→ Implementation

1. Minutiae Extraction from Fingerprint

A fingerprint is a unique pattern of ridges and valleys on the surface of a finger of an individual. A ridge is defined as a single curved segment, and a valley is a region between two adjacent ridges.
Minutiae points are the local ridge discontinuities, which are of two types:

➔ ridge endings

➔ bifurcation

A good-quality image has around 40 to 100 minutiae. It is these minutiae points that are used for determining the uniqueness of a fingerprint.

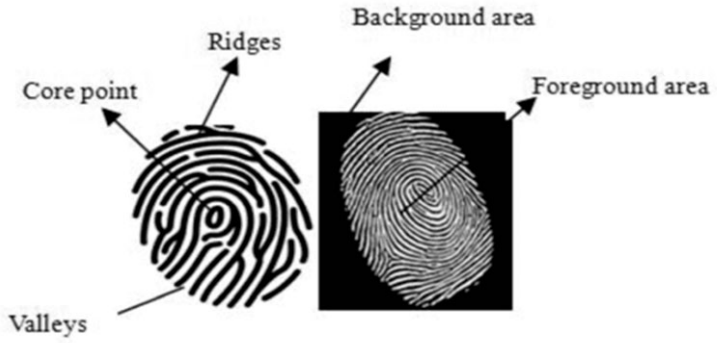

Fig 4.1 - Minutiae representation

The minutiae feature representation reduces the complex fingerprint recognition problem to a point pattern matching problem. The segmentation method needs to separate the foreground from the noisy background, which includes all ridge-valley regions and not the background, in order to achieve high-accuracy minutiae with various quality fingerprint photos.

Image enhancement techniques must also join broken ridges, remove artifacts between pseudo-parallel ridges, and avoid adding incorrect information. Last but not least, the minutiae points must be efficiently and precisely located by the minutiae detection method.

2. Dimensionality Reduction (Principal Component Analysis PCA)

It gets more and more difficult to handle data as the number of dimensions rises. The curse of dimensionality can be overcome by dimension reduction. In plain English, dimension reduction techniques shrink the quantity of data by removing pertinent information and discarding the rest as noise.

One of the most often used methods of linear dimension reduction is principal component analysis (PCA). By projecting the data onto a set of orthogonal axes, the projection-based approach of CA transforms the data. In essence, it seeks out the optimum linear combinations of the initial variables to ensure that the variation or spread along the new variable is as large as possible.

Two main procedures used in PCA

- ➔ Eigenvalue decomposition
- ➔ Singular value decomposition

Diagrammatic representation

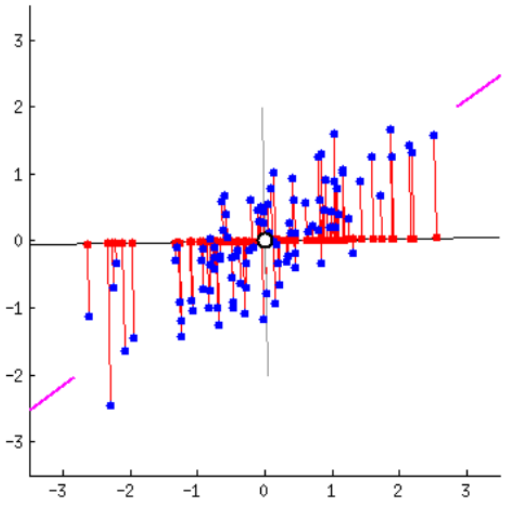

Fig 4.2 - PCA dimensionality reduction

3. Fernet Encryption algorithm

Fernet guarantees that a message encrypted using it cannot be manipulated or read without the key. Fernet is an implementation of symmetric (also known as "secret key") authenticated cryptography.

Fernet's cryptographic primitives include

- 128-bit AES in CBC mode.
- An HMAC with SHA-256.

There is a function `fernet(key)` it takes an ASCII value as input and provides us with a key that is of the form URL-safe base64-encoded 32-byte key. This must be kept secret. Anyone with this key is able to decrypt the information.

Then we will be using the `encrypt(data)` function where we pass the actual information to be encrypted and it will be called using the fernet key we generated earlier.

```
>>> f = Fernet(key)
>>> token = f.encrypt(b"my deep dark secret")
>>> token
```

Fig 4.3 - Fernet Encryption Syntax

When fernet is implemented correctly, an attacker can't read or meddle with a message that has been encrypted and authenticated with it. During the retrieval phase of the encrypted data, we can use the `decrypt(data)` function to retrieve the original data using the same key (Fig 4.4)

```
>>> f.decrypt(token)
b'my deep dark secret'
```

Fig 4.4 - Fernet Decryption Syntax

Chapter-5: Results and Discussions

We are measuring the efficiency of the fernet algorithm which is being implemented in our project with other existing encryption algorithms by tabulating results between a number of bytes processed versus time taken to process.

Fig 5.1.1 contains the speed benchmarks for some of the most commonly used cryptographic algorithms along with fernet. SSE2 intrinsics were used for multiple-precision multiplication.

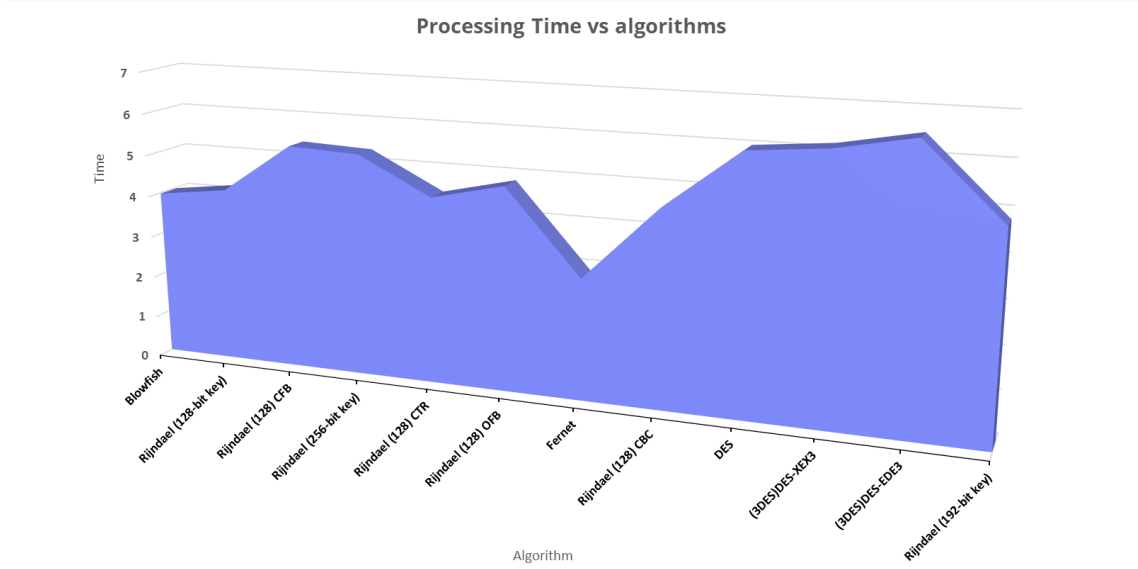

Fig 5.1.1 - Time taken vs Algorithm used for processing 256 MB data

Popular algorithms such as DES, 3DES, AES (Rijndael), Blowfish and fernet algorithms were implemented, and their performance was compared by encrypting input files of varying contents and sizes. The algorithms were implemented in a uniform language (Java), using their standard specifications, and were tested on two different hardware platforms, to compare their performance.

Fig 5.1.2 and 5.1.3 show the results of their experiments, where they have conducted it on two different machines: P-II 266 MHz and P-4 2.4 GHz.

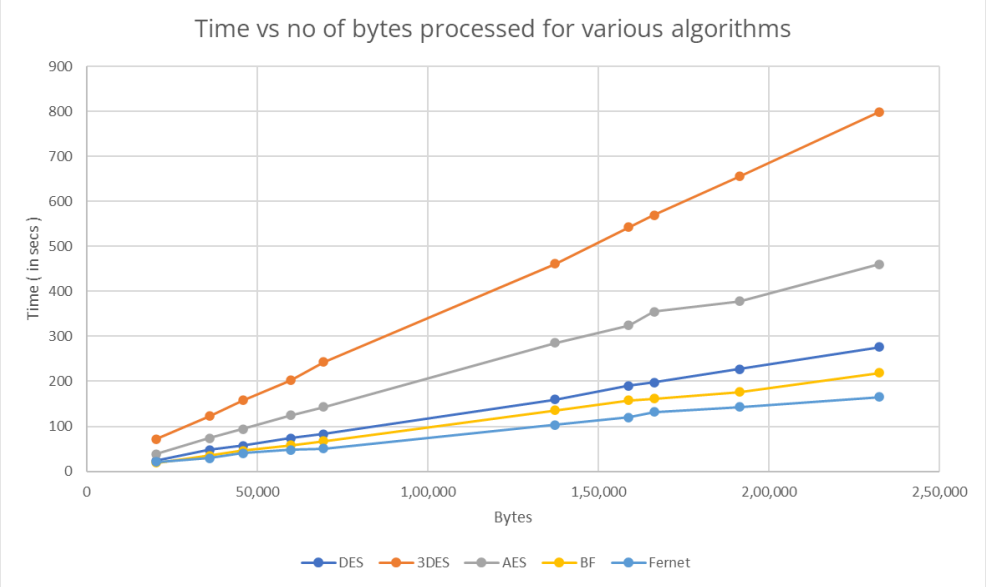

Fig 5.1.2 - Comparative execution times (in seconds) of encryption algorithms in ECB mode on a P-II 266 MHz machine

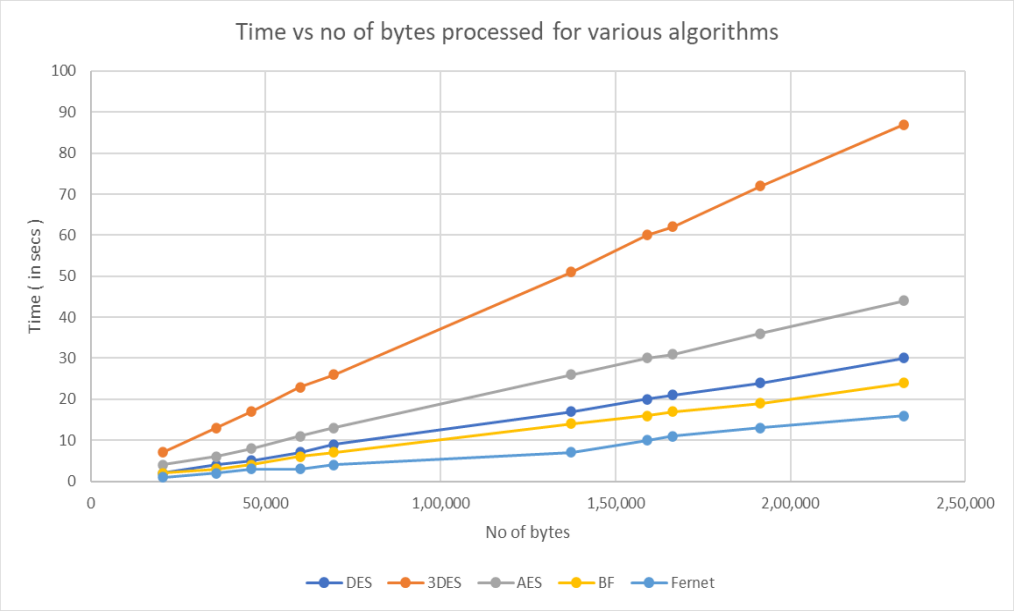

Fig 5.1.3 - Comparative execution times (in seconds) of encryption algorithms in ECB mode on a P-4 2.4 GHz machine

Justification:

Among its competitors, Fernet has the least processing time for processing 256 MB data because it specifically uses AES in CBC (Cipher Block Chaining) mode with a 128-bit key for encryption using PKCS7 padding. Also it uses HMAC for implementing SHA256 for authentication. PKCS7 with AES would always add at least 1 byte of padding, and will add enough data to make the input a multiple of the AES block size. This is why Fernet encryption algorithms stands first compared to its competitors.

Discussions:

Passkeys in Apple

According to Apple, Passkeys are a replacement for passwords that are designed to provide a more convenient, more secure, password-less sign-in experience on websites and apps. Passkeys are a standard-based technology that, unlike passwords, are resistant to phishing, always strong and designed so that there are no shared secrets. They simplify the account registration process for apps and websites, are easy to use and work across all your Apple devices, and even non-Apple devices within close physical proximity.

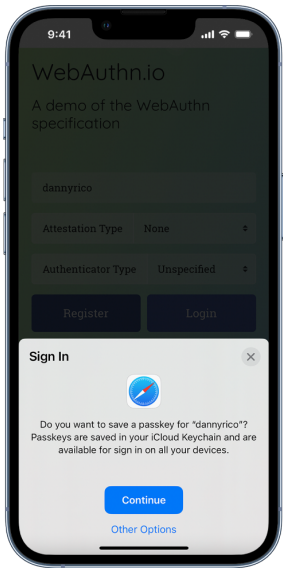

Fig 5.1.4 - Passkey feature in Apple devices (iPhone shown here)

How is it being used in google?

On Chrome on Android, passkeys are stored in the Google Password Manager, which synchronizes passkeys between the user's Android devices that are signed into the same Google account. Users are not restricted to using the passkeys only on the device where they are stored. Passkeys stored on phones can be used when logging into a laptop, even if the passkey is not synchronized to the laptop, as long as the phone is near the laptop and the user approves the sign-in on the phone. As passkeys are built on FIDO standards, all browsers can adopt them.

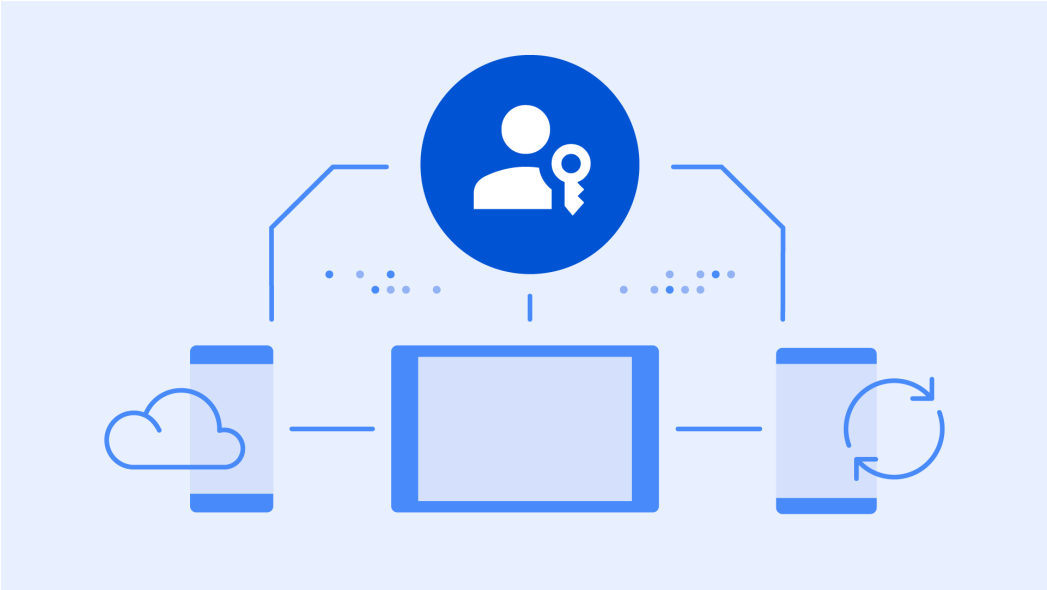

Fig 5.1.5 - Passkeys in Google implemented upon FIDO standards

Security issues and Loopholes in this system

➔ The Potential Problem With Cryptography

Cryptography largely depends on large prime numbers, making it quite difficult for keys to be hacked. However, researchers expect that in years to come, quantum computers will be able to break down public-key cryptography. This will be a big problem if passkeys cannot be re-implemented with something more quantum-proof.

➔ Websites Will Likely Retain Existing Passwords

It is important to note that it will take at least several years for most Internet users to switch to passkey as it's a huge transition.

If this happens eventually, our old passwords will not be destroyed when websites ask us to switch to passkeys. So, having one passkey-capable device may be a problem if your other devices are not compatible.

➔ Uneven Rate of Technological Development

While Apple and the wider tech community may see passkeys as the future of technological security, it is essential to note that implementing the technology across the board may be counterproductive.

While much of the Western world may easily adapt to using passkeys, several developing countries may not adopt the technology as fast. And if this doesn't happen, the primary aim of passkeys, which is creating a world without passwords, may not be feasible.