Secure and Stateless RESTful Web Service Using ID-Based Encryption

Sungchul Lee¹, Ju-Yeon Jo², Yoohwan Kim³
Department of Computer Science
University of Nevada Las Vegas, Las Vegas, NV 89154
¹ Email: lees173@unlv.nevada.edu
², ³ Email: {Juyeon.Jo, Yoohwan.Kim}@unlv.edu

Abstract

REST protocol has gained great popularity as a communication in the various areas such as web service and IoT devices. However, the current RESTful web service has shortcomings in the authentication process. We developed an ID-based authentication algorithm to achieve a secure and stateless RESTful web service. ID-based authentication process allows a server to handle the client’s request by acknowledging the client’s URI rather than storing client’s state, thereby simplifying the process. In this paper we discuss the shortcomings of current authentication mechanisms, the basic concept of ID-Based Encryption (IBE), and the process of stateless secure REST protocol.

Keywords – ID-based encryption, RESTful web service, ID-based Authentication

1 Introduction

Representational State Transfer (REST)-based web services are used very commonly nowadays. Especially the extensive use of Internet of Things (IoT) has brought a wider adoption of REST protocol. IoT is useful in research as well as commercial applications. For example, the Nevada Solar Energy-Water-Environment Nexus project collects a large amount of environmental data from various remote sensors [1] (conference) using IoT devices and the data is archived at the Nevada Climate Change Portal (NCCP) [2] (conference). In the previous research, we introduced REST API for communication and web service between sensor and server [3] (journal).

RESTful web service became popular thanks to its efficiency on handling the IoT data [5] (conference) and a large number of IoT devices. The first generation web service, Simple Object Access Protocol (SOAP) presented multiple limitations when it handled various devices [4] (conference). The second generation web service, RESTful web service, has several benefits to handle the IoT as it is based on Resource Oriented Architecture (ROA) [6] (journal). Accordingly, many web service companies have shifted their API to REST from SOAP. For example, Google stopped using search API based on SOAP on December, 5, 2006 [7] (online). Also, REST occupied majority, approximately 85%, of interface in Amazon in 2003 [8] (online). As a result of this trend, the majority of IoT and web service companies are developing their web interface based on RESTful web service.

2 Features and Limitations of Rest Protocol

REST protocol based on the Resource Oriented Architecture (ROA) [9] (dissertation) can handle various models of IoT. REST includes four attributes of the ROA. The first attribute is addressability. All resources are represented by Uniform Resource Identifiers (URI). This implies that all objects in the IoT can be identified through the unique URI. Each URI must be used to handle each object. Therefore, REST uses URI as if the URI was a variable of object. Figure 1 shows the process to identify and to control an IoT device by URI. The second attribute of REST is connectedness. In the IoT, connectedness is important for self-configuring. Connectedness is created based on addressability [10] (conference). Resources have to be linked with other related resources in order to be presented to servers or users. Due to these two attributes of REST, resources and connections are well organized by URI. As a result, well-organized URI helps an IoT system to expedite self-configuration. The third attribute is homogenous interface. Resources are used by the
basic four HTTP methods such as GET, PUT, POST and DELETE. Also, there are two more additional methods such as HEAD and OPTION. Metadata can be shown by HEAD method. OPTION method is for checking available methods. Various objects utilize the same control interface by the function of homogenous interface.

Finally, the fourth attribute of REST, which is most important for this research, is statelessness. Connections made between servers and clients do not create sessions. Servers do not store the client state. So, clients have to express their current state via HTTP header and body. Requested resources are only used once. Therefore, the request does not affect the next request. As there is no session, statelessness is good for balancing server loads. When one of servers is highly loaded than does other servers, the request is passed onto other servers without creating any additional work. Moreover, other beneficial attributes of statelessness are scalability [10] (conference).

However, there are authentication problems due to statelessness of REST. As REST does not utilize sessions between server and client. The client authentication process is necessary whenever servers receive the clients’ requests. In the next session, we explain the authentication methods for RESTful web service currently used such as, HTTP Basic/Digest Authentication, and Access Token (OAuth).

3 Current Authentication for RESTful Web Service

RESTful web services are constantly burdened with authenticating the users. A frequent user authentication process can cause CPU overload or waste a large amount of network resources. Servers have to authenticate the client information for every request. Instead of sending ID and password via POST method, most of companies are trying to solve the problem by using HTTP Basic Authentication, HTTP Digest Authentication, and Access Token (OAuth). In this section, we show the authentication methods that are used.

1) HTTP Basic Authentication: It uses the ID and password of a client to authenticate the client’s request in HTTP header. Figure 2 shows the HTTP Authentication phases. When a server requests an authentication for a client, the server sends a message “HTTP 401 Not Authorized” with WWW-Authenticated HTTP Header. Client’s ID and password get encoded with Base64 and are stored in the HTTP Authentication header. As they are not encrypted or hashed, they are usually sent over HTTPS or SSL. However, HTTP Basic Authentication has a critical issue that it doesn’t support a logout function. Also, it has security vulnerabilities of replay attack, injection attack and middleware hijacking [11] (conference). Due to the need for saving ID and password at HTTP Header, HTTP Basic Authentication can be exposed to such attacks.

2) HTTP Digest Authentication: It is an advanced version of HTTP Basic Authentication. HTTP Digest Authentication encrypts the client’s ID and password via hash such as MD5 [12] (conference). By creating a nonce on the client side, it can protect the hash from a Rainbow Table attack. Also the timestamp created on a server can secure message of a client from a replay attack. However, HTTP Digest Authentication has several vulnerabilities in regards to security. For instance, since it does not provide a method to verify the server to a client, HTTP Digest Authentication can be attacked by a Man-in-the-Middle attack. Such an attack changes HTTP Digest Authentication to HTTP Basic Authentication. In other words, the very purpose of HTTP Digest Authentication can be nullified.

3) Token Based Authentication (OAuth): It uses a Token instead of user’s ID and password [13] (conference). Figure 3 displays the way Token Based Authentication works when the information of a user gets authenticated. Resource Server (RS) redirects user’s browser to Private Key Generator (PKG) when a user requests RS a service. Firstly, users log in to Authorization Server (AS) to get a token. Secondly, RS gets the token from the AS. Lastly, RS verifies the messages from the user using the token. Consequently, due to the use of a token in communication between a user and RS, users’ ID and passwords are protected from a third party. Due to such an advantage, OAuth method is frequently implemented by various Web Service companies such as Twitter, Yahoo, Google, Facebook, Microsoft and so on.

Even though REST is based on the principle of statelessness, above authentication methods necessitates the concept of state across multiple requests. HTTP Basic Authentication and HTTP Digest Authentication need sessions to authenticate messages of clients. Servers have to store a
Figure 4. Data exchange with ID-based Authentication

session ID to run a communication between RS and clients. It means that such methods are not statelessness due to sessions. Otherwise, without using a session, HTTP Basic Authentication and HTTP Digest Authentication have to authenticate every single request of a client each time, thereby wasting time and resources of CPU and Network. Also, it has a security vulnerability due to contained (contaminated) ID and password. Due to the authentication problems of these two methods, most companies use OAuth instead of HTTP Basic Authentication and HTTP Digest Authentication. However, OAuth is also vulnerable for security breach, namely Covert Redirect Vulnerability (CRV). It uses redirection in Figure 3 by exploring a parameter. Attacker forges a login page (Figure 3 – ID and password) or by redirecting the URI to get client’s ID and password.

In the next session, we explain the basic concept of ID-Based Encryption (IBE), and then show the Boneh-Franklin scheme which is an IBE system proposed by Dan Boneh and Matthew K. Franklin in 2001 [14] (conference).

4 Proposed Stateless Lightweight Authentication

In the HTTP basic authentication, a client is authenticated by the server via a two-step process, but this has a security vulnerability as mentioned before [11]. OAuth uses a third party entity for authentication, that is, a client has to access the OAuth server to get a token in order to access a resource server (e.g., web server). Specifically, OAuth requires the extra steps from step 2 to step 6 in Figure 3. Generally, all current authentication methods for REST require additional steps, such as a redirection in OAuth, storage of a session key, or storage of ID and password in HTTP authentication. Those requirements make it difficult to maintain the statelessness of RESTful web service.

As a solution to this, we propose a lightweight stateless authentication mechanism for REST with ID-based cryptography, which we call ID-based Authentication (IBA). In IBA, the receiving machine’s URI itself is virtually the public key, so there is no need to exchange the public keys separately. If a client knows the URI of the resource server (RS), the client can derive the public key of the RS from the URI and the master public key, and vice versa. They can also sign the message for mutual authentication using their own private keys. Therefore both RS and the client can authenticate each other using only their URIs.

Furthermore, OAuth itself does not provide an encryption capability, and it typically relies on SSL/TLS for encryption as shown in step 7 in Figure 3, which incurs a high overhead. OAuth implementations also have some known security vulnerabilities [15] (online). On the other hand, our IBA does not require any extra step for encryption and signature. Figure 4 shows the steps in the proposed IBA. The details of the IBA are described in the later sections.

5 Basic Concept of ID-based Encryption

IBE is an asymmetric key algorithm. It was first proposed by Adi Shamir in 1984 [16] (conference) to simplify the certificate management process in e-mail systems. With IBE, the sender and the receiver do not need to obtain the public key certificates of each other. Unlike in RSA where a pair of public and private keys are generated at the same time by the user, a private key is created first by Private Key Generator (PKG) in IBE. PKG is a trusted third party entity that computes the clients' (Alice and Bob) private keys from their unique ID such as email addresses. The public keys are generated by the sender whenever needed from the destination’s unique ID and the PKG’s public parameters. This process is summarized in the following sequence. The symbols used in IBE are listed in Table 1.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>PKG</td>
<td>PKG public key</td>
</tr>
<tr>
<td>MKpub</td>
<td>PKG private key</td>
</tr>
<tr>
<td>KAlice</td>
<td>Alice public key</td>
</tr>
<tr>
<td>KAlicepri</td>
<td>Alice private key</td>
</tr>
<tr>
<td>KBob</td>
<td>Bob public key</td>
</tr>
<tr>
<td>KBobpri</td>
<td>Bob private key</td>
</tr>
<tr>
<td>C</td>
<td>Cipher text</td>
</tr>
<tr>
<td>M</td>
<td>Message</td>
</tr>
<tr>
<td>E</td>
<td>Encrypt</td>
</tr>
<tr>
<td>D</td>
<td>Decrypt</td>
</tr>
<tr>
<td>P</td>
<td>Master public point</td>
</tr>
<tr>
<td>s</td>
<td>Master private point</td>
</tr>
<tr>
<td>Bobemail</td>
<td>Bob email address</td>
</tr>
<tr>
<td>RSpub</td>
<td>Resource Server public key</td>
</tr>
<tr>
<td>RSpri</td>
<td>Resource Server private key</td>
</tr>
<tr>
<td>r</td>
<td>Random number</td>
</tr>
<tr>
<td>sQ</td>
<td>Client Private point</td>
</tr>
<tr>
<td>URIs</td>
<td>Resource Server URI address</td>
</tr>
<tr>
<td>URL</td>
<td>Client URI address</td>
</tr>
<tr>
<td>Cs</td>
<td>Signed cipher text</td>
</tr>
<tr>
<td>G</td>
<td>Generate key</td>
</tr>
</tbody>
</table>

According to Dan Boneh and Matthew K. Franklin [14] (conference), IBE protocol framework is defined by four steps, i.e., Setup, Extract, Encrypt, and Decrypt. Setup step works
for PKG to create a master private key and a master public key. The master private key is kept securely in the PKG and used to generate client’s private keys. The master public key (public parameters) is open to public. Extract step generates a client private key. When a client requests its own private key with ID, PKG generates a private key using the master private key. Encrypt step occurs when the sender encrypts a message using receiver’s public key generated by PKG. In Decrypt step, receiver decrypts the encrypted message using own private key generated by PKG. In the next section, we show the details of the above steps.

6 Boneh-Franklin ID-based Encryption

Boneh-Franklin (BF) scheme uses elliptic curves and finite fields for IBE protocol. Elliptic Curve Cryptography (ECC) was first suggested by V. S. Miller [16] (conference) and N.Koblitz [17] (conference). ECC is a public key cryptography with pairing based on cryptography. ECC can be divided into two groups, $G_1$ and $G_2$. (Explain what they are) The points $P_1$ and $P_2$ are on the elliptic curve which satisfies the following equation.

$$y^2 = x^3 + ax + b \mod p$$

When the point $P_1$ is added to the point $P_2$, the equation has only one point $P_3$. BF-IBE uses the concept to make a pair for client public key and private key.

1) Setup: PKG makes a master private key and a public key
   a) Selecting elliptic curve $E$ and a large prime $p$ in cycle group $Z_p \in E$.
   b) Making additive group $G_1$ using point $P$ which is prime order $p$ in the $E$.
   c) PKG chooses an arbitrary $P \in E$ of order $q$.
   d) Making a group $G_2$ using prime order $q$ depending on security parameter $k$.
   e) Corresponding pairing $e$: $G_1 \times G_1 \rightarrow G_2$
   f) Master private point: $0 < s < q - 1$
   g) PKG public key: $MK_{pub} = s \cdot P$
   h) Hash function $H_1$: $\{0,1\}^* \rightarrow G_1^*$
   i) Hash function $H_2$: $G_2 \rightarrow \{0,1\}^*$
   j) Message space $M = \{0,1\}^*$
   Ciphertext space $C = G_1^* \times \{0,1\}^*$

PKG publishes all system variables except the master private key.

2) Extract: PKG extracts the client public key and the private key.
   a) Client public key is generated by $H_1$ hash function.
      $$K_{Bob-pub} = H_1(\text{BobEmail})$$
   b) PKG generates a client private key using master private point $s$ and the client public key.
      $$K_{Bob-priv} = s \cdot K_{Bob-pub}$$

Set up and Extract execute only one time. Figure 5 shows the one-time key distribution sequence in ID-based encryption using public points $(p1, p2)$ and private points $(sQ, sQ')$.

![Figure 5. One time key distribution in ID-based](image)

3) Encrypt: Sender encrypts a message $(M)$ using receiver’s public key.
   a) Select a random $r$ in $Z_p$
   b) $U = rP$, $V = m \oplus H_2(g_{Bob})$.
   c) $C = (rP, m \oplus H_2(g_{Bob}'))$,
      where $g_{Bob}' = e(K_{Bob-pub}, MK_{pub}) \in G_2$

The resulting pair $C = \{U, V\}$ is the cipher text.

4) Decrypt: Receiver decrypts the ciphertext using receiver’s private key $(K_{Bob-priv})$ obtained from PKG, and $MK_{pub}$ in BF-IBE setup step.
   a) $MK_{pub} = e(K_{Bob-pub}, U)$
   b) $M = V \oplus H_2(g_{Bob})$

In the next section, we explain the details of the IBA for stateless RESTful communication.

7 Stateless Authentication for RESTful Service

IBA uses the ID-based cryptography and the unique URI in REST for authentication. As shown in Figure 1, all IoT objects can be represented by a unique URI. Therefore, we can use the REST URI as the unique ID needed in IBE.

Figures 4 and 5 show the authentication process in IBA. We assume that the PKG has already generated a PKG private point $s$ and a public key $s \cdot P (= MK_{pub})$ in the BF-IBE setup step.

1) Extract: PKG sends $p_1$, $p_2$ and $sQ$ in Figure 5, to RS and the client to create own private key and public key. This part is similar to the Extract step of BF ID-based encryption.
   a) RS and client create own private key and each other public key.
   b) Client creates its own private key via private point $(sQ)$ and master public key $(MK_{pub})$. Also, client generates RS’s public key.

Client: $Client_{priv} = sQ \cdot MK_{pub}$
$RS_{pub} = UR_{client} \cdot MK_{pub}$

2) Encrypt: Client encrypts a message $(M)$ using RS’s public key. Additionally, IBA uses a digital signature for mutual authentication. So the message is encrypted again under $Client_{priv}$ for digital signature [18] (conference).
   a) Client encrypts $M$ using $RS_{pub}$.
C = E (M, RSpub)

b) Client signs the encrypt message (C) using Clientpri.
C = E (C, Clientpub)

3) Decrypt: When RS receives an encrypted message, RS
verifies C with Client’s public key. Then the encrypted
message is decrypted with RS’s private key.

a) RS verifies the C, via Clientpub.
C = D (C, Clientpub).

b) RS decrypts C using RSpri.
M = D (C, RSpub).

As shown in the above, all authentication and encryption
are done with pre-generated public and private keys, and IBA
does not maintain client’s state during the authentication
process. There is no need for ID and password of a client or a
session ID. Thanks to this statelessness, clients’ information
is more secure than in HTTP Basic Authentication or HTTP
Digest Authentication. Moreover, unlike in OAuth, IBA
doesn’t use a redirection function, so the clients’ requests can
be handled more efficiently and securely.

8 Implementation of IBA Model on
REST

We implemented the stateless and secure RESTful web
service with IBE. In this section we show its structure and a
code segments. We used BF-IBE library made by Changyu
Dong [19] (online) that uses Jpair, a pure Java implementation
of bilinear pairing, available freely under the GNU general
public license.

It is composed of three main parts, PKG, RS and user. As
mentioned before, we assume that the connection between
PKG and RS (and between PKG and client) is secure by using
other mechanism, e.g., with SSL/TLS or offline. This is
acceptable because the communication between PKG and the
client (or RS) does not occur frequently. IBE is used between
client and RS.

A. Private Key Generator (PKG)

PKG makes its own master private key and a master public
key to use as the PKG’s secure key. The master key pair is
made by BF-IBE scheme with a random number. The process
of making the master key pair is shown below:
final static Pairing e = Predefined.ssTate();
final static KeyPair masterKey = BFCipher.setup (e, new
Random());
The master key pair used for client’s (RS and client) private
point pair (Q and Q’) is created with a random number and the
client’s URI. The process is described below:
KeyPair extractKey =
    BFCipher.extract (masterKey, Clienturi, new Random());
Point sQ = (BFUserPrivateKey) extractKey.getPrivate().getKey();

The master public key is open to anyone who wants to
make a client’s public key with URI. Again, we assume that
the communication between PKG and the client (or RS) are
secure using SSL or offline. PKG sends the information about

@Path("/requestSet/[id:]")
@Produces(MediaType.TEXT_HTML)
public String getMkey(@PathParam("id") String ID) throws
IOException{
    BFMasterPublicKey masterPublicKey=BFMasterPublicKey()
    IBEMaster.BFMasterPubliccKey();
    StringBuffer results = new StringBuffer(""");
    Pairing pair=masterPublicKey.getPairing();
    String str=""; // convert to string
    results.append("/PointPpub/");
    str=masterPublicKey.getPub().tostring();
    results.append(str + "/PointPpub/");
    results.append(masterPublicKey.getPub().toString());
    return results.toString();
}

Figure 6. Handling request of master public key

the private point pair (sQ) and the master public key pair when
requested by client and RS. These processes are demonstrated
in two code segments. Figure 6 is sample code of handling
request of the master public key on PKG. PKG extracts two
points (p1, p2) and sends them as a string to client (or RS),
when PKG receive request from client (or RS) via URI
(PKGuri/requestSet/{ClientID}). Note that REST can send a
string data only via URI and { ClientID } is the client’s (or RS)
unique URI. PKG extracts client’s private key using client’s
URI. Then it sends the private point pair to client when upon
receiving a request from the client.

B. Resource Server (RS)

An RS is a web server that provides web content to a client.
RS has its own URI and can make its own public key from
URI using the master public key. Also, RS only needs a master
public key and the client’s URI to get the client’s public key.
The client’s public key is created by:
BFUserPublicKey getUserPublicKey =
    new BFUserPublicKey (Clienturi, masterPublicKey);
RS has to request private point from PKG to derive its own
private key. RS generates its own private key using the master
public key and the private key. This process is shown below:
getUserPrivateKey =
    new BFUserPrivateKey(sQ, getMasterPublicKey);
RS encrypts a message using the client’s public key. RS
decrypts the encrypted message encrypted under RS’s public
key. As mentioned before, requesting private point and master
public key rarely occurs between PKG and RS. Also we
assumed that the communication is secure.

RS request master public key and private point (sQ) from
PKG. PKG generate private point (sQ) using RS’s URI as the
ID. And then, PKG send back the master public key and
private point (sQ) to RS. RS can make the RS’s private key
using master public key and private point (sQ). RS decrypts
the encrypted message using BFCipher library with RS’s
private key. The code segments implement secure
communication between RS and client for RESTful web
service using IBE.
C. Client

A client is a person or IoT that is connected web site, i.e., RS. We assume that the client has his own URI for communication with PKG and RS. The client requests a private point from PKG to make his own private key like RS. The private key is used for decrypting cipher text which is encrypted under the client’s public key. Also, a client can make RS’s public key via master public key which is openly announced by PKG. The procedure of generating a RS public key is shown below:

\`
BFUserPublicKey getUserPublicKey =
new BFUserPublicKey(RSuri, masterPublicKey);
\`

It is the same as the process of making the client public key except that Clienturi is replaced with RSuri. The client encrypts a message under RS’s public key when the client communicates with RS. The client also gets a master public key and private point from PKG, like in RS. In the client side, the client uses its own URI as the ID.

9 Performance Test

We measured the performance in elapsed time starting from the encryption and signature of a message at user and to the time of decryption and verification at RS. Since the communication between PKG and client occurs only rarely, we excluded the processes described in key generation and pass Master public key to client and RS in this testing. We only measured the performance of the steps in encryption, decryption and communication in Figure 5.

We compared the performances of RSA with 1024-bit key and BF-IBE with 160-bit key. BF-IBE is based on Elliptic Curve Cryptography (ECC) and the level of security for ECC with 160-bit key is equivalent to RSA with 1024-bit key [20] (journal). Unlike most other researches where the performances under different key sizes were tested, we tested the performances for different message sizes under the same key size.

The test result shown in Figure 7 indicates that the performance depends on the message size. There is no different in performance between RSA and BF-IBE in the small amount of data size. However, RSA is affected by the message size more heavily. This is due to the key size differences between RSA and BF-IBE which uses ECC. Since RSA must use a 1024-bit key while ECC uses only 160-bit key for equivalent security level, the encryption/description time in RAS grows more severely as the message size grows.

Generally, the encryption and transmission of a bigger message takes more time. Due to multimedia data such as pictures and video, the size of messages in Internet is growing larger. The increased time due to larger message should be considered to improve user experiences [21] (conference). Therefore, our IBE based solution is considered better for improving user experience.

10 Advantage of IBA in REST

The proposed IBA in REST have four major advantages such as statelessness, simplicity, performance, and reliability.

1) Statelessness: RS doesn’t need to store client’s state. Basically, REST communicates with eth resources using URI and HTTP method such as GET, PUT, POST and DELETE. The same process can be used in IBA and there is no need to store session id or token. The statelessness also has an advantage in balancing the server load. There is no need to serve multiple REST requests sequentially on one server and any server can process them independently.

2) Simplicity: IBA is simpler than OAuth in authentication processing. IBA does not require SSL/TLS or a redirection to AS. That is, IBA in Figure 4 does not need the extra steps from step 1 to step 7 in Figure 3.

3) Performance: OAuth or HTTP Authentication maintains a session between the time of logging in and the time of logging out. As such, a user is requires to log in to AS or RS for authentication when logged out. In contrast, IBA does not have a concept of login or logout. Therefore, IBA removes the overhead of login process such as steps 4, 5, and 6 in Figure 3. Additionally, as shown in section IX, IBA uses ECC and performs better than RSA-based authentication for large messages.

4) Reliability: OAuth relies on AS for all transactions. If AS becomes unavailable, it is impossible for RS and client to communicate. However, IBA does not rely on a 3rd party server. Unlike AS, the PKG is used only once for private key generation, so the unavailability of PKG does not affect normal communication between the clients and RS. As long as both parties are up and running, RS and client can communicate with each other directly any time.

11 Conclusions and Future Works

The majority of web service providers employ REST API instead of SOAP. However, there are some limitations in the authentication of the RESTful web service. A major concern is that it forces maintaining the client state at the server, which can also expose the client information to an attacker. This paper proposed a new method of IBA to solve this problem by not storing the client’s state implement.

In IBA, clients and servers get their private keys from PKG.
A client just needs to know the URI of the RS to perform authentication. The client can generate the RS’s public key by combining RS’ URI with the master public key. Likewise, the RS can get the client’s public key by combining client’s URI with the master public key. Both parties can encrypt a message under the receiver’s public key and sign under its own private key. This simplicity has a great benefit for IoT. IoT devices can be authenticated by IBA without maintaining a session or client status. We have described the four advantages of IBA over traditional authentication mechanisms, that is, statelessness, simplicity, better performance, and reliability. We also demonstrated the performance advantage of IBA over traditional RSA-based authentication.

In the future, we plan to implement a full RESTful web service with IBE. We also plan on testing the performance of IBE with and without signatures, and with non-IBE RESTful web service as well as studying other security issues.

ACKNOWLEDGEMENT

This material is based upon works supported by the National Science Foundation under grant number IIA-1301726.

DISCLAIMER

Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

REFERENCES