Dynamic PKI-Enabled File Encryption System

Overview

This project implements a 1. **PKI -> Public Key infrastructure** 2. **PKI-enabled file encryption and decryption system** with automated directory monitoring and **file sharing** capabilities. The system is designed to facilitate secure file sharing, authentication, and encryption in a networked environment, using certificates issued by a Root CA and signed by an Intermediate CA.

Features

1. **PKI Framework**:
    - Root CA initializes the chain of trust and generates Intermediate CA credentials.
    - Intermediate CA acts as the signing authority for host certificates.
2. **Host Certificates**:
    - Hosts generate unique certificate signing requests (CSRs) and obtain signed certificates from the Intermediate CA.
    - Signed certificates are installed with the Intermediate CA and Root CA certificates to complete the chain of trust.
3. **File Encryption and Decryption**:
    - Encrypts files dynamically using public keys from a shared public key store.
    - Decrypts encrypted files when accessed and re-encrypts them after closure.
4. **File Sharing**:
    - Hosts can share files over the network using a lightweight file-sharing server.
    - Includes support for file uploads and downloads within the same network.

Setup Instructions

1. Root CA Setup

Files and Structure

- All files outside the users@hosts directory belong to the Root CA.

Steps

1. Transfer the Root CA files to the server machine (Windows/Linux).
2. Install the required libraries:

   python -m pip install -r requirements.txt

   or simply:

   pip install -r requirements.txt

3. Initialize the Root CA:

   python AutoRootCA_PKI.py

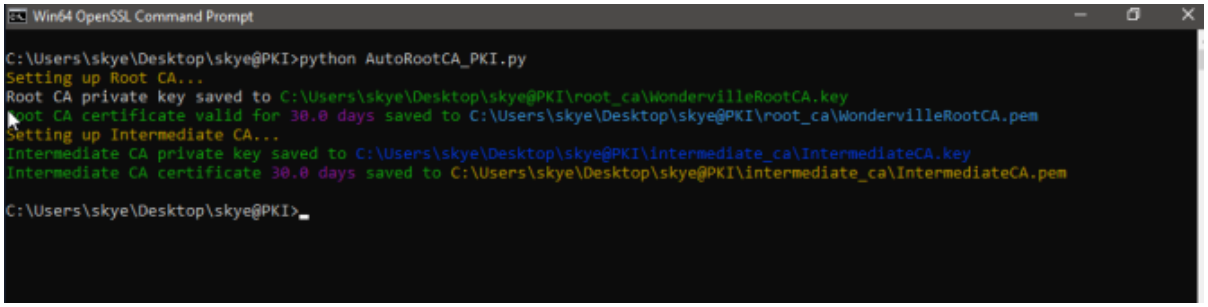

Illustration 1:

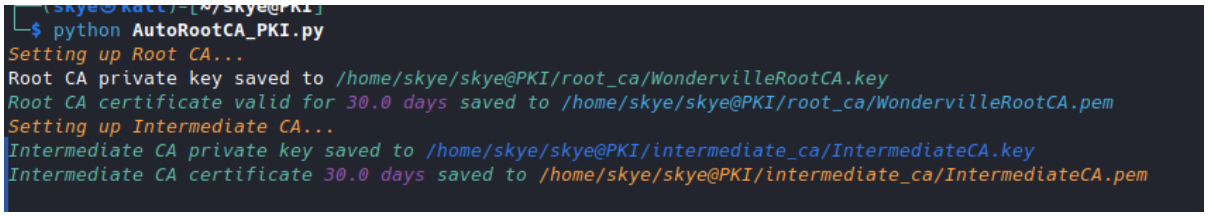

Illustration 2:

   This script generates the Root CA keys and sets up the Intermediate CA.

4. Start the Intermediate CA server:

   python WIN-N55Q3T15CyEL4_server.py

```
└─$ python WIN-N55Q3T15CyEL4_server.py
 * Serving Flask app 'WIN-N55Q3T15CyEL4_server'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on https://127.0.0.1:5000
 * Running on https://172.17.88.189:5000
Press CTRL+C to quit
```

Illustration 3:

or :

gunicorn -c gunicorn.conf.py WIN-N55Q3T15CyEL4_server:app

```
└─$ gunicorn -c gunicorn.conf.py  WIN-N55Q3T15CyEL4_server:app
[2024-11-25 00:56:28 +0300] [20837] [INFO] Starting gunicorn 23.0.0
[2024-11-25 00:56:28 +0300] [20837] [INFO] Listening at: http://0.0.0.0:5000 (20837)
[2024-11-25 00:56:28 +0300] [20837] [INFO] Using worker: gthread
[2024-11-25 00:56:28 +0300] [20838] [INFO] Booting worker with pid: 20838
[2024-11-25 00:56:28 +0300] [20839] [INFO] Booting worker with pid: 20839
[2024-11-25 00:56:28 +0300] [20840] [INFO] Booting worker with pid: 20840
[2024-11-25 00:56:28 +0300] [20841] [INFO] Booting worker with pid: 20841
```

Illustration 4:

- This starts a PKI server where hosts/users can connect to obtain, sign, or update their certificates.

---

2. Host/User Setup

Files and Structure

- Host assets are located in the users@hosts directory.
- These files should be transferred to the user's computer for local operations.

Steps

1. Transfer the host files to the user's computer.
2. Install the required libraries:

   python -m pip install -r requirements.txt

   or simply:

   pip install -r requirements.txt

3. You need to install openssl and add it to system path
    1. On windows Run: python opensslSetup.py located in the user@host folder
    2. Or manualy download and install the executable (You can find it in the user@host/src folder
4. Generate Certificates:

    python AutoCertsGen.py

    Require Administrator(Root) privileges see:

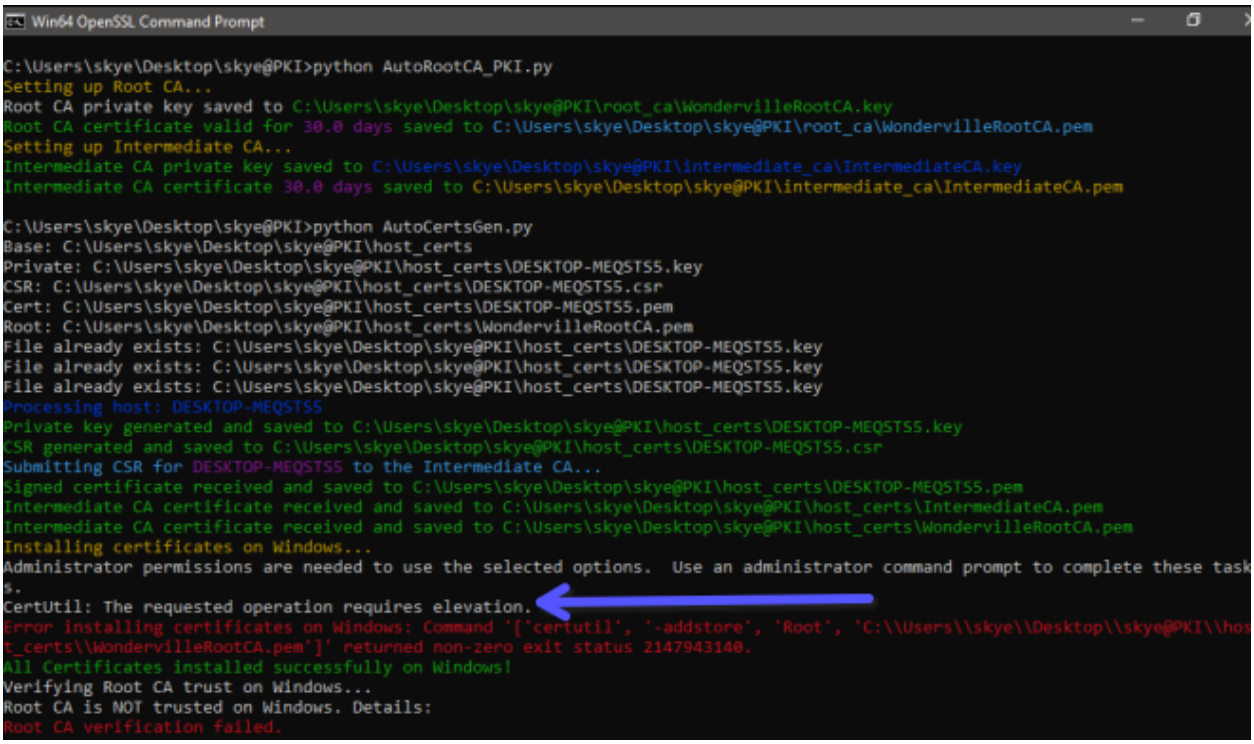

Illustration 5:

    - This script generates the host's unique certificate and requests signing from the Intermediate CA.
    - The Intermediate CA:
        - Signs the certificate.
        - Provides its own certificate and that of the Root CA.
    - These certificates are installed locally, completing the chain of trust.
5. Verify Certificates:

```
C:\Users\skye\Desktop\skye@PKI>certutil -verify -urlfetch  C:\Users\skye\Desktop\skye@PKI\host_certs\WondervilleRootCA.pem
Issuer:
    CN=Wonderville Root CA
    O=Wonderville Town Hall
    L=Wonderville
    S=New York
    C=US
  Name Hash(sha1): 0249019ff2c173e04b111f6a18da78e2a8ee5eb3
  Name Hash(md5): 0f384f23dac29e8403ee8ace03092c15
Subject:
    CN=Wonderville Root CA
    O=Wonderville Town Hall
    L=Wonderville
    S=New York
    C=US
  Name Hash(sha1): 0249019ff2c173e04b111f6a18da78e2a8ee5eb3
  Name Hash(md5): 0f384f23dac29e8403ee8ace03092c15
Cert Serial Number: 1a9585cfb43f0165b3b88c008b9027357f504bfa

dwFlags = CA_VERIFY_FLAGS_CONSOLE_TRACE (0x20000000)
dwFlags = CA_VERIFY_FLAGS_DUMP_CHAIN (0x40000000)
ChainFlags = CERT_CHAIN_REVOCATION_CHECK_CHAIN_EXCLUDE_ROOT (0x40000000)
HCCE_LOCAL_MACHINE
CERT_CHAIN_POLICY_BASE
-------- CERT_CHAIN_CONTEXT --------
ChainContext.dwInfoStatus = CERT_TRUST_HAS_PREFERRED_ISSUER (0x100)

SimpleChain.dwInfoStatus = CERT_TRUST_HAS_PREFERRED_ISSUER (0x100)

CertContext[0][0]: dwInfoStatus=10c dwErrorStatus=0
  Issuer: CN=Wonderville Root CA, O=Wonderville Town Hall, L=Wonderville, S=New York, C=US
  NotBefore: 11/23/2024 6:17 AM
  NotAfter: 12/23/2024 6:17 AM
  Subject: CN=Wonderville Root CA, O=Wonderville Town Hall, L=Wonderville, S=New York, C=US
  Serial: 1a9585cfb43f0165b3b88c008b9027357f504bfa
  Cert: 450257c638ca84b67d166bc95dcb0da147a53586
  Element.dwInfoStatus = CERT_TRUST_HAS_NAME_MATCH_ISSUER (0x4)
  Element.dwInfoStatus = CERT_TRUST_IS_SELF_SIGNED (0x8)
  Element.dwInfoStatus = CERT_TRUST_HAS_PREFERRED_ISSUER (0x100)
  ----------------  Certificate AIA  ----------------
  No URLs "None" Time: 0 (null)
  ----------------  Certificate CDP  ----------------
  No URLs "None" Time: 0 (null)
  ----------------  Certificate OCSP  ----------------
  No URLs "None" Time: 0 (null)
  ----------------------------------

Exclude leaf cert:
```

Illustration 6:

```
  Name Hash(sha1): 0249019ff2c173e04b111f6a18da78e2a8ee5eb3
  Name Hash(md5): 0f384f23dac29e8403ee8ace03092c15
Cert Serial Number: 1a9585cfb43f0165b3b88c008b9027357f504bfa

dwFlags = CA_VERIFY_FLAGS_CONSOLE_TRACE (0x20000000)
dwFlags = CA_VERIFY_FLAGS_DUMP_CHAIN (0x40000000)
ChainFlags = CERT_CHAIN_REVOCATION_CHECK_CHAIN_EXCLUDE_ROOT (0x40000000)
HCCE_LOCAL_MACHINE
CERT_CHAIN_POLICY_BASE
-------- CERT_CHAIN_CONTEXT --------
ChainContext.dwInfoStatus = CERT_TRUST_HAS_PREFERRED_ISSUER (0x100)

SimpleChain.dwInfoStatus = CERT_TRUST_HAS_PREFERRED_ISSUER (0x100)

CertContext[0][0]: dwInfoStatus=10c dwErrorStatus=0
  Issuer: CN=Wonderville Root CA, O=Wonderville Town Hall, L=Wonderville, S=New York, C=US
  NotBefore: 11/23/2024 6:17 AM
  NotAfter: 12/23/2024 6:17 AM
  Subject: CN=Wonderville Root CA, O=Wonderville Town Hall, L=Wonderville, S=New York, C=US
  Serial: 1a9585cfb43f0165b3b88c008b9027357f504bfa
  Cert: 450257c638ca84b67d166bc95dcb0da147a53586
  Element.dwInfoStatus = CERT_TRUST_HAS_NAME_MATCH_ISSUER (0x4)
  Element.dwInfoStatus = CERT_TRUST_IS_SELF_SIGNED (0x8)
  Element.dwInfoStatus = CERT_TRUST_HAS_PREFERRED_ISSUER (0x100)
  ----------------  Certificate AIA  ----------------
  No URLs "None" Time: 0 (null)
  ----------------  Certificate CDP  ----------------
  No URLs "None" Time: 0 (null)
  ----------------  Certificate OCSP  ----------------
  No URLs "None" Time: 0 (null)
  ----------------------------------

Exclude leaf cert:
  Chain: da39a3ee5e6b4b0d3255bfef95601890afd80709
Full chain:
  Chain: 450257c638ca84b67d166bc95dcb0da147a53586
------------------------------------
Verified Issuance Policies: All
Verified Application Policies: All
Cert is a CA certificate
Cannot check leaf certificate revocation status
CertUtil: -verify command completed successfully.
```

Illustration 7:

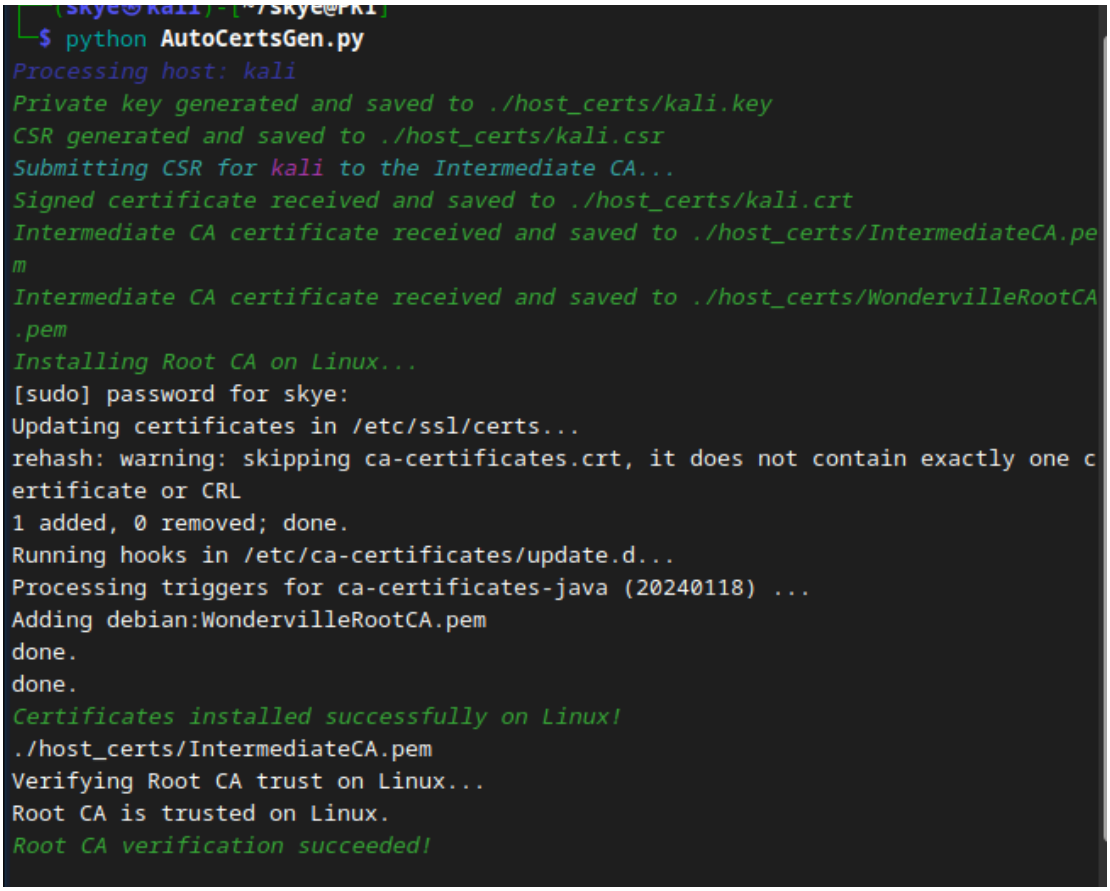

Illustration 8:

- Once installed, the host verifies the authenticity of the Intermediate CA certificate.

---

3. File Sharing

Steps

1. Start the File Sharing Server:

   gunicorn -c gunicorn.conf.py FileShareProServer:app

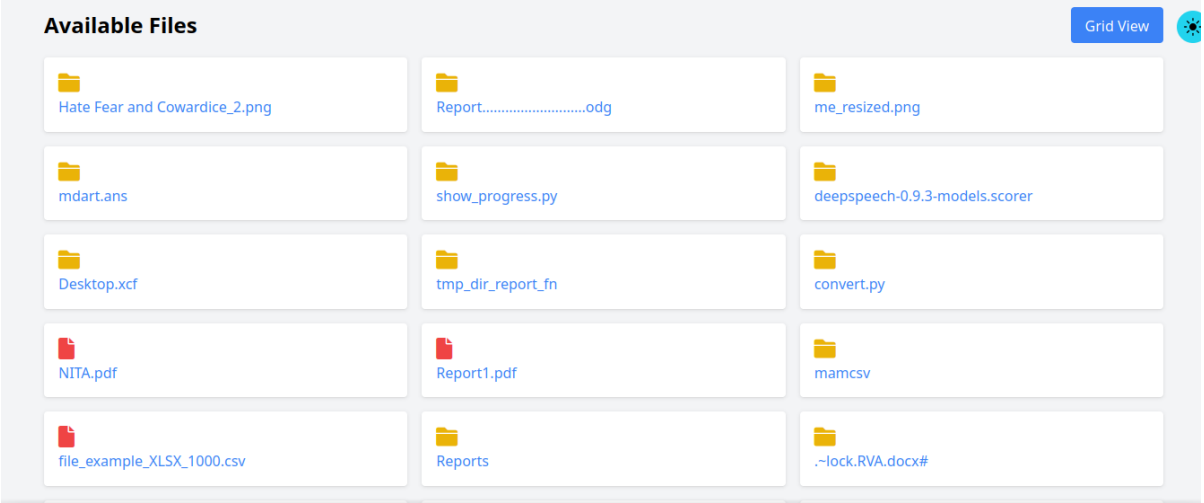

Illustration 9:

python FileShareProServer.py **NOT RECOMMENDED but still works fine**

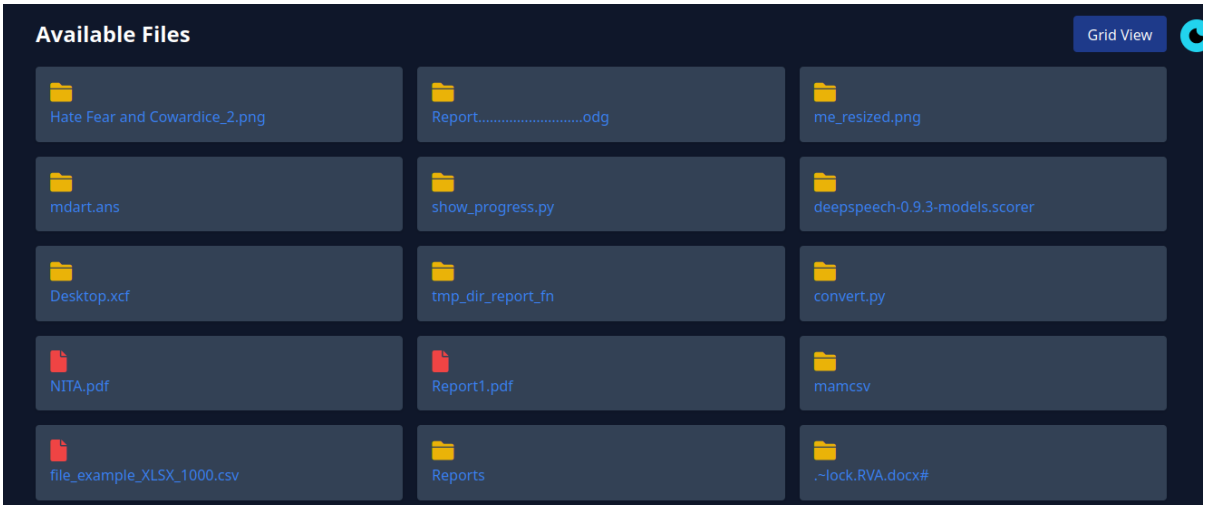

Illustration 10:

- This file is located in the App directory.

2. Configure the Server:
   - Open the .ini file and:
     - Set up folder-sharing options.
     - Define the download location.
3. Access the File Sharing Server:
   - Members of the same network can:

- Access the server using its IP address.
- Download files directly.
- Upload files to the host.

Advanced Usage

File Encryption and Decryption(Experimental)

==========**CAREFUL BELOW**==========

---

1. **Encrypt Files**:
    - Automatically encrypts files in the monitored directory.
    - Uses the recipient's public key, fetched dynamically from the public key store.
2. **Decrypt Files**:
    - Decrypts files when accessed, saving a temporary plaintext version.
    - Opens the decrypted file using the system's default application.
    - Re-encrypts the file after closure.
3. **File Monitoring**:
    - Uses the watchdog library to monitor file creation and modification events.

Public Key Store

- The public key store (PUBLIC_KEY_STORE) contains public keys for all machines in the network.
- Public keys are named after their respective machine identities (e.g., machine_name.pem).

Directory Monitoring Setup

Configuration

Edit the following variables in the script as needed: - **WATCHED_DIR**: - Path to the monitored directory (e.g., /mnt/shared_folder or \\server_ip\shared_folder). - **PUBLIC_KEY_STORE**: - Path to the directory containing public keys. -

PRIVATE_KEY_PATH: - Path to the host's private key file. -
ENCRYPTED_EXTENSION: - File extension for encrypted files (default: .enc).

Run the Script

1. Mount the shared folder:
    - **Unix/Linux**:

      sudo mount -t cifs -o username=<username>,password=<password> //<server>/<shared_folder> /mnt/shared_folder

    - **Windows**:

      net use X: \\<server>\<shared_folder> /user:<username> <password>

2. Start monitoring:

   python PKI_crypto.py monitor

Logging

Log File

- All activities are logged in file_activity.log:
    - Encryption and decryption events.
    - Errors and warnings.

Sample Log Output

2024-11-23 14:10:00 - INFO - Watching directory: /mnt/shared_folder
2024-11-23 14:12:01 - INFO - File encrypted: /mnt/shared_folder/document.txt.enc
2024-11-23 14:14:45 - INFO - Decrypted file saved temporarily at /tmp/document.txt
2024-11-23 14:15:12 - INFO - Temporary file /tmp/document.txt deleted securely.
2024-11-23 14:15:13 - INFO - File encrypted: /mnt/shared_folder/document.txt.enc

---

Troubleshooting

File Not Encrypted

- Ensure the file doesn't already have the .enc extension.
- Verify the public key exists in the PUBLIC_KEY_STORE.

Certificate Issues

- Verify the chain of trust by checking the Root CA and Intermediate CA certificates.
- Ensure the Intermediate CA server is running and accessible.

Shared Folder Not Accessible

- Check network connectivity and permissions.
- Ensure the shared folder is mounted properly.

Future Improvements

1. **Key Management**:
    - Automate updates to the public key store via an API or centralized server.
2. **Access Control**:
    - Implement authentication for shared folder access.
    - Restrict decryption based on roles.
3. **File Integrity**:
    - Add digital signatures to verify file authenticity before decryption.
4. **GUI**:
    - Develop a graphical interface for easier management of encryption and file sharing.

Contributors

- **[Wambua]** – Project Lead Follow Me on github
- Interrested in encryption -> https://github.com/skye-cyber/EncryptsionSuite

---