Securing MongoDB Cluster with TLS/SSL

mydbops.com/blog/securing-mongodb-cluster-with-tlsssl/

January 13, 2021

The world is in a pandemic situation and I believe everyone doing good.

Please Stay Home! Stay Safe! In this situation, most of the people are doing WFH, and I believe its the right time to talk about how to enable the Developer, OPS Team, and DBA can initiate secure encrypted TCP connection instead of Plain TCP connection to the Database.

In this blog am going to discuss the below list of topics to get an ampler understanding of the X.509 Certificate and configuring MongoDB cluster to use TLS/SSL.

1. What is the X.509 Certificate ?
2. TLS/SSL Authentication
3. Create a self-signed certificate for each replica-set member
4. Create a self-signed certificate for mongo client / Driver
5. Configure a Replica-Set to Support TLS/SSL

X.509 Certificate

Every time one talks about the X.509 certificate, people confused X.509 with TLS/SSL. Here I like to take a chance to give a little picture of the confusion.

- In cryptography, X.509 defines the format of Public Key Infrastructure certification
- TLS/SSL is an Application layer protocol
- TLS/SSL cryptographic protocols designed to provide communication security over a computer network
- Many TCP based protocols like HTTPS, SMTPS, FTPS, etc use TLS/SSL protocol.

X.509 Certificate In MongoDB

In MongoDB deployment, we can utilize the X.509 certificate in encrypting all MongoDB Traffic, User Authentication, and Internal Replica member authentication.

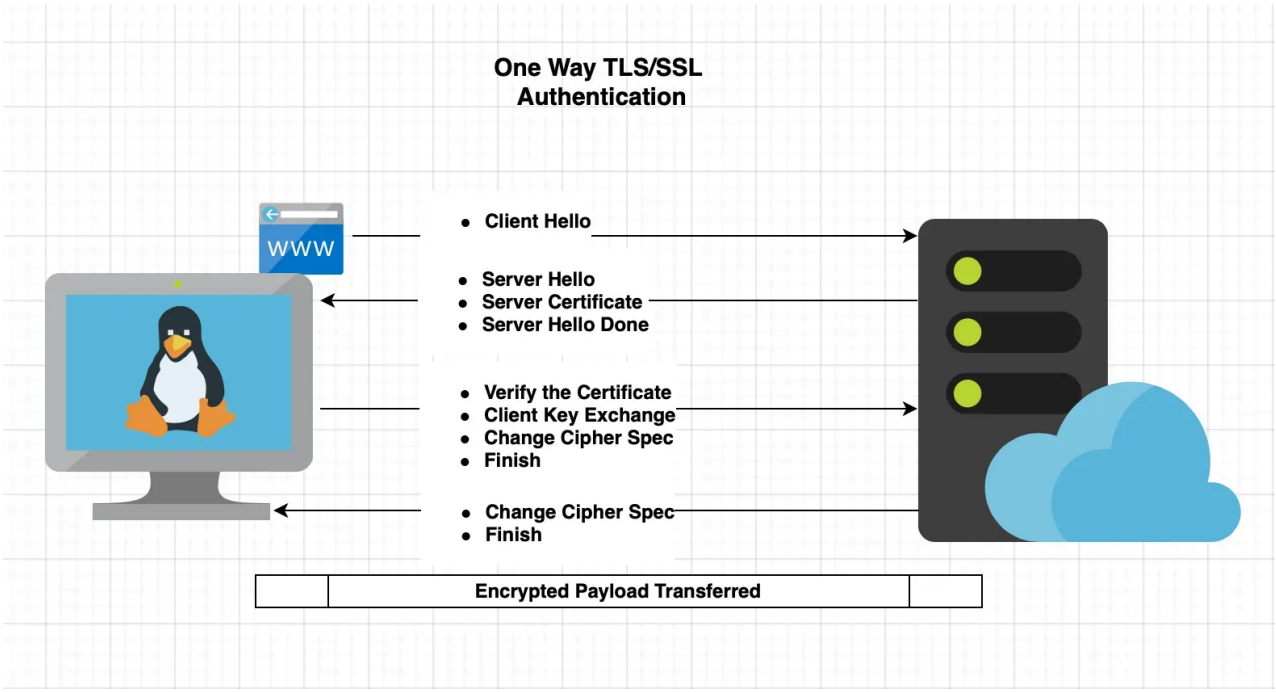
TLS/SSL Authentication Types:

After getting some clarity between X.509 and TLS/SSL, the next piece we need to understand SSL Authentication type.

One-Way TLS/SSL Authentication

In our day to day life, we are using One-way SSL Authentication in the way of accessing all websites running in HTTPS protocol(Banking, Social network, etc).
An important aspect of this authentication method is, Here client only validates/verify the server identity through the Certification Authority of the CA-signed public certificate shared by the server.

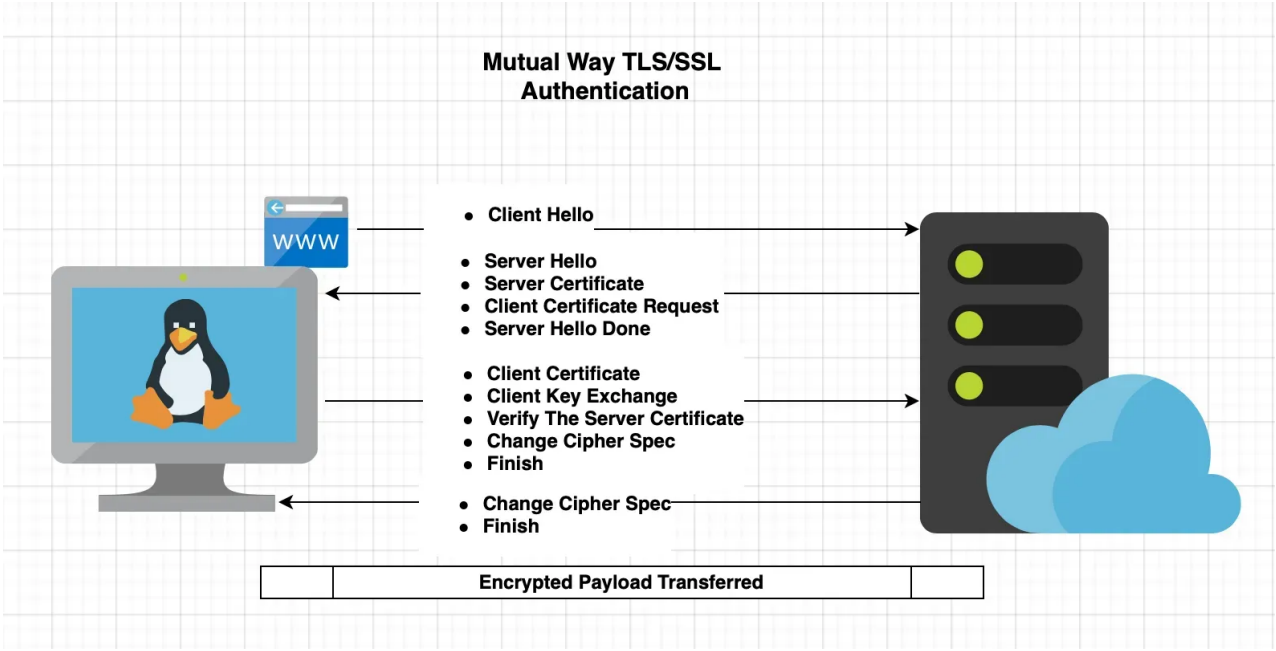
One-way SSL Auth



Mutual-Way TLS/SSL Authentication

In Mutual-Way SSL authentication, the server is authenticating itself to the client, and the client is authenticating itself to the server to establish a secure encrypted channel between them. Apart from a secure encrypted channel, it is all about ensuring that both parties involved in the communication are trusted.

Mutual-way SSL Auth



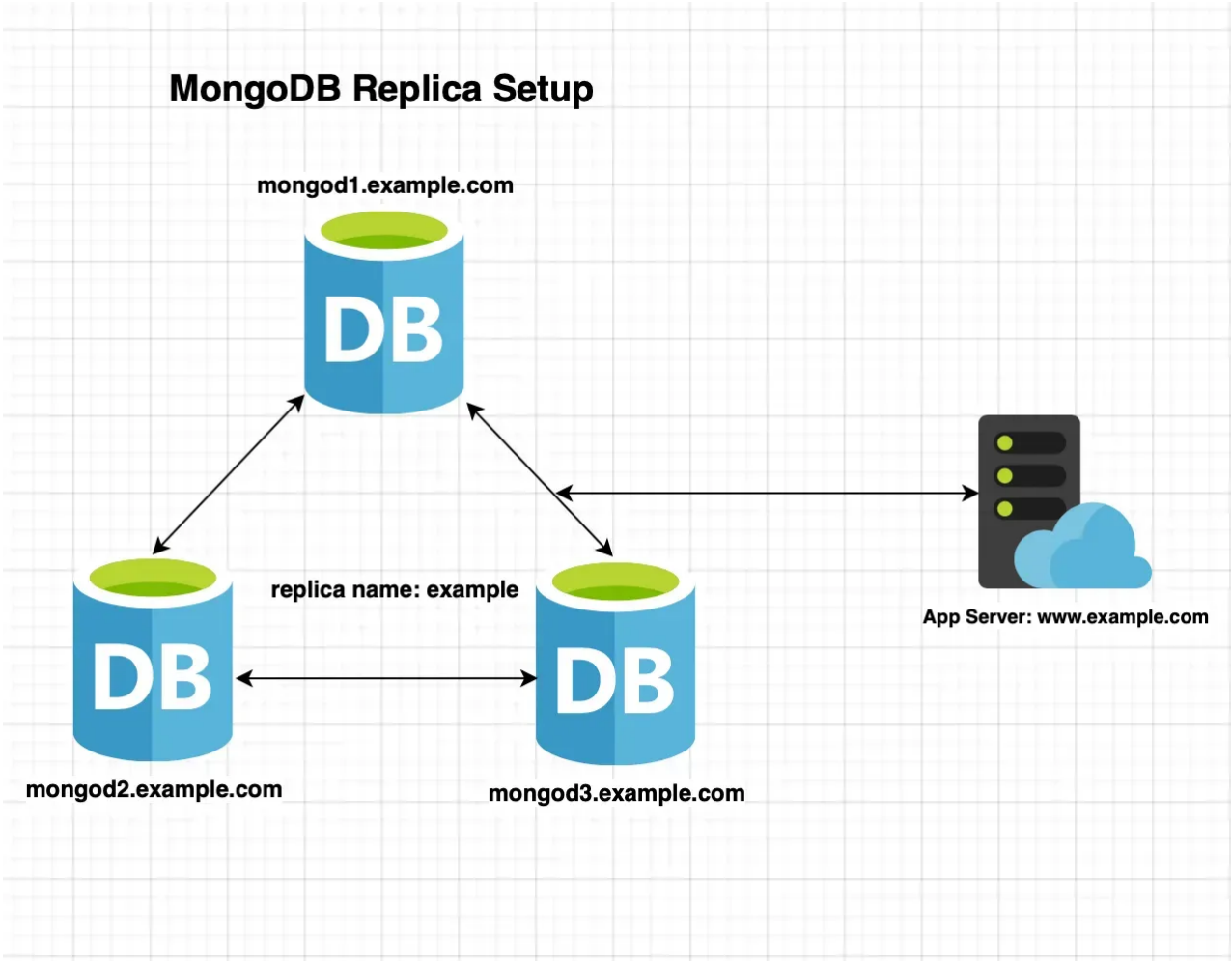
TLS/SSL Authentication in MongoDB

MongoDB only supports Mutual-way TLS/SSL Authentication, it helps to encrypt all of MongoDB's network traffic. And TLS/SSL ensures that MongoDB transactions are only readable by the intended client.

Create a self-signed certificate for each replica-set member

Before proceeding to create the certificates we need to understand replica set architecture is very relevant. Therefore I have presented an example replica set below, based on that we will create the certificate and do the all required Mongod configuration.

All the details we will see below.



To use encryption, we need to create certificates for all the replica members and have a certification authority (CA) to signs them. Since having a certification authority can be pretty costly, we decide to use self-signed certificates.

To proceed with certificate generation we need to have **OpenSSL** installed on our system and certificates need to satisfy these requirements:

- All the replica member and client certificates need to be signed by the same CA.
- The common name (CN) required during the certificate creation must correspond to the hostname of the host.
- Any other field requested in the certificate creation should be a non-empty value and, hopefully, should reflect our organisation details.
- It is also very important that all the fields, except the CN, should match those from the certificates for the other cluster members.

TLS/SSL certificate validation toward Common Name or SAN

when the user attempting to connect a host using a TLS/SSL certificate, the FQDN or IP will be validated against Common Name or SAN Fields of the certificate. After the successful validation only, the connection will be established.

```
mongo -u root --port=12029 --authenticationDatabase=admin -p 'root' -ssl --
sslCAFile example-ca-pub.crt --sslPEMKeyFile mongod1.pem --
host=mongod1.example.com
```

Create CA Private Certificate

Command:

```
openssl genrsa -out example-ca.key -aes256 8192

root@CA-server:~# openssl genrsa -out example-ca.key -aes256 8192
Generating RSA private key, 8192 bit long modulus (2 primes)
.....................................................................
.....................................+++
.....................................................................
.....+++
e is 65537 (0x010001)
Enter pass phrase for example-ca.key:
Verifying - Enter pass phrase for example-ca.key:
```

we need to enter a strong password and save it in a secure vault. because using the example-ca.key and above entered password only we will sign other replica members certificates.

Sign CA Public Certificate

Command

```
openssl req -x509 -new -extensions v3_ca -key example-ca.key -days 365 -out
example-ca-pub.crt

root@CA-server:~# openssl req -x509 -new -extensions v3_ca -key example-ca.key -
days 365 -out example-ca-pub.crt
Enter pass phrase for example-ca.key:
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:IN
State or Province Name (full name) [Some-State]:Karnataka
Locality Name (eg, city) []:BLR
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Example
Organizational Unit Name (eg, section) []:CA-Team
Common Name (e.g. server FQDN or YOUR name) []:CA-server
Email Address []: mmalai@example.com
```

This CA public certificate needs to share with all member certificate.

Generate the CSR and Private key for all member

For each replica member, we need to generate a CSR and Private Key. Finally, sign each CSR using the CA private key.

Remember: Fill out all the fields requested the same for each host, but remember to fill out a different common name (CN) that must correspond to the hostname.

Command

```
openssl req -nodes -newkey rsa:4096 -sha256 -keyout mongod1.key -out mongod1.csr
openssl req -nodes -newkey rsa:4096 -sha256 -keyout mongod2.key -out mongod2.csr
openssl req -nodes -newkey rsa:4096 -sha256 -keyout mongod3.key -out mongod3.csr

root@CA-server:~# openssl req -nodes -newkey rsa:4096 -sha256 -keyout mongod1.key
-out mongod1.csr
Generating a RSA private key
.....................................................................++++
.....................................++++
writing new private key to 'mongodb1.key'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:IN
State or Province Name (full name) [Some-State]:Karnataka
Locality Name (eg, city) []:BLR
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Example
Organizational Unit Name (eg, section) []:DBA-Team
Common Name (e.g. server FQDN or YOUR name) []:mongod1.example.com
Email Address []:mmalai@example.com

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
```

After generating the private key and CSR for each node. Now we need to sign all the all node CSR with CA Private key.

Signing CSR using CA Private & Public Key

Command

```
openssl x509 -req -in mongod1.csr -CA example-ca-pub.crt -CAkey example-ca.key -
out mongod1.crt
openssl x509 -req -in mongod2.csr -CA example-ca-pub.crt -CAkey example-ca.key -
out mongod2.crt
openssl x509 -req -in mongod3.csr -CA example-ca-pub.crt -CAkey example-ca.key -
out mongod3.crt
```

```
root@CA-server:~# openssl x509 -req -in mongod1.csr -CA example-ca-pub.crt -CAkey
example-ca.key -out mongod1.crt
Signature ok
subject=C = IN, ST = Karnataka, L = BLR, O = Example, OU = DBA-Team, CN =
mongod1.example.com, emailAddress = mmalai@example.com
Getting CA Private Key
Enter pass phrase for example-ca.key:
```

After sign each CSR, we will get one public key corresponding to that CSR. Then we need to concatenate respective replica member node private and public certificate to create a pem format file.

```
root@CA-server:~# cat mongod1.key mongodb1.crt > mongod1.pem
root@CA-server:~# cat mongod2.key mongod2.crt > mongod2.pem
root@CA-server:~# cat mongod3.key mongod3.crt > mongod3.pem
```

Once all the above step is completed, now we want to securely transport the respective node pem file and example-ca-pub.crt to respective replica member.

Create a self-signed certificate for mongo client / Driver

Here App server and DBA Team are clients for the Replica set, now we need to create a certificate for them.

Important things Remember for Client Certificate:

- Client certificates need to be signed by the same CA who signed all replica member certificate.
- Common name filed can enter wildcard name (*.example.com) or we can use the SAN field which allows multiple name entries on a single certificate.
- we need to specify a different value for O or OU filed compare to the replica set member certificate.

This one certificate helps to connect all the replica set members, so the common name must be a Wildcard name (*.example.com) or need to specify individual FQDN of each member of the replica set in Subject Alternative Name(SAN) field.

Create an example.conf file with the following content.

```
[req]
distinguished_name = req_distinguished_name
req_extensions = v3_req
default_keyfile = example-client.key
prompt = no

[req_distinguished_name]
C = IN
ST = Karnataka
L = BLR
O = Example
OU = App-Team
CN = mongod1.example.com

[v3_req]
keyUsage = keyEncipherment, dataEncipherment
extendedKeyUsage = serverAuth
subjectAltName = @alt_names

[alt_names]
DNS.1 = mongod1.example.com
DNS.2 = mongod2.example.com
DNS.3 = mongod3.example.com
```

Once example.conf file has been created, create CSR file and private key to be used with a certificate with the following command:

```
openssl req -new -nodes -out example-client.csr -config example.conf
```

Sign the client CSR using CA public and private key

```
openssl x509 -req -in example-client.csr -CA example-ca-pub.crt -CAkey example-
ca.key -out example-client.crt
```

Finally, you concatenate the key and the signed certificate.

```
cat example-client.key example-client.crt > example-client.pem
```

We finished all certificate creation from client and replica set members. now it’s the time to configure the all replica set member mongod.conf

Configure a Replica-Set to Support SSL

We need to instruct mongod about the certificates to enable the encryption.

Change the configuration file **/etc/mongod.conf** on each host adding the following rows:

```
# network interfaces
net:
port: 27017
bindIp: mongod1.example.com
ssl:
mode: requireSSL
PEMKeyFile: /etc/mongod/ssl/mongod1.pem
CAFile: /etc/mongod/ssl/example-ca-pub.crt
```

Restart the daemon

```
systemctl restart mongod
```

Now we are going to connect using mongo Client:

```
mongo --ssl --sslCAFile example-ca-pub.crt --host
rs0/mongod1.example.com:27017,mongod1.example.com:27017,mongod1.example.com:27017
--sslPEMKeyFile example-client.pem
```

Configure a Replica-Set to Support TLS

TLS Mode support from MongoDB 4.2.

We need to instruct mongod about the certificates to enable the encryption.

Change the configuration file **/etc/mongod.conf** on each host adding the following rows:

```
# network interfaces
net:
port: 27017
bindIp: mongod1.example.com
ssl:
mode: requireTLS
certificateKeyFil: /etc/mongod/ssl/mongod1.pem
CAFile: /etc/mongod/ssl/example-ca-pub.crt
```

Restart the daemon

```
systemctl restart mongod
```

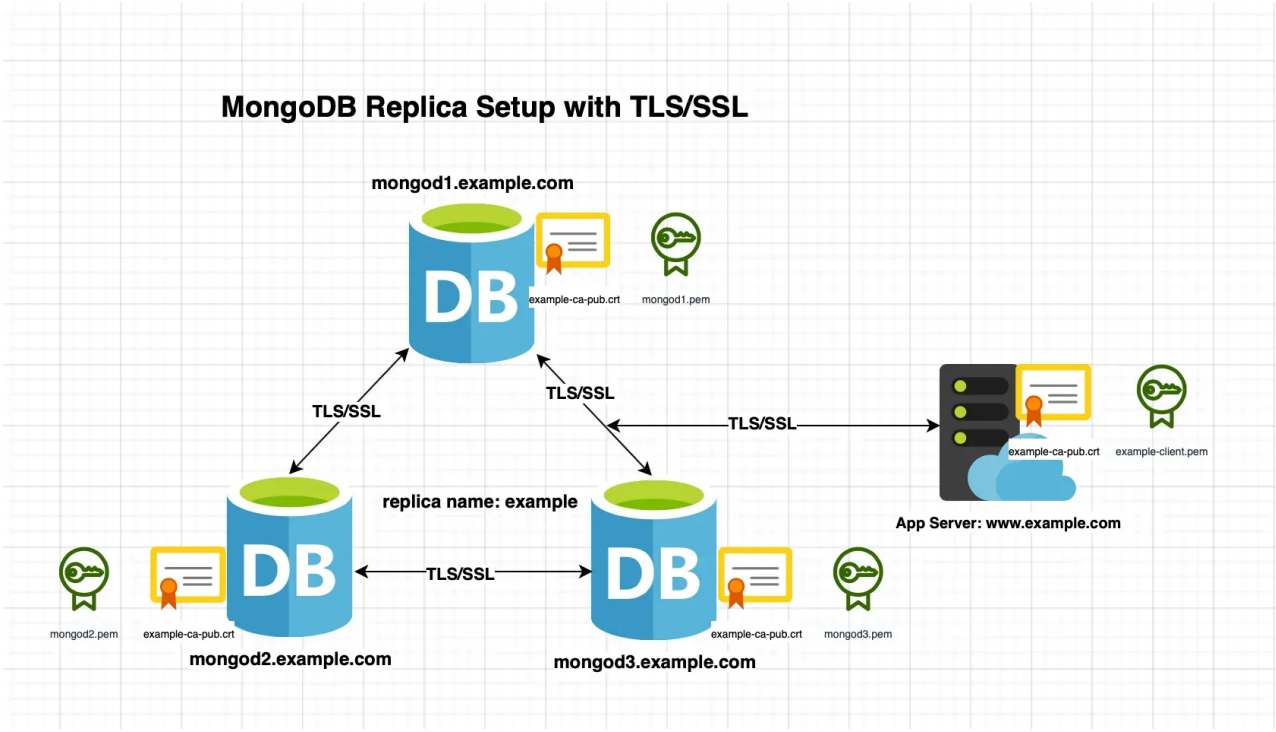
Make sure to put the proper file names on each host (mongod1.pem on mongod1 host and so on).

Now, as long as we have made no mistakes, we have a properly configured replica set that is using encrypted connections.

Now we are going to connect using mongo Client:

```
mongo --tls --tlsCAFile example-ca-pub.crt --host
rs0/mongod1.example.com:27017,mongod1.example.com:27017,mongod1.example.com:27011
--tlsCertificateKeyFile example-client.pem
```

Whatever mode TLS or SSL we are using, Make sure to put the proper file names on each host (mongod1.pem on mongod1.example.com host and so on)



Now, as long as we have made no mistakes, we have properly configured replica set that is using encrypted connections.

Everyone aware that certificate provides two things

1. Identity of host

2. Encryption.

But when we use a self-signed certificate it provides encryption but fails to provide the identity of the host. So always recommend our client to use CA-signed certificates in production.

Like you to enhance the security of your MongoDB servers our MongoDB engineers at Mydbops can provide the best services for you.

security MonogoDB