

Implementation Guide: Secure Credential Management with GPG and Ansible Vault

This guide will walk through the setup of the Ansible Secrets project, password (secret) encryption, the Ansible playbook for local deployment, and finally, the integration with Bash and Python scripts.

Placeholders Used in This Guide (REPLACE WITH YOUR ACTUAL VALUES):

- **Application Passwords:**

1. LDAP DM Password: `Ldap&DmP@sswOrd!2025`
2. LDAP RO Password: `LdapR0n1yP@sswOrd`
3. Oracle DB Password: `S3cure0racle!P@ss`

- **Single GPG Passphrase (to encrypt above passwords):**

`MyV3ryStr0ngGPGPassphr@s3`

- **Ansible Vault Password (to protect the GPG passphrase):**

`MyUltr@S3cureAnsib13VaultP@ss`

- **Project & Secret Directories:**

1. Ansible Project: `/opt/ansible_secrets`
2. Deployed Secrets: `/opt/credential_store`

- **Users & Groups:**

1. Service User: `service_account`
2. Access Group: `appsecretaccess`
3. Admin User (you): `flengye1`

Section 1: Initial Server Setup

These steps prepare the server environment with the necessary users, groups, and software.

1.1. Create Users and Groups

Run these commands on your RHEL server as a user with `sudo` privileges.

```
# Create the dedicated service user (these are EAD
accounts, but I suggest creating a local workstation
account)
sudo useradd --system --shell /sbin/nologin --comment
"Service account for Bash and Python apps" service_account

# Create the dedicated access group
sudo groupadd --system appsecretaccess

# Add the service user to the access group
sudo usermod -aG appsecretaccess service_account

# Add yourself and any other required users to the access
group
sudo usermod -aG appsecretaccess flengyel
# sudo usermod -aG appsecretaccess otheruser1

# IMPORTANT: Any user you add to the group must log out
and log back in
# for their new group membership to take effect.
```

1.2. Install Required Software

```
sudo dnf install ansible-core gnupg2 -y
```

1.3. Set Up Python Virtual Environment & Project Directory

This isolates your Ansible installation.

```
# Create and take ownership of the Ansible project
directory
sudo mkdir -p /opt/ansible_secrets
sudo chown 'flengyel:domain users' /opt/ansible_secrets
cd /opt/ansible_secrets

# Create a Python virtual environment inside the project
directory
python3 -m venv venv

# Activate the virtual environment
source venv/bin/activate

# Your shell prompt should now start with "(venv)".
# Install Ansible and the GPG library into the active
venv.
pip install ansible-core gnupg
```

Section 2: Credential and Ansible Vault Preparation

Now we'll prepare the encrypted secrets and the Ansible Vault to manage their decryption key.

2.1. Encrypt Your Application Passwords with GPG

This is a one-time setup step for each password.

```
# Ensure you are in your project directory and the venv is active
cd /opt/ansible_secrets

# Create a subdirectory for the GPG files
mkdir -p files
cd files

# Create the plaintext password files
printf 'Green&DmP@swd!2025' > green_dm_secret.txt
printf 'Yellow&DmP@swd!2025' > yellow_dm_secret.txt
printf 'LdapR0nlyP@sswOrd' > ldap_ro_secret.txt
printf 'S3cureOracle!P@ss' > oracle_db_secret.txt

# Encrypt all three files using the SAME GPG passphrase
GPG_PASSPHRASE='MyV3ryStr0ngGPGPassphr@s3'
for f in *.txt; do
    gpg --batch --yes --symmetric --cipher-algo AES256 \
        --passphrase "$GPG_PASSPHRASE" "$f"
done
```

```
# Verify the .gpg files were created
ls -l *.gpg
```

```
# Securely delete the plaintext files
shred --remove *.txt
cd ..
```

You now have `green_dm_secret.txt.gpg` ,
`yellow_dm_secret.txt.gpg` , `ldap_ro_secret.txt.gpg` , and
`oracle_db_secret.txt.gpg` in your `files/` subdirectory.

2.2. Prepare the Ansible Vault

```
# Ensure you are in the project root
(/opt/ansible_secrets)
# Create the vault password file
echo "MyUltraS3cureAnsibl3VaultP@ss" >
.ansible_vault_password
chmod 600 .ansible_vault_password

# Create the encrypted vault file to hold the GPG
passphrase
mkdir -p group_vars/all
ansible-vault create group_vars/all/vault.yml
```

An editor will open. Enter the following content (this is your single GPG passphrase):

```
app_gpg_passphrase: "MyV3ryStr0ngGPGP@ssphr@s3"
```

Save and close the file. It is now encrypted.

Section 3: Ansible Configuration and Playbook

3.1. Configure Ansible (`ansible.cfg` and `inventory`)

- Create `/opt/ansible_secrets/ansible.cfg` :

```
[defaults]
inventory = ./inventory
vault_password_file = ../ansible_vault_password
host_key_checking = False

[privilege_escalation]
become = true
become_method = sudo
become_user = root
become_ask_pass = true
```

- Create `/opt/ansible_secrets/inventory` :

```
[local_server]
localhost ansible_connection=local
ansible_python_interpreter={{ ansible_playbook_python }}
```

3.2. Create the Ansible Playbook (`deploy_secrets.yml`)

Create `/opt/ansible_secrets/deploy_secrets.yml` :

```

- name: Deploy Application Secrets Locally
  hosts: local_server
  vars:
    secrets_target_dir: "/opt/credential_store"
    service_user: "service_account"
    secret_access_group: "appsecretaccess"
    encrypted_secret_files:
      - green_dm_secret.txt.gpg
      - yellow_dm_secret.txt.gpg
      - ldap_ro_secret.txt.gpg
      - oracle_db_secret.txt.gpg
  vars_files:
    - group_vars/all/vault.yml

  tasks:
    - name: Ensure base directories and groups are set up
      ansible.builtin.include_tasks: tasks/setup.yml

    - name: Deploy the single GPG passphrase file from
      vaulted variable
      ansible.builtin.copy:
        content: "{{ app_gpg_passphrase }}"
        dest: "{{ secrets_target_dir }}/.gpg_passphrase"
        owner: "{{ service_user }}"
        group: "{{ secret_access_group }}"
        mode: '0440'
      no_log: true

    - name: Deploy all encrypted application password
      files
      ansible.builtin.copy:
        src: "./files/{{ item }}" # From the project's

```

```
files/ dir
    dest: "{{ secrets_target_dir }}/{{ item }}"
    owner: "{{ service_user }}"
    group: "{{ secret_access_group }}"
    mode: '0440'
    with_items: "{{ encrypted_secret_files }}"
```

Create a supporting task file for clarity, `tasks/setup.yml`:

```
mkdir -p /opt/ansible_secrets/tasks
```

Create `/opt/ansible_secrets/tasks/setup.yml`:

```
- name: Ensure the secret access group exists
  ansible.builtin.group:
    name: "{{ secret_access_group }}"
    state: present
    system: true

- name: Ensure service user is part of the secret access
group
  ansible.builtin.user:
    name: "{{ service_user }}"
    groups: "{{ secret_access_group }}"
    append: true

- name: Ensure secrets target directory exists with
correct permissions
  ansible.builtin.file:
    path: "{{ secrets_target_dir }}"
    state: directory
```



```
owner: "{{ service_user }}"
group: "{{ secret_access_group }}"
mode: '0750'
```

Section 4: Deployment

4.1. Run the Ansible Playbook

```
# Ensure you are in the project root and your venv is
active
cd /opt/ansible_secrets
source venv/bin/activate

# Run the playbook
ansible-playbook deploy_secrets.yml
```

4.2. Verify the Deployment

After the playbook runs successfully, check the deployed secrets directory.

```
sudo ls -lA /opt/credential_store/
```

The output should look like this (owner, group, permissions, and files must match):

```
total 12
-r--r----- 1 service_account appsecretaccess 111 Jun 07
08:44 .gpg_passphrase
```

```
-r--r----- 1 service_account appsecretaccess 1408 Jun 07
08:44 ldap_dm_secret.txt.gpg
-r--r----- 1 service_account appsecretaccess 1408 Jun 07
08:44 ldap_ro_secret.txt.gpg
-r--r----- 1 service_account appsecretaccess 1408 Jun 07
08:44 oracle_db_secret.txt.gpg
```

Section 5: Script Integration and Runtime Operation

Here is how your scripts can securely access the passwords. It's best to create a reusable function or helper script.

5.1. Reusable Bash Script (`get_secret.sh`)

This script will take a "secret name" (like `oracle_db`) and print its password to standard output.

- Create `/usr/local/bin/get_secret.sh` :

```
#!/usr/bin/env bash
set -euo pipefail

if [[ $# -ne 1 ]]; then
    echo "Usage: $0 <secret_name>" >&2
    echo "Example: $0 oracle_db" >&2
    exit 1
fi

SECRET_NAME="$1"
```

```

SECRETS_DIR="/opt/credential_store"
ENC_FILE="${SECRETS_DIR}/${SECRET_NAME}_secret.txt.gpg"
GPG_PASSPHRASE_FILE="${SECRETS_DIR}/.gpg_passphrase"

if [[ ! -r "$ENC_FILE" ]]; then
    echo "Error: Encrypted secret for '${SECRET_NAME}' not
found or not readable." >&2
    exit 1
fi

# Decrypt and print the password to stdout
gpg --batch --quiet --yes \
    --passphrase-file "$GPG_PASSPHRASE_FILE" \
    --decrypt "$ENC_FILE" 2>/dev/null

```

- Set its permissions:

```

sudo chown service_account:appsecretaccess
/usr/local/bin/get_secret.sh
sudo chmod 0750 /usr/local/bin/get_secret.sh # Owner rwx,
Group rx

```

- **How to use it in another Bash script:**

```

#!/bin/bash
# Get the Oracle password into a variable
ORACLE_PASS=$(/usr/local/bin/get_secret.sh oracle_db)
if [[ -z "$ORACLE_PASS" ]]; then
    echo "Failed to retrieve Oracle password." >&2
    exit 1

```

```
fi
```

```
echo "Successfully retrieved Oracle password of length  
${#ORACLE_PASS}"  
# Now use $ORACLE_PASS to connect to the database  
# ...  
unset ORACLE_PASS # Clean up
```

5.2. Reusable Python Module (`secret_retriever.py`)

This module provides a function to get secrets.

- Create

```
/usr/local/lib/ansible_secret_helpers/secret_retriever.py  
(ensure /usr/local/lib/ansible_secret_helpers is in your  
PYTHONPATH or your script is in the same dir).
```

```
#  
/usr/local/lib/ansible_secret_helpers/secret_retriever.py  
import os  
import subprocess  
  
SECRETS_DIR = "/opt/credential_store"  
GPG_PASSPHRASE_FILE = os.path.join(SECRETS_DIR,  
".gpg_passphrase")  
  
def get_password(secret_name: str) -> str:  
    """  
    Retrieves a decrypted password for a given secret  
    name.  
    Raises RuntimeError on failure.
```

```

"""
    enc_file = os.path.join(SECRETS_DIR, f"
{secret_name}_secret.txt.gpg")
    if not os.path.exists(enc_file):
        raise FileNotFoundError(f"Encrypted secret for
'{secret_name}' not found.")

    cmd = [
        "gpg", "--batch", "--quiet", "--yes",
        "--passphrase-file", GPG_PASSPHRASE_FILE,
        "--decrypt", enc_file
    ]

    try:
        result = subprocess.run(
            cmd, stdout=subprocess.PIPE,
stderr=subprocess.PIPE,
            universal_newlines=True, check=True
        )
        return result.stdout.strip()
    except subprocess.CalledProcessError as e:
        raise RuntimeError(f"GPG decryption failed for
'{secret_name}': {e.stderr}")
    except FileNotFoundError:
        raise RuntimeError("gpg command not found. Is
GnuPG installed?")

```

- Set its permissions:

```
sudo mkdir -p /usr/local/lib/ansible_secret_helpers
sudo chown service_account:appsecretaccess
/usr/local/lib/ansible_secret_helpers/secret_retriever.py
sudo chmod 0640
/usr/local/lib/ansible_secret_helpers/secret_retriever.py
# Owner rw, Group r
```

- **How to use it in another Python script:**

How to use it in another Python script (Improved Example):

This example demonstrates a more robust integration pattern based on a real-world script. It shows how to retrieve a secret and use it to connect to a database within a structured application.

```
#!/usr/bin/env python3
import csv
import sqlalchemy
import cx_Oracle as cx
import argparse as arg
import sys
import os

# --- Start: Required code block for secret retrieval ---
# Define the path to the helper library.
HELPER_LIB_PATH = "/usr/local/lib/ansible_secret_helpers"

# Add the path to the system path list so Python can find
the module.
# This makes the script work reliably from anywhere
```

```

(including cron).
if HELPER_LIB_PATH not in sys.path:
    sys.path.append(HELPER_LIB_PATH)

try:
    # Now that the path is set, the import will succeed.
    import secret_retriever
except ImportError:
    print(f"CRITICAL ERROR: Could not import
'secret_retriever'.", file=sys.stderr)
    print(f"Ensure '{HELPER_LIB_PATH}' exists and is
readable.", file=sys.stderr)
    sys.exit(1)
# --- End: Required code block ---

def create_secure_connection(dbhost, dbport, dbsid,
secret_name, user):
    """
    Retrieves a password from the credential store and
creates a DB connection.
    Returns a tuple of (engine, connection).
    """
    db_password = None
    engine = None
    conn = None
    try:
        # Retrieve the specified password using the helper
function
        db_password =
secret_retriever.get_password(secret_name)
        if not db_password:

```

```

        raise RuntimeError(f"Retrieved empty password
for '{secret_name}'")

    datasource_name = cx.makedsn(dbhost, dbport,
service_name=dbsid)
    connectstring = f'oracle+cx_oracle://{user}:
{db_password}@{datasource_name}'

    # Clear the plaintext password from memory as soon
as it's used
    db_password = None

    engine = sqlalchemy.create_engine(connectstring,
max_identifier_length=128)
    conn = engine.connect()
    return engine, conn

except Exception as e:
    print(f"Error creating database connection: {e}",
file=sys.stderr)
    # Ensure resources are cleaned up on failure
    if conn:
        conn.close()
    if engine:
        engine.dispose()
    sys.exit(1) # Exit with an error code

def main():
    """Main execution logic"""
    parser = arg.ArgumentParser(description="Check
password reset status for a given EMPLID.")

```



```
    parser.add_argument('emplid', type=str, help='Search
for a specific EMPLID.')
    args = parser.parse_args()

    dbhost = 'IAMPRDDB1.cuny.edu'
    dbport = '2483'
    dbsid = 'PDIMOIG_HUD'
    db_user = 'flengyel' # This username could also be
stored as a secret
    secret_name_for_db = 'oig_db' # The name of the secret
to fetch

    engine, conn = None, None
    try:
        # Establish the secure connection
        engine, conn = create_secure_connection(dbhost,
dbport, dbsid, secret_name_for_db, db_user)
        print(f"Successfully connected to {dbsid} as
{db_user}.")

        # --- Your application logic starts here ---
        where_clause = f" WHERE U.USR_EMP_NO =
'{args.emplid}'"
        count_statement = 'SELECT COUNT(*) FROM
GREEN_OIM.PWH P INNER JOIN GREEN_OIM.USR U ON P.USR_KEY =
U.USR_KEY' + where_clause

        count_result =
conn.execute(count_statement).scalar()
        if count_result == 0:
            print(f"User {args.emplid} must reset their
password.")
```

```
        else:
            print(f"OIM can sync password for
{args.emplid}")

        # ... add other queries here ...

    finally:
        # Ensure the connection is always closed
        if conn:
            conn.close()
        if engine:
            engine.dispose()
        print("Execution finished. Database connection
closed.")

if __name__ == "__main__":
    main()
```

Section 6: App script ownership and permissions

This is the recommended ownership and permission model for production Python and bash scripts:

- Owner: service_account
- Group: appsecretaccess
- Permissions: 0750 (-rwxr-x---)

Here are the ownership and permission mode commands for scripts used with Ansible Secrets. The script in this example is `getemplid.sh`, however, the commands below apply to Python scripts as well.

```
sudo chown service_account:appsecretaccess getemplid.sh
sudo chmod 0750 getemplid.sh
```