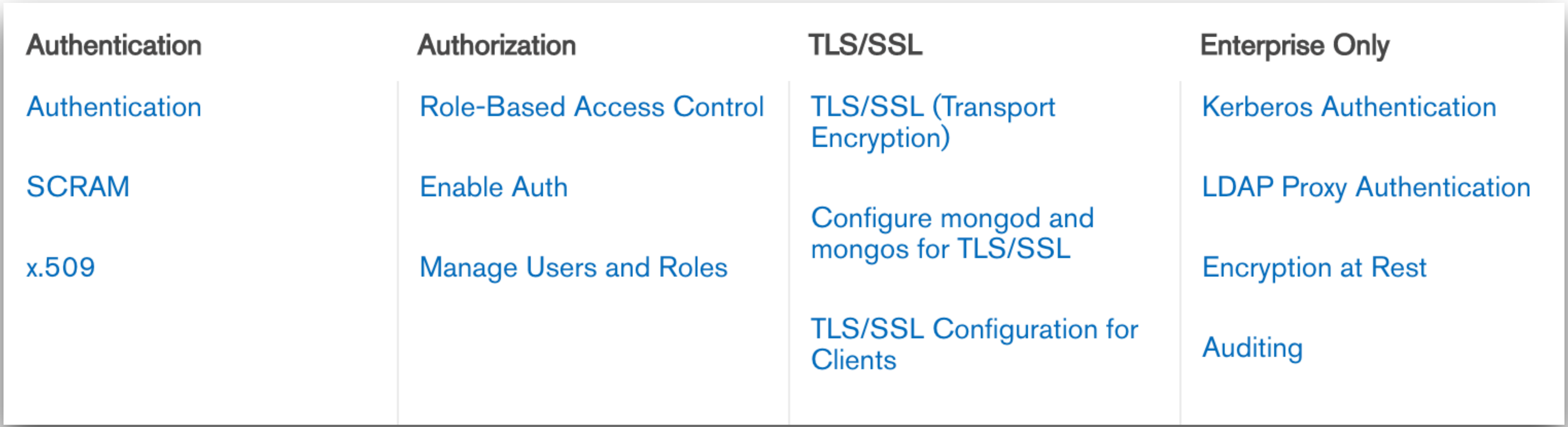
MongoDB Security

Kayartaya Vinod

MongoDB Security

- MongoDB provides various features, such as authentication, access control, encryption, to secure your MongoDB deployments.

- Some key security features include:



Authentication	Authorization	TLS/SSL	Enterprise Only
Authentication	Role-Based Access Control	TLS/SSL (Transport Encryption)	Kerberos Authentication
SCRAM	Enable Auth	Configure mongod and mongos for TLS/SSL	LDAP Proxy Authentication
x.509	Manage Users and Roles	TLS/SSL Configuration for Clients	Encryption at Rest
			Auditing

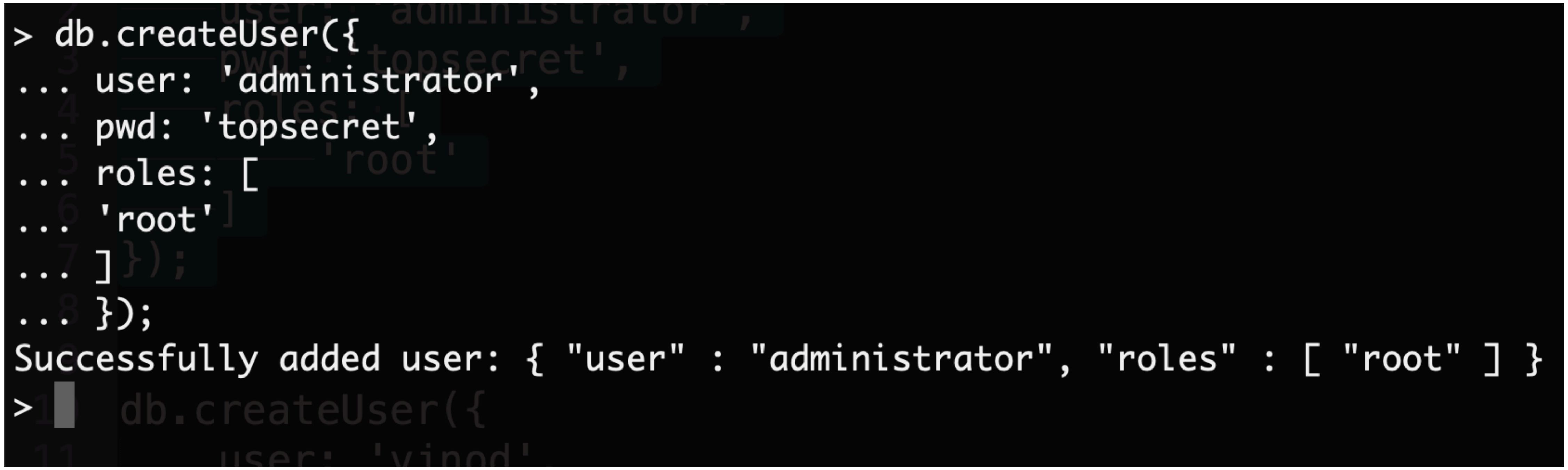
Authentication and Authorization

- Authentication verifies the identity of a user
- Authorization determines the verified user’s access to resources and operations.

Step #1: Create an administrator user account

- Start the server
- Connect to the server using mongo shell
- Switch to the ‘admin’ database
- Use the db.createUser() function to create the administrator user

Step #1: Create an administrator user account

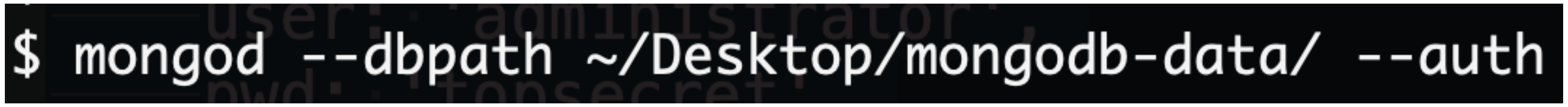


```
> db.createUser({
... user: 'administrator',
... pwd: 'topsecret',
... roles: [
... 'root'
... ]
... });
Successfully added user: { "user" : "administrator", "roles" : [ "root" ] }
>
```

- Once done, exit the shell.

Step #2: Restart the server in auth mode

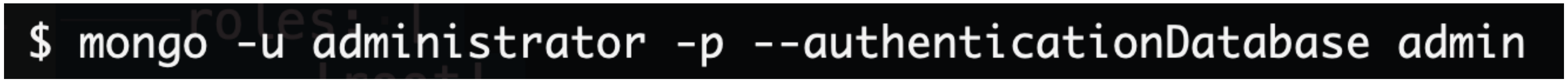
- Stop the server
- Start the server with the --auth flag



```
$ mongod --dbpath ~/Desktop/mongodb-data/ --auth
```

Step #3: Create new users

- Connect to the server using the ‘mongo’ command, but this time supply username, password and the authentication database



```
$ mongo -u administrator -p --authenticationDatabase admin
```

- It will prompt for the password

Step #3: Create new users

- If you wish to create the users in a separate database, then switch to that database
    - Ex: use authdb
- Otherwise, switch to the ‘admin’ database and run the db.createUser() function with appropriate values

Step #3: Create new users

```
db.createUser({
   user: 'vinod',
   pwd: 'secret',
   roles: [
      { role: ‘readWrite', db: ‘workdb1' },
      { role: ‘read', db: ‘workdb2' }
   ]
});
```

Step #3: Create new users

- When adding a user, you create the user in a specific database.
    - This database is the authentication database for the user.
- A user can have privileges across different databases; that is, a user’s privileges are not limited to their authentication database.
- By assigning to the user roles in other databases, a user created in one database can have permissions to act on other databases.

Step #3: Create new users

- The user’s name and authentication database serve as a unique identifier for that user.
  - That is, if two users have the same name but are created in different databases, they are two separate users.
- If you intend to have a single user with permissions on multiple databases, create a single user with roles in the applicable databases instead of creating the user multiple times in different databases.

Authenticate the user

- To authenticate as a user, you must provide a username, password, and the authentication database associated with that user.

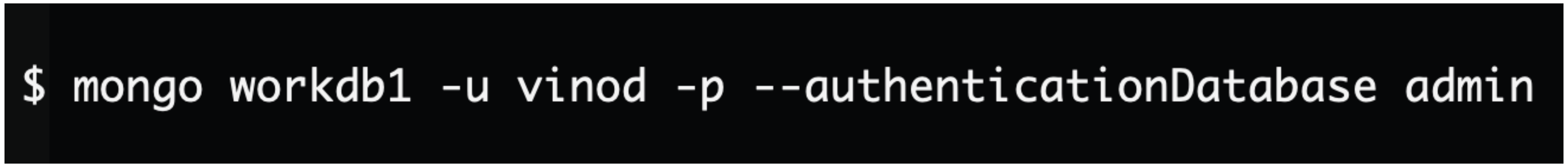
- To authenticate using the mongo shell, either:

    - Use the mongo command-line authentication options (--username, --password, and --authenticationDatabase) when connecting to the mongod or mongos instance, or

    - Connect first to the mongod or mongos instance, and then run the authenticate command or the db.auth() method against the authentication database.

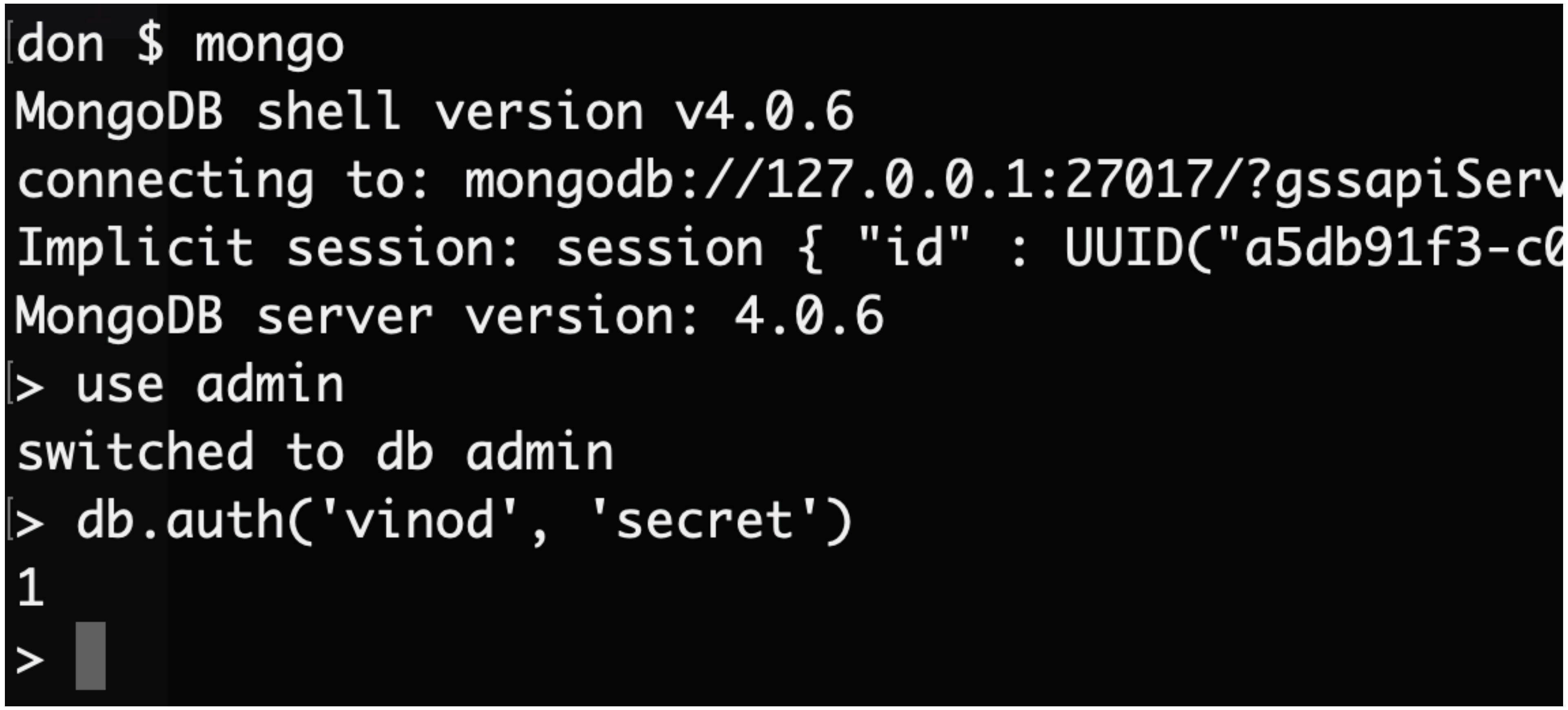
Authenticate the user

- First method:



```
$ mongo workdb1 -u vinod -p --authenticationDatabase admin
```

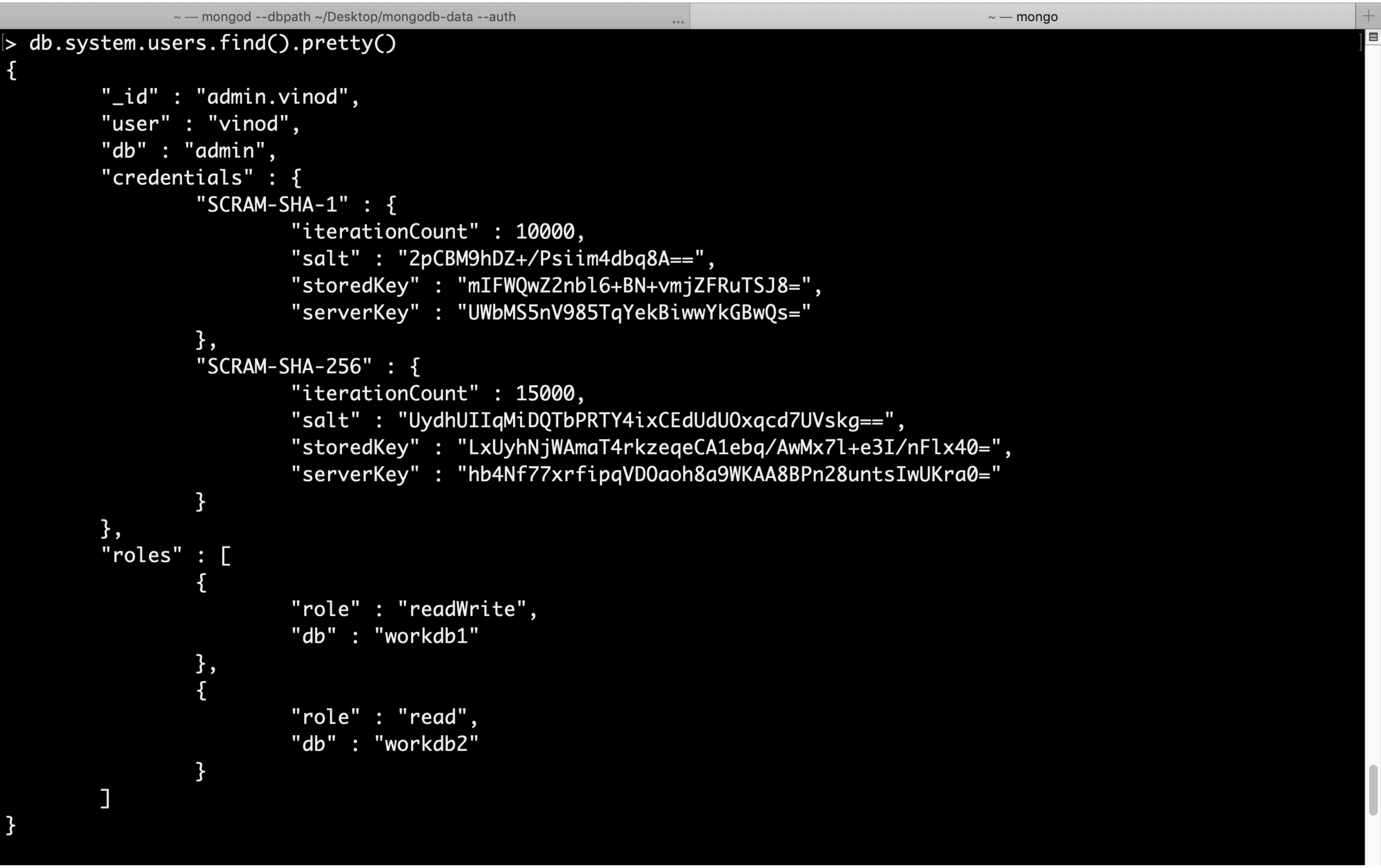
- Second method:



```
don $ mongo
MongoDB shell version v4.0.6
connecting to: mongodb://127.0.0.1:27017/?gssapiServ
Implicit session: session { "id" : UUID("a5db91f3-c0
MongoDB server version: 4.0.6
> use admin
switched to db admin
> db.auth('vinod', 'secret')
1
> 
```

Authenticate the user

- MongoDB stores all user information, including name, password, and the user's authentication database, in the system.users collection in the admin database.
- Do not access this collection directly but instead use the user management commands.



~ — mongod --dbpath ~/Desktop/mongodb-data --auth   ~ — mongo

```
> db.system.users.find().pretty()
{
        "_id" : "admin.vinod",
        "user" : "vinod",
        "db" : "admin",
        "credentials" : {
                "SCRAM-SHA-1" : {
                        "iterationCount" : 10000,
                        "salt" : "2pCBM9hDZ+/Psiim4dbq8A==",
                        "storedKey" : "mIFWQwZ2nbl6+BN+vmjZFRuTSJ8=",
                        "serverKey" : "UWbMS5nV985TqYekBiwwYkGBwQs="
                },
                "SCRAM-SHA-256" : {
                        "iterationCount" : 15000,
                        "salt" : "UydhUIIqMiDQTbPRTY4ixCEdUdUOxqcd7UVskg==",
                        "storedKey" : "LxUyhNjWAmaT4rkzeqeCA1ebq/AwMx7l+e3I/nFlx40=",
                        "serverKey" : "hb4Nf77xrfipqVDOaoh8a9WKAA8BPn28untsIwUKra0="
                }
        },
        "roles" : [
                {
                        "role" : "readWrite",
                        "db" : "workdb1"
                },
                {
                        "role" : "read",
                        "db" : "workdb2"
                }
        ]
}
```

MongoDB’s built-in roles

- MongoDB provides built-in roles that provide the different levels of access commonly needed in a database system.
- Built-in database user roles and database administration roles exist in each database.
- The admin database contains additional roles.

Database User Roles

- read
  - Provides the ability to read data on all non-system collections and on the following system collections: system.indexes, system.js, and system.namespaces collections.
- readWrite
  - Provides all the privileges of the read role plus ability to modify data on all non-system collections and the system.js collection.

Database Administration Roles

- dbAdmin
  - Provides the ability to perform administrative tasks such as schema-related tasks, indexing, and gathering statistics.
- dbOwner
  - The database owner can perform any administrative action on the database. This role combines the privileges granted by the readWrite, dbAdmin and userAdmin roles.
- userAdmin
  - Provides the ability to create and modify roles and users on the current database.

Backup and Restoration Roles

- backup
  - Provides minimal privileges needed for backing up data. This role provides sufficient privileges to use the MongoDB Cloud Manager backup agent, Ops Manager backup agent, or to use mongodump to back up an entire mongod instance.
- restore
  - Provides privileges needed to restore data from backups that do not include system.profile collection data. This role is sufficient when restoring data with mongorestore without the --oplogReplay option.

All-Database Roles

- readAnyDatabase
- readWriteAnyDatabase
- userAdminAnyDatabase
- dbAdminAnyDatabase

Superuser Roles

- root
  - Provides access to the operations and all the resources of the readWriteAnyDatabase, dbAdminAnyDatabase, userAdminAnyDatabase, clusterAdmin, restore, and backup combined.

Custom Roles

- The db.createRole() function can be used for creating a custom role

```
db.createRole(
   { role: "changeOwnPasswordCustomDataRole",
     privileges: [
        {
          resource: { db: "", collection: ""},
          actions: [ "changeOwnPassword", "changeOwnCustomData" ]
        }
     ],
     roles: []
   }
)
```

https://docs.mongodb.com/manual/reference/privilege-actions/