Improvised Security for RDF using Random Salt and Levels with Bi-Directional Translation to Relational Database

[1]Siva Srinivasa Rao Mothukuri, [2]Sai Ganesh Repaka

[1, 2] Information Technology, Acharya Nargarjuna University,
Guntur, Andhra Pradesh, India

Abstract - A Middleware Framework is required for Semantic Web applications to access data from relational databases and legacy storage systems, without replication or synchronization. Resource Description Framework is one such type of middleware framework which bridges the gap between two needs. Though, there were some efforts to publish relational data as Resource Description Framework triples, they present the existing data into respective RDF stack only in Read-only manner, also in dearth of security. This paper accomplishes such in vulnerabilities by enhancing a nexus to be bi-directional, allowing data updates specified as triples to be sent back to the relational database as tuples. The data can be Updated/Inserted/Deleted wherever required. Also, this paper deals with the Level-1 and Level-2 security hierarchy. In those two levels, a random "salt" is added up to the data followed by encryption.

Keywords - *Semantic Web, RDF, DES, AES, Base 64 encoding*

1. Introduction

RDF ("Resource Description Framework"), which is the standard for encoding metadata and other knowledge on the Semantic Web. In the Semantic Web, computer applications make use of structured information spread in a distributed and decentralized way throughout the current web. RDF is an abstract model, a way to break down knowledge into discrete pieces, and while it is most popularly known for its RDF/XML syntax, RDF can be stored in a variety of formats.

This paper describes conversion of triple to tuple with encryption standards of 1st level and 2nd level. 1st level deals with the encryption of the RDF triples using DES. Well, 2nd level deals with AES. As both DES and AES are predefined and easy to decrypt for back to original format, we use salt for each level of encryption so that encryption makes sense.

2. Translation to Relational Database

In this paper we deal with to-and-fro transfer, which is quite easy. But the problem is to handle Blank RDF triples, where triple contains data that is null. So as a part of handling, we use our basic concept of subject, predicate, and object by which we can capture required data and place it in relational database.

2.1 Handling Simple RDF Triples

Handling simple RDF triples require simple effort; using the id value of specific triple to get the required value. In general, we need to assign and handle "required id" to particular RDF triple which is purely dependent on the requirement of user.

Here we generate a simple RDF triple as a sample, in which we consider project and its description. The values are given with respect to "id" by which RDF triple is handled.

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:vcard="http://www.w3.org/2001/vcard-rdf/3.0#" >
        <rdf:Description rdf:about="http://localhost/project/1">
          <vcard:TEL>9933992211</vcard:TEL>
          <vcard:Locality>Gachibowli</vcard:Locality>
          <vcard:FN>RDF Triple Conversion Project</vcard:FN>
          <vcard:KEY>1</vcard:KEY>
          <vcard:UID>project</vcard:UID>
        </rdf:Description>
</rdf:RDF>
```

Algorithm to insert triple into Relational Database:

Algorithm : Insert_Triple()
Input : Given RDF Triple to be inserted
Output: Successful insertion of Triple into Relational Database.

Identification of Table to which Triple must be inserted based on subject and id
If foreign key relation exists

```
                If the triple value that contains relation
                  with previous table
                 If the triple contains Primary Key and not
                 a duplicate triple
                    Insert Triple into Database using
                    subject     with respect to columns in
                    relational database
                Else
                  Throw error primary key constrains
                  violated, Duplicate values
                End-If
                Else
                 Insert Triple into Database using subject
                with respect to columns in relational
                database
               Else
                Throw error foreign key constrains
        violated
              End-If
        If the triple contains Primary Key and not   a
        duplicate triple
                Insert Triple into Database using subject
                with respect to columns in relational
                database
        Else
                Throw error primary key constrains
                violated, Duplicate values
       End-If
    Else
        Insert Triple into Database using subject     with
        respect to columns in relational database
    End-If
End
```

2.2 Handling Complex RDF Triple

Handling complex RDF triple i.e. Blank node RDF triple is a difficult task when compared to Simple RDF triple. Here we should consider all three parameters in the RDF, i.e. Subject, Predicate, Object. As the subject gives null or some new assigned value that cannot be identified by the ID, it can be only identified by comparing the subject of a value with the predicate of parent tag. Generated sample complex RDF Triple is given below in which there is subject with A1, A2, A3 that must be handled using the subject of child tag with parent tag's predicate value.

```
    <rdf:RDF
      xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
      xmlns:vcard="http://www.w3.org/2001/vcard-rdf/3.0#" >
     <rdf:Description
rdf:about="http://localhost/employee/employee/venki">
      <vcard:LABEL>4</vcard:LABEL>
      <vcard:TEL>5656563</vcard:TEL>
      <vcard:Region>australi</vcard:Region>
      <vcard:ADR>melbone</vcard:ADR>
      <vcard:Street rdf:nodeID="A0"/>
      <vcard:FN>venki</vcard:FN>
      <vcard:KEY>4</vcard:KEY>
      <vcard:UID>emp_blnk</vcard:UID>
     </rdf:Description>
     <rdf:Description rdf:nodeID="A1">
      <vcard:ADR>mumbai</vcard:ADR>
      <vcard:Extadd>lakshmipuram</vcard:Extadd>
     </rdf:Description>
     <rdf:Description rdf:nodeID="A2">
      <vcard:ADR>hyd</vcard:ADR>
      <vcard:Extadd>svn colony</vcard:Extadd>
     </rdf:Description>
     <rdf:Description
rdf:about="http://localhost/employee/employee/siva">
      <vcard:LABEL>1</vcard:LABEL>
      <vcard:TEL>9090</vcard:TEL>
      <vcard:Region>ap</vcard:Region>
      <vcard:ADR>guntur</vcard:ADR>
      <vcard:Street rdf:nodeID="A3"/>
      <vcard:FN>siva</vcard:FN>
      <vcard:KEY>1</vcard:KEY>
      <vcard:UID>emp_blnk</vcard:UID>
     </rdf:Description>
     <rdf:Description rdf:nodeID="A0">
      <vcard:Extadd>dokey</vcard:Extadd>
     </rdf:Description>
     <rdf:Description
rdf:about="http://localhost/employee/employee/bhanu chandar">
      <vcard:LABEL>3</vcard:LABEL>
      <vcard:TEL>232323</vcard:TEL>
      <vcard:Region>tamil nadu</vcard:Region>
      <vcard:Street rdf:nodeID="A2"/>
      <vcard:FN>bhanu chandar</vcard:FN>
      <vcard:KEY>3</vcard:KEY>
      <vcard:UID>emp_blnk</vcard:UID>
     </rdf:Description>
     <rdf:Description
rdf:about="http://localhost/employee/employee/jatin">
      <vcard:LABEL>2</vcard:LABEL>
      <vcard:TEL>1212</vcard:TEL>
      <vcard:Region>maharastra</vcard:Region>
      <vcard:Street rdf:nodeID="A1"/>
      <vcard:FN>jatin</vcard:FN>
      <vcard:KEY>2</vcard:KEY>
      <vcard:UID>emp_blnk</vcard:UID>
     </rdf:Description>
     <rdf:Description rdf:nodeID="A3">
      <vcard:Extadd>brodipet</vcard:Extadd>
     </rdf:Description>
</rdf:RDF>
```

3. Improvised Security using Random Salt

3.1 Levels in Encryptions

As we generate the RDF as a plain text, it can't be transferred over the network. For example, it contains all the details of Credit-Cards or Passwords. For an attacker it is easy to take over the data using Man-in-Middle attack by sniffing the packets that are sent over the network which contains plain data.

We consider two levels of encryption:

1. **Level I**
   In Level I encryption we encrypt the RDF using simple DES Encryption algorithm which is 128/256 bit encryption. Even it is very easy to decrypt with same algorithm for an attacker. We make this difficult by adding "Random Salt", that is discussed later in this paper.

2. **Level II**
   In this Level, already encrypted RDF is given as input. We use AES encryption to encrypt the RDF with Random Salt. Now RDF is ready to deploy over the network.

3.2 Random Salt

Random Salt is very important for the encryption since it makes difficult for an attacker to analyze the data. We can consider salt as a simple string that may be names of people, or objects that is appended to RDF and then it is encrypted.

Here is the simple Project RDF which was illustrated in 2.1 after encryption without salt, Base 64 encoded.

0Wyt0FB701z1ipwxrJg/BL2G09lN5dJQWXlL7TLcaesMLE0hhNDti qusLe8oz0yD9r7NkqDyYGnTy1k+zTUFWhQg/WMtTMPh+UsP/Ws KOZgIBXLgmD0wYXp5CsEz4AbDatcNt+3DzFINimp/OE0Tvzuzdaf /idlpc9i2S2w2beSzb1bJgu2Eqwawfp5q1YGJdnJUTptwOkBSgPU4i72 KSFs+Bboo2ZgBHU0+AEmwcJq09aNZSCawc0eKr3c4pEN3gkukfL 6jtQk8BNJmHIkhjJ3WGhX1b+ZIAR5DnwnMR6DGyo8XcXY6iLU Yc2aQe531EcHd2o4cpj3iIdoB1ZEj8Q/9YO39kyV0P8l6VlSo7sJhIK TgJaDLv29hiVWEXUVXJZS0G+yhPkcVpr7ylttgYdQtR3G7vxdDEb OE0xaEBfTmDkLSNMseN8soYoTxN/9pCh4ReUJQRS5l3E3dnOcnd XwaXNYaSvNXIzolQ5FGz+OTJKzZJIO5s4ZFl4IZ6D0WnqjKJ4yB NQaMU1qIcyXwD/vi0hAJ/wgVsqzyuyhs0dMl4xMMqyMgTQ==

The same simple Project RDF after adding salt, Base 64 encoded.

Yi26uQgRPmaUd/DzqUOdl/knbAnE9PB8Y8lxoL4xK96VR8Q0kl3jas ldJrh3DcywY/aTMgmX6vvzZzVP+/fQ3DX3GeYj1b+Tj+FWm8F1W KmX5QYeUWi15PnWgLaFeN3ptU3bw6/riBbXUmFye/qltBjZvDeUg HQ4iDcuxnljvgBeB9Rls7mlUQI1eNsbe3qB5g49nm00grsUcdUqrF91l 6aEBnVeirQiOE2/L08CbRIAhpNtyuQo6lf6q7i7XjIemznfszi0ZVBrx M7Stj5iG38ltN2mBYMYH7c/g8zxUxPPYqnKkDhcigm+TJjJucdftPg 87h4CHDI4W6ZkGXYoiXLJuOgXtVnzFdWqD9djT/V2yB3NGeEjq 9EHN1zFXejKxz8CacIupf4HS5qmOAVfHW6/kiKlCr3c382t+urcT56 r4l8J6yTdoXR2DfIAMmYt5kR12E9NWAZfCRFdH8vUUFOPjXkG 4oqDHpWfQO1bTXRKw7OfgYtuQybP+Dt9Dy3WRftsWfB3Tu6cprj gcvRIU11Fleet2EkOv1dwN+nkzcq/WNXNSJNjppokxVkI1ovkR0Qt O/SGq/LimR3hdBM9GQ==

We don't use simple salt like person name; Salt is generated randomly using a random function. Salt will be attached to the encryption using delimiter at any cost to the obtained result after encryption.

Below figure illustrates how the levels of encryption takes place and flows over the network, followed by decryption at the end before operations to be performed on the relational database.

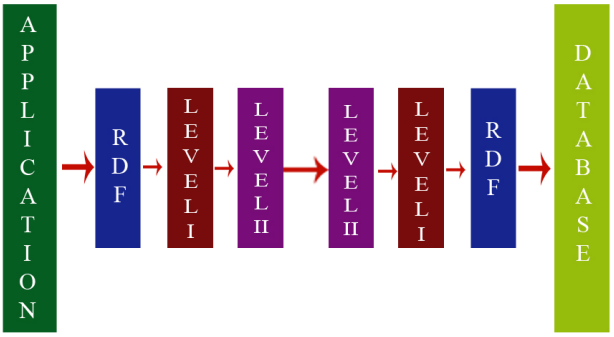

Fig. 1 RDF Generation and flow over network with encryption/decryption.

Algorithm : Encryption_using_RandomSalt()
Input : Given plain RDF File that contains Triples
Output: Successful encryption of RDF file with Levels and random Salt.
Identity RDF File
Generate Random Salt : RandomSalt()
Level I Encryption : DES(File,Salt)
Generate Random Salt : RandomSalt()
Level II Encryption: AES(DES_ENCRYPTED_File,Salt)
Return File

Function:RandomSalt()
Generate a Random Number
Split Random Number to an Array
For each Random Number in Array
Replace with an Alphabet/digit/symbol
End For
Join Random String Array to String
Return

Function: DES(File,Salt)
Append Salt to File
Encrypt the File using DES Standard Encryption Algorithm
Append the Salt using Delimiter at Start/End/any Position of File that must be specific
Return

Function: AES(DES_ENCRYPTED_File,Salt)
Append Salt to File

Encrypt the File using AES Standard Encryption Algorithm
Append the Salt using Delimiter at Start/End/any Position of File that must be specific
Return

Decryption follows the same procedure in reverse direction. We generate random salt for each and every level of encryption. Here is final output of sample project RDF file discussed earlier. In this the random salt is added at the end of file with delimiter "::". We use "::" for splitting the string. For the following example **Ayen20*&3bLjkI(302** is the random salt generated by the application.

U2FsdGVkX18DeEhxXQGi2v1BNkZErzkJlA7PQS7VGeOvOpzwE1DTn018A6pdw2koRe7RR94FqkY0OnUpGUcjjsJ1WXYfBUU3atJ8lhTKzBdJ9qkf2s98SIw8j7IODV3JeDilA40Gk6bH2jHJjhMH9IOSFokRTEhLnlx5hogzlj4GXPPnG6k2dDFRP5hMr32YFTw/7b30Vz/xIWSzG935tQ4XpYxvOcMbBnRW4RzI02yrxOcnvv69W1JT5o+s4zMceytrE3h3Eyho6nA0cDK6Bqyi4FszcYXmlaUamzFcVuujE8KklbRefWLkgOa7AM0WWFlSs+uKBXH5EC4D0HI9hBs9BfCZsNXeZoS3BAQ925XPErdtWDdgIiuj8ZhPSi0Fo1aG9dObeFK46twMk2eLk5p6a8+YpdPF6iiXv9msjgO48vHJfvHN4Q7ZNIdG3euU1VYPT7FFhAzVCpG+N+K6rEDB1ph2LptpWreiLqebUmXsXXPs2Jqb0I1Q+17bViGaxqWwsLLJYfxrkatyRneTMGZE28yZHZ+DQKvINJwgQD07+ei2ucloXTlS/7aNXcDANUmnoUWLbEZ2Mvv6zVPlSZm3tnqBvcladrd+MiWU/3zjZysxnZOPl+cZ0xsD8M+PX+jdl+BtFMi76Vi6YLgtquqWueiBIZz6JVFHLuBARL4zWCQJ4tM6/d6Dyi5xChmxU4C+Drumv/YuvrJfZmND8bPsfWkOKZqqU35gAwBRa6PW8rJNxzrQ8UA04uCHP5Bk7+c2KzczbLbFEUYLplIY8g::Ayen20*&3bLjkI(302:::==

4. Conclusions

By the way of conclusion, this paper states bi-directional translation to relation database using RDF which is encrypted for a better security. Security is provided level by level. We can expect further better features added to RDF to make easy to access.

References

[1] Sunitha Ramanujam , Vaibhav Khadilkar , Latifur Khan , Steven Seida – "Bi-Directional Translation of Relational Data using Virtual RDF Store" 2010 IEEE Fourth International Conference on Semantic Computing.

[2] A. Malhotra. W3C RDB2RDF Incubator Group Report. http://www.w3.org/2005/Incubator/rdb2rdf/XGR-rdb2rdf-20090126/, 2009.

[3] O. Erling and I. Mikhailov. RDF Support in the Virtuoso DBMS. In S.Auer, C. Bizer, C. Muller,¨ and A. V. Zhdanova, editors,*CSSW*, volume 113 of *LNI*, pages 59–68. GI, 2007.

[4] S. Auer, S. Dietzold, J. Lehmann, S. Hellmann, and D. Aumueller. Triplify: light-weight linked data publication from relational databases. In J. Quemada, G. Leon,´ Y. S. Maarek, and W. Nejdl, editors, *WWW*, pages 621–630. ACM, 2009.

[5] Y. An, A. Borgida, and J. Mylopoulos. Discovering the Semantics of Relational Tables Through Mappings. 4244:1–32, 2006.

[6] M. Hert, G. Reif, and H. Gall. Updating relational data via SPAR- QL/update. In F. Daniel, L. M. L. Delcambre, F. Fotouhi, I. Garrigos,´ G.Guerrini, Jose-Norberto Mazon,´ M. Mesiti, S. Muller¨-Feuerstein, J.Trujillo, T. Marius Truta, B. Volz, E. Waller, L. Xiong, and E. Zimanyi,´ editors, *EDBT/ICDT Workshops*, ACM International Conference Pro- ceeding Series. ACM, 2010.

[7] C. Bizer. D2R MAP - A Database to RDF Mapping Language. In*WWW (Posters)*, 2003.

[8] S. Ramanujam, A. Gupta, L. Khan, S. Seida, and B. Thuraisingham. R2D: A Framework for the Relational Transformation of RDF Data. *Int. J. Semantic Computing*, 3(4):471–498, 2009.

[9] S. Ramanujam, V. Khadilkar, L. Khan, S. Seida, M. Kantarcioglu, and B. Thuraisingham. Bi-directional Translation of Relational Data into Virtual RDF Stores. Technical ReportUTDCS-13-10, The University of Texas at Dallas, 2010. Available at www.utdallas.edu/ sxr063200/ D2RQ++Ver1.pdf.

[10] C. Bizer and R. Cyganiak. D2R Server - Publishing Relational Databases on the Semantic Web. Poster at 5*th* International Semantic Web Conference, 2006.

Siva Srinivasa Rao Mothukuri born in Guntur, in the year 1991. Acquired bachelor's degree from Acharya Nagarjuna University in 2012, also Certified in Ethical Hacking. Worked as Assistant Professor in R.V.R & J.C College of Engineering. Currently working as ASET in Tata Consultancy Services (TCS).

Sai Ganesh Repaka born in Guntur, in the year 1991. Acquired bachelor's degree from Acharya Nagarjuna University in 2012, also an Oracle Certified JAVA Professional. Currently working as ASET in Tata Consultancy Services (TCS).