**Cryptography Project Report**

**Vanessa – ChaCha20 File Encryption/Decryption Tool**

**Student: Pheng Manyta**

**ID: IDTB100292**

## Abstract

This project presents *Vanessa*, a command-line file encryption application implemented in Python. The tool applies **ChaCha20-Poly1305 authenticated encryption** to ensure data confidentiality and integrity, while employing **PBKDF2-HMAC-SHA256** to derive a strong 256-bit key from a user-supplied password. Vanessa supports encryption and decryption of arbitrary files using a simple CLI interface. The project demonstrates secure password-based encryption, proper use of randomness (salt and nonce), and authenticated decryption that rejects tampered data or incorrect passwords.

## 1. Introduction

Modern data security requires strong encryption to prevent unauthorized access. Password-protected files are widely used, but weak encryption or poor key handling can lead to data breaches. This project implements a secure encryption workflow using industry-standard cryptographic primitives.

Vanessa is designed as a **lightweight personal file encryption tool**. It aims to:

- Protect sensitive files locally
- Use modern cryptography rather than legacy ciphers
- Provide authenticated encryption rather than just confidentiality
- It is easy to operate through a CLI

## 2. Project Objectives

The main goals of the project are:

1. Implement ChaCha20-Poly1305 encryption/decryption in Python.
2. Derive a secure symmetric key from a password using PBKDF2.
3. Encrypt and decrypt arbitrary files (binary/text).
4. Ensure integrity using AEAD authentication tags.
5. Build a simple and safe command-line interface.
6. Handle errors gracefully (wrong password, tampering, missing files).

## 3. Theory and Background

### 3.1. ChaCha20 Stream Cipher

ChaCha20 is a **256-bit stream cipher** designed by Daniel J. Bernstein. Key properties:

- Uses add-rotate-xor (ARX) operations
- 20-rounds block function

- High performance on CPUs without AES acceleration
- Resistant to timing attacks

### 3.2. ChaCha20-Poly1305 (AEAD)

ChaCha20-Poly1305 is an **Authenticated Encryption with Associated Data (AEAD)** construction standardized in RFC 8439. It provides:

- Confidential (ChaCha20 XOR keystream)
- Integrity and authentication (Poly1305 MAC)

If ciphertext or tag is modified, decryption fails.

### 3.3. PBKDF2 Key Derivation

Human passwords are low entropy, so they must be stretched before use. PBKDF2 applies:

- HMAC-SHA256
- Salt to prevent rainbow-table attacks
- High iteration counts to slow brute force

In Vanessa:

- Salt: 16 bytes
- Iterations: 600,000
- Derived key size: 32 bytes (256-bit)

### 3.4. Salt and Nonce

- **Salt** randomizes key derivation (stored publicly).
- **Nonce (96-bit)** ensures a unique ChaCha20 keystream.
- Nonce must **never repeat for the same key**.

## 4. System Design

### 4.1. Workflow Overview

Encryption flow:

1. Ask user for password
2. Generate random salt
3. PBKDF2 derives 256-bit key
4. Generate random nonce
5. Encrypt file using ChaCha20-Poly1305
6. Write

```
[1-byte version][salt][nonce][ciphertext + tag]
```

Decryption flow:

1. Read version, salt, nonce
2. Derive same key using password + salt
3. Decrypt and verify authenticity
4. Output plaintext file

### 4.1. Directory Structure

```
vanessa.py        - CLI interface
encryption.py     - Key derivation + encrypt_file()
decryption.py     - decrypt_file()
requirements.txt  - Dependencies
README.md         - Documentation
```

## 5. Implementation Details

### 5.1. Programming Language

- Python 3
- cryptography library ≥ 41.0.0

### 5.2. Key Functions

**derive_key_from_password(password, salt)**

- UTF-8 encode password
- PBKDF2-HMAC-SHA256
- 600,000 iterations
- return 32-byte key

**encrypt_file(input, output, password)**

- Generate salt + nonce
- Derive key
- Encrypt using ChaCha20Poly1305.encrypt()

**decrypt_file(input, output, password)**

- Read salt + nonce
- Derive key
- ChaCha20Poly1305.decrypt()
    - Throws exception on wrong password/tampering

### 5.3. Vanessa CLI

- Uses argparse
- Two sub-commands: encrypt, decrypt
- Uses getpass (password not shown)
- Password confirmation on encryption

## 6. Testing and Results

### 6.1. Functional Testing

Test performed:

```
echo "Secret message" > test.txt
python vanessa.py encrypt test.txt test.enc
python vanessa.py decrypt test.enc result.txt
```

Result:

- result.txt matches original data
- Wrong password → decryption fails with an error
- Modified ciphertext → "Decryption failed"

### 6.2. Large Files

Binary files (PDF, txt) were encrypted and decrypted successfully.

### 6.3. Error Handling

- Missing file → FileNotFoundError
- Empty password → rejected
- Wrong version → rejected
- Keyboard interrupt → safely exits

## 7. Security Evaluation

### 7.1. Strengths

- Authenticated encryption (cannot modify ciphertext silently)
- Strong 256-bit symmetric key
- Salt prevents precomputed attacks
- High PBKDF2 iterations slow brute force
- No key stored on disk
- Uses standard ChaCha20-Poly1305 API
- Random nonce each execution

### 7.2. Security Limitations

- PBKDF2 is CPU-bound and weaker than memory-hard KDF (Argon2)
- Password still depends on human strength
- CLI does not include secure password wipe across platforms

## 8. Conclusion

The Vanessa project successfully demonstrates secure password-based file encryption using modern cryptography. ChaCha20-Poly1305 ensures confidentiality and authenticity, while PBKDF2 strengthens the user's password into a safe 256-bit key. The implementation is functional, tested, and secure against basic tampering. It is well-structured and easy to extend.