MONGODB

Integration Guide

Applicable Devices:

KMES Series 3

TABLE OF CONTENTS

[1] INTEGRATION OVERVIEW

[1.1] ABOUT MONGODB

MongoDB is a popular, open-source NoSQL database management system. Unlike traditional relational databases which use tables and rows to store data, MongoDB stores data in flexible, JSON-like documents with optional schemas. This allows for greater scalability and ease of use and the ability to handle large amounts of structured and unstructured data. With built-in replication and automatic sharding, MongoDB can effortlessly handle the demands of modern applications, making it an ideal choice for businesses looking to manage and analyze large amounts of data in real time. In addition, its robust security features and scalability make it a great choice for enterprise-level applications.

[1.2] WHAT IS KMIP?

The Key Management Interoperability Protocol (KMIP) is an extensible communication protocol that defines message formats for the manipulation of cryptographic keys on a key management server. This facilitates data encryption by simplifying encryption key management. Keys may be created on a server and then retrieved, possibly wrapped by other keys. Both symmetric and asymmetric keys are supported, including the ability to sign certificates. KMIP also allows for clients to ask a server to encrypt or decrypt data, without needing direct access to the key.

[1.3] PURPOSE OF THE INTEGRATION

From MongoDB's documentation website:

"MongoDB Enterprise 3.2 introduces a native encryption option for the WiredTiger storage engine. Outside MongoDB Atlas (where encryption is handled at the cloud provider level), encryption is only available for enterprise installations that use the WiredTiger Storage Engine. Secure management of the encryption keys is a critical requirement for storage encryption. MongoDB uses a master key that is not stored with the MongoDB installation. Only the master key is externally managed, other keys can be stored with your MongoDB instance. MongoDB's encrypted storage engine supports two key management options for the master key:

- Use of local key management via a keyfile.
- Integration with a third party key management appliance via the Key Management Interoperability Protocol (KMIP). **Recommended**

Important: MongoDB cannot encrypt existing data. When you enable encryption with a new key, the MongoDB instance cannot have any pre-existing data. If your MongoDB installation already has existing data, see Encrypt Existing Data at Rest for additional steps.

MongoDB Enterprise supports secure transfer of keys with compatible key management appliances. Using a key manager allows for the keys to be stored in the key manager. MongoDB Enterprise supports secure transfer of keys with Key Management Interoperability Protocol (KMIP) compliant key management appliances. Any appliance vendor that provides support for KMIP is expected to be compatible."

[1.4] OVERVIEW OF THE STEPS NEEDED FOR INTEGRATION

1. Create TLS certificates for connection and authentication between the MongoDB instance and the KMES Series 3
    a. Generate and sign the MongoDB client certificate
    b. Generate and sign the KMIP server connection pair certificate
2. Create a new role and identity on the KMES Series 3 for MongoDB
3. Configure the TLS certificate for the KMIP server connection pair
4. Copy the TLS certificates to the MongoDB instance
5. Connect MongoDB to the key manager (i.e., KMES Series 3) by starting mongod with the following options:
    - --enableEncryption
    - --kmipServerName
    - --kmipPort
    - --kmipServerCAFile
    - --kmipClientCertificateFile

[2] FUTUREX CERTIFICATION PROCESS

The Futurex Certification Process is a rigorous and standardized approach to testing and certifying integrations between third-party applications and Futurex's HSMs and key management servers (i.e., KMES Series 3). The certification process is designed to ensure that third-party application integrations are fully tested and validated in a lab environment before they are deployed in a production environment. Futurex's Integration Engineering team implements this process so that customers can have confidence that third-party applications will integrate seamlessly with Futurex's HSMs and KMES Series 3 devices, and that all operations will result in the expected behavior. The certification process involves several steps, including research, testing, troubleshooting, and certification, and is fully documented in an integration guide for each integration. The full process is outlined below:

1. Research the third-party application to gain a general understanding of the solution and the protocol it uses to integrate with an HSM or KMS device (i.e., PKCS #11, Microsoft CNG, JCE, OpenSSL Engine, KMIP).
2. Determine the scope of the third-party application's use of the HSM or KMS device, including the specific functionalities it utilizes (i.e., data encryption, key protection, entropy, etc.).
3. Install and configure the third-party application in a lab environment, where all testing and validation will take place.
4. Establish a connection between the third-party application and the Futurex device, which typically involves configuring TLS certificates and creating roles and identities that the third-party application will use to connect and authenticate to the Futurex device.
5. Initiate a request from the third-party application to the Futurex device, such as generating keys or certificates, encrypting or decrypting data, or other cryptographic functions.
6. If any errors occur during the testing process, the Integration Engineering team will diagnose the issues and take necessary corrective actions. If necessary, the team will also document the error(s) by creating engineering change requests (ECRs) to ensure all issues are addressed and resolved before certification.
7. After any necessary engineering changes have been made, a new end-to-end test will be performed to ensure that all errors have been resolved and that all operations are successful.
8. Certify the integration by creating an integration guide that covers all necessary prerequisites, configurations required in both the third-party application and the Futurex device, and how to test the functionality.

Overall, following these steps helps ensure that the integration between the third-party application and the Futurex device is fully tested and validated, and that any errors or issues are resolved before the integration is certified as fully supported.

[3] PREREQUISITES

Supported Hardware:

- KMES Series 3, version 6.3.1.3 and above, with the *KMIP* license enabled

Supported Operating Systems:

- Windows 7 and above
- Linux

Other:

- MongoDB Enterprise

[4] KMES SERIES 3 CONFIGURATION

Before KMIP connections can occur, the MongoDB instance and KMES Series 3 must establish a mutual trust relationship by validating their respective digitally signed certificates.

The following subsections outline how to generate TLS certificates for MongoDB and the KMIP server connection pair on the KMES Series 3. In addition to securing TLS communication, the certificates are also how MongoDB authenticates to the KMES. A role and identity will also be created on the KMES to give MongoDB the permissions it requires to generate the master key and use it for encryption operations.

[4.1] GENERATE AND SIGN THE MONGODB CERTIFICATE

There are two optional methods for generating and signing the MongoDB client certificate:

1. Using an external CA
2. Using the KMES Series 3 as the CA

[4.1.1] Method 1: Using an external CA

For this method, the external CA certificate(s) need to be imported into an empty Certificate Container on the KMES. A Certificate Signing Request (CSR) will then be generated, which the external CA will use to issue a TLS certificate for the MongoDB instance. The certificate will then be imported into the Certificate Container on the KMES that contains the external CA certificate.

1. Navigate to the **PKI** > **Certificate Authorities** menu and click the **[Add CA]** button at the bottom of the page.
2. Specify a name for the Certificate Container, such as "Externally Issued", then click **[OK]**. The new Certificate Container will be listed in the Certificate Authorities menu.
3. Right-click again on the **Externally Issued** Certificate Container and select **Import** > **Certificate(s)...**. This will open the Import Certificates dialog.
4. Click the **Add...** button in the bottom left-hand portion of the dialog, then find and select the external CA certificate(s) that will issue the MongoDB TLS certificate. The CA certificate(s) will populate in the **Verified** section of the Import Certificates dialog.
5. Click **[OK]** to save. The external CA certificate(s) should be listed now in tree form under the **Externally Issued** Certificate Container.
6. Next, we'll create a placeholder code signing certificate, from which a CSR can be generated. Right-click on the lowest level CA certificate in the tree and select **Add Certificate** -> **Pending...**. This will open the Create X.509 Certificate dialog.
7. In the **Subject DN** tab, set a Common Name for the certificate, such as "MongoDB".
8. Leave all other values as the default and click **[OK]**. The **MongoDB** placeholder certificate will be listed now under the external CA certificate(s).

9. Right-click on placeholder **MongoDB** certificate and select **Export** -> **Signing Request...**. This will open the Create PKCS #10 Request dialog.
10. Leave all of the settings in the **Subject DN** tab as the default values.
11. In the **V3 Extensions** tab, select the **Example TLS Client Certificate** profile.
12. In the **PKCS #10 Info** tab, specify a save location for the CSR, then click **[OK]**. There should be a message stating that the certificate signing request was successfully written to the location you specified.
13. The CSR file then needs to be taken to an external certificate authority. Using the CSR, the external CA will issue a TLS certificate.

    Note: After the external CA issues the the signed certificate, the certificate needs to be copied to the storage medium configured on the KMES.
14. In the **PKI** > **Certificate Authorities** menu on the KMES, right-click on the placeholder **MongoDB** certificate and select **Replace** -> **With Signed Certificate...**. This will open the Import Certificates dialog.
15. Click the **Add...** button in the bottom left-hand portion of the dialog, then find and select the externally signed TLS certificate. The certificate will populate under the CA certificate(s) in the Verified section.
16. Click **[OK]** to save.
17. The remaining steps in this section involve exporting the MongoDB certificate as a PKCS #12 file. To be able to do this, there is a configuration option that must be enabled. Navigate to Administration > Configuration > Options and check the box next to the second menu option, which says, "Allow export of certificates using passwords". Then click **[Save]**.
18. Now, right-click on the MongoDB certificate and select **Export** -> **PKCS12...**.
19. In the Export PKCS12 window, set a password for the PKCS #12 file and set Export Options to **Export Selected Certificate**, then click **[Next]**.
20. In the file browser, specify a name for the file and select a save location, then click **[Open]**.

    Note: The PKCS #12 file contains the signed MongoDB certificate and its associated private key, encrypted under the password set for the file. It needs to be copied to the machine that is running MongoDB, along with the external CA certificate chain that signed it.

[4.1.2] Method 2: Using the KMES Series 3 as the CA

1. Navigate to the **PKI** > **Certificate Authorities** menu and click the **[Add CA]** button at the bottom of the page.
2. Specify a name for the Certificate Container, such as "KMES Issued", then click **[OK]**. The new Certificate Container will be listed in the Certificate Authorities menu.
3. Right-click on the newly created **KMES Issued** Certificate Container and select **Add Certificate > New Certificate...**

4. In the **Subject DN** tab, select the **Classic** Preset and set a Common Name for the certificate, such as "Root".
5. In the **Basic Info** tab, leave all values set to the defaults.
6. In the **V3 Extensions** tab, select the **Example Certificate Authority** profile, then click **[OK]**. The **Root** CA certificate will be listed now inside the **KMES Issued** Certificate Container.
7. Right-click on the **Root** CA certificate you just created and select **Add Certificate** > **New Certificate...**
8. In the **Subject DN** tab, set a Common Name for the certificate, such as "MongoDB".
9. In the **Basic Info** tab, leave all values set to the defaults.
10. In the **V3 Extensions** tab, change the profile to **Example TLS Client Certificate**, then click **[OK]** to finish generating the certificate.
11. The remaining steps in this section involve exporting the MongoDB certificate as a PKCS #12 file. To be able to do this, there is a configuration option that must be enabled. Navigate to **Administration** > **Configuration** > **Options** and check the box next to the second menu option, which says, "Allow export of certificates using passwords". Then click **[Save]**.
12. Now, right-click on the MongoDB certificate and select **Export** -> **PKCS12...**.
13. In the Export PKCS12 window, set a password for the PKCS #12 file and set Export Options to **Export Selected Certificate**, then click **[Next]**.
14. In the file browser, specify a name for the file and select a save location, then click **[Open]**.

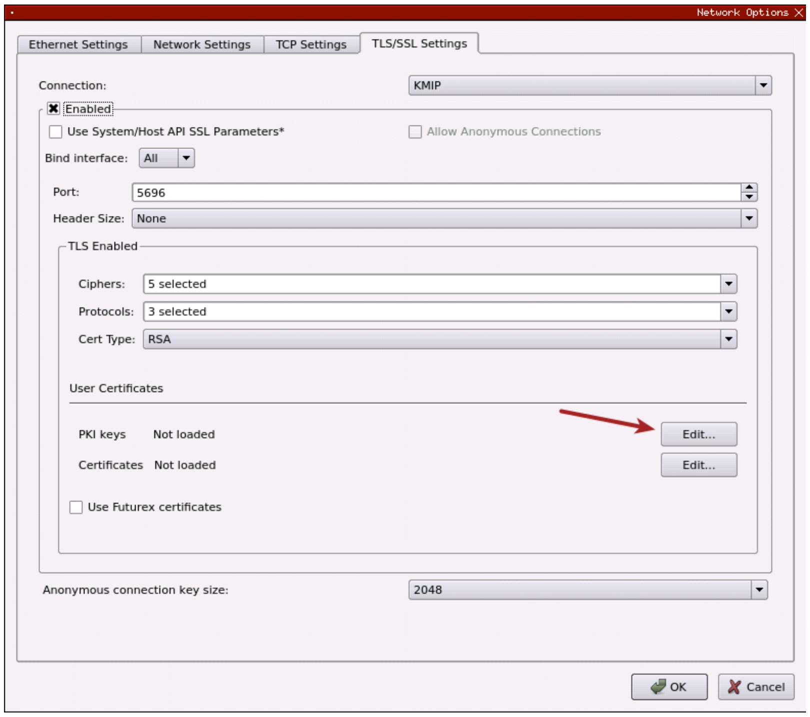
    Note: The PKCS #12 file contains the signed MongoDB certificate and its associated private key, encrypted under the password set for the file. It needs to be copied to the machine that is running MongoDB.

[4.2] CONFIGURE TLS CERTIFICATE FOR THE KMIP SERVER CONNECTION PAIR

[4.2.1] Generate a new PKI key pair and CSR for the KMIP connection pair

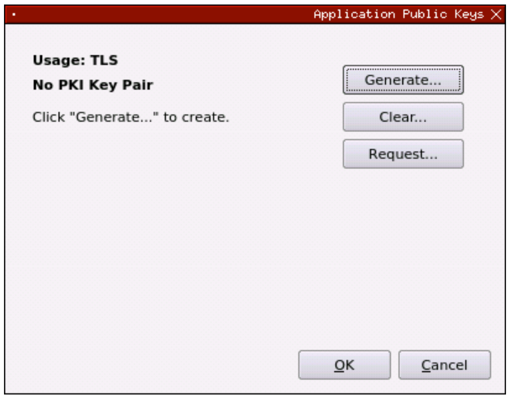
1. Navigate to **Administration** > **Configuration** > **Network Options** > **TLS/SSL Settings**.
2. Click the **Connection** dropdown and select the **KMIP** connection pair. Enable the KMIP connection pair if it is not already enabled.
3. Uncheck **Use System/Host API SSL Parameters** if it is selected.

4. In the User Certificates section, uncheck **Use Futurex certificates** if it is selected and click the **[Edit...]** button next to PKI keys.



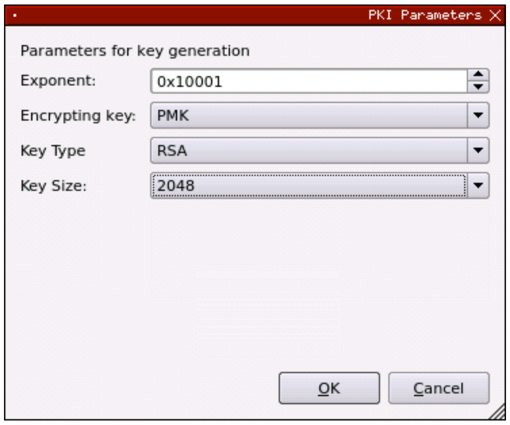
Connection: KMIP
☒ Enabled
☐ Use System/Host API SSL Parameters*
☐ Allow Anonymous Connections
Bind interface: All
Port: 5696
Header Size: None
TLS Enabled
Ciphers: 5 selected
Protocols: 3 selected
Cert Type: RSA
User Certificates
PKI keys Not loaded Edit...
Certificates Not loaded Edit...
☐ Use Futurex certificates
Anonymous connection key size: 2048

5. Click the **[Generate...]** button to create a new PKI Key Pair. This will open the PKI Parameters dialog.



Usage: TLS
No PKI Key Pair
Click "Generate..." to create.

6. Leave the default settings shown below and click **[OK]**.


PKI Parameters

Parameters for key generation

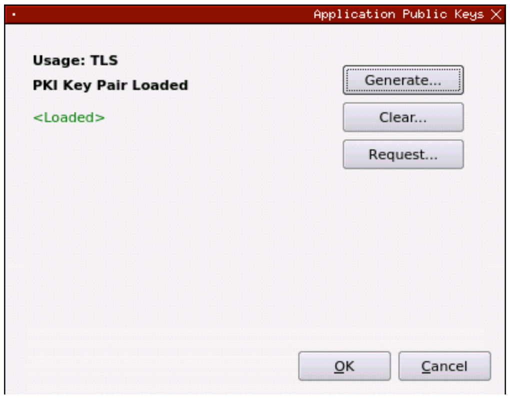
Exponent: 0x10001

Encrypting key: PMK

Key Type: RSA

Key Size: 2048

7. The Application Public Keys dialog should now show that the PKI Key Pair is **Loaded**. If this is the case, click **[Request...]**. This will open the Create PKCS #10 Request dialog.


Application Public Keys

Usage: TLS

PKI Key Pair Loaded

<Loaded>

8. In the **Subject DN** tab, change the **Common Name** value to the IP of the KMES.
9. In the **V3 Extensions** tab, set the profile to **Example TLS Server Certificate**.
10. In the **PKCS #10 Info** tab, specify a save location and name for the CSR file, then click **[OK]**.
11. A message box should appear saying that the certificate signing request was successfully written to the specified location. Click **[OK]**.
12. Click **[OK]** in the Application Public Keys dialog, then click **[OK]** once more in the main Network Options dialog.

[4.2.2] Issue a certificate from the KMIP connection pair CSR

1. Navigate to the **PKI** > **Certificate Authorities** menu, then right-click on the root CA certificate that issued the MongoDB TLS certificate in section 3.1 and select **Add Certificate** > **From Request...**.
2. In the file browser, find and select the KMIP connection pair CSR. Certificate information should populate in the Create X.509 From CSR window.
3. Leave all settings exactly as they are and click **[OK]** to save.
4. The signed KMIP server certificate should be listed now under the root CA certificate that issued it.

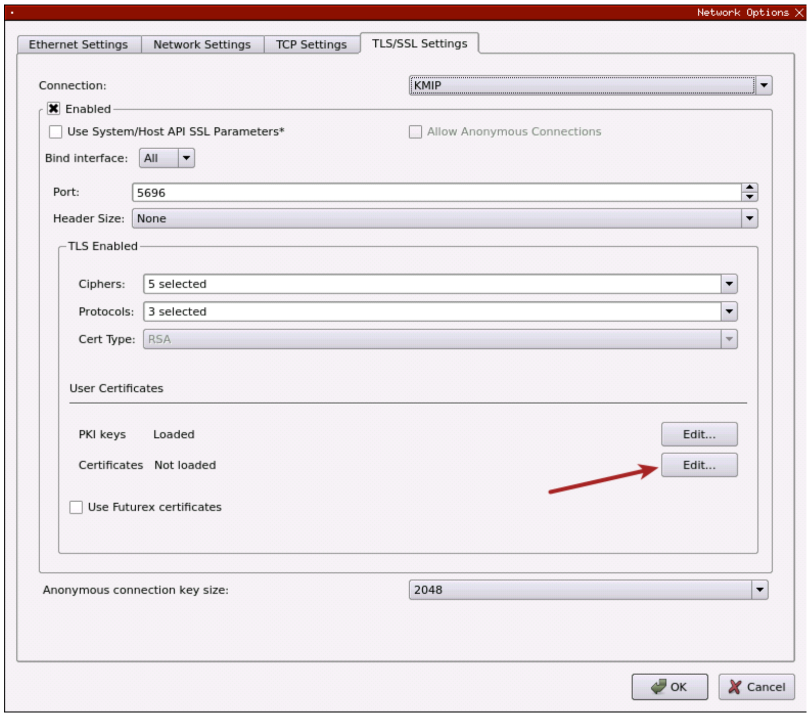
[4.2.3] Export the root CA and KMIP certificates as PEM files

For both the root CA certificate and the signed KMIP connection pair certificate, right-click on them and select **Export** -> **Certificate(s)...**. In the Export Certificate dialog for each, change the encoding to **PEM**, then specify a save location for the file.

Note: The root CA certificate needs to be copied to the machine that is running MongoDB.

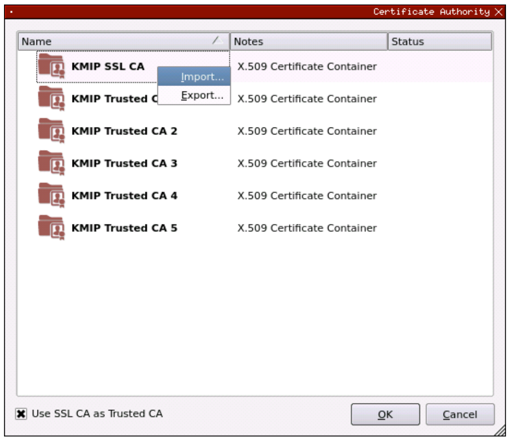
[4.2.4] Import the signed KMIP connection pair certificate

1. Navigate to **Administration** > **Configuration** > **Network Options** > **TLS/SSL Settings**.
2. Click the **Connection** dropdown and select the **KMIP** connection pair.
3. Click the **[Edit...]** button next to Certificates in the User Certificates section.



Connection: KMIP
[✓ Enabled]
[ ] Use System/Host API SSL Parameters*
[ ] Allow Anonymous Connections
Bind interface: All
Port: 5696
Header Size: None
TLS Enabled
Ciphers: 5 selected
Protocols: 3 selected
Cert Type: RSA
User Certificates
PKI keys Loaded
Certificates Not loaded
[ ] Use Futurex certificates
Anonymous connection key size: 2048

4. In the Certificate Authority dialog, right-click on the **KMIP SSL CA** X.509 Certificate Container, then select **Import...**.



Name	Notes	Status
KMIP SSL CA	X.509 Certificate Container	
KMIP Trusted C	X.509 Certificate Container	
KMIP Trusted CA 2	X.509 Certificate Container	
KMIP Trusted CA 3	X.509 Certificate Container	
KMIP Trusted CA 4	X.509 Certificate Container	
KMIP Trusted CA 5	X.509 Certificate Container	

☒ Use SSL CA as Trusted CA

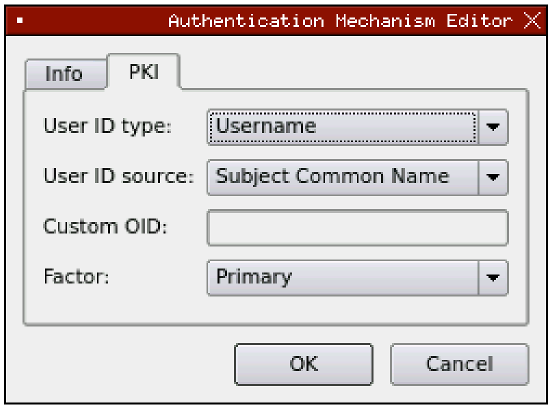
5. Click the **[Add...]** button at the bottom of the Import Certificates dialog. In the file browser, select both the root CA certificate and the signed KMIP server certificate and click **[Open]**. The certificates should now be listed in the Verified section of the Import Certificates dialog. Click **[OK]** to save.
6. It should now say **Signed loaded** next to Certificates in the User Certificates section of the Network Options dialog. Click **[OK]** to save.

[4.3] ADD A PKI IDENTITY PROVIDER CONFIGURED WITH THE TLS AUTHENTICATION MECHANISM

A new PKI Identity Provider needs to be created, assigned a TLS authentication mechanism, and added to an identity as a credential. This will allow MongoDB to authenticate with the KMES using its TLS certificate.

1. Navigate to the **Identity Management** > **Identity Providers** menu.
2. Right-click anywhere in the window and select **Add** > **Provider** > **PKI**. This will open the **Identity Provider Editor** dialog.
3. In the **Info** tab, specify a name for the Identity Provider and uncheck **Enforce Dual Factor**.
4. In the **PKI Options** tab, click the **[Select]** button. In the **Certificate Selector** dialog, expand the certificate tree you created in section 3.1 and select the CA certificate that signed the MongoDB and KMIP connection pair certificates, then click **[OK]**.
5. Click **[OK]** to finish creating the PKI Identity Provider.
6. Right-click on the Identity Provider you just created and select **Add** > **Mechanism** > **TLS**.
7. In the **Info** tab, specify a name for the authentication mechanism.

8. In the **PKI** tab, leave all fields set to the default values, as shown below:


Authentication Mechanism Editor

Info | PKI

User ID type: Username

User ID source: Subject Common Name

Custom OID:

Factor: Primary

9. Click **[OK]** to save.

[4.4] CREATE A ROLE AND IDENTITY FOR MONGODB WITH THE REQUIRED PERMISSIONS

A new role and identity need to be created on the KMES Series 3, which MongoDB will use for authentication during KMIP connections. The name of this identity must match exactly what is set later as the **Common Name** for the signed MongoDB certificate. This is how the KMES Series 3 authenticates the MongoDB device that is connecting via KMIP.

Creating a role

1. Log in to the KMES Series 3 application interface with the default Admin identities.
2. Go to the **Identity Management** > **Roles** menu, then click the **[Add...]** button. This will pull up the **Role Editor** dialog.
3. Under the **Info** tab, set the following:
    - Type -> **Application**
    - Name -> **MongoDB**
    - Login Required > **1**
4. Under the **Permissions** tab, and select the following permissions:
    - Cryptographic Operations -> Sign, Verify, Encrypt, Decrypt
    - Keys -> Add, Export
5. Under the **Advanced** tab, set Allowed Ports to **KMIP** only.
6. Click the **[OK]** button to finish creating the role.

Creating an identity

1. Go to the **Identity Management** > **Identities** menu, right-click in the window and select **Add** > **Client Application**. This will pull up the Identity Editor dialog.
2. Under the **Info** tab, select **Application** for the storage location, and specify a **name** for the identity.
3. Under **Assigned Roles,** select the role you created for MongoDB.
4. Under **Authentication**, remove the default API Key mechanism and click the **[Add]** button to add a new credential. In the Configure Credential dialog, select **TLS Certificate** in the Type dropdown, then select the Provider and Mechanism you created in section 3.3. Click **[OK]** to finish configuring the credential.
5. Click **[OK]** to finish creating the identity.

[4.5] GRANT THE MONGODB ROLE "USE" PERMISSIONS ON THE PKI IDENTITY PROVIDER AND THE CERTIFICATE CONTAINER

1. Navigate to the **Identity Management** > **Identity Providers** menu.
2. Right-click the PKI identity provider created in section 3.3 and select **Permission...**
3. Set the **Use** permission for the **MongoDB** role and click **[OK]** to save.
4. Navigate to the **PKI** > **Certificate Authorities** menu.
5. Right-click the certificate container created in section 3.1 and select **Permission...**
6. Set the **Use** permission for the **MongoDB** role and click **[OK]** to save.

[5] TEST A CONNECTION FROM MONGODB TO THE KMES SERIES 3 WITH THE CONFIGURED TLS CERTIFICATES USING OPENSSL

To confirm that the MongoDB client certificate enables a successful TLS connection to the KMIP port on the KMES Series 3, you can use OpenSSL. Instructions are provided below depending on whether you used an external CA or a CA on the KMES to issue the MongoDB client certificate.

[5.1] EXTERNALLY-ISSUED MONGODB CLIENT CERTIFICATE

If you're using an externally-issued MongoDB client certificate, you'll need to extract the client certificate and private key from the PKCS #12 file before attempting to connect. You'll also need to obtain the external CA certificate chain that signed the MongoDB client certificate and save it to a file.

The following instructions explain how to extract a signed certificate and private key from a PKCS #12 file and save them to their own files.

First, run the command below to extract the private key and signed certificate from the PKCS #12 file and save them in a single PEM file called mongodb_cert_and_privatekey.pem. Note that the -nodes flag is used to specify that the private key should not be encrypted.

```
$ openssl pkcs12 -in mongodb.p12 -nodes -out mongodb-cert-and-privatekey.pem
```

Once you have the mongodb_cert_and_privatekey.pem file, you can extract the signed certificate and private key into separate files using the commands below:

```
$ openssl rsa -in mongodb-cert-and-privatekey.pem -out mongodb-privatekey.pem
$ openssl x509 -in mongodb-cert-and-privatekey.pem -out mongodb-signed-cert.pem
```

Now, run the following OpenSSL command to test a connection to the KMIP connection pair on the KMES Series 3:

```
$ openssl s_client -connect <KMES-IP>:5696 -CAfile external-ca-chain.pem -cert mongodb-signed-cer-
t.pem -key mongodb-privatekey.pem
```

Note: Be sure to replace <KMES-IP> with the IP address of the KMES Series 3 and adjust the file names as necessary.

If the TLS handshake is successful, then the certificates were correctly configured on the KMES Series 3.

[5.2] KMES-ISSUED MONGODB CLIENT CERTIFICATE

If you're using a KMES-issued MongoDB client certificate, you'll need to extract the client certificate and private key from the PKCS #12 file before attempting to connect. You'll also need to obtain the root CA certificate that signed the MongoDB client certificate and save it to a file.

The following instructions explain how to extract a signed certificate and private key from a PKCS #12 file and save them to their own files.

First, run the command below to extract the private key and signed certificate from the PKCS #12 file and save them in a single PEM file called mongodb_cert_and_privatekey.pem. Note that the -nodes flag is used to specify that the private key should not be encrypted.

```
$ openssl pkcs12 -in mongodb.p12 -nodes -out mongodb-cert-and-privatekey.pem
```

Once you have the mongodb_cert_and_privatekey.pem file, you can extract the signed certificate and private key into separate files using the commands below:

```
$ openssl rsa -in mongodb-cert-and-privatekey.pem -out mongodb-privatekey.pem
$ openssl x509 -in mongodb-cert-and-privatekey.pem -out mongodb-signed-cert.pem
```

Now, run the following OpenSSL command to test a connection to the KMES Series 3:

```
$ openssl s_client -connect <KMES-IP>:5696 -CAfile root-ca-cert.pem -cert mongodb-signed-cert.pem -
key mongodb-privatekey.pem
```

Note: Be sure to replace <KMES-IP> with the IP address of the KMES Series 3 and adjust the file names as necessary.

If the TLS handshake is successful, then the certificates were correctly configured on the KMES Series 3.

[6] CONFIGURING ENCRYPTION IN MONGODB

[6.1] OVERVIEW

This section discusses server configuration to support encryption at rest in MongoDB. MongoDB Enterprise 3.2 introduces a native encryption option for the WiredTiger storage engine.

Secure management of the encryption keys is a critical requirement for storage encryption. MongoDB uses a master key that is not stored with the MongoDB installation. Only the master key is externally managed, other keys can be stored with your MongoDB instance.

MongoDB's encrypted storage engine supports two key management options for the master key:

- Use of local key management via a keyfile.
- Integration with a third party key management appliance (i.e., the KMES Series 3) via the Key Management Interoperability Protocol (KMIP). **Recommended**

Important: MongoDB cannot encrypt existing data. When you enable encryption with a new key, the MongoDB instance cannot have any pre-existing data. If your MongoDB installation already has existing data, see Encrypt Existing Data at Rest for additional steps.

Note: Changed in version 4.0
MongoDB Enterprise on Windows no longer supports AES256-GCM. This cipher is now available only on Linux.

[6.2] INTEGRATE USING A NEWLY-GENERATED KEY

[6.2.1] Start the MongoDB server and enable encryption by generating a new key on the KMES via KMIP

1. Create the directory /data/db to store the data directory files.

```
sudo mkdir -p /data/db/
```

2. Set the current user as the owner of the /data/db directory.

```
sudo chown -R $USER:$USER /data/db
```

3. Remove the MongoDB .sock file from the /tmp directory if one exists.

```
sudo rm /tmp/mongodb-27017.sock
```

4. Create a new master key on the KMES Series 3, which mongod will use to encrypt the keys mongod generates for each database.

```
mongod --dbpath /data/db --enableEncryption --kmipServerName <KMES-IP> --kmipPort 5696 --kmi-
pServerCAFile root-ca-cert.pem --kmipClientCertificateFile mongodb-cert-and-privatekey.pem --
port 27018
```

Note: The file you specify in the --kmipClientCertificateFile flag must contain both the signed MongoDB certificate and its associated private key.

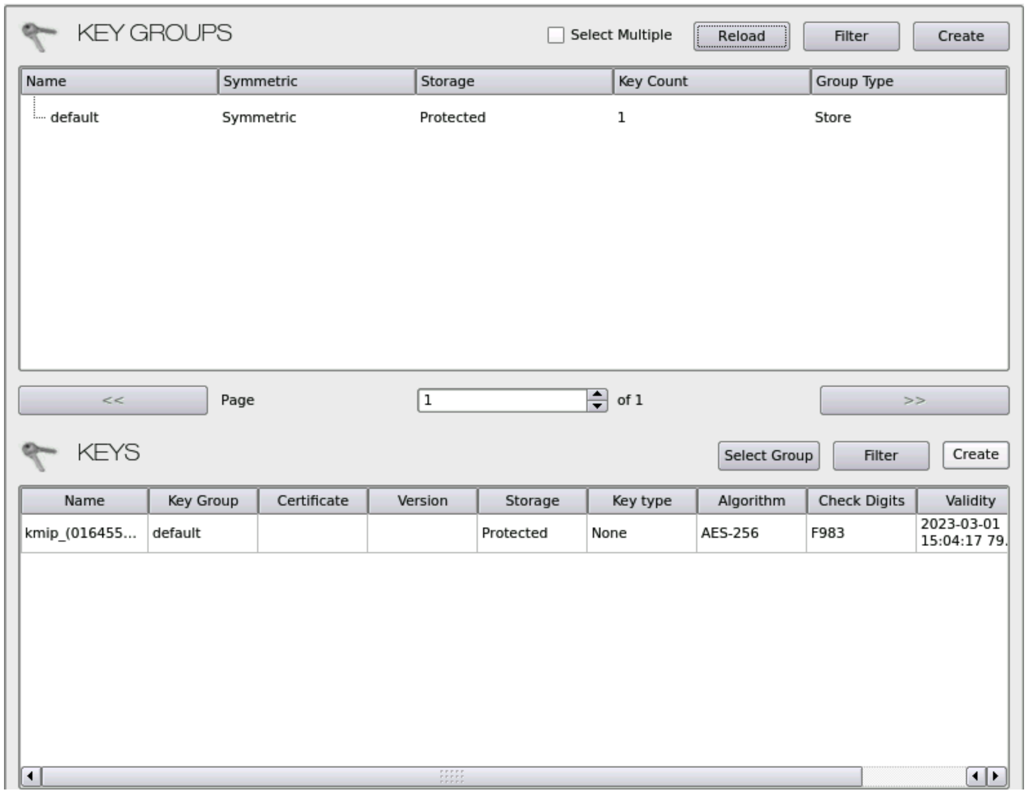
5. When connecting to the KMIP server, the mongod verifies that the specified --kmipServerName matches the Subject Alternative Name SAN (or, if SAN is not present, the Common Name CN) in the certificate presented by the KMIP server. If SAN is present, mongod does not match against the CN. If the hostname does not match the SAN (or CN), the mongod will fail to connect.

   To verify that the key creation and usage was successful, check the log file. If successful, the process will log the following messages:

```
[initandlisten] Created KMIP key with id: <UID>
[initandlisten] Encryption key manager initialized using master key with id: <UID>
```

[6.2.2] View the master key MongoDB created on the KMES

1. Log in to the KMES Series 3 application interface with the default admin identities.
2. Navigate to the **Key Management** > **Keys** menu.
3. Select the "default" key group to view the AES-256 key that MongoDB created via KMIP.



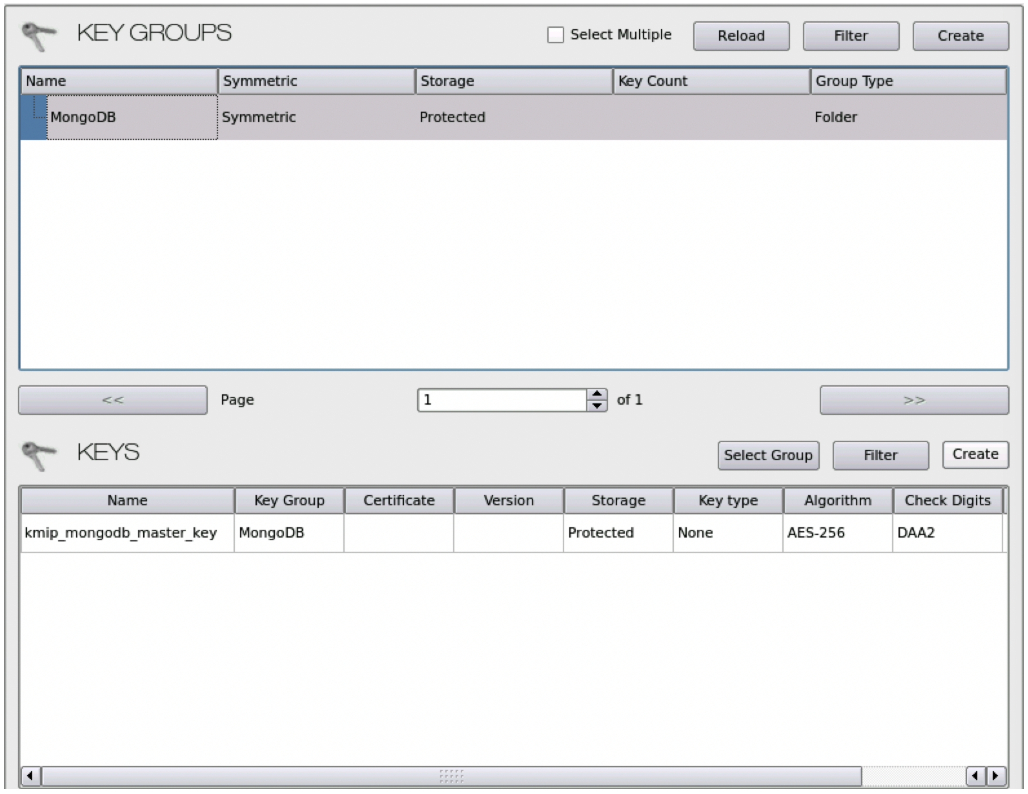
Name	Symmetric	Storage	Key Count	Group Type
default	Symmetric	Protected	1	Store

Name	Key Group	Certificate	Version	Storage	Key type	Algorithm	Check Digits	Validity
kmip_(016455...	default			Protected	None	AES-256	F983	2023-03-01 15:04:17 79.

[6.3] INTEGRATE USING AN EXISTING KEY

[6.3.1] Generate a key on the KMES for MongoDB to use as its master key

1. Log in to the KMES Series 3 application interface with the default admin identities.
2. Navigate to the Key Management > Keys menu.
3. Select the **[Create]** button in the Key Groups section.
4. Select **Symmetric** as the Key Type and **HSM Protected** for the Storage Location.
5. In the HSM Protected Key Group dialog:
    - Specify a **name** for the Key Group.
    - In the Service dropdown, select **Key Management Interoperability Protocol**.
    - Change the Key Length to **AES-256**.
    - Click **[OK]**.
6. Select the key group you just created and in the Keys section click the **[Create]** button and select **Random**.
7. Specify a name for the key and click **[OK]**.



Name	Symmetric	Storage	Key Count	Group Type
MongoDB	Symmetric	Protected		Folder

Name	Key Group	Certificate	Version	Storage	Key type	Algorithm	Check Digits
kmip_mongodb_master_key	MongoDB			Protected	None	AES-256	DAA2

Note: The name specified for the the key will be the UID value that you pass into the --kmipKeyIdentifier flag in the next subsection.

[6.3.2] Start the MongoDB server and enable encryption using the existing key on the KMES

1. Create the directory /data/db to store the data directory files.

```
sudo mkdir -p /data/db/
```

2. Set the current user as the owner of the /data/db directory.

```
sudo chown -R $USER:$USER /data/db
```

3. Remove the MongoDB .sock file from the /tmp directory if one exists.

```
sudo rm /tmp/mongodb-27017.sock
```

4. Start MongoDB using the existing key on the KMES Series 3, which mongod will use to encrypt the keys mongod generates for each database.

```
mongod --dbpath /data/db --enableEncryption --kmipServerName <KMES-IP> --kmipPort 5696 --kmi-
pServerCAFile root-ca-cert.pem --kmipClientCertificateFile mongodb-cert-and-privatekey.pem --
port 27018 --kmipKeyIdentifier <UID>
```

   Note: The file you specify in the --kmipClientCertificateFile flag must contain both the signed MongoDB certificate and its associated private key.

   Note: The UID value you specify in the --kmipKeyIdentifier flag needs to be the name of the key that was created on the KMES in the previous subsection.

5. When connecting to the KMIP server, the mongod verifies that the specified --kmipServerName matches the Subject Alternative Name SAN (or, if SAN is not present, the Common Name CN) in the certificate presented by the KMIP server. If SAN is present, mongod does not match against the CN. If the hostname does not match the SAN (or CN), the mongod will fail to connect.

   To verify that the key usage was successful, check the log file. If successful, the process will log the following message:

```
[initandlisten] Encryption key manager initialized using master key with id: <UID>
```

APPENDIX A: XCEPTIONAL SUPPORT

In today's high-paced environment, we know you are looking for timely and effective resolutions for your mission-critical needs. That is why our Xceptional Support Team does whatever it takes to ensure you have the best experience and support possible. Every time. Guaranteed.

- 24x7x365 mission critical support
- Level 1 to level 3 support
- Extremely knowledgeable subject matter experts

At Futurex, we strive to supply you with the latest data encryption innovations as well as our best-in-class support services. Our Xceptional Support Team goes above and beyond to meet your needs and provide you with exclusive services that you cannot find anywhere else in the industry.

- Technical Services
- Onsite Training
- Virtual Training
- Customized Consulting
- Customized Software Solutions
- Secure Key Generation, Printing, and Mailing
- Remote Key Injection
- Certificate Authority Services

Toll-Free: 1-800-251-5112

E-mail: support@futurex.com

ENGINEERING CAMPUS

864 Old Boerne Road

Bulverde, Texas, USA 78163

Phone: +1 830-980-9782

+1 830-438-8782

E-mail: info@futurex.com

XCEPTIONAL SUPPORT

24x7x365

Toll-Free: 1-800-251-5112

E-mail: support@futurex.com

SOLUTIONS ARCHITECT

E-mail: solutions@futurex.com