

Journal of International Technology and Information Management

Volume 28 | Issue 3


Article 2

2019

A Multilayer Secured Messaging Protocol for REST-based Services

Idongesit Efaemiode Eteng
Dr, ideteng@unical.edu.ng

Follow this and additional works at: <https://scholarworks.lib.csusb.edu/jitim>

 Part of the [Business Intelligence Commons](#), [Communication Technology and New Media Commons](#), [Computer and Systems Architecture Commons](#), [Data Storage Systems Commons](#), [Digital Communications and Networking Commons](#), [E-Commerce Commons](#), [Information Literacy Commons](#), [Management Information Systems Commons](#), [Management Sciences and Quantitative Methods Commons](#), [Operational Research Commons](#), [Science and Technology Studies Commons](#), [Social Media Commons](#), and the [Technology and Innovation Commons](#)

Recommended Citation

Eteng, Idongesit Efaemiode (2019) "A Multilayer Secured Messaging Protocol for REST-based Services," *Journal of International Technology and Information Management*. Vol. 28 : Iss. 3 , Article 2.
Available at: <https://scholarworks.lib.csusb.edu/jitim/vol28/iss3/2>

This Article is brought to you for free and open access by CSUSB ScholarWorks. It has been accepted for inclusion in *Journal of International Technology and Information Management* by an authorized editor of CSUSB ScholarWorks. For more information, please contact scholarworks@csusb.edu.

A Multilayer Secured Messaging Protocol for REST-based Services

ETENG Idongesit E.

Department of Computer Science

University of Calabar

P. M. B. 1115

Calabar

Email: ideteng@unical.edu.ng, idongesitessien@yahoo.com

OLUFEMI Oluwaseun O.,

Department of Computer Science

University of Calabar

P. M. B. 1115

Calabar

Email: femisheun@gmail.com

ABSTRACT

The lack of a descriptive language and security guidelines poses a big challenge to implementing security in Representational State Transfer (REST) architecture. There is over reliance on Secure Socket Layer/Transport Layer Security (SSL/TLS), which in recent times has proven to be fallible. Some recent attacks against SSL/TLS include: POODLE, BREACH, CRIME, BEAST, FREAK etc. A secure messaging protocol is implemented in this work. The protocol is further compiled into a reusable library which can be called by other REST services. The library can be reused by .NET applications and the implementation steps can also be followed by other REST services developers using other platforms.

Keywords: protocol, secure messaging, encryption, signature, confidentiality, message integrity,

INTRODUCTION

Many of today's businesses now rely on the web of intelligent services that can interface with other services and applications; desktop, web and even mobile devices. Each of these services and applications may handle different portions of data storage and business rules and may reside on different machines dispersed across networks all over the globe. The Web Service concept offers a way to unite information distributed between these critical business-centric applications, regardless of the hardware, operating system, language and platform for which the applications are developed. The two leading protocols for web services are; SOAP and REST. SOAP uses Extensible Mark-up Language (XML) for data exchange while REST uses XML, JavaScript Object Notation (JSON) and a host of other data formats for data exchange, prominent among which is JSON. REST is also more lightweight than SOAP. While SOAP is still retained by some prominent technology companies such as PayPal, Docusign and Salesforce, REST is quickly winning out and now represents over 70% of public APIs (Hunsaker, 2015).

In spite of the increase in the growth of popularity and the rate of adoption of REST services over SOAP web services, the issue of security in REST services has not been adequately addressed. Developers are left to implement their own security protocols. Since there are no standard security guidelines to follow, and the fact that REST-based web services are easier to implement than their SOAP equivalent, developers tend to haphazardly get their web services deployed in a hurry without taking cognizance of the security implications of their work; thereby, exposing their applications to serious vulnerabilities. According to Symantec (2013), web applications are getting attacked more frequently. In the past five years, there have been various application layer attacks, targeted at web services such as: Denial of Service (DoS), Unauthorized Access, Extensible Markup Language (XML) injection, Broken Authentication, Cross Site Scripting (XSS), Distributed Denial of Service (DDoS), Cross-Site Request Forgery (CSRF), Message Delay/Replay attacks etc.

The approach proposed in this paper is to implement a multi-layer approach which includes a series of cryptographic protocols, secure hashing, certificate parsing, certificate generation and some other protocols. The first layer of the security protocol is to implement an SSL/TLS while transporting the message over the wire while the second layer is to implement an end-to-end protection of the message which acts as a fallback layer if the security of the first layer is ever compromised. This paper therefore aims proffering an end to end security solution for

Representational State Transfer (REST) –based web services. Our key contributions in this paper include:

- A message signing and signature verification algorithm using “http signatures”
- A message encryption and decryption module using Advanced Encryption System (AES) and Rivest Shamir Adleman (RSA) algorithms.
- A nonce-based system for preventing Delay attacks.
- An SSL/TLS secured channel of communication using StartSSL.
- A payment collection Application Programming Interface (API) for Students’ Fees payment (service provider).
- An Interface that models the core banking posting platform of banks (service consumer). This service consumer accepts payments information from the service provider, integrating the security protocol developed.

All contributions are described in the Methodology section.

RELATED WORK

This section reviews works previously done regarding the security protocol for RESTful Application Programming Interfaces (APIs).

Serme et al., (2014) developed a security protocol modeled after Web Services Security (WS-Security). WS-Security is an extension to SOAP based Web Services that make use of such standards as Extensible Markup Language (XML) Encryption and XML Signature to provide an end-to-end security (Soldano, 2015). In addition to designing the security protocol, a performance evaluation comparison was done between the protocol and the WS-Security. Their result showed that the encryption was slower than signature for SOAP based WS-Security while their security protocol showed a better performance time for encryption than signature for small amount of data while the signature became faster than encryption when the size of data was large. By way of comparison, their work makes use of a third party certificate to provide Non-Repudiation property which comes at an additional cost. Certificate Authorities can sometimes be a burden as they can be vulnerable to attackers. When they are compromised, they can be forced to issue false certificates (Zeter, 2011). False certificates can be used to impersonate one of the parties involved in a transaction in order to get information. The protocol used in their work is called Sign – then – encrypt. It

means the sender signs the payload and then encrypts it before sending it to the recipient. The recipient then decrypts the message before verifying the signature. This process has an inherent disadvantage of the recipient having to decrypt all messages, even the ones that do not pass integrity test.

The end-to-end protocol described in this paper combines a unique ID generated for each client with the digital signature to ensure Non-repudiation. In addition, this work makes use of Secure Hash Algorithm-512 (SHA512) cryptographic primitive as the hash algorithm for ensuring the integrity of the message. The work described in this paper makes use of an encrypt – then – sign protocol, which means payloads are first encrypted before being signed at the sender end. This means information must pass the integrity test at the receiver end before the information is decrypted. The advantage of this protocol is that messages that would not pass integrity test are not wasted time on being decrypted as designed in the work of (Serme et al., 2014). Furthermore, even though the end-to-end protocol can function in the absence of SSL/TLS, it relies on the SSL/TLS as an additional fall back layer in case the protocol is breached.

Sporny, (2013) described a simple messaging protocol for web payments. His work is a draft submitted to the World Wide Web Consortium (W3C). That great work provides detailed guideline protocol for securing web applications but the work has no implementation beyond the fact that it serves as a guide. Moreover, the approach does not target REST specifically. This paper not only implements the protocol but also implements it as a re-usable library for the .NET and Java developers.

Mohamed & Mohamed, (2014) highlighted the security flaws of RESTful services in their work. While their work did not implement any specific security protocol, they did mention some security implementations for RESTful services; such as TLS plus Message Digest and Open standard for Authorization (OAuth). While OAuth1.0 is deemed a success, OAuth 2.0 was more complex, less interoperable and less secure.

Another popular approach to securing RESTful services is using “secure” Hypertext Transfer Protocol (HTTPS). (Lee & Mehta, 2013) presented a threat model to RESTful applications. They showed how RESTful services could be vulnerable to JSON input attack. JSON input attack provides a window of opportunity for attackers to execute malicious scripts. They showed how encryption via HTTPS could be used to mitigate input attacks. While HTTPS could mitigate input attacks, recent research shows that HTTPS is not infallible. Recent attacks on HTTPS include POODLE, CRIME, BEAST, BREACH, etc. (Meyer & Schwenk, 2013), (Kovacs, 2015). The approach in this paper implemented an end-to-end layer in addition to the TLS in order to harden the security.

Lenka & Nayak (2014) published a great paper on how to enhance data security in the cloud. They proposed a model that uses RSA for encryption and RSA with MD5 for authentication and integrity. Their choice of RSA to encrypt message directly is not great because RSA is an asymmetric algorithm and therefore very slow and very costly for encrypting large data. Their choice of RSA with MD5 for signature makes the signature scheme weak because MD5 is already broken (Sasaki & Aoki, 2009), (Mao et al, 2009) and (Ness, 2012). Since, asymmetric signature algorithms require that the hash function used be collision resistant (Sotirov, 2008), combining RSA with MD5 for signature would therefore make the resulting signature scheme weak. On the contrary, the signature scheme proposed in this work makes use of HMAC SHA-512 signature scheme. Apart from the fact that SHA512 is stronger than MD5, HMAC, unlike other asymmetric signing algorithms like RSA, DSA, ECDSA etc., is more resistant to collisions than their underlying hashing algorithms (Kim, 2006).

Similarly, (Fashoto et al., 2010) published an authentication model which uses RSA with MD5 for authentication and integrity check. All the flaws pointed out in the work of Lenka & Neyak (2014) applies to their work as well. This paper uses HMAC with SHA512 for authentication and integrity. The protocol also incorporates confidentiality.

Dudhe & Sherekkar (2014) carried out a performance analysis of SOAP and REST-based web services. They conducted different experiments on both and concluded that REST had a better performance than SOAP.

PROBLEM STATEMENT

Prob 1: A lack of standardized protocol for RESE-based web services.

Prob 2: The rigour of developers having to build security protocols into their applications.

METHODOLOGY

The paper addresses the key security features; Authentication, Authorization, Confidentiality, Integrity and Non-Repudiation as well as some well documented attacks like Replay attacks, POODLE, BREACH, CRIME etc. The security protocol employed a two-layer approach.

1. At the first layer was a protocol based approach, SSL/TLS over HTTP. HTTP is the de-facto communication protocol for the web. SSL/TLS over HTTP, which is represented as “https”, indicates that a secure channel of communication exists between the server’s web server and the client’s browser. The “s” appended to “http” indicates SSL/TLS. This layer was configured with strong cryptographic primitives like the AES256 GCM, ECDHE, SHA256, etc. If this layer is compromised, the second layer forms another protective layer over the message.
2. The second security layer describes an orderly set of steps to ensure authentication, integrity and encryption of the message. The end product of this layer is a reusable library. A high level overview of the protocol is as described below;
 - a. The receiver generates an asymmetric public key pair, publishes a message ID and public key of the key pair in a key management service
 - b. The receiver generates a symmetric key and an application ID for each client, stores the information in the database and securely transports the key and the ID to the client. The mode of transportation could be through an encrypted email other secure means.
 - c. The sender generates an AES symmetric key, encrypts the data to be sent to the receiver with this symmetric key using the AES algorithm
 - d. The sender uses the RSA public key of the receiver to encrypt the AES symmetric key
 - e. The sender digitally signs the encrypted message using the HMAC authentication scheme
 - f. The sender verifies the signature for message integrity, replay and timing attacks
 - g. The sender decrypts the encrypted cipher

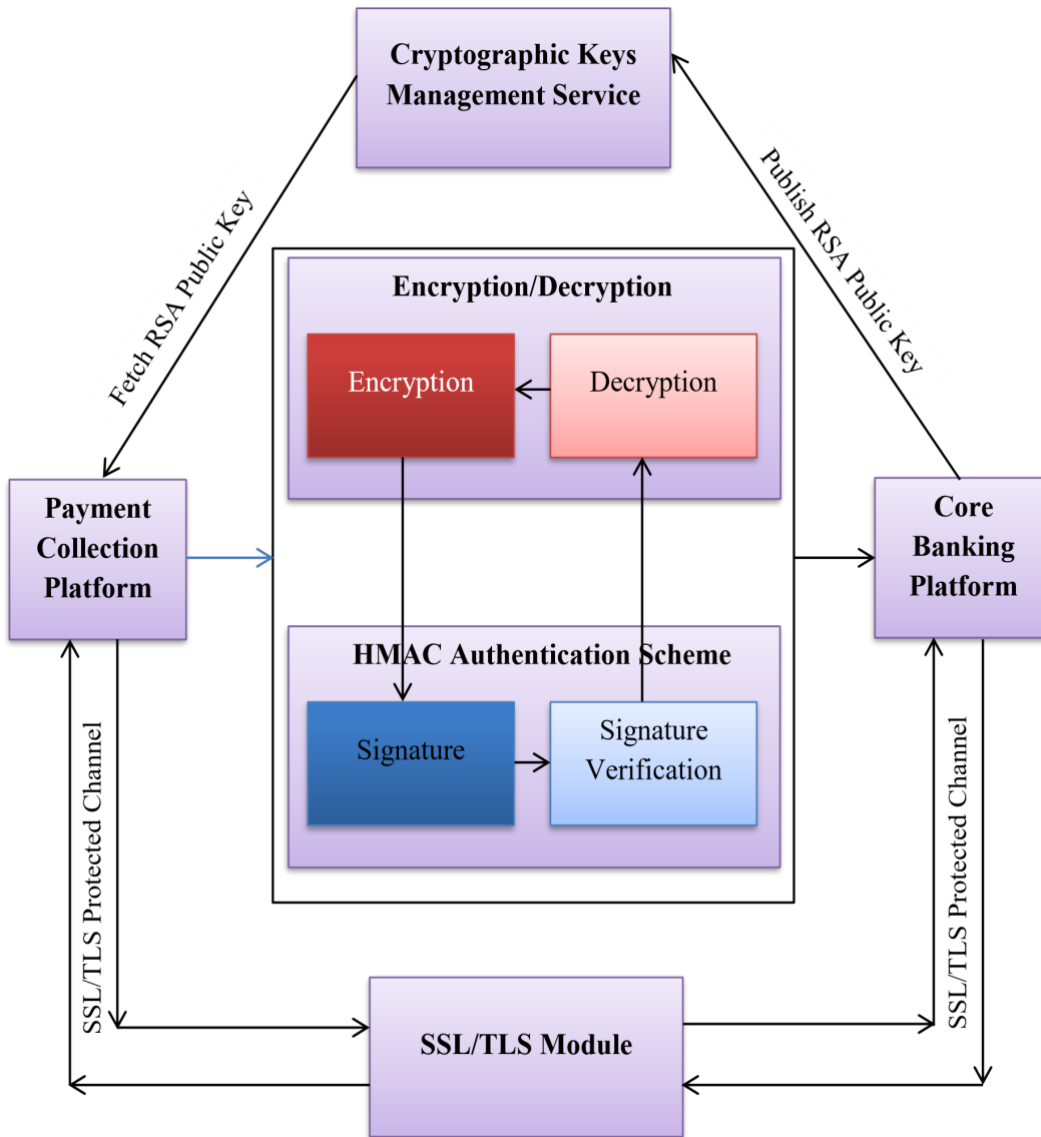
Development Model Used

The paper adopted a variant of the Feature Driven Development (FDD) methodology. FDD was devised by Jeff De Luca in 1997 and is one of the agile methods of application development. The project was split into the following sub-domains/major parts; and the interaction amongst the subdomains is depicted in the diagram of Figure 1.

1. Cryptographic Keys Management Service

2. HMAC Authentication Scheme
3. Encryption and Decryption
4. SSL/TLS

Figure 1: Basic Interaction Model amongst modules



Cryptographic Keys Management Service

The key management service administers cryptographic keys used for the message encryption and the symmetric keys used for digital signature. For the asymmetric key service, which involves a key pair (public and private), the public key is published on a service which is accessed through the secure channel (https) by all intending clients. The corresponding private key is saved in the configuration file and the configuration file is encrypted on the machine. For the signature scheme, an Application ID and symmetric key pair is generated for each client and stored in the database. These parameters (Application ID and Symmetric Key), alongside other parameters like nonce, timestamp and Uniform Resource Locator (URL) of the service are used to construct the signature.

Encryption and Decryption

This process makes use of the keys produced by the key management for its implementation. The encryption/decryption module is summarized below:

1. The plain text is encrypted with the AES256 symmetric encryption key.
2. The public key of the RSA is used to encrypt the AES symmetric key.
3. The recipient, which holds the corresponding private key of the RSA algorithm, uses the private key to decrypt the RSA cipher in order to obtain the AES symmetric key.
4. The AES symmetric key is then used to decrypt the AES encrypted cipher of step 1. The result of this decryption process is the plain text.

Figures 2 and 3 represent the activity diagrams for the encryption and decryption processes respectively

Figure 2. Activity Diagram for the Encryption Process

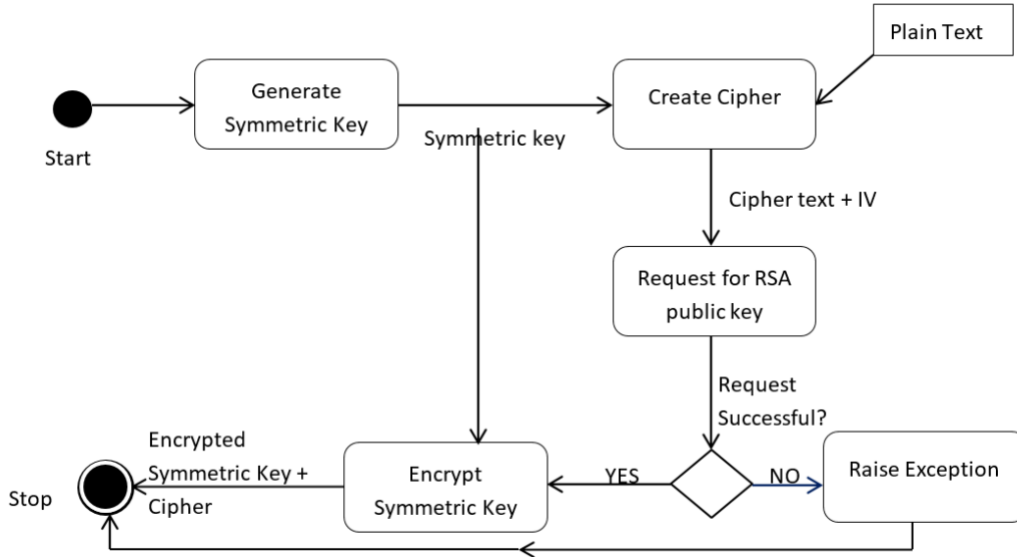
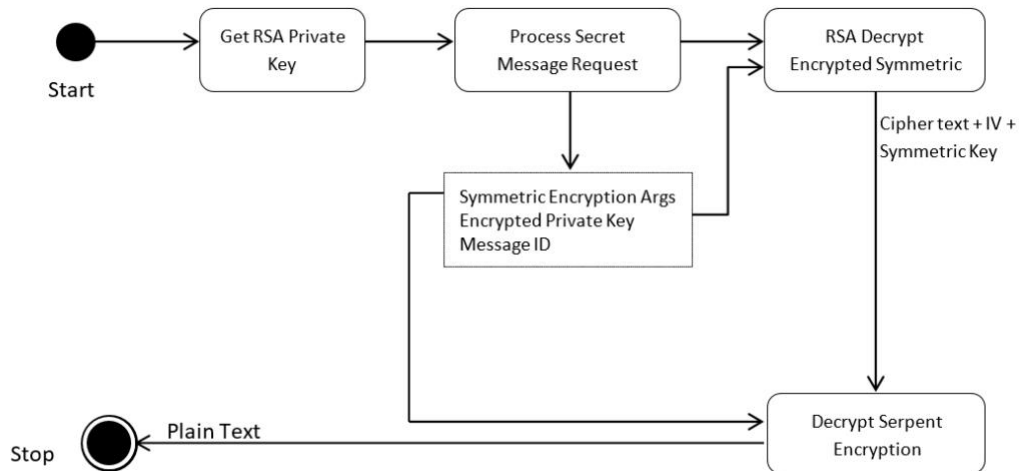


Figure 3: Activity Diagram for the Decryption Process

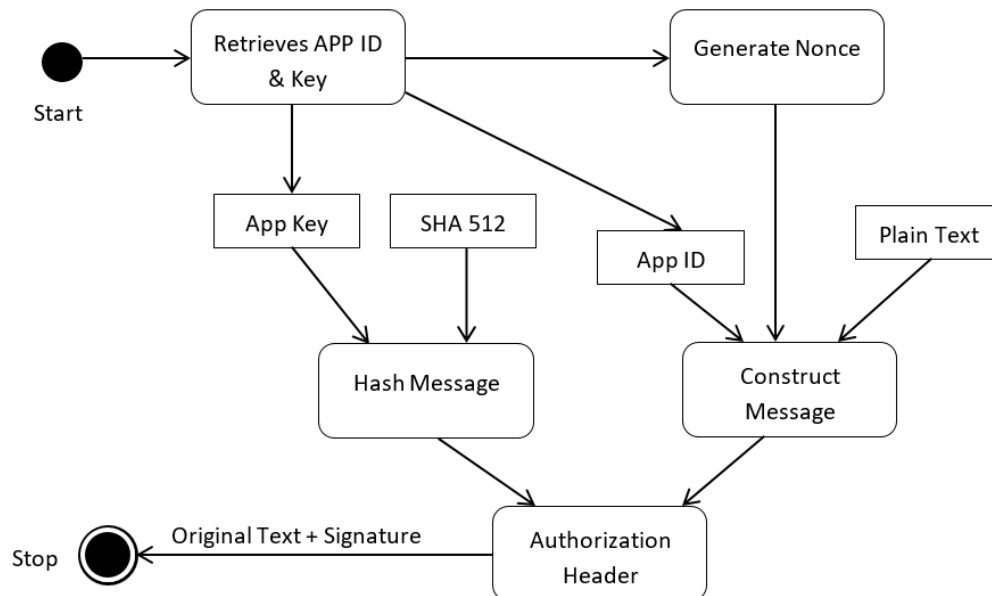


HMAC Authentication Scheme

This process makes use of the keys produced by the key management service for its implementation. The flow on the sender’s side is described below and illustrated by the activity diagram of figure 4:

1. The sender retrieves the Application ID and the Application key from a secure location.
2. The sender generates a unique ID, known as nonce and a timestamp.
3. The sender constructs a message to be transmitted to the receiver by concatenating the following parameters; Application ID, HTTP request method (POST, GET, PUT and DELETE), Uniform Resource Identifier (URI), time stamp, nonce, and payload. The payload is the actual message to be sent and is Base 64 encoded.
4. The unique digital signature for this message is generated by hashing the message constructed from the previous step using (SHA512) and the API Key retrieved from a secure location.
5. This digital signature is concatenated with the Application ID, the nonce and the timestamp generated in step 2 and this concatenated string is then sent together with the message as constructed in step 3, through the Authorization header of the request using the format: **[Authorization: fem APPID\nSignature\nNonce\nTimestamp]**

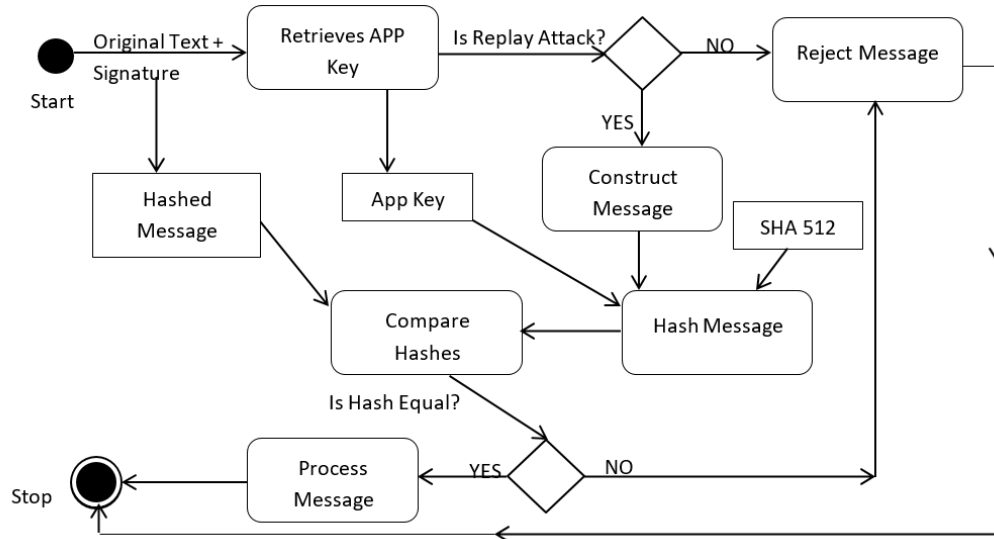
Figure 4. Activity Diagram of the Signing Process (Sender's Flow)



The flow on the receiver's end is described below and illustrated by the activity diagram of figure 5:

1. The receiver accepts all the data as sent by the message sender and retrieves the following parameters from the authorization header (APPID, Signature, Nonce and Time stamp).
2. The receiver retrieves the API key that corresponds to the APP ID from the secure repository.
3. If the APP ID exists in the repository, the receiver of the message then validates if the request is a replay request and rejects it, therefore preventing the API from any replay attacks. The time stamp alongside the unique nonce generated in step 2 of the client's flow is used to nullify the replay attacks. The receiver checks if the nonce has been used within certain acceptable time bounds, i.e. in this case, three minutes.
4. The receiver then rebuilds the message received from the sender by adhering to the same order the parameters were fed into the signature algorithm at the sender end. The parameter order and the encoding format must be the same as in the sender application. This parameter order and encoding format are communicated to the client/sender of the message is always made formal using proper documentation.
5. The receiver hashes the string generated in step 4 above by employing the same SHA512 algorithm used by the sender and the same API Key obtained from the key management service.
6. The resulting hash function, which is the signature generated by the receiver is compared to the one sent by the sender, if the hashes are the same, it means the message originates from who the sender claims to be since the key used belongs to the APP ID for the sender (Authentication). It also means the message has not been tampered with while in transit (Integrity).
7. The sender then processes the request, otherwise the request will not be accepted and HTTP status code "401", which means "Unauthorized", will be returned.

Figure 5: Activity Diagram of the Signature Verification Process (Receiver's Flow)



SSL/TLS

The handshaking procedure for communication between the sender and receiver was designed. Microsoft's SChannel software library was employed for the SSL/TLS implementation. Outlined below were the steps taken to design the SSL/TLS security protocol layer.

- The public/private key pair was generated.
- A Certificate Signing Request (CSR) was created from Internet Information Service (IIS) using the 2048-bit keys generated.
- A 256-bit certificate was acquired from "AlphaSSL". □ IIS was then configured to enable SSL.
- SSL/TLS configuration was then hardened to make it resistance to SSL/TLS attacks such as BEAST, CRIME, POODLE, BREACH etc.

SYSTEM TESTING

The three categories of tests carried out were:

- Penetration Test – this test category carries out penetrative tests on the protocol and specifically tests for vulnerabilities to some attacks.
- SSL/TLS Configuration Test – this test category assesses the strength of the SSL/TLS configurations.
- Implementation Test – The test was carried out to ascertain that the security protocol was implementable.

Penetration Test

Penetrative Tests were conducted on the server application implemented with the model to determine how resistant it is to unauthorized access, and to determine whether the encryption would be broken if the authentication scheme is breached. The reason for conducting the penetrative test on the service was to evaluate the level of security of the service built with the end-to-end layer (2nd Layer) of the security protocol. This test models the behavior of hackers.

The aim of this test is to determine whether or not the service endpoints, protected with HMAC Authentication scheme, can be attacked.

Test Tools for the Penetration Test

REST has no descriptive document like the WSDL which normally details the entry points to the service; it is therefore difficult to test RESTful applications. However, some tools exist that are often used to simulate attacks on REST services. An example of such tools is the “Open Web Application Security Project Zed Attack Proxy” (OWASP ZAP). Penetrative tests were carried out using the following tools:

- OWASP ZAP version 2.4.3
- Fiddler Web Debugger version 4.6.2.0

OWASP Zed Attack Proxy (ZAP) is an open source desktop application. It is an integrated testing tool that is used to find vulnerabilities in web application. “toolswatch.org” awarded it the top security tool in 2013 and is becoming a framework for advanced security testing. It has a feature known as “Plug-n-Hack” which is a plugin for Mozilla web browser that enables the application to listen to traffic on Mozilla browser.

Fiddler Web Debugger is a desktop application developed by “Telerik”. It is used to capture network traffic and debug web applications.

SSL/TLS Test

The last test carried out on the protocol was the SSL/TLS test. The purpose of this test is to evaluate the strength of the SSL/TLS. The core banking server was subjected to this test. The test examines the SSL/TLS configuration for the server for known SSL/TLS vulnerabilities such as POODLE, BEAST, BREACH, CRIME, etc. The tool used for testing the SSL/TLS layer is the “Qualys SSL Lab”. It is a web based application used to evaluate the SSL configuration of servers that implement SSL/TLS across the internet. SSL Lab also extends its API for use by many other SSL/TLS testing platforms.

The aim of this test is to evaluate the server’s SSL/TLS configuration and to examine it for known vulnerabilities such as BREACH, POODLE, BEAST, etc.

Test Tools for the SSL/TLS Test

The SSL/TLS test was carried using Qualys SSL Lab. It is a free web based assessment tool for analyzing the SSL/TLS configuration of any SSL server on the Internet. It is a collaborative effort of top cloud security experts such as Ivan Ristic, Christian Folini and others. Since its launch in 2009, it has been used to scan various servers. SSL Lab examines four major parameters on the SSL/TLS configured server and grades the server based on these parameters. The four major parameters on which the server configuration is evaluated are: Certificate, Protocol Support, Key Exchange and Cipher Strength. Of these four parameters, only Protocol Support, Key Exchange and Cipher Strength are used for the overall grading. TABLE 1 shows the metrics for grading the result while TABLE 2 shows the examined parameters and the percentage contribution of each of the parameters to the overall grade.

Table 1: Letter grade translation (Adapted from Ristic, 2014)

Numerical Score	Grade
Score \geq 80	A
Score \geq 65	B

Table 2. Category Criteria Rating Guide (Risstic, 2014)

Category	Score (%)
Protocol Support	30
Key Exchange	30
Cipher Strength	40
Score \geq 50	C
Score \geq 35	D
Score \geq 20	E
Score $<$ 20	F

Implementation Test

This test was carried out by using the two applications that were developed to implement this protocol. In this test, the model core banking posting platform, which has the two end points, was deployed on one machine while the other application, the payment collection API, was deployed on another machine.

The payment collection API sends fifty (50) payment records through a “POST” http method to the message service of the core banking service. The payment collection API also retrieves all fifty records in one single “GET” request. The payment collection API presents the HMAC scheme for the fifty posts.

RESULTS & DISCUSSION

We recall that two major problems were identified earlier in this paper as Prob 1 and Prob 2. This paper solves those problems by offering a twofold result.

Firstly, the following artifacts were developed: A message encryption and decryption module, a message signing and signature verification scheme using HMAC, an SSL/TLS secured channel of communication was set up, a payment collection application programming interface (API) for Students’ Fees payment (service provider) was developed, a model core banking posting platform was developed. **Secondly**, results from testing modules and their workability were also documented.

Using Feature Driven Development (FDD) software methodology, a two layer security protocol was developed. The first layer is a well hardened SSL/TLS configuration. The second layer is a well-designed end-to-end protocol that handles authentication, authorization, encryption and message integrity as well as timing and replay attack prevention. The end-to-end protocol uses HMAC-512 and a hybrid encryption system using the AES and RSA algorithms. The protocol was then compiled to a reusable library using C# language. Two different tests were carried out on this protocol: Penetration test and SSL/TLS configuration test. The Penetration Test was carried out using the Open Web Application Security Project Zed Attack Proxy (OWASP ZAP) application and Fiddler Web Debugger. The SSL/TLS test sought to test the SSL/TLS layer of the protocol for known vulnerabilities using a popular SSL/TLS test tool known as SSL Lab. The raw and scaled scores obtained from SSL Lab were 95% and 93% respectively. The results of Implementation test show that the protocol is implementable. The protocol is also resistant to such attacks as: Unauthorized, Timing and Replay attacks as shown by the result of the penetration test. The grade obtained from the SSL/TLS test is “A+”. The result also shows that the implementation is not vulnerable to currently known SSL attacks.

Below, a brief discussion of the Penetration and SSL/TLS Tests are given.

Results and Discussion of the Penetration Test

The “Open Web Application Security Project Zed Attack Proxy” (OWASP ZAP) tool was used to attack the application in a quick start mode and the Fiddler was used to analyze the captured traffic.

Result of the Penetration Test

When OWASP ZAP tool was used to attack, the tool did not break the authentication security mechanism built into the system. Error 401 was returned as shown in the diagram of FIG. 36. Similarly when Fiddler Web Debugger was used to capture the response from the server, the response code 401 was obtained. The details of the result obtained from OWASP ZAP when Mozilla was used to access the protected endpoint, and the result obtained from Fiddler Web Debugger, is as shown below:

“HTTP/1.1 401 Unauthorized

Cache-Control: no-cache

Pragma: no-cache

Expires: -1

Server: Microsoft-IIS/10.0

WWW-Authenticate: fem

X-AspNet-Version: 4.0.30319

X-SourceFiles: =?UTF-

8?B?QzpcUHVvamVjdHNcd2ViIGFwcFxBY2NlcHRvclxBY2NlcHRvci5XZWJcTWVzc2FnZQ==?=

Discussion of the Penetration Test

HTTP response code 401 indicates an unauthorized request while response code 200 indicates a success and that the requested resource is granted. Status code 401 from two testing tools, OWASP ZAP and Fiddler indicates that the protected

endpoint cannot be accessed without providing the right authentication scheme “fem” which was used to protect the endpoint.

Results and Discussion of the SSL/TLS Test

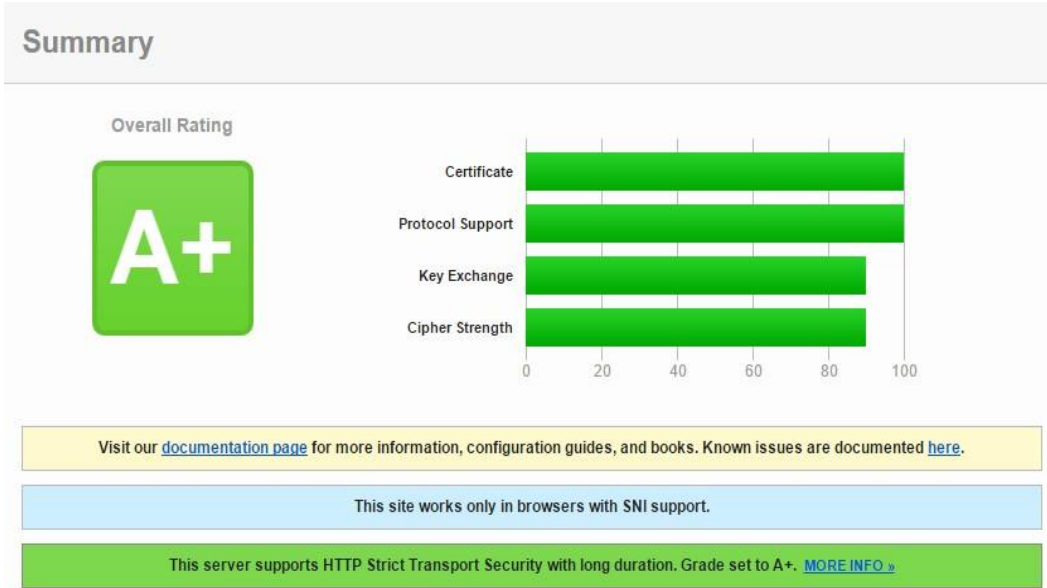
The test examines the SSL/TLS configuration for the server for known SSL/TLS vulnerabilities such as POODLE, BEAST, BREACH, CRIME, etc. As explained in chapter three, the four major parameters on which the server configuration is evaluated are: Certificate, Protocol, Support, Key Exchange and Cipher Strength.

Result of SSL/TLS Test

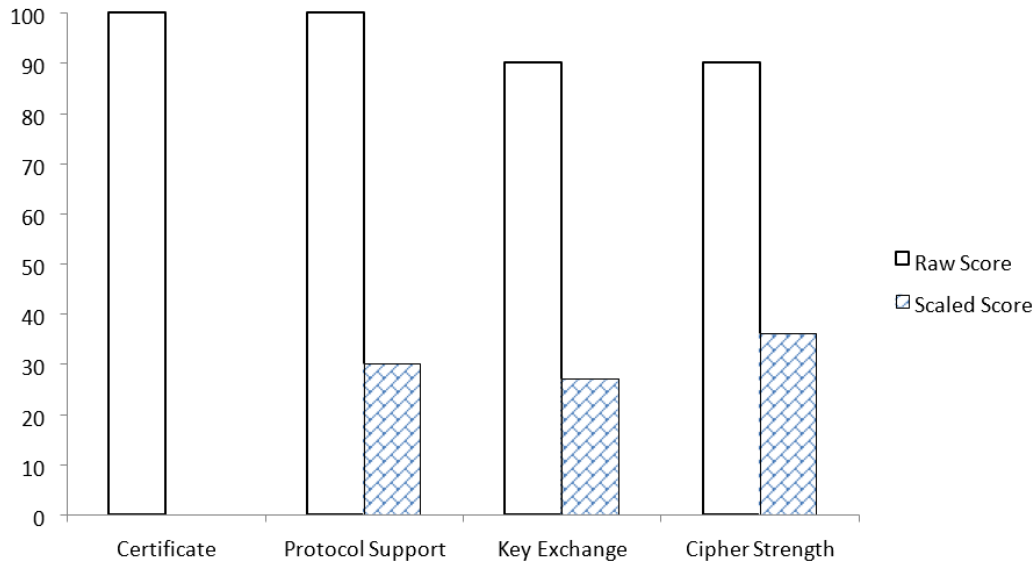
The default configuration of the SSL/TLS after installing the certificate yielded a grade B result when subjected to Qualys SSL Lab test. When the SSL/TLS configuration was hardened by ordering the cipher suites to use strong cryptographic protocols, removing weak algorithms like RC4 and providing support for Forward Secrecy, a grade A was obtained as shown in FIG. 38.

When HTTP Strict Transport Security was enabled and TLS1.0 and TLS1.1 TLS were removed, Qualys SSL Lab test yielded a grade A+. The result of the Qualys SSL Lab test on the SSL/TLS configuration is shown in TABLE 1. The transcript for the test is shown in Appendix A and from this transcript; all current SSL/TLS known vulnerabilities were avoided.

Table 3. Result of the Qualys SSL Lab test



Category	% Contribution to Overall	
	Result (%)	Grade
Certificate	100	0
Protocol Support	100	30
Key Exchange	90	27
Cipher Strength	90	36
Final Score	95	93



Discussion of SSL/TLS

The second column of TABLE 1 shows the raw result obtained from the test. The average score is 95%. The third column shows the scaled scores based on the metrics shown in TABLE 6. Based on the scaled scores on the third column, the overall results of the test is 93%. The overall grade for the test is “A” based on the metrics shown in TABLE 5. This result is an indication of an excellent configuration. The graph showing the result is shown in FIG. 39. The transcript of the entire test is shown in Appendix A and the transcript shows the server is not vulnerable to known SSL/TLS vulnerabilities such as POODLE, BEAST, BREACH, CRIME, etc.

The result also indicates that the SSL/TLS configuration supports Perfect Forward Secrecy (PFS). With PFS configuration, even if the encryption is broken and the secret key used to encrypt the session is recovered by an attacker, the attacker would not be able to decrypt previous messages sent with that key. ECDHE supports PFS; therefore, the cipher suites that contain ECDHE were moved to the top in order to ensure PFS. According to (Schum, 2014), ephemeral keys are temporary keys that are used to ensure forward secrecy. Systems that are configured to use PFS ensure better security than the ones not configured to use PFS (Higgins, 2013).

In order to obtain a grade A+ in Qualys, a property known as “TLS_FALLBACK_SCSV” must be enabled in the SSL/TLS Configuration.

TLS_FALLBACKSCSV prevents an attacker from using TLS version lower than the maximum present value in a client. Windows server was used for the deployment and IIS does not support TLS_FALLBACKSCSV; therefore, the only way to achieve a grade A+ on Qualys on windows server is by disabling TLS 1.1 and TLS 1.0. The downside of doing such is that browsers that do not support TLS 1.2 would not be able to communicate with the service. However this does not constitute much problem as most browsers today have support for TLS 1.2. Only the old ones do not support it. The SCSV part of the

TLS_FALLBACK_SCSV is an acronym for Signaling Cipher Suite Value.

TLS_FALLBACK_SCSV *"allows a browser to indicate to a server when the current connection being used or attempted is a fallback"* (Helme, 2014). This feature simply allows the browser to report to the server that it (browser) supports a better protocol than what it is currently attempting to communicate the server with. When it is present in the client handshake, the server knows that the client has a better protocol and is only being tricked or forced to use the current weaker protocol; therefore, the server will not accept the connection.

CONCLUSION

The end-to-end security layer works exactly the same way WS-Security works but without the extra burden of wrapping it in another protocol like SOAP. It implements the REST architecture fully. As shown by the results of the Implementation test, the protocol is not only implementable but also resilient to several known attacks such as replay, modification, eavesdropping, and unauthorized access from the results shown by the Penetration test. Weak encryption algorithms and hash functions such as RC4, MD5, DES and SHA-1 are avoided at both layers of the protocol because of their well-documented flaws. Perfect Forward Secrecy (PFS) is supported on this layer by generating a new cryptographic key for each encryption session. With PFS, previous messages cannot be decrypted by an attacker even if the encryption is cracked and the secret key used to encrypt the session is compromised.

The SSL/TLS, which forms another protective layer, is configured to use strong cryptographic primitives. As shown by the result of the SSL/TLS test, hardening of the SSL/TLS configuration improves the security and in turn the confidence the potential users would have in adopting this protocol.

REFERENCES

- Barnes, R. (2014). The POODLE Attack and the End of SSL 3.0. Retrieved 23, 2016, from <https://blog.mozilla.org/security/2014/10/14/the-poodle-attack-and-the-end—ssl-3-0/>
- Dudhe, A. Sherekar, S. (2014). Performance Analysis of SOAP and RESTful Mobile Web Services in Cloud Environment. International Journal of Computer Applications (0975-8887), RTINFOSEC, 1-4.
- Fashoto, S', Gbadeyan. J. & Okeyinka, A. (2010). Application of Digital Signature for Securing Communication Using RSA Scheme based on MD5. Paper presented at Proceedings of the International Conference on Software Engineering and Intelligent Systems, Ota, Nigeria.
- Helme, S. (2014). Getting an A+ rating on the Qualys SSL Test. Retrieved from <https://scotthelme.co.uk/a-plus-rating-qualys-ssl-test/>
- Higgins, P. (2013). Pushing for Perfect Forward Secrecy, an Important Web Privacy Protection. Retrieved from <https://www.eff.org/deeplinks/2013/08/pushing-perfect-forwards-secrecyinimportant-web-privacy-protection>
- Hunsaker, C. (2015). SSL: REST vs SOAP: Stormpath. When is REST better? Retrieved from <https://stormpath.com/blog/rest-vs-soap/>
- Kim, J., Biryukov, A., Preenel, B. & Hong, S. (2006). On the Security of HMAC and NMAC Based on HAVAL, MD4, MD5, SHA-0 and SHA-1. Paper presented at 5th International Conference on Security and Cryptography for Networks (SCN2006), Maiori, Italy. Doi:10.1007/11832072_17
- Kovacs, E. (2015). New Attack on RC4-Based SSL/TLS Leverages 13-Yr-Old Vulnerability. Retrieved December 23, 2015, from <http://www.securityweek.com/new-attack-rc4-based-ssltlsleverages-13-year-old-vulnerability>

- Lee, H. & Mehta, M. (2013). Defence Against REST-based Web Service Attacks for Enterprise Systems, *Journal of Communications of the IIMA*. 13(1), 57-68.
- Lenka, S. & Nayak, B. (2014). Enhancing Data Security in Cloud Computing Using RSA Encryption and MD5 Algorithm. *International Journal of Computer Science Trends and Technology (IJCST)*, 2(3), 60-64.
- Mao, M., Chen, S. & Xu, J. (2009). "Construction of the Initial Structure for Preimage Attack of MD5". Paper presented at International Conference on Computational Intelligence and Security (IEEE Computer Society), Beijing, China. Doi: 10.1109/CIS.2009.214
- Meyer, C. & Schwenk, J. (2013). Lessons Learned From Previous SSL/TLS Attacks A Brief Chronology of Attacks and Weaknesses. *Cryptology ePrint Archive* Retrieved November, 25, 2015, from <https://eprint.iacr.org/2013/049.pdf>
- Mohamed, I. & Mohamed, S. (2014b). Applying Security for RESTful Web Services – Limitations and Delimitation. *International Journal of Emerging Technology and Advanced Engineering*. 4(9), 327-330.
- National Security Agency Information Assurance Directorate[NSAIA]. (2008). Net-Centric Enterprise Services (NCES) Profile of Web Service Security: Simple Object Access Protocol (SOAP) Message Security (WSSE). Retrieved from https://www.owasp.org/index.php/Web_Services
- Ness, J., (2012). Microsoft Certification Authority Signing Certificates Added to the Untrusted Certificate Store. *TechNet Blogs, Security Research & Defense*.
- Ristic, I. (2014). *Bulletproof SSL: Understanding and Deploying SSL/TLS and PKI to Secure Servers and Web Applications*. London, United Kingdom: Feisty Duck Limited.
- Sasaki, Y., Aoki, K. (2009). *Finding Preimages in Full MD5 Faster Than Exhaustive Search* Berlin Heidelberg, Springer.

- Serme, G., de Oliveira, A. S., Massiera, J., & Roudier, Y. (2012, June). Enabling message security for RESTful services. In *2012 IEEE 19th International Conference on Web Services* (pp. 114-121). IEEE.
- Schum, C. (2014). Correctly Implementing Forward Secrecy. Presented for Gold Certification of GIAC, The Sans Institute, March, 2014.
- Soldano, A. (2015). Building the Web Service Stack of a JavaEE Compliant Server Around Apache CXF. Presented at Apachecon Europe, October, 2015, Budapest, Hungary.
- Sotirov, A., Stevens M., Appelbaum, J., Lenstra, A., Molnar, D., Osvik, D., Weger, B. (2008). MD5 considered harmful today-Creating a rogue CA certificate, Paper presented at 25th Annual Chaos Communication Congress, Berlin, Germany.
- Sporny, M. (2013). Verifiable Messaging over HTTP. Retrieved from <http://manu.sporny.org/2013/http-signatures/>
- Symantec, (2013). Internet Security Threats Reports 2014. Trends publication periodical, 19(1), 1-98
- Zeter, K. (2011). Hack Obtains 9 Bogus Certificates for Prominent Websites; Traced to Iran =. Retrieved November 11, 2015, from <http://www.wired.com/2011/03/comodo-compromise/>