**PROJECT REPORT**

**ON**

**FILE ENCRYPTION**

**(Submitted at, University Computer Centre, Khandari)**



**DR. BHIMRAO AMBEDKAR UNIVERSITY, AGRA**

SUBMITTED TO: SUBMITTED BY:

**MR. VIJAY BHADOURIA** **MR. ROHIT JADLI**

Roll No-1900555011021

COURSE:

**BCA (Bachelor of Computer Application)**

**2022**

**ACKNOWLEDGEMENT**

I would like to express my special thanks of gratitude to my teacher **Mr. Vijay Bhadouria,** who gave me the golden opportunity to do this project on the topic of **File Encryption and Decryption.** which also helped me in doing a lot of Research and I came to know about so many new things I am really thankful to them**.**

Secondly**,** I would also like to extend my gratitude to the Head of Department sir **Prof. A.K. Gupta** for providing me with all facility that was required.

**CERTIFICATE**

This is to certify that the project\Dissertation entitled, “**File Encryption**”, is bonafide word done by **Mr. Rohit Jadli** in partial fulfilment of **BCA (Bachelor of Computer Application)** examination and has been carried out under my direct supervision and guidance. This report or a similar report in the topic has not been submitted for any other examination and does not former part of any other course undergo by the candidate.

Signature of guide/supervisor

**DECLARATION**

I Here declare that the major project entitled,”**File Encryption**” is submitted in the department of computer science, **University Computer Centre, Khandari** campus is the authentic work carried out by me under the guidance of **Mr. Vijay Bhadouria** sir, University Computer Science **, Dr. Bhimrao Ambedkar University ,Agra.**

Rohit Jadli

**TABLE OF CONTENTS**

ABSTRACT

* INTRODUCTION

1.1 CRYPTOGRAPHY

1.2 HISTORY OF CRYPTOGRAPHY

1.3 TYPES OF CRYPTOGRAPHY

1.4 PYTHON

1.5 HISTORY OF PYTHON

1.6 DESIGN PHILOSOPHY AND FEATURES

* SYSTEM REQUIREMENTS
* FEATURES
* SYSTEM DESIGN

1.1 SOFTWARE TOOL USED

1.2 MODULES

* SOURCE CODE
* RESULT
* REFERENCES

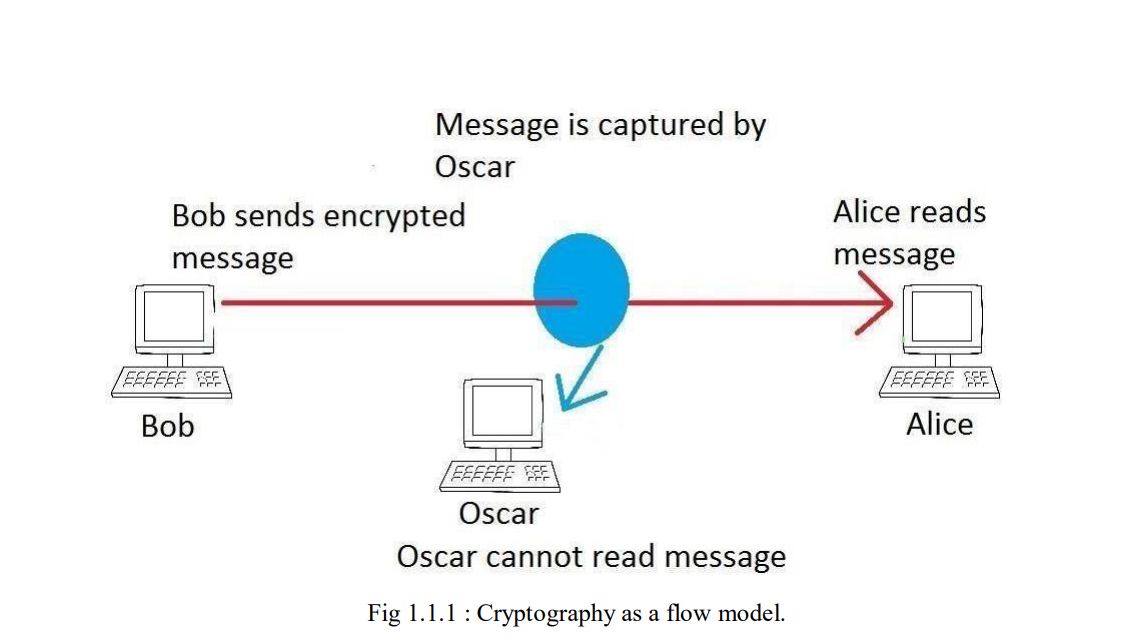
**ABSTRACT**

FICRYPT is a file encryption software which is fully designed or developed in PYTHON. Basically, it converts all types of file formats into unreadable or unexecuted file format. We encrypt all the files automatically with this software. It generates secure key automatically but user need to know his/her password when he/she encrypt the file because when they need to decrypt the file the password is must for decryption. This software cannot create a new file it will convert that particular file into unreadable or unexecuted file format.

**INTRODUCTION**

* 1. **CRYPTOGRAPHY**

**Cryptography**, or **cryptology** is the practice and study of techniques for secure communication in the presence of adversarial behavior. More generally, cryptography is about constructing and analyzing protocols that prevent third parties or the public from reading private message various aspects of information security such as data confidentiality, data integrity, authentication, and non-repudiation are central to modern cryptography. Modern cryptography exists at the intersection of the disciplines of mathematics, computer science, electrical engineering, communication science, and physics. Applications of cryptography include electronic commerce, chip-based payment cards, digital currencies, computer passwords, and military communications.



Cryptography prior to the modern age was effectively synonymous with encryption, converting information from a readable state to unintelligible nonsense. The sender of an encrypted message shares the decoding technique only with intended recipients to preclude access from adversaries. The cryptography literature often uses the names Alice (“A”) for the sender, Bob (“B”) for the intended recipient, and Eve (“eavesdropper”) for the adversary. Since the development of rotor cipher machines in World War I and the advent of computers in World War II, cryptography methods have become increasingly complex and their applications more varied.

Modern cryptography is heavily based on mathematical theory and computer science practice; cryptographic algorithms are designed around computational hardness assumptions, making such algorithms hard to break in actual practice by any adversary. While it is theoretically possible to break into a well-designed system, it is infeasible in actual practice to do so. Such schemes, if well designed, are therefore termed “computationally secure”; theoretical advances (e.g., improvements in integer factorization algorithms) and faster computing technology require these designs to be continually re-evaluated, and if necessary, adapted. Information-theoretically secure schemes that provably cannot be broken even with unlimited computing power, such as the one-time pad, are much more difficult to use in practice than the best theoretically breakable, but computationally secure, schemes.

The growth of cryptographic technology has raised a number of legal issues in the Information Age. Cryptography’s potential for use as a tool for espionage and sedition has led many governments to classify it as a weapon and to limit or even prohibit its use and export.[6] In some jurisdictions where the use of cryptography is legal, laws permit investigators to compel the disclosure of encryption keys for documents relevant to an investigation. Cryptography also plays a major role in digital rights management and copyright infringement disputes in regard to digital media.

* 1. **HISTORY OF CRYPTOGRAPHY**

Before the modern era, cryptography focused on message confidentiality (i.e., encryption)—conversion of messages from a comprehensible form into an incomprehensible one and back again at the other end, rendering it unreadable by interceptors or eavesdroppers without secret knowledge (namely the key needed for decryption of that message). Encryption attempted to ensure secrecy in communications, such as those of spies, military leaders, and diplomats. In recent decades, the field has expanded beyond confidentiality concerns to include techniques for message integrity checking, sender/receiver identity authentication, digital signatures, interactive proofs and secure computation, among others.

**Classic Cryptography**

Skytala stick with strip of paper wound around in spiral

Reconstructed ancient Greek scytale, an early cipher device

The main classical cipher types are transposition ciphers, which rearrange the order of letters in a message (e.g., 'hello world' becomes 'ehlol owrdl' in a trivially simple rearrangement scheme), and substitution ciphers, which systematically replace letters or groups of letters with other letters or groups of letters (e.g., 'fly at once' becomes 'gmz bu podf' by replacing each letter with the one following it in the Latin alphabet). Simple versions of either have never offered much confidentiality from enterprising opponents. An early substitution cipher was the Caesar cipher, in which each letter in the plaintext was replaced by a letter some fixed number of positions further down the alphabet. Suetonius reports that Julius Caesar used it with a shift of three to communicate with his generals. Atbash is an example of an early Hebrew cipher. The earliest known use of cryptography is some carved ciphertext on stone in Egypt (ca 1900 BCE), but this may have been done for the amusement of literate observers rather than as a way of concealing information.

The Greeks of Classical times are said to have known of ciphers (e.g., the scytale transposition cipher claimed to have been used by the Spartan military). Steganography (i.e., hiding even the existence of a message so as to keep it confidential) was also first developed in ancient times. An early example, from Herodotus, was a message tattooed on a slave's shaved head and concealed under the regrown hair. More modern examples of steganography include the use of invisible ink, microdots, and digital watermarks to conceal information.

In India, the 2000-year-old Kamasutra of Vātsyāyana speaks of two different kinds of ciphers called Kautiliyam and Mulavediya. In the Kautiliyam, the cipher letter substitutions are based on phonetic relations, such as vowels becoming consonants. In the Mulavediya, the cipher alphabet consists of pairing letters and using the reciprocal ones.

In Sassanid Persia, there were two secret scripts, according to the Muslim author Ibn al-Nadim: the šāh-dabīrīya (literally "King's script") which was used for official correspondence, and the rāz-saharīya which was used to communicate secret messages with other countries.

David Kahn notes in The Codebreakers that modern cryptology originated among the Arabs, the first people to systematically document cryptanalytic methods. Al-Khalil (717–786) wrote the Book of Cryptographic Messages, which contains the first use of permutations and combinations to list all possible Arabic words with and without vowels.

Ciphertexts produced by a classical cipher (and some modern ciphers) will reveal statistical information about the plaintext, and that information can often be used to break the cipher. After the discovery of frequency analysis, perhaps by the Arab mathematician and polymath Al-Kindi (also known as Alkindus) in the 9th century, nearly all such ciphers could be broken by an informed attacker. Such classical ciphers still enjoy popularity today, though mostly as puzzles (see cryptogram). Al-Kindi wrote a book on cryptography entitled Risalah fi Istikhraj al-Mu'amma (Manuscript for the Deciphering Cryptographic Messages), which described the first known use of frequency analysis cryptanalysis techniques.

book sized metal machine with large dial left page and nineteen small dials right page

16th-century book-shaped French cipher machine, with arms of Henri II of France

manuscript from Gabriel de Luetz d'Aramon in bound volume

Enciphered letter from Gabriel de Luetz d'Aramon, French Ambassador to the Ottoman Empire, after 1546, with partial decipherment

Language letter frequencies may offer little help for some extended historical encryption techniques such as homophonic cipher that tend to flatten the frequency distribution. For those ciphers, language letter group (or n-gram) frequencies may provide an attack.

Essentially all ciphers remained vulnerable to cryptanalysis using the frequency analysis technique until the development of the polyalphabetic cipher, most clearly by Leon Battista Alberti around the year 1467, though there is some indication that it was already known to Al-Kindi. Alberti's innovation was to use different ciphers (i.e., substitution alphabets) for various parts of a message (perhaps for each successive plaintext letter at the limit). He also invented what was probably the first automatic cipher device, a wheel which implemented a partial realization of his invention. In the Vigenère cipher, a polyalphabetic cipher, encryption uses a key word, which controls letter substitution depending on which letter of the key word is used. In the mid-19th century Charles Babbage showed that the Vigenère cipher was vulnerable to Kasiski examination, but this was first published about ten years later by Friedrich Kasiski.

Although frequency analysis can be a powerful and general technique against many ciphers, encryption has still often been effective in practice, as many a would-be cryptanalyst was unaware of the technique. Breaking a message without using frequency analysis essentially required knowledge of the cipher used and perhaps of the key involved, thus making espionage, bribery, burglary, defection, etc., more attractive approaches to the cryptanalytically uninformed. It was finally explicitly recognized in the 19th century that secrecy of a cipher's algorithm is not a sensible nor practical safeguard of message security; in fact, it was further realized that any adequate cryptographic scheme (including ciphers) should remain secure even if the adversary fully understands the cipher algorithm itself. Security of the key used should alone be sufficient for a good cipher to maintain confidentiality under an attack. This fundamental principle was first explicitly stated in 1883 by Auguste Kerckhoffs and is generally called Kerckhoffs's Principle; alternatively and more bluntly, it was restated by Claude Shannon, the inventor of information theory and the fundamentals of theoretical cryptography, as Shannon's Maxim—'the enemy knows the system'.

Different physical devices and aids have been used to assist with ciphers. One of the earliest may have been the scytale of ancient Greece, a rod supposedly used by the Spartans as an aid for a transposition cipher. In medieval times, other aids were invented such as the cipher grille, which was also used for a kind of steganography. With the invention of polyalphabetic ciphers came more sophisticated aids such as Alberti's own cipher disk, Johannes Trithemius' tabula recta scheme, and Thomas Jefferson's wheel cypher (not publicly known, and reinvented independently by Bazeries around 1900). Many mechanical encryption/decryption devices were invented early in the 20th century, and several patented, among them rotor machines—famously including the Enigma machine used by the German government and military from the late 1920s and during World War II. The ciphers implemented by better quality examples of these machine designs brought about a substantial increase in cryptanalytic difficulty after WWI.

* 1. **Types of cryptography**

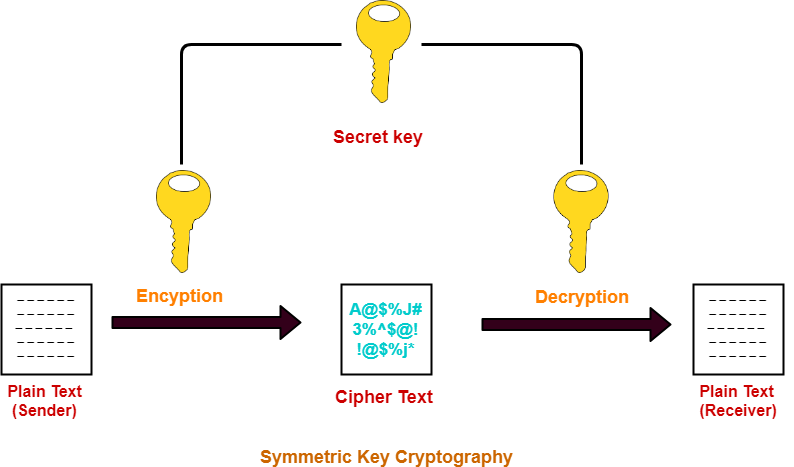
**Symmetric-key cryptography**

Symmetric-key cryptography, where a single key is used for encryption and decryption.

Symmetric-key cryptography refers to encryption methods in which both the sender and receiver share the same key (or, less commonly, in which their keys are different, but related in an easily computable way). This was the only kind of encryption publicly known until June 1976.

One round (out of 8.5) of the IDEA cipher, used in most versions of PGP and OpenPGP compatible software for time-efficient encryption of messages

Symmetric key ciphers are implemented as either block ciphers or stream ciphers. A block cipher enciphers input in blocks of plaintext as opposed to individual characters, the input form used by a stream cipher.



The Data Encryption Standard (DES) and the Advanced Encryption Standard (AES) are block cipher designs that have been designated cryptography standards by the US government (though DES's designation was finally withdrawn after the AES was adopted). Despite its deprecation as an official standard, DES (especially its still-approved and much more secure triple-DES variant) remains quite popular; it is used across a wide range of applications, from ATM encryption to e-mail privacy and secure remote access. Many other block ciphers have been designed and released, with considerable variation in quality. Many, even some designed by capable practitioners, have been thoroughly broken, such as FEAL.

Stream ciphers, in contrast to the 'block' type, create an arbitrarily long stream of key material, which is combined with the plaintext bit-by-bit or character-by-character, somewhat like the one-time pad. In a stream cipher, the output stream is created based on a hidden internal state that changes as the cipher operates. That internal state is initially set up using the secret key material. RC4 is a widely used stream cipher. Block ciphers can be used as stream ciphers by generating blocks of a keystream (in place of a Pseudorandom number generator) and applying an XOR operation to each bit of the plaintext with each bit of the keystream.

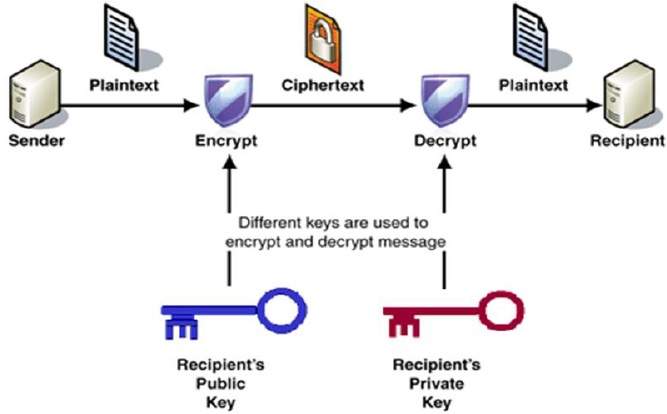
Message authentication codes (MACs) are much like cryptographic hash functions, except that a secret key can be used to authenticate the hash value upon receipt; this additional complication blocks an attack scheme against bare digest algorithms, and so has been thought worth the effort. Cryptographic hash functions are a third type of cryptographic algorithm. They take a message of any length as input, and output a short, fixed-length hash, which can be used in (for example) a digital signature. For good hash functions, an attacker cannot find two messages that produce the same hash. MD4 is a long-used hash function that is now broken; MD5, a strengthened variant of MD4, is also widely used but broken in practice. The US National Security Agency developed the Secure Hash Algorithm series of MD5-like hash functions: SHA-0 was a flawed algorithm that the agency withdrew; SHA-1 is widely deployed and more secure than MD5, but cryptanalysts have identified attacks against it; the SHA-2 family improves on SHA-1, but is vulnerable to clashes as of 2011; and the US standards authority thought it "prudent" from a security perspective to develop a new standard to "significantly improve the robustness of NIST's overall hash algorithm toolkit." Thus, a hash function design competition was meant to select a new U.S. national standard, to be called SHA-3, by 2012. The competition ended on October 2, 2012, when the NIST announced that Keccak would be the new SHA-3 hash algorithm. Unlike block and stream ciphers that are invertible, cryptographic hash functions produce a hashed output that cannot be used to retrieve the original input data. Cryptographic hash functions are used to verify the authenticity of data retrieved from an untrusted source or to add a layer of security.

**Public-key cryptography**

Public-key cryptography, where different keys are used for encryption and decryption.

Symmetric-key cryptosystems use the same key for encryption and decryption of a message, although a message or group of messages can have a different key than others. A significant disadvantage of symmetric ciphers is the key management necessary to use them securely. Each distinct pair of communicating parties must, ideally, share a different key, and perhaps for each ciphertext exchanged as well. The number of keys required increases as the square of the number of network members, which very quickly requires complex key management schemes to keep them all consistent and secret.

Whitfield Diffie and Martin Hellman, authors of the first published paper on public-key cryptography.



In a groundbreaking 1976 paper, Whitfield Diffie and Martin Hellman proposed the notion of public-key (also, more generally, called asymmetric key) cryptography in which two different but mathematically related keys are used—a public key and a private key. A public key system is so constructed that calculation of one key (the 'private key') is computationally infeasible from the other (the 'public key'), even though they are necessarily related. Instead, both keys are generated secretly, as an interrelated pair. The historian David Kahn described public-key cryptography as "the most revolutionary new concept in the field since polyalphabetic substitution emerged in the Renaissance".

In public-key cryptosystems, the public key may be freely distributed, while its paired private key must remain secret. In a public-key encryption system, the public key is used for encryption, while the private or secret key is used for decryption. While Diffie and Hellman could not find such a system, they showed that public-key cryptography was indeed possible by presenting the Diffie–Hellman key exchange protocol, a solution that is now widely used in secure communications to allow two parties to secretly agree on a shared encryption key. The X.509 standard defines the most commonly used format for public key certificates.

Diffie and Hellman's publication sparked widespread academic efforts in finding a practical public-key encryption system. This race was finally won in 1978 by Ronald Rivest, Adi Shamir, and Len Adleman, whose solution has since become known as the RSA algorithm.

The Diffie–Hellman and RSA algorithms, in addition to being the first publicly known examples of high-quality public-key algorithms, have been among the most widely used. Other asymmetric-key algorithms include the Cramer–Shoup cryptosystem, ElGamal encryption, and various elliptic curve techniques.[citation needed]

A document published in 1997 by the Government Communications Headquarters (GCHQ), a British intelligence organization, revealed that cryptographers at GCHQ had anticipated several academic developments. Reportedly, around 1970, James H. Ellis had conceived the principles of asymmetric key cryptography. In 1973, Clifford Cocks invented a solution that was very similar in design rationale to RSA. In 1974, Malcolm J. Williamson is claimed to have developed the Diffie–Hellman key exchange.

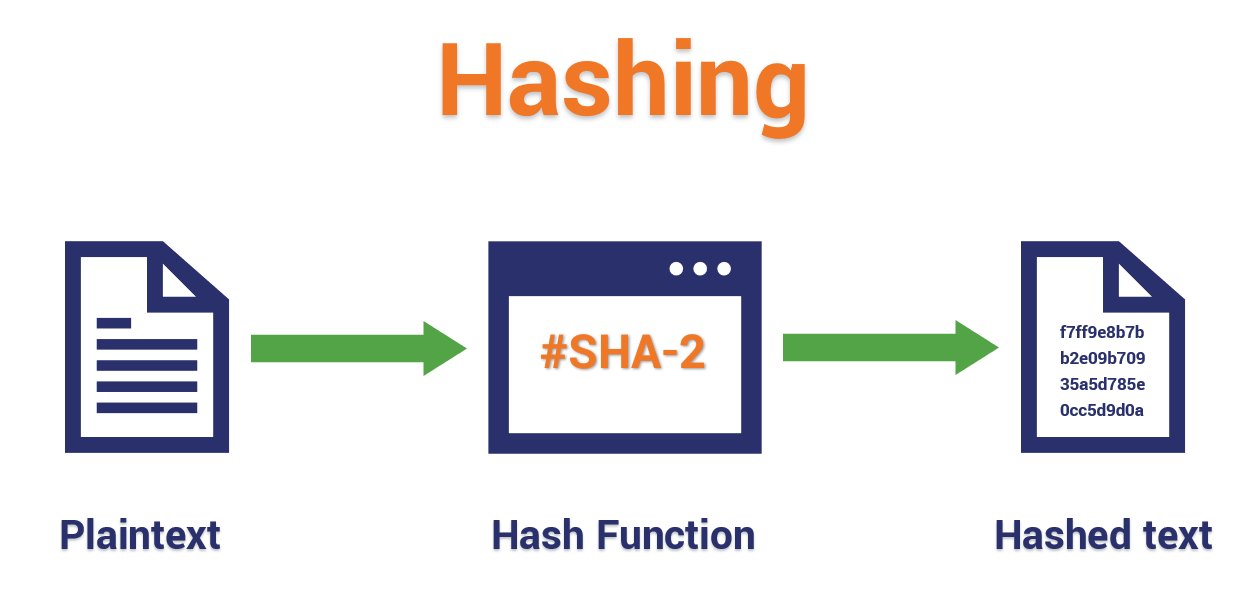
In this example the message is only signed and not encrypted. 1) Alice signs a message with her private key. 2) Bob can verify that Alice sent the message and that the message has not been modified.

Public-key cryptography is also used for implementing digital signature schemes. A digital signature is reminiscent of an ordinary signature; they both have the characteristic of being easy for a user to produce, but difficult for anyone else to forge. Digital signatures can also be permanently tied to the content of the message being signed; they cannot then be 'moved' from one document to another, for any attempt will be detectable. In digital signature schemes, there are two algorithms: one for signing, in which a secret key is used to process the message (or a hash of the message, or both), and one for verification, in which the matching public key is used with the message to check the validity of the signature. RSA and DSA are two of the most popular digital signature schemes. Digital signatures are central to the operation of public key infrastructures and many network security schemes (e.g., SSL/TLS, many VPNs, etc.).

Public-key algorithms are most often based on the computational complexity of "hard" problems, often from number theory. For example, the hardness of RSA is related to the integer factorization problem, while Diffie–Hellman and DSA are related to the discrete logarithm problem. The security of elliptic curve cryptography is based on number theoretic problems involving elliptic curves. Because of the difficulty of the underlying problems, most public-key algorithms involve operations such as modular multiplication and exponentiation, which are much more computationally expensive than the techniques used in most block ciphers, especially with typical key sizes. As a result, public-key cryptosystems are commonly hybrid cryptosystems, in which a fast high-quality symmetric-key encryption algorithm is used for the message itself, while the relevant symmetric key is sent with the message, but encrypted using a public-key algorithm. Similarly, hybrid signature schemes are often used, in which a cryptographic hash function is computed, and only the resulting hash is digitally signed.

**Cryptographic Hash Functions**

Cryptographic Hash Functions are cryptographic algorithms that are ways to generate and utilize specific keys to encrypt data for either symmetric or asymmetric encryption, and such functions may be viewed as keys themselves. They take a message of any length as input, and output a short, fixed-length hash, which can be used in (for example) a digital signature. For good hash functions, an attacker cannot find two messages that produce the same hash. MD4 is a long-used hash function that is now broken; MD5, a strengthened variant of MD4, is also widely used but broken in practice. The US National Security Agency developed the Secure Hash Algorithm series of MD5-like hash functions: SHA-0 was a flawed algorithm that the agency withdrew; SHA-1 is widely deployed and more secure than MD5, but cryptanalysts have identified attacks against it; the SHA-2 family improves on SHA-1, but is vulnerable to clashes as of 2011; and the US standards authority thought it "prudent" from a security perspective to develop a new standard to "significantly improve the robustness of NIST's overall hash algorithm toolkit." Thus, a hash function design competition was meant to select a new U.S. national standard, to be called SHA-3, by 2012. The competition ended on October 2, 2012, when the NIST announced that Keccak would be the new SHA-3 hash algorithm. Unlike block and stream ciphers that are invertible, cryptographic hash functions produce a hashed output that cannot be used to retrieve the original input data. Cryptographic hash functions are used to verify the authenticity of data retrieved from an untrusted source or to add a layer of security.



Python is a high-level, interpreted, general-purpose programming language. Its design philosophy emphasizes code readability with the use of significant indentation.

* 1. **PYTHON**

**Python** is dynamically-typed and garbage-collected. It supports multiple programming paradigms, including structured (particularly procedural), object-oriented and functional programming. It is often described as a "batteries included" language due to its comprehensive standard library.

Guido van Rossum began working on Python in the late 1980s as a successor to the ABC programming language and first released it in 1991 as Python 0.9.0. Python 2.0 was released in 2000 and introduced new features such as list comprehensions, cycle-detecting garbage collection, reference counting, and Unicode support. Python 3.0, released in 2008, was a major revision that is not completely backward-compatible with earlier versions. Python 2 was discontinued with version 2.7.18 in 2020.

Python consistently ranks as one of the most popular programming languages.

**1.5 HISTORY OF PYTHON**

Python was conceived in the late 1980s by Guido van Rossum at Centrum Wiskunde & Informatica (CWI) in the Netherlands as a successor to the ABC programming language, which was inspired by SETL, capable of exception handling and interfacing with the Amoeba operating system. Its implementation began in December 1989. Van Rossum shouldered sole responsibility for the project, as the lead developer, until 12 July 2018, when he announced his "permanent vacation" from his responsibilities as Python's "benevolent dictator for life", a title the Python community bestowed upon him to reflect his long-term commitment as the project's chief decision-maker. In January 2019, active Python core developers elected a five-member Steering Council to lead the project.

Python 2.0 was released on 16 October 2000, with many major new features. Python 3.0, released on 3 December 2008, with many of its major features backported to Python 2.6.x and 2.7.x. Releases of Python 3 include the 2to3 utility, which automates the translation of Python 2 code to Python 3.

Python 2.7's end-of-life was initially set for 2015, then postponed to 2020 out of concern that a large body of existing code could not easily be forward-ported to Python 3. No further security patches or other improvements will be released for it. With Python 2's end-of-life, only Python 3.6.X and later were supported. Later, support for 3.6 was also discontinued. In 2021, Python 3.9.2 and 3.8.8 were expedited as all versions of Python (including 2.7) had security issues leading to possible remote code execution and web cache poisoning.

In 2022, Python 3.10.4 and 3.9.12 were expedited and so were older releases including 3.8.13, and 3.7.13 because of many security issues in 2022. Python 3.9.13 is the latest 3.9 version, and from now on 3.9 (and older; 3.8 and 3.7) will only get security updates.

**1.6 Design philosophy and features**

Python is a multi-paradigm programming language. Object-oriented programming and structured programming are fully supported, and many of its features support functional programming and aspect-oriented programming (including metaprogramming and metaobjects [magic methods). Many other paradigms are supported via extensions, including design by contract and logic programming.

Python uses dynamic typing and a combination of reference counting and a cycle-detecting garbage collector for memory management. It uses dynamic name resolution (late binding), which binds method and variable names during program execution.

Its design offers some support for functional programming in the Lisp tradition. It has filter, mapandreduce functions; list comprehensions, dictionaries, sets, and generator expressions. The standard library has two modules (itertools and functools) that implement functional tools borrowed from Haskell and Standard ML.

Its core philosophy is summarized in the document The Zen of Python (PEP 20), which includes aphorisms such as:

1. **Beautiful is better than ugly.**
2. **Explicit is better than implicit.**
3. **Simple is better than complex.**
4. **Complex is better than complicated.**
5. **Readability counts.**

Rather than building all of its functionality into its core, Python was designed to be highly extensible via modules. This compact modularity has made it particularly popular as a means of adding programmable interfaces to existing applications. Van Rossum's vision of a small core language with a large standard library and easily extensible interpreter stemmed from his frustrations with ABC, which espoused the opposite approach.

Python strives for a simpler, less-cluttered syntax and grammar while giving developers a choice in their coding methodology. In contrast to Perl's "there is more than one way to do it" motto, Python embraces a "there should be one—and preferably only one—obvious way to do it" philosophy. Alex Martelli, a Fellow at the Python Software Foundation and Python book author, wrote: "To describe something as 'clever' is not considered a compliment in the Python culture."

Python's developers strive to avoid premature optimization and reject patches to non-critical parts of the Python reference implementation that would offer marginal increases in speed at the cost of clarity. When speed is important, a Python programmer can move time-critical functions to extension modules written in languages such as C; or use PyPy, a just-in-time compiler. Python is also available, which translates a Python script into C and makes direct C-level API calls into the Python interpreter.

Python's developers aim for it to be fun to use. This is reflected in its name—a tribute to the British comedy group Monty Python—and in occasionally playful approaches to tutorials and reference materials, such as examples that refer to spam and eggs (a reference to a Monty Python sketch) instead of the standard foo and bar.

A common neologism in the Python community is pythonic, which has a wide range of meanings related to program style. "Pythonic" code may use Python idioms well, be natural or show fluency in the language, or conform with Python's minimalist philosophy and emphasis on readability. Code that is difficult to understand or reads like a rough transcription from another programming language is called un-pythonic.

Python users and admirers, especially those considered knowledgeable or experienced, are often referred to as Pythonistas.

**SYSTEM REQUIREMENTS**

**OS**: Windows Only

**Processor**: Intel i3 or AMD A4 or above

**RAM**: Minimum of 256 MB or higher

**HDD**: 10GB or higher

No Graphics Card Needed to run this software

**FEATURES**

* Easy To Use
* User doesn’t need to know the key
* It will not take extra space by creating new file
* It converts normal file to an un-executable format
* Provide Protection

**SYSTEM DESIGN**

* 1. **SOFTWARE TOOL USED**

**VISUAL STUDIO CODE**

Visual Studio Code, also commonly referred to as VS Code, is a source-code editor made by Microsoft for Windows, Linux and macOS. Features include support for debugging, syntax highlighting, intelligent code completion, snippets, code refactoring, and embedded Git. Users can change the theme, keyboard shortcuts, preferences, and install extensions that add additional functionality.

In the Stack Overflow 2021 Developer Survey, Visual Studio Code was ranked the most popular developer environment tool, with 70% of 82,000 respondents reporting that they use it.

**HISTORY OF VS CODE**

Visual Studio Code was first announced on April 29, 2015, by Microsoft at the 2015 Build conference. A preview build was released shortly thereafter.

On November 18, 2015, the source of Visual Studio Code was released under the MIT License, and made available on GitHub. Extension support was also announced. On April 14, 2016, Visual Studio Code graduated from the public preview stage and was released to the Web. Microsoft has released most of Visual Studio Code's source code on GitHub under the permissive MIT License, while the releases by Microsoft are proprietary freeware.

**FEATURES**

Visual Studio Code is a source-code editor that can be used with a variety of programming languages, including Java, JavaScript, Go, Node.js, Python, C++ and Fortran. It is based on the Electron framework, which is used to develop Node.js Web applications that run on the Blink layout engine. Visual Studio Code employs the same editor component (codenamed "Monaco") used in Azure DevOps (formerly called Visual Studio Online and Visual Studio Team Services).

Out of the box, Visual Studio Code includes basic support for most common programming languages. This basic support includes syntax highlighting, bracket matching, code folding, and configurable snippets. Visual Studio Code also ships with IntelliSense for JavaScript, TypeScript, JSON, CSS, and HTML, as well as debugging support for Node.js. Support for additional languages can be provided by freely available extensions on the VS Code Marketplace.

Instead of a project system, it allows users to open one or more directories, which can then be saved in workspaces for future reuse. This allows it to operate as a language-agnostic code editor for any language. It supports many programming languages and a set of features that differs per language. Unwanted files and folders can be excluded from the project tree via the settings. Many Visual Studio Code features are not exposed through menus or the user interface but can be accessed via the command palette.

Visual Studio Code can be extended via extensions, available through a central repository. This includes additions to the editor and language support. A notable feature is the ability to create extensions that add support for new languages, themes, and debuggers, perform static code analysis, and add code linters using the Language Server Protocol.

Source control is a built-in feature of Visual Studio Code. It has a dedicated tab inside of the menu bar where you can access version control settings and view changes made to the current project. To use the feature you must link Visual Studio Code to any supported version control system (Git, Apache Subversion, Perforce, etc.). This allows you to create repositories as well as to make push and pull requests directly from the Visual Studio Code program.

Visual Studio Code includes multiple extensions for FTP, allowing the software to be used as a free alternative for web development. Code can be synced between the editor and the server, without downloading any extra software.

Visual Studio Code allows users to set the code page in which the active document is saved, the newline character, and the programming language of the active document. This allows it to be used on any platform, in any locale, and for any given programming language.

Visual Studio Code collects usage data and sends it to Microsoft, although this can be disabled. Due to the open-source nature of the application, the telemetry code is accessible to the public, who can see exactly what is collected.

* 1. **MODULES**

**TKINTER**

Tkinter in pythonis the standard GUI library for Python. Python when combined with Tkinter provides a fast and easy way to create GUI applications. Tkinter provides a powerful object-oriented interface to the Tk GUI toolkit.

Creating a GUI application using Tkinter is an easy task. All you need to do is perform the following steps −

Import the Tkinter module.

Create the GUI application main window.

Add one or more of the above-mentioned widgets to the GUI application.

Enter the main event loop to take action against each event triggered by the user.

**Example**

#!/usr/bin/python

import Tkinter

top = Tkinter.Tk()

# Code to add widgets will go here...

top.mainloop()

Now, the output of this program is



**Geometry Management**

All Tkinter widgets have access to specific geometry management methods, which have the purpose of organizing widgets throughout the parent widget area. Tkinter exposes the following geometry manager classes: pack, grid, and place.

**The pack() Method** − This geometry manager organizes widgets in blocks before placing them in the parent widget.

**The grid() Method** − This geometry manager organizes widgets in a table-like structure in the parent widget.

**The place() Method** − This geometry manager organizes widgets by placing them in a specific position in the parent widget.

**Cryptography**

Cryptography is the practice of securing useful information while transmitting from one computer to another or storing data on a computer. Cryptography deals with the encryption of plaintext into ciphertext and decryption of ciphertext into plaintext. Python supports a cryptography package that helps us encrypt and decrypt data. The fernet module of the cryptography package has inbuilt functions for the generation of the key, encryption of plaintext into ciphertext, and decryption of ciphertext into plaintext using the encrypt and decrypt methods respectively. The fernet module guarantees that data encrypted using it cannot be further manipulated or read without the key.

Methods Used:

**generate\_key()** : This method generates a new fernet key. The key must be kept safe as it is the most important component to decrypt the ciphertext. If the key is lost then the user can no longer decrypt the message. Also if an intruder or hacker gets access to the key they can not only read the data but also forge the data.

**encrypt(data)**: It encrypts data passed as a parameter to the method. The outcome of this encryption is known as a “Fernet token” which is basically the ciphertext. The encrypted token also contains the current timestamp when it was generated in plaintext. The encrypt method throws an exception if the data is not in bytes.

**Parameters:**

**data (bytes)** – The plaintext to be encrypted.

**Return value:** A ciphertext that cannot be read or altered without the key. It is URL-safe base64-encoded and is referred to as Fernet token.

**decrypt(token,ttl=None)**: This method decrypts the Fernet token passed as a parameter to the method. On successful decryption the original plaintext is obtained as a result, otherwise an exception is thrown.

**Parameters:**

**Token (bytes)** – The Fernet token (ciphertext) is passed for decryption.

**ttl (int)** – Optionally, one may provide an integer as second parameter in the decrypt method. The ttl denotes the time about how long a token is valid. If the token is older than ttl seconds (from the time it was originally created) an exception is thrown. If ttl is not passed as a parameter, then age of the token is not considered. If the token is somehow invalid, an exception is thrown.

**Returns value**: Returns the original plaintext.

**OS**

The OS module in Python provides functions for interacting with the operating system. OS comes under Python’s standard utility modules. This module provides a portable way of using operating system-dependent functionality. The \*os\* and \*os.path\* modules include many functions to interact with the file system.

**SOURCE CODE**

from tkinter import \*

from PIL import ImageTk, Image

from tkinter import messagebox

import tkinter

from tkinter import filedialog

from cryptography.fernet import Fernet

import time

import os

from tkinter import ttk

key=""

filepath=""

#==================== EXIT FUNCTION ====================

def bahar():

bahar=tkinter.messagebox.askyesno("FICRYPT","DO YOU WANT TO EXIT ?")

if bahar>0:

main\_window.destroy()

#======================== HELPDESK FUNCTION ===================================

def help\_wind():

win=Toplevel()

win.geometry("1920x1080")

win.grab\_set()

tlbl=Label(win,text="HELPDESK",font=("times new roman",37,"bold"),bg="white",fg="red")

tlbl.place(x=0,y=0,width=1530,height=60)

img\_top=Image.open(r'C:\Users\RohitJadli\Desktop\Final Year Project\photos/1.png')

img\_top=img\_top.resize((1920,1080))

photoimg\_top=ImageTk.PhotoImage(img\_top)

flbl=Label(win,image=photoimg\_top)

flbl.place(x=0,y=55,width=1920,height=1080)

e\_lbl=Label(flbl,text="EMAIL",font=("times new roman",40,"bold"),fg="blue",bg="white")

e\_lbl.place(x=680,y=100)

r\_lbl=Label(flbl,text="rohitjadli03@gmail.com",font=("times new roman",40,"bold"),fg="blue",bg="white")

r\_lbl.place(x=510,y=300)

v\_lbl=Label(flbl,text="shankhwarvishal2001@gmail.com",font=("times new roman",40,"bold"),fg="blue",bg="white")

v\_lbl.place(x=410,y=450)

win.wm\_iconbitmap(r'C:\Users\RohitJadli\Desktop\Final Year Project\enc.ico')

win.mainloop()

#========================= DECRYPTION WINDOW ===================================

def prg\_dcr():

root = Tk()

root.title('FICRYPT')

root.geometry("720x300+400+220")

win.destroy()

def run2():

button1.destroy()

#======== PROGRESSBAR FUNC ============================

progressBar['maximum'] = 100

global g

for i in range(1,101):

time.sleep(0.01)

progressBar["value"] = i

progressBar.update()

g=i

lbl=Label(frame,font=("times new roman",20,"bold"),fg="green",bg="black")

lbl.place(x=350,y=150,height=50)

lbl.config(text=str(g)+"%")

messagebox.showinfo("SUCCESS","FILE DECRYPT SUCCESSFULLY")

def dec(filepath):

global key

with open('filekey.key', 'rb') as filekey:

key = filekey.read()

fernet = Fernet(key)

with open(filepath, 'rb') as enc\_file:

encrypted = enc\_file.read()

decrypted = fernet.decrypt(encrypted)

with open(filepath, 'wb') as dec\_file:

dec\_file.write(decrypted)

dec(filepath)

root.destroy()

#================= FRAME =======================

frame=Frame(root,bg="black")

frame.place(x=0,y=0,width=720,height=300)

plbl=Label(root,text="PROCESS",font=("times new roman",37,"bold"),bg="blue",fg="red")

plbl.place(x=0,y=0,width=720,height=70)

button1 = Button(frame,text="START",command=run2,font=("times new roman",22,"bold"),fg="white",bg="red")

button1.place(x=313,y=210,width=120)

progressBar = ttk.Progressbar(frame,orient="horizontal", length=680,mode="determinate")

progressBar.place(x=17,y=120)

root.wm\_iconbitmap(r'C:\Users\RohitJadli\Desktop\Final Year Project\enc.ico')

root.mainloop()

def decrypt\_wind():

global win

win=Toplevel()

win.geometry("720x320+400+220")

win.grab\_set()

def bahar():

bahar=tkinter.messagebox.askyesno("FICRYPT","DO YOU WANT TO EXIT ?")

if bahar>0:

win.destroy()

def openfile():

global filepath

filepath=filedialog.askopenfilename(title="Select File For Decryption",filetypes=[("All Files","\*.\*")])

lbl\_enc=Label(win,text=filepath,font=("times new roman",16,"bold"),bg="black",fg="white")

lbl\_enc.place(x=60,y=115,width=600,height=50)

#return filepath

def rmvfile():

lbl\_enc=Label(win,text="",font=("times new roman",16,"bold"),bg="black",fg="white")

lbl\_enc.place(x=60,y=115,width=600,height=50)

tlbl=Label(win,text="Selecting File For Decryption",font=("times new roman",37,"bold"),bg="blue",fg="red")

tlbl.place(x=0,y=0,width=720,height=70)

frame3=Frame(win,bg="black")

frame3.place(x=0,y=70,width=720,height=383)

rmv\_btn=Button(frame3,text="REMOVE FILE",command=rmvfile,font=("times new roman",15,"bold"),bd=3,relief=RIDGE,fg="white",bg="red")

rmv\_btn.place(x=520,y=180,width=150,height=50)

dcr\_btn=Button(frame3,text="DECRYPT",command=prg\_dcr,font=("times new roman",21,"bold"),bd=3,relief=RIDGE,fg="white",bg="red")

dcr\_btn.place(x=240,y=110,width=200,height=50)

add\_btn=Button(frame3,text="ADD FILE",command=openfile,font=("times new roman",15,"bold"),bd=3,relief=RIDGE,fg="white",bg="red")

add\_btn.place(x=30,y=180,width=150,height=50)

ext\_btn=Button(frame3,text="EXIT",command=bahar,font=("times new roman",20,"bold"),bd=3,relief=RIDGE,fg="white",bg="red")

ext\_btn.place(x=265,y=180,width=150,height=50)

win.wm\_iconbitmap(r'C:\Users\RohitJadli\Desktop\Final Year Project\enc.ico')

win.mainloop()

#======================= ENCRYPTION ==================================

# PROGRESS BAR AND ENCRYPTION FUNCTION WITH WINDOW

def prg\_enc():

root = Tk()

root.title('FICRYPT')

root.geometry("720x300+400+220")

win.destroy()

def run1():

button1.destroy()

progressBar['maximum'] = 100

global g

for i in range(1,101):

time.sleep(0.01)

progressBar["value"] = i

progressBar.update()

g=i

lbl=Label(frame,font=("times new roman",20,"bold"),fg="green",bg="black")

lbl.place(x=350,y=150,height=50)

lbl.config(text=str(g)+"%")

messagebox.showinfo("SUCCESS","FILE ENCRYPT SUCCESSFULLY")

root.destroy()

enc(filepath)

def enc(filepath):

global key

key = Fernet.generate\_key()

with open('filekey.key', 'wb') as filekey:

filekey.write(key)

with open('filekey.key', 'rb') as filekey:

key = filekey.read()

fernet = Fernet(key)

with open(filepath, 'rb') as file:

original = file.read()

encrypted = fernet.encrypt(original)

with open(filepath, 'wb') as encrypted\_file:

encrypted\_file.write(encrypted)

'''my\_file="{}".format(filepath)

base = os.path.splitext(my\_file)[0]

os.rename(my\_file, base + '.enc')'''

#================= FRAME =======================

frame=Frame(root,bg="black")

frame.place(x=0,y=0,width=720,height=300)

plbl=Label(root,text="PROCESS",font=("times new roman",37,"bold"),bg="blue",fg="red")

plbl.place(x=0,y=0,width=720,height=70)

button1 = Button(frame,text="START",command=run1,font=("times new roman",22,"bold"),fg="white",bg="red")

button1.place(x=313,y=210,width=120)

progressBar = ttk.Progressbar(frame,orient="horizontal", length=680,mode="determinate")

progressBar.place(x=17,y=120)

root.wm\_iconbitmap(r'C:\Users\RohitJadli\Desktop\Final Year Project\enc.ico')

root.mainloop()

def encrypt\_wind():

global win

win=Toplevel()

win.geometry("720x320+400+220")

win.grab\_set()

def bahar():

bahar=tkinter.messagebox.askyesno("FICRYPT","DO YOU WANT TO EXIT ?")

if bahar>0:

win.destroy()

def openfile():

global filepath

filepath=filedialog.askopenfilename(title="Select File For Encryption",filetypes=[("All Files","\*.\*")])

lbl\_enc=Label(win,text=filepath,font=("times new roman",16,"bold"),bg="black",fg="white")

lbl\_enc.place(x=60,y=115,width=600,height=50)

def rmvfile():

lbl\_enc=Label(win,text="",font=("times new roman",16,"bold"),bg="black",fg="white")

lbl\_enc.place(x=60,y=115,width=600,height=50)

tlbl=Label(win,text="Selecting File For Encryption",font=("times new roman",37,"bold"),bg="blue",fg="red")

tlbl.place(x=0,y=0,width=720,height=70)

frame2=Frame(win,bg="black")

frame2.place(x=0,y=70,width=720,height=383)

rmv\_btn=Button(frame2,text="REMOVE FILE",command=rmvfile,font=("times new roman",15,"bold"),bd=3,relief=RIDGE,fg="white",bg="red")

rmv\_btn.place(x=520,y=180,width=150,height=50)

enr\_btn=Button(frame2,text="ENCRYPT",command=prg\_enc,font=("times new roman",21,"bold"),bd=3,relief=RIDGE,fg="white",bg="red")

enr\_btn.place(x=240,y=110,width=200,height=50)

add\_btn=Button(frame2,text="ADD FILE",command=openfile,font=("times new roman",15,"bold"),bd=3,relief=RIDGE,fg="white",bg="red")

add\_btn.place(x=30,y=180,width=150,height=50)

ext\_btn=Button(frame2,text="EXIT",command=bahar,font=("times new roman",20,"bold"),bd=3,relief=RIDGE,fg="white",bg="red")

ext\_btn.place(x=265,y=180,width=150,height=50)

win.wm\_iconbitmap(r'C:\Users\RohitJadli\Desktop\Final Year Project\enc.ico')

win.mainloop()

main\_window=Tk()

#=========== BACKGROUND IMAGE =========================

bg1=Image.open(r"C:\Users\RohitJadli\Desktop\Final Year Project\photos\bg1.png")

bg1=bg1.resize((1920,1080))

bgimg1=ImageTk.PhotoImage(bg1)

bg\_lbl=Label(main\_window,image=bgimg1)

bg\_lbl.place(x=0,y=0,width=1920,height=1080)

#=========== UPPER LABEL ================================

tlbl=Label(bg\_lbl,text="FICRYPT - FILE ENCRYPTION SOFTWARE",font=("times new roman",40,"bold"),bg="black",fg="red")

tlbl.place(x=0,y=0,width=1530,height=70)

#================ ENCRYPTION BUTTON =========================

img1=Image.open(r'C:\Users\RohitJadli\Desktop\Final Year Project\photos/data-encryption.png')

img1=img1.resize((160,140))

fotoimg1=ImageTk.PhotoImage(img1)

b1=Button(bg\_lbl,image=fotoimg1,cursor="hand2",command=encrypt\_wind)

b1.place(x=100,y=300,width=220,height=220)

b1\_1=Button(bg\_lbl,text="ENCRYPT",cursor="hand2",command=encrypt\_wind,font=("times new roman",20,"bold"),bg="black",fg="red")

b1\_1.place(x=100,y=480,width=220,height=60)

img2=Image.open(r'C:\Users\RohitJadli\Desktop\Final Year Project\photos/decrypt.png')

img2=img2.resize((220,220))

fotoimg2=ImageTk.PhotoImage(img2)

b2=Button(bg\_lbl,image=fotoimg2,cursor="hand2",command=decrypt\_wind)

b2.place(x=450,y=300,width=220,height=220)

b2\_1=Button(bg\_lbl,text="DECRYPT",cursor="hand2",command=decrypt\_wind,font=("times new roman",19,"bold"),bg="black",fg="red")

b2\_1.place(x=450,y=480,width=220,height=60)

img3=Image.open(r'C:\Users\RohitJadli\Desktop\Final Year Project\photos/help-desk.png')

img3=img3.resize((210,180))

fotoimg3=ImageTk.PhotoImage(img3)

b3=Button(bg\_lbl,image=fotoimg3,cursor="hand2",command=help\_wind)

b3.place(x=800,y=300,width=220,height=220)

b3\_1=Button(bg\_lbl,text="HELPDESK",cursor="hand2",command=help\_wind,font=("times new roman",19,"bold"),bg="black",fg="red")

b3\_1.place(x=800,y=480,width=220,height=60)

#===================EXIT================================

img4=Image.open(r'C:\Users\RohitJadli\Desktop\Final Year Project\photos/exit.png')

img4=img4.resize((160,140))

fotoimg4=ImageTk.PhotoImage(img4)

b4=Button(bg\_lbl,image=fotoimg4,cursor="hand2",command=bahar)

b4.place(x=1150,y=300,width=220,height=220)

b4\_1=Button(bg\_lbl,text="EXIT",cursor="hand2",command=bahar,font=("times new roman",19,"bold"),bg="black",fg="red")

b4\_1.place(x=1150,y=480,width=220,height=60)

main\_window.geometry("1920x1080")

main\_window.title("Ficrypt - File Encryption Software")

main\_window.wm\_iconbitmap(r'C:\Users\RohitJadli\Desktop\Final Year Project\enc.ico')

main\_window.mainloop()

**RESULT**



This is the main window of our software.

In this main window there are four options :-

1. ENCRYPT
2. DECRYPT
3. HELPDESK
4. EXIT

**ENCRYPTION:**

Now we have to encrypt files click on ENCRYPT button.



when we click on the ENCRYPT button this above window will appear on the screen.

In this window, there are also four options

1. ENCRYPT – for encryption
2. ADD FILE – for adding file
3. REMOVE FILE – for remove file
4. EXIT – for exit this window

Now we have to click on ADD FILE button for adding files for encryption process.



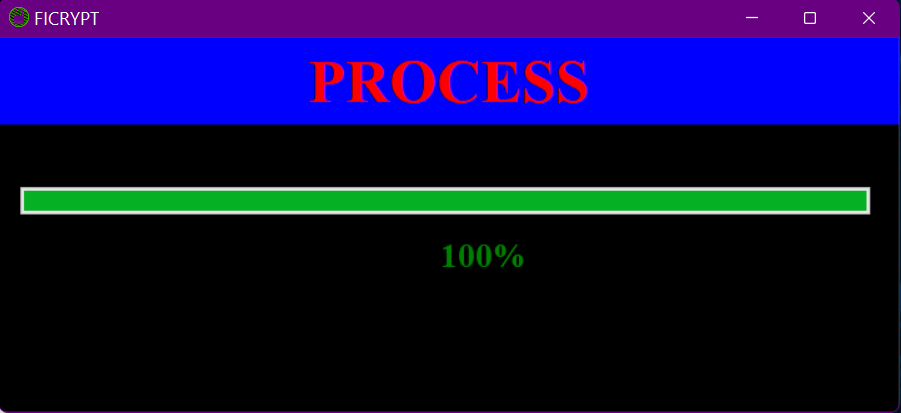
If you are choosing wrong file then click on the REMOVE FILE button and after we have to click on ADD FILE button to add new file



After adding file we have to click on ENCRYPT button for the main encryption process.



Now click on this START button to start the process of encryption.



After this process is completed then it will a success message.

**DECRYPTION:**



This is the main window of our software.

In this main window there are four options :-

1. ENCRYPT
2. DECRYPT
3. HELPDESK
4. EXIT

**DECRYPTION:**

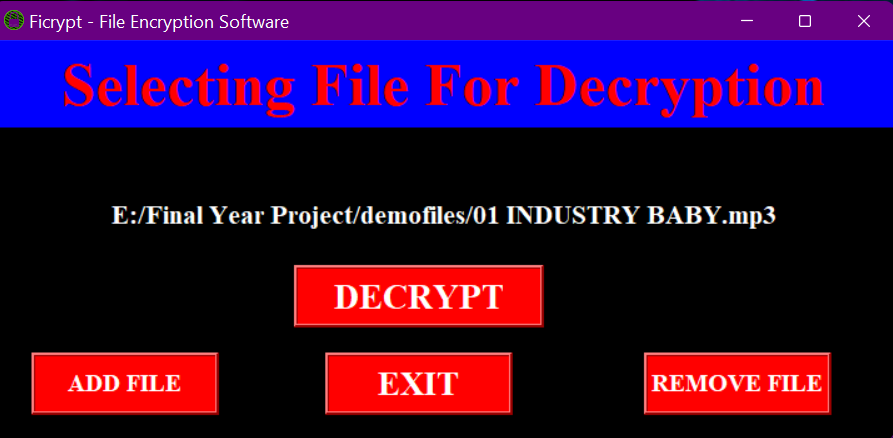
Now we have to encrypt files click on DECRYPT button.

when we click on the DECRYPT button this above window will appear on the screen.

In this window, there are also four options

1. DECRYPT – for decryption
2. ADD FILE – for adding file
3. REMOVE FILE – for remove file
4. EXIT – for exit this window

Now we have to click on ADD FILE button for adding files for decryption process.



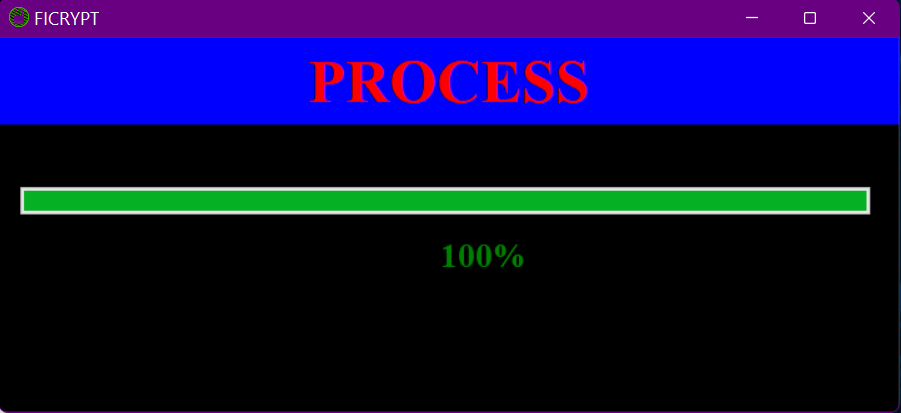
If you are choosing wrong file then click on the REMOVE FILE button and after we have to click on ADD FILE button to add new file



After adding file we have to click on DECRYPT button for the main decryption process.



Now click on this START button to start the process of decryption.



After this process is completed then it will a success message.

**REFERENCES**

1. Martin C. Brown, “Python: The Complete Reference”, Published in 20 March 2018, Book For Python
2. R. Nageswara Rao, “ Core Python Programming, 3ed: Covers Fundamentals to advanced topics like OOPs, Exceptions, Data Structures, Files, Thread, Net” Published in 1 September 2021, Book For Python
3. Stallings Williams, “Cryptography And Network Security – Principles And Practice (fourth edition)”, Published in 30 June 2017, Book For Cryptography
4. Nael Stephenson, “CRYPTONOMICON”, Published in 3 May 2000, Book For Cryptography