1. Prepare the environment:
   1. Ensure that you have the necessary dependencies installed, such as tar, gzip, gnupg, and ssh (for remote server copy).
   2. Make sure you have the backup.sh, restore.sh, and backup\_restore\_lib.sh scripts in the same directory.
   3. Create an empty directory where you want to store the backup files and update the corresponding path in the scripts.
2. Create the test files:
   1. Run the create\_test\_files.sh script to generate the test files with different modification times in the current directory:
   2. >>$ bash create\_test\_files.sh
   3. This will create the test\_files directory with the required file structure and modification times.
3. Backup Script (backup.sh):
   1. Run the backup script backup.sh with the appropriate command-line parameters:
   2. >> $ bash backup.sh /path/to/source /path/to/backup "encryption\_key" 7
   3. Replace /path/to/source with the path to the test\_files directory created in Step 2.
   4. Replace /path/to/backup with the path to the directory where you want to store the backup files.
   5. Provide an appropriate encryption key for the backup.
   6. The 7 represents the number of days to consider for backing up modified files.
4. Verify the backup:
   1. Check the backup directory specified in Step 3 to ensure that the backup files are created.
   2. Confirm that each modified directory has a corresponding compressed tar file with the correct timestamp and extension .tgz.gpg.
   3. Additionally, check that all individual files are grouped into one tar.gz file and encrypted.
5. Restore Script (restore.sh):
   1. Run the restore script restore.sh with the appropriate command-line parameters:
   2. >> $ bash restore.sh /path/to/backup/backup\_directory /path/to/restore "decryption\_key"
   3. Replace /path/to/backup/backup\_directory with the path to the backup directory created in Step 3.
   4. Replace /path/to/restore with the path to the directory where you want to restore the backup files.
   5. Provide the decryption key used during the backup process.
6. Verify the restore:
   1. Check the restore directory specified in Step 5 to ensure that the restored files match the original files in the test\_files directory.
   2. Compare the timestamps and contents of the restored files with the original files to confirm they are identical.
7. Automate the backup using cron:
   1. Edit the cron configuration file using the appropriate command (crontab -e) and add an entry to schedule the backup script to run daily. For example:
   2. >> 0 0 \* \* \* /bin/bash /path/to/backup.sh /path/to/source /path/to/backup "encryption\_key" 7
   3. Adjust the paths and encryption key as necessary.

By following these steps, you can run the full scenario and test case, including creating test files, performing the backup, restoring the backup, and automating the backup process using cron.

Validation and Testing

1. Validation:
   1. Ensure that the backup and restore scripts (backup.sh and restore.sh) are properly using the modularized functions from backup\_restore\_lib.sh.
   2. Verify that all required parameters are being passed correctly to the scripts.
   3. Check if the scripts react to invalid parameters or missing directories/files by displaying appropriate error messages and exiting with a non-zero status code.
2. Unit Testing:
   1. Create a separate directory structure with test files and directories to simulate the backup and restore operations.
   2. Write test cases to cover various scenarios and edge cases, such as:
      1. Backup script:
         1. Validating parameter inputs.
         2. Creating the backup directory.
         3. Handling directories with modified files.
         4. Compressing and encrypting individual directories.
         5. Compressing and encrypting the overall backup file.
         6. Deleting the original files and directories.
         7. Copying the backup to a remote server (if applicable).
   3. Restore script:
      * 1. Validating parameter inputs.
        2. Creating the temporary restore directory.
        3. Decrypting and extracting individual directories.
        4. Cleaning up temporary files.
   4. Write test functions that call the backup and restore scripts with the appropriate parameters and assertions to check if the expected results are achieved.
   5. Run the test functions and ensure that all tests pass successfully.
3. Integration Testing:
   1. Perform end-to-end testing by running the backup script to create backups and the restore script to restore the backups.
   2. Use a separate test environment or backup and restore from a test backup directory to avoid any potential data loss or conflicts.
   3. Verify that the backup and restore operations complete successfully and the restored files match the original ones.
   4. Test scenarios such as:
      1. Backing up a directory with modified files.
      2. Restoring a backup to a different directory.
      3. Testing the remote server copy functionality (if applicable).
4. Error Handling Testing:
   1. Test error handling scenarios, such as:
      1. Providing incorrect or missing parameters.
      2. Specifying invalid directories or files.
      3. Testing the behavior when encryption keys are incorrect or missing.
      4. Verifying that appropriate error messages are displayed, and the scripts exit with non-zero status codes.