**TSTP for Authentication for Product Management and Maintenance interfaces**

**(2.1.1 CSR)**

<DUT Details: > 5G core NF UDR

<DUT Software Version>

1. Ubuntu 22.04.4 LTS (Jammy Jellyfish)
2. Open Software Alliance/oai-udr v2.0.1

<Digest Hash of OS>

1bf0e470c9bea818ddf7c73e83a06a70c8e3f1d2b03b51392ab2966829d5b00a (Ubuntu)

<Digest Hash of Configuration>

f8595cac360c09228f494fe75eb5a839641b7ef2cb9ca468ca70874fd80baede (oai-udr)

<Applicable ITSAR:> Unified Data Repository of 5G (UDR) of 5G

<ITSAR Number and Version No.> ITSAR111112311 V1.0.0

<OEM supplied Document List:> List of documents as mentioned in preconditions of Sr. no. 6 below.

1. <**ITSAR Section Number and Name**> Section 1: Access and Authorization

1. <**Security Requirement No & Name**> 2.1.1 Authentication for Product Management and Maintenance interfaces
2. **<Requirement Description:>**

UDR shall support mutual authentication of entities on management interfaces, the authentication mechanism can rely on the management protocols used for the interface or

other means.

Secure cryptographic controls prescribed in Table 1 of the latest document “Indian Telecom

Security Assurance Requirements (ITSAR) for Cryptographic Controls” shall only be used for UDR management and maintenance”.

[Ref [3]: TSDSI STD T1.3GPP 33.117-17.1.0 V1.1.0. Section 4.2.3.4.4.1]

1. **DUT Confirmation Details:**
2. Note down the DUT details as provided by the vendor (Present DUT).
3. DUT name: HP Pavilion All-in-One - 24-q274in
4. Model Number: Product: Z8G10AA
5. Hardware Model No: 24-q274in
6. Hardware Serial Number: Serial: 8CC70315BK

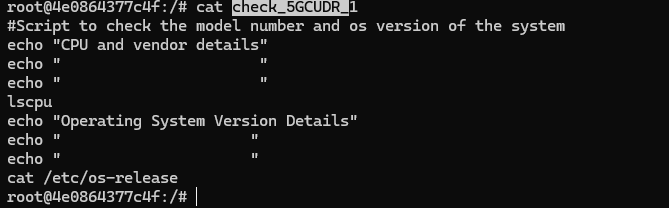
(Refer to the Fig. 15 for verification of DUT name, Hardware Model Number, Hardware Serial number and product number)

1. Use the command line interface of the device to be tested.

Execute the shell script check5gcudr\_1 to get the hardware details (Processor information, Processor Model number, Vendor details, Cache information) and the OS related information.

**Execute the script:**

$ **cat check\_5gcudr\_1**



**Fig. 1 Script to display CPU and OS details**

The **lscpu** command in the above script will display all the details related to:

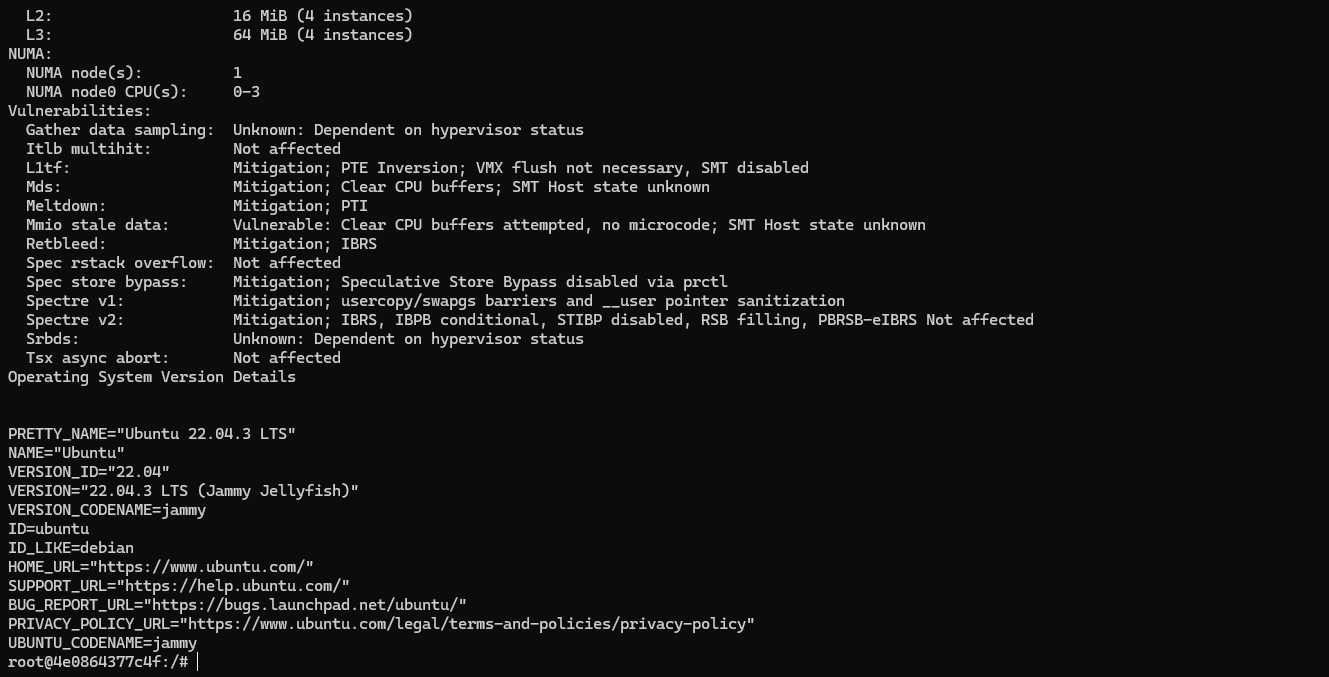
1. CPU
2. Architecture
3. Vendor ID
4. Model Name
5. Virtualization Features and cache details.

The **cat /etc/os-release** command in the script will display:

1. OS name
2. version number and
3. version ID details

.

**Fig. 2 Output screen displaying the hardware-CPU details**



**Fig. 3 Output screen displaying the CPU and OS details**

**If the UDR NF is containerized in a docker, then:**

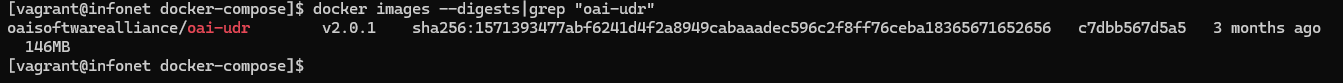
1. Obtain the license and software version details as well as the hash value of the UDR NF by executing the command **$docker images –digests |grep “oai-udr”.**

**docker pull oaisoftwarealliance/oai-udr**

**docker run -d --name oai-udr -p 8080:8080 oaisoftwarealliance/oai-udr**

**docker ps**

Output Display:



**Fig. 4 Command and Output screen displaying the UDR NF details**

1. Obtain the hash of the UDR configuration file by executing the command **$sha256 udr.conf.**

**root@infonet:/home/infonet# find / -name udr.conf 2>/dev/null**

**/var/lib/docker/overlay2/d58c7020d8b70cc627c9d40f5b19c9e3743b8b0f5b19b1cae033efcef60b7253/merged/openair-udr/etc/udr.conf**

**/var/lib/docker/overlay2/b29da6644030fb90ef7b3f68a3c5fa582d34e38b30e8f81b65bb2bcd0acdf035/diff/openair-udr/etc/udr.conf**

**root@infonet:/home/infonet# docker exec -it oai-udr sha256sum /openair-udr/etc/udr.conf**

**f8595cac360c09228f494fe75eb5a839641b7ef2cb9ca468ca70874fd80baede /openair-udr/etc/udr.conf**

**Or We can open it from root( Copy the File to the Host and Hash It)**

**docker cp oai-udr:/openair-udr/etc/udr.conf ./udr.conf**

**sha256sum udr.conf**

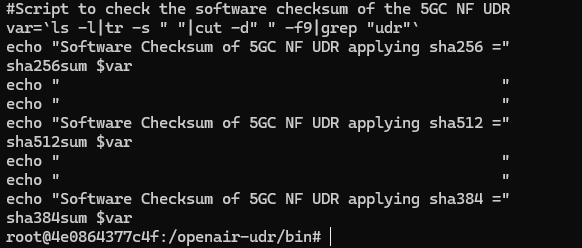
Output Display:



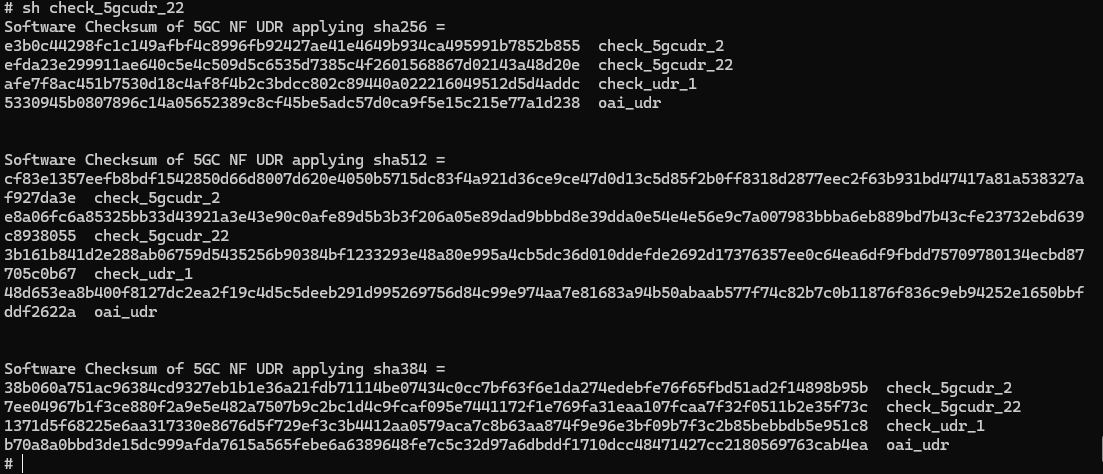
**Fig. 5 Command and Output screen displaying the software checksum of the UDR configuration file.**

1. Obtain all possible software checksums of the UDR NF by executing the script check\_5gcudr\_2.

**$sh check\_5gcudr\_2**



**Fig. 6 Script to apply software checksums of NF UDR**



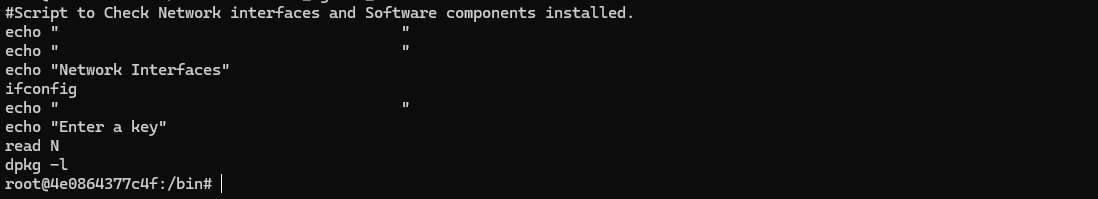
**Fig. 7 Output screen displaying the software checksum of UDR NF.**

1. **DUT Configuration:**

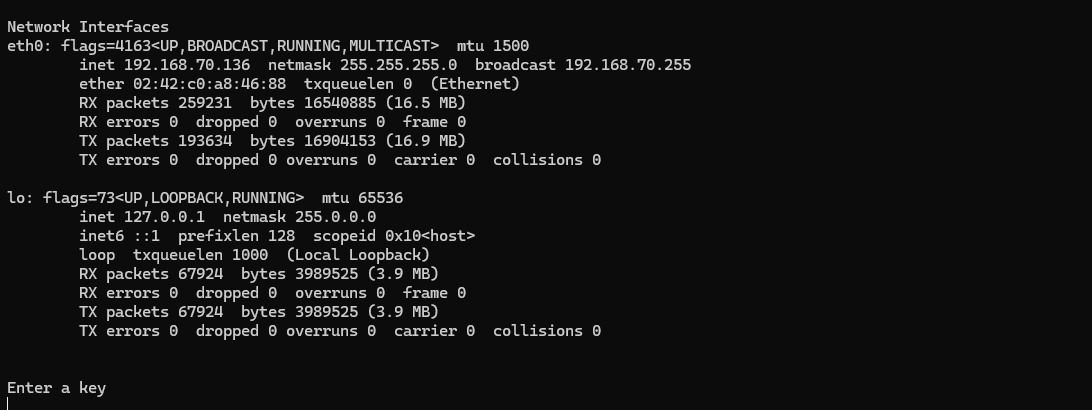
To obtain the Device under Test configuration and installed utilities related information, execute the script check\_5gcudr\_4.

The script uses the commands **ifconfig, dpkg -l** and **cat /etc/udr.conf** to list all the network interfaces along with IP addresses, list all software packages involved and the UDR NF configuration.

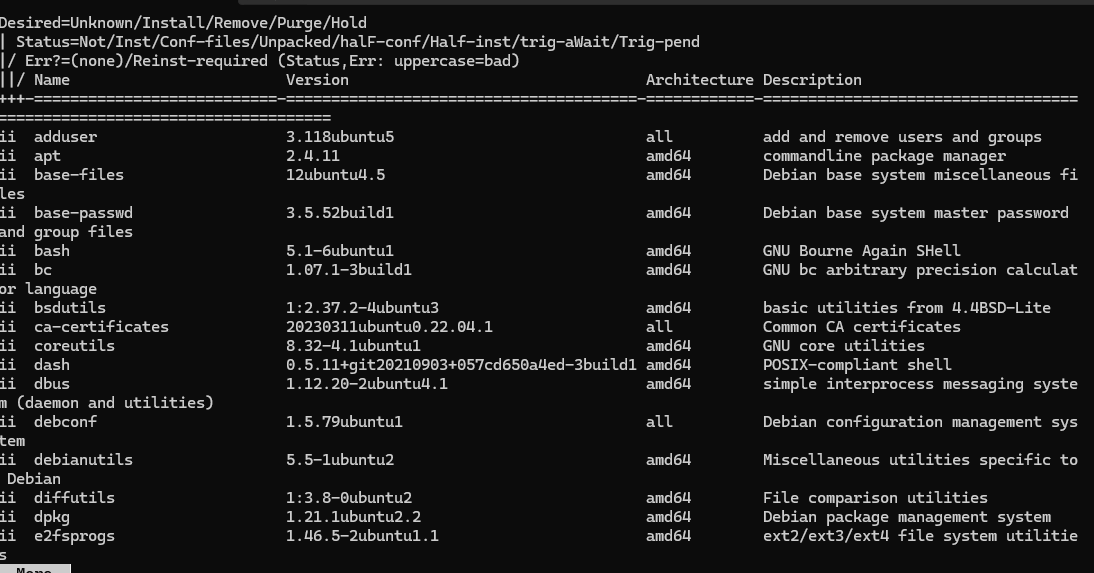
**$sh check\_5gcudr\_4**

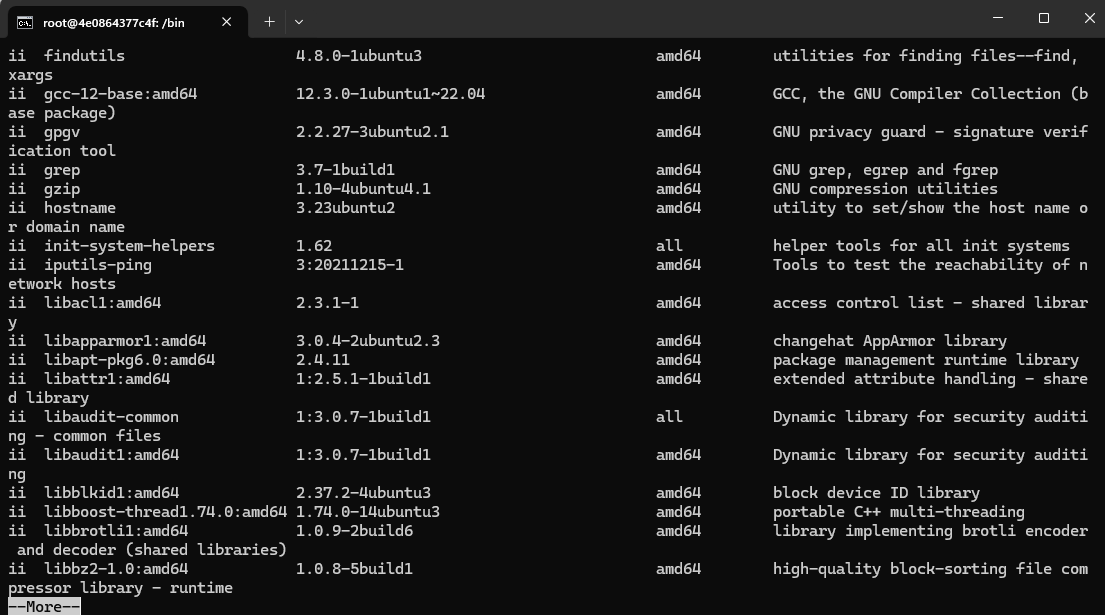


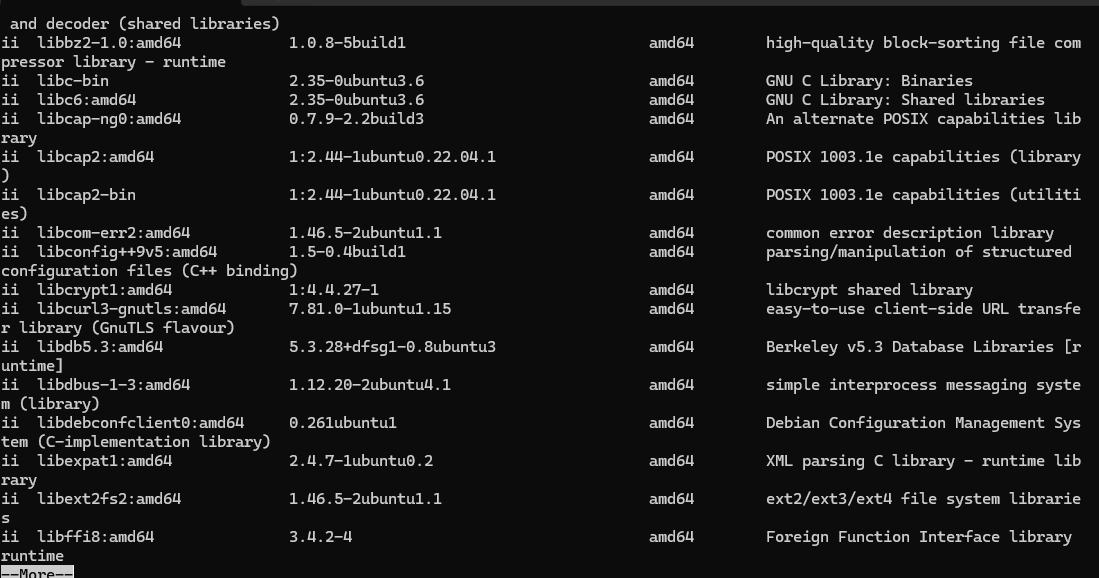
**Fig. 8 Script to display all network interfaces and software packages installed**

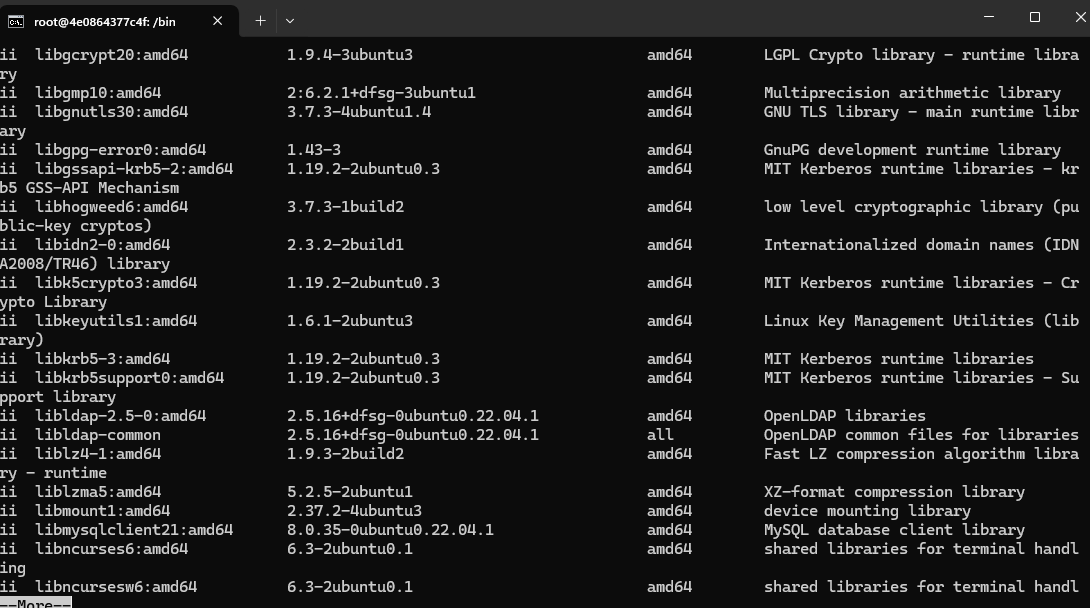
****

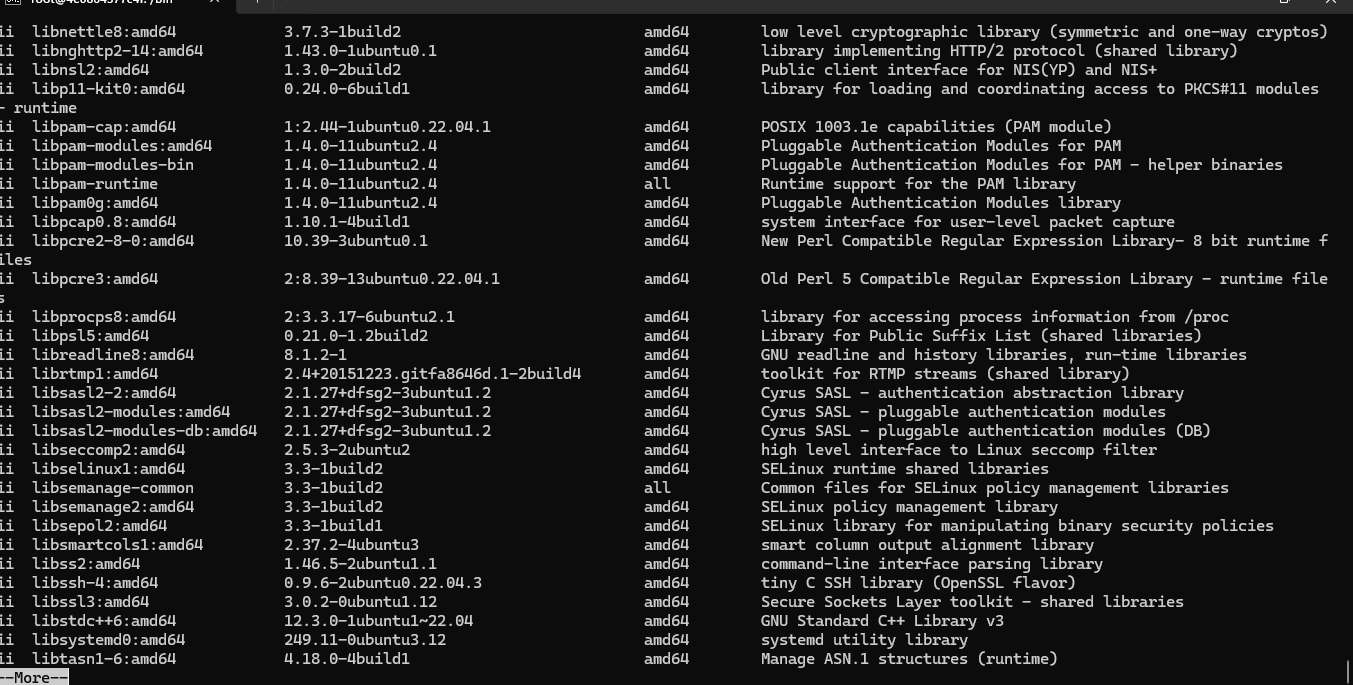
**Fig. 9 Output screen displaying the Network Interfaces**

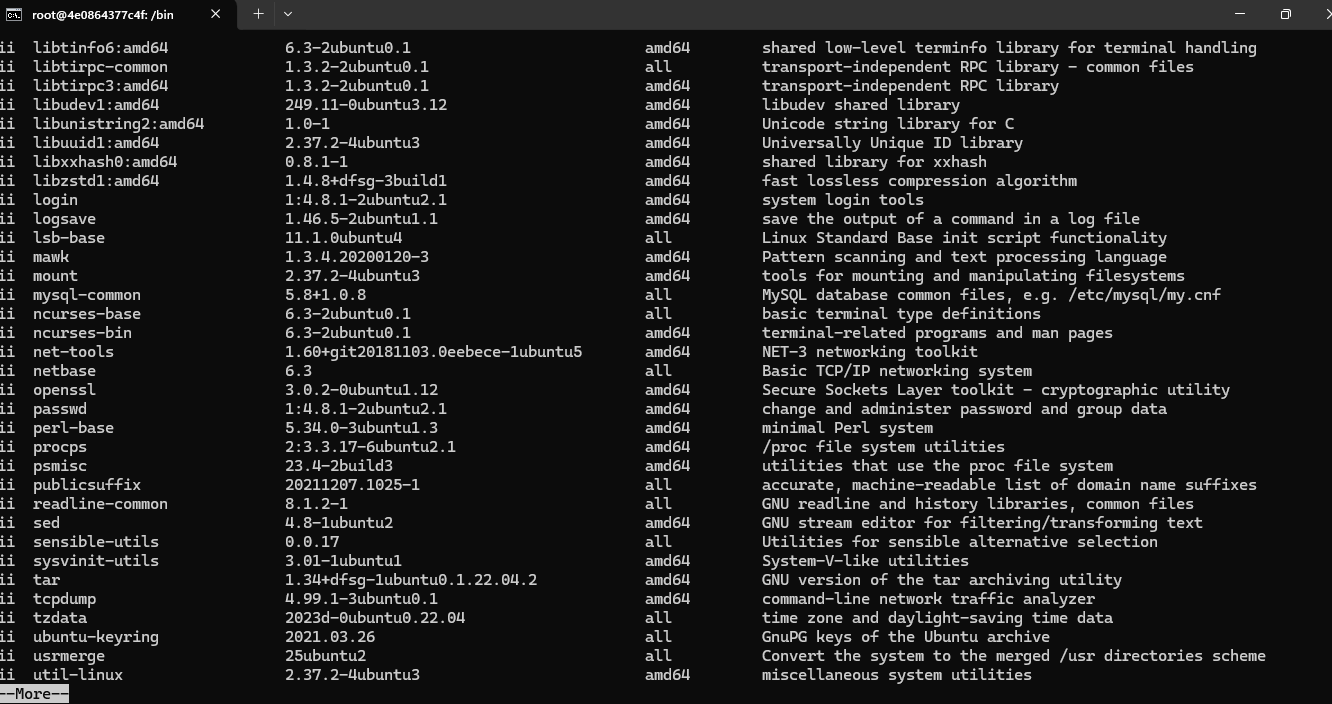


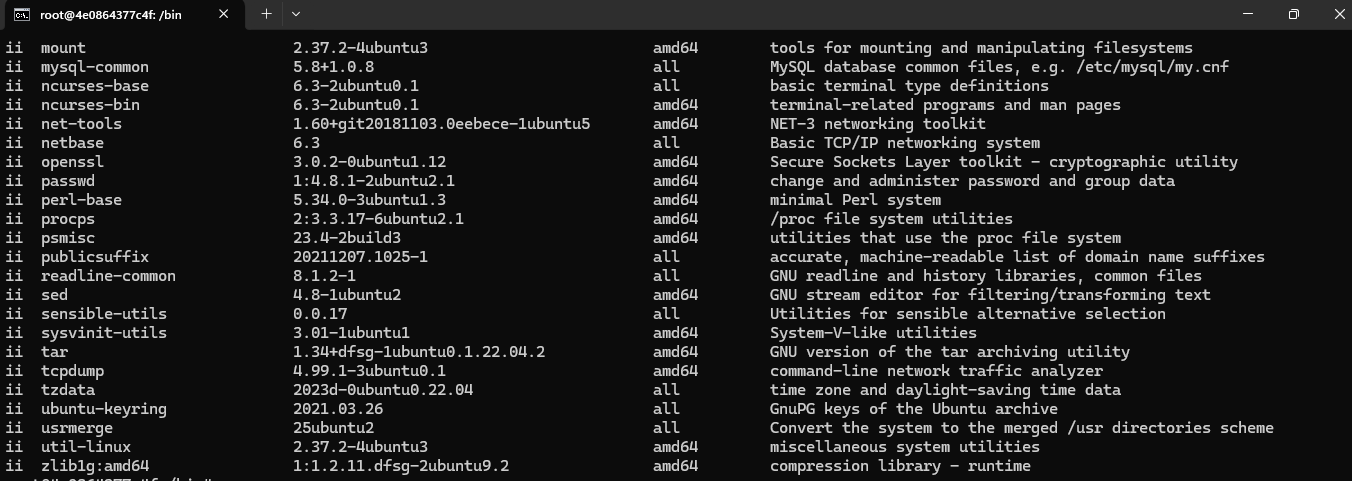












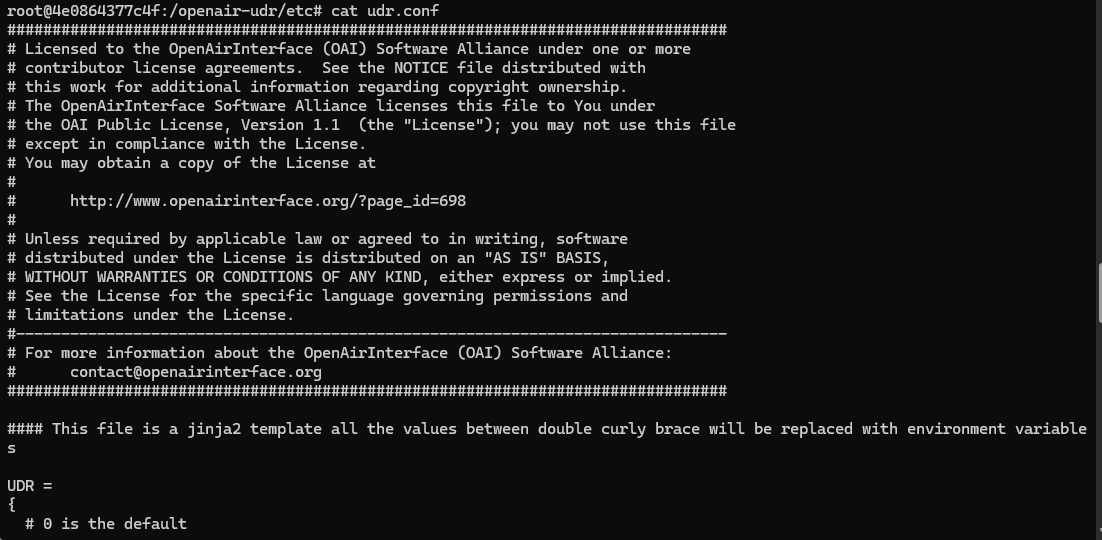
**Fig. 10 Output screen displaying the software packages installed**

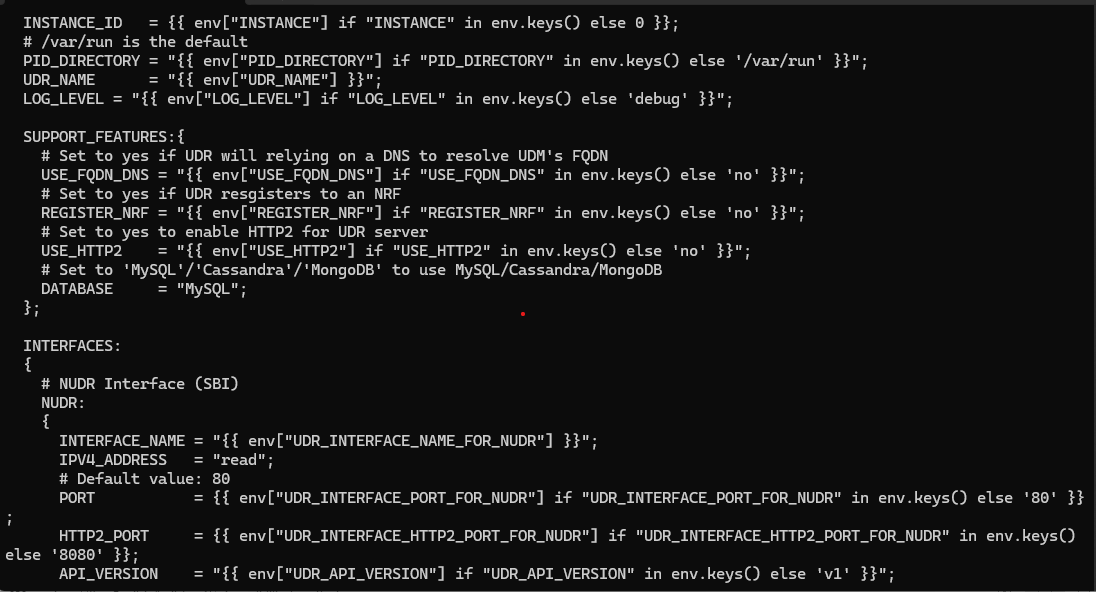
Since different Linux distributions use different package managers, the appropriate packet manager can be replaced in the script.

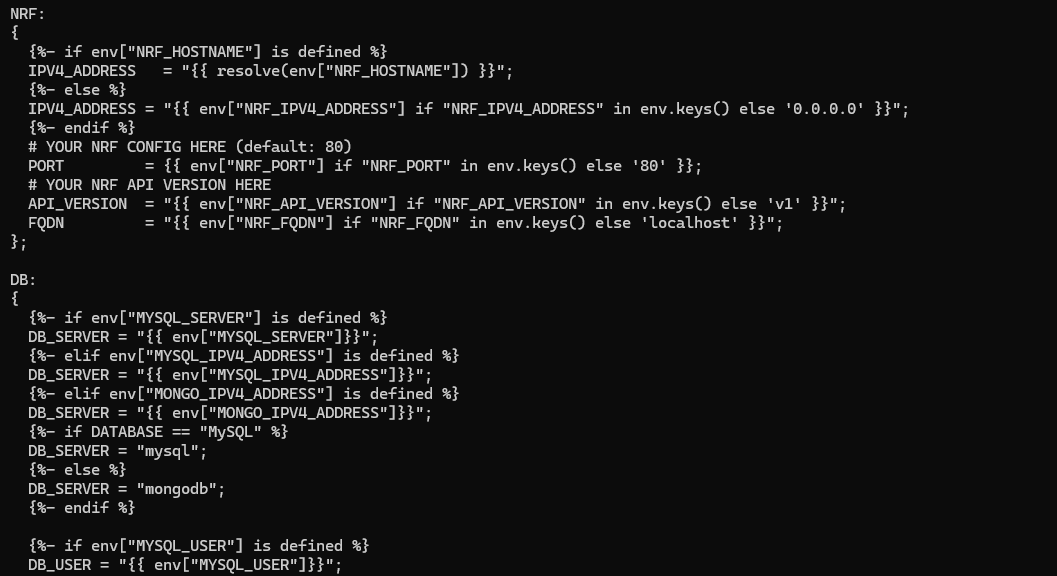
|  |  |
| --- | --- |
| **Linux Distribution** | **Package Manager** |
| Debian, Ubuntu | dpkg |
| Red Hat Enterprise Linux, CentOS, Fedora | yum |
| Arch Linux and derivatives | pacman |
| SUSE Linux Enterprise, openSUSE | zypper |
| Gentoo Linux | emerge |
| Alpine Linux | apk |
| Slackware Linux | pkgtool |

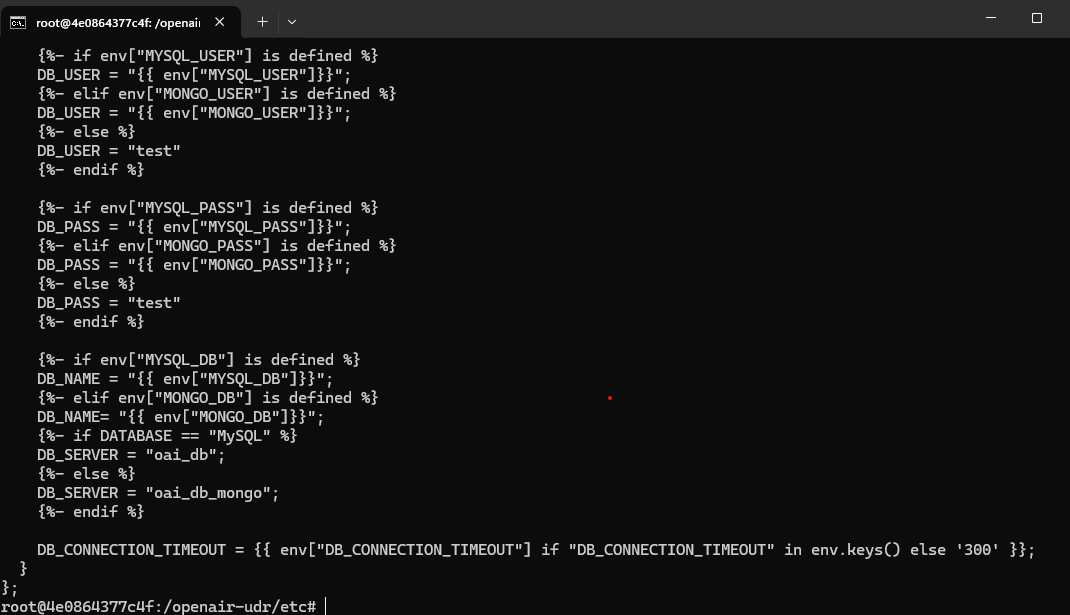
Additionally, the Tester can execute the command

**$cat /etc/udr.conf**









**Fig. 11 Output screen displaying the udr.conf file of the /etc directory**

1. **Preconditions:**
2. The vendor provides documents that list each of the management protocols and describes the authentication mechanism used for each database of UDR NF.
3. The vendor provides documents of all databases of UDR NF that use management protocols and the authentication mechanism used for each one follows secure cryptographic controls prescribed in Table1 of the latest document “Indian Telecom Security Assurance Requirements (ITSAR) for Cryptographic Controls” shall only be used for UDR management and maintenance”.
4. **Test Objective/Purpose:**

Verify that there is mutual authentication of entities for management interfaces on the UDR NF.

**Procedure and execution steps:**

1. **Test Plan**

**8.1 Test Case Scenarios:**

8.1.1 Test Scenario to check the UDR NF shall support mutual authentication of entities on

management interfaces, the authentication mechanism can rely on the management

protocols used for the interface or other means.

Secure cryptographic controls prescribed in Table 1 of the latest document “Indian

Telecom Security Assurance Requirements (ITSAR) for Cryptographic Controls” shall

only be used for UDR management and maintenance”. Use CLI commands/shell

scripts to check for the above requirement.

* + 1. Test Scenario for checking the UDR NF shall support mutual authentication of

entities on management interfaces, the authentication mechanism can rely on the

management protocols used for the interface or other means.

Secure cryptographic controls prescribed in Table 1 of the latest document “Indian

Telecom Security Assurance Requirements (ITSAR) for Cryptographic Controls” shall

only be used for UDR management and maintenance” through database workbench

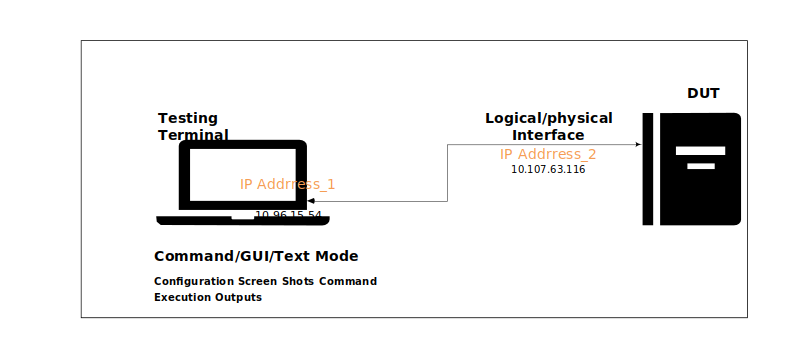
(Database specific monitoring tools). Execute Database workbench and connect to the

database IP and Port to get Database check (Every database will have its own

workbench, e.g.,MySQL has ‘MYSQL Workbench’, Postgres has ‘pgAdmin’, but not

all GUI’s of the databases are available publicly).

* 1. **Test Bed Diagram:**



**Fig. 12 Test bed diagram**

* 1. **Tools Required:**

1. Command Line Interface
2. GUI (Database Server Workbench)
   1. **Test Execution Steps:**

The accredited evaluator's test lab is required to execute the following steps:

1. The tester checks that the authentication mechanisms have been configured on the UDR NF.

2. The tester triggers communication between the UDR NF and a test entity that has a legitimate authentication credential.

3. Then, the tester triggers communication between the UDR NF and a test entity that doesn't have a legitimate authentication credential.

1. **Expected Results:**

1) Mutual authentication is successful and communication between UDR NF and the entity with correct credentials can be established.

2) Mutual authentication fails and communication between the UDR NF and the entity with

incorrect credentials cannot be established.

1. **Expected Format of Evidence:**

A testing report provided by the testing agency which will consist of the following

information:

Evidence suitable for the interface, e.g., screenshot containing the operation results, test result

pass/fail recorded by tester.

**(Positive case Evidence shall be provided in the form of screenshots as in Section 11 follow the 3GPP TS 33.117 version 18.3.0 Release 18).**

1. **Test Execution**

**Test Case Number: 01**

1. **Test Case Name:** Test Name: TC\_MUTUAL\_AUTHENTICATION-ON\_NETWORK\_PRODUCT\_MANAGEMENT\_PROTOCOLS
2. **Test Case Description:**  Test needs to be conducted full-fills the requirement to mutual authentication of entities for management interfaces on the UDR NF.
3. **Execution Steps:**

Check if the documentation provided by the vendor(s), accompanying the DUT, includes:

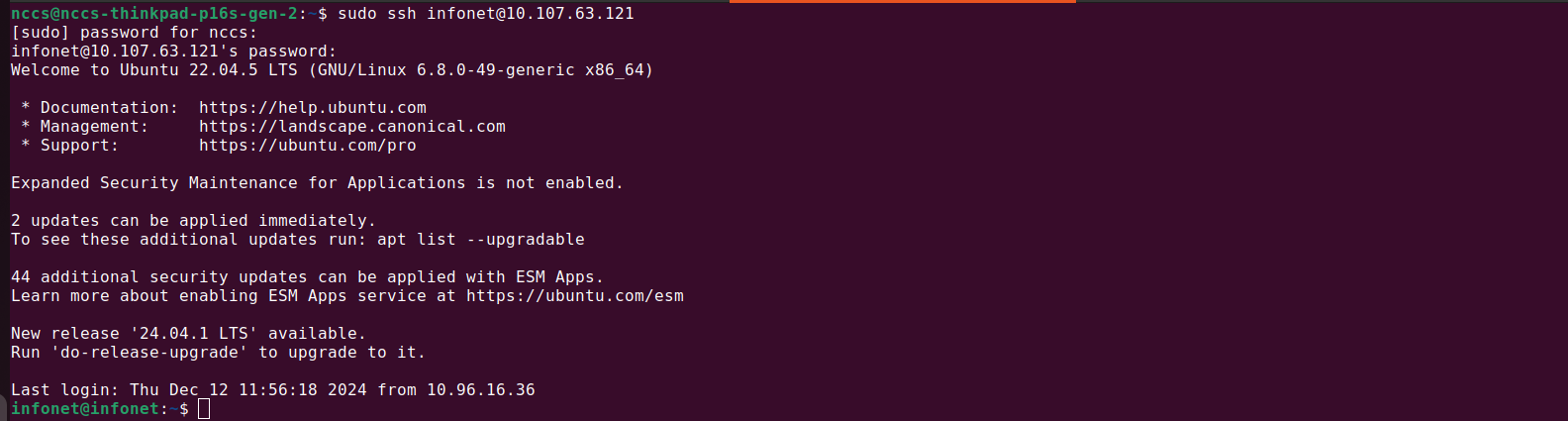
* + - 1. The list of databases used management protocols and the authentication mechanism used for each one.
      2. The vendor declares that all databases used management protocol and the authentication mechanism follows secure cryptographic controls prescribed in Table1 of the latest document “Indian Telecom Security Assurance Requirements (ITSAR) for Cryptographic Controls” shall only be used for UDR management and maintenance”.

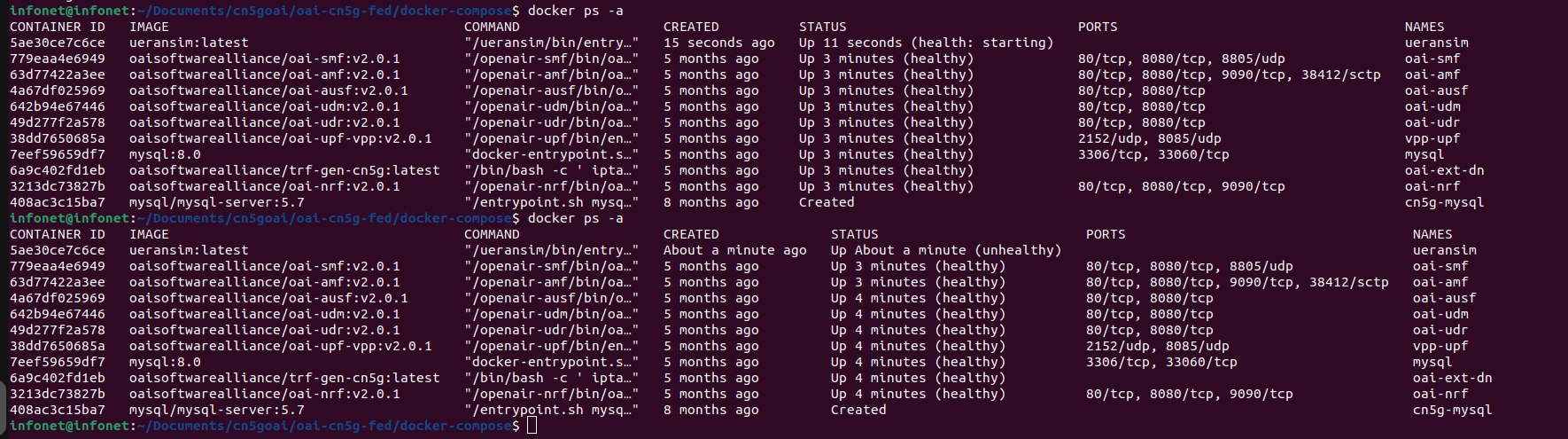
1. The tester logs in to database server:

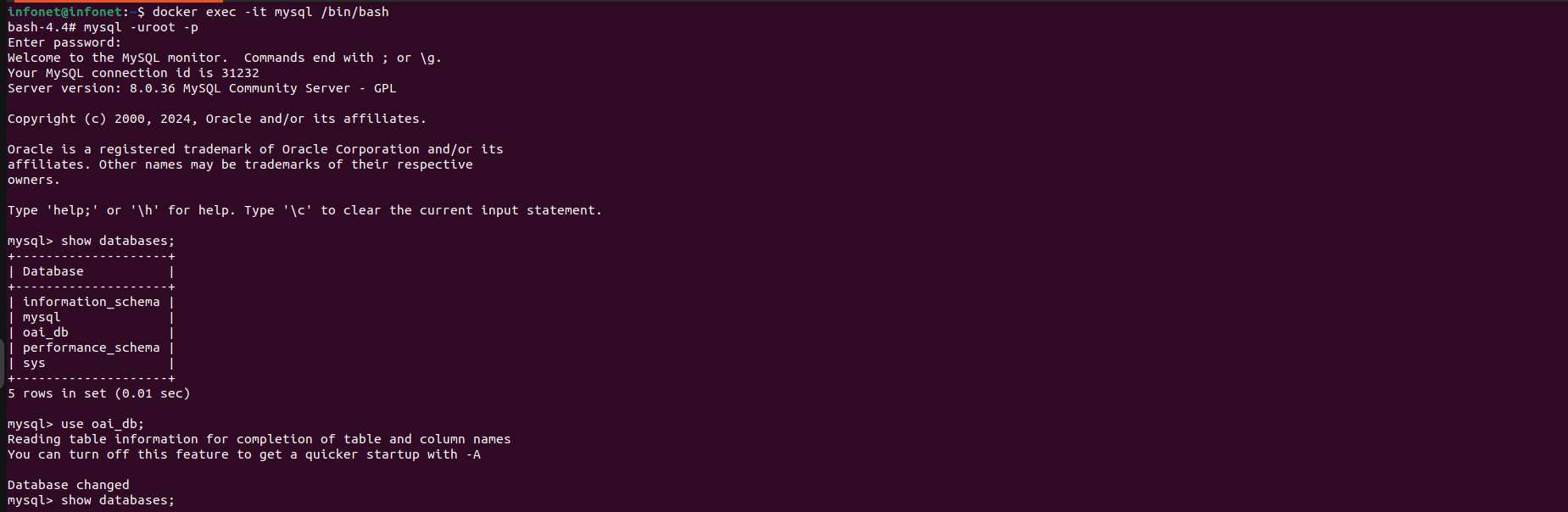
**Execute the commands: ‘sudo mysql -u root -p’**

Mysql> Show Databases;

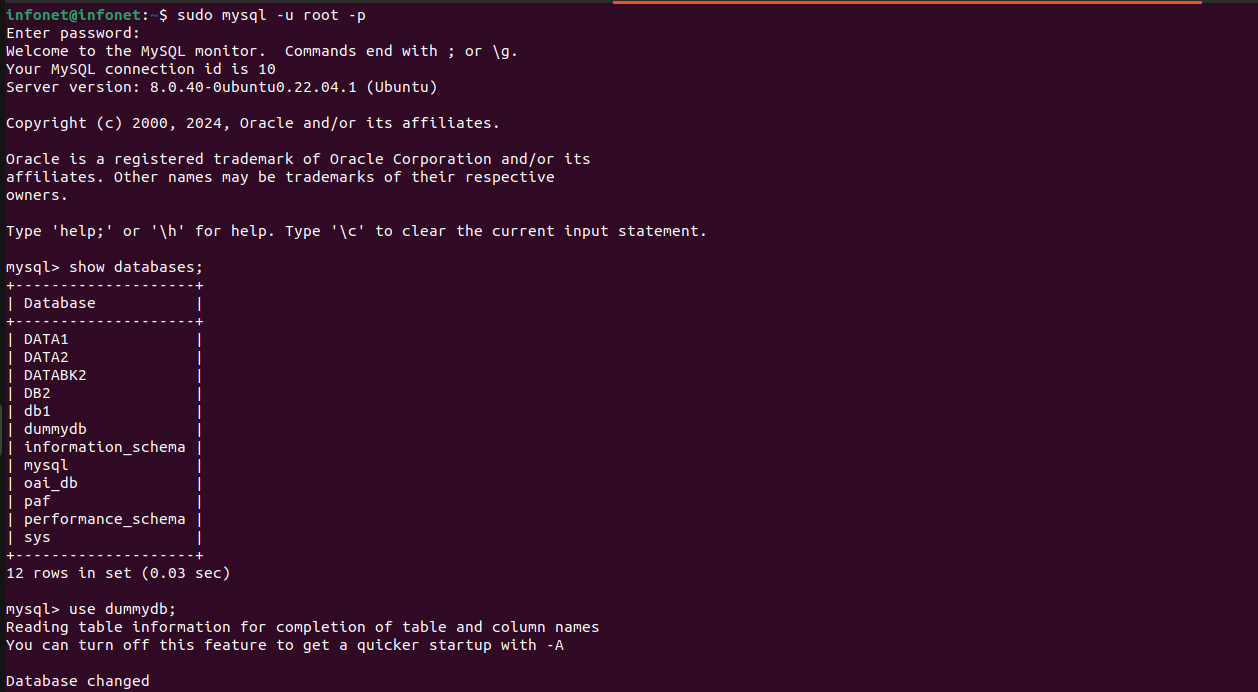
Mysql>Use dummydb;







**Fig. 13 The screen output displays the ssh login of the server, use ‘docker ps-a’ to check all container health, use command ‘docker exec -it mysql /bin/bash’ login mysql database and the databases of UDR NF.**



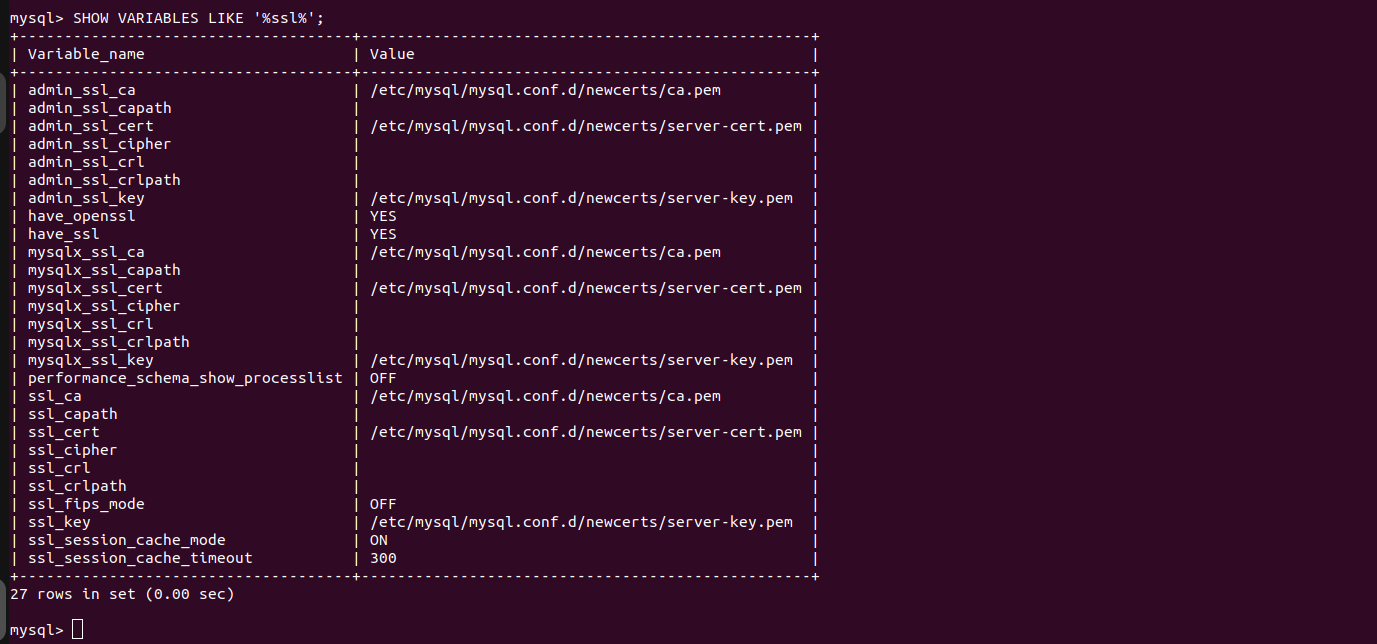
**Fig. 14 The screen output displays the databases of UDR NF.**

1. To verify authentication mechanisms have been configured on the UDR NF

Look for the **have\_ssl** and **have\_openssl** values:

* 1. If they are set to YES, SSL is already enabled.
  2. If they are set to DISABLED, SSL is either not configured or the UDR database installation lacks SSL support.

**Execute the commands: SHOW VARIABLES LIKE '%ssl%';**



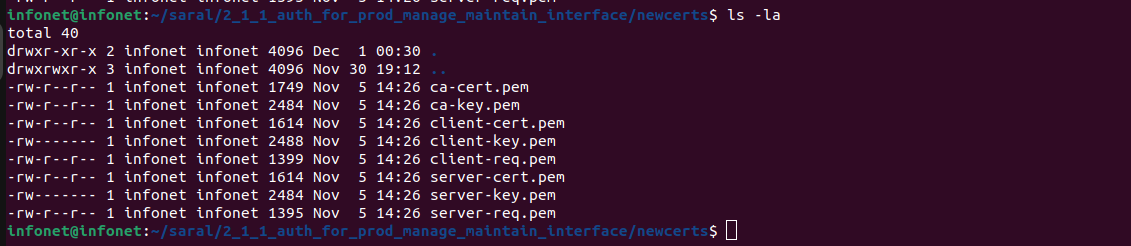
**Fig. 15 The screen output displays have\_openssl and have\_ssl value ‘set YES’ and ssl\_ca, ssl\_cert and ssl\_key value configuration of UDR NF database server.**

1. **Execute the commands: ‘ls -la’ to** verify access right of certificate and key on server.

**Mutual Authentication on UDR Database using SSL protocol**

Explanation:

1. **SSL Certificates explanation**:
   * **CA Certificate**: Generated as the root Certificate Authority (CA) for mutual authentication.
   * **Server Certificate**: The server uses this certificate to prove its identity to clients.
   * **Client Certificate**: The client uses this certificate to prove its identity to the server.





## **1. Create a Secure Directory for Certificates**

bash

CopyEdit

sudo mkdir -p /etc/mysql/newcerts

sudo chmod 700 /etc/mysql/newcerts

cd /etc/mysql/newcerts

This directory will hold your CA, server, and client certificates/keys.

## 2. Generate a CA (Certificate Authority) Key and Certificate

1. Generate the CA key (RSA-3072 bits)

bash

CopyEdit

sudo openssl genrsa -out ca-key.pem 3072

1. Create a self-signed CA certificate using SHA-256

bash

CopyEdit

sudo openssl req -x509 -new -nodes -sha256 \

-key ca-key.pem \

-days 365 \

-out ca-cert.pem \

-subj "/C=IN/ST=YourState/L=YourCity/O=YourOrg/OU=IT/CN=MyRootCA"

* + -days 365: Certificate validity (1 year). Adjust as needed.
  + -subj: Provides a non-interactive way to specify certificate details.
  + -sha256: Complies with ITSAR recommended hashing (at least SHA-256).
  + 3072-bit RSA: Complies with ITSAR recommended key length.

## 3. Generate the Server Key and Certificate

1. Generate the server key (RSA-3072 bits)

bash

CopyEdit

sudo openssl genrsa -out server-key.pem 3072

1. Create a Certificate Signing Request (CSR)

bash

CopyEdit

sudo openssl req -new -sha256 \

-key server-key.pem \

-out server-req.pem \

-subj "/C=IN/ST=YourState/L=YourCity/O=YourOrg/OU=IT/CN=server.local"

1. Sign the Server CSR with the CA

bash

CopyEdit

sudo openssl x509 -req -in server-req.pem -sha256 \

-CA ca-cert.pem -CAkey ca-key.pem -CAcreateserial \

-out server-cert.pem -days 365

This produces server-cert.pem signed by ca-cert.pem.

1. Secure Permissions

bash

CopyEdit

sudo chown mysql:mysql server-key.pem server-cert.pem ca-cert.pem

sudo chmod 600 server-key.pem

## 4. Generate the Client Key and Certificate (for Mutual TLS)

1. Generate the client key (RSA-3072 bits)

bash

CopyEdit

sudo openssl genrsa -out client-key.pem 3072

1. Create a Client CSR

bash

CopyEdit

sudo openssl req -new -sha256 \

-key client-key.pem \

-out client-req.pem \

-subj "/C=IN/ST=YourState/L=YourCity/O=YourOrg/OU=IT/CN=client.local"

1. Sign the Client CSR with the CA

bash

CopyEdit

sudo openssl x509 -req -in client-req.pem -sha256 \

-CA ca-cert.pem -CAkey ca-key.pem -CAcreateserial \

-out client-cert.pem -days 365

1. Secure Permissions

bash

CopyEdit

sudo chown mysql:mysql client-key.pem client-cert.pem

sudo chmod 600 client-key.pem

After these steps, you should have:

* ca-cert.pem (CA certificate)
* ca-key.pem (CA private key)
* server-cert.pem (server certificate)
* server-key.pem (server private key)
* client-cert.pem (client certificate)
* client-key.pem (client private key)

## 5. Configure MySQL to Use SSL/TLS

Edit MySQL’s configuration file, typically at /etc/mysql/mysql.conf.d/mysqld.cnf (or /etc/my.cnf on some distributions). Under the [mysqld] section, add:

ini

CopyEdit

[mysqld]

ssl-ca = /etc/mysql/newcerts/ca-cert.pem

ssl-cert = /etc/mysql/newcerts/server-cert.pem

ssl-key = /etc/mysql/newcerts/server-key.pem

# Optional: If your MySQL version supports admin SSL variables:

# admin\_ssl\_ca = /etc/mysql/newcerts/ca-cert.pem

# admin\_ssl\_cert = /etc/mysql/newcerts/server-cert.pem

# admin\_ssl\_key = /etc/mysql/newcerts/server-key.pem

Then restart MySQL:

bash

CopyEdit

sudo systemctl restart mysql

or

bash

CopyEdit

sudo service mysql restart

## 6. Verify the SSL/TLS Configuration

Log into MySQL and run:

sql

CopyEdit

SHOW VARIABLES LIKE '%ssl%';

You should see:

* have\_ssl and have\_openssl = YES
* ssl\_ca, ssl\_cert, ssl\_key pointing to /etc/mysql/newcerts/...

Check the directory permissions:

bash

CopyEdit

sudo ls -ld /etc/mysql/newcerts

You should see something like:

bash

CopyEdit

drwxr-xr-x 2 mysql mysql 4096 Feb 27 11:49 /etc/mysql/newcerts

If the permissions aren’t set to at least 755, then the mysql user might not be able to read the directory.

1. Change the owner of the directory to mysql (if not already):

bash

CopyEdit

sudo chown mysql:mysql /etc/mysql/newcerts

1. Set the directory permissions so that it’s world-readable (755):

bash

CopyEdit

sudo chmod 755 /etc/mysql/newcerts

1. Verify the changes:

bash

CopyEdit

sudo ls -ld /etc/mysql/newcerts

The output should now show permissions like drwxr-xr-x with owner and group set to mysql.

After these changes, MySQL should be able to read the certificate files in that directory. Restart MySQL afterward:

bash

CopyEdit

sudo systemctl restart mysql

Then check your SSL variables in MySQL again:

sql

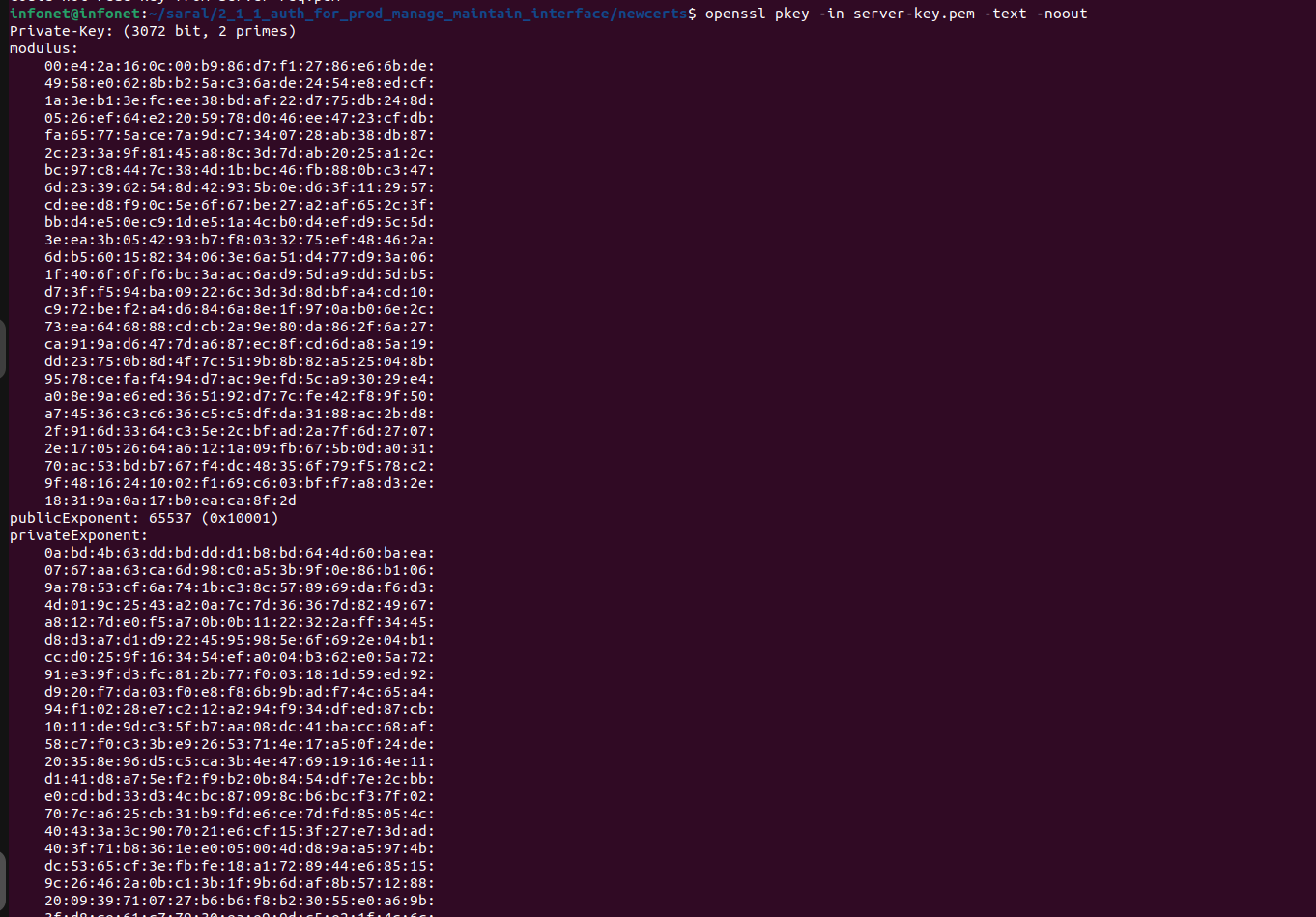
CopyEdit

SHOW VARIABLES LIKE '%ssl%';

**Fig. 16 The screen output displays CA certificate and key for server, client and their access rights in database server. For executing commands “SHOW VARIABLES LIKES ‘tls\_version’ to get TLS version details TLS\_version TLSv1.2, TLSv1.3 in the database.**

1. To verify key exchange algorithm RSA-3072 bit on the UDR NF

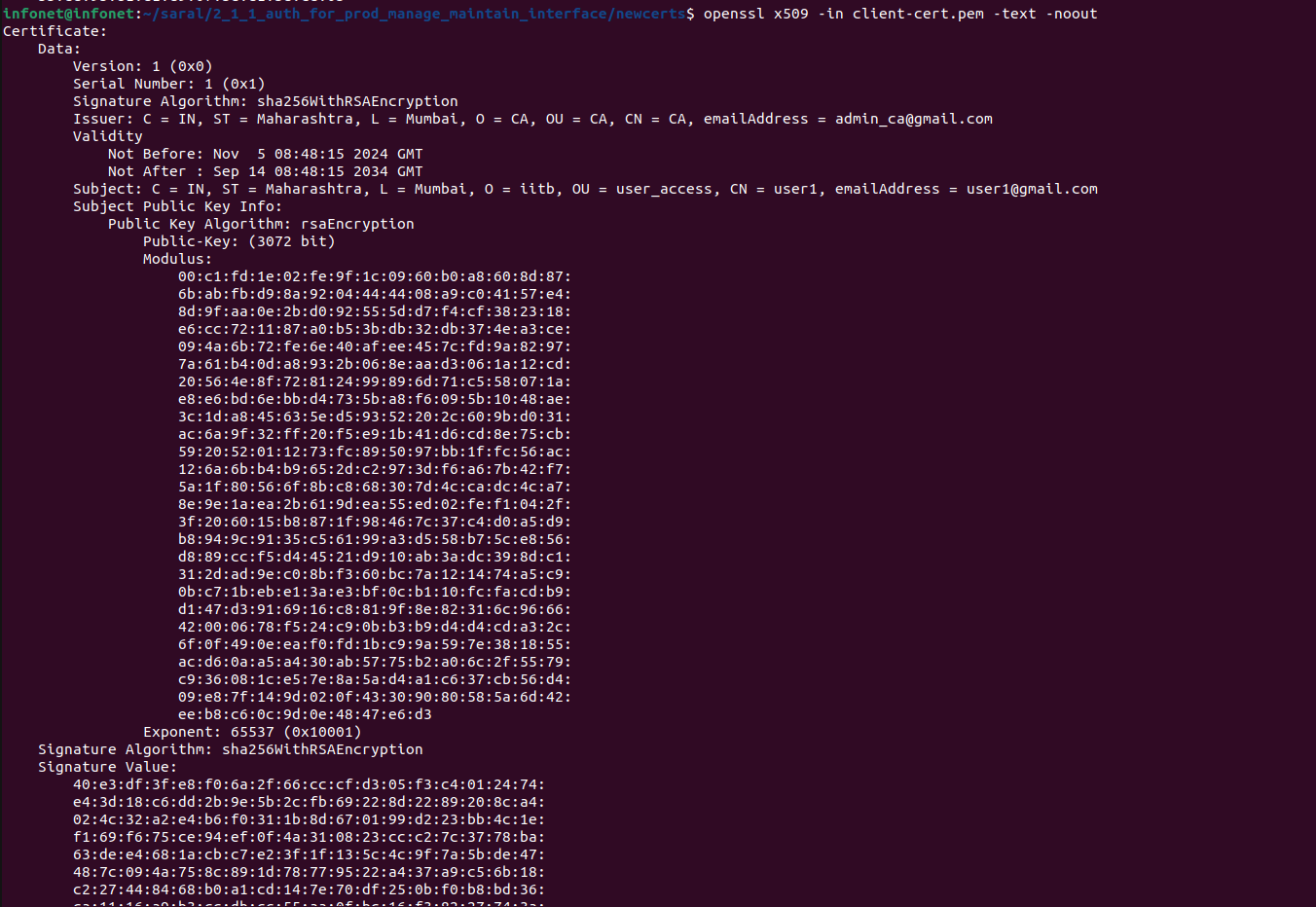
**Execute the commands: ‘openssl pkey -in /etc/mysql/newcerts/server-key.pem -text -noout’** to verify RSA encryption(RSA-3072 bit). This confirms if the private key is valid and displays its modulus.



**Fig. 17 The screen output examines server-key.pem using RSA-3072 bits showing their modulus and exponent. Used OpenSSL 3.0.2 15 Mar 2022 (Library: OpenSSL 3.0.2 15 Mar 2022).ITSAR number ‘ITSAR001962411’, ITSAR name NCCS/ITSAR/Standards Applicable for Group of Equipment/Cryptographic Controls/Cryptographic Controls (Applicable to all ITSARs) and version 2.0.0.**

1. To verify certificate rsa encryption RSA-3072 bit on the UDR NF

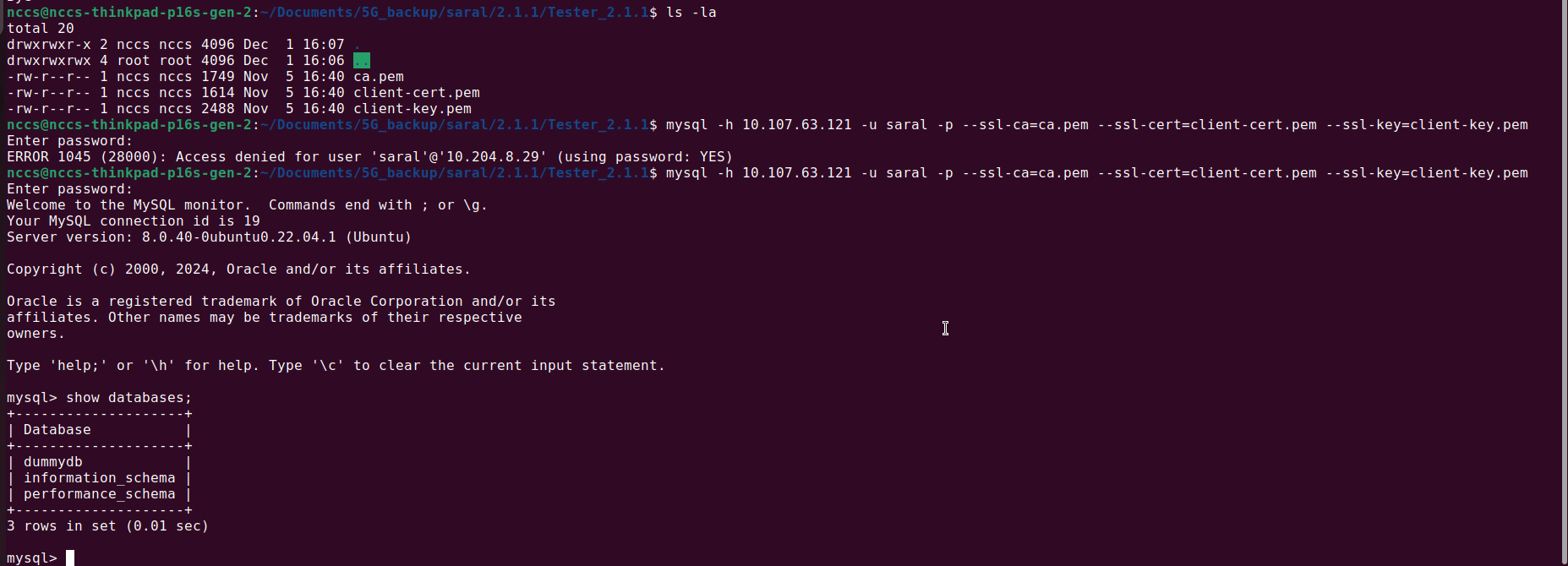
**Execute the commands: ‘openssl x509 -in /etc/mysql/newcerts/client-cert.pem -text -noout** to verify RSA encryption (RSA-3072 bit). This displays the certificate details, including its issuer, validity period, and subject.



**Fig. 18 The screen output examines certificate details, including its issuer, validity period, and subject of client-cert.pem. Used OpenSSL 3.0.2 15 Mar 2022 (Library: OpenSSL 3.0.2 15 Mar 2022). ITSAR number ‘ITSAR001962411’, ITSAR name NCCS/ITSAR/Standards Applicable for Group of Equipment/Cryptographic Controls/Cryptographic Controls (Applicable to all ITSARs) and version 2.0.0.**

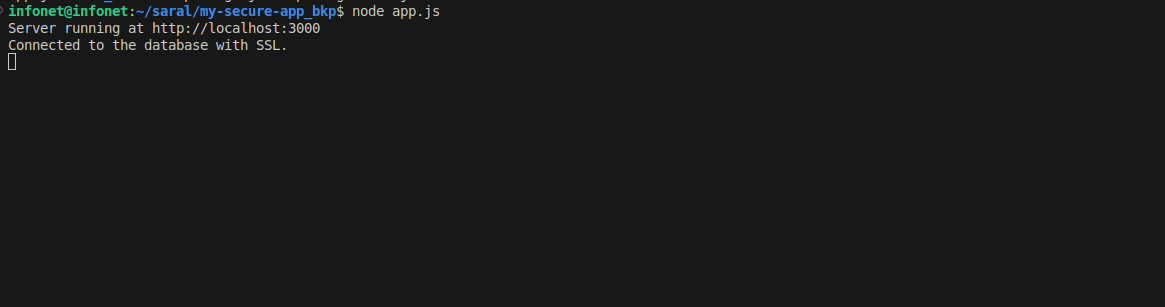
1. **Execute the commands: ‘ls -la’ to verify access right of certificate and key on client system.**

**Execute the command on client system Tester folder:** ‘mysql -h 10.107.63.121 -u saral -p --ssl-ca=ca.pem --ssl-cert=client-cert.pem --ssl-key=client-key.pem’.



**Fig. 19 The screen output displays CA certificate and key for Client and their access rights in client system. The tester triggers communication between the UDR NF database server and a client system that doesn't have a legitimate authentication credential that shows access denied for user and client with correct credentials can be established connection with the server.**

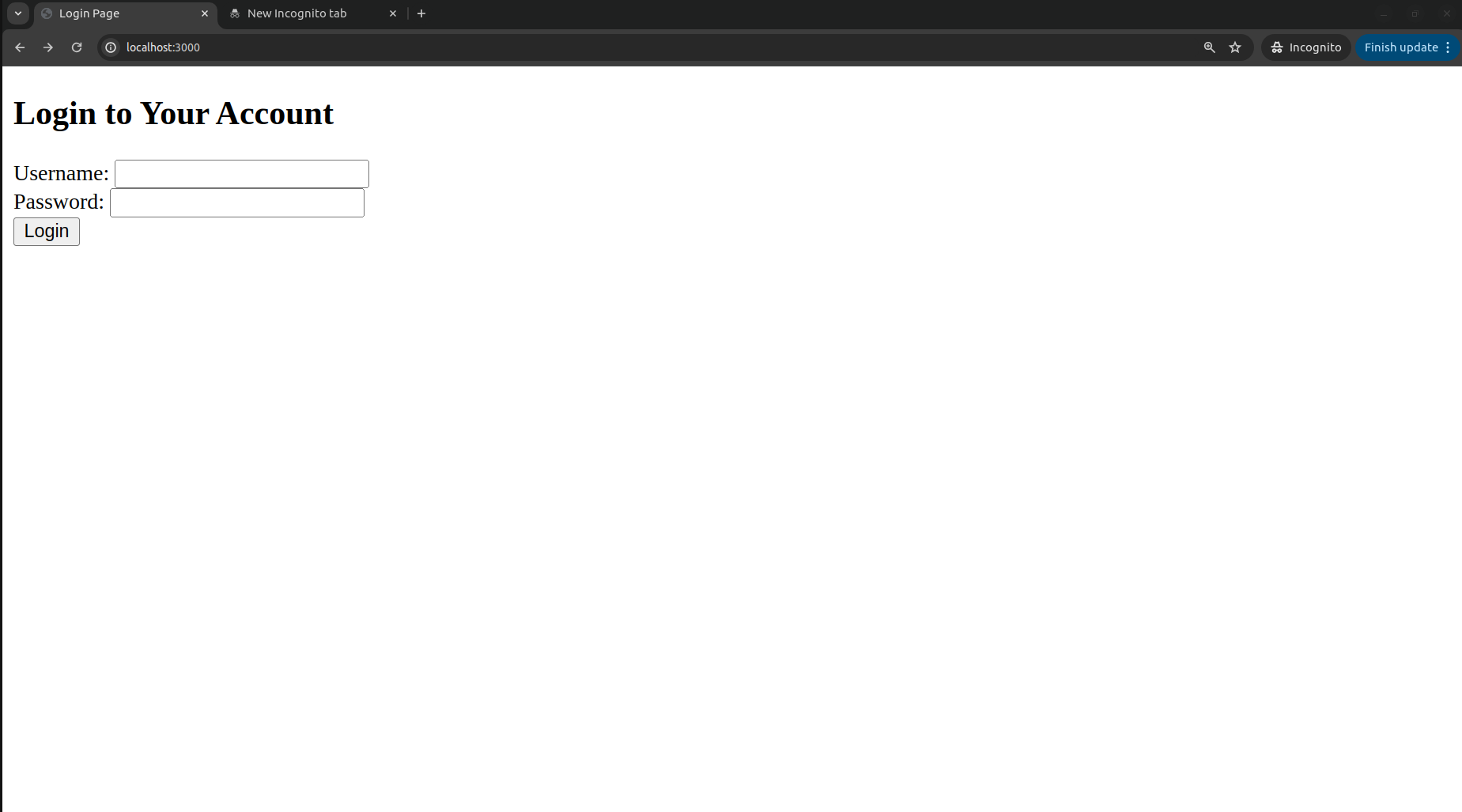
iii) **Execute the commands:** ‘node app.js’ and verify the server running status and connection with http/https over SSL/TLS.



**Fig. 20 The screen output displays the server running on localhost port 3000 and connected**

**to the database with http/https over SSL/TLS.**

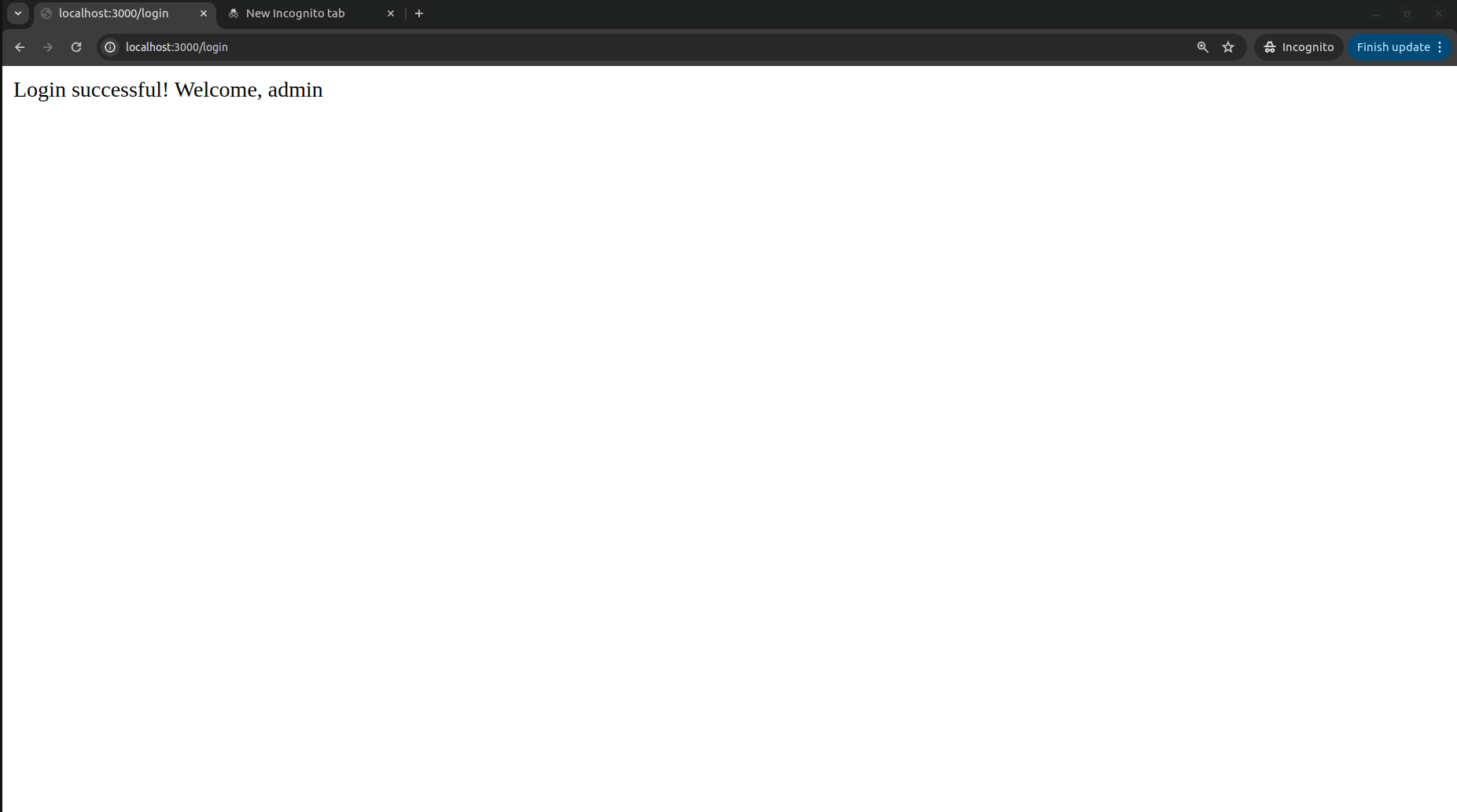
iv) Verify the server running on localhost port 3000 and connect to the database with SSL in the webbrowser. Login interface will display as shown in figure below.



**Fig. 21 The screen output displays the server running on localhost port 3000 and connected**

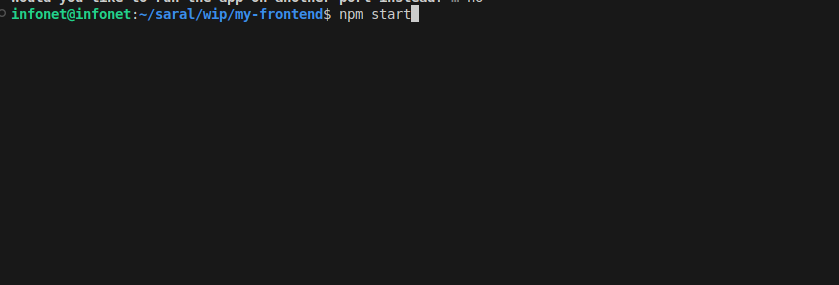
**to the database with http/https over SSL/TLS in web Browser.**

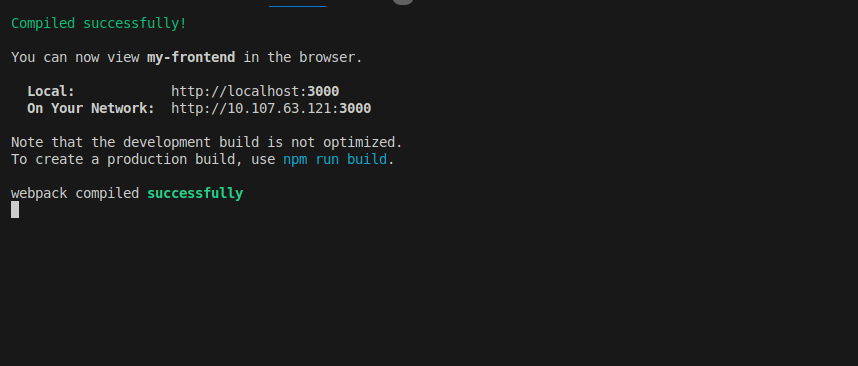
v) When you enter the correct authentication credentials ‘login successful and welcome, admin’ will be displayed in the browser as shown in figure below. SSL certificate gets verified.



**Fig. 22 The screen output displays the server running on localhost port 3000 and connected to the database with SSL/TLS in web Browser.**

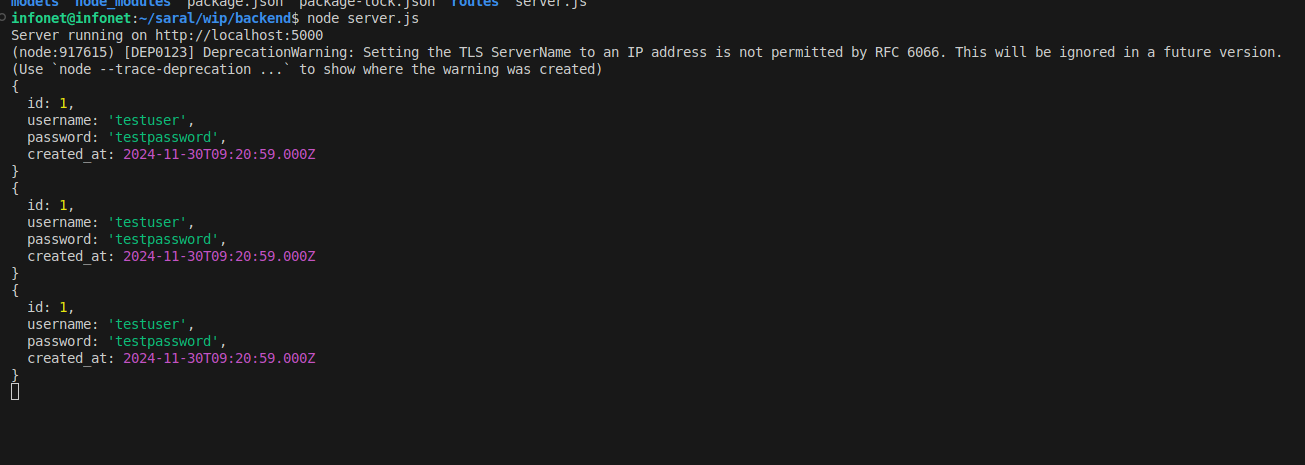
**vi) Execute the commands:** ‘npm start’ and verify the server running status and connection with http/https over SSL/TLS.





**Fig. 23 The screen output displays command run ‘npm start’ and webpack frontend compiled successfully.**

**vii) Execute the commands:** ‘node server.js’ and verify the server running status and connection with http/https over SSL/TLS.



**Fig. 24 The screen output displays the backend starting with ‘node server.js’ the server running on localhost port 5000 and setting the ‘TLS’ serverName with credential user ‘testuser’ and password 'testpassword’.**

viii) When you enter the correct authentication credentials ‘Logged in successfully’ will be displayed in the browser as shown in figure below. TLS certificate gets verified.



**Fig. 25 The screen outputs the server login successfully with correct authentication credentials connected to the database with http/https over SSL/TLS in web Browser.**

ix) When you enter the incorrect authentication credentials ‘Login failed’ will be displayed in the browser as shown in figure below.



**Fig. 26 The screen outputs the server login failed with incorrect authentication credentials.**

Obtain the screen outputs as seen in figures 13, 14, 15,16,17, 18, 19, 20,21,22,23,24,25 and 26.

1. **Test Observations:**

The Tester verifies that there is mutual authentication of entities for management interfaces on the UDR NF of all databases. All databases having management protocols and the authentication mechanism used for each one follows Secure cryptographic controls prescribed in Table1 of the latest document “Indian Telecom Security Assurance Requirements (ITSAR) for Cryptographic Controls” shall only be used for UDR management and maintenance”as provided by the vendor’s documentation through command(s)/script as above.

**Case 1.1:** The tester checks all UDR NF database **authentication mechanisms configuration that does not configure properly and unfollows** secure cryptographic controls prescribed in Table1 of the latest document “Indian Telecom Security Assurance Requirements (ITSAR) for Cryptographic Controls” shall only be used for UDR management and maintenance”. i.e **(Negative CASE**).

**Case 1.2:** The tester checks all UDR NF database **triggers communication between the UDR NF and a client system that has a legitimate authentication credential** **and follows** secure cryptographic controls prescribed in Table1 of the latest document “Indian Telecom Security Assurance Requirements (ITSAR) for Cryptographic Controls” shall only be used for UDR management and maintenance”. Mutual authentication is successful and communication between UDR NF and the entity with correct credentials can be established. i.e **(Positive CASE**).

**Case 1.3:** The tester checks all UDR NF database **triggers communication between the UDR NF and a client system that doesn't have a legitimate authentication credential** **and follows** secure cryptographic controls prescribed in Table1 of the latest document “Indian Telecom Security Assurance Requirements (ITSAR) for Cryptographic Controls” shall only be used for UDR management and maintenance”. Mutual authentication fails and communication between the UDR NF and the entity with incorrect credentials cannot be established. i.e **(Positive CASE**).

The Tester verifies that there is mutual authentication of entities for management interfaces on the UDR NF of all databases. All databases of management protocols and the authentication mechanism used for each one follows Secure cryptographic controls prescribed in Table1 of the latest document “Indian Telecom Security Assurance Requirements (ITSAR) for Cryptographic Controls” shall only be used for UDR management and maintenance”.

In case of a negative test result, the tester:

-indicates the result as ‘**Fails’;**

**else,**

- indicates the result as **‘Pass’.**

|  |  |  |  |
| --- | --- | --- | --- |
| **Sr. No.** | **Test Case Name** | **Pass/Fail** | **Remarks** |
|  | TC\_MUTUAL\_AUTHENTICATION-ON\_NETWORK\_PRODUCT\_MANAGEMENT\_PROTOCOLS |  |  |

**Script for app.js**

const express = require('express');

const mysql = require('mysql2');

const fs = require('fs');

const path = require('path');

require('dotenv').config();

const app = express();

const port = 3000;

// Load SSL certificates

const caCert = fs.readFileSync(path.join(\_\_dirname, 'ssl', 'ca-cert.pem'));

const clientCert = fs.readFileSync(path.join(\_\_dirname, 'ssl', 'client-cert.pem'));

const clientKey = fs.readFileSync(path.join(\_\_dirname, 'ssl', 'client-key.pem'));

// Create a secure MySQL connection

const connection = mysql.createConnection({

host: process.env.DB\_HOST,

user: process.env.DB\_USER,

password: process.env.DB\_PASSWORD,

database: process.env.DB\_NAME,

port: process.env.DB\_PORT,

ssl: {

ca: caCert,

cert: clientCert,

key: clientKey

}

});

// Test the database connection

connection.connect((err) => {

if (err) {

console.error('Database connection failed:', err.stack);

return;

}

console.log('Connected to the database with SSL.');

});

// Middleware to parse incoming requests

app.use(express.urlencoded({ extended: true }));

app.use(express.json());

// Serve static HTML files for the login interface

app.get('/', (req, res) => {

res.sendFile(path.join(\_\_dirname, 'views', 'login.html'));

});

// Route for handling login

app.post('/login', (req, res) => {

const { username, password } = req.body;

// Query the database to authenticate the user

const query = 'SELECT \* FROM application\_users WHERE username = ? AND password = ?';

connection.connect((err) => {

if (err) {

// Authentication failed

res.status(401).send('Invalid username or password');

} else {

// Authentication succeeded

res.send('Login successful! Welcome, ' + username);

connection.end();

}

});

});

// Start the Express.js server

app.listen(port, () => {

console.log(`Server running at http://localhost:${port}`);

});

// ============================================================================================================

// const fs = require('fs');

// const express = require('express');

// const mysql = require('mysql2');

// require('dotenv').config();

// const app = express();

// const port = 3000;

// // Middleware to parse incoming requests

// app.use(express.urlencoded({ extended: true }));

// app.use(express.json());

// // Serve static HTML files for the login interface

// app.get('/', (req, res) => {

// res.sendFile(\_\_dirname + '/views/login.html');

// });

// // Route for handling login

// app.post('/login', (req, res) => {

// const { username, password } = req.body;

// // Attempt to establish a MySQL connection with the provided credentials

// const tempConnection = mysql.createConnection({

// host: '127.0.0.1',

// user: username,

// password: password,

// database: process.env.DB\_NAME,

// ssl: {

// ca: fs.readFileSync(\_\_dirname + '/ssl/ca-cert.pem'),

// cert: fs.readFileSync(\_\_dirname + '/ssl/client-cert.pem'),

// key: fs.readFileSync(\_\_dirname + '/ssl/client-key.pem')

// }

// });

// tempConnection.connect((err) => {

// if (err) {

// // Authentication failed

// res.status(401).send('Invalid username or password');

// } else {

// // Authentication succeeded

// res.send('Login successful! Welcome, ' + username);

// tempConnection.end();

// }

// });

// });

// // Start the Express.js server

// app.listen(port, () => {

// console.log(`Server running at http://localhost:${port}`);

// });

####

The cryptographic algorithm used in the private key is **RSA**. This can be inferred from the format and the specific clues in the PEM block provided:

1. **Header Information**:
   * The private key block starts with -----BEGIN PRIVATE KEY-----, indicating that it follows the PKCS #8 standard for private keys. PKCS #8 can encapsulate various types of private keys, but further clues point to RSA.
2. **Key Details**:
   * Inside the file, references to RSA operations can often be confirmed by analyzing the structure of the encoded data if decoded using tools like openssl.

You can verify the type of the key using this command:  
bash  
Copy code  
openssl pkey -in private\_key.pem -text -noout

* + If it is an RSA private key, it will display the RSA-specific parameters (modulus, private exponent, etc.).

1. **RSA Identification**:
   * RSA (Rivest-Shamir-Adleman) is one of the most common cryptographic algorithms used for key pairs in such contexts.

To verify SSL keys and certificates using openssl, you can use the following commands for different purposes:

### **1. Verify Private Key**

To check the integrity of a private key:

bash

Copy code

openssl rsa -in private.key -check

This will confirm if the private key is valid and displays its modulus.

### **2. Verify Certificate**

To check if the certificate is valid:

bash

Copy code

openssl x509 -in certificate.crt -text -noout

This displays the certificate details, including its issuer, validity period, and subject.

### **3. Verify the Match Between Private Key and Certificate**

To ensure the private key matches the certificate, compare their moduli:

bash

Copy code

openssl rsa -in private.key -modulus -noout

openssl x509 -in certificate.crt -modulus -noout

The output for both commands should match. If they don’t, the private key and certificate do not belong together.

### **4. Verify Certificate Signing Request (CSR)**

To check if the CSR matches the private key:

bash

Copy code

openssl req -in certificate.csr -noout -verify

### **5. Verify Chain of Trust**

To validate the chain of trust for a certificate against a CA:

bash

Copy code

openssl verify -CAfile ca.pem certificate.crt

### **6. Verify Client Certificate**

To check client certificate details:

bash

Copy code

openssl x509 -in client-cert.pem -text -noout

### **7. Debug SSL/TLS Connection**

To test a certificate and key combination:

bash

Copy code

openssl s\_server -cert server-cert.pem -key server-key.pem -accept 443

This starts an SSL/TLS server for debugging.

Let me know if you need detailed examples for any of these commands!

The X509v3 extensions section is part of an X.509 digital certificate, providing additional metadata about the certificate's usage, constraints, and relationships. Here’s an explanation of the specific extensions you mentioned:

### **1. X509v3 Subject Key Identifier**

* **What it is**: A unique identifier for the public key contained in the certificate. It allows quick identification of the certificate in operations like chain building or trust verification.
* **In your case**: 13:F8:C7:05:55:72:3E:51:DE:CB:69:48:BF:C0:92:F6:C2:17:1D:2E is a hash of the certificate's public key, uniquely identifying it.
* **Purpose**:
  + Helps distinguish certificates, especially in environments with multiple keys.
  + Facilitates efficient lookup of the certificate when validating trust.

### **2. X509v3 Authority Key Identifier**

* **What it is**: A unique identifier for the public key of the certificate's issuer (Certificate Authority or CA).
* **In your case**: The value is the same as the Subject Key Identifier because this is a self-signed certificate (the certificate is its own issuer).
* **Purpose**:
  + Allows the linking of a certificate to its issuing authority.
  + Used in trust chain validation to ensure the certificate belongs to the trusted CA.

### **3. X509v3 Basic Constraints**

* **What it is**: Specifies whether the certificate is a Certificate Authority (CA) and the depth of the certificate chain it can sign.
* **Key fields**:
  + **critical**: Indicates this extension is essential for validation. If the application doesn’t understand it, the certificate is rejected.
  + **CA:TRUE**: Specifies that the certificate is a CA, meaning it can issue certificates to others.
  + **Path length**: Not included here, but this field limits the maximum number of subordinate CAs that can be signed by this CA.

### **Summary**

This certificate:

* Is a **Certificate Authority** because of CA:TRUE.
* Is **self-signed**, as evidenced by the identical Subject and Authority Key Identifiers.
* Contains unique identifiers for efficient validation in cryptographic operations.

These fields are crucial in securing PKI (Public Key Infrastructure) operations like SSL/TLS connections, signing, and encryption.

The OpenSSL software library is a robust, commercial-grade, full-featured toolkit for general-purpose cryptography and secure communication. It is developed under the OpenSSL [Mission](https://openssl-mission.org/) with support from the OpenSSL [Foundation](https://openssl-foundation.org/) and OpenSSL [Corporation](https://openssl-corporation.org/)

### **PKCS #8 (Public Key Cryptography Standards #8)**

**PKCS #8** is a standard format for storing private keys. It is defined by RSA Security in the Public Key Cryptography Standards (PKCS). It supports both encrypted and unencrypted private keys and is widely used in cryptographic applications for portability and interoperability.

### **Key Features of PKCS #8:**

#### **1. Versatile Private Key Storage**

* PKCS #8 can store private keys for any asymmetric cryptographic algorithm, including RSA, DSA, and EC (Elliptic Curve).
* It defines a structure to encapsulate private keys, making them algorithm-agnostic.

#### **2. Supports Encryption for Security**

* Private keys in PKCS #8 can be stored in an **encrypted format** for secure transport or storage.
* Encryption is applied using a password or passphrase.

#### **3. Interoperability**

* Standardized format ensures compatibility across various cryptographic libraries and systems.
* Commonly used in applications such as OpenSSL, Java KeyStore, and other cryptography tools.

#### **4. PEM or DER Encoding**

* PKCS #8 keys can be encoded in:
  + **PEM format**: Base64-encoded with headers like -----BEGIN PRIVATE KEY----- or -----BEGIN ENCRYPTED PRIVATE KEY-----.
  + **DER format**: Binary format for compact storage and transport.

#### **5. Two Types of Key Formats**

* **Unencrypted Private Key**:
  + Used for raw storage of the private key without encryption.
  + PEM header: -----BEGIN PRIVATE KEY-----.
* **Encrypted Private Key**:
  + Encapsulates the private key in an encrypted form.
  + PEM header: -----BEGIN ENCRYPTED PRIVATE KEY-----.

#### **6. Algorithm Flexibility**

* PKCS #8 provides information about the cryptographic algorithm associated with the private key.
* Supports extensibility for new algorithms through the use of Object Identifiers (OIDs).

#### **7. Ease of Key Management**

* PKCS #8 simplifies key lifecycle management (creation, storage, sharing) by providing a unified structure.

### **Structure of PKCS #8 Private Key**

A PKCS #8 private key consists of the following fields:

1. **Algorithm Identifier**: Specifies the cryptographic algorithm (e.g., RSA, EC).
2. **Private Key**: Encapsulated private key in DER format.
3. **Optional Attributes**: May include metadata about the key.

### **Example PKCS #8 Headers**

**Unencrypted Private Key**:  
vbnet  
Copy code  
-----BEGIN PRIVATE KEY-----

(Base64-encoded data)

-----END PRIVATE KEY-----

**Encrypted Private Key**:  
vbnet  
Copy code  
-----BEGIN ENCRYPTED PRIVATE KEY-----

(Base64-encoded data)

-----END ENCRYPTED PRIVATE KEY-----

### **Common Use Cases**

1. **Key Exchange**: Storing and sharing private keys between systems or tools.
2. **Security Applications**: Transporting encrypted private keys securely.
3. **Certificate Signing Requests (CSRs)**: Using private keys for signing requests.

### **Generating and Using PKCS #8 Keys**

1. **With OpenSSL**:

Convert a private key to PKCS #8:  
bash  
Copy code  
openssl pkcs8 -topk8 -inform PEM -outform PEM -in private.key -out private\_pkcs8.key -nocrypt

Create an encrypted PKCS #8 key:  
bash  
Copy code  
openssl pkcs8 -topk8 -inform PEM -outform PEM -in private.key -out private\_pkcs8.key -passout pass:yourpassword

1. **In Java**:
   * Java KeyStore uses PKCS #8 for private key storage.

### **Comparison to PKCS #12**

* PKCS #8 is **only for private keys**, whereas PKCS #12 stores private keys **along with certificates and additional data** in a single container.
* PKCS #8 is lightweight and simpler for cases where only the private key is needed.

### **Summary**

PKCS #8 is a flexible and secure standard for private key storage, offering encryption, algorithm independence, and compatibility across systems. It is crucial for key management in cryptographic systems and secure communications.

**PKCS #12** (Public Key Cryptography Standards #12) is a binary format for storing cryptographic objects, including private keys, certificates, and other related information. It is widely used for securely transferring sensitive information between systems, such as exporting or importing certificates and private keys for SSL/TLS purposes.

### **Key Features of PKCS #12:**

1. **Storage of Cryptographic Material**:
   * Includes private keys, public keys, certificates, and optionally other objects like CA chains.
   * These are bundled into a single file.
2. **File Extensions**:
   * PKCS #12 files commonly have extensions like .p12 or .pfx.
3. **Password Protection**:
   * The file is encrypted with a password to protect the private keys and other sensitive contents.
   * Provides confidentiality during transport or storage.
4. **Interoperability**:
   * Supported by many systems, including browsers, servers, and cryptographic libraries, for importing/exporting certificates and keys.

### **Common Use Cases:**

1. **SSL/TLS Certificates**:
   * PKCS #12 is often used to transfer private keys and certificates to web servers, such as importing an SSL/TLS certificate into IIS, Apache, or Nginx.
2. **Code Signing**:
   * Developers use PKCS #12 files for storing code-signing certificates and private keys.
3. **Secure Email**:
   * Secure email systems use PKCS #12 to store and transport S/MIME certificates and keys.

### **Structure of PKCS #12:**

A PKCS #12 file typically contains:

1. **Private Key**: Essential for SSL/TLS or digital signatures.
2. **Certificate**: Public part of the cryptographic pair, often used for verification.
3. **Certificate Chain**: Intermediate and root certificates, ensuring the chain of trust.
4. **Optional Attributes**: Metadata about the certificates or keys.

### **Example Tools for Handling PKCS #12:**

**OpenSSL**: Commonly used to create and manipulate .p12 files.  
bash  
Copy code  
# Convert PEM files (key and certificate) to PKCS #12

openssl pkcs12 -export -out certificate.p12 -inkey private.key -in certificate.crt -certfile ca\_bundle.crt

1. **Keytool**: Java’s keystore tool for handling PKCS #12 files.
2. **Windows MMC**: Used to import/export .pfx files into the Windows Certificate Store.

### **Summary:**

PKCS #12 is a secure container format that simplifies the management and transfer of cryptographic materials like private keys and certificates. Its encryption and widespread support make it an essential part of secure communications and authentication.

Welcome to *GnuTLS* project pages

* **Overview**GnuTLS is a secure communications library implementing the [SSL, TLS and DTLS protocols](https://datatracker.ietf.org/wg/tls/charter/) and technologies around them. It provides a simple C language application programming interface (API) to access the secure communications protocols as well as APIs to parse and write X.509, PKCS #12, and other required structures.  
  The project strives to provide a secure communications back-end, [simple to use](https://gnutls.org/manual/html_node/Simple-client-example-with-X_002e509-certificate-support.html" \l "Simple-client-example-with-X_002e509-certificate-support) and integrated with the rest of the base Linux libraries. A back-end designed to work and be secure out of the box, keeping the complexity of TLS and PKI out of application code.
* **Features**
  + Support for TLS 1.3, 1.2, 1.1, 1.0 protocols, and (optionally) SSL 3.0
  + Support for [DTLS 1.2](https://tools.ietf.org/html/rfc6347), and DTLS 1.0, protocols
  + Support for certificate path validation, as well as [DANE](https://www.gnutls.org/manual/html_node/Verifying-a-certificate-using-DANE.html" \l "Verifying-a-certificate-using-DANE) and [trust on first use](https://www.gnutls.org/manual/html_node/Verifying-a-certificate-using-trust-on-first-use-authentication.html" \l "Verifying-a-certificate-using-trust-on-first-use-authentication).
  + Support for the [Online Certificate Status Protocol (OCSP)](https://www.gnutls.org/manual/html_node/OCSP-certificate-status-checking.html).
  + Support for public key methods, including RSA and Elliptic curves, as well as password and key authentication methods such as [SRP](https://www.gnutls.org/manual/html_node/Authentication-using-SRP.html" \l "Authentication-using-SRP) and [PSK](https://www.gnutls.org/manual/html_node/Authentication-using-PSK.html" \l "Authentication-using-PSK) protocols.
  + Support for all the strong encryption algorithms, including AES and Camellia.
  + Support for CPU-assisted cryptography with VIA padlock and AES-NI instruction sets.
  + Support for cryptographic accelerator drivers via [/dev/crypto](http://www.cryptodev-linux.org/).
  + Supports natively [HSMs and cryptographic tokens](https://www.gnutls.org/manual/html_node/Smart-cards-and-HSMs.html" \l "Smart-cards-and-HSMs), via PKCS #11 and the [Trusted Platform Module (TPM)](https://www.gnutls.org/manual/html_node/Trusted-Platform-Module.html" \l "Trusted-Platform-Module).
  + Runs on most Unix platforms and Windows.
* **License**The core library licensed under the [GNU Lesser General Public License version 2.1](https://www.gnu.org/licenses/old-licenses/lgpl-2.1.html) (LGPLv2.1+). The LGPL license is compatible with a wide range of free licenses, and even permits you to use GnuTLS in non-free proprietary programs.
* **Documentation:**You can obtain [GnuTLS' manual at lulu.com](https://www.lulu.com/shop/nikos-mavrogiannopoulos-and-simon-josefsson/the-gnutls-manual/paperback/product-18963264.html) or download [any of the electronic formats](https://www.gnutls.org/documentation.html).

For more information on GnuTLS features, see the [wikipedia article comparing different TLS implementations](https://en.wikipedia.org/wiki/Comparison_of_TLS_Implementations).

The Transport Layer Security Protocol (TLS) is a cryptographic protocol that is used to secure

a connection between a client and a server. It is the successor of the outdated Secure Socket

Layer (SSL) protocol. What makes TLS stands out from SSL is that TLS is more secure than SSL in several aspects as TLS supports newer ciphersuites, stronger encryption algorithms and

hash functions, as well as a more reliable handshake.

### **Explanation**

1. **Certificates:**
   * The server uses server-cert.pem and server-key.pem for its public-private key pair.
   * The client verifies the server's certificate against ca-cert.pem.
2. **TLS Handshake:**
   * The client and server perform a TLS handshake:
     + The server sends its public key (certificate).
     + The client verifies the certificate using the CA's public key.
     + A session key is securely established between the client and server.
3. **Secure Communication:**
   * After the handshake, the session key is used for encrypting and decrypting data between the client and server.
4. **Files Required:**
   * Generate certificates using OpenSSL:

####### Public vs Private key###################################

Differences between public and private keys

The primary difference between public and private keys are summarized in the table below:

|  |  |
| --- | --- |
| **Public Key** | **Private key** |
| Are used to encrypt | Are used to decrypt |
| Are shared publicly | Only the owner has the key |
| Can be used for encrypting or signing | Used to decrypt or validate signature |
| Cannot be used to guess private key | Cannot be used to guess public key |

Benefits of public private key

By using a public and private key for encryption and decryption, recipients can be confident that the data is what the sender says it is and the sender is who they say they are. The recipient is assured of the confidentiality, integrity, and authenticity of data exchange.

Confidentiality is ensured because the content secured with the public key can only be decrypted with the private key, so that only the intended recipient access the information.

Integrity is ensured because the decryption process includes the step of checking that the received message matches the sent message. This validates that the message has not been changed in transit.

Authenticity is ensured because each message is digitally signed using the sender’s private key. The only way to decrypt that private key is with the corresponding public key, which the recipient can access. By signing the message with the sender’s private key, this guarantees for the recipient that the message really did come from the sender.</p>

## **How PreVeil Harnesses the Strength of Keys**

PreVeil leverages the power of the public-private keys to provide unbeatable data security through end-to-end encryption. With this advanced cryptographic system, messages and attachments are encrypted on the sender’s device using the recipient’s public key.

The encrypted data can only be decrypted on the recipient’s device with their corresponding private key, so that sensitive information remains confidential throughout transmission and storage. By eliminating the a central point of attack while data is in transit or at rest on the server, PreVeil’s end-to-end encryption offers robust protection against unauthorized access and interception.

With user-friendly features like automatic key management and secure logs, PreVeil ensures that security doesn’t compromise usability. With PreVeil’s end-to-end encryption service, organizations and individuals can rest assured that their data is protected at all times.

<https://www.preveil.com/blog/public-and-private-key/>

