answer could be Vin Diesel) Which is not in your favorite movie type? (Answer could be Super Hero or Action movies) These would be generated if the user prefers a movie type or actor; therefore, user preference information is needed.

Using the proposed anonymous trust framework, we are able to include a personalized authentication method. The framework achieves enhancement of traditional password (something that the user knows) authentication, to include answers from user preferences, therefore adding personality (something that the user is) to the authentication method. Such enhancement can also be used to secure the password across the content network, such that we may use it to renew OAuth tokens when needed.

The following subsection introduces the final component of our proposed system, a centralized integrity checker server to detect compromised applications.

Periodical Integrity Checking

Having an application running on the client's browser completely means that the application's integrity may be compromised without much effort, using even tools that come packaged with the browser by default. Unfortunately, there is no way that this can be prevented in the event of a malicious user gaining access to our application. Such is not the case on traditional client/server applications, as other have shown that one can stop cross-site scripting with high effectiveness [39]. What we can do is detect when suspicious activity is taking place and block malicious users from accessing or modifying our content.

The SPA can check its integrity and report it periodically to the content server by sending non-personal information. Such information may include: a hash of the source code, the length in bytes of the source code, and last modified/last activity time stamps. Server can detect anomalies, by comparing against all running clients behavior. Anomalies (i.e., suspicious activity) is detected when the behavior is an outlier of the statistics for normal users. Statistical data may be combined with deep packet analysis for better results leading to protection against complex malware such as botnets [40].

When suspicious applications are detected, instead of blocking the application immediately, the personalized reCaptcha is sent to the application, such that the current user can be re-authenticated.

Experimental Setup

This section discusses the experimental setup established to implement and test the proposed approach.

For illustration purposes, we created a small HTML service on Java , using Spring MVC, which is on of the leading technologies for traditional client/server applications. We compared that implementation with the SPA equivalent, by increasing the number of simultaneous requests, using JMeter.

For the SPA implementation, the technology composing the MEAN stack has been employed (MongoDB, ExpressJS, AngularJS, and NodeJS). The client side is based on the AngularJS framework, which allows for HTTP interceptors where we can place our integrity checker service, which reads the file stats and content located in the $route.templateUrl$ JSON.

The server side is written in NodeJS, and contains REST endpoints which are supported by the ExpressJS framework. We have simulated the different content nodes for distributed secure credentials using different port numbers on each instance.

The stability of the system as a whole has been tested using a fuzzer that is available under Node's NPM manager. The fuzzer module outputs a 500 code when the server crashes, which leads to a vulnerability. The fuzzer module can also be used to put random string into the source code of the angular SPA. Lastly, a script has been created to simulate attacks onto the SPA, both source code modification as well as feeding malicious inputs.

Results and Discussion

The purpose of the first part of the experiment, has been to test if a SPA is indeed beneficial as a load distributions system. Using only the simple HTML server, we have obtained Figs. 4.6 to 4.8. On these figures we can appreciate that SPAs provide a more stable response time as the number of simultaneous request increases. This in some sense may reflect a natural robustness against naive Denial of Service attacks, as the load is leverage from the server and more responsibility falls under the client side.

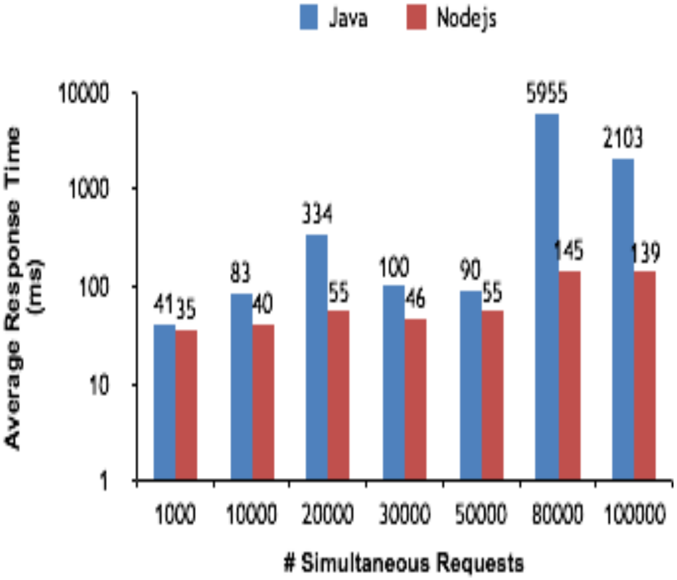


Figure 4.6: SPA vs Client Server Response Time Comparison

SPA's response time take longer to rise because they are more latency oriented. Heavy CPU dependent applications would still take more advantage of the traditional implementations, as Javascript is best for IO operation, not so much for CPU operations.

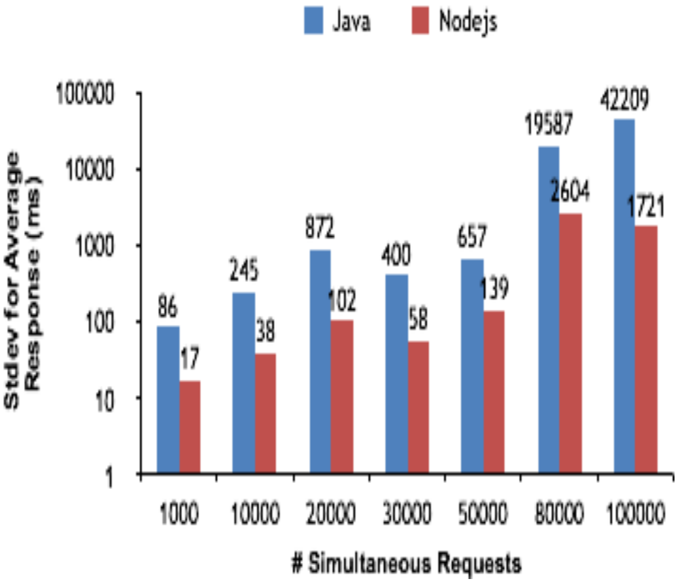


Figure 4.7: SPA vs Client Server Response Time Stability

SPAs have a smaller standard deviation as the number of requests increases. This means that the average response time will be close for all requests. Traditional applications on the other hand, when the volume increases, will have requests that take less time, some may take more time, and other may never be completed.

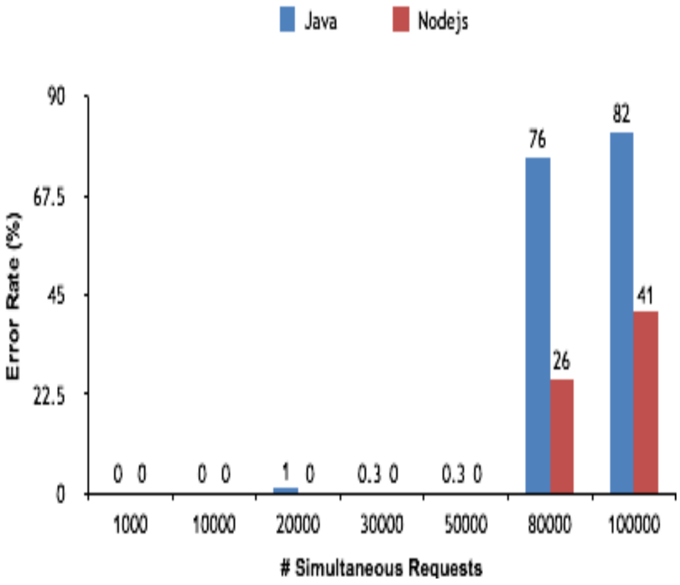
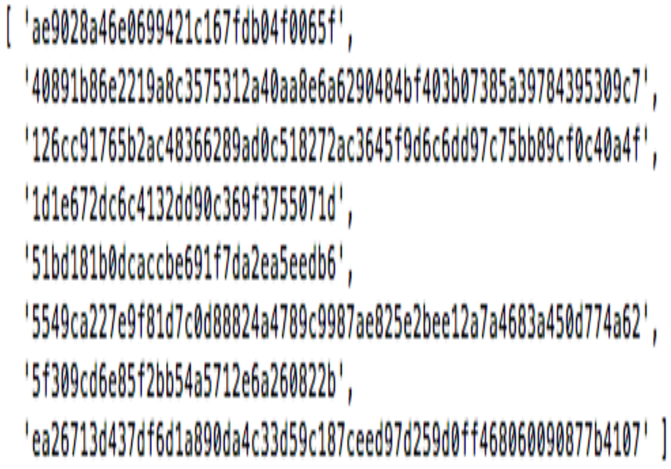


Figure 4.8: SPA vs Client Server Error Rates

For both types of applications, the error rate is practically none under low traffic. However, as traffic increases we see that the error rate is comparably smaller in SPA, this leads to less chances of crashing because of high load. Thus, stronger against Denil of Service attacks.

Now that we have seen that SPAs provide a performance advantage, the next experiment demonstrates how our proposed approach can secure the application across its life cycle. First, Fig. 4.9 shows how a simple password such as ”test1234” could be fragmented into 8 groups, each scattered across different nodes in the network. As seen in the figure, using AES128 and the randomized characters, the plain text password increase in size drastically; thus, combining the new size and the distribution of the password, there is no apparent correlation between the cipher and plain text password.

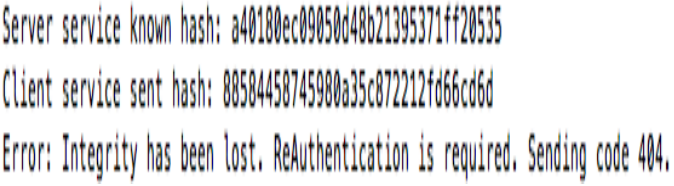


```
[ 'ae9028a46e0699421c167fdb04f0065f',
  '40891b86e2219a8c3575312a40aa8e6a6290484bf403b07385a39784395309c7',
  '126cc91765b2ac48366289ad0c518272ac3645f9d6c6dd97c75bb89cf0c40a4f',
  '1d1e672dc6c4132dd90c369f3755071d',
  '51bd181b0dcaccbe691f7da2ea5eedb6',
  '5549ca227e9f81d7c0d88824a4789c9987ae825e2bee12a7a4683a450d774a62',
  '5f309cd6e85f2bb54a5712e6a260822b',
  'ea26713d437df6d1a890da4c33d59c187ceed97d259d0ff4680600908 77b4107' ]
```

Figure 4.9: Distributed Password Encryption

Example of password generation for plain text ”test1234”. Each group is generated from an 8 character (real + randomized) password data. Result is 8 encrypted subpasswords which get distributed across the ndoes.

Fig. 4.10 shows how the server is able to detect when the application source code is changed, with the goal of disguising a malicious attempt as legitimate These types of modifications have been detected with a 100% accuracy.

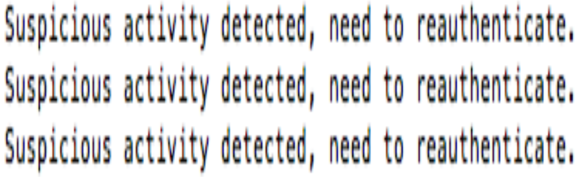


```
Server service known hash: a40180ec09050d48b21395371ff20535
Client service sent hash: 8858445874598 0a35c872212fd66cd6d
Error: Integrity has been lost. ReAuthentication is required. Sending code 404.
```

Figure 4.10: Compromised Integrity Detection

In the event of source code modification, the server can detect such modification with no effort required.

Finally, Fig. 4.11 shows the success when trying to detect suspicious activity. As seen in the figure, the server does manage to detect this type of attack, which normally will be in the form of the malicious inputs. The server detected this type of attack with a 95% accuracy. The type of features used for detecting this kind of modification only included the size of the $scope variable contents in bytes, which may promise better results if some other feature is used in combination (e.g., modification time interval).



```
Suspicious activity detected, need to reauthenticate.
Suspicious activity detected, need to reauthenticate.
Suspicious activity detected, need to reauthenticate.
```

Figure 4.11: Suspicious Acitivity Detection

Suspicious acitivity, or otherwise known as malicious modification to the input data, can also be detected with very high accuracy (95%).

The next section closes this chapter with some remarks and future work on the SPA security and performance)

Conclusion

This chapter proposes different approaches for security in SPA. When combined, each of these approaches have obtained a trust environment, where compromised applications can be detected, such that action may be taken upon it.

The personalized reCaptchas have shown to establish an easy credentials-free way to re-authenticate the user. When we combined this with the distributed credentials, we obtained a system that can take advantage of OAuth technology, while still using periodical authentication to ensure legitimacy of the user. Furthermore, having the password encrypted and sliced across different content nodes, gives resilience to the event that one of the nodes is compromised; under such an event, only one encrypted portion of the password would be known. By monitoring each other, nodes can detect compromised nodes, and thus, invalidate (i.e., require personalized reCaptcha) all users who have pieces of the password stored at the compromised machine.

Overall, the proposed framework has shown security across the multiple stages that the application could encounter, as well as performance improvements.

As future direction, we plan to look at the social engineering and how it may be used to enforce or crack this type of security, on SPA. This in time can help to establish more robust systems. We also plan to look at how this system performs against all the types of attacks shown in [33] and [34]. Finally, we will include an analysis of the CPU requirements and any stress test on the server side to see how much impact the security check has on the server side CPU.

CHAPTER 5: CONCLUSIONS AND FUTURE WORK

Conclusions

In this thesis, we evaluated pervasive content provision by constructing HWMN networks to achieve the IoT. HMWN provide means to achieve COTS solutions, which exploit the throughput, coverage, and delay of content provisioning. Furthermore, in this work we concentrated on establishing mechanisms for secure connections in such heterogeneous networks. With secure links HWMN can alleviate content provisioning on the server side, by incorporating client nodes as the content processing unit. Letting the client process its own content allows servers to concentrate more on providing the back-end services.

Less load on the servers imply more support for incoming connections as well as resilience to malicious or changing environments. To prevent malicious users from using the backbone, the servers can afford to employ advance security defenses such as distributed intrusion detection, as they are not processing content, only delivering content. The content network can adapt to changing environments, for example temporary server failure, since the clients processing the content can cache such content within their SPA, and retry uploading at a later time. SPAs also gave real time updates on content so that everyone learn of changes in content. This in turn allows for clients to become temporary service providers, as they can process the content and re-deliver it to adjacent networks.

Future Work

Secure HWMNs combined with a SPA front end showed promising results and possibilities. Now we must exploit the HWMN nature to provide content and services into different domains. To achieve such goal, one can explore how Artificial Intelligence and Parallel Computing can help optimizing the HWMN. As future projects to follow this thesis we can mention to formulate a mechanism to optimally deploy HWMNs, maximizing performance and security, while minimizing delays, power, and costs. Once an optimal configuration has been deploy, it would mandate to have an adaptive mechanism so the network can react to dynamic environments, providing fault tolerance with maximize performance. Finally, one can take advantage of the distributed nature of HWMN, to process content in parallel, optimizing resources and providing fault tolerance, as well as error detection and correction.

APPENDIX : BIOGRAPHICAL SKETCH

Hector M Lugo-Cordero was born in July 15 of 1983 at the city of San Juan, Puerto Rico. Hector is the son of Hector M Lugo-Santiago and Diana Cordero-Arias. He has a younger brother names Luis A. Lugo-Cordero and a younger sister named Liz Y. Lugo-Cordero.

On April 2006 Hector passed the Fundamentals of Engineering exam and then on October 2006 he passed the Principles and Practice of Engineering Exam.

In June of 2006 he received the tittle of Bachelor in Science in Computer Engineering with Magna Cum Laude (high honors) from the University of Puerto Rico Mayaguüez Campus. He then continued his studies on August 2006 at the University of Puerto Rico Mayaguüez Campus in the area of Computer Engineering with a specialty in Computer Networks under the supervision of Dr. Kejie Lu, from which he earned the tittle of Master in Science in Computer Engineering in June 2009.

Later he became a PhD candidate at the University of Central Florida under the advisement of Dr. Ratan K. Guha, once more concentrating in Computer Engineering in August 2009. He also decided in January 2012 to pursue a dual degree, pursing a Masters in Science in Computer Science.

After several internships at Disney, he became a Cast Member in May 2015 under the team of Level 2. Here he had the opportunity to present several projects to the company in the areas of application development, databases, data warehouse, productivity enhancement, and service availability.

His research interests include software development, computer networks, artificial intelligence, and green energy applications.

LIST OF REFERENCES

[1] I.F. Akyildiz and Xudong Wang. A survey on wireless mesh networks. *Communications Magazine, IEEE*, 43:S23–S30, 2005.

[2] X. Larrucea, G. Benguria, and S. Schuster. Mdsoa for achieving interoperability. *IEEE ICCBSS 2007*, 2007.

[3] H.M. Lugo-Cordero, Kejie Lu, D. Rodriguez, and S. Kota. A novel service-oriented routing algorithm for wireless mesh network. In *Military Communications Conference, 2008. MILCOM 2008. IEEE*, pages 1–6, 2008.

[4] A. Roy, S.K. Das, and K. Basu. A predictive framework for Location-Aware resource management in smart homes. *Mobile Computing, IEEE Transactions on*, 6(11):1270–1283, 2007.

[5] D. Rodriguez, K. Lu, and C. Aceros. Sirlab-netsig integration for environmental surveillance monitoring in wireless mesh sensor networks. In *Circuits and Systems (LASCAS), 2011 IEEE Second Latin American Symposium on*, pages 1–4, Feb. 2011.

[6] Hector M Lugo-Cordero. Wireless mesh networks: Service oriented design and channel optimization. Master's thesis, University of Puerto Rico at Mayaguez, 11 2008.

[7] Kejie Lu, Yi Qian, and Hsiao-Hwa Chen. Wireless broadband access: Wimax and beyond - a secure and service-oriented network control framework for wimax networks. *Communications Magazine, IEEE*, 45(5):124 –130, may 2007.

[8] M. Mamei and F. Zambonelli. Programming pervasive and mobile computing applications: The tota approach. *ACM Transactions on Software Engineering and Methodology*, 18(4), 2009.

[9] V. Ricquebourg, D. Menga, B. Marhic, L. Delahoche, D. Durand, and C. Loge. Service oriented architecture for context perception based on heterogeneous sensors network. In *IEEE Industrial Electronics, IECON 2006 - 32nd Annual Conference on*, pages 4557–4562, 2006.

[10] J.M. Reyes Alamo and J. Wong. Service-oriented middleware for smart home applications. In *Wireless Hive Networks Conference, 2008. WHNC 2008. IEEE*, pages 1–4, 2008.

[11] Chun-Yu Chen, Yu-Ping Tsoul, Shu-Chen Liao, and Cheng-Ting Lin. Implementing the design of smart home and achieving energy conservation. In *Industrial Informatics, 2009. INDIN 2009. 7th IEEE International Conference on*, pages 273–276, 2009.

[12] Tinghuai Ma, Yong-Deak Kim, Qiang Ma, Meili Tang, and Weican Zhou. Context-aware implementation based on CBR for smart home. In *Wireless And Mobile Computing, Networking And Communications, 2005. (WiMob'2005), IEEE International Conference on*, volume 4, pages 112–115 Vol. 4, 2005.

[13] H.M. Lugo-Cordero, R.K. Guha, and Kejie Lu. A secure service-oriented routing algorithm for heterogeneous wireless mesh networks. In *Global Telecommunications Conference (GLOBECOM 2011), 2011 IEEE*, pages 1–5, Dec 2011.

[14] Hector M. Lugo-Cordero, Ratan K. Guha, Kejie Lu, and Domingo Rodriguez. Secure service distribution for versatile service-oriented wireless mesh networks. *2013 8th International Conference on Malicious and Unwanted Software: "The Americas" (MALWARE)*, 0:88–94, 2011.

[15] Jie Lin, Yunliang Zhang, Zhiyong Tan, and Yiqi Dai. A novel secure distributed disk system. In *Advanced Communication Technology, 2009. ICACT 2009. 11th International Conference on*, volume 02, pages 1476 –1481, feb. 2009.

[16] T. Werner, C. Fuchs, E. Gerhards-Padilla, and P. Martini. Nebula - generating syntactical network intrusion signatures. In *Malicious and Unwanted Software (MALWARE), 2009 4th International Conference on*, pages 31 –38, oct. 2009.

[17] Ronghua Tian, R. Islam, L. Batten, and S. Versteeg. Differentiating malware from cleanware using behavioural analysis. In *Malicious and Unwanted Software (MALWARE), 2010 5th International Conference on*, pages 23 –30, oct. 2010.

[18] T.C. Chieu, Thao Nguyen, and Liangzhao Zeng. Secure search of private documents in an enterprise content management system. In *e-Business Engineering, 2007. ICEBE 2007. IEEE International Conference on*, pages 105 –112, oct. 2007.

[19] Yong Zhang, Wei-Xin Li, and Xia-Mu Niu. Secure cipher index over encrypted character data in database. In *Machine Learning and Cybernetics, 2008 International Conference on*, volume 2, pages 1111 –1116, july 2008.

[20] A. Singh, M. Srivatsa, and Ling Liu. Efficient and secure search of enterprise file systems. In *Web Services, 2007. ICWS 2007. IEEE International Conference on*, pages 18 –25, july 2007.

[21] Abdelkader H. Ouda, David S. Allison, and Miriam A. M. Capretz. Security protocols in service-oriented architecture. In *Proceedings of the 2010 6th World Congress on Services*, SERVICES '10, pages 185–186, Washington, DC, USA, 2010. IEEE Computer Society.

[22] Yi Qian, Kejie Lu, Bo Rong, Hector Lugo, and David Tipper. An optimal key management scheme for wireless sensor networks. In *Military Communications Conference, 2007. MILCOM 2007. IEEE*, pages 1 –6, oct. 2007.

[23] David Wheeler and Roger Needham. Tea, a tiny encryption algorithm. pages 97–110. Springer-Verlag, 1995.

[24] Daniel J. Bernstein. Cache-timing attacks on aes. Technical report, 2005.

[25] Xiaodong Lin. Cat: Building couples to early detect node compromise attack in wireless sensor networks. In *Global Telecommunications Conference, 2009. GLOBECOM 2009. IEEE*, pages 1 –6, 30 2009-dec. 4 2009.

[26] J. Kennedy and R. Eberhart. Particle swarm optimization. volume 4, pages 1942–1948, nov. 1995.

[27] H.M. Lugo-Cordero and R.K. Guha. A secured distribution of server loads to clients. In *Military Communications Conference, 2015. MILCOM 2015. IEEE*, Oct 2015.

[28] A. Mesbah and A. van Deursen. Migrating multi-page web applications to single-page ajax interfaces. In *Software Maintenance and Reengineering, 2007. CSMR '07. 11th European Conference on*, pages 181–190, March 2007.

[29] M. Hillenbrand, J. Gotze, J. Muller, and P. Muller. A single sign-on framework for web-services-based distributed applications. In *Telecommunications, 2005. ConTEL 2005. Proceedings of the 8th International Conference on*, volume 1, pages 273–279, June 2005.

[30] Gianluigi Me, D. Pirro, and R. Sarrecchia. A mobile based approach to strong authentication on web. In *Computing in the Global Information Technology, 2006. ICCGI '06. International Multi-Conference on*, pages 67–67, Aug 2006.

[31] OAuth community site. `http://oauth.net`. Accessed: 2015-05-04.

[32] M. Jebalia, A. Ben Letaifa, M. Hamdi, and S. Tabbane. A revocation game model for secure cloud storage. In *High Performance Computing Simulation (HPCS), 2014 International Conference on*, pages 1016–1017, July 2014.

[33] Sun San-Tsai. Simple but not secure: An empirical security analysis of oauth 2.0-based single sign-on systems. In *Proceedings of the ACL 2012 System Demonstrations*.

[34] Sun San-Tsai and Konstantin Beznosov. The devil is in the (implementation) details: An empirical analysis of oauth sso systems. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 378–390. ACM, 2012.

[35] Yulai Xie, K.-K. Muniswamy-Reddy, Dan Feng, D.D.E. Long, Y. Kang, Zhongying Niu, and Zhipeng Tan. Design and evaluation of oasis: An active storage framework based on t10 osd standard. In *Mass Storage Systems and Technologies (MSST), 2011 IEEE 27th Symposium on*, pages 1–12, May 2011.

[36] S. Abedinzadeh and S. Sadaoui. Agent trust management based on human plausible reasoning: Application to web search. In *Privacy, Security, Risk and Trust (PASSAT), 2012 International Conference on and 2012 International Confernece on Social Computing (SocialCom)*, pages 760–765, Sept 2012.

[37] A. Matos, S. Sargento, and Rui L. Aguiar. Waypoint routing: A network layer privacy framework. In *Global Telecommunications Conference (GLOBECOM 2011), 2011 IEEE*, pages 1–6, Dec 2011.

[38] Rafael E. Banchs and Haizhou Li. Iris: A chat-oriented dialogue system based on the vector space model. In *Proceedings of the ACL 2012 System Demonstrations*, ACL '12, pages 37–42, Stroudsburg, PA, USA, 2012. Association for Computational Linguistics.

[39] J. Shanmugam and M. Ponnavaikko. A solution to block cross site scripting vulnerabilities based on service oriented architecture. In *Computer and Information Science, 2007. ICIS 2007. 6th IEEE/ACIS International Conference on*, pages 861–866, July 2007.

[40] D. Smallwood and A. Vance. Intrusion analysis with deep packet inspection: Increasing efficiency of packet based investigations. In *Cloud and Service Computing (CSC), 2011 International Conference on*, pages 342–347, Dec 2011.