THALES

CipherTrust Manager Platform

Using Thales CipherTrust Manager REST API with AWS PAAS

Document Version 1

Contents

PREFACE

Note: In Sept of 2020 Thales has rebranded the KeyManager named KeySecure or KeySecure Next Gen to CipherTrust Manager (CM). It combines capabilities from both the legacy Gemalto KeySecure and the Vormetric Data Security Manager products. Any reference in documentation to KeySecure , NextGen or Data Security Manager can be considered to now be the newly branded CipherTrust Manager (CM) product. See following link for more details:
https://cpl.thalesgroup.com/encryption/ciphertrust-manager

Using CipherTrust Manager with AWS PAAS provides working examples on how to implement scenarios for Bringing Your Own Encryption (BYOE) using REST API's. The benefits of BYOE are as follows:

- Customer owns the encryption and encryption keys
- Keys created in a FIPS certified appliance.
- Single appliance for all encryption keys.
- Consolidated encryption key management and reporting.

DOCUMENTATION VERSION HISTORY

Product/Document Version	Date	Changes
V1.0	4/2021	M. Warner

ASSUMPTIONS

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

- CipherTrust Manager
- Key management
- Data encryption
- Familiarity with REST

GUIDE TO DOCUMENTATION

Related documents are available to registered users on the Thales Web site at https://cpl.thalesgroup.com/ or https://thalesdocs.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in this guide or collateral documents that conflict with the definitions or terms in the License Agreement, shall be superseded by the definitions and terms of the License Agreement.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.

GETTING STARTED

Use Cases

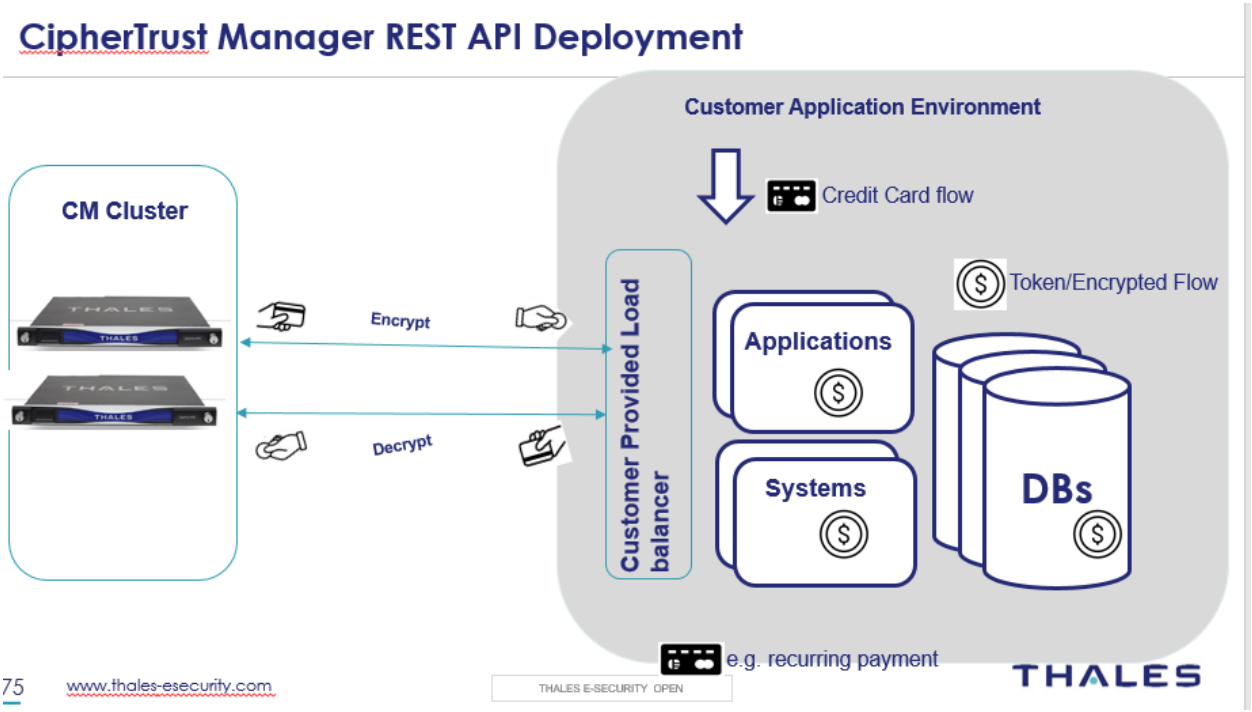
Thales CipherTrust Manager REST API can be used for many different use cases. Typically, it is used for scenarios when a company has sensitive data in a field of a particular file or a column in a database and they want to encrypt or tokenize the sensitive data. Use cases can include:

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

Architecture

CipherTrust Manager REST API Example.

The solution provided in this document is based on having an external key manager to create and manage the encryption keys. This appliance comes in both physical and virtual versions and is call the CipherTrust Manager. Most implementations will have at least two CipherTrust Managers handling requests. The platform operates as a cluster and it is easy to add more nodes to the cluster if needed.



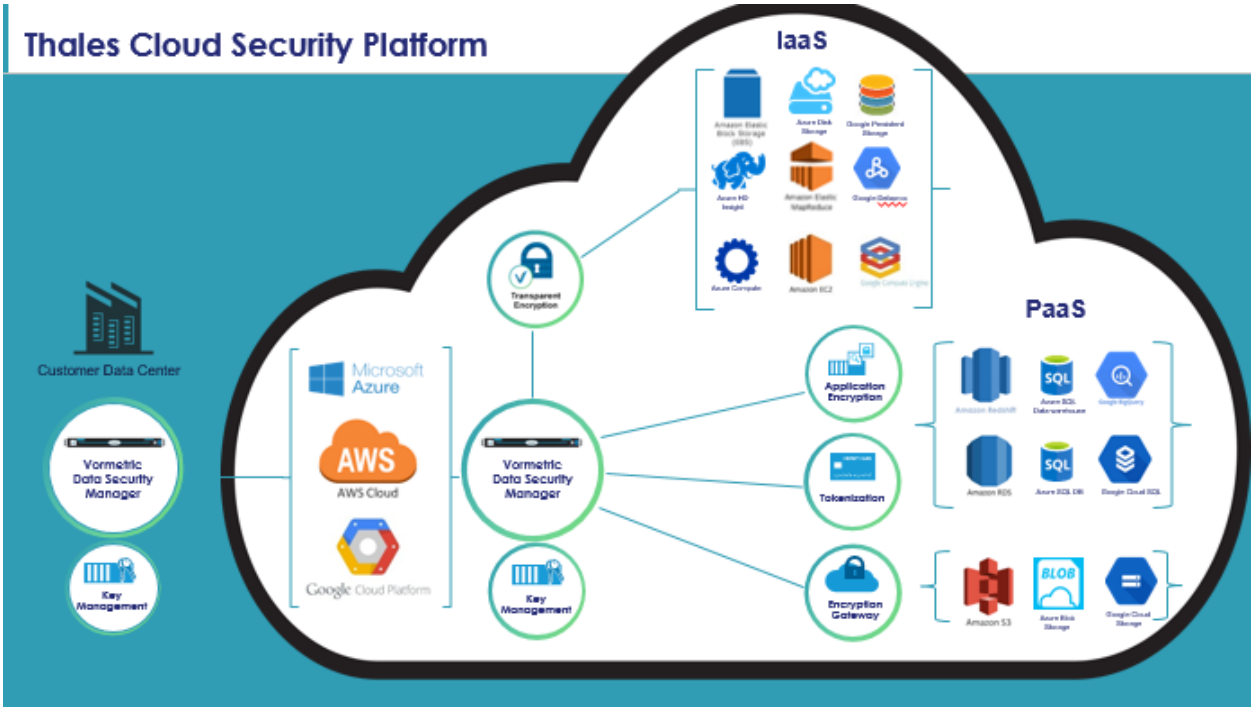
Note: The load balancer is not included with the CipherTrust Manager REST API and must be implemented by the customer. Thales also provides a thick client sdk called CipherTrust Application Protection (formally named protectapp) that can also implement encrypt/decrypt and it does provide load balancer with the client implementation. For more information, please visit: https://cpl.thalesgroup.com/encryption/application-data-protection

Documentation

- https://yourcmipaddress/playground_v2/api

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. Although not demonstrated in this document Thales also has the ability to tokenize data as well.



Example Applications

The examples provided in this document use a helper class located at: https://github.com/thalescpl-io/CipherTrust_Application_Protection/tree/master/rest/src/main/java/com/thales/cm/rest/cmhelper

AWS DynamoDB Application Functionality

This sample application protects sensitive data at ingest time and also decrypts it after doing the insert using the CipherTrust Manager REST API to protect the address and email.

Application Modifications.

As you can see from below there are only a couple of places the application needs to be modified in order to implement the ability to encrypt and decrypt data. The method called is **cmRESTProtect.**

```
package kylookhttp;

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

import com.thales.cm.rest.helper.CipherTrustManagerHelper;

public class AWSDynoDBandCipherTrustREST {
        CipherTrustManagerHelper ctmh = null;
        //
        private static final String ADDRESS = "Address";
        private static final String EMAIL = "EmailAddress";
        private static final String TABLE = "ThlesDynoDBCrypto";
        final static AmazonDynamoDB ddb = new AmazonDynamoDBClient();

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

                AWSDynoDBandCipherTrustREST awsresrest = new AWSDynoDBandCipherTrustREST();
                awsresrest.ctmh = new CipherTrustManagerHelper();

                if (args.length != 4) {
                        System.err.println("Usage: java AWSDynoDBandCipherTrustREST userid password
keyname ctmip  ");
                        System.exit(-1);
                }
                awsresrest.ctmh.dataformat = "alphanumeric";
                awsresrest.ctmh.username = args[0];
                awsresrest.ctmh.password = args[1];
                awsresrest.ctmh.cmipaddress = args[3];
                try {
                        String tkn = awsresrest.ctmh.getToken();

                        awsresrest.ctmh.key = args[2];

                } catch (IOException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                } catch (Exception e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }

                String address = "123 Anystreet Rd., Anytown, USA";

                String results = null;
                String email = "alice@example.com";
                final Map<String, AttributeValue> item = new HashMap<>();

                results = awsresrest.ctmh.cmRESTProtect("fpe", address, "encrypt");
                item.put(ADDRESS, new AttributeValue().withS(results));
                results = null;
                results = awsresrest.ctmh.cmRESTProtect("fpe", email, "encrypt");
                item.put(EMAIL, new AttributeValue().withS(results));

                ddb.putItem(TABLE, item);

                final Map<String, AttributeValue> item2 = new HashMap<>();
```

```
                address = "321 Washington Ave., Despair, USA";
                results = null;
                email = "sam@example.com";
                results = awsresrest.ctmh.cmRESTProtect("fpe", address, "encrypt");
                item2.put(ADDRESS, new AttributeValue().withS(results));
                results = awsresrest.ctmh.cmRESTProtect("fpe", email, "encrypt");
                item2.put(EMAIL, new AttributeValue().withS(results));

                ddb.putItem(TABLE, item2);


                final Map<String, AttributeValue> item3 = ddb
                                .getItem(TABLE, Collections.singletonMap(EMAIL, new
AttributeValue().withS(awsresrest.ctmh.cmRESTProtect("fpe", results, "decrypt")))).getItem();

                address = item3.get(ADDRESS).getS();
                System.out.println("address in dyndb " + address);
                results = awsresrest.ctmh.cmRESTProtect("fpe", address, "decrypt");
                System.out.println("decrpted address in dyndb " + results);

        }
}
```

AWS RDS Example

This example encrypts sensitive data before it does the JDBC insert data into a MySQL instance of RDS. It then decrypts the sensitive data by calling the fpedecryptdata method. This examples uses a mode called format preserved encryption (FPE) which keep the original data type and size of the input data. The benefit of this is the database tables do not have to change to allow for this kind of encryption. Here is the code to test using the MySQL JDBC class file and the CM REST API.

```
package kylookhttp;

import java.io.IOException;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.Calendar;
import java.util.Date;

import com.thales.cm.rest.helper.CipherTrustManagerHelper;

public class AWSMysqlRDSandCipherTrustREST {

        CipherTrustManagerHelper ctmh = null;

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

                AWSMysqlRDSandCipherTrustREST awsresrest = new AWSMysqlRDSandCipherTrustREST();
                awsresrest.ctmh = new CipherTrustManagerHelper();

                if (args.length != 4) {
                        System.err.println("Usage: java AWSMysqlRDSandCipherTrustREST userid password
keyname ctmip  ");
                        System.exit(-1);
                }

                awsresrest.ctmh.username = args[0];
                awsresrest.ctmh.password = args[1];
                awsresrest.ctmh.cmipaddress = args[3];
                awsresrest.ctmh.dataformat = "alphanumeric";
                try {
                        String tkn = awsresrest.ctmh.getToken();

                        awsresrest.ctmh.key = args[2];
```

```
		} catch (IOException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		} catch (Exception e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}

		Calendar calendar = Calendar.getInstance();
		Date startDate = calendar.getTime();

		Connection connection = ConnectionObjectawsrds.getConnection();
		connection.setAutoCommit(false);

		awsresrest.fpeencryptdata(connection);
		awsresrest.fpedecryptdata(connection);

		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);
	}

	void fpeencryptdata(Connection connection) throws Exception {

		String SQL = "insert into person values (?,?,?,?)";
		int batchSize = 2;
		int count = 0;
		int[] result;

		PreparedStatement pstmt = connection.prepareStatement(SQL);
		String results = null;
		String sensitive = null;
		for (int i = 1; i <= 10; i++) {
			sensitive = "bobjones" + i + "@something.com";
			System.out.println(sensitive);
			results = this.ctmh.cmRESTProtect("fpe", sensitive, "encrypt");
			pstmt.setString(1, results);
			sensitive = i + " Anystreet Rd., Anytown, USA";
			System.out.println(sensitive);
			results = this.ctmh.cmRESTProtect("fpe", sensitive, "encrypt");
			pstmt.setString(2, results);
			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();

	}

	void fpedecryptdata(Connection connection) throws Exception {

		Statement stmt = null;
		try {
```

```
                        stmt = connection.createStatement();
                        String results;

                        String sql = "SELECT email, address, age, category FROM person";
                        ResultSet rs = stmt.executeQuery(sql);

                        while (rs.next()) {
                                // Retrieve by column name

                                String email = rs.getString("email");
                                String address = rs.getString("address");
                                int age = rs.getInt("age");
                                int category = rs.getInt("category");
                                System.out.println("Encrypted email: " + email);
                                System.out.println("Encrypted address:" + address);
                                results = this.ctmh.cmRESTProtect("fpe", email, "decrypt");
                                System.out.println("Decrypted email: " + results);

                                results = this.ctmh.cmRESTProtect("fpe", address, "decrypt");
                                System.out.println("Decrypted address:" + results);

                        }
                        rs.close();

                } catch (SQLException se) {
                        // Handle errors for JDBC
                        se.printStackTrace();
                } catch (Exception e) {
                        // Handle errors for Class.forName
                        e.printStackTrace();
                } finally {
                        // finally block used to close resources
                        try {
                                if (stmt != null)
                                        connection.close();
                        } catch (SQLException se) {
                        } // do nothing
                        try {
                                if (connection != null)
                                        connection.close();
                        } catch (SQLException se) {
                                se.printStackTrace();
                        } // end finally try
                } // end try
                System.out.println("Goodbye!");

        }
}
```

Output:

```
Encrypted email: 1OH3uyzgf@uBM83uk0Q.H67
Encrypted address:M qk2ccTdXk t7., yE5aJB9, m7J
Decrypted email: bobjones1@something.com
Decrypted address:1 Anystreet Rd., Anytown, USAEtc…
```

Data in the RDS database.

*	email	address	age	category
1	1OH3uyzgf@uBM83uk0Q.H67	M qk2ccTdXk t7., yE5aJB9, m7J	1	2
2	r5pmd6FRC@RE4nFRIP0.YtQ	p 4hSQfihUt bm., F5gZanh, p0k	2	4
3	dCl5LpoGo@fcZfmm09R.6F8	G iSjcIW6zd wg., WWWYc7H,...	3	6
4	lE2VIYFf9@9TpJQgISN.YBm	7 SslxHlvl0 bD., nmWIjow, 622	4	8
5	HNdF7GyMW@snCIuFYt6.qTV	U bNIvzhi9y p9., Oo9090Q, H...	5	10
6	BCPB9Jwuv@6HUbIbAHN.6OF	H gCtQZmpgG U4., Wm1Iqv4,...	6	12
7	Hg94BZYR0@TgUH6os3q.lWp	d nbWeUwlGl KD., nSGhZ0X, KYl	7	14
8	lSMQ5OEPZ@Kj66oE3j0.16C	p 5vlFler49 r7., iklIvOp, Xac	8	16
9	umiQPFm38@TXBSS2FRN.c0u	I zmhjD0jXc 82., h3cyj6t, I6D	9	18
10	Rint3zo5Dh@h7AvPxsFh.oEY	iZ 8UjqbVNvB Yj., QiJidva, dn1	10	20

Cloud Service Provider & BYOK vs BYOE

This diagram describes the encryption and key management options that are available for customers who have workloads in the cloud. There are various degrees of security and as you can see BYOE offers customers the most control and highest levels of security since they own both the keys and the encryption.