Title Page

Project Title: DynamicEncrypt – A Secure File Vault with Pluggable Crypto Modules
Student Names: Aryan Dani, Aarush Shah, Sobaan Jagirdar
PRN: 1272250888, 1272250890, 1272250891
Batch : A-3
Subject Name: Project Based Learning
Course: BTECH CSE AI&DS
Faculty Name: Swati Jadhav
Submission Date: November 8, 2025

Abstract

This report presents the design and development of "DynamicEncrypt," a C++ desktop application that provides secure local file encryption and decryption. The system is built using the Qt 6 framework and features a dynamic plugin architecture that allows cryptographic algorithms to be loaded at runtime without modifying the core application. The project demonstrates key concepts of modern C++ such as RAII for secure memory cleanup, templates for type-safe key handling, and polymorphism for extensible plugin loading. The resulting application is a robust and user-friendly file vault designed to meet evolving security requirements.

Objectives

- Develop a secure file vault application using C++ and the Qt 6 framework.
- Implement a dynamic plugin system that enables flexible, extensible cryptographic algorithms.

- Demonstrate modern C++ features including RAII, templates, smart pointers, and polymorphism.
- Build a clean, user-friendly GUI for encryption, decryption, and key management.
- Ensure secure memory handling, atomic file I/O operations, and long-term adaptability.

Problem Definition

Storing sensitive personal or professional data on local machines without encryption exposes users to privacy risks, data theft, and unauthorized access. Operating systems provide basic file restrictions, but these are insufficient against determined attackers. Furthermore, cryptographic standards frequently evolve, making it necessary for secure applications to stay adaptable.

DynamicEncrypt addresses this problem by providing strong, local encryption with the added flexibility of pluggable cryptographic modules. Users are not restricted to one algorithm. Instead, the application allows new algorithms to be integrated easily, ensuring long-term security, maintainability, and adaptability.

System Requirements and Build Process

System Requirements

- Operating System: Windows 10/11
- Compiler: C++20-compliant compiler (MinGW included in Qt installation)

- Build System: CMake 3.21+
- Framework: Qt 6.1.0 or newer

Build Steps

Clone the Repository

```
git clone https://github.com/aryan-dani/Dynamic_Encrypt.git
cd Dynamic_Encrypt
```

Configure with CMake

Run from the project root:

```
cmake -S . -B build -G "MinGW Makefiles"
-DCMAKE_PREFIX_PATH="C:/Qt/6.10.0/mingw_64"
```

(Ensure that the `CMAKE_PREFIX_PATH` matches your Qt installation directory.)

Build the Project

```
cmake --build build
```

Running the Application

Use the provided batch script for correct environment setup:

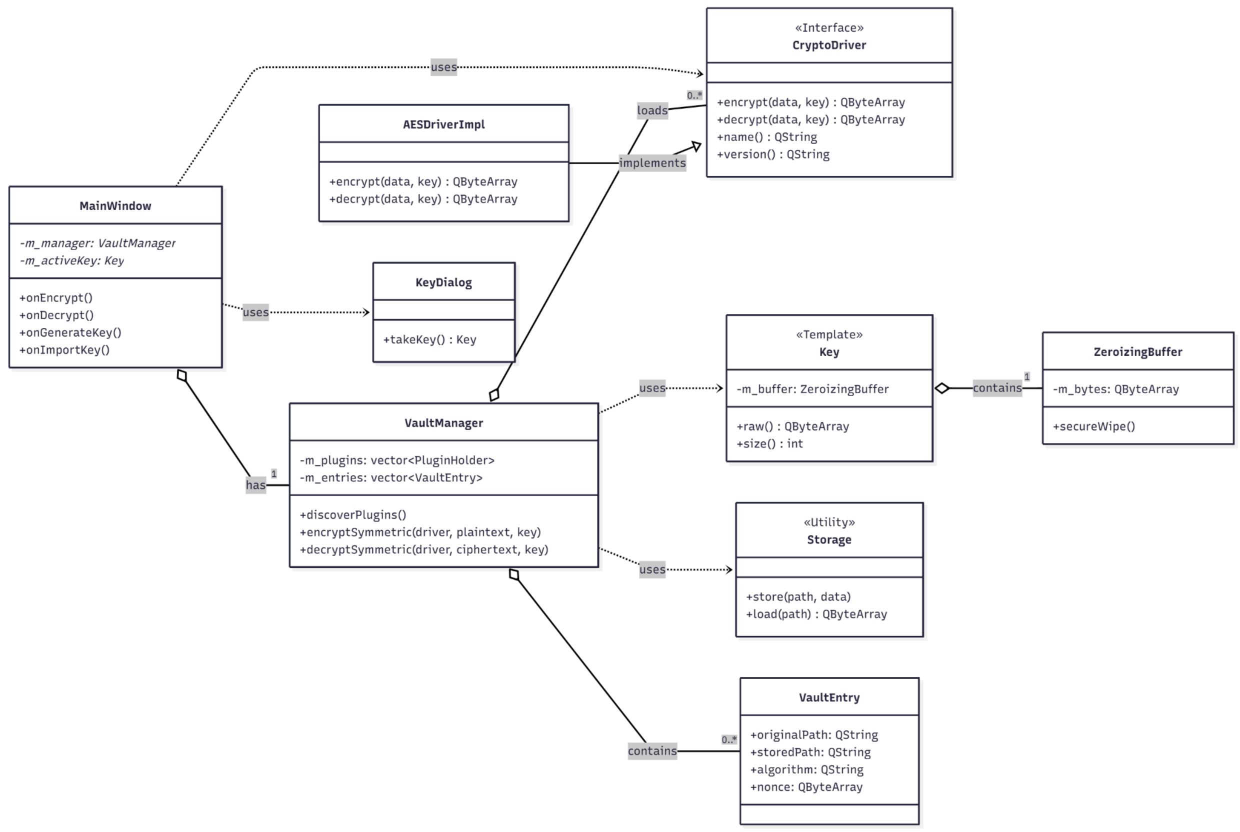
```
.\run_app.bat
```

Running Tests

To execute unit tests:

```
.\build\bin\core_tests.exe
```

System Design / Class Diagram



This diagram should include the following major components and relationships:

- VaultManager
- CryptoDriver (Interface)
- AESDriverImpl
- Key<KeyTag>

- ZeroizingBuffer
- Storage
- VaultEntry
- MainWindow

You can create this diagram using tools like draw.io or Lucidchart.

Implementation Details

The system consists of three major parts: the core backend engine, the plugin architecture, and the Qt-based graphical interface.

---

1. VaultManager – Core Engine

VaultManager serves as the heart of the application. It loads cryptographic plugins at runtime using QPluginLoader, manages encrypted file metadata, and provides high-level encryption and decryption functions for the GUI.

It handles plugin discovery, plugin validation, and the orchestration of file operations.

---

2. CryptoDriver – Plugin Interface

CryptoDriver defines the abstract interface that all encryption modules must implement. It includes pure virtual functions such as:

- `encrypt()`

- `decrypt()`
- `name()`
- `version()`

Qt's `Q_DECLARE_INTERFACE` macro allows the plugin system to recognize classes implementing this interface. This ensures that the application can load any compatible encryption module at runtime.

---

3. Key Wrapper – Type-Safe Key Management

The `Key<KeyTag>` template class enforces compile-time type safety. Tags such as `SymmetricKeyTag` and `AsymmetricKeyTag` prevent incorrect key usage.

Keys are stored inside a `ZeroizingBuffer`, ensuring that sensitive material is securely erased from memory upon destruction.

---

4. ZeroizingBuffer – Secure Memory Handling

ZeroizingBuffer is responsible for secure in-memory storage of cryptographic material. Its destructor automatically wipes the buffer using a volatile pointer, preventing the compiler from optimizing out the memory wipe. Copying is explicitly disabled to avoid accidental duplication of sensitive data.

This RAII-based design ensures that secrets never remain in RAM longer than necessary.

---

5. Storage – Safe and Atomic File I/O

The Storage class handles all secure file read/write operations.

It uses `QSaveFile`, which writes to a temporary file and commits only once the operation succeeds. This prevents corruption even if the application crashes mid-write. Sensitive data is handled in memory only through controlled buffers.

---

6. VaultEntry – Metadata Record

Each encrypted file is represented as a VaultEntry. It stores:

- Original file path
- Encrypted output path
- Algorithm used
- Nonce/IV
- Timestamp

This metadata allows the application to track all encrypted files and organize them cleanly in the GUI.

---

7. MainWindow – Graphical User Interface

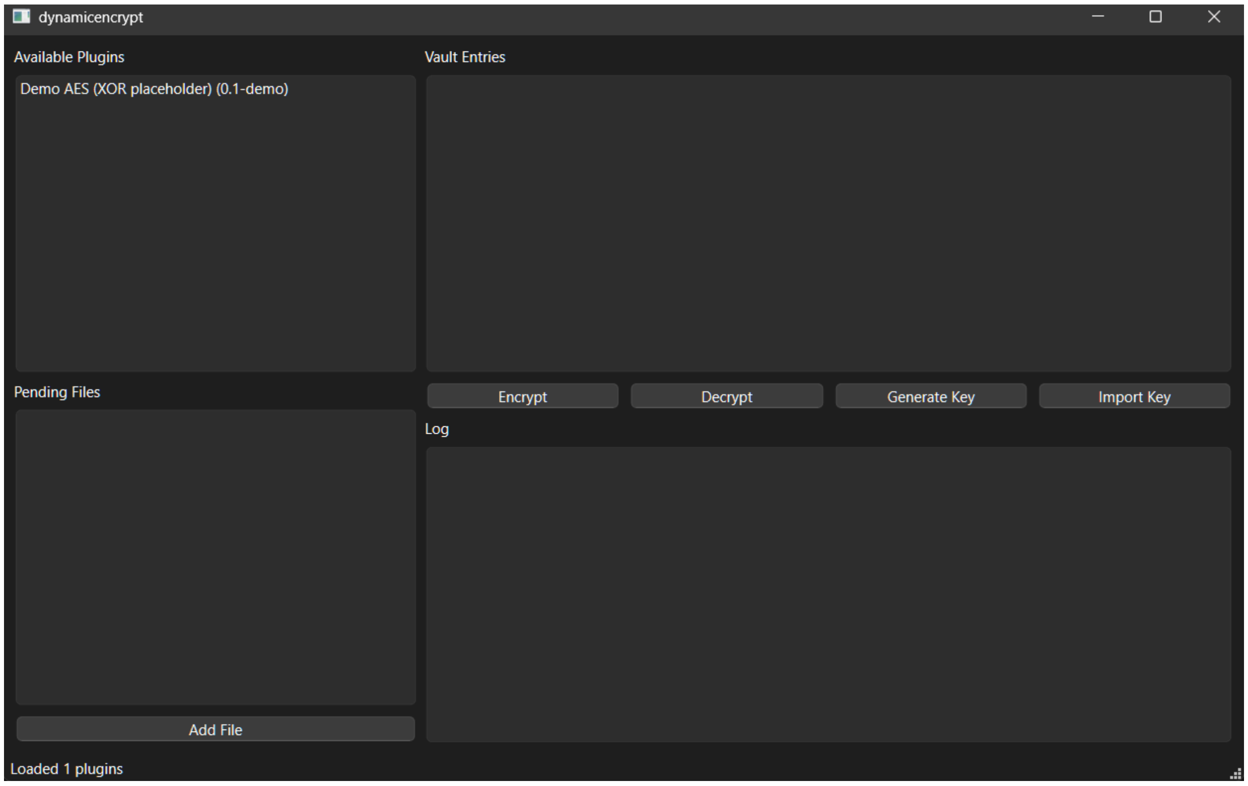
The MainWindow class is built using Qt Widgets. It provides:

- A list of available encryption plugins
- A list of files pending encryption
- A view of encrypted vault entries

- Buttons for encrypting and decrypting
- Key generation and selection dialogs

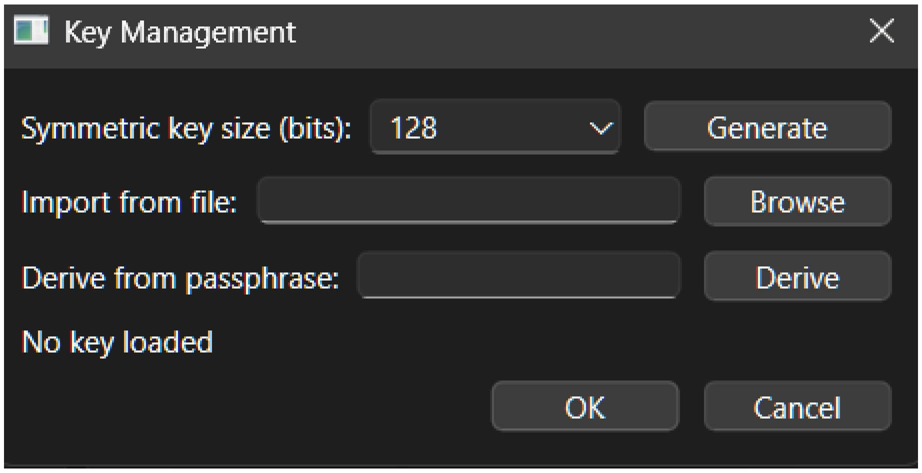
Qt’s signal-slot mechanism connects user actions directly to VaultManager functionality.

Output Screenshots



1. Main Application Window

Shows the plugin list on the left and the vault on the right, immediately after startup.


Symmetric key size (bits): 128

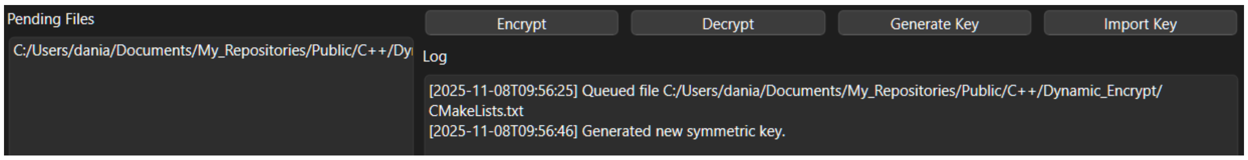
Import from file:

Derive from passphrase:

No key loaded

2. Key Generation Dialog

Displays the interface used to generate new symmetric keys.


Pending Files

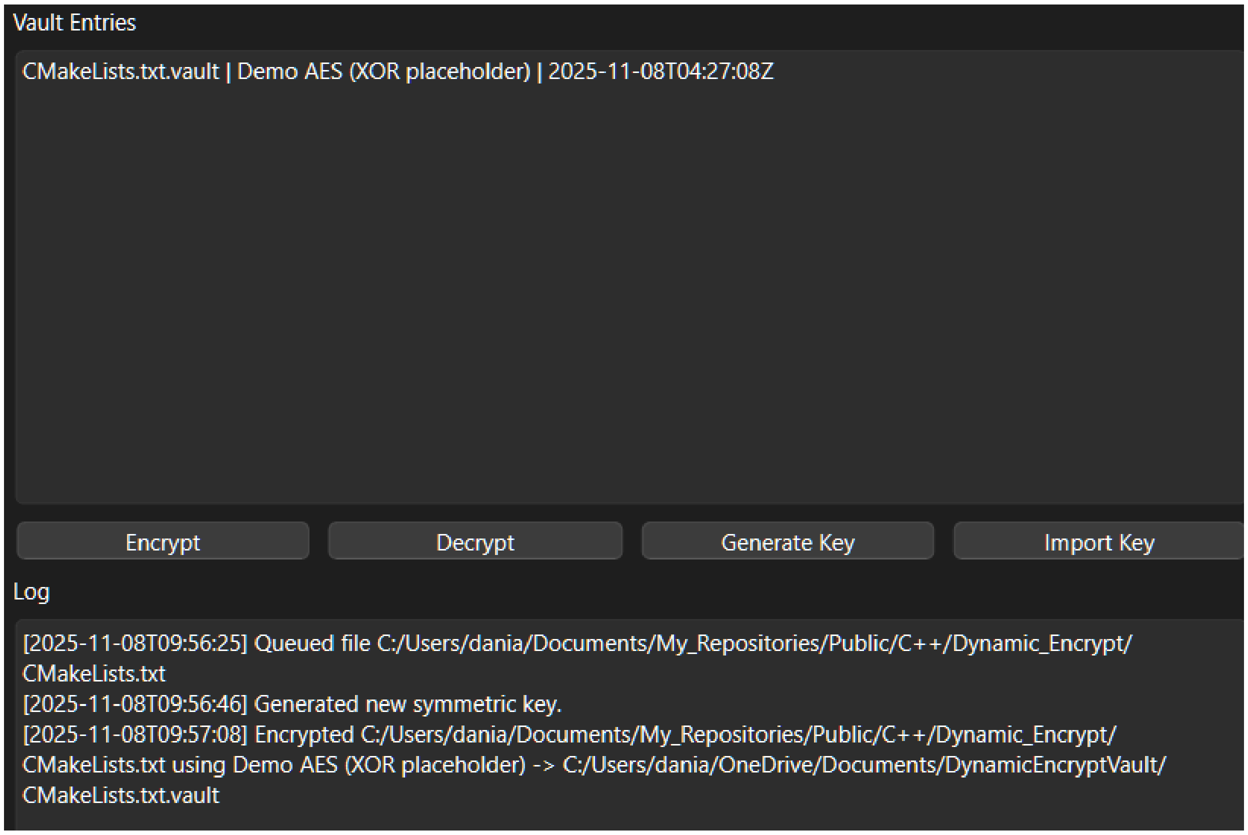
C:/Users/dania/Documents/My_Repositories/Public/C++/Dy

Log

```
[2025-11-08T09:56:25] Queued file C:/Users/dania/Documents/My_Repositories/Public/C++/Dynamic_Encrypt/
CMakeLists.txt
[2025-11-08T09:56:46] Generated new symmetric key.
```

3. File Encryption Process

Shows a file added to the pending list and successfully encrypted into the vault.



```
CMakeLists.txt.vault | Demo AES (XOR placeholder) | 2025-11-08T04:27:08Z
```

```
[2025-11-08T09:56:25] Queued file C:/Users/dania/Documents/My_Repositories/Public/C++/Dynamic_Encrypt/
CMakeLists.txt
[2025-11-08T09:56:46] Generated new symmetric key.
[2025-11-08T09:57:08] Encrypted C:/Users/dania/Documents/My_Repositories/Public/C++/Dynamic_Encrypt/
CMakeLists.txt using Demo AES (XOR placeholder) -> C:/Users/dania/OneDrive/Documents/DynamicEncryptVault/
CMakeLists.txt.vault
```

4. File Decryption Prompt

Displays the save dialog prompting the user to choose a location for the decrypted output.

Conclusion & Future Scope

DynamicEncrypt successfully delivers a secure, extensible file encryption utility using modern C++ and Qt. The project demonstrates the implementation of a flexible plugin architecture, secure memory handling, and an intuitive graphical interface.

Future Enhancements

- Replace the XOR placeholder cipher with an AES-256-GCM implementation using OpenSSL or Botan.
- Introduce password-based key protection with PBKDF2 or Argon2.
- Add asymmetric encryption plugins (e.g., RSA for public-key workflows).
- Improve the GUI with drag-and-drop support, progress bars, and advanced vault views.
- Package and test for cross-platform deployment on macOS and Linux.

References

- Bjarne Stroustrup – *The C++ Programming Language (4th Edition)*
- Scott Meyers – *Effective Modern C++*
- Qt Documentation – https://doc.qt.io/
- Crypto++ Library – https://www.cryptopp.com/
- Qt Plugin System Guide
- OpenSSL and Botan Cryptography Documentation