Design and Implementation of RESTful Non-repudiation Services

Article *in* Journal of Computing and Information Science in Engineering · January 2012

CITATIONS
0

READS
1,507

1 author:

Saman A. Barakat
University of Duhok
4 PUBLICATIONS **29** CITATIONS

SEE PROFILE

ISSN 1746-7659, England, UK
Journal of Information and Computing Science
Vol. 8, No. 3, 2013, *pp*.183-208

Design and Implementation of RESTful Non-repudiation Services

Saman Barakat
Department of Computer Sciences, University of Zakho,
Duhok, Kurdistan-Region, Iraq
saman.barakat@gmail.com

(*Received October 30, 2012, accepted April 4, 2013*)

Abstract. Security issues can be a barrier to make successful online businesses because the Internet can make critical information vulnerable [1, 2]. Non-repudiation is a security feature that is related to integrity and authenticity [3]. Providing non-repudiation for online communication is a key factor to achieve a successful electronic business [2]. Non-repudiation should ensure that each involvement in an online interaction cannot be denied. Besides, non-repudiation, fairness between the parties also plays an important role to achieve successful electronic businesses. One solution used to achieve fair non-repudiation services is by using a trusted third party (TTP) that implements fair non-repudiation protocols [4]. This project uses work that has been done by Cook et al in 2006 [5] as a starting point, which was a non-repudiation service project that uses SOAP web service technology. However, this project aims to implement non-repudiation services using Representational State Transfer (REST) architecture style principles in order to obtain significant advantages that REST technology provides such as scalability and simplicity.

Keywords: *Non-repudiation services, Non-repudiation protocols, Fair exchange protocols, Resful web services, REST.*.

1. Introduction

The Internet has become one of the major ways of conducting business. It makes accessing new and larger business easier. It also enables integration between different businesses and services using different types of programming middleware and web services easier [6]. Web services have been used to implement and integrate web applications. Web applications might be e-commerce business, banking service, e-voting, electronic trading, digital contract exchanging, and so on [3]. As a result, many businesses have migrated to the Internet taking advantage of its efficiency and low cost [2].

Security concerns can be an impediment to achieve successful online business since online transactions across insecure environment such as the Internet can make valuable information vulnerable [1, 2]. Consequently, security issues such as confidentiality, integrity, and authenticity have been studied in order to establish successful online businesses [3].

An aspect related to integrity and authenticity is non-repudiation [3]. Providing non-repudiation for online communication is a key factor in building a successful electronic business [2]. Non-repudiation should ensure that each involvement in an online interaction cannot be denied. For instance, if Alice and Bob are two parties that want to make an online transaction or exchange an electronic document, neither Alice nor Bob can deny their involvement in the exchange. Therefore, the non-repudiation of origin should be generated for the receiver and the non-repudiation of the receipt should be generated for the originator or sender [3].

In addition to non-repudiation, fairness between the parties also plays an important role in establishing successful electronic businesses. Fairness between participants within online businesses means that each online transaction between participants must be fair at the end of the transaction under any circumstances. For example, if Alice and Bob have been involved in an electronic transaction either both obtain what they expect or neither of them gains any advantage over the other.

One possible solution for non-repudiation and fairness issues is to use non-repudiation services. Non-repudiation services provide electronic evidence such as digital signatures entitled to all parties involved in online transactions [2]. Non-repudiation services implement fair non-repudiation protocols to guarantee both non-repudiation for the actions and fairness for each party involved in a particular transaction [3]. The

accountability of participating in a transaction can be guaranteed by using the two types of non-repudiation evidences which are non-repudiation of origin (NRO) and non-repudiation of receipt (NRR) [1]. The NRO is digital evidence generated by the originator and entitled to the recipient; while the NRR is digital evidence generated by the recipient and entitled to the originator.

One approach used to achieve fair non-repudiation services is by using a trusted third party (TTP) that implements fair non-repudiation protocols [4]. The trusted third parties have three main types; inline TTP, online TTP and offline TTP [2, 3]. These types of TTPs will be discussed in details in section 2.

This project uses work that has been done by Cook et al in 2006 [5] as a starting point, which was a non-repudiation service project that uses soap web service technology. However, this project aims to implement a non-repudiation service using Representational State Transfer (REST) architecture style principles in order to obtain significant advantages that REST technology provides such as scalability and simplicity.

Representational State Transfer (REST) is an architecture style for distributed web applications introduced in 2000 by Roy Fielding in his doctoral dissertation [10]. REST architecture style deals with data on the web service as resources and uses Uniform Resource Identifiers (URIs) to access these resources which is a unique identifier for each resource on the web [11-13]. The REST architectural style introduces several roles and constrains for distributed web applications.

The most significant benefit of designing the system as a RESTful web service is scalability. By applying REST constraints, the system will be highly scalable. First, by decoupling the client side concerns from server side concern, the developers do not need to worry about the server implementation only they need to focus on the client side and this can help clients to integrate the system with their business quickly. Secondly, the scalability of the system increased by applying the hierarchy constraint [10]. In this case, the system can be scaled by adding intermediaries to the system in order to balance the load between machines on the network.

Another advantage of using a RESTful web service is simplicity. Clients' requests will be stateless [10]; therefore, the system does not require managing and maintaining request state across multiple client requests. Moreover, request messages are fully descriptive and can be understood by intermediaries and the ultimate service. REST will be discussed in depth in section 2.

The aim of this project is to design and implement RESTful Non-Repudiation Services. It will implement three types of fair exchange protocols; inline, online and offline protocol. The interactions are between only two clients; Alice and Bob. Therefore, in order to achieve this aim, the project is divided into several objectives as follows:

1. Implement Inline Protocol: The first objective is to design the project using inline TTP protocol [2, 3, 7] for two clients Alice and Bob, in order to understand the basic functionality of Non-Repudiation Services and the interactions between clients.

2. Implement Online and Offline Protocols: The second objective is to implement both online TTP and offline TTP protocols [2, 3]. The main reason of implementing online and offline protocol is to reduce trusted third party's responsibility and move it to clients in which clients will do verification of signatures as well as storing information. The other reason of implementing online and offline protocol is to show the scalability of the Non-Repudiation
Services to adapt to different protocols.

3. Using Proxies: The next objective is to use proxies in the system [14-16]. The main reason of using proxies is to execute non-repudiation protocols instead of clients. In addition, the proxy will keep clients away from implementation details such as signing, encryption and decryption [5]. The other advantage of using proxies is to provide more security in the system by using SSL within insecure communications.

4. Testing and Evaluating the System: The final objective to complete the project is to provide testing and evaluation of the project. Testing section is necessary in order to show that every method is working properly. The evaluation is to evaluate the overall project and show whether the project has achieved its main aim or not.

By achieving the aim and objectives, the project will provide clients the flexibility to choose between different types of non-repudiation protocols. In addition, the project will provide REST properties such as scalability and simplicity.

1.1. Dissertation Structure

The structure of the rest of this dissertation is as follows:

- The background will be discussed in section 2. It will aim to introduce what is non-repudiation and fairness in the interactions between parties. Also, it will discuss REST Style and its constraints. Finally, it will present some of the related non-repudiation web services.
- Section 3 will discuss the overall design of the Non-Repudiation Services and define different components and services used in the system.
- Section 4 will discuss the implementation of the Non-Repudiation Services.
- Section 5 will introduce the testing strategy used for testing protocols used in the project.
- Section 6 will evaluate the performance of each protocol used in the project. In addition, it will show the advantages of implementing the Non-Repudiation Services as RESTful.
- Section 7 will state some future work in order to improve the project.
- Section 8 will state conclusions corresponding to the project.

2. Backgrounds and Related Work

This section contains a detailed background about non-repudiation and fairness of interactions between clients. Also, it includes techniques used to achieve fairness between participants. The next part of this section is dedicated to REST style and will state REST constraints and show what are advantages and disadvantages of applying them. The final part of the section will be used to present some of the related nonrepudiation webservices.

2.1 Non-repudiation and Fairness

The main purpose of non-repudiation is to maintain irrefutable evidence related to online interactions between clients [3]. It produces and collects evidence at the time of interaction between parties. Moreover, it prevents a misbehaving party to deny a transaction that has been done between the misbehaving and another party. Therefore, it should ensure that each involvement in online interaction cannot be denied. In order to illustrate the non-repudiation and fairness and show their effects, two scenarios will be discussed. Firstly, exchanging of a document without using fair non-repudiation protocol. Then the exchange will be discussed in term of non-repudiation and fairness using Coffey and Saidha protocol.

Consider Alice and Bob are two clients that want to make an online transaction such as signing and exchange an electronic contract. They agreed in advance about the protocol for exchanging the contract. In this exchange, Bob expects a document with sender's signature while Alice expects a receipt from Bob. The normal case of exchanging the protocol, as can be seen from diagram 1, might be as follows:

1- Alice signs the document and sends it via a network to Bob.

2- Bob, in return, signs the document and sends it to Alice.

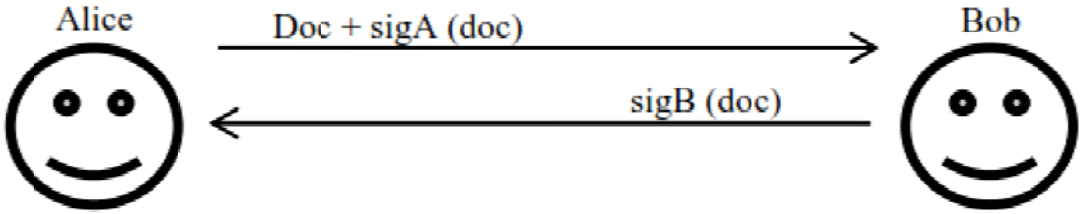

Figure 1. Exchange a document without using fair non-repudiation protocols.

Such weak protocol might cause a number of security issues. In the first step, when Bob obtains the document with Alice's signature, he has an advantage over Alice since he has a signed document from Alice and Alice does not have his signature yet. Therefore, if Bob is not honest, he can cheat Alice firstly by denying that he has received the document from Alice or simply he can just ignore her. In this case, Alice will face a selective receipt problem.

Secondly, he might use the received contract to earn better contract. For example, he might go to another client and convince him to sign a better contract with him.

This weak exchange protocol provides both participants the ability to cheat each other. In the case of

dispute, they do not have valid evidence to preserve their rights. In these kinds of online exchanges, the most crucial things are non-repudiation and fairness.

Non-repudiation is to provide parties irrefutable evidence in the case of dispute and fairness in order to forbid parties to obtain advantages over each other. Therefore, to achieve a non-repudiation, Bob needs to obtain the document with evidence proving that Alice is the owner of the document and also, Alice obtains irrefutable evidence shows that Bob has received the document. Fairness also is an important aspect of online transaction. Fairness means neither Alice nor Bob gain an advantage over each other. In the previous example, Bob always had an advantage over Alice because she was the initiator. Fair exchange protocols are used to preserve participants' rights in online interactions. Coffey and Saidha introduced a protocol to provide non-repudiation and fairness [7]. The protocol requires a trusted third party (TTP) to provide fairness between participants.

To illustrate how the protocol works, consider the following example. Alice wants to send a document with a non-repudiation of origin (NRO) to Bob and Bob returns to Alice a non-repudiation of receipt (NRR) to Alice (see figure 2).

The following notation is used to describe Coffey and Saidha protocol:

M = document.

H = hash function

NRO = SigA(H(M))

NRR = SigB(H(NRO))

The main interactions between Alice and Bob using Coffey and Saidha inline protocol are as follows:

1- A → TTP: M, NRO

2- TTP → B: H (NRO)

3- B → TTP: NRR

4- TTP → B: M, NRO

5- TTP → A : NRR

In step one, Alice sends the document with a non-repudiation of origin (NRO) to the trusted third party. In step two, TTP sends a hash of the NRO to Bob H (NRO). Then at step three, Bob signs the H(NRO) and sends NRR to TTP in order to be able to obtain the document. Then Bob can obtain the document with its NRO at the same time Alice can obtain no-repudiation of receipt (NRR).

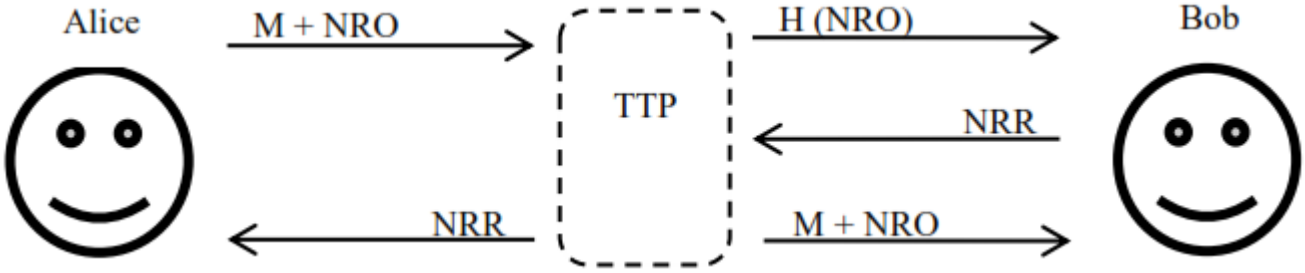

Figure 2. Exchanging a document using Coffey and Saidha protocol.

Four types of fairness in online exchanges have been defined: *weak*, *strong*, *true* and *probabilistic* [3]. The weak fairness indicates that the sender does not receive its evidence, while the receiver obtains its evidence from the sender, and the sender only obtains a proof about this fact. In the strong fairness, it must ensure that the sender gets a non-repudiation of receipt and the receiver obtains the document and the non-repudiation of origin at the end of the transaction, or none of them gains any extra information. The third type of fairness is *true fairness*. The protocol provides *true fairness* if and only if it supplies strong fairness. On top of that, the produced evidence are independent of the protocol execution which means that it cannot be known whether the TTP has involved in the protocol execution or not [3]. *Probabilistic fairness* presumes that a protocol is fair, if at the end of the action the probability of the strong fairness between the sender and the receiver, is very close to one [3].

Different approaches have been used to achieve fair exchange between parties. The first solution used to achieve fair exchange was to use *gradual exchange of expected information* between parties [3]. This solution was unrealistic in the real life because it was required that each party must had the same power of computation as well as there was a large amount of transmissions for each interaction. An improvement has

been introduced with other types of protocols called *Probabilistic protocols* [3]. In these types of protocols, the equivalent in computing power was not required anymore; however, they still had issues with large transmissions. It is called Probabilistic because it only gave Probabilistic guarantee of fairness. Another approach has used to achieve fair exchanges, which is deterministic guarantee, offered by trusted third party (TTP) that implements fair non-repudiation protocols [4]. The trusted third parties have three main types inline TTP, online TTP and offline TTP [2, 3]. The first one is inline TTP, which was presented by Coffey and Saidha [2, 7]. It is a trusted third party that works as a delivery agent; because, it is involved in each message transmission between senders and receivers. The second type is online TTP [2, 3, 8]. In this solution, TTP does not take part in all message transmissions between parties, but it might take part in some required requests such as exchanging secret keys or abort operation. The third type is offline TTP [2, 3, 9]. This approach presumes that well behaved players will lead to a successful transaction without any interaction from the TTP. Offline TTP is involved in a transaction between parties in the case of failure, or if one of the parties either wants to abort the exchange operation, or resolve the exchange. The TTP is involved in the transaction in order to make sure that all parties either obtain what they want, or none of them gain any advantage as a result of any failure.

2.2 Representation State Transfer (REST)

Roy Fielding 2000 [10] introduced in his doctoral dissertation Representation State Transfer (REST), which is an architecture style for distributed web applications. REST is a hybrid architecture style driven from sets of network based architecture style which was discussed in his dissertation in Section 3. REST style handles with information on the web as resources. These resources are uniquely identified using Uniform Resource Identifier (URI). Therefore, these resources can be accessed using Uniform Resource Locator (URL). REST architecture style introduces sets of constraints such as client-server, stateless, cache, uniform interface, layered system and code on demand. The main goals of REST architecture are "*scalability of component interactions, generality of interfaces, independent deployment of components, and intermediary components to reduce interaction latency, enforce security, and encapsulate legacy systems"* [10].

Resource: Any data that can be saved on the web and can be represented as a stream of bits is considered a resource. In other words, any concept that users might access using the Web is considered resources. For instance, users, documents, photos, videos, or information retrieved as a result of computing a particular query are regarded as resources in addition to that, functions or methods that do a specific computation are also considered resources [10, 17].

Client-Server: The first constraint applied to REST architectural style is a client server constraint [10]. The main purpose of this constraint is to decouple the interests in client side and server side programming. By applying client-server constraint to the system, the concern about the user interface and data storage is distinguished. As a result, it enables the user interface to be portable between many platforms. For instance, the system is implemented using Java and uses a web browser as an interface that allows users to make requests to resources; however, the client who wants to use the system might use different platforms instead of a web browser to make a request. In addition, it improves the scalability of the system by reducing the complexity in the server side.

Stateless: Stateless constraint is added to client server communications [10]. This constraint presumes that the interactions between clients and server must be stateless in which, any request issued by clients must have all necessary information to serve that request in the server side, and without taking benefit from stored context on the server.

Richardson and Ruby [17] have distinguished between two kinds of states which are application state and resource state. Application state is the state that is desirable to keep out of the server. The server only deals with application state when the client makes a particular request. This means whenever the clients called the server, they must include all required information to complete the request. Consider the search engines as an example; search engine such as google.com needs to pass all necessary parameters in order to serve the request. The server deals with the client's application state only in the case of making requests. Meanwhile, resource states are useful data that are stored on the server side such as photos, videos and necessary documents of users. The resource state must not be kept out of server because it will make the system inefficient. For instance, facebook.com allows its users to upload photos and videos to their accounts

and share them with other friends; however, it would be inefficient system if it does not allow keeping pictures within server.

Stateless constraint has a number of advantages and trade-off. The most significant advantages produced by stateless constraint are visibility, reliability and scalability [10]. Visibility of the system is improved because the system is not required to look deeply into each request generated by a client in order to understand the request. Reliability is enhanced hence the task of recovering from partial failures becomes easier. Scalability is enhanced as a consequence of not saving request's state on the server side, and leads the server to free resources quickly. Furthermore, it reduces the complexity of the implementation on the server side because managing resources between requests are not required by the server.

The main downside of this constraint is that it might reduce the performance of the network due to increase the sending of repeated contextual information in a series of requests [10]. For example, whenever a client wants to make an exchange of a document, he always needs to send the sender and the receiver's id as well as the document's id with the request in order to be processed.

Cache: Cache constraint is added to the hybrid style to improve the efficiency of the network [10]. It requires that information within a replay for each request be labeled as cacheable or non-cacheable impeccably or explicitly.

The essential advantage of using cache constraint in REST architectural style is to decrease some interactions between clients and server, improve the network efficiency, scalability, and reduce latency for requests.

The design trade-off can be noticed in the reliability, in which it decreases the reliability of the system because cached data might be different from source data returned from the request.

Uniform Interface: Uniform Interface in REST architecture style between components is a key factor that distinguishes the REST style from other Internet based style [10]. REST uses basic methods provided by Hypertext Transfer Protocol (HTTP) for most requests and they are GET, PUT, POST and Delete. Firstly, GET method is used to retrieve information from resources. Secondly, PUT method is used to create or update existing resources. The POST method is used to create a resource. Finally, Delete method is used to delete an existing resource.

The uniform interface in REST does not indicate which method is the best to use, but it indicates that all RESTful web services should use these methods in the same way. In other words, it does not presume that GET is the best HTTP method for retrieving information but it says that any time that GET method is used it means "read" [17].

HTTP methods have two properties; safe and idempotent [18]. As can be seen from Table 1 GET, HEAD and OPTION methods are safe and idempotent; while PUT and DELETE methods are idempotent but not safe. The final HTTP method is POST which is neither safe nor idempotent. Safe methods mean that the methods do not have any unintended effects on the resources if they are implemented correctly [18]. Users can make requests with these methods without concern about undesired changes to the resources.

Safe methods must be implemented as read only in order to ensure that they do not have a side effect while clients use them. For instance, when a client makes a GET request to a search engine, the request does not change or modify the resources. He only reads information from resources. At the same time different clients might make the same request without affecting one another.

Idempotent methods indicate that a request can be called more than once and will produce the same result [18]. PUT and DELETE might be requested more than one time without considering about unwanted effects; therefore, Idempotent property is important in the case of the network failure. For example, a client wants to delete a picture from a resource, the client should use DELETE request to delete the picture. In the case that the request could not reach the server, she/he can make the request again. Meanwhile, if the picture has been deleted and the client makes the request again to delete the picture, she/he will gain the same result which is the picture is deleted from the resource.

By applying this constraint to the system, the system as a whole is simplified and communication visibility is enhanced [10]. Moreover, independent development is supported since the implementations are decoupled from the servers. The disadvantage of uniform interface is the efficacy, because transferring of the data is in standardized form instead of the form which is needed by the application. In order to gain uniform interface, multiple architecture constrains are required to guide the behavior of the components.

Table 1. Properties of HTTP methods.

Methods	Safe?	Idempotent?
GET	YES	YES
OPTION	YES	YES
HEAD	YES	YES
DELETE	NO	YES
PUT	NO	YES
POST	NO	NO

The four constraints that define REST are identification of resources; manipulation of resources through representations; self-descriptive messages; and, hypermedia as the engine of application state.

1. Uniform Resource Identifiers (URI): Each resource should be addressed uniquely with a Uniform Resource Identifier that makes the resource accessible through the web using URL. The URI should be unique for the resource which means it is not allowed to have more than one resource with the same address in order to avoid ambiguity. However, the resource may have one or more than one address [17]; because, it makes maintaining and manipulating of resources easier for the clients. For example, if a company produces mobile phones and presents them as phones resource, this resource might be accessed using different URIs such as product id www.example.com/phones/id or can be accessed by name www.example.com/phones/name.

2. **Representations:** Representation is some information about the state of a resource [17]. The information can be retrieved from the server or can be sent to the server to create a new resource. Different clients might manipulate the same resource in different representations [19]. For example, a resource might be represented as HTML format, JSON or XML and so on.

3. Self-Descriptive Messages: RESTful requests must be descriptive. Descriptive means that each method is used according to its verb. For instance, GET is used for retrieving information only. Moreover, the requests must contain enough information to describe how the request should be processed. It must be clear to understand by the server [10, 17]. For example, if a customer wants to know a list of all phones produced by the company between 2000 and 2012, then the request must include all required information to process the request upon these details for example www.example.com/phones/2000/2012.

4. Hypermedia as The Engine of Application State (HATEOAS): HATEOAS is a constraint of the REST application architecture style that distinguishes it from the most other web-based application architectures. It means that the server provides extra information in representation state that enables clients to change the state of application to another state [20]. The information is dynamic hypermedia links, which tell the clients how to change the application state to another one only by following links. In other words, resource of web service should be connected to each other so that it enables clients to navigate between resources using hypermedia [17]. Search engines are good examples of HATEOAS. When searching for a particular subject within search engines, it will provide a representation state that contains links related to that subject. These representations enable clients to change the application state to another state. The client is not required to know details about the service; he only needs to follow the links. The server might provide the HATEOAS by inserting hypermedia links into JSON object, XML document or HTML document [17].

Layered System: Layered system is added to REST style to improve internet scale requirements [10]. The layered system style enables architecture to be designed in hierarchical layers. By arranging components in a way that each layer is communicated to one another and cannot see further than that layer. The layered system might improve scalability by using intermediaries in the system to balance the load between multiple servers. The primary disadvantage of layered systems is that they add overhead and latency to the processing of requests. For example, implementing proxy provides hierarchy architecture in which the end user sends his requests to the proxy and the proxy forward the request to appropriate resources**.**

Code-on-Demand: The final constraint added to REST style is code-on-demand [10]. It allows client code to be downloaded and executed as a script or an applet. This helps the clients by decreasing some necessary pre-implementation. It is good if one knows that all clients within same organization use Java.

2.1. Related Non-repudiation Web Services

Cook et al in 2006 [5] introduced a flexible framework for a fair non repudiation service and it was implemented using web services technology. Two types of the fair non repudiation protocols have been used in the framework; however, the service can adapt different types of protocols. The protocols were used in the framework were driven from Coffey and Saidha protocol with some additional modifications.

Three main modifications have been done to the protocol. Firstly, providing the sender with evidence of submission to the trusted third party (TTP) as well as providing a non-repudiation of validation. Finally, exception handling is supported during the executing of the sub-protocol. The first protocol is further modified to support *light-weight end user*. In this protocol the TTP is in charge of the verification of signatures and long term storage for documents, signatures and receipts. The second protocol uses *light weight DA*, which lets the users take the responsibility of the long term storage and verification of evidence and decrease the load on the *DA*.

Another non-repudiation framework has been presented in 2008 by Peiris et al [21] for e-government application in Sri Lank. In this approach, they concentrate on two types of non-repudiation evidence which are the non-repudiation delivery and non-repudiation of origin. However, their approach does not support fairness in the interaction between the originator and receiver.

3. System Design

This section will describe the overall system architecture and defines all the components used in the system. There are three main parts in Non-Repudiation Services which are trusted third party service, timestamp and verification service and client. Each of these components' implementations will be explained in details during the next section according to the security section, data storage and their functions. A Non-repudiation Services have a set of resources in each service. This section will show how the resources and Uniform Resource Identifiers are designed and shows how the main interactions between services for each protocol are used.

3.1. Non-repudiation Services Architecture

Non-Repudiation Services comprise of three main parts as can be seen from figure 3; trusted third party, clients and timestamp and verification service. Clients are divided into two types of clients who are sender (Alice) and receiver (Bob).

Non-Repudiation System provides three types of fair non-repudiation exchange protocols; inline fair exchange protocol introduced by Coffey and Saidha [7], online fair exchange protocol introduced by Zhou and Gollman [8] and offline fair non-repudiation exchange which was introduced by Asokan, Shoup, and Waidner [9]. However, the system can adapt different protocols easily because each of these protocols has been implemented as RESTful resource and any other protocol can be implemented without effecting existing protocols.

In order to make sure that the clients can use the system easily and make the execution of protocols is clear to understand, proxy server has been implemented in clients' service. The main reason of implementing the proxy is to make the communications between end users simple and easy by executing relevant protocols instead of clients. The proxy is responsible for signing and security in the interaction between clients and the end users are not aware of how the encryption has been done or which cryptography algorithm has been used. The only thing the end users need to do is making requests through the REST API. For example, when Alice sends a document to TTP via inline protocol to be sent later to another client, she makes a post request to her proxy with the correct URL. The proxy will retrieve the document from her database and encrypt it then send it to TTP.

The timestamp and verification service is included in the system as can be seen from figure 3. This service is used by the trusted third party as well as clients. The main reason of this service is to provide timestamp evidence and also verify signatures.

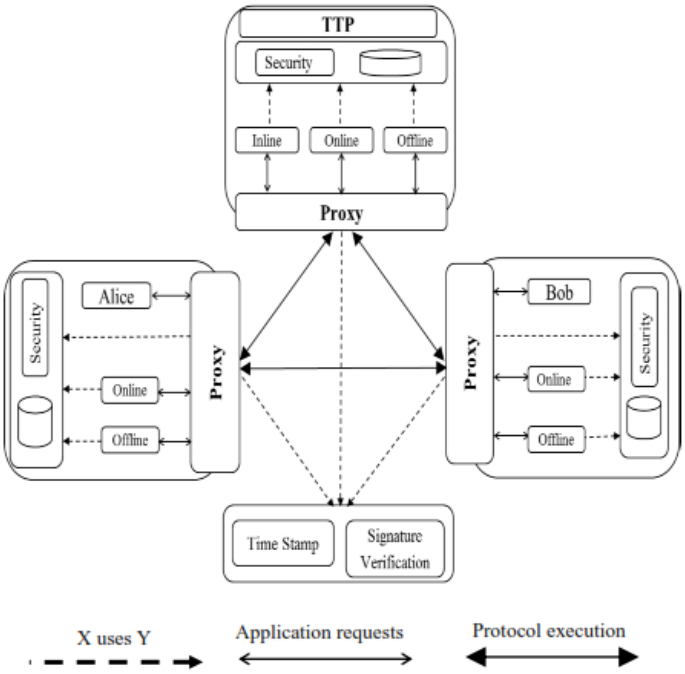

Figure 3. Non-repudiation Services architecture.

Trusted Third Party Service: It is a trusted third party that implements three different types of fair non-repudiation protocols: inline protocol, online protocol and offline protocol [3]. Inline TTP or delivery agent was presented by Coffey and Saidha in 1996 [3, 7]. It involves all the interactions between sender and receiver. The online TTP [2, 3, 8] only takes part in some required requests such as exchanging secret keys or abort sub-protocol. The offline TTP [2, 3, 9] or optimistic protocol involves a transaction between parties only in the case of failure or if one of the parties misbehaves. Therefore, the TTP consists of three main resources called inline, online and offline which represent fair non-repudiation exchange protocols. Each resource has a unique Uniform Resource Identifier (URI) which can be accessed by using a URL.

TTP also has a document resource which is responsible for storing the documents such as pdf, word and text files as well as their metadata such as document author, document title and type of document. All protocol resources need to use document resource in order to obtain the document and its information from the database. This design feature might help to co-locate the storage related to the document with the resource protocols. Another advantage is that the resources can be placed in different machines in order to provide high performance.

Proxy Resources: Each protocol resource (inline, online and offline) has a relevant proxy resource (inlineProxy, onlineProxy and offlineProxy). The main reason behind using the proxy approach is that it acts on behalf of end users [5] to encrypt, sign, and execute appropriate protocols. It will keep the end user away from implementation details as well as executing the protocol. Therefore, the end user does not have to worry about implementing protocols correctly. For example, if the client wants to send the document to another client using inline protocol, he does not need to worry how the document is encrypted or what algorithm he should use. He only needs to make a request to his proxy then the proxy will do the rest such as encryption, nonrepudiation of origin and timestamp signature.

Timestamp and Verification Service: This service is designed to provide two functions to The Non-Repudiation System: timestamp and verification of signatures. It consists of two resources which are timestamp and verification. Firstly, the timestamp is a resource that provides timestamp for signature. The timestamp is digital evidence that proves the real time of signing the document. The timestamp is important because if the client changes his key pair, the non-repudiation of origin cannot be verified since the verification uses the public key to verify the signature which has signed using the private key. Therefore, the timestamp provides the clients evidence that the signature was signed before changing his key.

The second resource within timestamp and verification service is verification resource. It is a resource that is used to verify the non-repudiation of origin and non-repudiation of receipt. It is used by the trusted

third party and clients during an exchange of non-repudiation evidence. TTP should verify this evidence before storing in the database in order to store a valid document with its non-repudiation. Online and offline protocols the responsibility of verification of evidence is transferred to the clients; therefore, they need to verify incoming digital evidence using this service. For example, when Alice sends the non-repudiation of origin to the receiver, Bob, using online or offline protocol, Bob should make sure that the evidence is related to the document and it is valid.

3.2. Protocol Resources

Inline Resource: This resource is implemented on TTP only. The main purpose of this resource is to provide Coffey and Saidha fair exchange protocol in the nonrepudiation System [7]. The clients only have to make requests to inline resource Application Programming Interface (API) to execute the protocol. The resource will take care about the verification and validation of messages as well as storing information in the database. In addition to that, the abort and resolve sub-protocols are also implemented on TTP side.

Online Resource: The reason of this resource is to implement Zhou and Gollman fair non-repudiation exchange protocol [8]. This resource is implemented in clients' and TTP sides. The responsibility of verification and validation is transferred to clients; therefore, they are responsible for storing information obtained while executing online protocol. However, the responsibility of TTP in online protocol is related to fairness rather than participation in exchanging the documents. The online TTP participates in secret key exchange as well as abort and resolve sub-protocols.

Offline Resource: Offline resource provides offline fair non-repudiation exchange protocol introduced by Asokan, Shoup and Waidner [9]. This protocol all communication is between the participants Alice and Bob except in the case of abort or resolved in which case the TTP is involved. As online protocol, the clients, who use offline interaction, are in charge of saving data in the database.

Documents Resource: The main purpose of this resource is to store and retrieve documents and documents' metadata in the storage unit. It is implemented in both services Client and TTP. All protocol resources retrieve the documents and their metadata through the document resource. This feature can help to place the database related to documents in different machines.

3.3. Applied REST Constraints

This section shows which of REST constraints have been applied to the project. Then it explains the applied REST constraints within project context, and provides examples from project design for each constraint.

Stateless: It means that keeping unwanted representation states out of the server. There are two types of representation states; application states and resource states. Application state is undesired to be stored on the server; while the resource state is necessary to be stored on the server.

All requests within Non-repudiation Services are stateless because the project does not store any application states from clients on the server but only stores the resource states. Non-repudiation Services project stores documents and corresponding metadata in addition to exchange information because they are necessary information for each transaction. While, results of retrieving a list of a particular exchange are considered an application state and it should not be stored on the server.

Client-Server: It means that the client side concerns and the server side should be decoupled. This REST constraint has been applied to Non-repudiation Services because the client side and server side are completely separated. The project (clients and server) is implemented in the Java programming language. However, the clients can be implemented in a different language that can generate proper HTTP requests such as C#, Ruby and PHP.

Layered System: It means that the component within the project should be organized in hierarchy way. It allows proxies and gateway interceptor to be placed between the client and server. Layered system assumes that when clients make requests to a server, they cannot know whether they are communicating with the end server or not. This property has been applied to the project. Clients are connected to their proxies, but they cannot determine whether there are more proxies or interceptor between them and end server or not. Clients only know the layer which communicates through. For instance, when Alice makes a GET request to obtain

a receipt of a particular exchange, she passes her request to her proxy to communicate with the server.

Uniform Interface: Uniform interface is REST constraints that include four constraints which are identification of resources, manipulation of resources through these representations, self-descriptive messages and hypermedia as the engine of application state (HATEOAS).

Identification of Resources: This means that all resources should be accessible through URI. This constraint is considered in the project in which all resources within the Non-repudiation project are identified using URIs.

Manipulation of Resources through Representations: It means that clients can add, update and delete resources through representations. This property has been considered in this project which is providing clients the ability to manipulate resources using representations. For example, Alice creates a new resource when she sends a document to be stored in TTP. Also, a client can update a resource for instance when Bob sends non-repudiation of receipt to TTP, he makes a PUT request to update existing exchange.

In addition, resources might be represented in a different data format. For example, information about exchange can be presented as JSON or text while documents are converted to byte array before being sent to the server.

Self-Descriptive Messages: The Non-repudiation project has been implemented with self-descriptive messages. Self-descriptive messages mean that the message must contain all necessary information in order to be processed. For instance, when Alice wants to obtain all exchange operations between her and Bob, she must pass her id and Bob's id in order the server can understand the request.

Self-descriptive also means that the HTTP methods must be used properly and without contradicting with their verb meaning. GET method should be used for retrieving information only. POST should be used to create a new resource. PUT method should be used to update an existing resource. These properties have been applied to all methods within Non-repudiation Project. For example, when Alice sends a document to TTP, she uses POST method and Bob uses a PUT method in order to send a non-repudiation of receipt to TTP.

Hypermedia as The Engine of Application State (HATEOAS): It means changing the current application state to another state using dynamic hypermedia links provided by the server. The Non-repudiation project enables clients to interact with different resources using hypermedia links. This means that clients can change the current state of application to another state related to the current state. However, the server does not provide dynamic hypermedia for all methods within the Non-repudiation project, but only for some methods. Dynamic hypermedia can be provided by embedding resource URI within representations. For instance when Bob makes a GET request to obtain all exchanges with Alice, he will get a list of all exchanges. Then all succeeded exchanges will have extra links to download the document as well as to obtain the non-repudiation of origin.

3.4. Non-repudiation Services API

REST API is a set of Uniform Resource Identifiers (URI) that allows clients to access to resources. Non-repudiation Services project provides three types of fair exchanges protocols inline, online and offline as resources. Each of these types has a different API because of the nature of the protocols. These protocols are implemented as individual resources within Non-repudiation Services project. Each resource has a unique URI that helps clients to distinguish the executing of each protocol.

Moreover, each resource has a number of sub-resources. A sub-resource is a method which represents a particular step of executing of a protocol. Numbers of sub-resources are different in each resource depending on how the protocol is executed. Sub-resources are also identified by a unique URI. In order to access to sub-resource or to execute a step of one of the protocols, clients need to make a request to the URI of the root resource followed by sub-resource's URI. For example, when Bob wants to send a non-repudiation of receipt to inline TTP, his proxy needs to make a request to the inline resource's URI followed by the URI of the method in order to send the receipt to the TTP.

As mentioned previously, each resource has a unique URI. Each URI consists of a set of parameters. The following is a brief description of the main parameters within URIs:

- *http://localhost/ttp*: It means that the resource is on the TTP server.

- *v1*: It represents the version of the system.
- *sid:* It is a unique identifier for the sender.
- *rid:* It is a unique identifier for the receiver.
- *proxy:* It represents the proxy resources or protocol interface.
- *inline/ttp*: It means that this URI is used to execute inline protocol.

Inline Resource URI within TTP:
http://localhost/ttp/v1/inline/ttp/sender/sid/exchanges/receiver/rid.
Online Resource URI within TTP:
http://localhost/ttp/v1/online/ttp/sender/sid/exchanges/receiver/rid.
Offline Resource URI within TTP:
http://localhost/ttp/v1/offline/ttp/sender/sid/exchanges/receiver/rid.
Inline Proxy Resource URI within Alice's Service:
http://localhost/alice/proxy/v1/inline/ttp/sender/sid/exchanges/receiver/rid.
Online Resource URI within Alice's Service:
http://localhost/alice/v1/online/ttp/sender/sid/exchanges/receiver/rid.
Online Proxy Resource URI within Alice's Service:
http://localhost/alice/proxy/v1/online/ttp/sender/sid/exchanges/receiver/rid
Offline Resource URI within Alice's Service:
http://localhost/alice/v1/offline/ttp/sender/sid/exchanges/receiver/rid.
Offline Proxy Resource URI within Alice's Service:
http://localhost/alice/proxy/v1/offline/ttp/sender/sid/exchanges/receiver/rid.
Inline Proxy Resource URI within Bob's Service:
http://localhost/bob/proxy/v1/inline/ttp/sender/sid/exchanges/receiver/rid
Online Resource URI within Bob's Service:
http://localhost/bob/v1/online/ttp/sender/sid/exchanges/receiver/rid.
Online Proxy Resource URI within Bob's Service:
http://localhost/bob/proxy/v1/online/ttp/sender/sid/exchanges/receiver/rid
Offline Resource URI within Bob's Service:
http://localhost/bob/v1/offline/ttp/sender/sid/exchanges/receiver/rid.
Offline Proxy Resource URI within Bob's Service:
http://localhost/bob/proxy/v1/offline/ttp/sender/sid/exchanges/receiver/rid.
Timestamp Resource URI within Timestamp and Verification Service:
http://localhost/timestamp.verification.service/v1/timestap
Verification Resource URI within Timestamp and Verification Service:
http://localhost/timestamp.verification.service/v1/verification/service

3.5. Executing Non-repudiation Protocols

In order to show how the Non-repudiation System works, the inline protocol will be discussed as an example. In this part the role of each method in executing of the protocol will be described then the inline protocol execution steps will be discussed.

Inline Resource: Inline resource is Coffey and Saidha fair exchange protocol implementation. It provides clients to exchange documents in fairness and without cheating. It consists of several methods which will be described later. The inline resource is considered as a root resource for methods. The root resource should have a unique URI in order to be accessed. The URI of the inline resource is http://localhost/ttp/v1/inline/ttp/sender/sid/exchanges/receiver/rid. Therefore, to access to inline resource methods, the clients need to pass a request to the root API followed by method's API. The following are the inline resource methods.

getDocument (String, String, String): This method is used by the receiver to retrieve a document from TTP. It takes three string parameters; sender id, receiver id and document id. The purpose of this method is to retrieve a document from the TTP. The method will check whether the receiver has sent the corresponding

non-repudiation of receipt or not. It only allows the receiver to obtain the document when TTP has received the corresponding non-repudiation of receipt in advance. The get document method has a unique URI, which enables the receiver to access to the method using HTTP GET. The method URI, which is a compliment for main resource URI, is "docid/document".

getExchanges (String, String): Clients can use this method in order to know available exchanges as well as states of current exchanges. This method takes two parameters which are sender id and receiver id in order to process the request. The method can be accessed by using HTTP GET to inline resource URI.

getSignature(String, String, String): This method is used by the receiver in order to obtain a signature (non-repudiation of origin). It takes three parameters; sender id, receiver id and document id. The receiver makes HTTP GET request to the inline resource URI followed by "docid/signature". It only allows the receiver to obtain the signature when TTP has received the corresponding non-repudiation of receipt in advance. TTP will return a JSON object to the receiver. The object will contain non-repudiation of origin, timestamp and time.

putReceipt(String, String, String, byte[], byte[], String): This method is used by the receiver to send the non-repudiation of receipt to TTP. It takes sender id, receiver id, document id, receipt, timestamp and time. The method checks the validity of receipt and the timestamp. If both are valid TTP will update corresponding exchange operation and make the document, the receipt and the signature are ready to be retrieved from the TTP. The put method can be accessed by using HTTP PUT request to inline URI followed by "docid/receipt". When a client makes a PUT receipt, he either will get status code 202, which means the receipt is accepted or failure code.

createExchange(String, String, String, byte[], byte[], String): This method is used by the sender in order to create an exchange. It takes five parameters; sender id, receiver id, document id, signature, timestamp and time. Before creating an exchange record, TTP checks the validity of each of the signature and timestamp. The method has a unique API which enables clients to create exchanges. It can be accessed to by using the URI of the inline resource. The method takes HTTP POST request in order to create exchanges. The response of the request will be either a success message with status code 201 or failure code.

getDocumentHash(String, String, String): This method is used by the receiver in order to obtain a hash of the document. It takes three parameters sender id, receiver id and a document id. The receiver makes HTTP GET request to inline resource URI followed by the method URI, which is "docid/hash". The response will be either a byte array of hash document with status 200 or failure code. The hash of the document is required by the receiver in order to generate the non-repudiation of receipt.

getReceipt(String, String, String): This method is used by the sender in order to obtain a non-repudiation of receipt. It takes three parameters; sender id, receiver id and document id. The sender makes HTTP GET request to the inline resource's URI followed by the method URI, which is "docid/receipt". It only allows the sender to obtain the receipt when it is available. TTP will return a JSON object to the sender. The JSON object will contain non-repudiation of receipt, timestamp and time.

abortExchange (String, String, String, String): Abort method is used by both sender and receiver in order to abort a particular exchange. It takes four parameters; sender id, receiver id, document id and abort token. A client makes an HTTP PUT request to the inline resource's URI followed by the method URI, which is "docid/abort". The method checks whether the exchange has succeeded or not. If it succeeded, it returns to the client a message shows that the exchange has succeeded with status code 400. If the exchange has not succeeded, the TTP aborts the exchange and returns to the client a message showing that the exchange has aborted with status code 200. Otherwise it will return a failure code.

resolveExchange (String, String, String, String): The resolve method is used by both sender and receiver in order to resolve a particular exchange. It takes four parameters; sender id, receiver id, document id and resolve token. A client makes an HTTP PUT request to the inline resource's URI followed by the method URI, which is "docid/resolve". The method checks whether the exchange has succeeded or not. If it is succeeded, it returns to the client the exchange has succeeded with status code 200. If the exchange has not succeeded and the receiver has not sent a corresponding receipt yet, the TTP will abort the exchange and

return to the client a message which shows that the exchange has aborted with status code 400. Further, if the receiver has sent the receipt and the exchange is not aborted, the TTP will return a success message with status code 200. Otherwise, it will return a failure code.

Executing Inline Protocol: Inline protocol consists of a number of executing steps. Figure 4 shows the steps between clients and the TTP. These steps will be discussed in the same order of execution of the protocol as follows:

1. Alice stores a document to her local database to be sent later to TTP. She makes a POST request in order to be stored in the database.
2. In the second step, Alice needs to store the document in TTP's database by making a POST request to her proxy. The proxy will retrieve the document from local database then encrypt it using AES algorithm. The secret key is encrypted with RSA algorithm using the TTP's public key. Then send the data to TTP.
3. After that, Alice creates an exchange with Bob by making a POST request to her proxy. The proxy will retrieve the document from the local database the sign it. The proxy then will generate the non-repudiation of origin (signature) then send the signature to the timestamp service using a POST request in order to obtain the timestamp signature and time. After that the signature and timestamp signature and time are sent to the TTP.
4. In order to obtain the document, Bob must send the non-repudiation of receipt. Therefore, Bob makes a PUT request to his proxy. The proxy will retrieve the hash of the signature from TTP. The proxy will sign hash of the signature to generate non-repudiation receipt. Then the proxy will send the receipt to timestamp service to obtain timestamp signature and time. Then the proxy will send these values to TTP to make the document be ready for download.
5. Alice now can obtain the non-repudiation of receipt from the TTP by using GET request to her proxy.
6. Bob can obtain the document by making GET request to his proxy and also he can obtain the signature and timestamp signature.

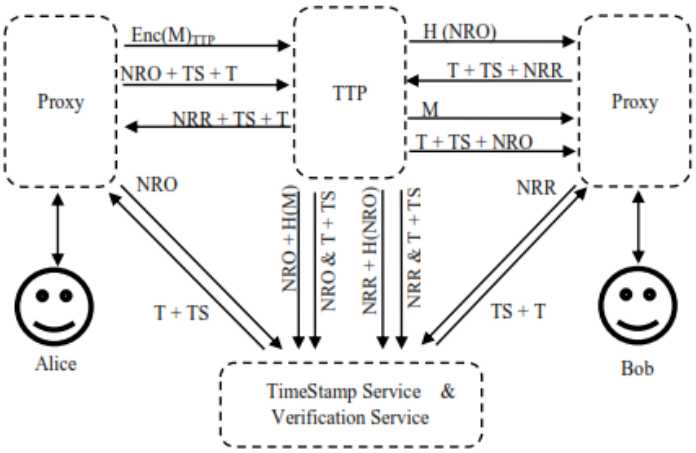

Figure 4. Inline protocol execution.

4. System Implementation

This section will discuss different technology and approaches that have been used in the project. It will show why these technologies have been chosen and how they are implemented. It will discuss database section first then after that will discuss security section.

4.1. Language and Tools

This project will be implemented using Java language. Eclipse Indigo has been used as Java editor. In order to make REST request, jersey libraries have been used. Jersey is the open source project used for building RESTful Web services. Jersey supports annotations defined in JAX-RS making developing web service easier for developers by using Java and Java VM. One of the JAX-RS annotations is @Path, which identifies the URI for a specific resource. The project will be tested using JUnit and FindBugs plugins. They are open source plugins that help developers to check the functionality of their code as well as check their code for

bugs.

4.2. Proxies

The Apache HTTP Server has been used as a proxy within Non-repudiation Services. The proxy is used in the project in order to pass HTTP requests to resources that are running on tomcat server. The proxy is used in the project in different services and for different purposes. For instance, when Alice sends a document to Bob, she makes a POST request to her proxy. The proxy will pass the request to the relevant proxy resource in order to generate a signature and a timestamp then send data to Bob.

4.3. Project Database

The Non-repudiation Services use MongoDB [22, 23] as a database system. MongoDB is an open source project based on document oriented database, which is written in c++ and supported by 10gen. It is schema free and designed to be scalable by storing the information as BSON object, instead of tables as in relational databases. BSON object is a binary JSON. JSON (JavaScript Object Notation) is a lightweight data format that consists of collections of key/value pairs. The goal of MongoDB is to fill the gap between the relational databases and key-value stores. In order to have the functionality of the key value which are fast and scalable and features of relational databases such as indexes and dynamic query [23, 24].

There are many reasons behind choosing MongoDB as a database system for the project. The main reasons are the data model, query model and scalability of MongoDB.

Data Model: The database in MongoDB consists of many collections, which are used to contain any type of documents. Each document consists of a set of fields that have set of key value pairs [25]. MongoDB modifies the attributes individually which means that if a specific attribute has been changed only that attribute will be sent back to the database. The documents have unique id works as primary key in relational databases in order to make fast queries [25].

The Non-repudiation Services project consists of three databases which are *ttpDB*, *aliceDB* and *bobDB*. Each of the TTP, Alice and Bob has two collections within their database. TTP's collection called *ttpDocument* and *ttpExchange*, while Alice's collection called *aliceDocument* and aliceExchange and Bob's collection called *bobDocument* and *bobExchange*. These pairs of collection in each side work similar and have the same purpose. One of the collections is used to store documents and its metadata. The other collection is used to store information about interactions between clients.

MongoDB support arrays and nested objects and provides indexing for them; moreover, it has a special feature for arrays called "multikeys", which allows arrays to be used as indexes. Arrays in MongoDB may contain tags for documents and that helps in search for documents by tags. However, MongoDB only allows query and indexes to be made on one collection [25]. For example, in order to update a specific exchange within ttpExchange collection, the sender id, receiver id and doc id are used as tags to find the exchange.

The relationships in MongoDB are presented by embedded objects and arrays; therefore, the data model might be used as a binary tree; otherwise, there are two techniques. Firstly, denormalization of the data model, which means the MongoDB might replicate some documents in the database. This option might be good when there is no regular update in the system. The second option is to build client side joins. This option increases the complexity in the client side and network traffic in the system [25].

Query Model: Queries in MongoDB are dynamic which enable users to search any information inside the collection using any criteria, without being restricted by specific indexes. In addition, it implements queries over all documents within the same collection, which improves the performance of the queries [26]. For instance, all exchanges within TTP's "*ttpExchanges*" collection can be retrieved by passing an empty selector as following:

```
Mongo m = new Mongo();
DB ttpDB = m.getDB("ttpDB");
ttpDB.ttpExchanges.find({});
```

In order to obtain an exchange for a particular document, the document id is passed to the collection as follows:

```
ttpDB.ttpEexchanges.find({"docId":"501002c51d8"});
```

MongoDB provides indexes for all documents in the collection to improve the performance, which is a data structure that contains data associated with special field in the MongoDb's collection. MongoDB's query optimizer uses the index to sort quickly and order all the documents inside the collection [27]. The index can be created by "ensureIndex()" function.

MongoDB has another method to provide indexes for the values inside the documents called "multikey" [25, 28]. It is a special feature with arrays in which enable arrays to be used as index automatically for a specific value in the collection's document.

Scalability: MongoDB scales automatically using a sharding or partitioning technique. It means distributing data over the system's machine automatically and in a specific order. Sharding in the MongoDB enables the system to be scaled horizontally over multiple nodes. MongoDB takes the responsibility of balancing and failure over the nodes while converting single database to sharded cluster [25, 29].

A Sharded cluster consists of three main components shards, config servers and routing processes (Mongos) to which the application servers connect (see Figure 5). The shard includes one or many servers, store the information using mongod process. The config servers store the cluster's metadata, which includes basic information on each shard server and the chunks contained therein. Finally, the routine processes perform the tasks requested by the clients, which sends the request to the required shard then merges the final result and returned to the client [25, 29].

The documents, inside the MongoDB collection, are partitioned using shard key. The shard key, can have multiple fields just like indexes, is responsible for partitioning the collection into shards. Every shard store associated documents using this key. The documents within each shard arranged into chunks. Each chunk can store data up to 64MB, and if the data has reached the maximum size, the chunk will split into two chunks [25, 29].

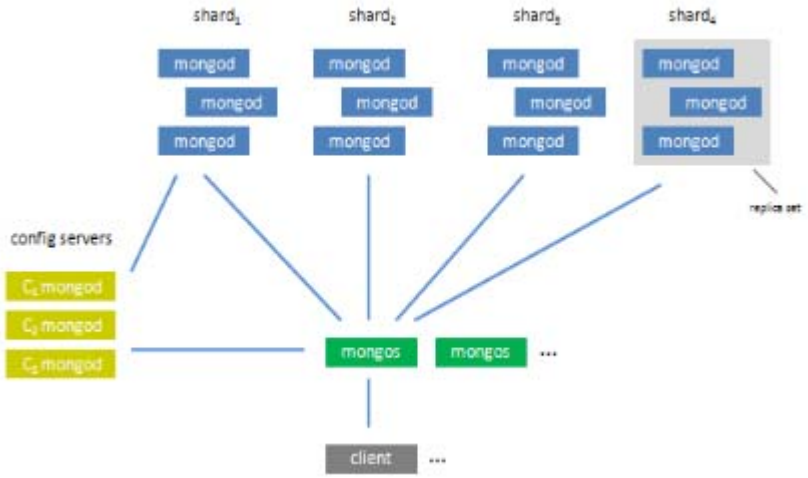

Figure 5. MongoDB shard cluster [29].

4.4. Security Section

Cryptography: Cryptography is used to encrypt private information called plaintext and convert it into indecipherable information called ciphertext in order to keep information secret to some people. There are two main types of encryption: symmetric (also known as secret key), and asymmetric (or public key cryptography) [30]. In symmetric cryptography, the same secret key is used to do encryption and decryption of a message to both encrypt and decrypt the data. However, protecting the private key is critical to preserve the confidentiality of data. On the other hand, asymmetric cryptography uses a public/private key pair in order to encrypt and decrypt data. Data encrypted with the public key and is decrypted with the private key. Keeping the private key confidential is critical to this scheme. Symmetric algorithms are generally faster than Asymmetric algorithms. Asymmetric algorithms are not designed for efficiently preserving data with large size. In practice, asymmetric algorithms are used to exchange smaller secret keys which are used to initialize symmetric algorithms.

Two types of encrypting algorithms have been implemented which are *Advanced Encryption Standard*

(AES) and *RSA*. *AES* is a symmetric key algorithm which means that same key is used to cipher and decipher the data. AES is fast to encrypt and decrypt large data but the key is the weak point in these types of algorithms. Therefore, the asymmetric key algorithm is used to protect the key from being stolen. RSA uses asymmetric keys in which the client will encrypt the message using the TTP public key the TTP will decrypt the message using its own private key. RSA takes longer time to encrypt and decrypt large amount of data than AES. Therefore, AES is used to encrypt and decrypt the data using AES's symmetric key then RSA is used to encrypt and decrypt the AES key.

Digital Signatures: Digital signatures are evidence that gives authenticity of messages [31]. It uses public/private keys in order to generate and verify signatures. The signature can be generated by encrypting a message using the private key. Only the owner of the private key can generate the signature. The signature can be verified by anyone who uses the corresponding public key.

Hash Functions: One-way functions are public functions that can be used to identify digital objects without involving secret keys [31]. For instance, in the project Alice signs the hash of the document rather than the whole document. In this case, the hash of the document is considered as a unique id for the document. It is computationally infeasible to create the original text from a hash of the text or to obtain same hash value for a different text. Moreover, hash functions can also provide authenticity and integrity of message [31, 32]. For example, when Alice sends a document to TTP with corresponding signature, TTP computes the hash of the document and send the signature as well as document's hash to the timestamp and verification service in order to be verified.

Portecle: Portecle is open source software that provides user friendly GUI application [33]. It can be used for the following purposes:

- *Create, load, save, and convert keystores.*
- *Generate DSA and RSA key pair entries with self-signed version 1 X.509 certificates.*
- *Import X.509 certificate files as trusted certificates.*
- *Import key pairs from PKCS #12 and PEM bundle files.*
- *Clone and change the password of key pair entries and keystores.*
- *View the details of certificates contained within keystore entries, certificate files, and SSL/TLS connections.*
- *Export keystore entries in a variety of formats.*
- *Generate and view certification requests (CSRs).*
- *Import Certificate Authority (CA) replies.*
- *Change the password of key pair entries and keystores.*
- *Delete, clone, and rename keystore entries.*
- *View the details of certificate revocation list (CRL) files*

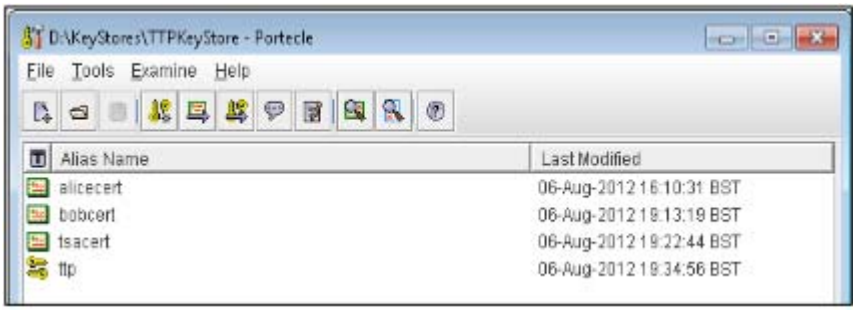

Figure 6. Portecle software.

Portecle software has been used to do the following in a Non-repudiation project (see Figure 6):

1. Generate key stores for clients, TTP and TSA.
2. Also, it has been used to generate key pairs for each one of them.
3. Export x.509 certificates for each of them.
4. Import x.509 certificates and store them within key store.

The main reason of using *Portecle* is that it is a simple software to use. Also, it is easy to understand. For

example, to generate a key store, it just requires going to file then generates key store. Also, to generate a key pair, it just needs to click tools then chooses generate key pairs it will ask to choose the size of the key as well as type of algorithm. After that, it will ask to fill the certificate details. Another reason of using *Portecle* is that it enables exporting and importing certificate easily.

4.5. Java Cryptography Architecture (JCA)

The JCA is a part of the Java platform [30]. It includes a "provider" architecture and a set of APIs. These APIs provide digital signatures, message digests, certificates and certificate validation, encryption, key generation and management. These APIs enables developers to integrate security into their software easily.

The architecture was designed around three principles, which are Implementation independence, Implementation interoperability and Algorithm extensibility. Firstly, Implementation independence, which means implementing security algorithms are not required [30]. Instead of that, security services can be requested from the Java platform. Secondly, Implementation interoperability, which means providers are interoperable across applications [30]. In other words, applications and providers are not bound. Finally, Algorithm extensibility which means the Java platform contains a set of built-in providers that execute a basic set of security services that are largely used and additional custom providers can be added [30].

The project has implemented a security service which is called *SecurityService* class. This class is available in all parts of the project (proxies, TTP and timestamp and verification service). Three classes from JCA have been used in the *SecurityService* class which is signature, digest and cipher classes.

The Signature Class: It is designed to produce digital signatures [30]. It is an engine class that provides a cryptographic digital signature algorithm such as *DSA* or *RSAwithMD5*. It takes a byte array with a private key in order to generate a signature. The signature is a fixed size string of bytes, which has the following properties:

- The signature can be generated only by the owner of private/public key pairs.
- Private Key should not be recovered from the public key or the signature.
- The authenticity and integrity of the input should be possible to be verified using the relative public key.

A signature object is initialized for signing with a private key and is given the data to be signed. The result will be a signature byte array. When verification is needed, signature object is created and initialized for verification. The object takes the public key corresponded to the private key that generates the signature. The data and the signature bytes are fed to the signature object, and if the data and signature match, the signature object reports are successful.

Signature Object States: Signature object has three states which are *UNINITIALIZED*, *SIGN* and *VERIFY*. The signature object state is *UNINITIALIZED* when the object is created and returns to this state after each operation. The SIGN state is used when it requires to sign data and *VERIFY* state is used when it requires verifying a signature. In order to change the state of signature object there are two methods called *initSign* and *initVerify* which change the state to *SIGN* and *VERIFY*, respectively.

The project provides two types of digital evidence non-repudiation of origin and non-repudiation of receipt. Non-repudiation of origin is provided by Alice while non-repudiation of receipt is provided by Bob. Both sign the hash of a document in order to provide the non-repudiation evidence. They use *sign* and *verify* methods within the *SecurityService* class in order to generate a signature and verify a signature as follows:

sign(byte[] fileHash): This method is used to sign the document, in which a *KeyStore* object is obtained by using *getInstance()* static factory methods. The key store needs to be loaded to the memory, so the private key to be obtained to be used later to sign the file hash. Signature object is obtained using *getInstance()* method with transformation "*SHA1WithRSA*" which means the object will digest the data first to be controlled better then signed using RSA algorithm, the object then initialized with Alice's private key then update the signature object with the file hash and finally sign it using sign method .

verify(byte[] signature, byte[] documentHash): This method is used to verify a signature created or signed by the sender. It takes two variables; a signature and hash of a document. A KeyStore object is obtained by using *getInstance()* static factory methods then it must be loaded to the memory to be used. A Certificate object is obtained from the key store using *getCertificate* method by specifying the name of the certificate.

The public key is obtained from the certificate to be used in the verification process. The signature object is created then initialized as a verification object with the public key. The signature object is updated with the document hash and it does the verification using the verify method which takes the signature. The result of the verification is boolean value either it is true in case of a valid signature or false in case of invalid signature.

The MessageDigest Class: The MessageDigest is a security class designed to provide *SHA-1* or *MD5* cryptographic secure message digests [30]. Message digest class takes a byte array as input and produces a fixed-size output. For example, the *MD5* algorithm produces 16 bytes digest, and *SHA1* generates 20 bytes. This output byte array called a hash or *digest*.

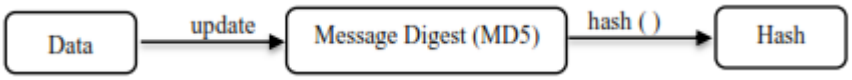

Figure 7. Message digest.

A digest has two properties:

- The hash function produces a unique output for each input which means it should be impossible to hash two different inputs and produce the same value.
- The hash output should not tell anything about the input data used to produce the hash.

Message digests are used to generate unique and reliable identifiers of data. They are sometimes called "checksums" or the "digital fingerprints" of the data. Changes to just one bit of the message should produce a different digest value.

The message digest has been used in non-repudiation services to produce a unique identifier for documents. Document hash is used by clients to provide non-repudiation evidence. Alice provides a non-repudiation of origin by signing hash of a document. Likewise, Bob produces a non-repudiation of receipt by signing hash of a document. The hash method takes a byte array as a parameter and then generates a hash file which is also a byte array as can be seen in figure 7.

- **hash(byte[] document):** This method is used to digest the message, in which MessageDigest object is obtained using *getInstance()* and with transformation "*sha-1*". The digest object is updated using the update method which takes the document as a byte array then digest the document byte array or hash it using digest method which produces 20 byte result as follows:

```
public byte[] hash(byte[] document) throws
Exception
{MessageDigest md = essageDigest
              .getInstance("SHA-1");
md.update(document);

return md.digest();
}
```

The Cipher Class: The cipher class provides the ability to encrypt and decrypt data [30]. Encryption is the process of converting plain text to meaningless text called cipher text. This cipher text will be meaningless for the third party who does not have a key used to generate the cipher text. Decryption is the inverse process: that of taking ciphertext and a key and producing cleartext.

- **rsaCipher(byte[] plainText):** This method is used to cipher the byte array plain text using TTP certificate which is located in Alice's key store. It creates the key store object with instance "*JCEKS*", then the key store is loaded to obtain the TTP certificate from the key store. A Cipher object is gained using *getInstance()* static factory method from Cipher class with a specific transformation which is the name of a cryptographic algorithm such as RSA and may be followed by a mode and padding scheme. Then to use the object for the encryption it must be initialized with the encrypt mode with TTP certificate. The data now can be encrypted using the cipher object by using the *dofinal()* method.
- **aesCipher(byte[] plaintext, SecretKey sk):** This method is used to encrypt the plain text using

symentric key or secret key. The cipher object is obtained using Cipher class's static factory *getinstance()* with transformation "*AES/ECB/PKCS5Padding*", which means that the type of algorithm is AES, encrypting mode is Electronic Codebook and padding is *PKCS5Padding*. The cipher object is then initialized with the encrypt mode and secret key. After initializing the cipher object now it can be used to encrypt the data using doFinal method which is used to encrypt the data then the *aesCipher* function will return the encrypted data as a byte array as follows:

```
public byte[] aesCipher(byte[] plainText, SecretKey
sk)throws Exception
 {
 Cipher aesCipher = Cipher
         .getInstance("AES/ECB/PKCS5Padding");
  aesCipher.init(Cipher.ENCRYPT_MODE, sk);
  return aesCipher.doFinal(plainText); }
```

- **generateAESKey():** It is a method used to generate a secret key for AES algorithm. In which *GenerateKey* object can be obtained by using getInstance methods with transformation AES then using that object to generate the secret key by calling *generateKey()* method as follows:

```
public SecretKey generateAESKey()throws Exception
 {  KeyGenerator keygen = KeyGenerator
                     .getInstance("AES");
  return keygen.generateKey();
 }
```

- **rsaDecipher(byte[] cipherText):** This method is used to decrypt the encrypted message using RSA algorithm. *Keystore* object is obtained using key store to get instance method with RSA algorithm, and then key store must be loaded to be used. The TTP private key is obtained from the loaded key store using *getPrivateKey* method. A Cipher object is obtained using *getInstance* method for "RSA" algorithm, then using that object to decrypt the cipher text by initializing the object with the decrypt mode and private key then decipher the cipher text by executing *doFinal()* method.
- **aesDecipher(byte[] cipherText, byte[] sk):** This method is used to decrypt the cipher text using AES algorithm, in which it takes a cipher text and the secret key as a byte array. The secret key object is created using SecretKeySpace, which takes the secret key as a byte array and "RSA" as the name of the algorithm. After creating the secret key the cipher object is created and initialized with decrypt mode and secret key then decipher the encrypted text using doFinal method.

5. System Testing

This section describes the systematic testing of the Non-repudiation Services and an explanation of the testing bed established. Test outcomes are compared with the predefined expected outcome.

5.1. Testing Strategies

There are two main types of tests have been done to the Non-repudiation project which are unit testing and interaction testing.

Unit Testing: The purpose of these tests is to ensure and verify the functionality of a specific section of the code. For instance, it provides testing for security service method such as encryption decryption, signing and verification methods. In addition, it tests most of the database's methods such as *createExchange* and *createDocument*. This test has been done by testing each method using JUnit. Each test comprised of a purpose and inputs for the test. These inputs are compared to expected value in order to be tested (see table 2). For example, in order to ensure that the encryption using key pair is working correctly, the rsaCipher method from *SecurityService* class has been tested. The method is tested by encrypting a string (byte of string) using TTP public key. Then decrypt the encrypted result using the TTP private key. In order to

complete the test, the result of decryption is compared with the original string.

Table 2. Unit tests.

No.	Purpose of test	Inputs	Expected outcome	Actual outcome
1	To see if the encryption of aesCipher method is working or not	A hash of a string has been encrypted using aesCipher method	The expected outcome is a cipher text and can be deciphered using the same key	The method produces the expected result which is cipher text can be decipher with the same key
2	To see if the encryption of rsaCipher method is working.	A hash of a string has been encrypted using TTP's public key.	The expected outcome will be the encrypted result can be deciphered using TTP's private key	The method produce the expected result which is cipher text can be decipher with the TTP's private
3	To see if the signature provided by sign method can be verified or not	A hash of a String has been signed using Alice private key	The expected result will be the signature can be verified using Alice's public key	The result was as expected

Another type of unit testing has been done by using FindBugs plugin. FindBugs is a free plugin that helps developers to find coding bugs easily. It is added to eclipse in order to check the project of existing bugs. The project has been scanned for coding bugs using this tool. Several bugs have been found within the project during checking for bugs, and later these bugs have been solved. For example, in the *sign* method within the *SecurityService* class, a key store data was loaded as an input stream to the memory without closing the stream. However, it cannot be guaranteed that the project is totally free of bugs.

Interaction Testing: This aims to test the interaction of all non-repudiation protocols which have been used in the project. It tests and calculates overhead of methods related to execute each protocol. For instance, send a document, send a receipt, abort an exchange and resolve an exchange.

Another reason of interaction testing is to ensure that the project resources are implemented according to HTTP methods property which are safe and idempotent (see table 3). For instance, when Bob sends his receipt for an inline exchange, he uses PUT request. This method has been tested in order to make sure that there is no side effect of requesting the method more than once.

Overhead of each request has been tested using JUnit. Each request was executed for five times, and each time the estimate executing time (ET) has been calculated.

Table 3. Testing inline protocol requests

No	Purpose of request	HTTP method	Average execution time (milliseconds)	Result
1	Store a document in the local database	POST	101.2	Pass
2	Store a document in the TTP	POST	89	Pass
3	Send the document to Bob via TTP	POST	135.6	Pass
4	Send the receipt to Alice via TTP	PUT	152	Pass
5	Get the document from TTP	GET	101	Pass
6	Get the receipt from the TTP	GET	105.6	Pass
7	Get the Signature from the TTP	GET	83	Pass
8	Abort an exchange	PUT	89	Pass
9	Resolve an exchange	PUT	84.8	Pass

.

6. System Evaluation

This section of the dissertation is allocated to evaluate the Non-repudiation Services project. The evaluation consists of analysing what has been done and compare it with the project aim and objectives. Moreover, it provides the advantages of implementing the non-repudiation services as RESTful web services.

The main aim of the project was to design and implement a RESTful NonRepudiation Services. The project was divided into several objectives in order to make the implementation easier.

The first objective was to implement an inline fair exchange protocol which was introduced by Coffey and Saidha [7]. To accomplish this objective, the protocol was implemented by providing a set of methods and APIs to clients in order to execute the protocol.

The second objective was to implement online and offline fair exchange protocols. The purpose of implementing online and offline fair exchange protocol within Non-repudiation Services was to reduce the involvement of TTP and give more responsibility to clients. This objective also achieved by providing clients additional sets of methods and APIs that executed each protocol. In addition, storage units were needed on both client sides in order to keep documents and exchange information. Therefore, a storage unit has been provided for each client using MongoDB. Moreover, as a result of reduce relaying on TTP, clients were needed to ensure accountability and authenticity of incoming evidence by themselves. Thus, the verification method is implemented as a service that enables clients to verify non-repudiation evidence.

The third objective was to implement proxies within the project that executes the protocol instead of clients. In order to achieve this objective the HTTP Apache server has been installed and configured to take HTTP request from clients and redirect it to resources run on tomcat server. Each protocol is implemented as a resource and has relevant resource that executes the protocol instead of clients.

The fourth objective was to provide unit tests for Non-repudiation Services. This objective achieved by providing two types of test for the projects. The first one was to test a specific code or method such as encrypting or signing. The second one was related to interactions of the protocols such as sending documents and sending receipt. Moreover, interaction tests measure the overhead of each request for each protocol that is implemented in the project so as to help the developer or client to choose between these types of protocols.

By achieving these objectives, it can be claimed that the main aim is achieved which is Design and implement RESTful Non-repudiation Services. The project is RESTful because the project respects REST constraints as following:

Client-Server: The Non-repudiation project consists of client and server interaction. In case of inline protocol, Alice makes requests to her proxy and the proxy makes a request to TTP which is a server. While, in online and offline protocols a client communicates to the server of other clients.

Stateless: All requests within the project are stateless because the application states are not stored on the server, but it only stores the resource states.

Layered System: This constraint is also implemented in the Non-repudiation services. Clients are connected to their proxies, and the proxies are connected to the server. More proxies can be added to the project in order to provide load balancing in the project.

Identification of Resources: This constraint is considered in the project in which all resources within the Non-repudiation project are identified using URI and can be accessed to by using HTTP requests.

Manipulation of Resources through Representations: This constraint is applied to the project which provides clients the ability to manipulate resources using representations.

Self-Descriptive Messages: All methods within the Non-repudiation project are implemented as self-descriptive methods. All requests contain all necessary information in order to be processed. Furthermore, all HTTP methods are used properly and without contradicting with their verb meaning. The GET method is used for retrieving information only. POST is used to create a new resource. PUT method is used to update an existing resource. These properties have been applied to all methods within the Non-repudiation Project.

HATEOAS: The Non-repudiation project enables clients to interact with different resources using hypermedia links. This means that clients can change the current state of application to another state related to the current state.

By applying REST constraints to the project and obtaining a RESTful Non-repudiation Services, the project

gains the following properties:

Scalability: The project gains high scalability. It is achieved by applying client-server constraint and decoupling the client side concerns from server side concern. The developers will integrate their business with the project quickly and easily. Secondly, the scalability of the system is increased by applying hierarchy constraint [10]. In this case, the system can be scaled by adding intermediaries to the system in order to balance the load between machines on the network as well as add more functionality such as implement different types of fair non-repudiation protocols.

Simplicity: Another advantage of RESTful web services is simplicity. Clients' requests are stateless [10]; therefore, the system does not require managing and maintaining requests state across multiple client requests. Moreover, request messages are fully descriptive and can be understood by intermediaries and the ultimate service.

Generality of Interfaces: Applying uniform interfaces constraints provide the Non-repudiation project generality of interfaces. Generality of interface also enables the system to be used by other developers and integrate the project with their system since all methods are descriptive and applied properly according to their vocabulary meaning.

7. Further Work

The Non-repudiation Services project has been designed and implemented as RESTful web services. However, the project includes only two clients; Alice who acts as a sender and Bob as a receiver. One possible future work might be implementing the project for an unlimited number of clients. This implementation requires providing registration service. Moreover, it requires considering about exchanging of public keys in which a client will need to exchange the public key with TTP as well as exchange public key with other clients.

Another work to do in the future is that provide authentication in the project. Authentication means that validating of the identification of a client who wants to use the Non-repudiation Services [20]. Identification means that a client announces who she/he is, and the authentication means a client proves that who she/he claimed to be.

There are variant protocols used to provide authentication for RESTful web services which are basic authentication, digest authentication and OAuth authentication [18, 20].

Another security issue with the Non-repudiation project that can be implemented in the future is authorization. After providing authentication for clients, the project requires to provide authorization. Authorization is to make sure that an authenticated client is permitted to access to a particular resource or not [20].

In the future, the job of proxies can be improved. The proxies can be improved by providing the non-repudiation for arbitrary HTTP requests that may happen rather than providing the non-repudiation for only a document. This approach may help to provide authorization for clients because clients will sign each request that may send to the TTP. For instance, if Bob wants to retrieve a non-repudiation of origin of a document, he requires signing a GET request which is used to retrieve the NRO, then send it to TTP then TTP will check whether Bob is authorized to access make this request or not. In this case, each interaction between clients and TTP cannot be denied.

Another improvement can be applied to the project in the future by providing automatic exchanges. Automatic exchanges mean that when Alice sends a document to Bob, her proxy does the remaining of interaction instead of her. The same thing is applied to Bob, when he accepts Alice's offer, his proxy will complete the rest of interactions instead of him such as sending a non-repudiation of receipt, receiving non-repudiation of origin and the document. This approach will reduce clients' interactions in which the sender only requires to make a send request and the receiver only has to accept the offer or reject. In this case, clients will interact with one another easily and simply. This approach could be achieved by involving a notification service in the project such as Amazon Notification Service (SNS) [34].

A possible scenario of Alice and Bob using inline protocol might be as follows:

- Alice makes a POST request to her proxy in order to generate non- repudiation evidence, timestamp signature and set a timer for abort. Then send the document with this information to TTP and wait. If

the exchange does not succeed before time out, then the proxy will send abort request to TTP.

- After TTP receives Alice's request, it should send a notification to Bob's Proxy and sends exchange information such as a hash of the signature and timer to the proxy.
- Bob's proxy also sends a notification to Bob informing him about the exchange.
- In this case, either Bob will accept the offer from Alice or reject it. If he rejects the offer, his proxy will send an abort request to TTP.
- Whereas, if he accepts the offer, his proxy will provide non-repudiation of receipt then send it to TTP.
- TTP will check the timer. If it is still valid, it will change the status of the exchange to succeed and send a notification to both proxies in order to inform them that the exchange has succeeded.
- Alice's proxy will retrieve the receipt then send a notification to Alice to inform her that the exchange has succeeded. The notification may contain a link to show the receipt.
- Similarly, Bob's proxy will retrieve the document and relevant signature and send a notification to Bob in order to inform him that the exchange has succeeded. The notification may contain a link to the document as well as a link to the signature.

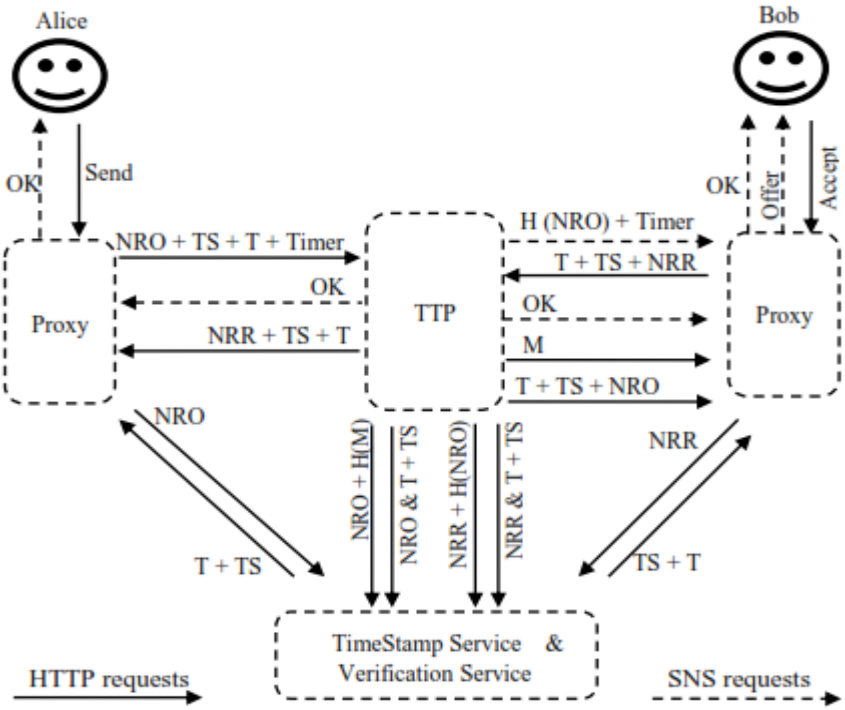

Figure 8. Non-repudiation Services with SNS

8. Conclusion

The aim of this project was to design and implement a RESTful Non-repudiation Service. In order to achieve this aim, the project was divided into four objectives. The first objective was to implement inline fair exchange protocol which was introduced by Coffey and Saidha. The second objective was to implement online and offline fair exchange protocols. The third one was to implement proxies in order to execute the protocols on behalf of clients and keep them away from the actual implementation. The final objective was to provide unit testing for the project. In order to check the project is working properly. There were two types of test; one of them to test unit codes and check whether it works correctly or not. The other test was to test the interactions between clients, in which this test unit checked all requests that can be made by clients.

By implementing these objectives successfully, the main aim has been achieved by designing and implementing RESTful Non-repudiation Services. The project was designed without contradicting with any of REST constraints; therefore, the scalability and simplicity have been gained.

However, some issues could be solved if more time was dedicated to the project such as providing a user registration service, authentication and authorization. In addition, a suggestion has been proposed to implement proxies in order to provide non-repudiation for arbitrary HTTP requests. Moreover, an approach has been proposed in order to reduce the client's interactions and the complexity of the protocol executing by involving SNS in the project. These issues with proxy improvements have been left to the future..

9. Acknowledgment

I would like to thank the Ministry of Higher Education of Kurdistan for providing me a scholarship to study MSc in United Kingdom. Also, I would like to thank my supervisor Dr. Nick Cook for his invaluable suggestions that helped me to complete the project. Finally, I would like to thank my parents for unlimited support and love during my entire education.

10.References

[1] Jianying, Z. and D. Gollmann. An efficient non-repudiation protocol. in Computer Security Foundations Workshop, 1997. Proceedings., 10th. 1997.

[2] Zhou, J., Non-Repudiation in Electronic Commerce. 1st edition ed2001: Artech House.

[3] Kremer, S., O. Markowitch, and J.Y. Zhou, An intensive survey of fair non-repudiation protocols. Computer Communications, 2002. **25**(17): p. 1606-1621.

[4] Bo, M., W. Shaomei, and X. Qianxing. A fair non-repudiation protocol. in Computer Supported Cooperative Work in Design, 2002. The 7th International Conference on. 2002.

[5] Cook, N., P. Robinson, and S.K. Shrivastava, Design and implementation of Web services middleware to support fair non-repudiable interactions. International Journal of Cooperative Information Systems, 2006. **15**(4): p. 565-597.

[6] Liew, C.C., et al. Non-repudiation in an agent-based electronic commerce system. In Database and Expert Systems Applications, 1999. Proceedings. Tenth International Workshop on. 1999.

[7] Coffey, T. and P. Saidha, Non-repudiation with mandatory proof of receipt. Computer Communication Review, 1996. **26**(1): p. 6-17.

[8] Jianying, Z. and D. Gollman. A fair non-repudiation protocol. in Security and Privacy, 1996. Proceedings., 1996 IEEE Symposium on. 1996.

[9] Asokan, N., V. Shoup, and M. Waidner. Asynchronous protocols for optimistic fair exchange. in Security and Privacy, 1998. Proceedings. 1998 IEEE Symposium on. 1998.

[10] Fielding, R.T., Architectural Styles and the Design of Network-based Software Architectures, in Information and Computer Science2000, UNIVERSITY OF CALIFORNIA, IRVINE.

[11] Dambal, V. A simple approach to REST-enable Java Business Services. 14 Jun 2010 01/05/2012]; Available from: http://www.ibm.com/developerworks/webservices/library/wsRESTservices/.

[12] Oracle.com. Introduction to RESTful Web Services and Jersey. 2010 15/05/2012]; Available from: http://docs.oracle.com/cd/E19776-01/820-4867/ggnyk/index.html.

[13] Tilkov, S. A Brief Introduction to REST. Dec 10, 2007 01/05/2012]; Available from: http://www.infoq.com/articles/rest-introduction.

[14] Foundation, T.A.S. Proxy Support HOW-TO. 1999-2012 10/05/2012]; Available from: http://tomcat.apache.org/tomcat-7.0-doc/proxy-howto.html.

[15] Foundation, T.A.S. Reverse Proxy HowTo. 1999-2012 20/4/2012]; Available from: http://tomcat.apache.org/connectors-doc/generic_howto/proxy.html.

[16] niq. Running a Reverse Proxy in Apache. 2009 Fri Dec 25 15:37:50 2009 20/04/2012]; Available from: http://www.apachetutor.org/admin/reverseproxies.

[17] Richardson, L. and S. Ruby, RESTful Web Services. 1 edMay 2007, United States of America: O'Reilly Media.

[18] Allamaraju, S., RESTful Web Services Cookbook. 1 edMarch 2010: O'Reilly Media / Yahoo Press.

[19] Masse, M., REST API Design Rulebook. 1 edOctober 2011, United States of America: O'Reilly Media.

[20] Burke, B., RESTful Java with JAX-RS. 1 edNovember 2009, United States of America: O'Reilly Media.

[21] Peiris, H., L. Soysa, and R. Palliyaguru. Non-Repudiation Framework for E-Government Applications. in Information and Automation for Sustainability, 2008. ICIAFS 2008. 4th International Conference on. 2008.

[22] 10gen. MongoDB. 30/072012]; Available from: http://www.mongodb.org/.

[23] Chodorow, K. and M. Dirolf, MongoDB: The Definitive Guide. 1 edSeptember 2010, United States of America: O'Reilly Media.

[24] 10gen. What is MongoDB? 20/05/2012]; Available from: http://www.10gen.com/what-is-mongodb.

[25] Orend, K., Analysis and Classification of NoSQL Databases andEvaluation of their Ability to Replace an Object-relationalPersistence Layer, 2010, Technical University of Munich: Munich.

[26] MongoDB.org. Querying. 30/07/2012]; Available from: http://www.mongodb.org/display/DOCS/Querying.

[27] MongoDB.org. Indexes. 30/07/2012]; Available from: http://www.mongodb.org/display/DOCS/Indexes.

[28] MongoDB.org. Multikeys. 30/07/2012]; Available from: http://www.mongodb.org/display/DOCS/Multikeys.

[29] MongoDB.org. Sharding Introduction. 30/07/2012]; Available from: http://www.mongodb.org/display/DOCS/Sharding+Introduction.

[30] Oracle.com. Java ™ Cryptography Architecture (JCA) Reference Guide. 25/04/2012]; Available from: http://docs.oracle.com/javase/6/docs/technotes/guides/security/crypto/CryptoSpec.html.

[31] 31. Schneier, B., Secrets and Lies: Digital Security in a Networked World2000, United States of America: Wiley.

[32] Anderson, R.J., Security Engineering: A Guide to Building Dependable Distributed Systems. 2 ed2008, United States of America: Wiley.

[33] Grant, W., M. Majczyk, and V. Skyttä. Portecle. 10/05/2012]; Available from: http://portecle.sourceforge.net/.

[34] Amazon.com. Amazon Simple Notification Service. 10/08/2012]; Available from: http://aws.amazon.com/sns/.