CipherTrust Manager Platform

Azure BYOK & BYOE tips and techniques.

Document Version 1.1

Contents

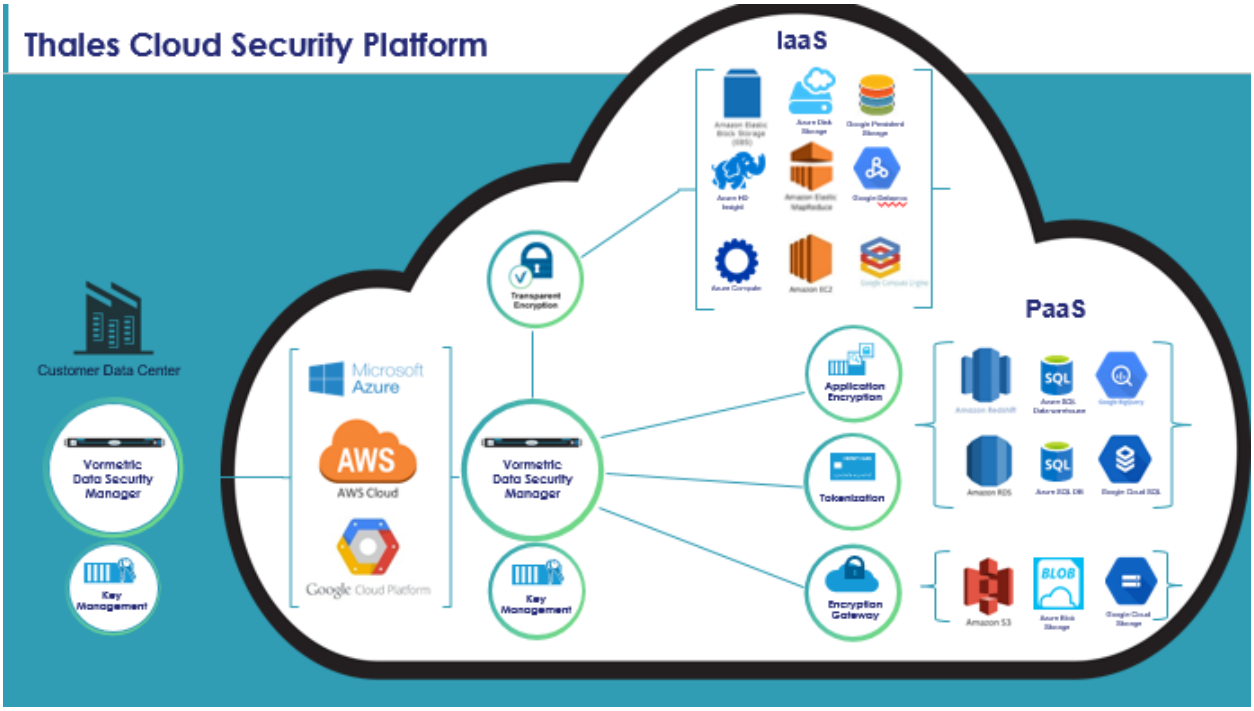
PREFACE

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

Azure BYOE & BYOK tips & techniques, provides examples on how to implement both bringing your own encryption (BYOE) and bringing you own key (BYOK) to the Azure Cloud Provider. Installation steps for the various products are NOT covered in this document since they are covered in the installation guides already provided by Thales.

Listed below is a diagram showing the different kinds of cloud offerings by the various cloud providers (IAAS,PAAS,SAAS). BYOK is available for any cloud service that allows for external key import. To find out what services are available please check with each cloud provider. For Azure check the following link: https://docs.microsoft.com/en-us/azure/security/azure-security-encryption-atrest

Thales BYOE is available for IAAS, PAAS and cloud storage. This document will **not** cover the IAAS for BYOE, since the Thales solution using CipherTrust Transparent Encryption (CTE), is essentially the same as if the machine were running on premise.



The BYOE examples provided in this document will demonstrate how to implement BYOE for the PAAS Cloud using the CipherTrust Manager (CM) REST API and the BYOE for Azure cloud storage will utilize the CipherTrust Transparent Encryption agent.

DOCUMENTATION VERSION HISTORY

Product/Document Version	Date	Changes
V1.0	4/2021	Initial document release. MWarner.

ASSUMPTIONS

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

- CipherTrust Manager (Key Manager)
- Tokenization
- Key management
- Data encryption
- Familiarity with REST
- Microsoft Azure

GUIDE TO Thales 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

There are many reasons why customers are looking at both BYOK and BYOE as it relates to the cloud. For example, one independent organization called the Cloud Security Alliance has outlined a best practice for Encryption and Key Management:

> Platform and data-appropriate encryption…shall be required and Encryption Keys:
> Shall not be stored in the cloud but
> Shall be maintained by the cloud consumer or trusted key management provider.

Keep in mind when implementing BYOK the cloud provider will be implementing the encryption at the disk level which is comparable to full disk encryption. With BYOE, you have complete control over who can access what data and when with your own policies. You also have control over the frequency of the key rotations and audit logs as well. See the following link for more information on BYOE.
https://cpl.thalesgroup.com/encryption/bring-your-own-encryption

BYOK Use Case

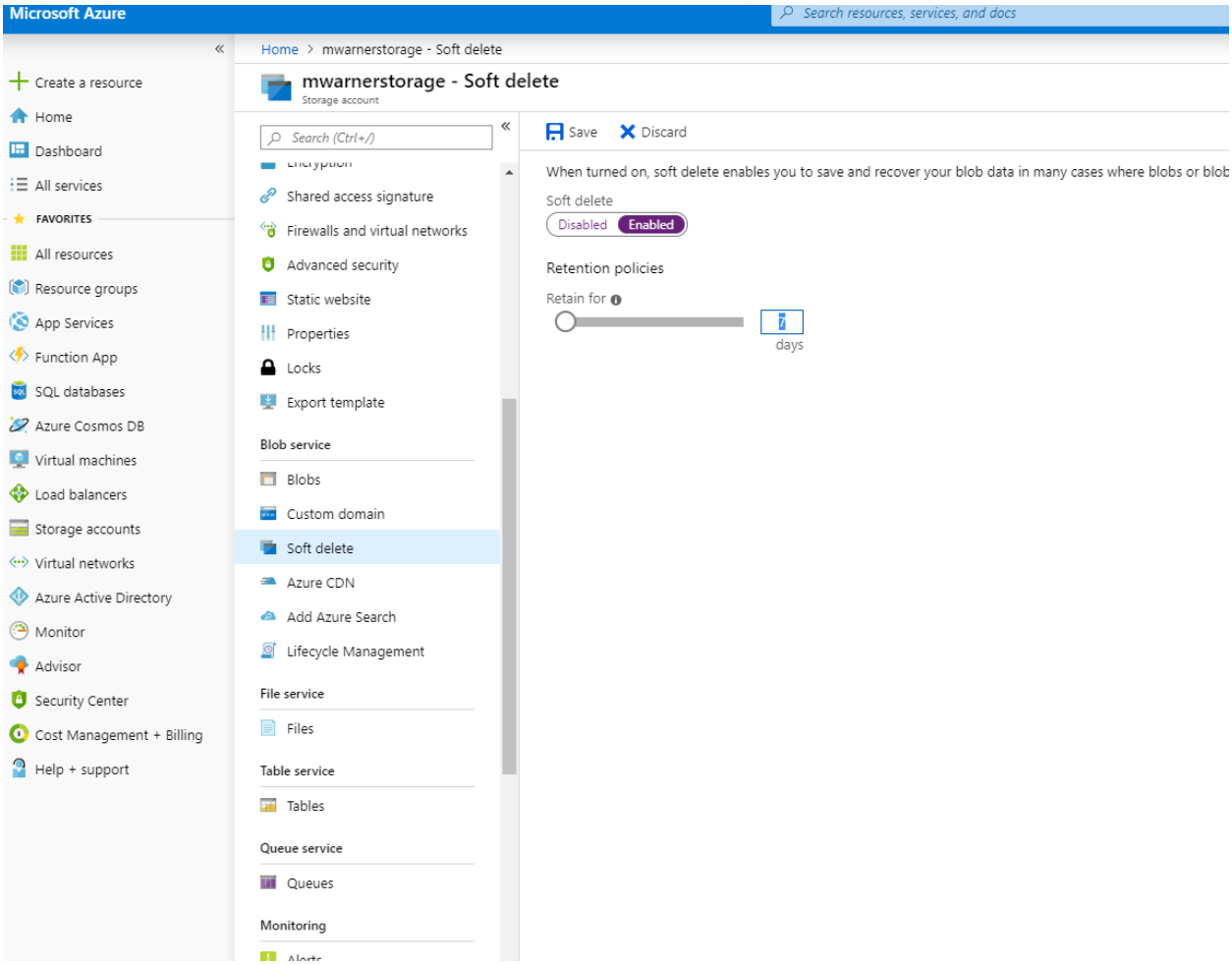
BYOK for Azure uses the Thales Ciphertrust Cloud Key Manager (CCKM) product. CCKM can be implemented both on premise or in the cloud. See link for brief video.
https://cpl.thalesgroup.com/resources/encryption/ciphertrust-cloud-key-manager-introduction-video

The examples provided in this document leverage the "***Server-side encryption using customer-managed keys in Azure Key"*** use case explained in the following link.

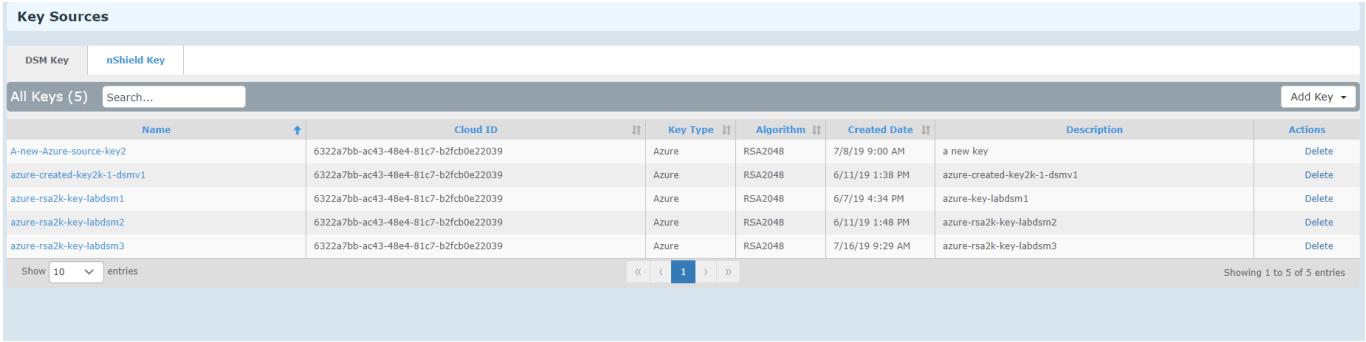
https://docs.microsoft.com/en-us/azure/security/azure-security-encryption-atrest

BYOK Example

This example will show how to use the Thales CipherTrust Cloud Key Manager (CCKM) product to provide your own encryption keys to protect various Azure services. In order to use BYOK in Azure it is necessary to have an Azure key vault with soft delete option enabled. Please see the appendix for more details on how to set this up. It is important to update the storage account to be enabled for soft delete as well. See screenshot below.

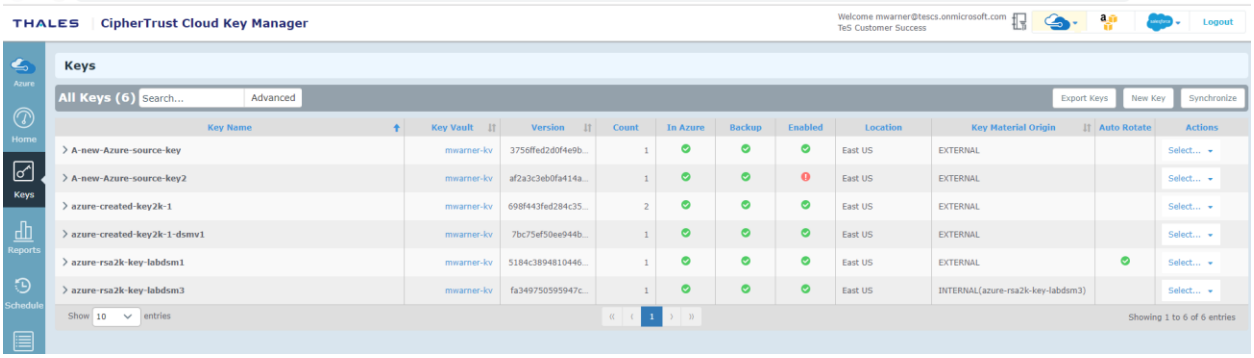


Step 1. Create a key in the CipherTrust Manager with CCKM using the Key Sources icon “Add Key” button. At this point the key is on the Key Manager.

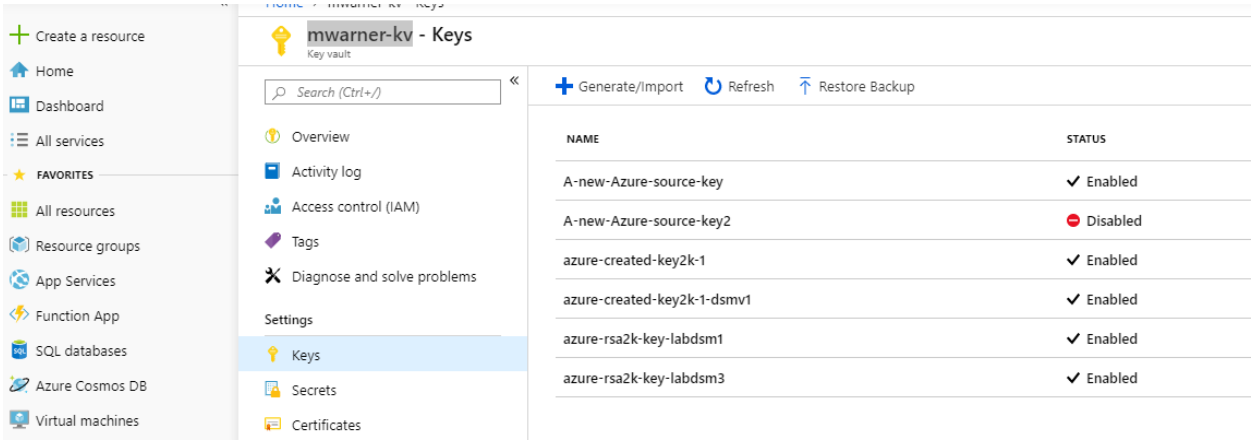


For this example we created a key called (**azure-rsa2k-key-labdsm3**)

Step 2. Upload the key from the Thales CM it to Azure key vault by selecting the “Keys” icon. Here is an example screenshot:



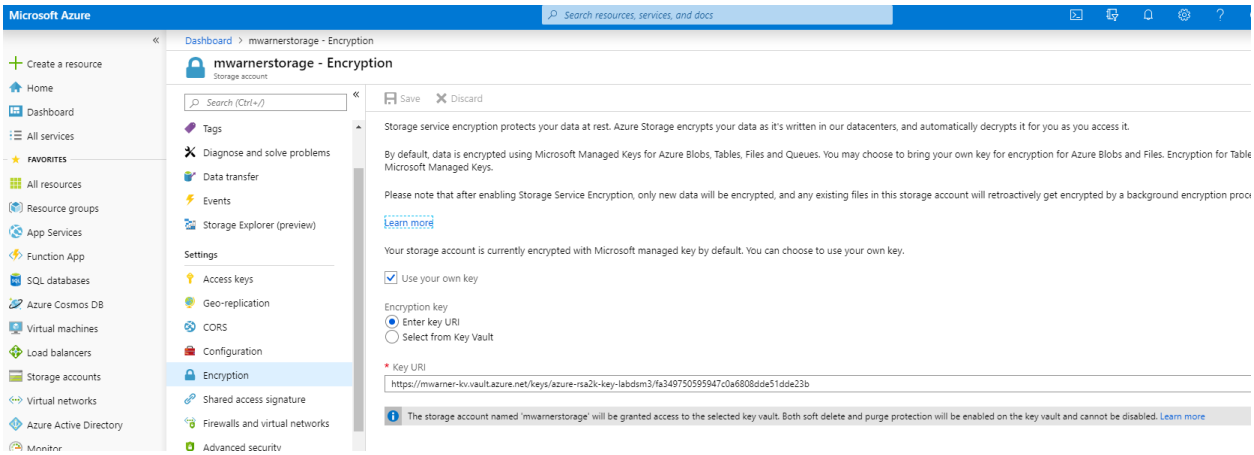
The key now shows up in the Azure key vault you picked. In this particular example, the keyvault is (mwarner-kv) as show below in the Azure portal.



Step 3. In the Azure portal pick any storage account in Azure you want to encrypt and select the “Use your own key” button.

Step 4. Pick the vault and the key you want to use.

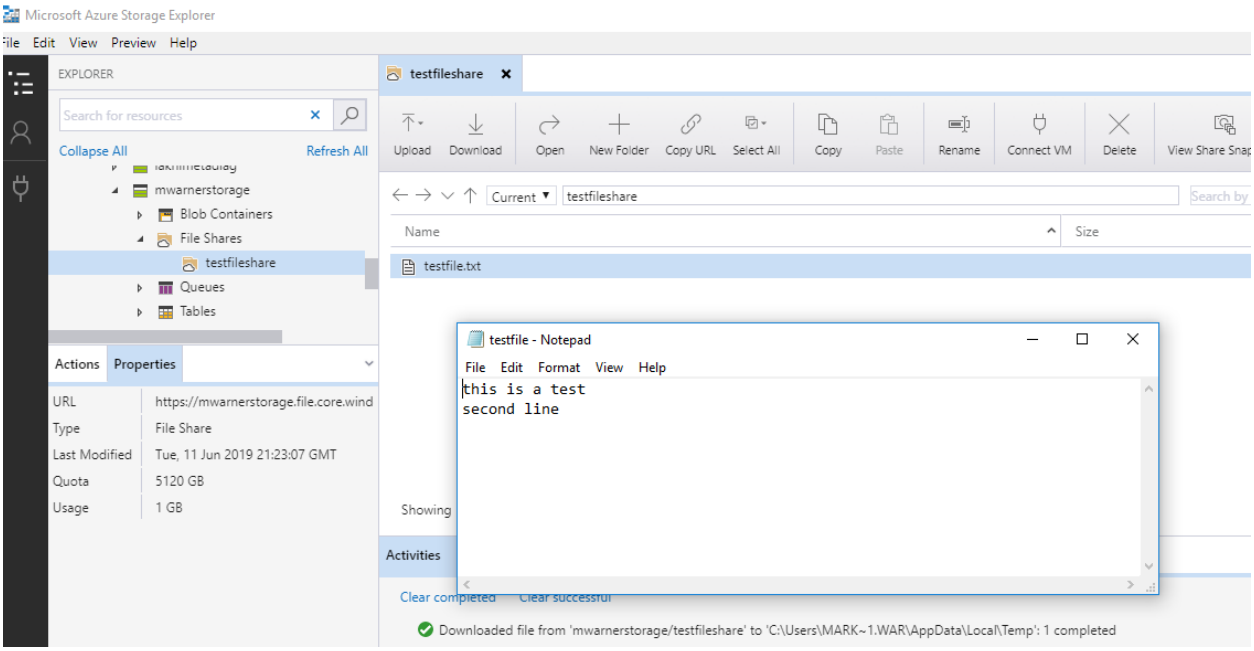
Step 5. Pick the “**azure-rsa2k-key-labdsm3**” that was first created in the CM and then uploaded to the Azure vault.



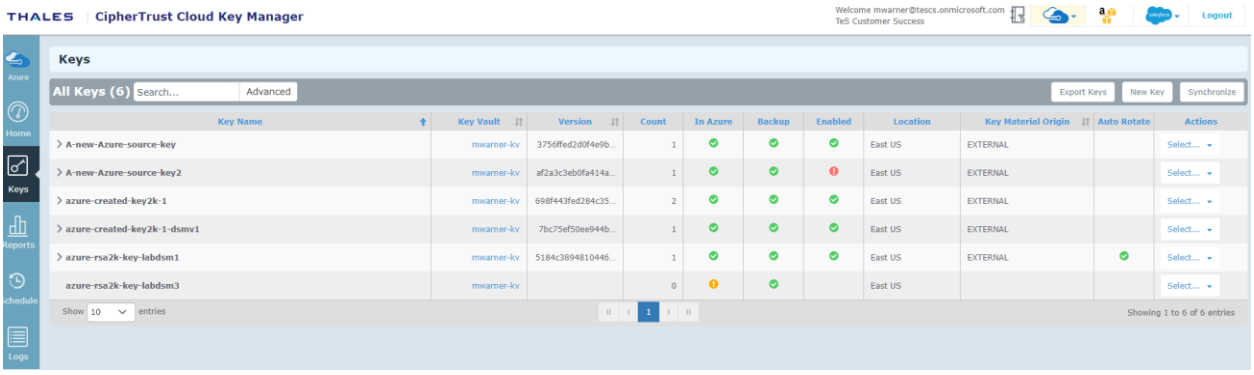
Any Azure service that you decide to protect with the **azure-rsa2k-key-labdsm3** in the Azure Vault will now wrap any Azure DEK key that is created during this process.

Test BYOK using file share.

Now access the file you crated in the above. In my example, I used the Azure storage explorer.



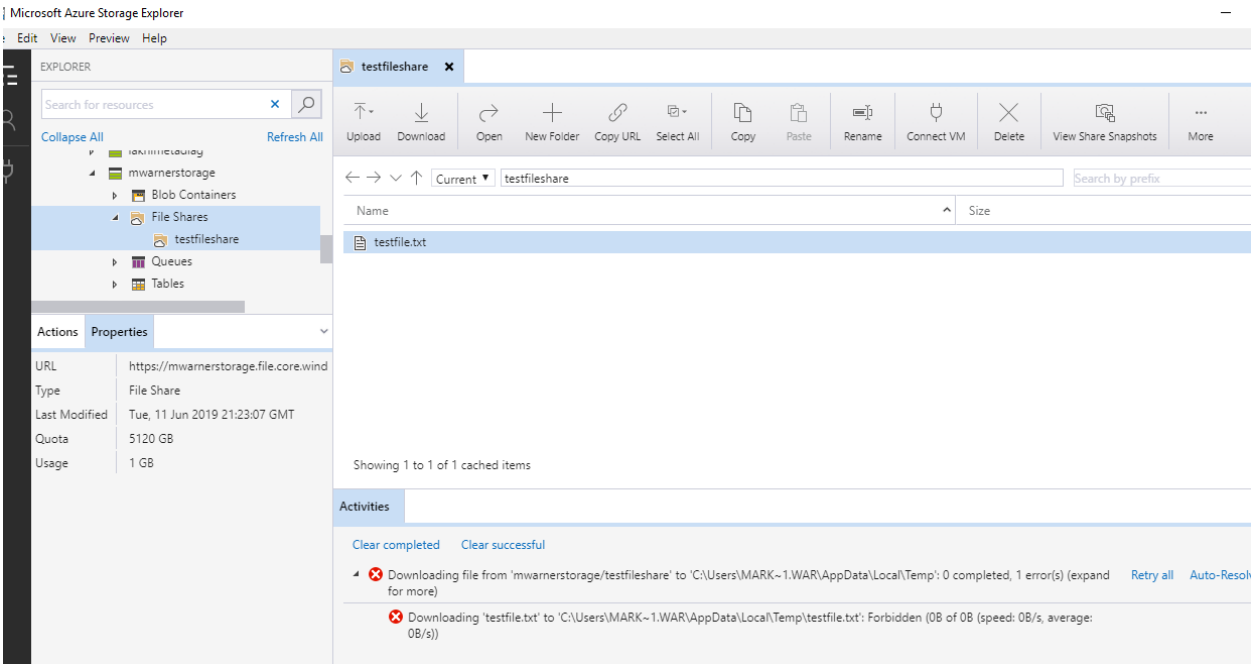
Now soft delete the key in CCKM.



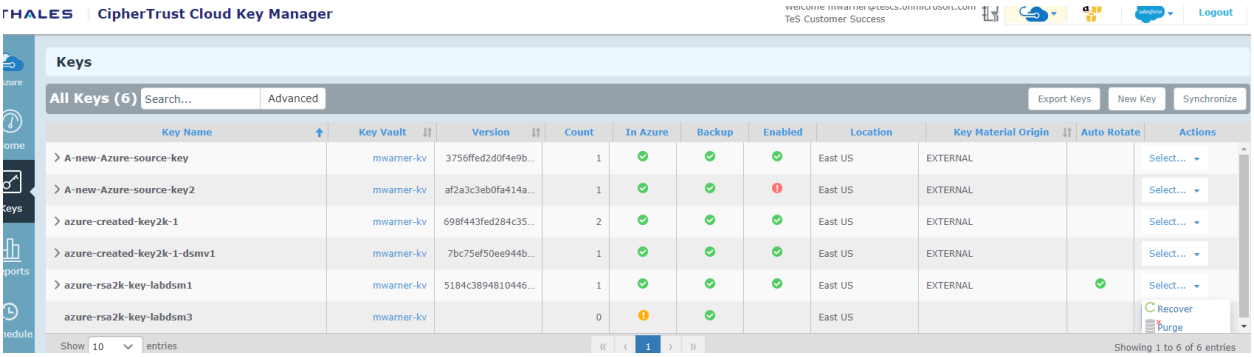
If you look in the Azure portal in the azure vault the key should not be listed.

Now try to access the key with the Azure storage explorer.

Should get the message below trying to access the file in Azure storage explorer.



Now recover the key in CCKM. (Notice the recover option lower right)



Should be able to access the file again now that the key has been recovered.

BYOK Benefits Summary

Listed below is a brief summary of the advantages of using a customer managed key (CMK).

	Azure	VORMETRIC CCKM
Key Ownership	Microsoft	Owned by Customer
Automatic Key Backup with Customer	NO	YES
Automatic Key Rotation	NO	YES
Key Management reporting	SIEM necessary	YES
Multi subscription Key Visibility	NO	YES
All Keys Backups in FIPS L3	NO	YES

BYOE Use Case

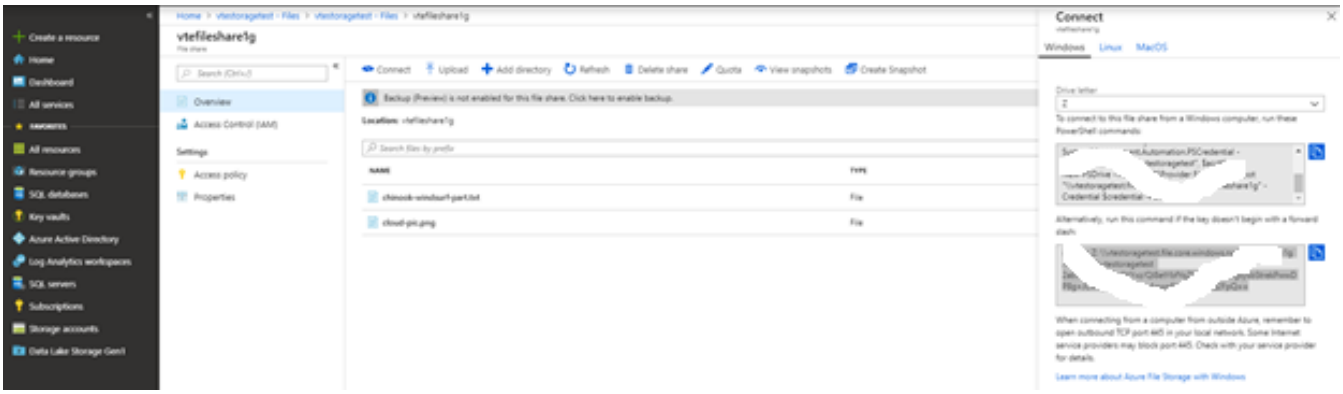
As mentioned above Thales provides BYOE solutions for IAAS, PAAS and cloud storage. This section will cover PAAS and cloud storage scenarios since the IAAS implementation is really the same as the Thales on premise solution, which utilizes the CTE agent.

BYOE Example with Azure File Share

Implementation steps.

Step 1. Create an Azure storage account.

Step 2. Created a LRS/GRS storage file share within the new storage account. Listed below is an example of a file share with two files in it.



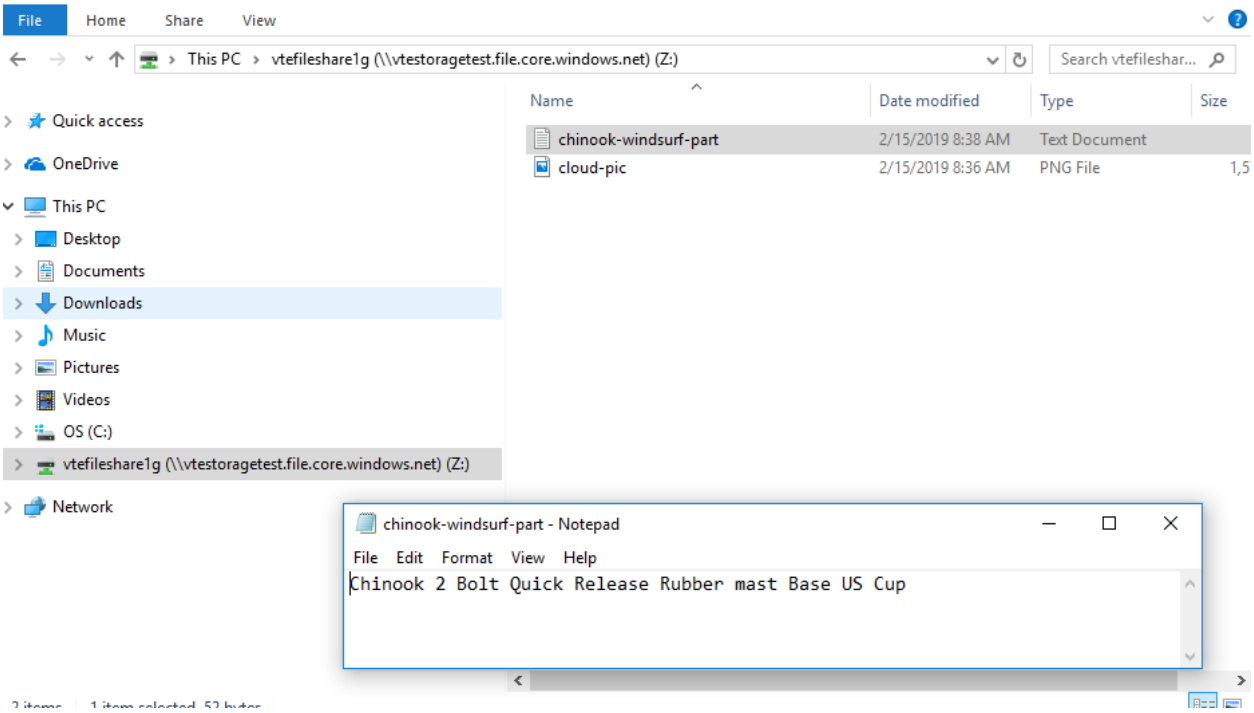
Step 3. Click the connect button to obtain the powershell mount command. It will generate the powershell scripts needed to mount the azure file share. Listed below is an example.

```
$acctKey = ConvertTo-SecureString -String "yoursuperlongstringwmAnwnFUKZZWvvQ2FpQ==" -
AsPlainText -Force
$credential = New-Object System.Management.Automation.PSCredential -ArgumentList
"Azure\vtestoragetest", $acctKey
New-PSDrive -Name Z -PSProvider FileSystem -Root
"\\vtestoragetest.file.core.windows.net\vtefileshare1g" -Credential $credential –Persist
```

After issuing the above commands you should now have a new mount point which can be guarded by Thales Transparent Encryption Agent.

```
Name           Used (GB)     Free (GB) Provider      Root                                        CurrentLocation
----           ---------     --------- --------      ----                                        ---------------
Z                   0.00          1.00 FileSystem    \\vtestoragetest.file.core.windo...

PS C:\Users\mark.warner>
```

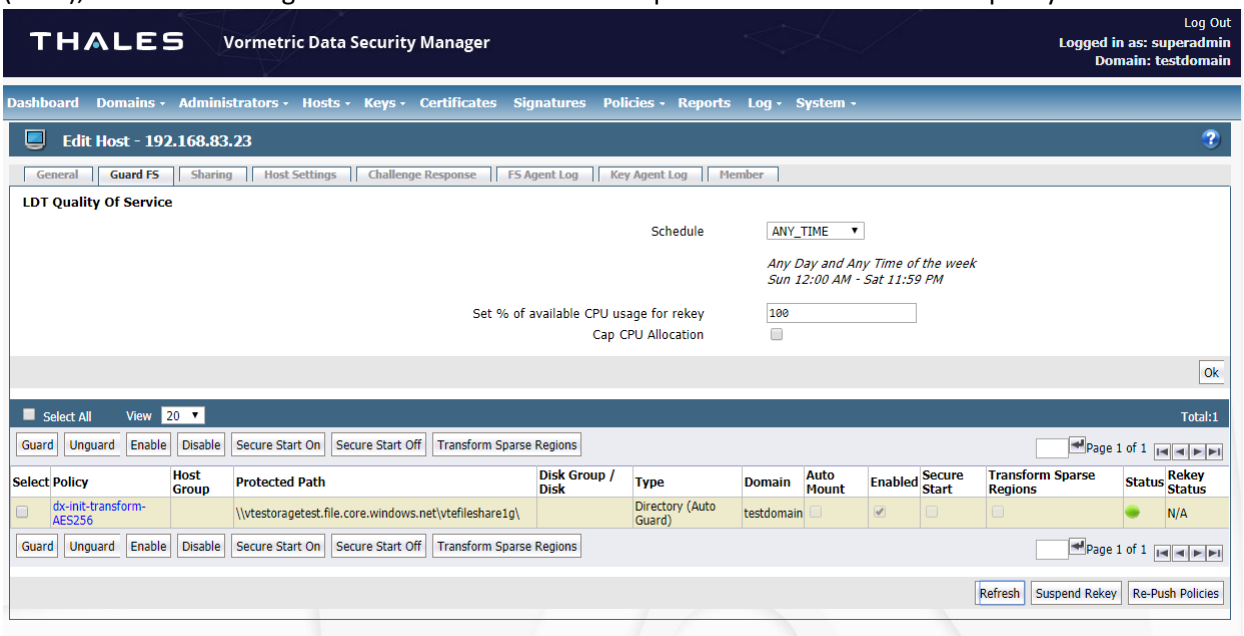
Since the data has not been encrypted, you can see clear text as the screen below shows.



Step 4. Install the Thales CipherTrust Transparent Encryption Agent on the windows machine you are interested in mounting the SMB file share from Azure. Then create a standard Thales dataxform policy to encrypt the data.

(Note: As of Feb 2019, In order to use an Azure file share outside of the Azure region it is hosted in, such as on-premises or in a different Azure region, the OS must support SMB 3.0. Please see following link for more details: https://docs.microsoft.com/en-us/azure/storage/files/storage-how-to-use-files-windows

The screen below is showing the Thales Guard Point in the legacy Vormetric Data Security Manager (DSM), with a status of green for the Azure SMB share protected with a dataxform policy.



The CipherTrust Manager UI would essentially be the same setup and has a new more modern UI then the Vormetric DSM.

Should see messages in logs.
2981256 2019-02-15 09:49:48.56I 192.168.83.23 CGA3193I: [SecFS, 0] PID[4] Successfully guarded [\\vtestoragetest.file.core.windows.net\vtefileshare1g]
2981255 2019-02-15 09:49:48.516 I 192.168.83.23 CGA3001I: [SecFS, 0] PID[4] EVENT: Path (sa=43,lock=1,type=1,dir=\\vtestoragetest.file.core.windows.net\vtefileshare1g) successfully guarded.

Note: Logs in the CipherTrust Manager would be in json format.

Step 5. Issue the following command on the above mount point to encrypt the data with the Thales batch utility.

C:\>dataxform --rekey --gp \\vtestoragetest.file.core.windows.net\vtefileshare1g \ --preserve_modified_time

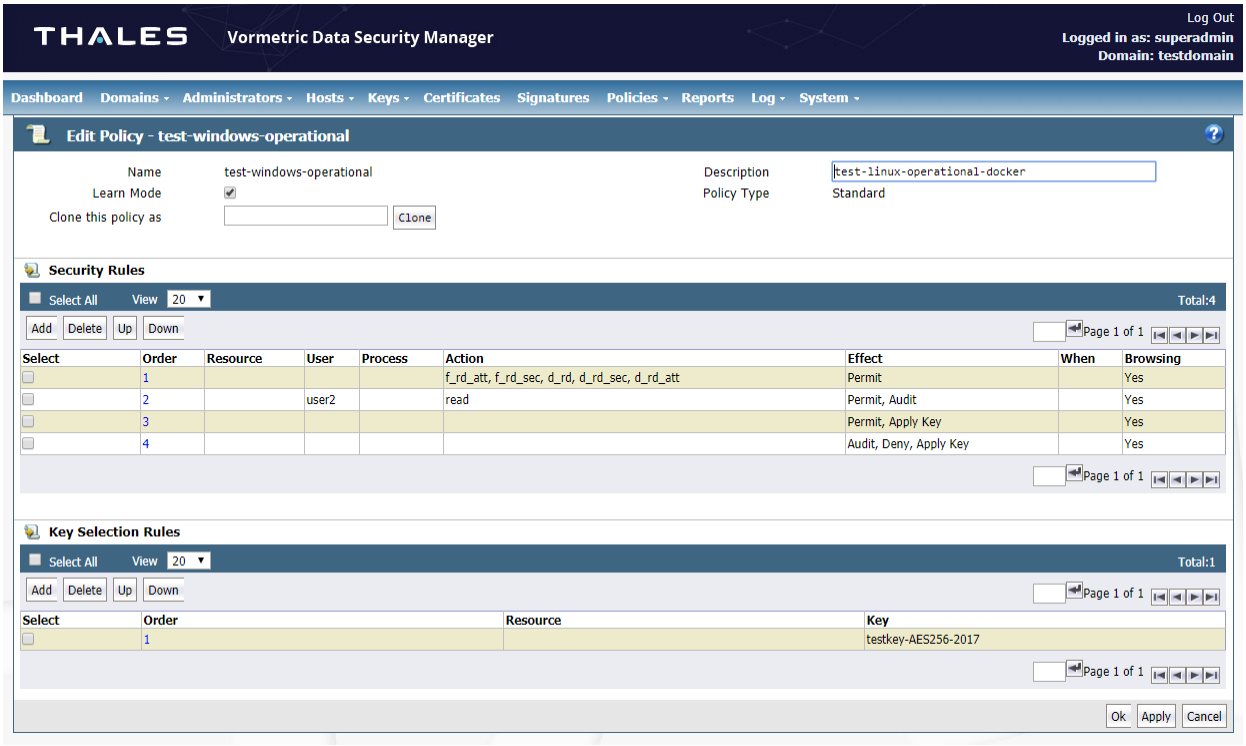
```
Checking if \\vtestoragetest.file.core.windows.net\vtefileshare1g\ is a guard
  point with a rekey policy applied
\\vtestoragetest.file.core.windows.net\vtefileshare1g\ is a guard point with a
  rekey policy applied
About to perform the requested data transform operation
-- Be sure to back up your data
-- Please do not attempt to terminate the application
If Shadow Copy was used on your system, you must back up your data before
  attempting to run dataxform. Once dataXform has been completed, you may
  restart Shadow Copy.  Note, however, that all  Shadow Copy backups made
  prior to running dataxform will be unusable and should be discarded.
  Attempting to restore your cleartext Shadow Copy backups made prior to
  running dataxform into your encrypted data will result in data corruption.
Do you wish to continue (y/n)?y
Scan found 3 files (1 MB) in 1 directories for guard point
  \\vtestoragetest.file.core.windows.net\vtefileshare1g\
Transformed 3 files (1 MB) of 3 files (1 MB) for guard point
  \\vtestoragetest.file.core.windows.net\vtefileshare1g\
The data transform operation took 0 hours, 0 minutes and 5 seconds
The data transform program ran from Fri Feb 15 09:41:34 2019 until Fri Feb 15
  09:41:39 2019
Data transform for guard point
  \\vtestoragetest.file.core.windows.net\vtefileshare1g\ finished
```

Step 6. Remove the dataxform info file by issuing the command below.

C:\>dataxform --gp \\vtestoragetest.file.core.windows.net\vtefileshare1g\ --clea nup

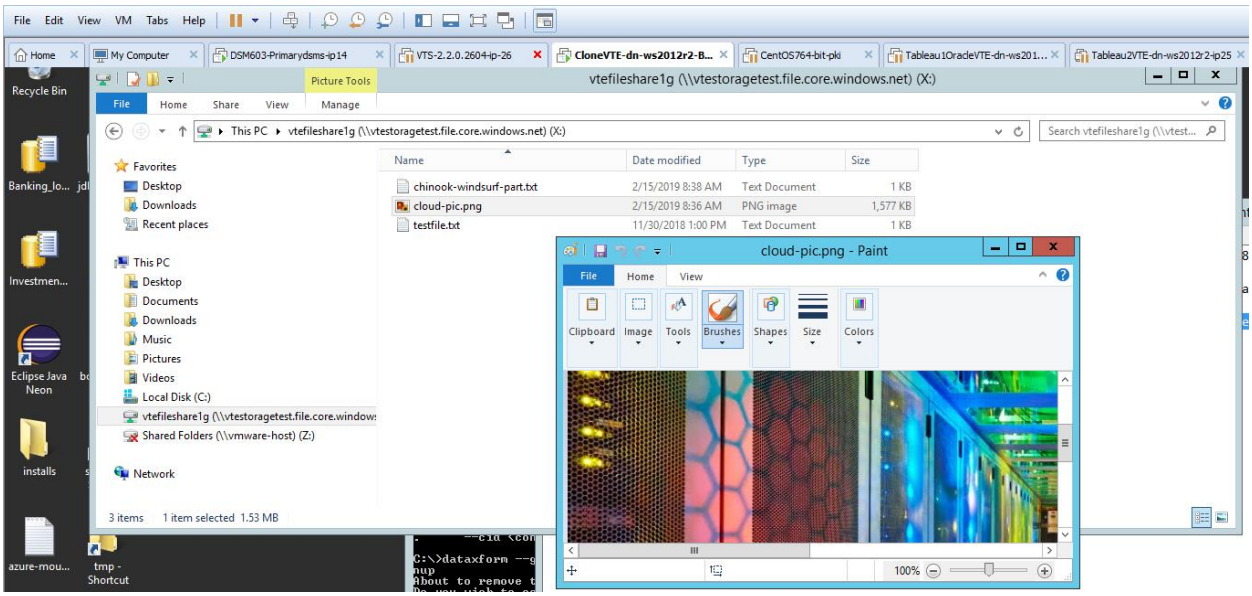
About to remove the data transformation status files
Do you wish to continue (y/n)?y
Removal of data transformation status files completed

Step 7. Now remove the Thales dataxform policy and apply the standard Thales online policy. The screenshot below is an example of an online policy created on the legacy Vormetric Data Security Manager but the UI for CipherTrust Manager would essentially contain the same information just displayed in a more modern UI.

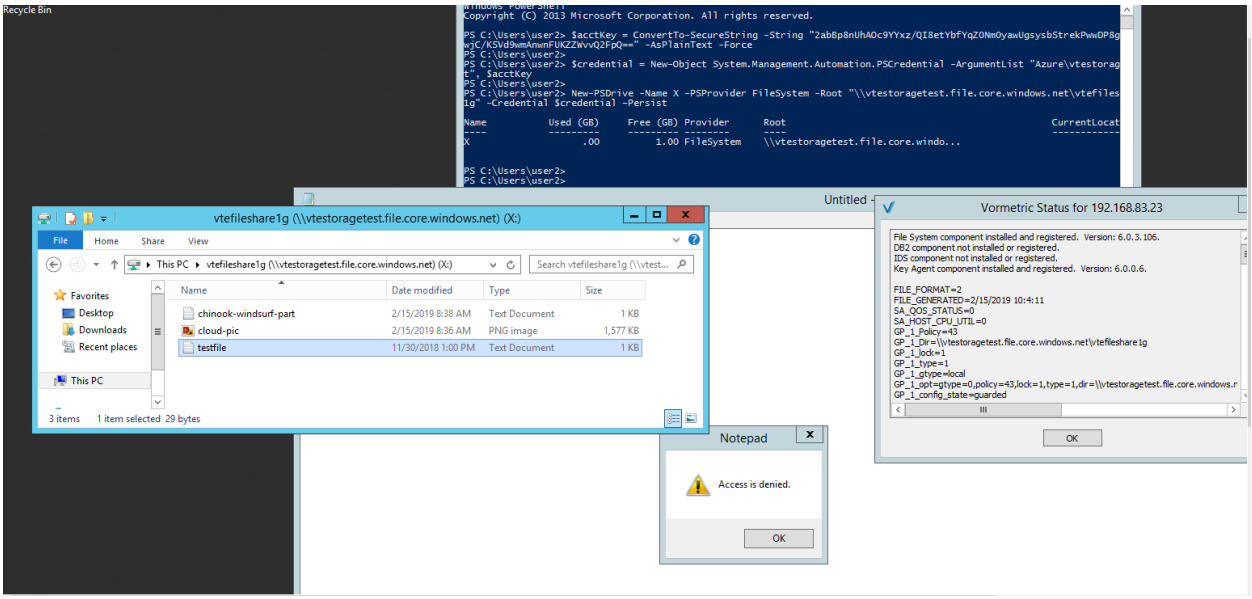


Testing.

Now try accessing the data with a valid user once the status of the above guard point is green. Sees unencrypted data.



Now try to access the data with a user on the same machine (user2) who should be blocked.

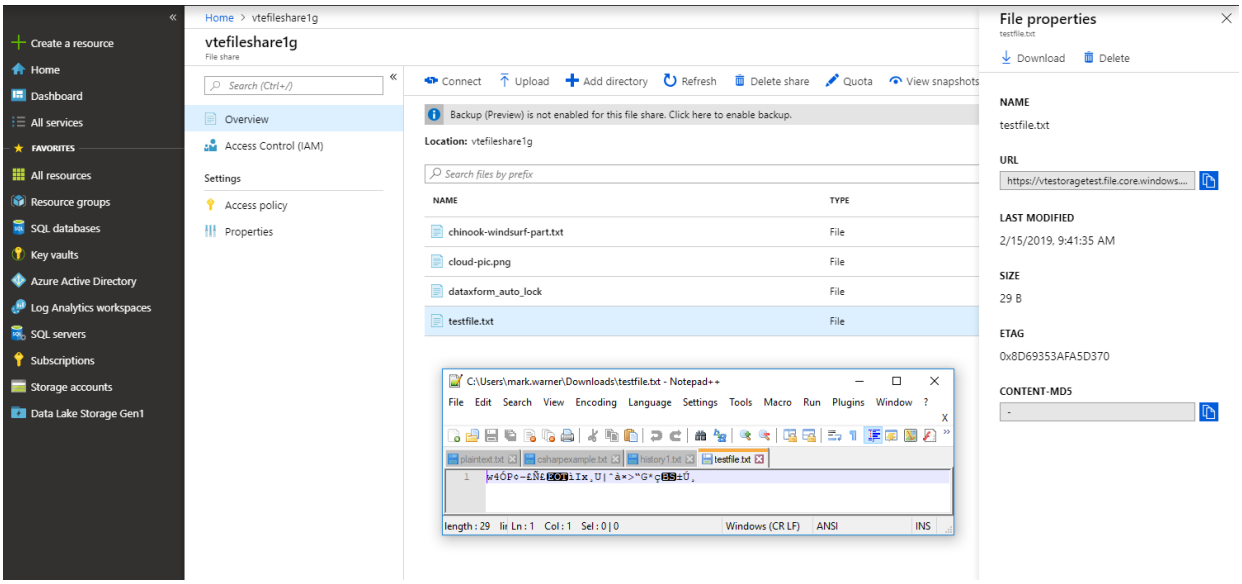


Get access is denied and logs generated. *Note: Logs in the CipherTrust Manager would be in json format.*

2981260 2019-02-15 10:04:56.446 W 192.168.83.23 CGA3002W: [SecFS, 0] PID[4064] EVENT: [PID:0000000000000FE0] ACCESS DENIED for file :\vtestoragetest.file.core.windows.net\vtefileshare1g\testfile.txt for operation MMap Read due to no 'Apply Key' rule in policy

2981259 2019-02-15 10:04:53.095 W 192.168.83.23 CGA3002W: [SecFS, 0] PID[2572] EVENT: [PID:0000000000000A0C] ACCESS DENIED for file :\vtestoragetest.file.core.windows.net\vtefileshare1g\cloud-pic.png for operation MMap Read due to no 'Apply Key' rule in policy

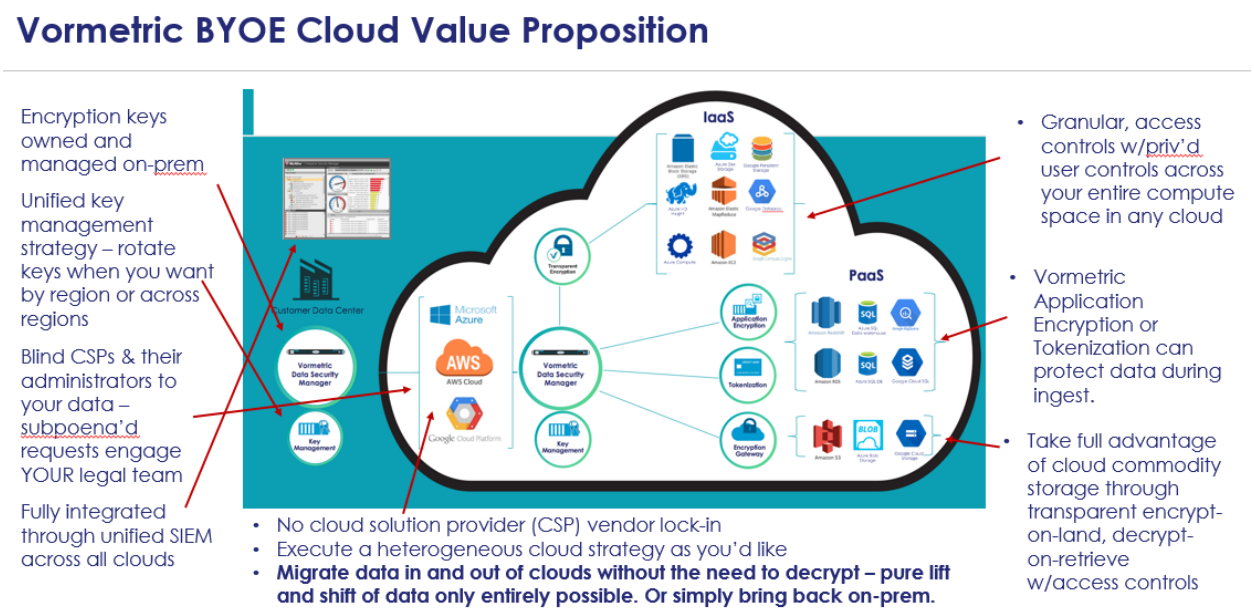
Now try to access file directly from the Azure portal or from a machine **without** the Thales agent results in encrypted data.



Only gets encrypted data.

BYOE Benefits Summary

Listed below is a brief summary of the benefits of using BYOE.



BYOE Example with SQL Azure & Cosmos

Implementation steps.

Azure.

Create an SQL Azure database instance and a sql database within the instance. Also create an SQL Cosmos db account.

Thales

Install the Thales CipherTrust Manager (CM) and the CipherTrust Cloud Key Manager (CCKM). Installation is supported for both on premise or in the Azure cloud. Search for Thales in the Azure Marketplace:

https://azuremarketplace.microsoft.com/en-us/marketplace/apps?search=Thales&page=1

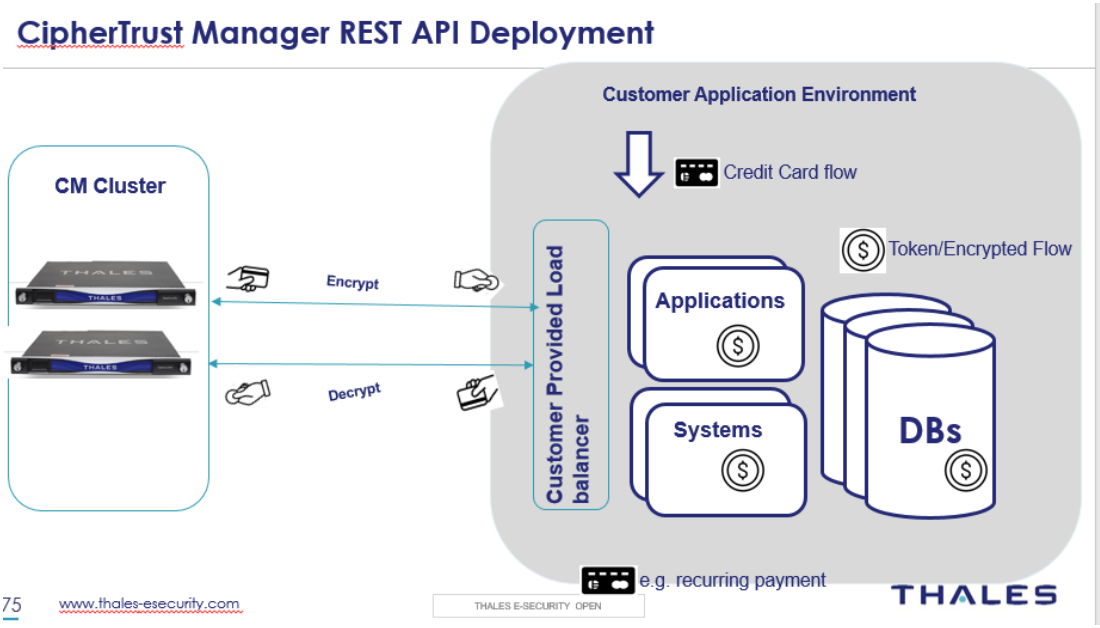
Thales CipherTrust Manager REST API can be used for many different use cases. Typically, it is used for scenarios when a company has sensitive data in a field of a particular file or a column in a database and they want to encrypt or tokenize the sensitive data. Use cases can include:

- 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
- https://thalesdocs.com/

Azure 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.

The Azure examples provided below are for both Cosmos and SQL Azure.

BYOE – Azure PAAS Example Applications

The first sample application used is for the use case when a customer is using AZURE Cosmos and the second example shows how to protect data when using AZURE SQL. Both tokenization and encryption have Format Preserve Encryption (FPE) mode of operation, which makes it much easier to implement since the database schemas will not have to change.

Azure Cosmos Example

This sample application creates a Cosmos table and then inserts data into it using the Thales CipherTrust Manager REST API to protect the lastname. 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.`

Azure Cosmos Application Modifications.

This example was based on existing Cosmos DB code found at: https://github.com/Azure-Samples/azure-cosmos-db-sql-api-async-java-getting-started

The `cmRESTProtect` method below are code changes that are required in order encrypt the sensitive data.

```
    public static Family getJohnsonFamilyDocument() throws Exception {
        Family JohnsonFamily = new Family();
        JohnsonFamily.setId("Johnson-" + System.currentTimeMillis());
        JohnsonFamily.ctmh = new CipherTrustManagerHelper();

        JohnsonFamily.ctmh.dataformat = "alphanumeric";
        JohnsonFamily.ctmh.username = "admin";
        JohnsonFamily.ctmh.password = "Vormetric123!";
        JohnsonFamily.ctmh.cmipaddress = "192.168.159.160";
                try {
                        String tkn = JohnsonFamily.ctmh.getToken();

                        JohnsonFamily.ctmh.key = "MyAESEncryptionKey26";

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


        String lastname = "Johnson";
                String enclastname = JohnsonFamily.ctmh.cmRESTProtect("fpe", lastname, "encrypt");

                JohnsonFamily.setLastName(enclastname);

        Parent parent1 = new Parent();
        parent1.setFirstName("John");

        Parent parent2 = new Parent();
        parent2.setFirstName("Lili");

        return JohnsonFamily;
    }
```

The `cmRESTProtect` method was implemented in the Main.java class to decrypt sensitive data.

```
private void executeSimpleQueryAsyncAndRegisterListenerForResult(CountDownLatch completionLatch) {

        this.ctmh = new CipherTrustManagerHelper();
        this.ctmh.dataformat = "alphanumeric";
        this.ctmh.username = "admin";
        this.ctmh.password = "Vormetric123!";
        this.ctmh.cmipaddress = "192.168.159.160";
                try {
                        String tkn = this.ctmh.getToken();

                        this.ctmh.key = "MyAESEncryptionKey26";

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

                // Set some common query options
                FeedOptions queryOptions = new FeedOptions();
                queryOptions.setMaxItemCount(10);
                queryOptions.setEnableCrossPartitionQuery(true);

                String collectionLink = String.format("/dbs/%s/colls/%s", databaseName, collectionName);
                Observable<FeedResponse<Document>> queryObservable = client.queryDocuments(collectionLink,
                                "SELECT * FROM Family WHERE Family.lastName != 'Andersen'",
queryOptions);

                queryObservable.observeOn(scheduler).subscribe(page -> {
```

```
                        heavyWork();

                        System.out.println("Got a page of query result with " + page.getResults().size()
+ " document(s)"
                                        + " and request charge of " + page.getRequestCharge());

                        List l = page.getResults().stream().map(d ->
d.get("lastName")).collect(Collectors.toList());
                        String results;


                        for (Iterator iterator = l.iterator(); iterator.hasNext();) {
                                Object object = (Object) iterator.next();

                                try {
                                        results = this.ctmh.cmRESTProtect("fpe", object.toString(),
"decrypt");
                                        System.out.println("Decrypted data = " + results);
                                } catch (Exception e1) {
                                        // TODO Auto-generated catch block
                                        e1.printStackTrace();
                                }

                                System.out.println("stored data in cosmos: " + object.toString());

                        }

                        System.out.println("Document Last Names "
                                        + page.getResults().stream().map(d ->
d.get("lastName")).collect(Collectors.toList()));

                        System.out.println(
                                        "Document Ids " + page.getResults().stream().map(d ->
d.getId()).collect(Collectors.toList()));
                },
                                // terminal error signal
                                e -> {
                                        e.printStackTrace();
                                        completionLatch.countDown();
                                },

                                // terminal completion signal
                                () -> {
                                        completionLatch.countDown();
                                });
        }
```

Sample output

```
Decrypted data = Andersen
stored data in cosmos: nQo5L6YD
Decrypted data = Wakefield
stored data in cosmos: 2fZmuJyOD
Decrypted data = Johnson
stored data in cosmos: KMWCViu
Decrypted data = Smith
stored data in cosmos: 6Ubqq
Document Last Names [nQo5L6YD, 2fZmuJyOD, KMWCViu, 6Ubqq]
Document Ids [Andersen-1619638628449, Wakefield-1619638628738,
Johnson-1619638635251, Smith-1619638635504]
```

SQL Azure Example

This example encrypts sensitive data before it does the JDBC insert data into a SQL instance running on Azure. It then decrypts the sensitive data by calling the fpedecryptdata method. This example 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 SQL Server JDBC class file and the CM REST API.

SQL Azure Application Modifications.

The `cmRESTProtect` method below are the code changes that are required in order to implement encryption and decryption.

```
package com.vormetric.rest.azure_examples;

import java.io.IOException;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import com.thales.cm.rest.cmhelper.CipherTrustManagerHelper;

public class SQLAzureCipherTrustREST {

        CipherTrustManagerHelper ctmh = null;

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

                SQLAzureCipherTrustREST azurerest = new SQLAzureCipherTrustREST();
                azurerest.ctmh = new CipherTrustManagerHelper();

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

                        azurerest.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 connectionUrl =
"jdbc:sqlserver://yourdb.database.windows.net:1433;database=thalescmtest;user=root;password=Yoursupersecre
t123!;encrypt=true;trustServerCertificate=false;hostNameInCertificate=*.database.windows.net;loginTimeout=
30;";
                Connection connection = DriverManager.getConnection(connectionUrl);
                azurerest.fpeencryptdata(connection);
                azurerest.fpedecryptdata(connection);

        }
```

```
        void fpeencryptdata(Connection connection) throws Exception {

                String sensitive = null;

                String insertSql = "insert into creditscore values (?,?)";
                connection.setAutoCommit(true);
                PreparedStatement prepsInsertCreditInfo = connection.prepareStatement(insertSql);

                for (int i = 1; i <= 9; i++) {
                        sensitive = i + "73-56-5628";
                        String encssn = this.ctmh.cmRESTProtect("fpe", sensitive, "encrypt");

                        prepsInsertCreditInfo.setString(1, encssn);
                        prepsInsertCreditInfo.setInt(2, 778+i);
                        boolean returnvalue = prepsInsertCreditInfo.execute();

                        System.out.println("completed insert with " + returnvalue);

                }
        }

        void fpedecryptdata(Connection connection) throws Exception {

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

                        String sql = "SELECT ssn, score FROM creditscore";
                        ResultSet rs = stmt.executeQuery(sql);

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

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

        }
}
```

Sample output

*	ssn	score
1	xUa-90-NIvn	779
2	aSl-Pn-OcHq	780
3	aap-DR-VUOh	781
4	sGt-iG-VF2F	782
5	Syb-8P-81Uq	783
6	leF-Uw-owdh	784
7	Jnl-8P-GXY9	785
8	tdy-yH-0lRB	786
9	sZO-ZR-7Rvp	787

Appendix

Helper Class Example

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

Enabling Soft Delete on Azure Vault

In order to use BYOK for Azure the vault must be enabled for soft delete. Here are the steps to enable it.

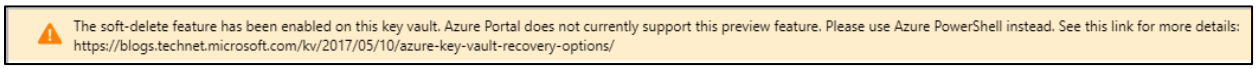
a) Enable soft-delete for your Azure Key Vault using PowerShell. There is no way to do this in the GUI. Enter the following commands one after the other and note that this command grabs the key vault by name.

```
PS Azure:\> ($resource = Get-AzureRmResource -ResourceId (Get-AzureRmKeyVault -VaultName "dsampson").ResourceId).Properties | Add-Member -MemberType "NotePrope
PS Azure:\> Set-AzureRmResource -resourceid $resource.ResourceId -Properties $resource.Properties

Confirm
Are you sure you want to update the following resource: /subscriptions/a549cd04-c63d-4a3d-be08-6e15ae95efa2/resourceGroups/dsampson/providers/Microsoft.KeyVaul
[Y] Yes  [N] No  [S] Suspend  [?] Help (default is "Y"): Y


Name              : dsampson
ResourceId        : /subscriptions/a549cd04-c63d-4a3d-be08-6e15ae95efa2/resourceGroups/dsampson/providers/Microsoft.KeyVault/vaults/dsampson
ResourceName      : dsampson
ResourceType      : Microsoft.KeyVault/vaults
ResourceGroupName : dsampson
Location          : westus
SubscriptionId    : a549cd04-c63d-4a3d-be08-6e15ae95efa2
Tags              : {}
Properties        : @{sku=; tenantId=3ba3fb88-9323-4abe-a23b-a7c9ef72d1b2; accessPolicies=System.Object[]; enabledForDeployment=False; enabledForDiskEncryption
                    enableSoftDelete=True; vaultUri=https://dsampson.vault.azure.net/; provisioningState=Succeeded}



PS Azure:\>
```

```
($resource = Get-AzureRmResource -ResourceId (Get-AzureRmKeyVault -VaultName "vishalSD").ResourceId).Properties | Add-Member -MemberType "NoteProperty" -Name "enableSoftDelete" -Value "true"

Set-AzureRmResource -resourceid $resource.ResourceId -Properties $resource.Properties
```

b) Verify soft-delete is enabled for the key vault with the command below. Another way to check is to log into the Azure portal; there will be a message at the top of the page for the key vault.

```
Get-AzureRmKeyVault -VaultName "vishalSD"
```



This message is displayed in the Azure web interface when soft-delete has been enabled. Not all features and functions are available in the UI, and must be done with PowerShell.

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.