Department of Computer Science and Technology

Project Report

Terminal Based Cryptography Framework
(**GoCrypt**)

Software Engineering (CSH335B)

Team
VANSHIKA THAPA (2K23CSUN01364, CSTI-3A)
SOHAM TANWAR (2K23CSUN01358, CSTI-3A)
PRAVEEN KUMAR SHARMA (2K23CSUN01353, CSTI-3A)

Under the supervision of
Dr. Deepti Thakral, Professor, DoCST

MANAV RACHNA
||vidyanatariksha||

MANAV RACHNA
UNIVERSITY
Declared as State Private University vide Haryana Act 26 of 2014

Manav Rachna University, Sector-43, Aravali Hills, Faridabad : Year 2024

0. Contribution to SRS

- Vanshika Thapa:
  - Focused on the formatting and design of the SRS document.
  - Ensured that the layout is clear, consistent, and adheres to best practices for technical documentation.
- Soham Tanwar:
  - Responsible for creating the use case diagram, illustrating the interactions between users and the cryptography framework.
  - This diagram helps to visualize user requirements and system functionalities.
- Praveen Kumar Sharma:
  - Worked on designing and distributing the survey to gather requirements and feedback from stakeholders.
  - Analyzed the collected data to draw insights for the SRS.
- Collaborative Efforts:
  - All team members contributed to writing research content and coding efforts to ensure a comprehensive understanding of the cryptography framework.
  - Engaged in brainstorming sessions to identify and articulate generic requirements for the framework, ensuring that all aspects of user needs and system functionalities were considered.

1. Introduction

1.1 Purpose

This document provides a comprehensive description of the Cryptography Framework. It outlines the framework's purpose, key features, and capabilities, including its encryption, decryption, and key management systems. The document will detail the interfaces of the framework, its operational constraints, and how it securely responds to external inputs and data. The intended audience includes both stakeholders and developers, and this proposal is presented for approval to the organization's security and IT teams.

1.2 Scope

This Cryptography Framework is designed to provide essential tools for a cybersecurity professional or cryptography enthusiast. The framework's primary objective is to enhance productivity by automating a variety of cryptographic operations that would otherwise be done manually. By improving efficiency and simplifying cryptographic tasks, this framework will meet user needs while being intuitive and easy to use.

1.3 Definitions, Acronyms, and Abbreviations

Hashing : One-way data transformation
Password Cracking : Guessing passwords methods
ROT : Character shift encryption
Bases : Number systems for encoding
SRS : Software requirements document
CLI : Text-based command interface
Encryption/Decryption : Secure data conversion
Encoding/Decoding : Data format conversion

1.4 References

OWASP Foundation
NIST Cryptographic Standards and Guidelines
GitHub
CyberChef
Serious Cryptography by Jean-Philippe Aumasson

2. The Overall Description

2.1 Product Perspective

2.1.1 System Interfaces

- Operating System (OS): The framework will interact with the OS for file operations (input/output), process management, and memory allocation. It will be compatible with Linux, macOS, and Windows operating systems.
- Command-Line Interface (CLI): The framework will provide a user-friendly CLI to accept commands for various cryptographic operations, including encryption, decryption, hashing, and encoding/decoding. The CLI will handle argument parsing, error handling, and display of results.
- File System: The framework will interface with the file system to read input files (for data encryption/decryption) and write the processed data to output files. It will support common file types.
- External Libraries: The system will interact with cryptographic libraries (such as OpenSSL or similar) to perform the core encryption, decryption, and hashing functions, leveraging well-established algorithms and standards.

2.1.2 Hardware Interfaces

- Processor (CPU): The framework will rely on the system's CPU for performing cryptographic operations such as hashing, encryption, and decryption. Since Go is highly efficient, it will work on both x86_64 and ARM-based architectures.
- Memory (RAM): The application will use system memory for temporarily storing data during processing (such as encrypted/decrypted data). However, no significant memory demands are expected, and it can run on systems with as little as 512MB of RAM.
- Storage (Disk): The framework requires minimal disk space to store the Go binaries and libraries. Additionally, it will read from and write

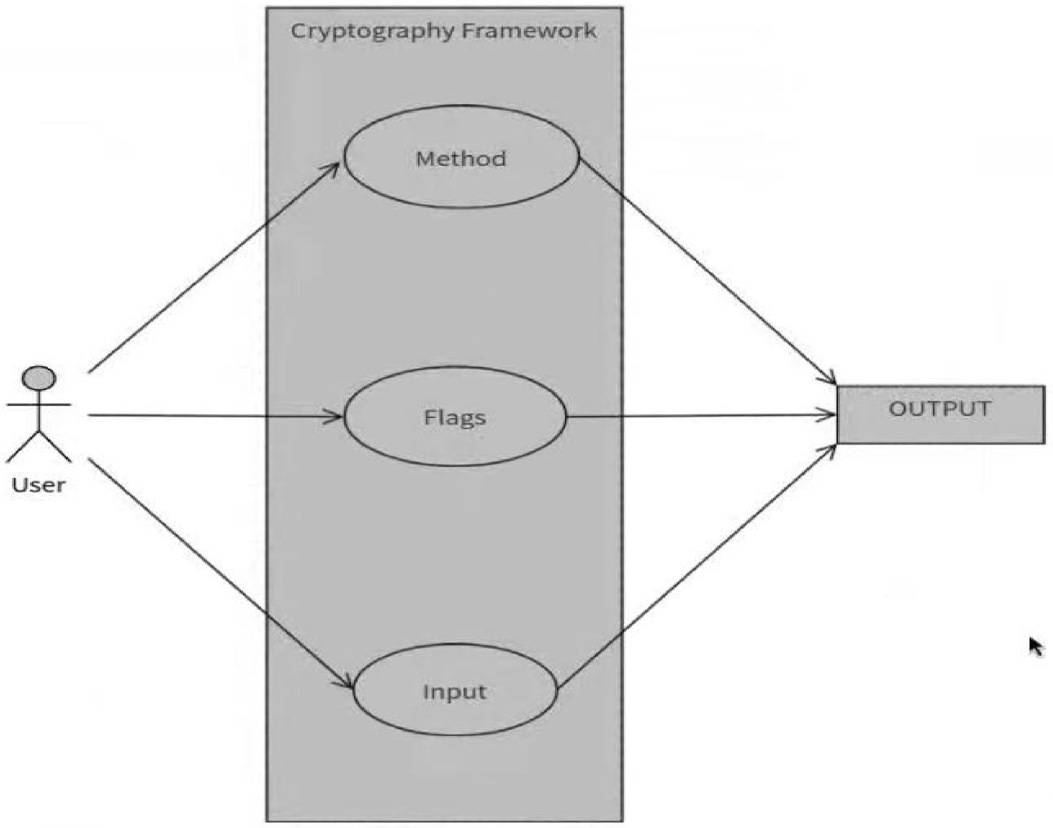
to disk for file-based operations such as encrypting/decrypting or saving output data.

- Input/Output Devices: The application will interface with standard input (keyboard) and output (display/terminal) devices for user interaction via the command line interface. No specialized hardware is required.

2.2 Product Functions

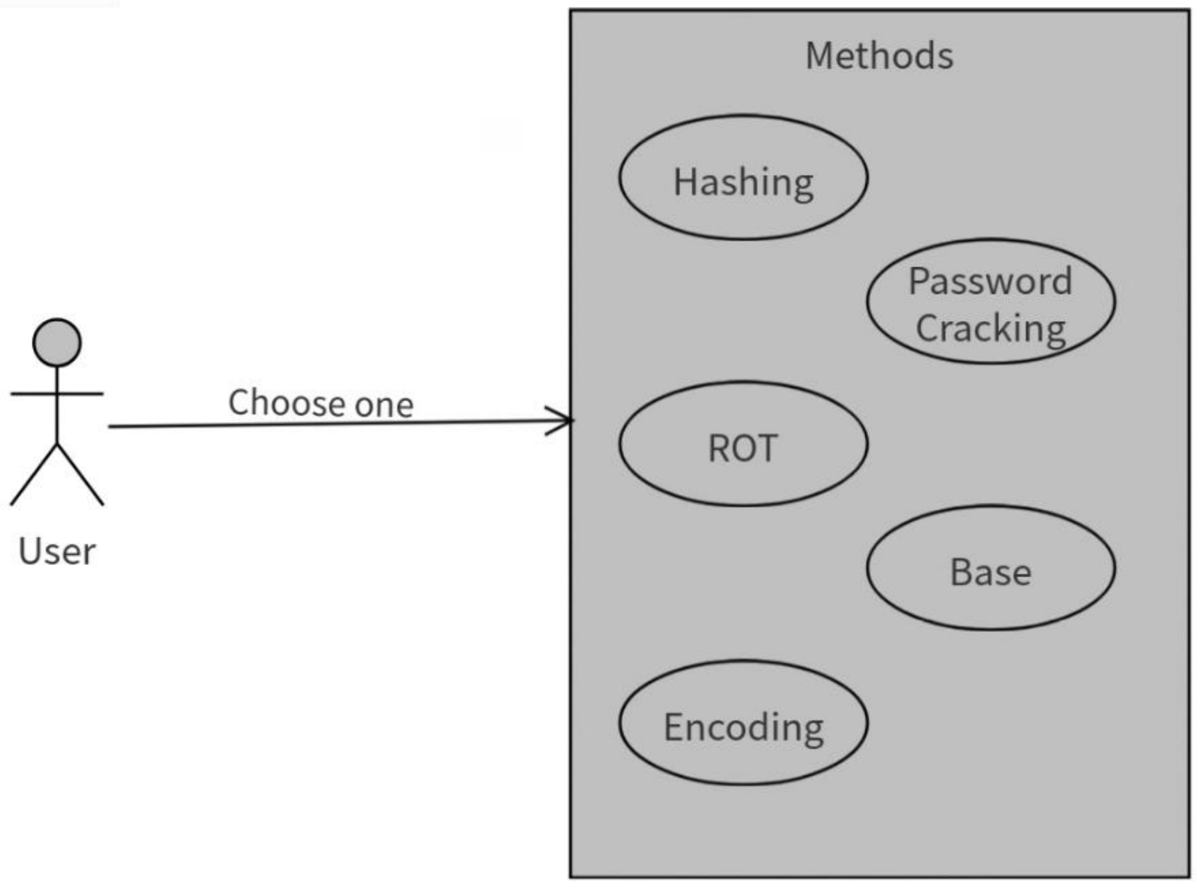
- Symmetric Encryption
    - Implements AES encryption in multiple modes, including CBC (Cipher Block Chaining), GCM (Galois/Counter Mode), and CTR (Counter Mode).
    - Supports key sizes of 128, 192, and 256 bits for varying levels of security.
- Asymmetric Encryption
    - Provides RSA encryption and decryption functions.
    - Supports RSA key sizes of 2048, 3072, and 4096 bits to accommodate different security and performance needs.
- Digital Signatures
    - Implements RSA and ECDSA (Elliptic Curve Digital Signature Algorithm) for generating and verifying digital signatures, ensuring data authenticity and integrity.
- Hashing
    - Implements cryptographic hash functions, including SHA-2 and SHA-3, to provide data integrity checks and digital fingerprints.
- Random Number Generation
    - Offers a cryptographically secure random number generator (CSPRNG) for generating secure random numbers needed for keys, initialization vectors, and nonces.

2.3 System Environment



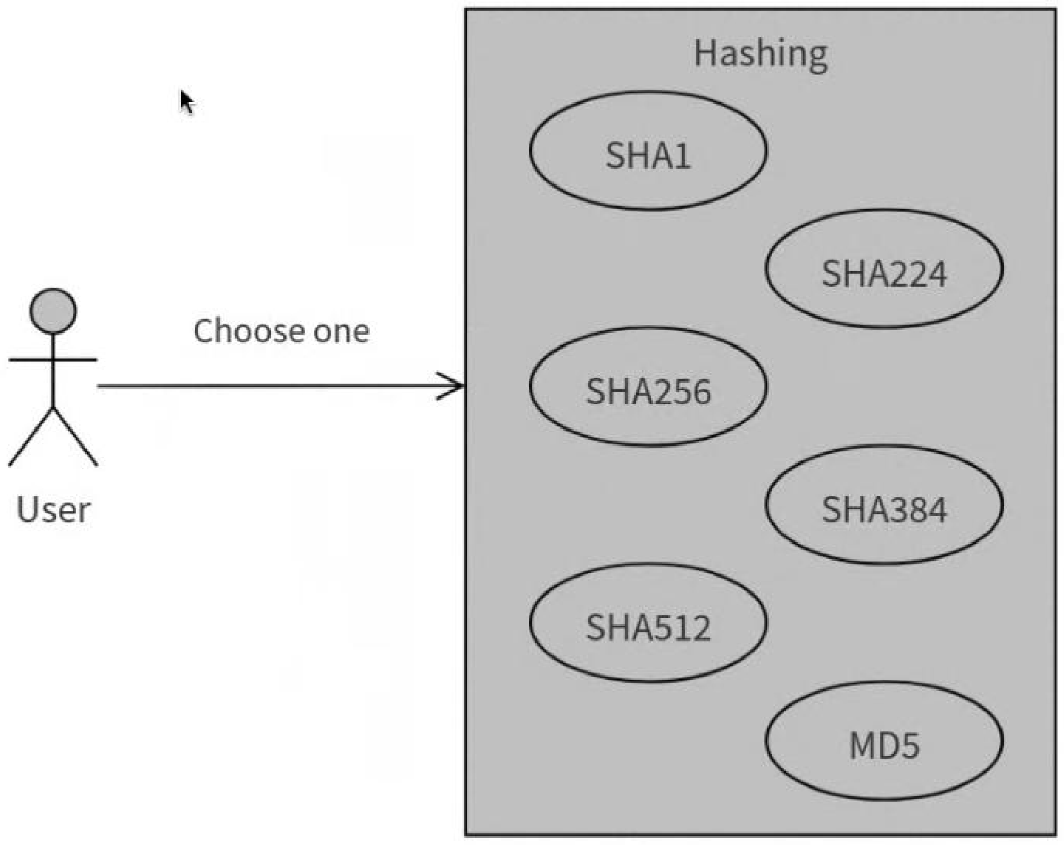
- The user selects the cryptographic method, specifies any necessary flags, and provides the input data, which is then processed to generate the desired output.

2.3.1 Cryptographic Methods



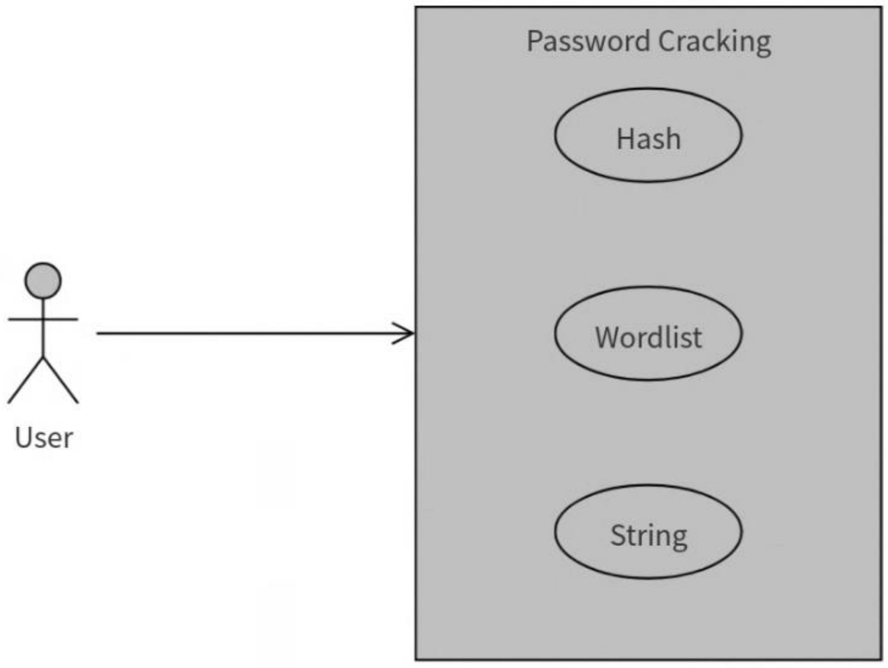
- The user can select from the following cryptographic methods:
  - Hashing
  - Password Cracking
  - ROT
  - Base Conversion
  - Encoding
- These methods enable various transformations and representations of data for secure processing.

2.3.1.1 Hashing



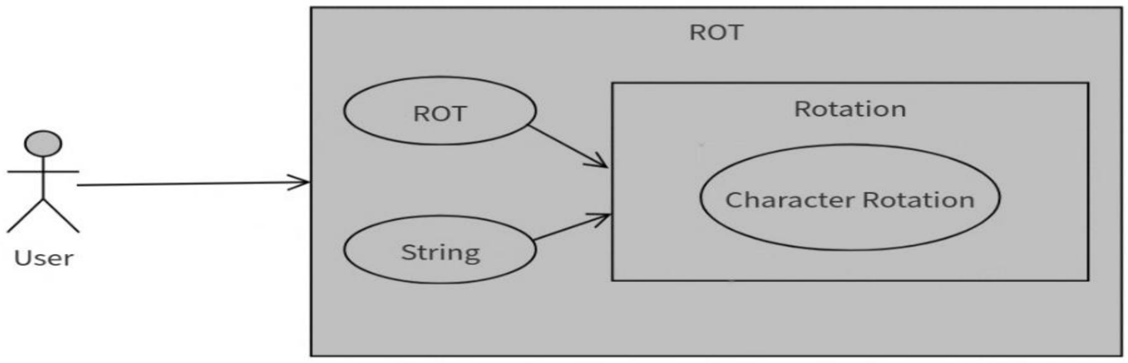
- The user can choose from the following hashing algorithms:
  - MD5
  - SHA-1
  - SHA-224
  - SHA-256
  - SHA-384
  - SHA-512
- These algorithms generate fixed-size hash values from the input data, providing various levels of security and output size.

2.3.1.2 Password Cracking



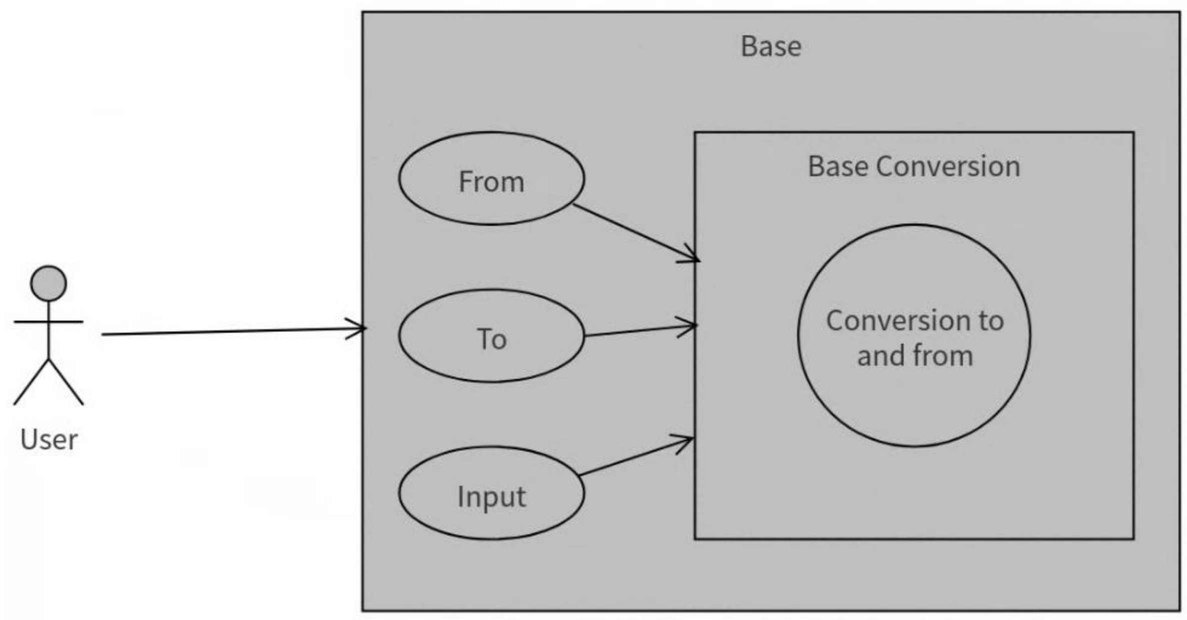
- In the Password Cracking method, users can input a hashed password and attempt to crack it by using a wordlist for brute-force attacks or by manually testing specific strings.

2.3.1.3 ROT



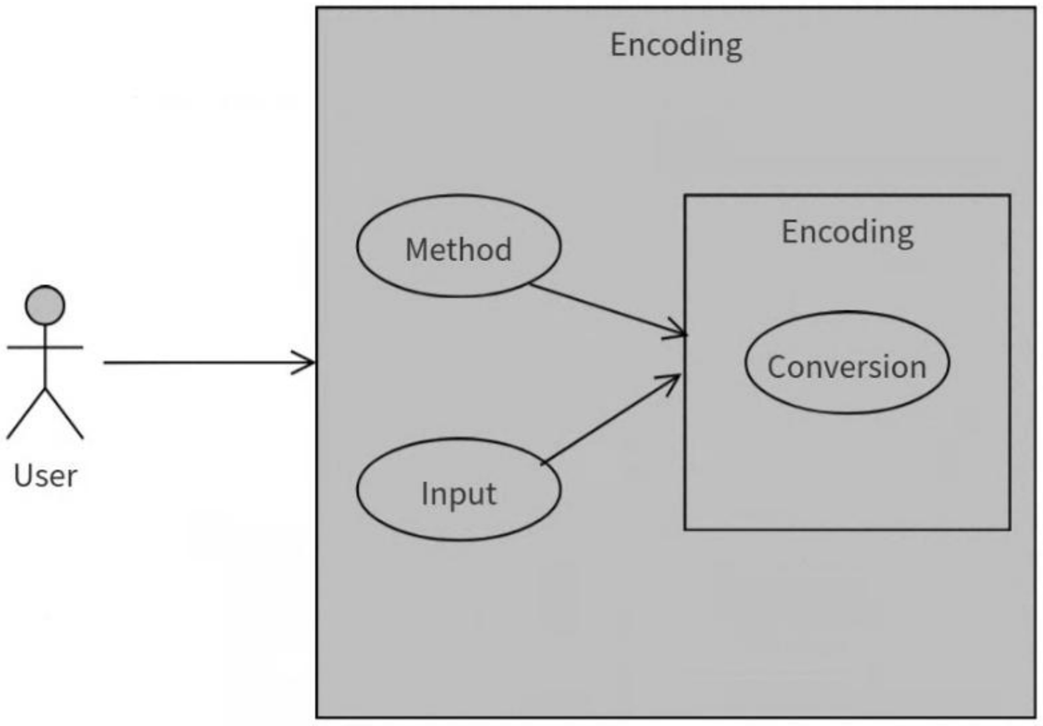
- In the ROT method, users can apply a rotation cipher, which shifts the characters in the input by a specified number of positions in the alphabet. This simple substitution cipher, like ROT13, is often used for basic obfuscation or encoding text by rotating characters through a fixed offset.

2.3.1.4 Base Conversion



- In the Base Conversion method, users can convert numerical data between different bases, such as Base 2 (binary), Base 10 (decimal), and Base 16 (hexadecimal). This allows users to easily switch between various numeral systems, commonly used in computing and cryptography for data representation and manipulation.

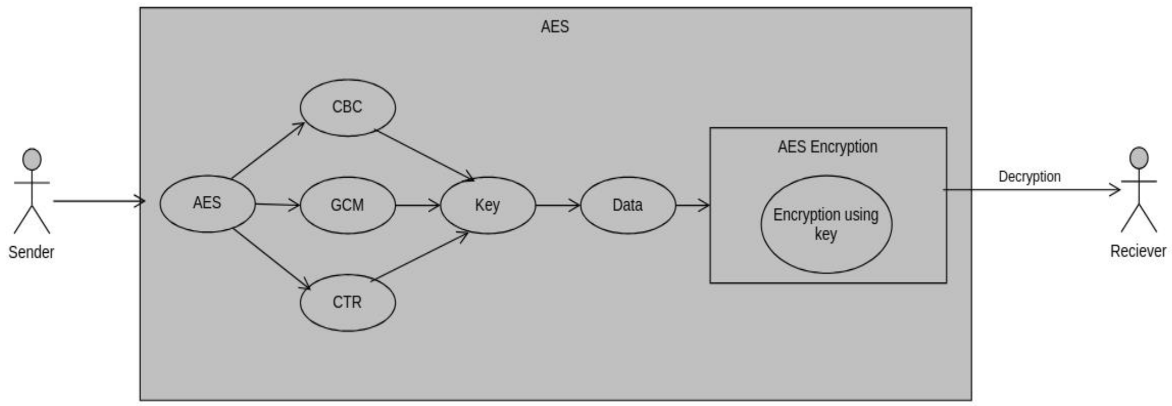
2.3.1.5 Encoding



- In the Encoding method, users can convert data into a specific format for secure transmission or storage. This includes encoding schemes like Base Encoding, URL Encoding, and others, which transform data into a readable or transmittable form, ensuring it can be safely handled by different systems without corruption.

2.4 Functional Requirements

2.4.1 Symmetric Encryption (AES)



2.4.1.1 Brief Desc ription

- This application offers robust symmetric encryption capabilities using AES (Advanced Encryption Standard). Users can encrypt and decrypt data in various modes, including CBC (Cipher Block Chaining), GCM (Galois/Counter Mode), and CTR (Counter Mode), with key sizes of 128, 192, or 256 bits.
- The user-friendly interface allows users to select encryption modes, generate or input encryption keys, and easily manage plaintext and encrypted data. The application ensures secure handling of keys and provides clear feedback during encryption and decryption processes, making it accessible while maintaining strong security standards.

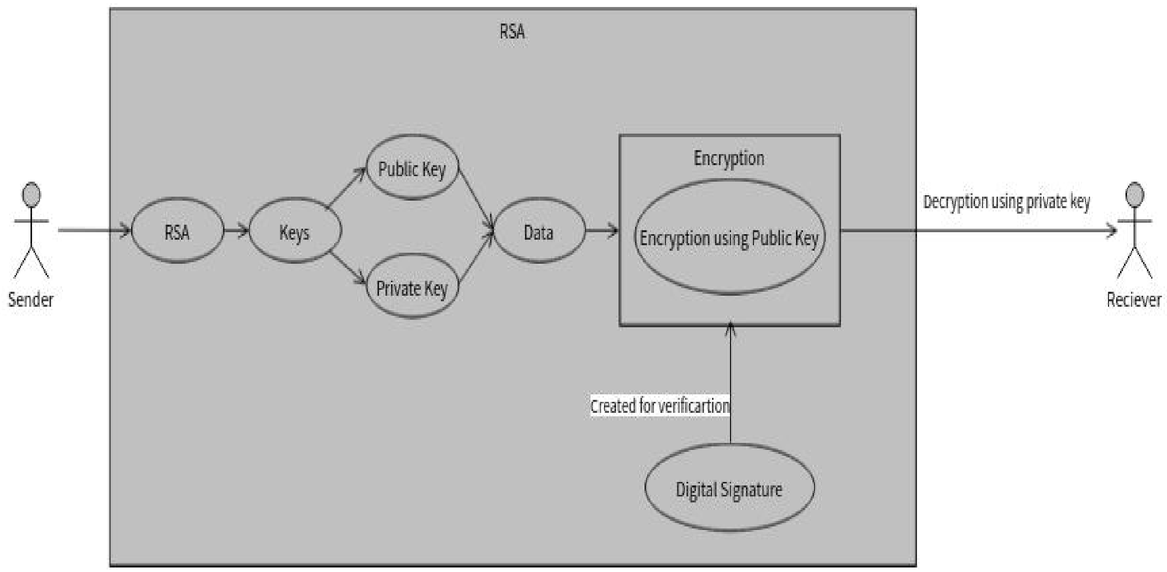
2.4.1.2 Initial Step-By-Step Description

1. Selecting the Encryption Tool:
   - The user opens the encryption application or service designed to handle AES encryption.
2. Choosing Encryption Mode:
   - The user is presented with options for different AES modes (CBC, GCM, CTR).
   - The user selects the mode based on their needs:
     - CBC for secure file encryption.

        - GCM for both encryption and integrity verification (good for network communications).
        - CTR for fast encryption of large amounts of data.
3. Setting Key Size:
    - The user is prompted to choose a key size (128, 192, or 256 bits).
    - The user selects a key size based on the desired level of security, keeping in mind that larger keys offer stronger security.
4. Generating or Inputting the Key:
    - The user can either generate a secure key using the application or input an existing key.
    - If generating a key, the user may see a message confirming the key's generation and security level.
5. Entering Plaintext Data:
    - The user is provided with a field to enter or upload the plaintext data they wish to encrypt (e.g., text, files).
    - The user inputs the data.
6. Performing the Encryption:
    - After confirming all settings (mode, key size, plaintext), the user clicks an "Encrypt" button.
    - The application processes the data and displays a success message along with the encrypted output, which may be presented as a file or string.
7. Storing or Transmitting Encrypted Data:
    - The user has options to save the encrypted data to a file or copy it to the clipboard for sharing.
    - The user is advised to securely store the encryption key, as it will be needed for decryption.
8. Decrypting Data:
    - When the user wants to access the original data, they return to the application.
    - The user selects the decryption option, inputs the encrypted data, and the same key used for encryption.

- ○ The user selects the same encryption mode used during encryption.
- ○ The user clicks "Decrypt," and the application displays the decrypted plaintext.

2.4.2 Asymmetric Encryption


RSA
Sender
RSA
Keys
Public Key
Private Key
Data
Encryption
Encryption using Public Key
Decryption using private key
Reciever
Created for verificartion
Digital Signature

2.4.2.1 Brief Description

- This application provides users with robust asymmetric encryption capabilities, specifically using RSA, to securely encrypt and decrypt data. Users can generate RSA key pairs with key sizes of 2048, 3072, or 4096 bits. The application also supports digital signatures through RSA and ECDSA, allowing users to sign and verify documents for authenticity.
- Additionally, users can hash data using SHA-2 and SHA-3 cryptographic hash functions and utilize a cryptographically secure random number generator (CSPRNG) for secure key generation and other needs. The user-friendly interface guides users through encryption, decryption, signing, and verification processes, ensuring a seamless and secure experience.

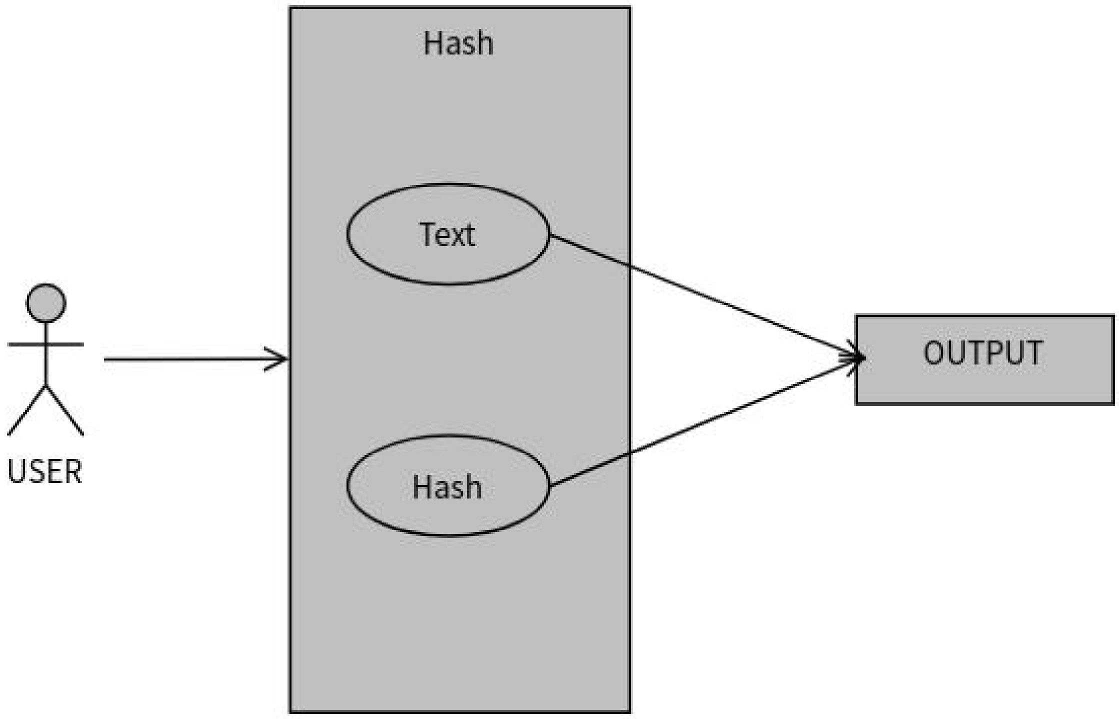
2.4.2.2 Initial Step-By-Step Description

1. Accessing the Encryption Tool:
    - The user opens the application or service that supports RSA encryption and digital signatures.
2. Generating RSA Key Pair:
    - The user selects the option to generate an RSA key pair.
    - The user is prompted to choose a key size (2048, 3072, or 4096 bits).
    - After selection, the application generates the public and private keys, displaying them securely.
    - The user is advised to save the private key securely, as it will be needed for decryption and signing.
3. Encrypting Data:
    - The user navigates to the encryption section.
    - The user inputs or uploads the plaintext data they wish to encrypt.
    - The user selects the public key for encryption.
    - The user clicks "Encrypt," and the application outputs the encrypted data, which the user can save or share.
4. Decrypting Data:
    - When the user needs to access the encrypted data, they return to the application.
    - The user selects the decryption option, inputs the encrypted data, and selects their private key.
    - The user clicks "Decrypt," and the application displays the original plaintext.
5. Creating Digital Signatures:
    - The user accesses the digital signature feature.
    - The user uploads the data (e.g., a document) they want to sign.
    - The user selects their private key for signing.
    - The user clicks "Sign," and the application generates a digital signature, which can be attached to the document.
6. Verifying Digital Signatures:

- The user receives a signed document and wants to verify its authenticity.
- The user uploads the signed document and the accompanying digital signature.
- The user selects the public key of the signer.
- The user clicks “Verify,” and the application confirms whether the signature is valid.

7. Hashing Data:
    - The user can access the hashing feature to hash data using SHA-2 or SHA-3.
    - The user inputs the data they wish to hash and selects the desired hashing algorithm.
    - The user clicks “Hash,” and the application outputs the hash value.
8. Using Cryptographically Secure Random Number Generation:
    - The user accesses the random number generation feature when they need secure random values (e.g., for key generation).
    - The user clicks “Generate Random Number,” and the application provides a cryptographically secure random number.

2.4.3 Hashing



2.4.3.1 Brief Description

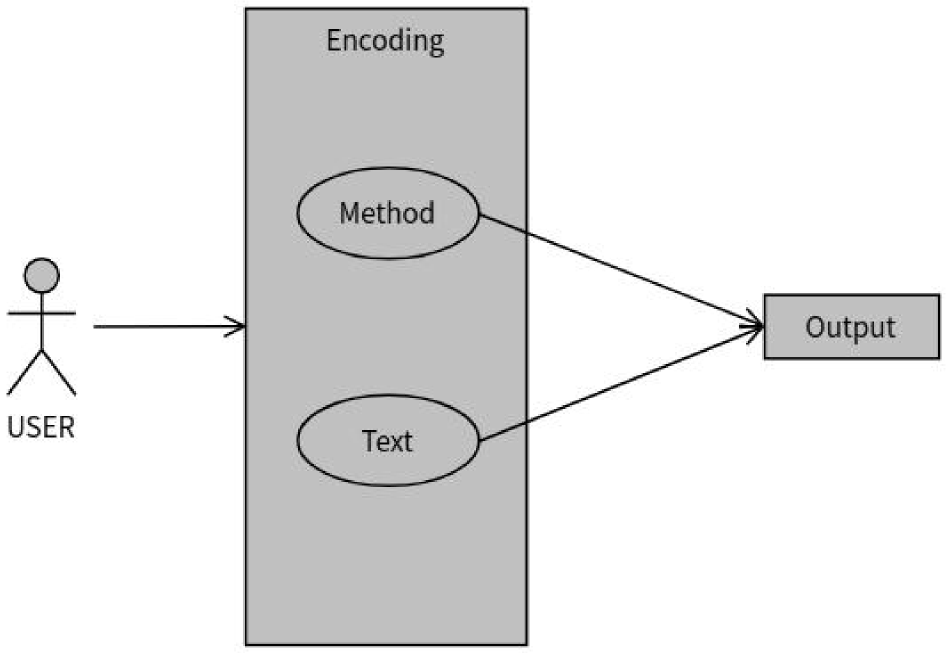
- This application provides users with robust hashing capabilities using SHA-2 and SHA-3 cryptographic hash functions. Hashing is essential for data integrity verification and secure storage of sensitive information like passwords. Users can hash data quickly and reliably, ensuring that the output is a fixed-size hash value that uniquely represents the input data.

2.4.3.2 Initial Step-By-Step Description

1. Accessing the Hashing Tool:
   - The user opens the application and navigates to the hashing section.
2. Selecting the Hashing Algorithm:
   - The user chooses between SHA-2 and SHA-3 from the available options, based on their security needs.
3. Entering Data to Hash:
   - The user inputs the data they want to hash into a designated field (e.g., text, files).

4. Initiating the Hashing Process:
    - The user clicks the “Hash” button to process the input data.
5. Viewing the Hash Output:
    - The application generates the hash value and displays it to the user, typically in hexadecimal format.
6. Copying or Saving the Hash:
    - The user can copy the generated hash for immediate use or save it for future reference, such as for integrity checks.

2.4.4 Encoding



2.4.4.1 Brief Overview

- Encoding is the process of converting data from one format to another, typically to facilitate storage, transmission, or processing. Unlike encryption, which aims to protect data confidentiality, encoding is primarily about transforming data into a different format for usability and compatibility. Common encoding methods include Base64, URL encoding, and hexadecimal encoding. These methods are widely used in web development, data transfer, and multimedia applications.

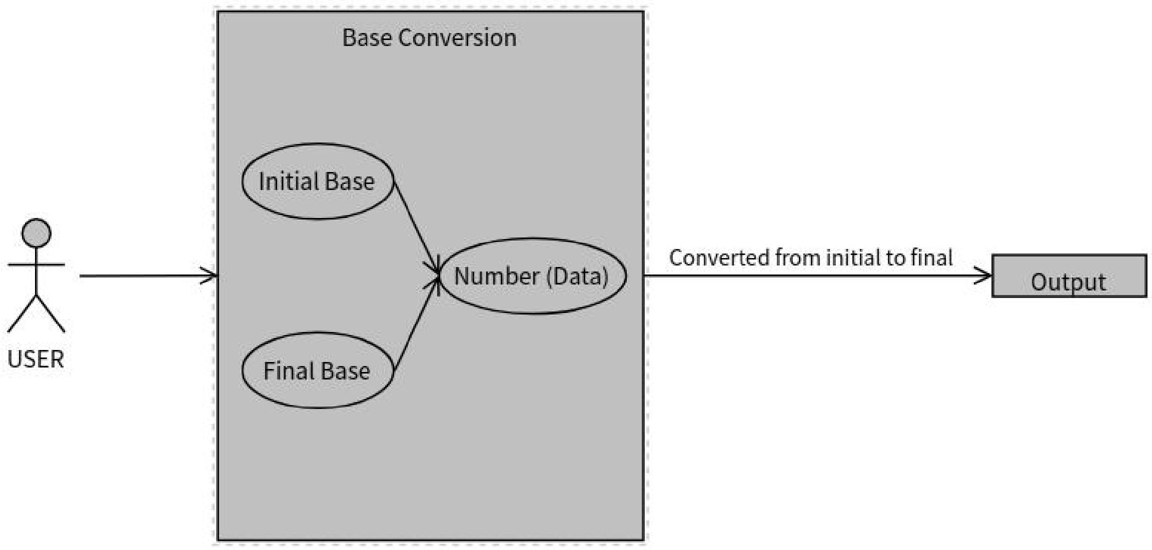
2.4.4.2 Initial Step-By-Step Description

1. Identifying the Data:
    - The user determines the data that needs to be encoded, such as text, binary data, or multimedia files.
2. Choosing an Encoding Scheme:
    - The user selects an appropriate encoding scheme based on the use case:
        - Base64: Commonly used for encoding binary data in text formats, especially for email attachments and web data.
        - URL Encoding: Transforms special characters in URLs into a format that can be transmitted over the internet (e.g., spaces become `%20`).
        - Hexadecimal Encoding: Represents binary data in a hexadecimal format, useful in programming and data analysis.
3. Encoding the Data:
    - The user inputs the data into an encoding tool or software.
    - The user selects the desired encoding method and initiates the encoding process.
4. Viewing the Encoded Output:
    - The application generates the encoded data and displays it to the user in the chosen format.
5. Using the Encoded Data:

- The user can copy the encoded output for immediate use, such as embedding in web pages, storing in databases, or sending over networks.

6. Decoding (if necessary):
    - If the user later needs to revert to the original data, they can use a decoding tool or function that corresponds to the chosen encoding scheme.

2.4.5 Base Conversion



2.4.5.1 Brief Description

- Base conversion is the process of converting numbers from one numeral system (base) to another. Common bases include decimal (base 10), binary (base 2), octal (base 8), and hexadecimal (base 16). Base conversion is essential in various fields, including computer science, digital electronics, and data encoding, as different systems use different bases for representation.

2.4.5.2 Initial Step-By-Step Description

1. Identifying the Original Base:
    - Determine the base of the number you wish to convert. Common bases include:
        - Binary (base 2)
        - Decimal (base 10)
        - Octal (base 8)
        - Hexadecimal (base 16)
2. Choosing the Target Base:
    - Decide which base you want to convert the number to.
3. Converting to Decimal (if necessary):
    - If converting from a base other than decimal, first convert the original number to decimal:
        - For binary: Multiply each bit by 2n2^n2n, where nnn is the position of the bit from the right (starting from 0).
        - For octal: Multiply each digit by 8n8^n8n.
        - For hexadecimal: Multiply each digit by 16n16^n16n.
    - Sum the results to get the decimal equivalent.
4. Converting from Decimal to the Target Base:
    - Divide the decimal number by the target base and record the remainder.
    - Continue dividing the quotient by the target base until the quotient is 0, recording remainders at each step.
    - The remainders, read in reverse order, give the equivalent number in the target base.
5. Formatting the Result:
    - Ensure the final result is presented correctly for the target base (e.g., using appropriate digits for hexadecimal).

2.5 Non-Functional Requirements

2.5.1 Performance

- The framework will be meticulously optimized to achieve exceptional performance levels, ensuring that it operates with minimal overhead. This means that the efficiency of the GoCrypt framework will be closely comparable to that of native Go cryptographic packages. By prioritizing performance, users can expect swift execution times and reduced latency in cryptographic operations, thereby enhancing the overall responsiveness of applications that rely on this framework.

2.5.2 Security

- In terms of security, all implementations within the GoCrypt framework will strictly adhere to the latest cryptographic standards and best practices. This commitment to security ensures that the framework will incorporate state-of-the-art encryption techniques and algorithms, safeguarding against contemporary threats and vulnerabilities. Regular updates will be implemented to align with evolving security protocols, thus providing users with robust protection for their sensitive data.

2.6.3 Usability

- To enhance usability, the API will be designed to be intuitive and user-friendly, allowing developers to easily integrate cryptographic functionalities into their applications. Comprehensive and thorough documentation will accompany the API, detailing usage instructions, code examples, and best practices. This extensive documentation aims to facilitate a seamless onboarding experience for new users, ensuring that both novice and experienced developers can effectively leverage the framework's capabilities without unnecessary hurdles.

2.5.4 Compatibility

- The GoCrypt framework will be built with compatibility in mind, ensuring that it seamlessly integrates with standard Go tools and development practices. This compatibility allows developers to incorporate GoCrypt into their existing workflows without disruption.

By adhering to established conventions and standards within the Go ecosystem, users can expect a smooth integration process, making it easier to adopt and utilize the framework in their projects.

3. Requirements Specification

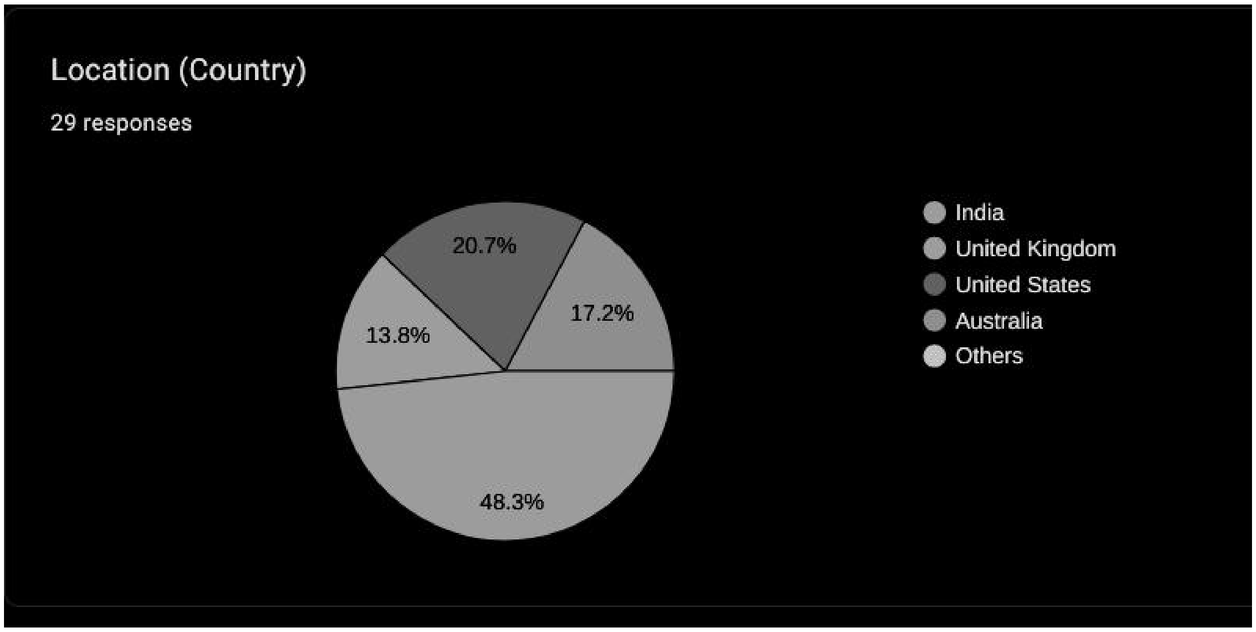
3.1 User Characteristics

- The GoCrypt framework will primarily target developers with varying levels of experience in the Go programming language, from beginners seeking to implement secure coding practices to seasoned professionals looking for advanced cryptographic functionalities. Users will typically work on web and mobile applications, requiring secure data transmission and storage solutions, while also including security professionals and researchers interested in robust encryption methods.
- Users will expect high performance and efficiency, with minimal overhead during cryptographic operations. Additionally, they will seek an intuitive API, comprehensive documentation, and compatibility with standard Go tools, ensuring a smooth integration process. This framework aims to empower users to implement strong security measures while facilitating ease of use and adherence to best practices in cryptography.

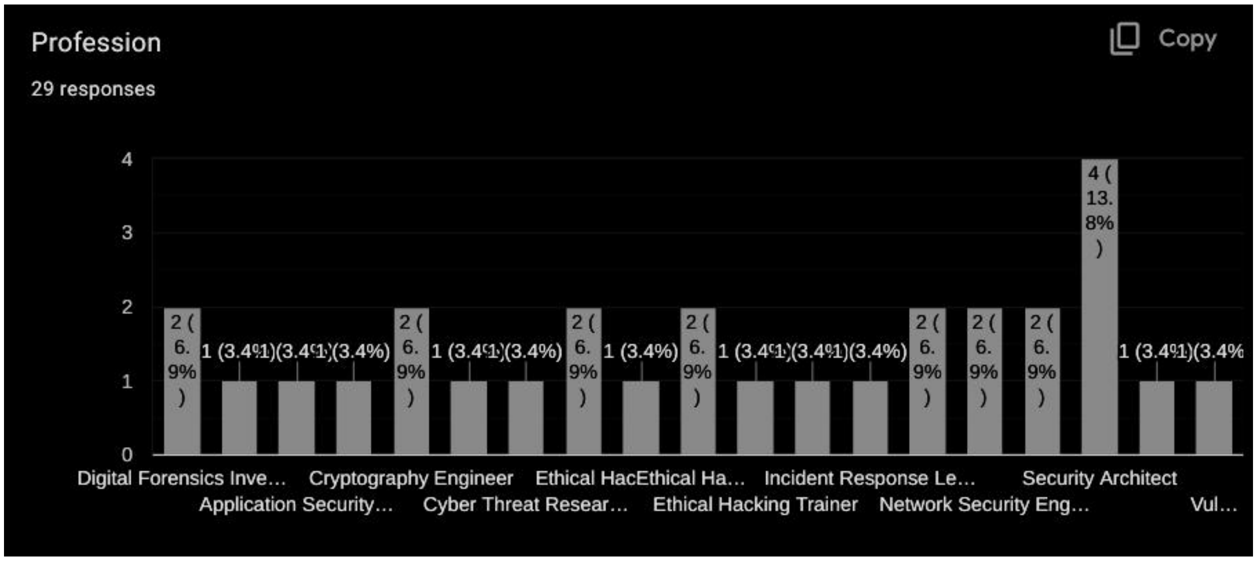
3.2 Survey

- To gather requirements and assess the current state of our cryptography framework project, we created a comprehensive Google Form for our survey. This form was designed to capture valuable feedback from stakeholders, including developers, security analysts, and end-users. We structured the survey with a mix of multiple-choice questions, rating scales, and open-ended prompts to ensure that respondents could express their needs and insights effectively. The survey focused on key areas such as desired features, existing challenges with current systems, and overall expectations from the new framework. By distributing the form via email and relevant communication channels, we aimed to reach a diverse audience, ensuring that we gathered a wide range of perspectives to inform our project's development. The collected data will play a crucial role in shaping our framework's design and functionality, ensuring that it aligns with user needs and industry best practices.

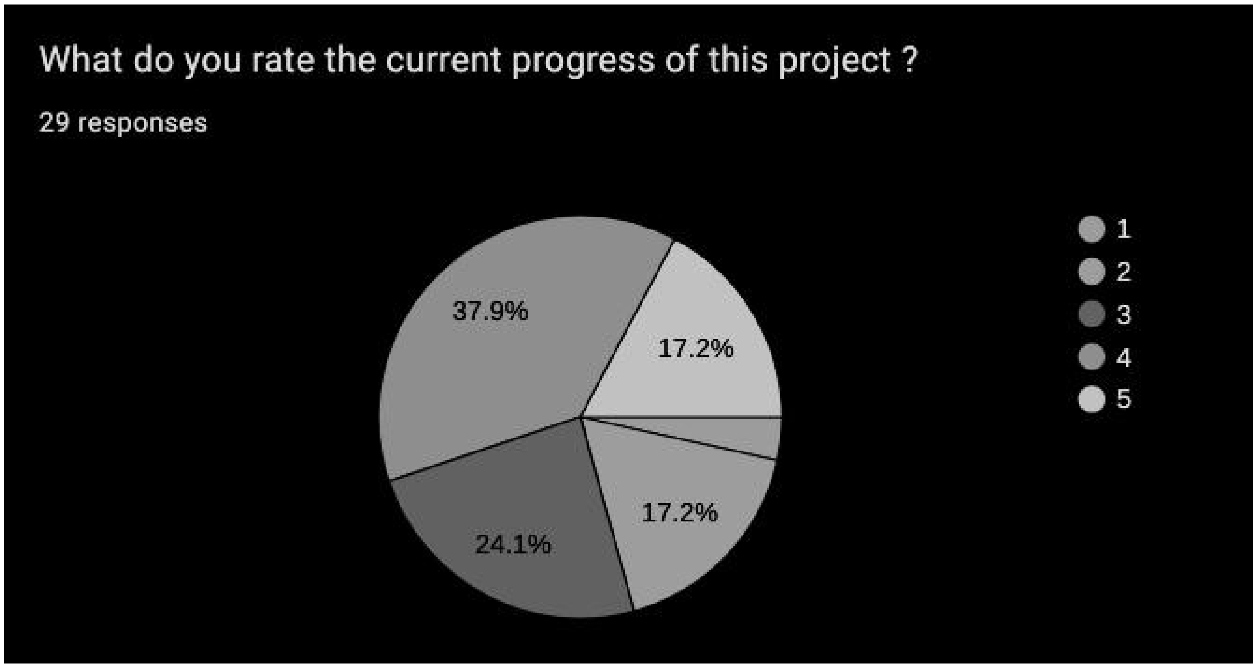
3.2.1 Location Survey



3.2.2 Profession Survey



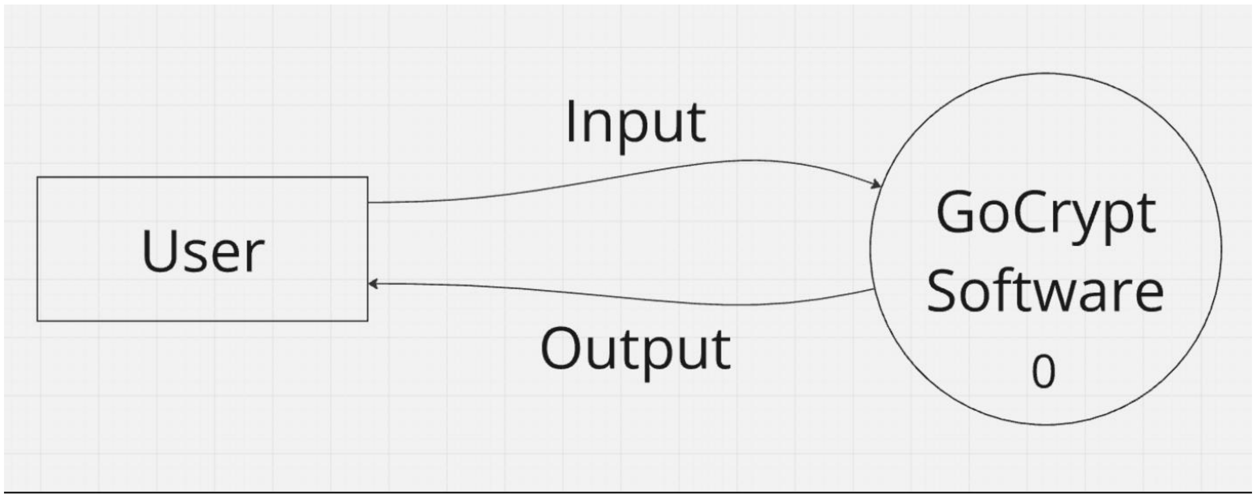
3.2.3 Review on the current state



4. Data Flow Diagrams (DFDs)

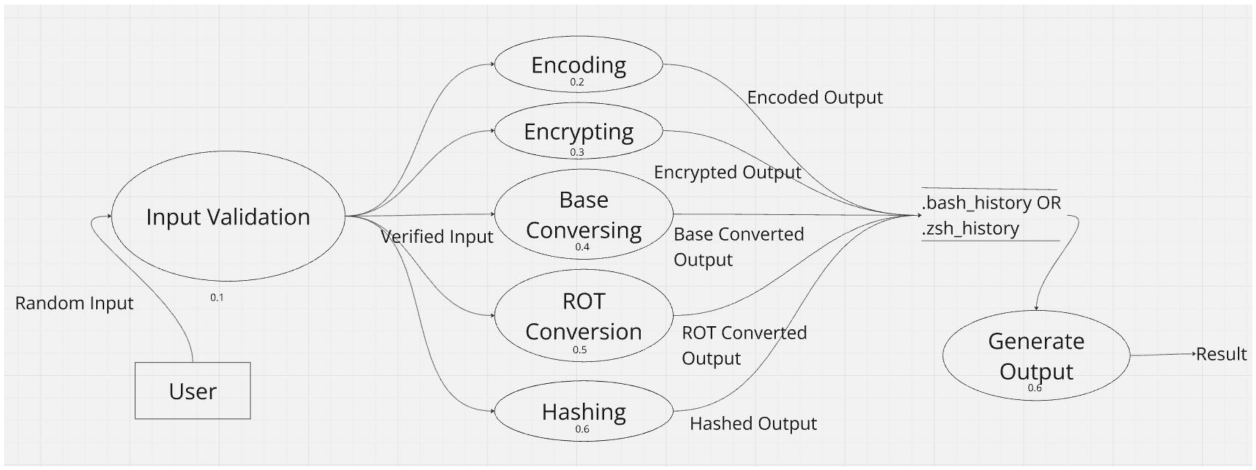
4.1 Level 0 DFD

- The Level 0 Data Flow Diagram provides a high-level overview of the cryptography tool. It highlights the interaction between the user and the system, focusing on the primary processes such as data input, cryptographic operations (encoding, encryption, hashing, etc.), and the delivery of results. This diagram illustrates the system as a single process with external entities and data stores interacting with it.



4.2 Level 1 DFD

- The Level 1 Data Flow Diagram delves deeper into the system, breaking down the primary process into specific sub-processes. It details the flow of data through various operations such as input validation, encoding methods, encryption algorithms, hashing functions, and password cracking mechanisms. This level showcases how individual components interact to transform user inputs into processed outputs, emphasizing the logical breakdown of tasks within the system.



5. Entity-Relationship (ER) Diagram

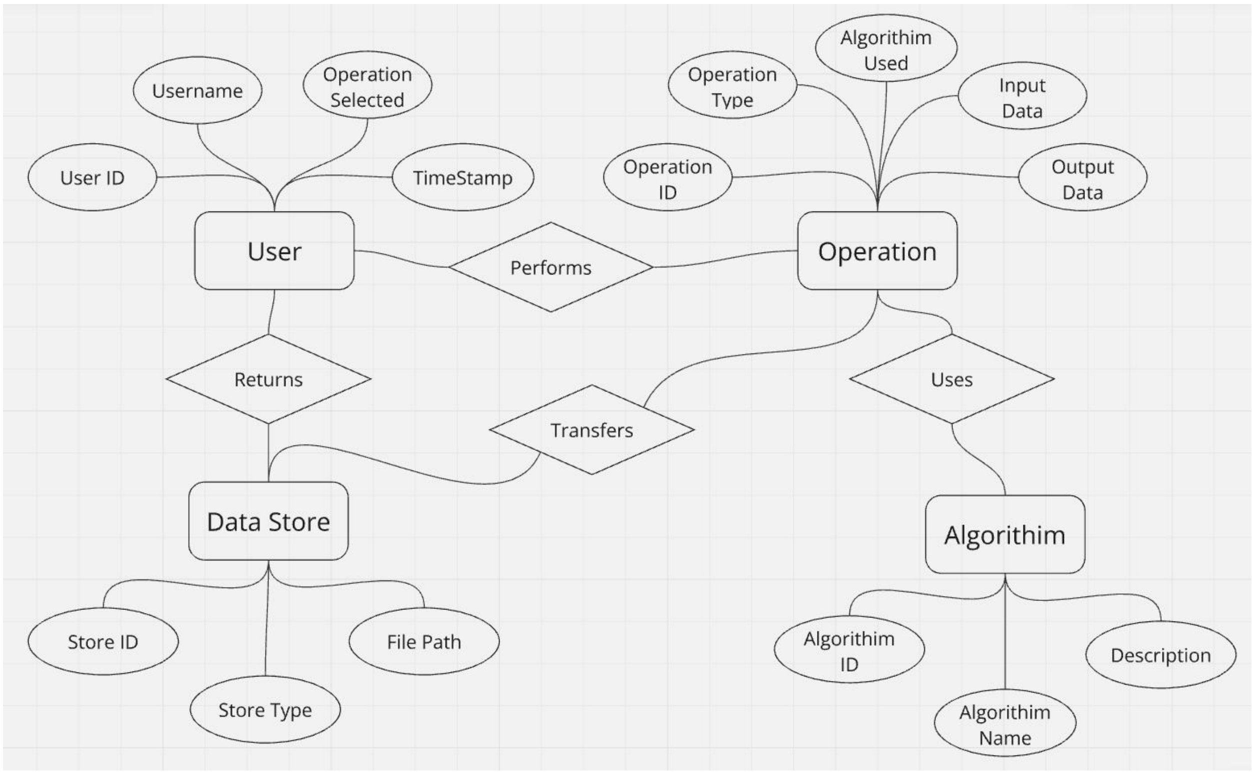
- The ER Diagram models the structure of the cryptography tool by identifying the core entities, their attributes, and the relationships between them. It represents how users interact with the system by performing operations that involve specific algorithms and, in some cases, password cracking. The diagram also incorporates data storage components, which are crucial for managing logs, wordlists, and hash databases.

5.1 Entities and Attributes

1. User : UserID (Primary Key), Username, OperationSelected, Timestamp.
2. Operation : OperationID (Primary Key), OperationType (e.g., Encode, Encrypt, Hash, etc.), AlgorithmUsed, InputData, OutputData.
3. Algorithm : AlgorithmID (Primary Key), AlgorithmName (e.g., AES, ROT13, Base64), Description.
4. DataStore : StoreID (Primary Key), StoreType (e.g., Logs, Wordlist, Hash Database), FilePath.

5.2 Relationships

1. Performs : User performs an operation by providing input and choosing an technique
2. Uses : Operation uses one of the algorithm to perform an operation
3. Transfers : Operation transfers the output data to data store
4. Returns : Data store transfers data to the user when the operation is complete



6. Functionality

- This tool is designed to handle a wide array of cryptographic tasks, enabling users to perform encoding, encryption, hashing, password cracking, and data conversions with ease. The functionalities have been thoughtfully implemented to strike a balance between utility and simplicity, ensuring that the tool is versatile yet intuitive to use.

6.1 Core Functionalities

1. **Encoding and Decoding**:
    - Transform data into formats like Base64 or hexadecimal for safe transmission or storage.
    - Decode the data back into its original form for validation or further use.
2. **Encryption and Decryption**:
    - Perform data encryption using algorithms like AES and RSA to secure sensitive information.
    - Decrypt the data when the correct key is provided, enabling authorized access.
3. **Hashing**:
    - Generate cryptographic hash values (e.g., MD5, SHA-256) to ensure data integrity.
    - Use salted hashes to enhance password storage security.
4. **Password Cracking**:
    - Test hashed passwords against dictionary wordlists to uncover weak or commonly used passwords.
    - Utilize external wordlists for advanced cracking scenarios.
5. **Base Conversion**:
    - Convert numbers or data between binary, decimal, hexadecimal, and octal numeral systems.
6. **ROT (Rotation) Conversion**:
    - Encode and decode text using ROT13, ROT47, or user-defined rotations for simple substitution ciphers.
7. **Integration**:
    - Designed to work seamlessly with other tools, allowing for flexible integration into larger cryptographic workflows.

6.2 Key Design Aspects

1. **Terminal-Based UI**:
    - A lightweight, command-line interface ensures the tool is highly portable, fast, and resource-efficient.
2. **Platform Independence**:

- Built using **Go**, the tool runs on any system with a terminal, including Windows, macOS, and Linux, with no additional dependencies.

3. **Extensibility**:
    - The modular design allows for easy addition of new functionalities or algorithms, ensuring the tool can adapt to evolving user needs.
4. **Community Support**:
    - As an open-source project, it benefits from community contributions for features, bug fixes, and improvements, allowing for continuous development and evolution.

7. Estimation

- The estimations for this project focus on Size, Time, and Cost, considering the tool's purpose as a lightweight, terminal-based cryptography utility. Since this is an open-source project, the primary resource is human effort, while monetary costs are negligible. The modular design and reliance on pre-built libraries ensure that the project remains practical and efficient to develop.

7.1 Size Estimation

- The estimated size of this project is approximately 5 KLOC (5000 lines of code).
- The project is a lightweight terminal-based tool, eliminating the need for extensive support structures or frameworks.
- Built with the Go programming language, it benefits from built-in cross-platform support, simplifying deployment and reducing development overhead.
- Size estimation is practical, focusing on efficient integration with existing libraries and utilities to streamline functionality.
- The tool prioritizes core cryptographic functionalities, including encoding, encryption, hashing, password cracking, and data conversion, ensuring a compact yet powerful design.

- The modular design of the tool ensures that it remains lightweight while maintaining compatibility with other cryptographic tools and frameworks.

7.2 Time Estimation

- The development effort and time were calculated using the Basic COCOMO model, which is appropriate for small to medium-sized projects like this one.

COCOMO Calculation

The COCOMO model is defined as:

$$\text{Effort (person-months)} = a \times (\text{KLOC})^b$$

$$\text{Time (months)} = c \times (\text{Effort})^d$$

For an organic project (simple and small-scale), the coefficients are:

$$a = 2.4,\ b = 1.05,\ c = 2.5,\ d = 0.38.$$

Given the size of 5 KLOC:

Effort

$$\text{Effort} = 2.4 \times (5)^{1.05} \approx 12.55\,\text{person-months.}$$

Time

$$\text{Time} = 2.5 \times (12.55)^{0.38} \approx 7.25\,\text{months.}$$

Personnel

$$\text{Personnel} = \frac{\text{Effort}}{\text{Time}} = \frac{12.55}{7.25} \approx 1.73 \approx 2\,\text{contributors (part-time).}$$

With an Effort $= 12.55$ person-months and a development timeline of 7.25 months, the project requires approximately 2 contributors working part-time.

- The calculated timeline reflects the simplicity of the project and its reliance on pre-built libraries and Go's robust standard library, which minimize development time. The project scope is manageable, requiring no extensive team or prolonged duration.

7.3 Cost Estimation

- Since this is an open-source project, monetary costs are virtually non-existent. The primary cost is human effort, which can be estimated based on the time and effort required to complete the project.

Human Effort Calculation

If monetized, the effort cost would be approximately:

$$12.55\,\text{person-months} \times 160\,\text{hours/month} \times 20\,\$/\text{hour} = 40,160\,\$.$$

- This project is open source, and contributions are voluntary, driven by community support through GitHub.
- There are no direct costs for development or runtime since it runs entirely offline and is compatible with any system with a terminal.
- The support and contributions from the community significantly reduce development burdens while making the tool freely accessible.