THALES

Vormetric Data Security Platform

Vormetric Data Security Manager (DSM)

Using Vormetric Crypto Server with AWS PAAS

Document Version 1

Vormetric Data Security Platform
Using Vormetric Application Encryption (VAE) with the Vormetric Crypto Server
Data Security Manager (DSM) Version 6.1.0
November 2018, Documentation Version 1.0

ANY HARDWARE OR TECHNOLOGY, INCLUDING, WITHOUT LIMITATION, ANY FAILURE OF DATA TO BE PROPERLY PROCESSED OR TRANSFERRED TO, IN OR THROUGH LICENSEE'S COMPUTER ENVIRONMENT OR ANY FAILURE OF ANY TRANSMISSION HARDWARE, TECHNOLOGY, OR SYSTEM USED BY LICENSEE OR ANY LICENSEE CUSTOMER. VORMETRIC SHALL HAVE NO LIABILITY FOR, AND LICENSEE SHALL DEFEND, INDEMNIFY, AND HOLD VORMETRIC HARMLESS FROM AND AGAINST, ANY SHORTFALL IN PERFORMANCE OF THE SOFTWARE, OTHER HARDWARE OR TECHNOLOGY, OR FOR ANY INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS, AS A RESULT OF THE USE OF THE SOFTWARE IN ANY ENVIRONMENT. LICENSEE SHALL DEFEND, INDEMNIFY, AND HOLD VORMETRIC HARMLESS FROM AND AGAINST ANY COSTS, CLAIMS, OR LIABILITIES ARISING OUT OF ANY AGREEMENT BETWEEN LICENSEE AND ANY THIRD PARTY. NO PROVISION OF ANY AGREEMENT BETWEEN LICENSEE AND ANY THIRD PARTY SHALL BE BINDING ON VORMETRIC.

Protected by U.S. patents:

6,678,828

6,931,530

7,143,288

7,283,538

7,334,124

9,202,077

9,203,619

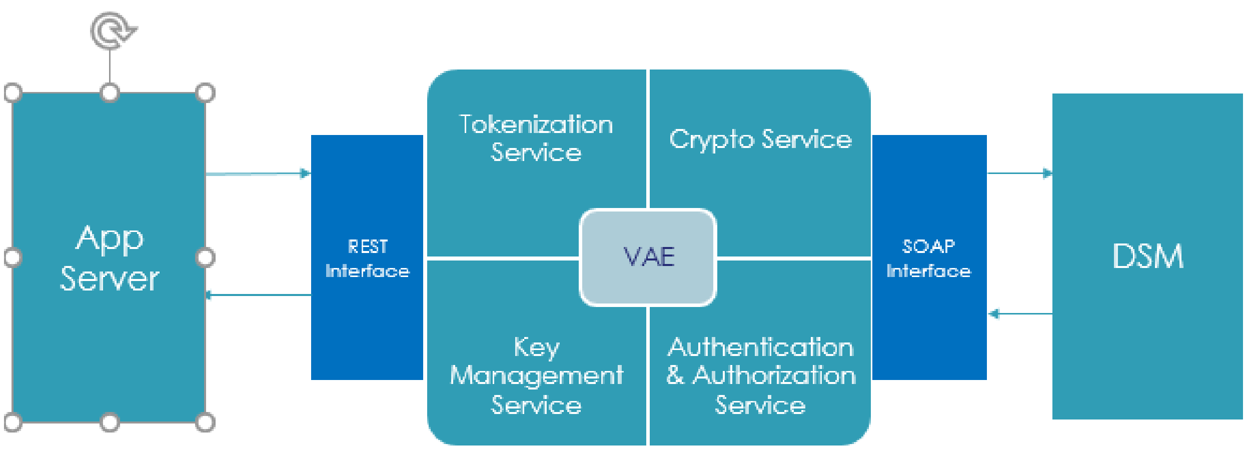
Thales Data Security includes a restricted license to the embedded IBM DB2 database. That license stipulates that the database may only be used in conjunction with the Vormetric Security Server. The license for the embedded DB2 database may not be transferred and does not authorize the use of IBM or 3rd party tools to access the database directly.

Contents

PREFACE

Using Vormetric Crypto Server with AWS PAAS provides working examples on how to implement tokenization or encryption using REST API's. This is the use case where a customer wants to bring your own encryption (BYOE). They own the encryption and encryption keys. Listed below is a diagram showing all of the capabilities of the Vormetric Crypto Server.

Tokenization, Crypto & Key Management REST APIs - Architecture



Vormetric Crypto Server

The examples provided in this document will demonstrate how to minimize code changes using a simple framework to encapsulate many of the common REST API's which should reduce the time to implementation tokenization or encryption.

DOCUMENTATION VERSION HISTORY

Product/Document Version	Date	Changes
V1.0	11/2018	Initial document release

ASSUMPTIONS

This documentation assumes the reader is familiar with the following Vormetric products and processes:

- Vormetric Data Security Manager (DSM)
- Tokenization
- Key management
- Data encryption
- Familiarity with REST

GUIDE TO VORMETRIC DOCUMENTATION

Related documents are available to registered users on the Vormetric Web site at https://support.vormetric.com

SERVICES UPDATES AND SUPPORT INFORMATION

The license agreement that you have entered into to acquire the Thales products (“License Agreement") defines software updates and upgrades, support and services, and governs the terms under which they are provided. Any statements made in this guide or collateral documents that conflict with the definitions or terms in the License Agreement, shall be superseded by the definitions and terms of the License Agreement. Any references made to “upgrades” in this guide or collateral documentation can apply either to a software update or upgrade.

SALES AND SUPPORT

For support and troubleshooting issues:

- https://help.thalesesecurity.com/hc/en-us
- Email questions to support@vormetric.com or call 877-267-3247

For Vormetric Sales:

- http://enterprise-encryption.vormetric.com/contact-sales.html
- Email questions to sales@vormetric.com or call 888-267-373

GETTING STARTED

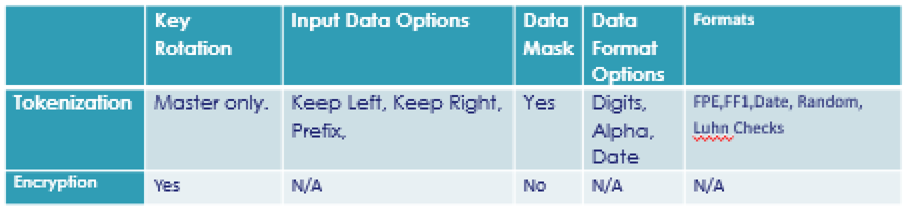
Use Cases

Vormetric Crypto Server can be used for many different use cases. Typically, it is used for scenarios when a company has sensitive data in a field of a particular file or a column in a database and they want to encrypt or tokenize the sensitive data. Use cases can include:

- Encrypt or tokenize SSN or credit card number data at the point of entry of an application.
- Encrypt or tokenize PII data that might be in a file.
- Encrypt or tokenize sensitive data before inserted into a PAAS based offering.

Methods

Listed below is a matrix that describes some of the differences between tokenization and encryption using the Vormetric Crypto Server.



	Key Rotation	Input Data Options	Data Mask	Data Format Options	Formats
Tokenization	Master only.	Keep Left, Keep Right, Prefix.	Yes	Digits, Alpha, Date	FPE,FF1,Date, Random, Luhn Checks
Encryption	Yes	N/A	No	N/A	N/A

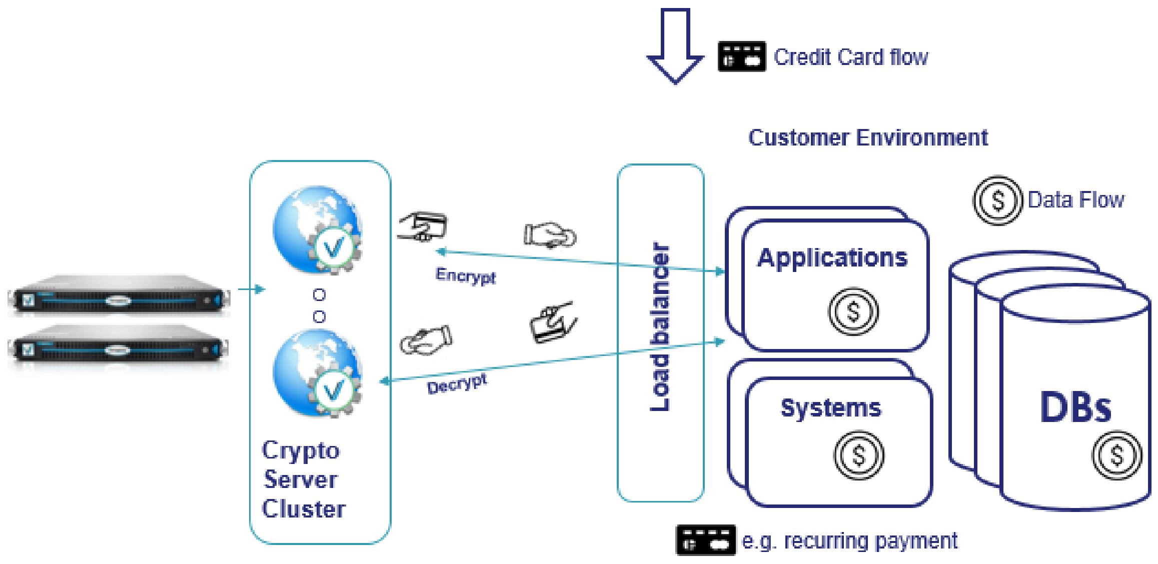
The factors in the matrix above can be used as a starting point for project teams. Weighs should be assigned to the various factors depending priorities of the customer so an appropriate technology can be chosen.   For a more detailed description on the differences between tokenization and encryption, please see the Appendix or this link.

https://www.rsaconference.com/events/us17/agenda/sessions/4591-cybersecurity-vs-tokenization

Architecture

VAE Crypto Server

Most implementations will have at least two Crypto Servers handling requests.   The crypto server operates as a cluster and it is easy to add more nodes to the cluster if needed.



Note: The load balancer is not included with the Cyrpto Server and must be implemented by the customer. For a more detail diagram of the Crypto Server Architecture please see the Appendix.

Crypto Server REST API Requirements

The following requirements are necessary for VTS 2.2 and above.

- Data Security Manager (DSM) version 6.0.3 or greater
- Administrator access to the DSM:
    - System or Domain administrators: required for REST API method
    - Security administrators: "Key" Role required for all methods

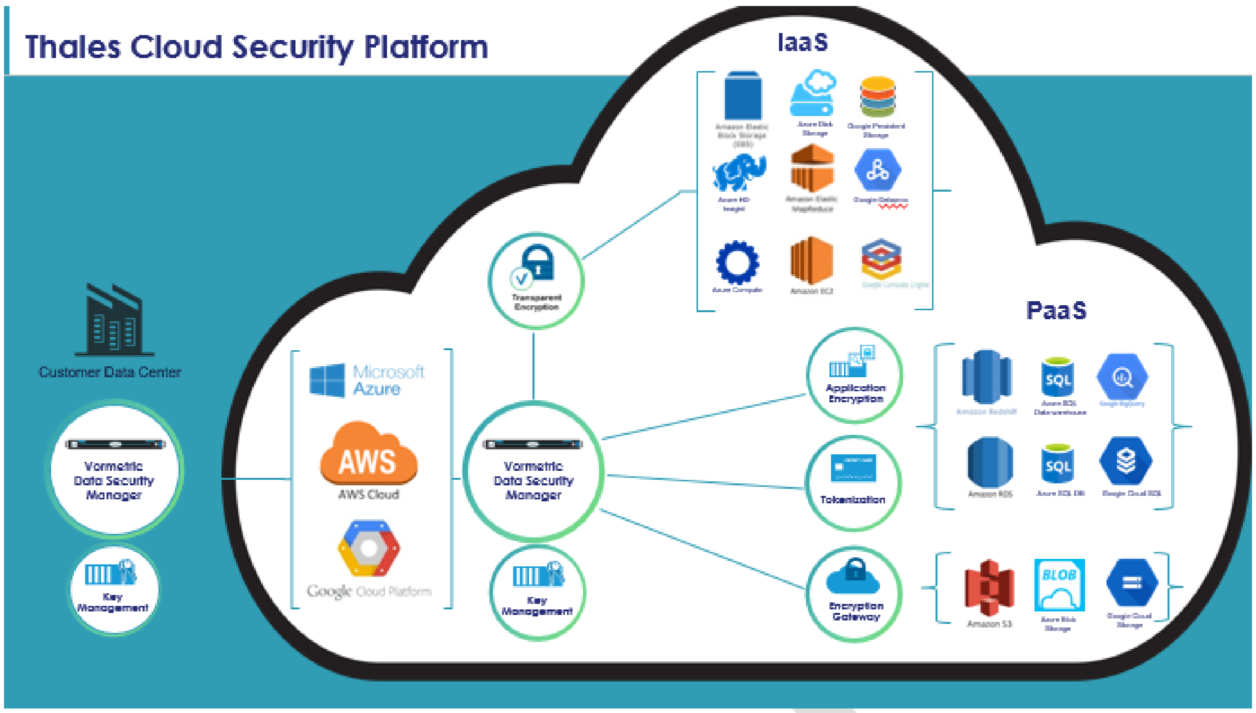
Documentation

- https://yourcryptoserver/vts/crypto/v1/doc/
- VTS_2.2.95_Install_Admin_Prog_Guide.pdf

Please note any reference to VTS used in this document it is to be considered synonymous with the Vormetric Crypto Server.

AWS PAAS

PAAS based capabilities do not allow for any kind of installation of software which means that any encryption of data must be implemented during the ingest process. Listed below is a diagram showing how either Application Encryption or Tokenization can be implemented to protect sensitive data in one of the PAAS based products.



Example Applications

The first sample application used is for the use case when a customer is using AWS DynamoDB and the second example shows how to protect data when using AWS RDS. Both examples are using tokenization vs encryption, but the encryption API's would work basically the same. Sample helper classes are provided in the Appendix that show how encryption and decryption can easily be implemented as well.

AWS DynamoDB Application Functionality

This sample application creates a DynamoDB table and then inserts data into it using the Vormetric Tokenization API to protect the address. Amazon offers the ability to encrypt data at rest for DynamoDB at database creation time, but this does not protect the sensitive data from any kind malicious query.

There is an open source client library that encrypts data in transit called the Amazon DynamoDB Encryption Client. It currently only support Java and python. Also, the DynamoDB Encryption Client is not compatible with the AWS Encryption SDK or the Amazon S3 Encryption Client. You cannot encrypt with one client-side library and decrypt with another. The Amazon DynamoDB Encryption Client is not able to tokenize data as well. See appendix for more information.

Application Design

The example application documented below uses a couple of Vormetric helper classes that encapsulate the tokenization and detokenization functionality. Project teams building applications should leverage similar designs for best practices.

Environment Variables.

In order to make calls to the Crypto Server it is necessary to have a number of input values such as user credentials, crypto server ip address or name, key to be used etc. This application example uses these environment variables when making a call. A simple batch script can be used to call this application and set the environment variables. Here is an example for windows.

```
set vtsdebug=1
SET vtsuserid=vtsroot
echo %vtsuserid%
set vtspassword=yourpwd!
echo %vtspassword%
set vtstokenserver=192.168.159.141
set vcstokengroup=TextTokenGroup
set vcstokentemplate=Text
start JavaVAEUI.jar
```

Environment variables are also needed for the AWS credentials as well.

Application Modifications.

As you can see from below there are only three places the application needs modifications in order to implement the ability to encrypt and decrypt data, imports, define a new instance of the setting class and calling the token or detoken helper methods.

```
import java.util.*;
import com.amazonaws.services.dynamodbv2.*;
import com.amazonaws.services.dynamodbv2.model.AttributeValue;

import com.vormetric.rest.sample.VormetricCryptoServerHelper;
import com.vormetric.rest.sample.VormetricCryptoServerSettings;

public class AWSDynamoDBAndVTS {

	//
	private static final String ADDRESS = "Address";
	private static final String EMAIL = "EmailAddress";
	private static final String TABLE = "VormetricDynamoDBCrypto";
	final static AmazonDynamoDB ddb = new ~~AmazonDynamoDBClient()~~;

	public static void main(final String[] args) throws Exception {

		String action = "encrypt";

		VormetricCryptoServerSettings vcs = new VormetricCryptoServerSettings();
```

```
        String address = "Alice Lovelace, 123 Anystreet Rd., Anytown, USA";
//              String address = "34567883";
                String results = null;
                String email = "alice@example.com";
                action = "tokenize";
                results =
VormetricCryptoServerHelper.doTokenizeData(vcs.getvcstokenserver(),
vcs.getvcsuserid(), vcs.getvcspassword(),  vcs.getvcsTokengroup(),
vcs.getvcsTokentemplate(), address, action);
                // Alice stores her address

                final Map<String, AttributeValue> item = new HashMap<>();
                item.put(EMAIL, new AttributeValue().withS(email));
                item.put(ADDRESS, new AttributeValue().withS(results));
                ddb.putItem(TABLE, item);

                address = "Mallory Evesdotir, 321 Evilstreed Ave., Despair, USA";
                results = null;
                email = "mallory@example.com";

                results =
VormetricCryptoServerHelper.doTokenizeData(vcs.getvcstokenserver(),
vcs.getvcsuserid(), vcs.getvcspassword(), vcs.getvcsTokengroup(),
vcs.getvcsTokentemplate(), address, action);
                // Alice stores her address
                final Map<String, AttributeValue> item2 = new HashMap<>();
                item2.put(EMAIL, new AttributeValue().withS(email));
                item2.put(ADDRESS, new AttributeValue().withS(results));
                ddb.putItem(TABLE, item2);

                action = "detokenize";
                email = "alice@example.com";
                final Map<String, AttributeValue> item3 = ddb
                                .getItem(TABLE, Collections.singletonMap(EMAIL, new
AttributeValue().withS(email))).getItem();

                address = item3.get(ADDRESS).getS();
                System.out.println("address in dyndb " +  address);
                results =
VormetricCryptoServerHelper.doDeTokenizeData(vcs.getvcstokenserver(),
vcs.getvcsuserid(), vcs.getvcspassword(),  vcs.getvcsTokengroup(),
vcs.getvcsTokentemplate(), address, action);

                email = "mallory@example.com";
                final Map<String, AttributeValue> item4 = ddb
                                .getItem(TABLE, Collections.singletonMap(EMAIL, new
AttributeValue().withS(email))).getItem();

                address = item4.get(ADDRESS).getS();
                results =
VormetricCryptoServerHelper.doDeTokenizeData(vcs.getvcstokenserver(),
vcs.getvcsuserid(), vcs.getvcspassword(),  vcs.getvcsTokengroup(),
vcs.getvcsTokentemplate(), address, action);

        }

}
```

```
See Appendix for details on helper classes.
```

Sample Output:

```
url =
sun.net.www.protocol.https.DelegateHttpsURLConnection:https://192.168.159.141/vts/res
t/v2.0/tokenize/
Tokenized payload: {"token":"u7Di4 LAypMXVq, Ox6 tVFMR5dbe EX., PayOXUA,
RD5","status":"Succeed"}
Tokendata u7Di4 LAypMXVq, Ox6 tVFMR5dbe EX., PayOXUA, RD5
url =
sun.net.www.protocol.https.DelegateHttpsURLConnection:https://192.168.159.141/vts/res
t/v2.0/tokenize/
Tokenized payload: {"token":"LkwFbKu 0ohNSLm8x, cx2 ebEDyvOgpB HJ9., wdUSQMg,
4w0","status":"Succeed"}
Tokendata LkwFbKu 0ohNSLm8x, cx2 ebEDyvOgpB HJ9., wdUSQMg, 4w0
address in dyndb u7Di4 LAypMXVq, Ox6 tVFMR5dbe EX., PayOXUA, RD5
Original data payload: {"data":"Alice Lovelace, 123 Anystreet Rd., Anytown,
USA","status":"Succeed"}
Original data Alice Lovelace, 123 Anystreet Rd., Anytown, USA
Original data payload: {"data":"Mallory Evesdotir, 321 Evilstreed Ave., Despair,
USA","status":"Succeed"}
Original data Mallory Evesdotir, 321 Evilstreed Ave., Despair, USA
```

AWS RDS Example

This example uses JDBC to insert data into a MySQL instance of RDS.

Application Design

The example application documented below uses a couple of Vormetric helper classes that encapsulate the tokenization and detokenization functionality. Project teams building applications should leverage similar designs for best practices. It also has a helper class to manage the JDBC connection.

Application Modifications.

As you can see from below there are only three places the application needs modifications in order to implement the ability tokenize data, imports, define a new instance of the setting class and calling the token or detoken helper methods.

Here is the code to test using standard MySQL JDBC class file and VTS tokenization. It creates 1000 rows of data and does a commit every 100 records.

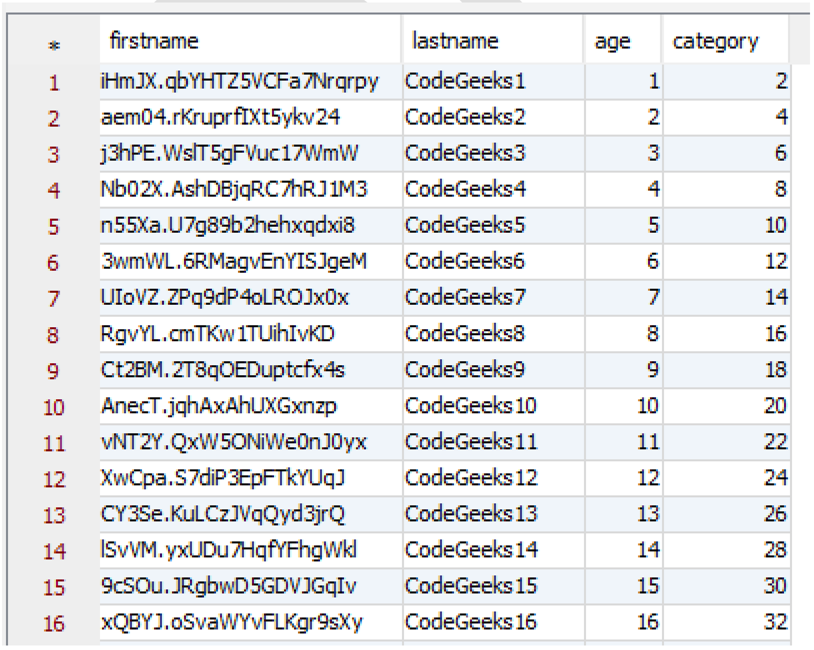
```
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.util.Calendar;
import java.util.Date;

import com.vormetric.rest.sample.VormetricCryptoServerHelper;
import com.vormetric.rest.sample.VormetricCryptoServerSettings;

public class AWSMysqlRDSandVTSExample {

        public static void main(String[] args) throws Exception {
```

```
                    String action = "tokenize";
                    VormetricCryptoServerSettings vcs = new
VormetricCryptoServerSettings();

                    Calendar calendar = Calendar.getInstance();

                    // Get start time (this needs to be a global variable).
                    Date startDate = calendar.getTime();


                    Connection connection = ConnectionObject.getConnection();
                    String SQL = "insert into person values (?,?,?,?)";
                    int batchSize = 100;
                    //            int batchSize = 50;
                    int count = 0;
                    int[] result;

                    connection.setAutoCommit(false);
                    PreparedStatement pstmt = connection.prepareStatement(SQL);
                    String results = null;
                    String sensitive = null;
                    for(int i=1;i<= 1000;i++){
                       sensitive = "Java"+i*Math.random();
                           results =
VormetricCryptoServerHelper.doTokenizeData(vcs.getvcstokenserver(),
vcs.getvcsuserid(), vcs.getvcspassword(), vcs.getvcsTokengroup(),
vcs.getvcsTokentemplate(), sensitive, action);

                      pstmt.setString(1,results);
                      pstmt.setString(2,"CodeGeeks"+i);
                      pstmt.setInt(3,i);
                      pstmt.setInt(4, i+i);
                      pstmt.addBatch();

                      count++;

                      if(count % batchSize == 0){
                             System.out.println("Commit the batch");
                             result = pstmt.executeBatch();
                             System.out.println("Number of rows inserted: "+
result.length);
                             connection.commit();
                      }
                    }

                    if(pstmt!=null)
                           pstmt.close();
                    if(connection!=null)
                           connection.close();

                     Calendar calendar2 = Calendar.getInstance();

                           // Get start time (this needs to be a global variable).
                           Date endDate = calendar2.getTime();
                           long sumDate = endDate.getTime() - startDate.getTime();
```

```
                                    System.out.println("Total time " + sumDate);

                        }
}
```

```
Commit the batch
Number of rows inserted: 100
Commit the batch
Number of rows inserted: 100
Commit the batch
Number of rows inserted: 100
Commit the batch
Number of rows inserted: 100
Commit the batch
Number of rows inserted: 100
Commit the batch
Number of rows inserted: 100
Commit the batch
Number of rows inserted: 100
Commit the batch
Number of rows inserted: 100
Commit the batch
Number of rows inserted: 100
Commit the batch
Number of rows inserted: 100
Total time 31128
```

Output:



*	firstname	lastname	age	category
1	iHmJX.qbYHTZ5VCFa7Nrqrpy	CodeGeeks1	1	2
2	aem04.rKruprfIXt5ykv24	CodeGeeks2	2	4
3	j3hPE.WslT5gFVuc17WmW	CodeGeeks3	3	6
4	Nb02X.AshDBjqRC7hRJ1M3	CodeGeeks4	4	8
5	n55Xa.U7g89b2hehxqdxi8	CodeGeeks5	5	10
6	3wmWL.6RMagvEnYISJgeM	CodeGeeks6	6	12
7	UIoVZ.ZPq9dP4oLROJx0x	CodeGeeks7	7	14
8	RgvYL.cmTKw1TUihIvKD	CodeGeeks8	8	16
9	Ct2BM.2T8qOEDuptcfx4s	CodeGeeks9	9	18
10	AnecT.jqhAxAhUXGxnzp	CodeGeeks10	10	20
11	vNT2Y.QxW5ONiWe0nJ0yx	CodeGeeks11	11	22
12	XwCpa.S7diP3EpFTkYUqJ	CodeGeeks12	12	24
13	CY3Se.KuLCzJVqQyd3jrQ	CodeGeeks13	13	26
14	lSvVM.yxUDu7HqfYFhgWkl	CodeGeeks14	14	28
15	9cSOu.JRgbwD5GDVJGqIv	CodeGeeks15	15	30
16	xQBYJ.oSvaWYvFLKgr9sXy	CodeGeeks16	16	32

Here is the code for the connections class which connects to the RDS instance.

```
import java.sql.Connection;
import java.sql.DriverManager;

public class ConnectionObject {
  private static Connection conn = null;

  public static Connection getConnection(){
              String url = "jdbc:mysql://vormetric-crypto-server.cosqxcdfdl0s.us-east-
1.rds.amazonaws.com:3306/vtsdemo?useSSL=false";

            try{
                      Class.forName("com.mysql.jdbc.Driver").newInstance();
                      conn = DriverManager.getConnection(url,"yourrdsuserid","yourrdspwd");
            }catch(Exception e){
                      e.printStackTrace();
            }
            return conn;
  }
}
```

Appendix

Helper Classes Examples

As mentioned helper classes encapsulate most of the tokenize or detokenize functionality. Project teams building applications should leverage similar designs for best practices. Here are a couple of the methods that were used in the example application.

This is the encrypt method in the `VormetricCryptoServerHelper class file.`

```
public static String doTokenizeData(String tokenserver, String userid, String
password, String tokengroup,
                    String tokentemplate, String data, String action) throws
Exception {

             VormetricCryptoServerHelper.disableCertValidation();
             HttpsURLConnection con =
VormetricCryptoServerHelper.getUrlConnection(tokenserver, action);
             System.out.println("url = " + con.toString());
             String encidpwd =
VormetricCryptoServerHelper.getEncodedIdPassword(userid, password);

             String payload = "{\"data\":\"" + data + "\",\"tokengroup\":\"" +
tokengroup + "\",\"tokentemplate\":\""
                          + tokentemplate + "\"}";

             con.setRequestMethod("POST");
             con.setRequestProperty("Authorization", "Basic " + encidpwd);
             con.setDoOutput(true);
```

```
		OutputStreamWriter output = new
OutputStreamWriter(con.getOutputStream());
		output.write(payload);
		output.close();

		BufferedReader br = new BufferedReader(new
InputStreamReader(con.getInputStream()));
		String brdata;
		StringBuffer returnvalue = new StringBuffer();
		while ((brdata = br.readLine()) != null) {
			returnvalue.append(brdata);
		}
		System.out.println("Tokenized payload: " + returnvalue.toString());
		String tokenvalue = JsonPath.read(returnvalue.toString(),
"$.token").toString();

		System.out.println("Tokendata " + tokenvalue);
		br.close();
		con.disconnect();

		return tokenvalue;

	}

//Sample code to detokenize

	public static String doDeTokenizeData(String tokenserver, String userid,
String password, String tokengroup,
			String tokentemplate, String data, String action) throws
Exception {

		VormetricCryptoServerHelper.disableCertValidation();
		HttpsURLConnection con =
VormetricCryptoServerHelper.getUrlConnection(tokenserver, action);

		String encidpwd =
VormetricCryptoServerHelper.getEncodedIdPassword(userid, password);

		String payload = "{\"token\":\"" + data + "\",\"tokengroup\":\"" +
tokengroup + "\",\"tokentemplate\":\""
				+ tokentemplate + "\"}";

		con.setRequestMethod("POST");
		con.setRequestProperty("Authorization", "Basic " + encidpwd);
		con.setDoOutput(true);

		OutputStreamWriter output = new
OutputStreamWriter(con.getOutputStream());
		output.write(payload);
		output.close();

		BufferedReader br = new BufferedReader(new
InputStreamReader(con.getInputStream()));

		String brdata;
		StringBuffer returnvalue = new StringBuffer();
```

```
                while ((brdata = br.readLine()) != null) {
                        returnvalue.append(brdata);
                }
                System.out.println("Original data payload: " + returnvalue.toString());
                String originaldata = JsonPath.read(returnvalue.toString(),
"$.data").toString();

                System.out.println("Original data " + originaldata);

                br.close();
                con.disconnect();

                return originaldata;
        }
```

It is also possible to encrypt and decrypt as well. Here are some sample helper classes.

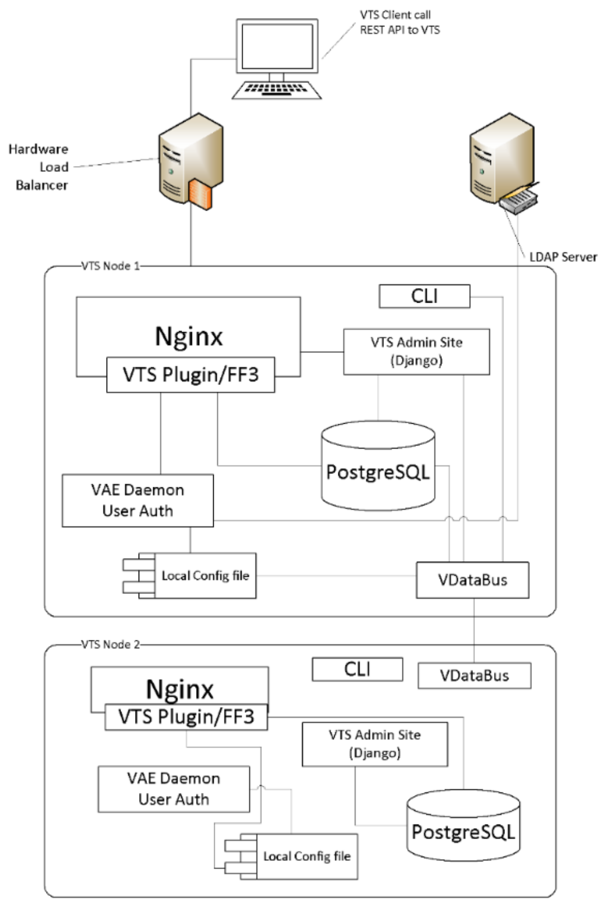
```
        public static String doEncryptData(String tokenserver, String userid, String
password, String inputdata, String alg,
                        String ivnumber, String action, String encryptdecryptkey) throws
Exception {

                disableCertValidation();
                HttpsURLConnection con = getUrlConnection(tokenserver, action);
                String encidpwd = getEncodedIdPassword(userid, password);
                String payload = "{\"plaintext\":\"" +
Base64.getEncoder().encodeToString(inputdata.getBytes("UTF-8"))
                                + "\",\"alg\":\"" + alg + "\",\"params\" : {\"iv\":\""
                                +
Base64.getEncoder().encodeToString(ivnumber.getBytes("UTF-8")) + "\"},\"kid\": \"" +
encryptdecryptkey
                                + "\"}";
                con.setRequestMethod("POST");
                con.setRequestProperty("Authorization", "Basic " + encidpwd);
                con.setDoOutput(true);
                OutputStreamWriter output = new
OutputStreamWriter(con.getOutputStream());
                output.write(payload);
                output.close();
                BufferedReader br = new BufferedReader(new
InputStreamReader(con.getInputStream()));
                String brdata;
                StringBuffer returnvalue = new StringBuffer();
                while ((brdata = br.readLine()) != null) {
                        returnvalue.append(brdata);
                }
                String base64string = JsonPath.read(returnvalue.toString(),
"$.ciphertext").toString();
                br.close();
                con.disconnect();
                return base64string;
        }
```

This is the decrypt method in the `VormetricCryptoServerHelper class file`

```
	public static String doDeCryptData(String tokenserver, String userid, String
password, String inputdata, String alg,
			String ivnumber, String action, String encryptdecryptkey) throws
Exception {

		disableCertValidation();
		HttpsURLConnection con = getUrlConnection(tokenserver, action);
		String encidpwd = getEncodedIdPassword(userid, password);
		String payload = "{\"ciphertext\":\"" + inputdata + "\",\"alg\":\"" +
alg + "\",\"params\" : {\"iv\":\""
				+
Base64.getEncoder().encodeToString(ivnumber.getBytes("UTF-8")) + "\"},\"kid\": \"" +
encryptdecryptkey
				+ "\"}";
		con.setRequestMethod("POST");
		con.setRequestProperty("Authorization", "Basic " + encidpwd);
		con.setDoOutput(true);
		OutputStreamWriter output = new
OutputStreamWriter(con.getOutputStream());
		output.write(payload);
		output.close();
		BufferedReader br = new BufferedReader(new
InputStreamReader(con.getInputStream()));
		String brdata;
		StringBuffer returnvalue = new StringBuffer();
		while ((brdata = br.readLine()) != null) {
			returnvalue.append(brdata);
		}
		String base64string = JsonPath.read(returnvalue.toString(),
"$.plaintext").toString();
		byte[] base64bytes = Base64.getDecoder().decode(base64string);
		String base64originaldata = new String(base64bytes);
		br.close();
		con.disconnect();
		return base64originaldata;
	}
```

The VormetricCryptoServerSettings contains getters and setters for the various attributes needed for the Crypto Server.

Crypto Server Detailed Architecture.



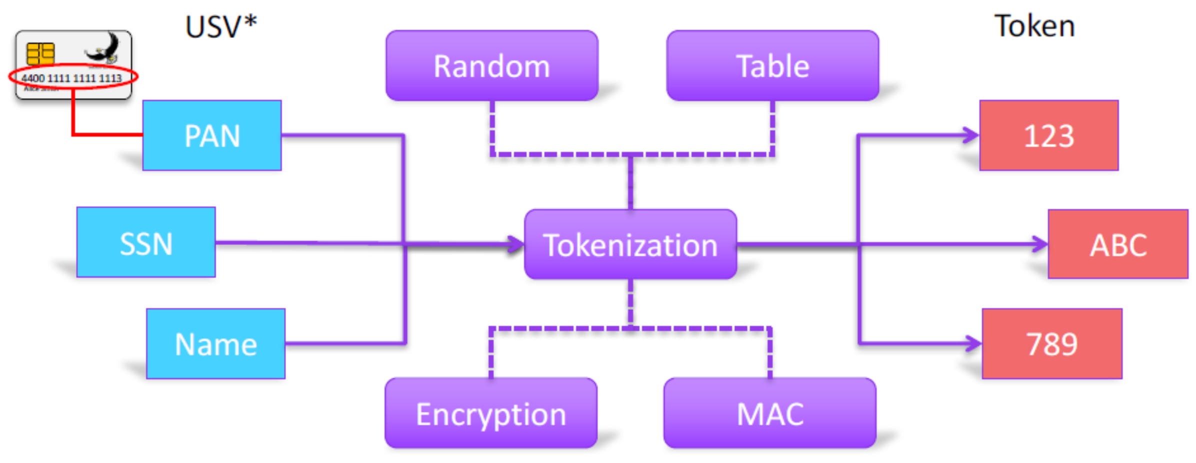
Types of Tokenization

https://www.rsaconference.com/events/us17/agenda/sessions/4591-cybersecurity-vs-tokenization

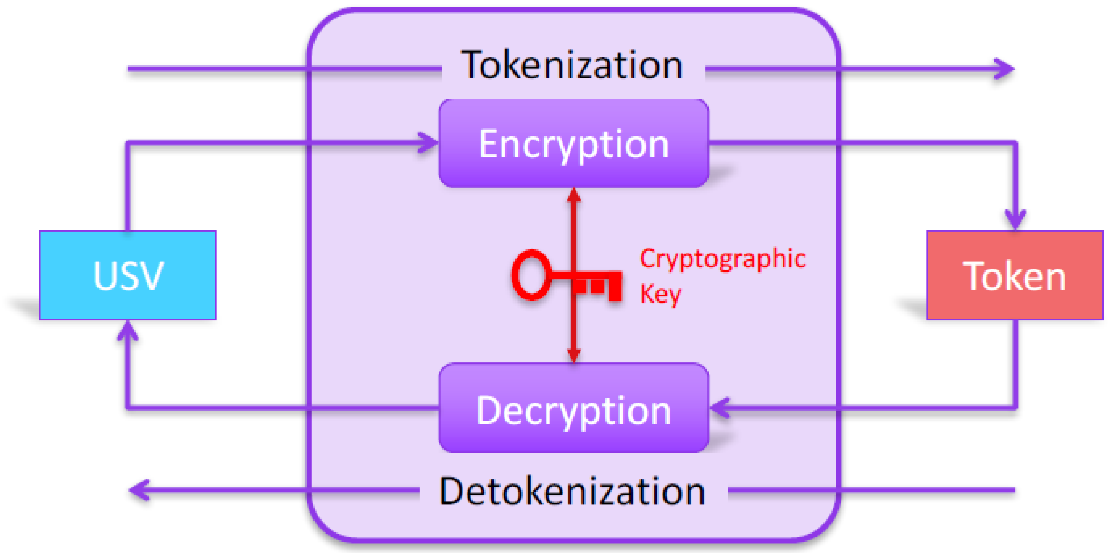
Definition of Tokenization
PCI: Tokenization is a process by which the Primary Account Number (PAN) is replaced with a surrogate value called a token.
— Index Token is a cryptographic token that replaces the PAN, based on a given index for an
unpredictable value
ANSI X9: The process of mapping a plaintext value (i.e., the underlying sensitive value) to an existing or newly-generated surrogate value (i.e., a token). A token is a surrogate value used in place of the original value in certain, well-defined situations, but that is not used in place of the original value in every way that the original value is used.
EMV: A process by which the Primary Account Number (PAN) is replaced with a

surrogate value called a Payment Token. Tokenization may be undertaken to enhance transaction efficiency, improve transaction security, increase service transparency, or to provide a method for third-party enablement.
Conflicting definitions, so, what is a token?

What is Tokenization and Why Do It?

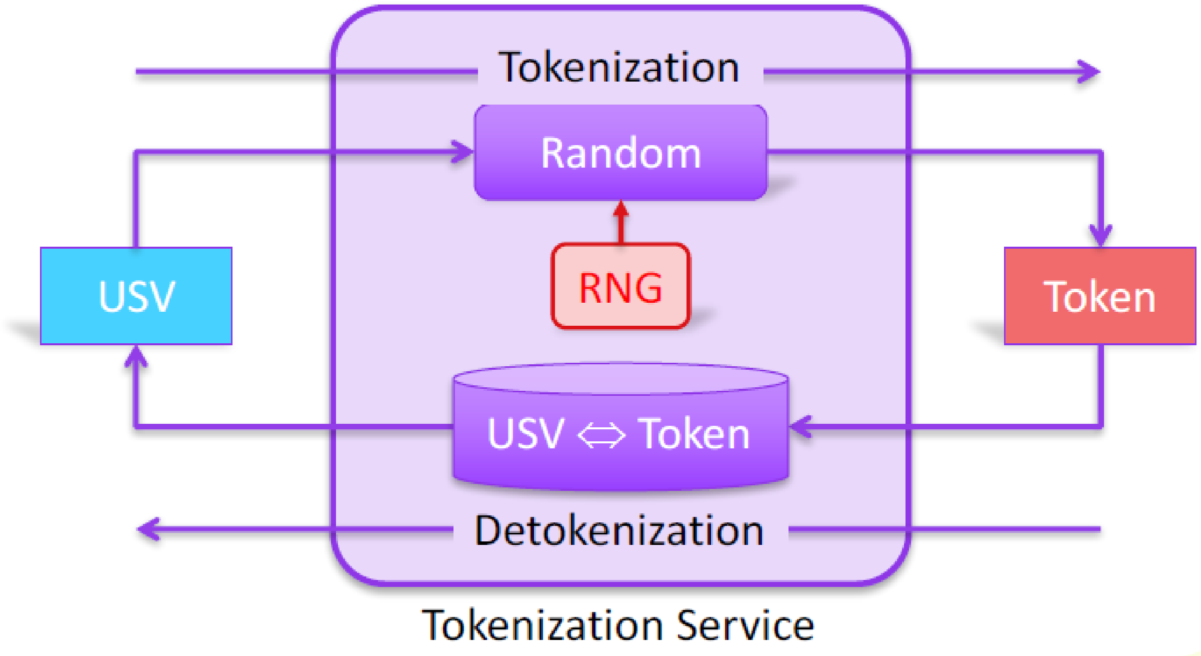


*Underlying Sensitive Value – protecting data



Tokenization Service

All non-random formats use this method for tokenization.



In a random or random-Luhn format, the template uses a pseudo-random number generator (PRNG) instead of an encryption algorithm. In addition, the very first symmetric key configured in the VTS GUI is not a true encryption key, but instead acts as a seed value for the PRNG.

Amazon DynamoDB Encryption Client

The Amazon DynamoDB Encryption Client is a software library that helps you to protect your table data before you send it to Amazon DynamoDB. After you create and configure the required components, the DynamoDB Encryption Client transparently encrypts and signs your table items when you add them to a table, and verifies and decrypts them when you retrieve them. This is accomplished by the creation of special mapper classes that must be used in order to implement this capability. This kind of framework could also be implemented for the Vormetric encryption and tokenization as well.

You can use the DynamoDB Encryption Client with encryption keys from any source, including your custom implementation or a cryptography service, such as AWS Key Management Service (AWS KMS) or AWS CloudHSM. As of December 2018 they are only available in Java and Python.

Also, the DynamoDB Encryption Client is not compatible with the AWS Encryption SDK or the Amazon S3 Encryption Client. You cannot encrypt with one client-side library and decrypt with another.