THALES

CipherTrust Manager (CM) Platform

Using the CM REST API & ProtectApp SDK

To encrypt data in AWS PAAS

Document Version 1

Content

PREFACE

Using Thales CipherTrust Manager REST/API or ProtectApp with AWS PAAS document provides examples on how to implement encryption using REST API's or via the ProtectApp JCE SDK.

The use case covered in this document is for scenarios when a customer wants to implement bring your own encryption (BYOE) to a cloud service provider PAAS service. The benefit of BYOE is increased security because both the encryption and encryption keys are owned by the customer.

DOCUMENTATION VERSION HISTORY

Product/Document Version	Date	Changes
V1.0	08/2020	M. Warner -Initial document release

ASSUMPTIONS

This documentation assumes the reader is familiar with the following topics:

- Java
- Key management
- Data encryption
- Familiarity with REST

This document uses the AWS PAAS platform as an example but the code would be the same if Azure PAAS environment were utilized instead.

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

GUIDE TO Thales DOCUMENTATION

Related documents are available to registered users on the Thales Web site at
https://supportportal.thalesgroup.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definitions or terms in the License Agreement, shall be superseded by the definitions and terms of the License Agreement.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

The goal of this document is to provide some simple examples showing how the CM REST API and the CM ProtectApp SDK can be used to BYOE to PAAS services in the cloud. The code examples are not production ready samples, but only to demonstrate capability.

Thales CipherTrust Manager REST/API or ProtectApp 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 the sensitive data. Keep in mind both the REST API and the ProtectAPP SDK provide Format Preserved Encryption (FPE) which preserves the format of the original 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.

The examples provided in this document are showing how to encrypt sensitive data in AWS RDS Mysql database

Methods

Listed below is a matrix that describes some of the differences between the REST API and encryption using the Thales CipherTrust Manager ProtectApp SDK.

	Agent Install	Authentication options	API	Key Rotation	Load Bal	Connection Pooling	Key cache	Performance	Platforms
CM Protect App	Yes	Username & Password,SSL with Client Certificates, Local SafeNet KeySecure Configuration	JCE	Yes	Yes	Yes	Yes	Fastest	Unix,Linux and windows
CM REST API	No	JWT and username & password.	REST	Yes	No	No	No		All

The factors in the matrix above can be used as a starting point for project teams. Weights should be assigned to the various factors depending on the priorities of the customer so an appropriate technology can be chosen.

CipherTrust Manager REST API Requirements

The following requirements are necessary for the REST API.

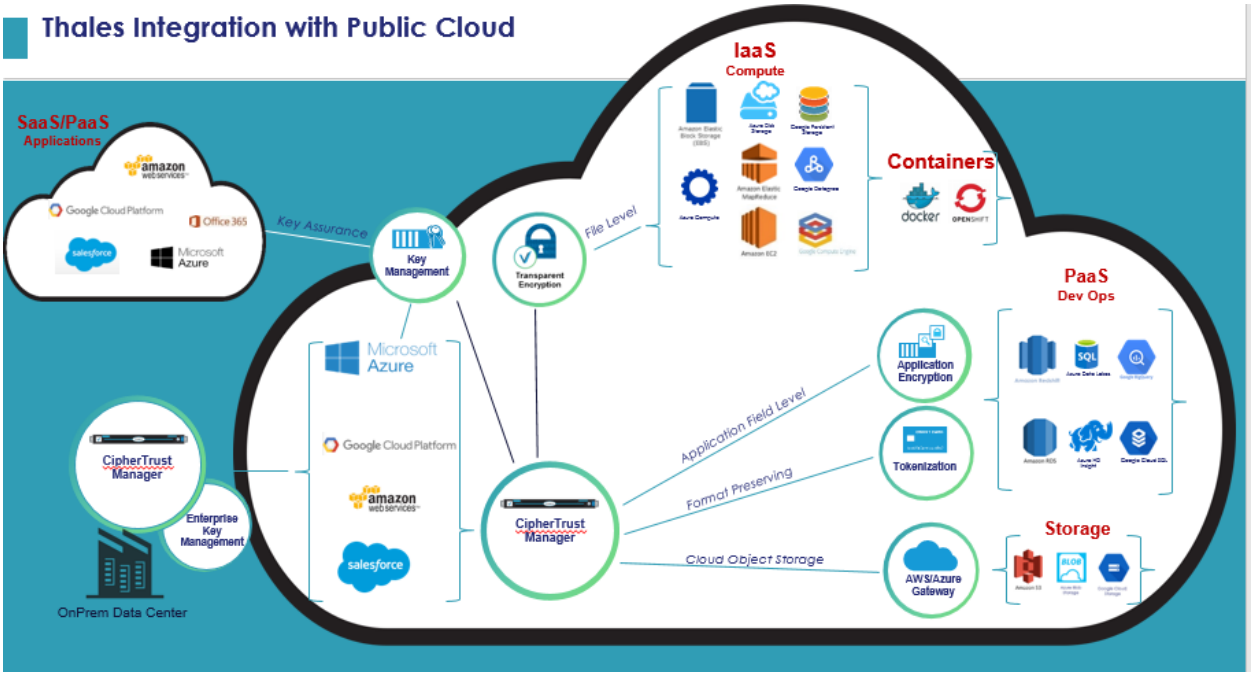
- KeySecure/Next Gen 1.9.1 or greater
- Administrator access to the CM:
    - Security administrators: “Access to the Key Users Group.”

Documentation

- SafeNet ProtectApp JCE User Guide
- https://ciphertrustmanagerip/playground_v2/api/Alarms

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 protect sensitive data in one of the PAAS based products.



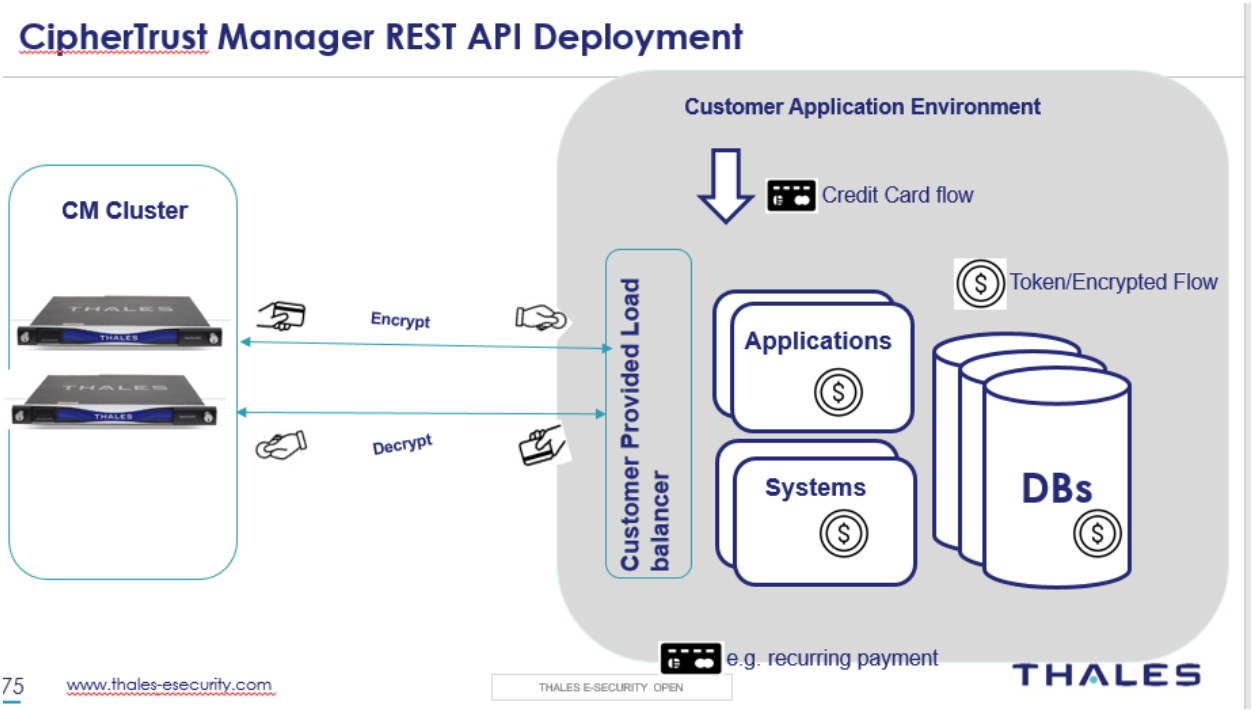
Example Applications.

The examples below used the same key in CM for encryption using an AES 256bit key and 15 bytes of random data to be encrypted.

CipherTrust Manager REST API Example.

Architecture

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. As you can see below when using the REST API the customer must provide their own load balancer.



The first example uses the CipherTrust Manager REST API to encrypt the data. There are two different modes or algorithms that were used, GCM and Format Preserved Encryption (FPE). These examples use different endpoint URL's and the format of the json payload is different.

Import statements were excluded to keep the document short. Please note this code is for testing only and should not be used for production. As you can see, the only 3rd party libraries that were used were for jsonpath to parse json and `OkHttpClient` to make rest calls.

Here is the code to test using standard MySQL JDBC class file and CipherTrust Manager REST API. As an example when passing parameters, it can create 1000 rows of data and a commit every 100 records.

```
public class AWSMysqlRDSCMRestApi {
	String token = null;
	String key = null;
```

```
	String CMip = "192.168.1.25";
	// public static final String CMIP = "192.168.1.25";
	public static final String endbracket = "}";
	public static final String quote = "\"";
	public static final String comma = ",";

	public static final String plaintexttag = "{\"plaintext\":";
	public static final String tag = "kBr5A0fbPjPg7lS1bB6wfw==";
	public static final String iv = "VCC3VwxWu6Z6jfQw";
	public static final String mode = "gcm";
	public static final String aadtag = "\"aad\":";
	public static final String idtag = "\"id\":";
	public static final String typetag = "\"type\":";
	public static final String type = "name";
	public static final String aad = "YXV0aGVudGljYXRl";
	public static final String ciphertexttag = "{\"ciphertext\":";
	public static final String tagtag = "\"tag\":";
	public static final String ivtag = "\"iv\":";
	public static final String modetag = "\"mode\":";

	public static final MediaType JSON = MediaType.get("application/json;
charset=utf-8");
	public static final MediaType JSONTXT = MediaType.get("text/plain");
	OkHttpClient client = getUnsafeOkHttpClient();

	String postfpe(String url, String text) throws IOException {
		RequestBody body = RequestBody.create(JSONTXT, text);
		Request request = new
Request.Builder().url(url).post(body).addHeader("Authorization", "Bearer " +
this.token)
				.addHeader("Accept", "text/plain").addHeader("Content-
Type", "text/plain").build();
		try (Response response = client.newCall(request).execute()) {
			return response.body().string();
		}
	}

	String post(String url, String json) throws IOException {
		RequestBody body = RequestBody.create(json, JSON);
		Request request = new
Request.Builder().url(url).post(body).addHeader("Authorization", "Bearer " +
this.token)
				.addHeader("Accept",
"application/json").addHeader("Content-Type", "application/json").build();
		try (Response response = client.newCall(request).execute()) {
			return response.body().string();
		}
	}

	private static String getToken(String CMip, String username, String password)
throws IOException {

		OkHttpClient client = getUnsafeOkHttpClient();
		MediaType mediaType = MediaType.parse("application/json");

		String grant_typetag = "{\"grant_type\":";
		String grant_type = "password";
```

```
            String passwordtag = "\"password\":";
            String usernametag = "\"username\":";
            String labels = "\"labels\": [\"myapp\",\"cli\"]}";

            String authcall = grant_typetag + quote + grant_type + quote + comma +
usernametag + quote + username + quote
                        + comma + passwordtag + quote + password + quote + comma +
labels;

            RequestBody body = RequestBody.create(mediaType, authcall);
            Request request = new Request.Builder().url("https://" + CMip +
"/api/v1/auth/tokens").method("POST", body)
                        .addHeader("Content-Type", "application/json").build();

            Response response = client.newCall(request).execute();
            String returnvalue = response.body().string();
            System.out.println("response " + returnvalue);
            String jwt = JsonPath.read(returnvalue.toString(), "$.jwt").toString();
            String refreshtoken = JsonPath.read(returnvalue.toString(),
"$.refresh_token").toString();
            System.out.println("jwt = " + jwt);
            System.out.println("refresh = " + refreshtoken);
            return jwt;

      }

      private static OkHttpClient getUnsafeOkHttpClient() {
            try {
                  // Create a trust manager that does not validate certificate
chains
                  final TrustManager[] trustAllCerts = new TrustManager[] { new
X509TrustManager() {
                        @Override
                        public void
checkClientTrusted(java.security.cert.X509Certificate[] chain, String authType)
                                    throws CertificateException {
                        }

                        @Override
                        public void
checkServerTrusted(java.security.cert.X509Certificate[] chain, String authType)
                                    throws CertificateException {
                        }

                        @Override
                        public java.security.cert.X509Certificate[]
getAcceptedIssuers() {
                              return new java.security.cert.X509Certificate[] {};
                        }
                  } };

                  // Install the all-trusting trust manager
                  final SSLContext sslContext = SSLContext.getInstance("SSL");
                  sslContext.init(null, trustAllCerts, new
java.security.SecureRandom());
                  // Create an ssl socket factory with our all-trusting manager
```

```
                  final SSLSocketFactory sslSocketFactory =
sslContext.getSocketFactory();

                  OkHttpClient.Builder builder = new OkHttpClient.Builder();
                  builder.sslSocketFactory(sslSocketFactory, (X509TrustManager)
trustAllCerts[0]);
                  builder.hostnameVerifier(new HostnameVerifier() {
                        @Override
                        public boolean verify(String hostname, SSLSession session)
{
                              return true;
                        }
                  });

                  OkHttpClient okHttpClient = builder.build();
                  return okHttpClient;
            } catch (Exception e) {
                  throw new RuntimeException(e);
            }
      }

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

            if (args.length != 8) {
                  System.err
                              .println("Usage: java AWSMysqlRDSCMRestApi userid
password keyname numberofrecords batchsize mode operation CMip  " );
                  System.exit(-1);
            }

            String username = args[0];
            String password = args[1];
            String keyName = args[2];
            int numberofrecords = Integer.parseInt(args[3]);
            int batchsize = Integer.parseInt(args[4]);
            String mode = args[5];
            String operation = args[6];
            String CMip = args[7];

            AWSMysqlRDSCMRestApi awsresrest = new AWSMysqlRDSCMRestApi();
            awsresrest.key = keyName;
            awsresrest.CMip = CMip;
            awsresrest.token = awsresrest.getToken(CMip, username, password);

            Calendar calendar = Calendar.getInstance();

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

            Connection connection = ConnectionObject.getConnection();

            if (mode.equalsIgnoreCase("fpe")) {
                  if (operation.equalsIgnoreCase("both")) {
                        fpeencrypt(awsresrest, connection, mode, numberofrecords,
batchsize);
                        fpedecryptdata(awsresrest, connection, mode);
                  } else
```

```
                        fpeencrypt(awsresrest, connection, mode, numberofrecords,
batchsize);
                } else {
                        if (operation.equalsIgnoreCase("both")) {
                                enrypt(awsresrest, connection, mode, numberofrecords,
batchsize);
                                decryptdata(awsresrest, connection, mode);
                        } else
                                enrypt(awsresrest, connection, mode, numberofrecords,
batchsize);
                }

                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);
        }

        static void fpedecryptdata(AWSMysqlRDSCMRestApi awsresrest, Connection
connection, String action)
                        throws Exception {

                Statement stmt = null;
                try {
                        stmt = connection.createStatement();
                        String results;

                        String sql = "SELECT PersonID, LastName, FirstName, Address, City
FROM Persons";
                        ResultSet rs = stmt.executeQuery(sql);
                        String firstpart = "--data-binary '";
                        String thirdpart = "' --compressed";

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

                                int id = rs.getInt("PersonID");
                                String last = rs.getString("LastName");
                                String first = rs.getString("FirstName");
                                String addr = rs.getString("Address");
                                String city = rs.getString("City");
                                System.out.print(", last: " + last);
                                // System.out.println("data: " + results);
                                System.out.print("ID: " + id);
                                String[] parts = last.split(" ");
                                String sensitive = parts[1];
                                sensitive = sensitive.replaceAll("\'", "");
                                String text = firstpart + sensitive + thirdpart;
                                results = awsresrest.postfpe("https://" + awsresrest.CMip
+ "/api/v1/crypto/unhide2?keyName="
                                                + awsresrest.key + "&hint=digit", text);
```

```
                        parts = results.split(" ");
                        sensitive = parts[1];
                        sensitive = sensitive.replaceAll("\'", "");
                        System.out.println("Original Data " + sensitive);
                        System.out.print(", First: " + first);
                        System.out.println(", addr: " + addr);
                }
                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!");

}

static void decryptdata(AWSMysqlRDSCMRestApi awsresrest, Connection
connection, String action) throws Exception {

        Statement stmt = null;
        try {
                stmt = connection.createStatement();
                String results;

                String sql = "SELECT PersonID, LastName, FirstName, Address, City
FROM Persons";
                ResultSet rs = stmt.executeQuery(sql);

                //String key =
"1498bd7dab2045a0ad245aa2c37c913106472a736d994d3992e0f1306bbee229";

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

                        int id = rs.getInt("PersonID");
                        String last = rs.getString("LastName");
                        String first = rs.getString("FirstName");
                        String addr = rs.getString("Address");
                        System.out.print(", last: " + last);
                        String tag = rs.getString("City");
```

```
                        String decryptjson = ciphertexttag + quote + last + quote
+ comma + tagtag + quote + tag + quote
                                        + comma+ modetag + quote + mode + quote +
comma
                                        + typetag + quote + type + quote + comma +
                                        idtag + quote + awsresrest.key + quote +
comma + ivtag + quote
                                        + iv + quote + comma + aadtag + quote + aad +
quote + endbracket;
                        results = awsresrest.post("https://" + awsresrest.CMip +
"/api/v1/crypto/decrypt", decryptjson);
                        //System.out.println("value " + results);

                        String plaintextbase64 = JsonPath.read(results.toString(),
"$.plaintext").toString();

                        byte[] decryoriginaldata =
Base64.getDecoder().decode(plaintextbase64);
                        results = new String(decryoriginaldata);
                        System.out.println("data: " + results);
                        System.out.print(", First: " + first);
                        System.out.println(", addr: " + addr);
                }
                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!");

    }


    static void fpeencrypt(AWSMysqlRDSCMRestApi awsresrest, Connection connection,
String action, int nbrofrecords,
                int batchqty) throws Exception {

        String SQL = "insert into Persons values (?,?,?,?,?)";
```

```
            int batchSize = batchqty;
            // int batchSize = 50;
            int count = 0;
            int[] result;
            int size = nbrofrecords;
            connection.setAutoCommit(false);
            PreparedStatement pstmt = connection.prepareStatement(SQL);
            String results = null;
            String sensitive = null;

            for (int i = 1; i <= size; i++) {

                  sensitive = randomNumeric(15);

                  String firstpart = "--data-binary '";
                  String thirdpart = "' --compressed";
                  String text = firstpart + sensitive + thirdpart;
                  results = awsresrest.postfpe(
                              "https://" + awsresrest.CMip +
"/api/v1/crypto/hide2?keyName=" + awsresrest.key + "&hint=digit",
                              text);

                  //System.out.println("value " + results);
                  String ciphertext = JsonPath.read(results.toString(),
"$.data").toString();

                  pstmt.setInt(1, i);
                  pstmt.setString(2, ciphertext);
                  pstmt.setString(3, "FirstName");
                  pstmt.setString(4, sensitive + " Addr");
                  pstmt.setString(5, action);
                  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();

      }

      static void enrypt(AWSMysqlRDSCMRestApi awsresrest, Connection connection,
String action, int nbrofrecords,
                  int batchqty) throws Exception {

            String SQL = "insert into Persons values (?,?,?,?,?)";
            int batchSize = batchqty;
```

```
		int count = 0;
		int[] result;
		int size = nbrofrecords;
		connection.setAutoCommit(false);
		PreparedStatement pstmt = connection.prepareStatement(SQL);
		String results = null;
		String sensitive = null;

		String plaintextbase64 = null;

		for (int i = 1; i <= size; i++) {

			sensitive = randomNumeric(15);
			byte[] dataBytes = sensitive.getBytes();
			plaintextbase64 = Base64.getEncoder().encodeToString(dataBytes);

			String encryptjson = plaintexttag + quote + plaintextbase64 +
quote + comma + tagtag + quote + tag + quote
					+ comma + modetag + quote + mode + quote + comma +
idtag + quote + awsresrest.key + quote + comma
					+ ivtag + quote + iv + quote + comma + aadtag +
quote + aad + quote + endbracket;
			//System.out.println("encryptjson json " + encryptjson);

			results = awsresrest.post("https://" + awsresrest.CMip +
"/api/v1/crypto/encrypt", encryptjson);

			//System.out.println("value " + results);
			String ciphertext = JsonPath.read(results.toString(),
"$.ciphertext").toString();
			String tagtext = JsonPath.read(results.toString(),
"$.tag").toString();
			pstmt.setInt(1, i);
			pstmt.setString(2, ciphertext);
			pstmt.setString(3, "FirstName");
			pstmt.setString(4, sensitive + " Addr");
			pstmt.setString(5, tagtext);
			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();

	}
```

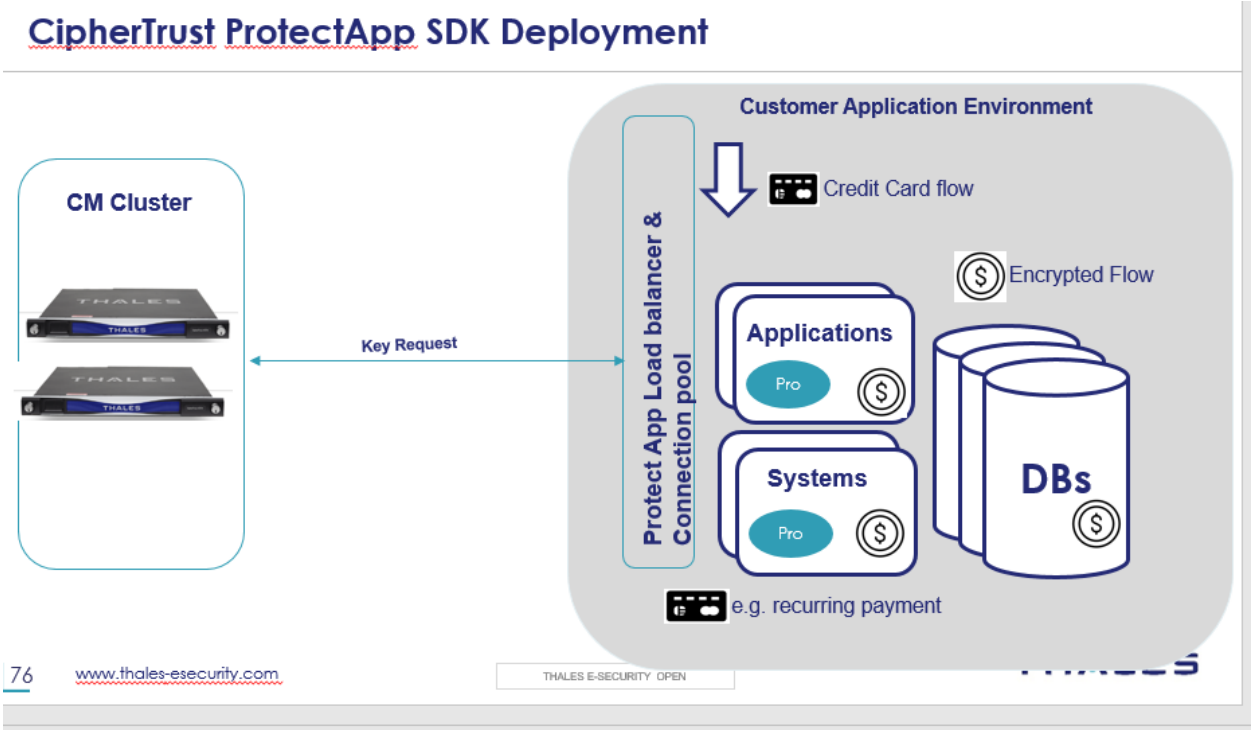
```
	// private static final String ALPHA_NUMERIC_STRING =
	// "ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789";
	private static final String ALPHA_NUMERIC_STRING =
"ABCDEFGHIJKLMNOPQRSTUVWXYZ";

	public static String randomAlphaNumeric(int count) {
		StringBuilder builder = new StringBuilder();
		while (count-- != 0) {
			int character = (int) (Math.random() *
ALPHA_NUMERIC_STRING.length());
			builder.append(ALPHA_NUMERIC_STRING.charAt(character));
		}
		return builder.toString();
	}

	private static final String NUMERIC_STRING = "0123456789";

	public static String randomNumeric(int count) {
		StringBuilder builder = new StringBuilder();
		while (count-- != 0) {
			int character = (int) (Math.random() * NUMERIC_STRING.length());
			builder.append(NUMERIC_STRING.charAt(character));
		}
		return builder.toString();
	}

}
```

CipherTrust Manager ProtectApp JCE SDK Example.

This example uses the CM ProtectApp JCE which is a Java Cryptography Extension provider that enables you to integrate your Java applications with the cryptographic and key-management abilities of the Thales CM. All applications, servlets, or scripts see a conventional JCE interface and issue simple Java–based (JCE) commands to the CM to perform cryptographic operations. CM ProtectApp JCE enables your Java client to perform cryptographic operations either by requesting that operations be performed on the Thales CM (remote mode) or by caching keys on the client and performing crypto locally (local mode).

Architecture

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. As you can see below when using the ProtectApp SDK it includes its own load balancing and connection pooling. The diagram below shows how the local mode of operation would work with the keys cached on the host. A request for a key would only be done if the time threshold has been exceeded.



The example uses the CipherTrust Manager ProtectAPP API to encrypt the data. To use the ProtectApp JCE SDK it is necessary to:

1. Install the ProtectAPP SDK
2. Set the necessary property file settings as indicated in the documentation and
3. Copy the appropriate jar files to you project directory

At this point, your application can start to make encryption calls. There are two different modes or algorithms that were used, GCM and Format Preserved Encryption (FPE). The GCM example used the AES/GCM/NoPadding cipher and the FPE example used the FPE/FF1/CARD10 cipher.

Here is the code to test using standard MySQL JDBC class file and CipherTrust Manager ProtectApp JCE SDK. As an example when passing parameters, it can create 1000 rows of data and a commit every 100 records.

```
public class AWSMysqlRDSProtectAppExample3 {

	public static void main(String[] args) throws Exception {
		// Valid options are encrypt or tokenize

		if (args.length != 7) {
			System.err.println("Usage: java AESGCMEncryptionDecryptionSample
user password keyname "
					+ "authTagLength iv aad data");
			System.exit(-1);
		}
		String username = args[0];
		String password = args[1];
		String keyName = args[2];
```

```
            int authTagLength = Integer.parseInt(args[3]);
            String iv = args[4];
            String aad = args[5];
            String data = args[6];

            byte[] ivBytes = IngrianProvider.hex2ByteArray(iv);
            byte[] aadBytes = IngrianProvider.hex2ByteArray(aad);
            byte[] dataBytes = data.getBytes();

            String action = "javagcm";
// String action = "baseline";

            Calendar calendar = Calendar.getInstance();

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

            Connection connection = ConnectionObject.getConnection();

            /*
             * if(!args[3].contains("null")) { tweakAlgo = args[3]; }
             */
            String algorithm = null;
            if (action.equalsIgnoreCase("javafpe")) {
                  System.out.println("iv: " +
IngrianProvider.byteArray2Hex(ivBytes));
                  System.out.println("AAD: " +
IngrianProvider.byteArray2Hex(aadBytes));
                  NAESession session = null;
                  session = NAESession.getSession(username, password.toCharArray(),
"hello".toCharArray());
                  NAEKey key = NAEKey.getSecretKey(keyName, session);
                  String tweakData = null;
                  String tweakAlgo = null;
                  algorithm = "FPE/FF1/CARD10";
                  FPEParameterAndFormatSpec param = new
FPEParameterAndFormatBuilder(tweakData).set_tweakAlgorithm(tweakAlgo)
                              .build();
                  fpeencrypt(connection, action, 1000, 100, key, param);
                  //fpedecryptdata( connection,  action,  key,  param);
            } else if (action.equalsIgnoreCase("javagcm")) {
                  System.out.println("iv: " +
IngrianProvider.byteArray2Hex(ivBytes));
                  System.out.println("AAD: " +
IngrianProvider.byteArray2Hex(aadBytes));
                  NAESession session = null;
                  session = NAESession.getSession(username, password.toCharArray(),
"hello".toCharArray());
                  NAEKey key = NAEKey.getSecretKey(keyName, session);
                  // algorithm = "AES/GCM/NoPadding";
                  GCMParameterSpec spec = new GCMParameterSpec(authTagLength,
ivBytes, aadBytes);
                  enrypt(connection, action, 1000, 100, key, spec);
                  //enrypt(connection, action, 1000, 100, key, spec);
            } else {

                  baseline(connection, action, 1000, 100);
```

```
		}



		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);
	}

	static void fpedecryptdata(Connection connection, String action, NAEKey key,
FPEParameterAndFormatSpec param)
			throws Exception {

		Statement stmt = null;
		try {
			stmt = connection.createStatement();
			String results;
			String	algorithm = "FPE/FF1/CARD10";
			String sql = "SELECT PersonID, LastName, FirstName, Address FROM
Persons";
			ResultSet rs = stmt.executeQuery(sql);
			Cipher decryptCipher = Cipher.getInstance(algorithm,
"IngrianProvider");
			// to decrypt data, initialize cipher to decrypt
			decryptCipher.init(Cipher.DECRYPT_MODE, key, param);


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

				int id = rs.getInt("PersonID");
				String last = rs.getString("LastName");
				String first = rs.getString("FirstName");
				String addr = rs.getString("Address");
				System.out.print(", last: " + last);

				byte[] decrypt = decryptCipher.doFinal(last.getBytes());
				results = new String(decrypt);

				// System.out.println("data: " + results);
				System.out.print("ID: " + id);

				System.out.print(", last decrypted: " + results);
				System.out.print(", First: " + first);
				System.out.println(", addr: " + addr);
			}
			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!");

	}

	static void decryptdata(Connection connection, String action, NAEKey key,
GCMParameterSpec param)
			throws Exception {

		Statement stmt = null;
		try {
			stmt = connection.createStatement();
			String results;

			String sql = "SELECT PersonID, LastName, FirstName, Address FROM
Persons";
			ResultSet rs = stmt.executeQuery(sql);
			// STEP 5: Extract data from result set
			// Display values
			Cipher decryptCipher =
NAECipher.getNAECipherInstance("AES/GCM/NoPadding", "IngrianProvider");
			// to decrypt data, initialize cipher to decrypt
			decryptCipher.init(Cipher.DECRYPT_MODE, key, param);


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

				int id = rs.getInt("PersonID");
				String last = rs.getString("LastName");
				String first = rs.getString("FirstName");
				String addr = rs.getString("Address");
				System.out.print(", last: " + last);

				byte[] decrypt =
decryptCipher.doFinal(IngrianProvider.hex2ByteArray(last));
				results = new String(decrypt);

				// System.out.println("data: " + results);
				System.out.print("ID: " + id);
```

```
                        System.out.print(", last decrypted: " + results);
                        System.out.print(", First: " + first);
                        System.out.println(", addr: " + addr);
                }
                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!");

    }

    static void fpeencrypt(Connection connection, String action, int nbrofrecords,
int batchqty, NAEKey key,
                FPEParameterAndFormatSpec param) throws Exception {

        String SQL = "insert into Persons values (?,?,?,?,?)";
        int batchSize = batchqty;
        int count = 0;
        int[] result;
        int size = nbrofrecords;
        connection.setAutoCommit(false);
        PreparedStatement pstmt = connection.prepareStatement(SQL);
        String results = null;
        String sensitive = null;

        Cipher encryptCipher = NAECipher.getNAECipherInstance("FPE/FF1/CARD10",
"IngrianProvider");
        encryptCipher.init(Cipher.ENCRYPT_MODE, key, param);

        for (int i = 1; i <= size; i++) {

                sensitive = randomNumeric(15);

                byte[] outbuf = encryptCipher.doFinal(sensitive.getBytes());

                results = new String(outbuf);
                pstmt.setInt(1, i);
                pstmt.setString(2, results);
```

```
                        pstmt.setString(3, "FirstName");
                        pstmt.setString(4, sensitive + " Addr");
                        pstmt.setString(5, action);
                        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();

        }

        static void enrypt(Connection connection, String action, int nbrofrecords, int
batchqty, NAEKey key,
                        GCMParameterSpec spec) throws Exception {

                String SQL = "insert into Persons values (?,?,?,?,?)";

                int batchSize = batchqty;
                int count = 0;
                int[] result;
                int size = nbrofrecords;
                connection.setAutoCommit(false);
                PreparedStatement pstmt = connection.prepareStatement(SQL);
                String results = null;
                String sensitive = null;

                Cipher encryptCipher =
NAECipher.getNAECipherInstance("AES/GCM/NoPadding", "IngrianProvider");
                encryptCipher.init(Cipher.ENCRYPT_MODE, key, spec);

                byte[] encrypt = null;

                for (int i = 1; i <= size; i++) {

                        sensitive = randomAlphaNumeric(15);
                        byte[] dataBytes = sensitive.getBytes();
                        encrypt = encryptCipher.doFinal(dataBytes);
                        results = IngrianProvider.byteArray2Hex(encrypt);
                        // System.out.println("Encrypt: " + results);

                        pstmt.setInt(1, i);
                        pstmt.setString(2, results);
                        pstmt.setString(3, "FirstName");
                        pstmt.setString(4, sensitive + " Addr");
                        pstmt.setString(5, action);
```

```
                        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();

        }
        static void baseline(Connection connection, String action, int nbrofrecords,
int batchqty) throws Exception {

                String SQL = "insert into Persons values (?,?,?,?,?)";

                int batchSize = batchqty;
                int count = 0;
                int[] result;
                int size = nbrofrecords;
                connection.setAutoCommit(false);
                PreparedStatement pstmt = connection.prepareStatement(SQL);
                String results = null;
                String sensitive = null;



                for (int i = 1; i <= size; i++) {

                        sensitive = randomAlphaNumeric(15);
                        byte[] dataBytes = sensitive.getBytes();

                         results = "baseline";
                        pstmt.setInt(1, i);
                        pstmt.setString(2, results);
                        pstmt.setString(3, "FirstName");
                        pstmt.setString(4, sensitive + " Addr");
                        pstmt.setString(5, action);
                        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();

		}

		// private static final String ALPHA_NUMERIC_STRING =
		// "ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789";
		private static final String ALPHA_NUMERIC_STRING =
"ABCDEFGHIJKLMNOPQRSTUVWXYZ";

		public static String randomAlphaNumeric(int count) {
			StringBuilder builder = new StringBuilder();
			while (count-- != 0) {
				int character = (int) (Math.random() *
ALPHA_NUMERIC_STRING.length());
				builder.append(ALPHA_NUMERIC_STRING.charAt(character));
			}
			return builder.toString();
		}

		private static final String NUMERIC_STRING = "0123456789";

		public static String randomNumeric(int count) {
			StringBuilder builder = new StringBuilder();
			while (count-- != 0) {
				int character = (int) (Math.random() * NUMERIC_STRING.length());
				builder.append(NUMERIC_STRING.charAt(character));
			}
			return builder.toString();
		}


}
```

Appendix

Performance Testing

Minimal resources were used for the AWS/RDS MySql testing (db.t2.micro, 1GB RAM, 1vCPU & GP SSD). The goal was to capture relative time differences comparing REST to ProtectApp. A local on premise CM was deployed with 8GB of RAM and 2 Cores running in a VM and the machine running the sample app was on another local VM.

In summary, the REST calls took about 25% to 37% longer than the baseline.

Non cached ProtectApp calls took up to 10% longer than the baseline.

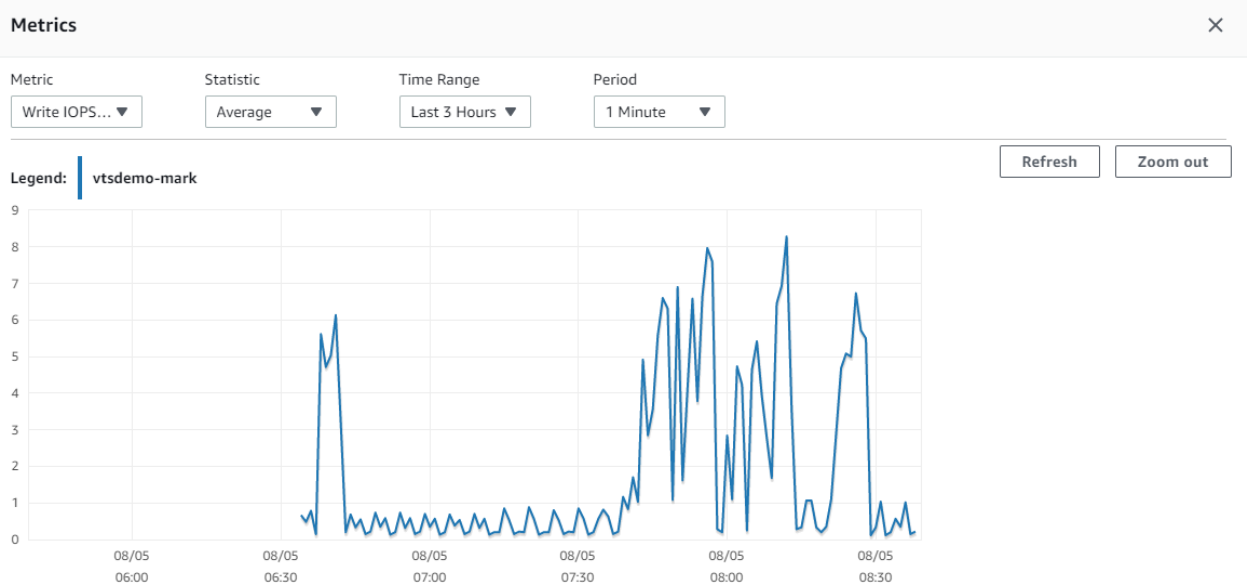
There were no differences in performance for cached ProtectApp calls compared to the baseline.

Each test consisted of encrypting a single column of 15 bytes and inserting 5 columns of data into a table. Each test inserted 1000 rows with a batch size of 100.

Here is the DDL for the table used in the example:

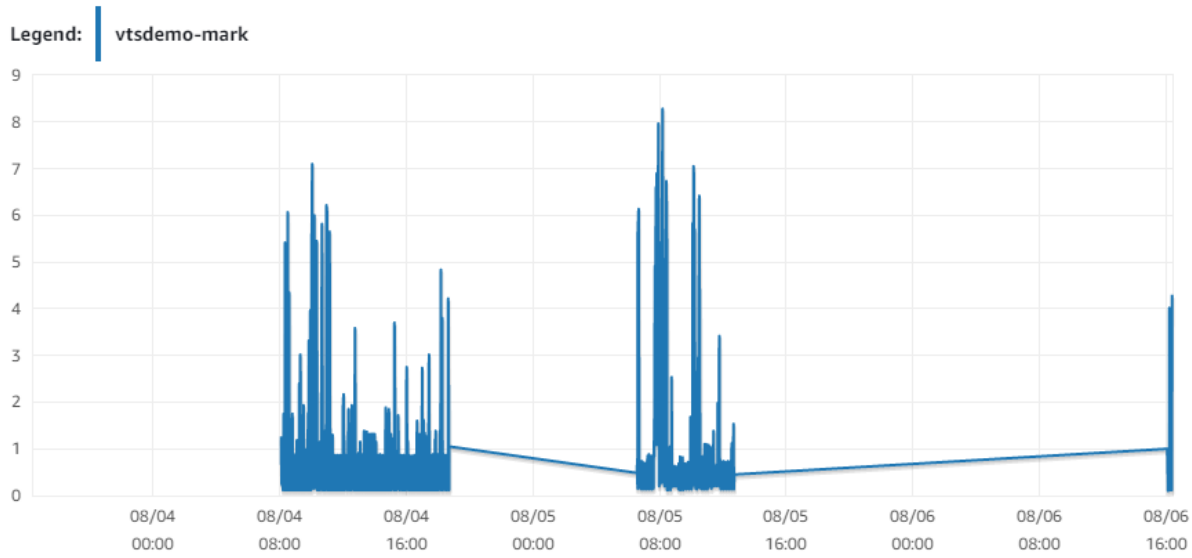
```
CREATE TABLE Persons (
    PersonID int,
    LastName varchar(255),
    FirstName varchar(255),
    Address varchar(255),
    City varchar(255)
);
```

A lot of the time variation was due to RDS IOPS differences. Here is an example of the IOPS during the testing:



Sets of 5 were run for both FPE and GCM algorithms for both REST and ProtectApp with the high and low of each set dropped and then an average was used for the results. No significant differences were observed between GCM and FPE. If a local database were used with more control over the environment then more noticeable differences between the various tests may be expected.

Here is an example of the variation over a few days.



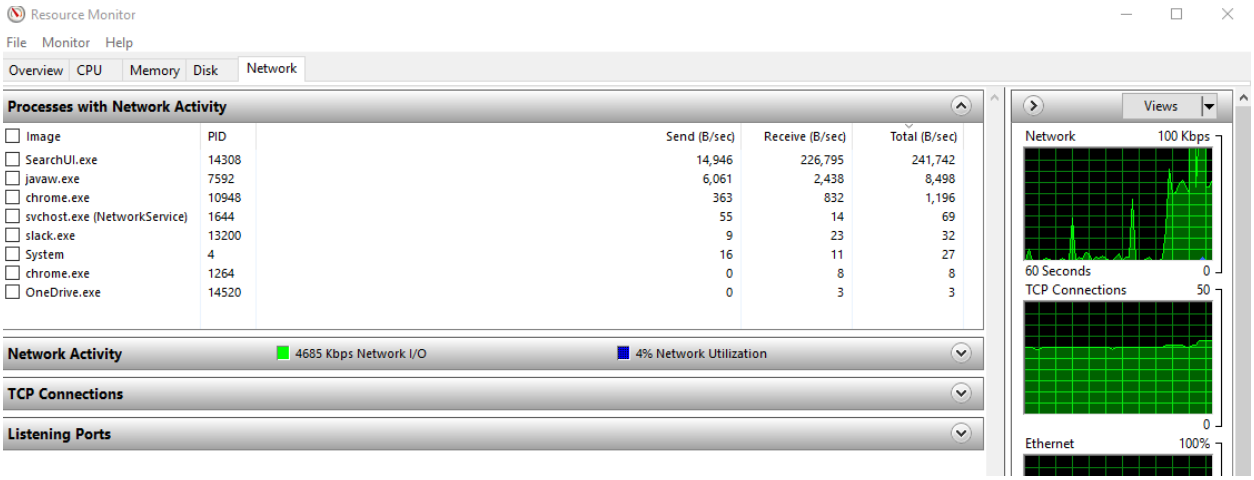
Note: Testing results were only compared to each other on single days to limit variation in IOPS encountered over different days as indicated above.

Cached vs Non-Cached Network Traffic.

All tests results provided in this document were conducted using cached keys. These screenshots are only provided to show how using cached mode does provide less traffic and processing is done on the client.

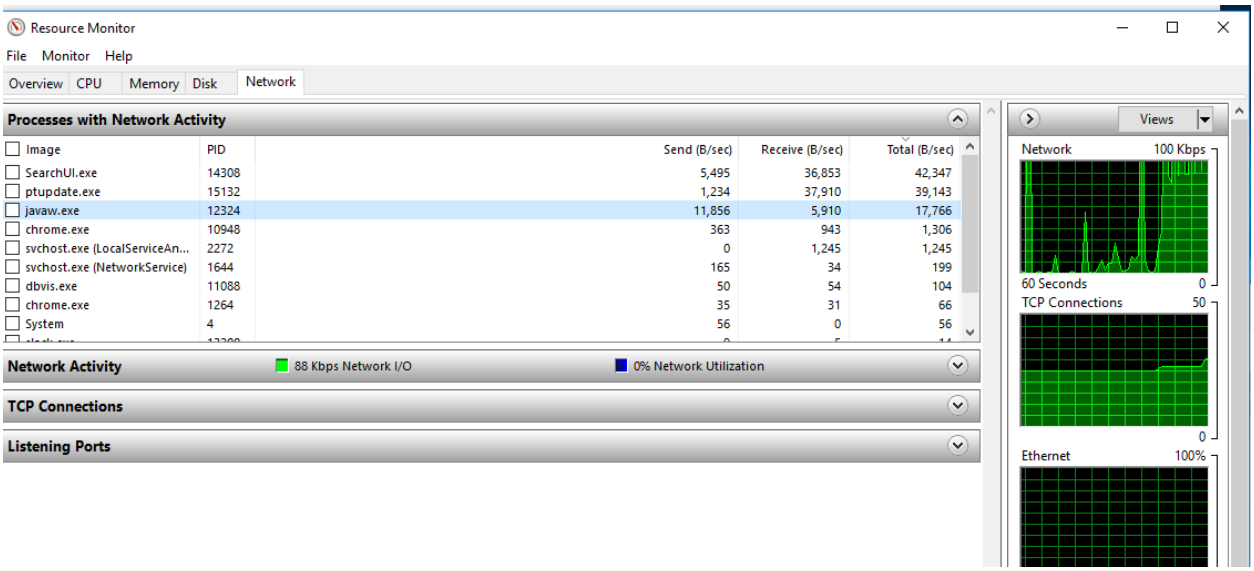
Cached Example.

Looking at the javaw.exe process below you can see that it averaged out around 6000 send bytes.



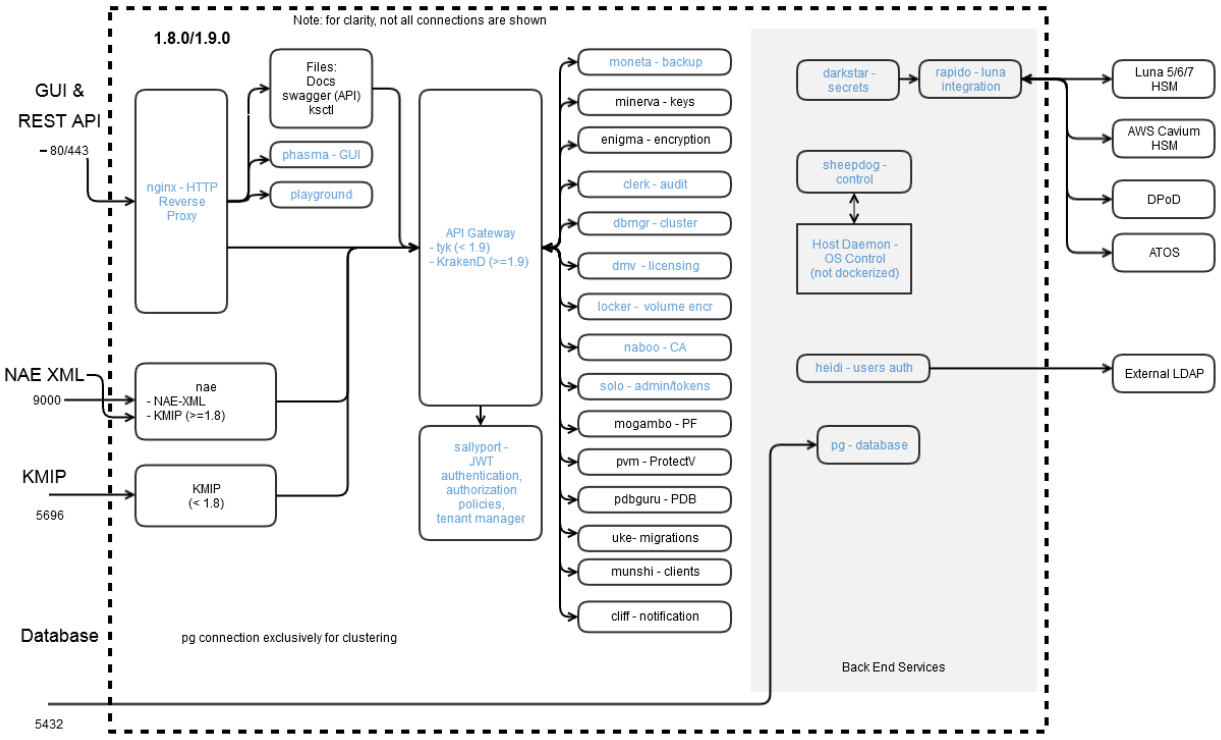
Non Cached Example.

Looking at the javaw.exe process below you can see that it averaged out around 11,800 send bytes.



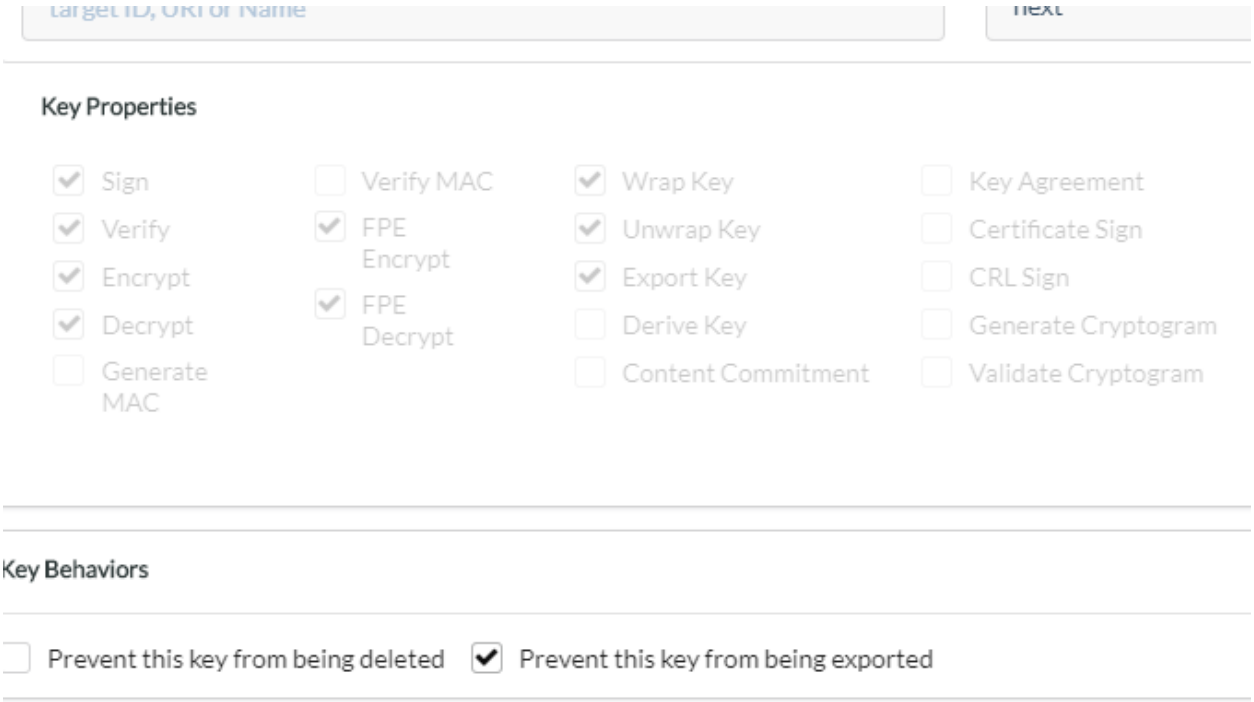
CipherTrust Manager Detailed Architecture.

The diagram below shows CipherTrust Manager is based on micro services using Docker containers.



Enabling local mode for ProtectAPP

There are a few requirements to enable local mode processing for ProtectApp. Having the appropriate key settings and IngrianNAE.properties file settings.



When running in local mode it is necessary to have the allow Export Key and **Uncheck** the "***Prevent this key from being exported"*** or you will encounter this error.

```
Exception in thread "main" java.security.InvalidKeyException:
java.security.InvalidKeyException: 1440: Key is not exportable
	at com.ingrian.security.nae.CipherValidator.a(CipherValidator.java:227)
	at com.ingrian.security.nae.CipherValidator.a(CipherValidator.java:188)
	at
com.ingrian.security.nae.AdvAbstractNAECipher.engineInit(AdvAbstractNAECipher.java:777)
	at javax.crypto.Cipher.init(Cipher.java:1394)
	at javax.crypto.Cipher.init(Cipher.java:1327)
	at
AWSMysqlRDSProtectAppExample3.enrypt(AWSMysqlRDSProtectAppExample3.java:300)
	at AWSMysqlRDSProtectAppExample3.main(AWSMysqlRDSProtectAppExample3.java:88)
```

It is also necessary to have the following settings in your IngrianNAE.properties file.

Valid values: yes, no, tcp_ok
# Default: no
# Recommended: no
Symmetric_Key_Cache_Enabled=tcp_ok
Asymmetric_Key_Cache_Enabled=tcp_ok

```
# [Client Key Caching]
# [Symmetric_Key_Cache_Expiry]
# Time period since key was cached after which a symmetric key
# may be removed from cache. Symmetric_Key_Cache_Expiry can be specified
# in any time units (default - seconds)
# Setting this value to 0 is equivalent to an infinite timeout.
# Note: This field is also applicable to Asymmetric key cache expiry
# Default: 43200 (12 hours)

Symmetric_Key_Cache_Expiry=43200
```