TO DESIGN AND DEVELOP A PROTOCOL FOR ANTI FORENSIC DISK ENCRYPTION TOOL EXAMINATION USING PRINCIPLES OF SOFTWARE REVERSE ENGINEERING

Thesis Submitted for the Award of the Degree of

DOCTOR OF PHILOSOPHY

In

Computer Science and Engineering

By

Zakariyya Hassan Abdullahi

Registration Number: 11816039

Supervised By
Dr. Shailendra Kumar Singh
Assistant Professor
Department of Computer Science and Engineering
Lovely Professional University, Punjab India

Co Supervised by
Dr. Moin Hasan
Assistant Professor
Department of Computer Science and Engineering
Jain Deemed-to-be University, Bengaluru India

LOVELY PROFESSIONAL UNIVERSITY, PUNJAB

2023

DECLARATION

I hereby declare that the presented work in the thesis entitled “ To Design and Develop a Protocol for Anti-Forensic Disc Encryption Tool Examination Using Principles of Software Reverse Engineering” in fulfilment of the degree of **Doctor of Philosophy (Ph.D. in Computer Science and Engineering)** is the outcome of research work carried out by me under the supervision of Dr. Shailendra Kumar Singh, working as an assistant professor in the Department of Computer Science and Engineering at Lovely Professional University Punjab, India. And under the co-supervision of Dr. Moin Hasan, working as an assistant professor in the Department of Computer Science and Engineering at Jain Deemed-to-Be University, Bengaluru, India, in keeping with the general practice of reporting scientific observations, due acknowledgements have been made whenever the work described here has been based on the findings of another investigator. This work has not been submitted in part or in full to any other university or institute for the award of any degree.

[illegible]

Signature of Scholar

Name of the scholar: Zakariyya Hassan Abdullahi

Registration No: 11816039

Department/school: Computer Science and Engineering

Lovely Professional University, Phagwara

Punjab, India

CERTIFICATE

This is to certify that the work reported in the Ph. D. thesis entitled " To Design and Develop A Protocol for Anti Forensic Disk Encryption Tool Examination Using Principles of Software Reverse Engineering" submitted in fulfillment of the requirement for the reward of degree of **Doctor of Philosophy (Ph.D.)** in the Computer Science and Engineering , is a research work carried out by Zakariyya Hassan Abdullahi (Registration No 11816039) , is bonafide record of his/her original work carried out under my supervision and that no part of thesis has been submitted for any other degree, diploma or equivalent course.

Shailendra

Signature of Supervisor

Name of supervisor: Dr. Shailendra Kumar Singh

Assistant Professor

Department of Computer Science and Engineering

Lovely Professional University, Punjab, India

MoinHasan.

Signature of Co-Supervisor

Name of Co-Supervisor: Dr. Moin Hasan

Assistant Professor

Department of Computer Science and Engineering

Jain Deemed-to-be University, Bengaluru, India

ACKNOWLEDGMENT

I am grateful to Almighty Allah for giving me life and wisdom to pursue my Ph.D. in Computer Science and Engineering at Lovely Professional University, India. I want to thank my supervisors, Dr. Shailendra Kumar Singh and Dr. Moin Hasan, for their guidance and support throughout my Ph.D. journey. I also extend my thanks to Hussaini Adamu Federal Polytechnic Kazaure, Nigeria, for the research opportunity provided through TETFund intervention. Special appreciation goes to Professor Khalid Hassan Abdullahi, Dr. Abubakar Magaji Abdullahi, and Sani Magaji Abdullahi for their help and Adviser. I am deeply thankful to my mother (Mariya Muhammad) for her encouragement, prayers, and support during my studies. To my devoted wife, Zainab Nura, and my children, Halima Zakariyya Hassan, Muhammad Zakariyya Hassan, and Abdullah Zakariyya Hassan, I am grateful for their patience and prayers. I want to acknowledge my friends and colleagues for their immense encouragement and contributions to my success, though their names are too many to mention individually. I will always remember and appreciate their help, Thank you all.

DEDICATION

This work is humbly dedicated to my late father, Hassan Abdullahi (Baffa), and my late brother, Shuaibu Hassan Abdullahi.

ABSTRACT

In the field of digital forensics, the battle against cybercriminals has led to the emergence of anti-forensic methods, particularly in the form of disk encryption tools aimed at obstructing forensic investigations. To effectively address these challenges, this study focuses on the creation of a comprehensive protocol for examining anti-forensic disk encryption tools using principles of software reverse engineering. The protocol is meticulously designed to delve into the inner workings of these encryption tools, uncovering concealed functionalities, vulnerabilities, and data obfuscation tactics utilized to evade detection. Through a systematic application of reverse engineering techniques, forensic investigators can acquire valuable insights into the functioning of these tools and detect potential countermeasures to combat their actions. Using data encryption software like VeraCrypt has become more common to protect data confidentiality. However, this creates a challenge for digital forensic investigators since encrypted data appears random and unreadable. VeraCrypt makes it even harder with features like concealed volumes and concealed operating systems, which not only encrypt data but also allow users to deny its existence. This makes it difficult for investigators to disprove such claims and detect encrypted content. By providing a standardized and structured approach to dissecting anti-forensic disk encryption tools, the proposed protocol offers a crucial resource to enhance the capabilities of digital forensic investigators in addressing the rapidly evolving landscape of anti-forensic technologies. This, in turn, reinforces the protection of digital evidence and upholds the integrity of legal proceedings, ensuring that cybercriminals are held accountable for their actions.

The investigation is essential in providing investigators with the tools they need to stay on top of the war against cybercrime and maintain the validity of digital evidence in the pursuit of justice as the digital landscape continues to change. Our world's digitization has accelerated the rise of cybercrimes and spurred the development of digital forensic tools. Anti-forensic technologies have arisen in reaction, impeding the effectiveness of digital evidence retrieval techniques. Reverse engineering software is thought to be a viable strategy for an extensive examination is required while examining anti-forensic tools. Reverse engineering software refers to the process of carefully examining a computer programme or software application in order to decipher its structure, functions, and behaviours. This study explores the assessment of anti-forensic technology through the use of software reverse engineering techniques. VeraCrypt, an open-source data encryption software, is increasingly used to secure data confidentiality.

However, detecting encrypted data can be challenging due to its random appearance and the presence of hidden volumes and operating systems. Usually, when a suspect's computer is turned on, forensic investigators attempt to gain access to it. but this thesis focuses on worst-case scenarios where access is limited. The thesis introduces a process model to identify encrypted data through raw byte data analysis, but the hidden volume and operating system remain elusive. The thesis explores leakage of information to establish VeraCrypt's use and strengthens the case against the suspect.

TABLE OF CNTENTS

List of Figures

List of Tables

List of Abbreviations

AF	Anti Forensic
AES	Advance Encryption Standard
APFS	Apple File System
CBC	Cipher Block Chaining
CAD	Computer-Aided Design
CLI	Command-Line Interface
CCD	Code Clone detection
CHS	Cylinder-Head-Sector
DF	Digital Forensic
DFS	Deniable File System
DES	Data Encryption Standard
3DES	Three Data Encryption Standard
ECD	Ever-Changing Disk
ECC	Elliptic Curve Cryptography
HDD	Hard Disk Drive
IDA	Pro Interactive Disassembler
LAN	Local Area Network
MLC	Multi- Level Cell
MLSCA	Multilingual Source Code Analysis
MDRE	Model Driven Reverse Engineering
OOD	Object-Oriented Design
PGP	Pretty Good Privacy
PBA	Pre-Boot Authentication
PRE	Protocol -Reverse Engineering
RE	Reverse Engineering
RSA	Rivest Shamir Adleman

SRE	**Software Reverse Engineering**
SSD	**Solid State Drives**
SOM	**Self-Organizing Map**
SHA	**Secure Hash Algorithm**
XTS	**Sased Tweaked-Codebook**

CHAPTER I
INTRODUCTION

In the digital age, the protection of sensitive data is paramount. Encryption tools, such as disk encryption software, play a crucial role in safeguarding information from unauthorized access. However, while encryption is a vital security measure, it can also be misused for illicit purposes. To ensure the balance between privacy and security, it is essential to have effective mechanisms for examining and scrutinizing disk encryption tools, particularly in forensic investigations.

Digital information technology has significantly impacted society, enabling new opportunities and enhancing operations. The globalization of the Internet and digital mobile networks has led to the growth of digital systems like GSM, which has over a billion users today. These systems offer inherent benefits such as noise insensitivity, long-distance transmission, quality copying, and data removal. However, they are also susceptible to misuses, such as tampering with mail systems. To safeguard digital systems without compromising their advantages, it is necessary to change information to defend itself regardless of transmission or storage. Cryptology, a field that investigates this issue, provides a valuable summary of this topic [1].

1.1 Introduction

In today's highly interconnected world, the security and privacy of personal and sensitive data have become topics of great public concern. As data is often placed in the hands of online third parties, the final safeguard for data confidentiality is found in local storage, where physical disconnection minimizes the risk of unauthorized access. Nonetheless, even in local storage scenarios, precautionary measures must be taken to safeguard against potential threats. One prevalent threat scenario involves a thief pilfering a user's personal hard drive and gaining access to its contents. This particular threat model has been extensively researched, resulting in the development of numerous robust disk encryption solutions over time [2],[3].

With developing a methodology for investigating disc encryption techniques from an Anti-forensics' perspective, this invention intends to meet the novel challenges in the field of digital forensics. Anti-forensics is the term used to describe the intentional methods used to thwart or dodge digital investigations, and it frequently entails the use of encryption tools to hide damaging evidence.

In this context, our research focuses on the examination of disk encryption tools, a critical aspect of digital forensics, by leveraging the principles of software reverse engineering. The process of disassembling and examining software is known as "software reverse engineering." to understand its inner workings, functionality, and potential vulnerabilities. By applying these principles, we aim to develop a protocol that will allow forensic analysts to gain insights into the operation of encryption tools designed with anti-forensic objectives. These techniques, such as data encryption, file hiding, and data obfuscation, make it difficult for traditional forensic methods to uncover crucial evidence. To address this issue, this thesis proposes a technique that employs software reverse engineering to identify and counter the use of anti-forensic measures.

The use of disc encryption tools has gradually become widespread in recent years, as they offer a secure method for protecting sensitive data. However, this has also led to the development of anti-forensic disc encryption tools that resist forensic analysis and make it difficult for forensic analysts to recover the encryption key and gain access to the encrypted data. These anti-forensic tools use various techniques, such as code obfuscation, dynamic code generation, anti-debugging techniques, and rootkit techniques, to hide their presence and resist analysis.

To counter the use of anti-forensic disc encryption tools, it is essential to develop a protocol for examining them using the principles of software reverse engineering. The goal of this protocol is to enable forensic analysts to identify any anti-forensic techniques used in the encryption tool and provide recommendations for improving its security. The purpose of examining an encrypted disc should be to safeguard the security of the information stored on it, which is a legitimate and ethical use case. This includes verifying that the encryption tool is using strong encryption algorithms, the key generation process is secure, and the user authentication process is robust. The examination should not be used to evade forensic investigations or conceal illegal activities. Any attempt to do so would be unethical and potentially illegal. It's important to conduct the examination with integrity and to respect the privacy and safety of the information stored on the encrypted disk. Throughout history, there have been instances where people needed to conceal information. Singh's research talks about different times when people used to write in secret, as in the battles of 480 B.C. between Greece and Persia and when Julius Caesar and Mary Queen of Scots used secret codes. They did this to make sure only the right people could understand the message. If someone who wasn't supposed to see the message got it, there could be serious problems. For example, Mary Queen of Scots was accused of treason when her decoded messages were used as proof against her [4].

1.2 Digital Forensic and Information Security

Challenges in keeping information safe include making sure data is accurate and private in systems. Sometimes, restricting user access can make systems slower, but using real-time protection and encryption are better options. "Fundamentals of Digital Forensics" is a helpful guide to computer investigations, explaining how they work, what evidence is involved, and the limits. The book, made with the Swedish Police and Jan-ke Peterson, is for students and professionals who want to learn about this field [5].

Digital forensics, sometimes referred to as computer forensics, is a discipline within forensic science that centers on the discovery, safeguarding, scrutiny, and presentation of electronic information with potential utility as legal evidence. This domain encompasses the gathering and assessment of data derived from diverse electronic devices, including but not limited to computers, laptops, mobile phones, tablets, and storage media. The primary goal of digital forensic inquiries is to recognize and record evidence related to cybercrimes, such as hacking, fraud, identity theft, and intellectual property theft. The different processes stages of digital forensic investigation are shown in figure 1.1.

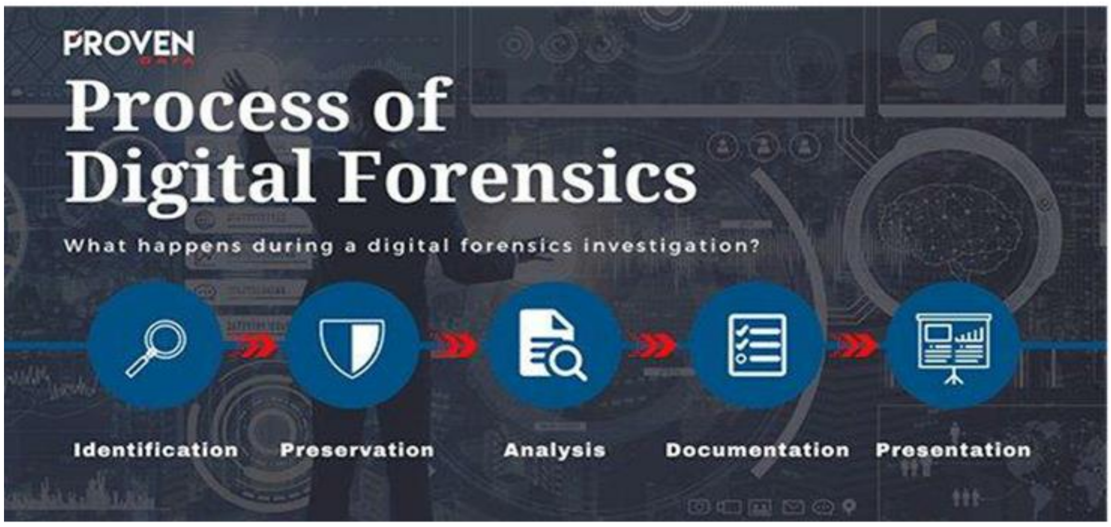

PROVEN DATA
Process of Digital Forensics
What happens during a digital forensics investigation?
Identification
Preservation
Analysis
Documentation
Presentation

Figure 1.1 Different Stages of Digital Forensic Investigation Process

Digital forensic investigations require specialized training and expertise in computer science, information technology, and criminal justice. Digital forensic investigators must also adhere to strict legal and ethical standards to confirm that the evidence they collect is acceptable in court and does non-violate the rights of individuals

Table 1.1 Different Types and Aspects of Digital Forensic

Aspect of Digital Forensic	**Definition and Purpose**
Computer Forensics	Computer forensics analyzes digital evidence for criminal cases.
Mobile Device Forensics	Mobile device forensics analyzes digital evidence on smartphones, tablets, and wearables for criminal cases.
Network Forensics	Digital forensics analyzes digital evidence on networks to monitor traffic, identify security breaches, and provide evidence in criminal cases.
Cloud Forensics	Cloud forensics examines digital evidence in cloud computing for criminal cases.
Malware Analysis	Malware analysis identifies malware behavior, source, and develops prevention strategies.
Incident Response	Analysis of security incidents to identify scope, impact, contain damage, and prevent future threats.
Digital Forensic Analysis Tools	Digital forensic analysis tools enhance accuracy, efficiency, and consistency.
Legal and Ethical Considerations	Legal and ethical considerations for digital evidence gathering, safeguarding, and examination.

Table 1.1 outlines digital forensics' key aspects, including computer, mobile, network, cloud, malware analysis, incident response, tools, and legal and ethical considerations. These areas aid in criminal investigations, cybersecurity, and detecting security breaches.

1.3 Encryption

Encryption is a process in which plain-texts are converted into cipher-texts using an encryption algorithm and a key to ensure the secure transmission or storage of encrypted data. This procedure entails the utilization of an algorithm along with a key. The algorithm determines how the encryption is performed, while the key is used for encoding the message, which ultimately determines the encryption method used. Decryption is the opposite of encryption, where the cipher text is transformed back into plain text. It is a process that converts unreadable cipher text into the original message. The decryption process relies on two elements: a decryption algorithm and a key. The decryption algorithm specifies the technique used for decryption, which is typically the same as the encryption algorithm.

A key in cryptography can be a combination of alphabetic or binary characters, or even special symbols. The password is applied to the plain text during encryption and to the cipher text during decryption. The selection of the key is crucial in cryptography as it greatly impacts the security of the encryption algorithm. For instance, if Alice uses a key of 3 to encrypt the plain text "Pr," the same key will be used for decryption to retrieve the original message.

Plain Text: The original message you want to communicate with someone is referred to as plain text. In encryption, the actual message being sent is called plain text. For example, if Alice wants to send a message to Robert saying "Hello Friend, how are you?" the plain text version of the message would be "Hello Friend, how are you?"

Cipher Text: The message that is transformed into an unreadable or meaningless format, which cannot be understood by anyone else, is called cipher text. In cryptography, before transmitting the actual message, the original message is converted into an incomprehensible format. For instance, a cipher text like "Ajd672#@91ukl8*^5 percent" is generated as a cipher text for the message "Hello Friend, how are you?"

There are two categories of encryption techniques: symmetric and asymmetric encryption. Symmetric encryption employs an identical key for both the encryption and decryption processes, such as AES, DES, and 3DES.Sari et al., 2019 [6] scrutinizes the application of AES in encrypting both blocks and multimedia data, demonstrating that all categories of attacks have the potential to compromise AES. In response, a novel approach called T-DES encryption has been proposed to enhance data transmission speed, yielding PSNR and SSIM values surpassing 40 dB. Encryption is a mathematical process that converts plaintext into ciphertext using a key. It is reversible, meaning that when the correct key is used to decrypt the ciphertext, it returns to the original plaintext. The plaintext that comes out of the decryption process must be an exact copy of the original, protecting both confidentiality and integrity. Modifications to the key or ciphertext do not give away any part of the real plaintext [7]. Different encryption algorithms and processes exist, each designed to protect data confidentiality, integrity, and availability. However, effective key management is crucial to ensure that encryption remains secure and reliable.

A public key and a private key are used in asymmetric encryption, commonly referred to as public-key encryption. Although the private key is used to decipher ciphertext, the public key is used to encrypt plaintext. While the private key needs to be kept a secret, the public key can be shared with anybody. Asymmetric encryption schemes, such as RSA, ECC (Elliptic Curve Cryptography), and Diffie-Hellman, are examples. These algorithms are applicable to both Local

Area Networks (LANs) and Wide Area Networks (WANs). The potential use of forensic file identification techniques in identifying encryption software has not been thoroughly explored. Despite a six-month, resource-intensive investigation, no straightforward method for expediting the decryption of passwords or encryption keys was discovered, it did yield a number of significant findings that will help identify when an encryption application is being used, among other characteristics of the encryption application [8].

Encryption can be applied to various types of data, including files, messages, and entire disks. Disk encryption encompasses the encryption of the complete disk, including the operating system and all user data, to protect against unauthorized access if the device is lost or stolen. Examples of disk encryption tools include BitLocker (Windows), File Vault (Mac), and VeraCrypt (cross-platform) [9]

Table 1.2 Description of Different Types of Encryption Techniques

Encryption Type	Description	Examples
Symmetric Key Encryption	Uses the same key for both encoding and decoding	DES, 3DES, AES, Blowfish, RC4
Asymmetric Key Encryption	Makes use of two distinct keys for encryption and decryption	RSA, DSA, ECC
Hash Functions	One-way encryption that produces a unique fixed-length hash value for any input	SHA-256, MD5
Stream Ciphers	Encrypts data bit by bit, typically in real-time	RC4, Salsa20
Block Ciphers	Divides data into fixed-size blocks and encrypts each block separately	AES, Blowfish
Hybrid Encryption	Uses a grouping of symmetric and asymmetric key encryption	PGP, SSL/TLS

Table 1.2. outlines various encryption types, including symmetric, asymmetric, hash functions, stream ciphers, block ciphers, and hybrid encryption. Symmetric encryption uses the same key for both encryption and decryption, while asymmetric encryption uses two distinct keys. Hash functions create fixed-length hashes, stream ciphers encrypt data bit by bit but can be susceptible to attacks, and hybrid encryption combines both. The authors delve into the factors that determine the safety of encryption, which include the strength of the algorithm, the secrecy of the encryption key, and the protection of its implementation. Although encryption cannot provide absolute security, its proper usage can effectively safeguard sensitive data. Security attacks tend to target electronic devices such as laptops, personal digital assistants, flash drives, and external hard drives, exploiting their vulnerabilities [10].

People spend a lot of money to keep their data safe because losing important data could cause irreversible damage and affect their competitiveness [1]. Keeping of data secure is the most important form of security, even more crucial than network security. This is because only securely encrypted data can be safely sent. Lost or stolen laptops can expose sensitive information to unauthorized people who will find them. Personal computers can also be compromised through different threats. Laptops are particularly at risk because they are often used in public places without protection. Researchers didn't find easy solutions for these vulnerabilities. Simple software changes made by developers are usually not effective. High-capacity laptops are found by unauthorized accessed by individuals when they're lost or stolen [10]. Therefore, storing data securely is very important, and we need to protect it from unauthorized access. Different technologies like access control and auditing help improve data storage security. However, in these methods, the actual data isn't encrypted. This means that if someone can access the storage device, they can read the data inside [11].

An encryption algorithm's main function is to keep messages sent through an insecure channel in encrypted form. Two mathematical operations make up a standard encryption algorithm: the encryption function E (Encryption) and the decryption operation, denoted as D (Decryption) = E1, involves the sender, often referred to as Alice, encrypting the original message P (plaintext) using the encryption function. The resulting ciphertext, C = E(P), is then transmitted over the insecure channel. Upon receiving C, the intended recipient, Bob, can recover the plaintext by computing D(C) = P.

For this method to successfully secure Alice's message for exclusive reading by Bob, several conditions must be met. Firstly, Bob should be the sole individual possessing knowledge of the decryption function D, with Alice potentially being an exception. Secondly, the construction of the transformation E must be such that a potential ciphertext interceptor, often termed Eve, cannot practically extract any information about the plaintext, except perhaps its length. Lastly, the processing power required for the transformations E and D should not be excessively high as shown in Figure 1.2.

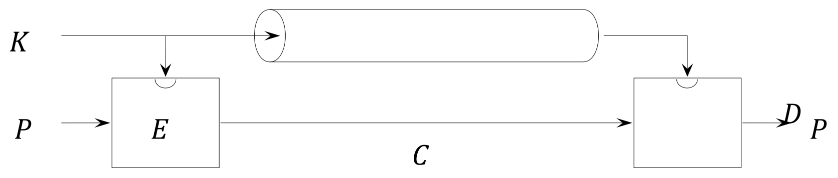


Figure 1.2 Model for Symmetric Encryption

To retrieve D, apply this particular transformation. Diffie and Hellman recognized that the confidentiality of the encryption function was not essential if one could generate what is known as trapdoor one-way functions. These are functions that are simple to evaluate, but they cannot be effectively reversed without additional information (the trapdoor). Soon after, examples of trapdoor one-way functions were discovered, enabling the creation of useful public key encryption techniques like RSA [12].

Jadhav et al., 2017 [13] has discussed how the Data Encryption Standard (DES), a government-approved method to protect sensitive data, is being replaced by the more advanced encryption standard (AES) for security. AES is widely used because it's better at both security and speed. While there are software options, they might not be fast enough for tasks like routers and wireless systems. Cryptography, which involves encryption and decryption, is a key way to protect data from hackers. Encryption makes data hard to understand for outsiders, turning original data into unreadable cypher text.

The article describes that cryptography changes a message into a hard-to-read form before sending it through an insecure channel. People without permission would struggle to understand the message because of the encryption. But the person who's allowed to read it can change it back to the original form [14]. The authors talk about how important and commonly used cryptography is to protect data from hackers. It's all about making data safe by changing it into unreadable text and then changing it back. Encryption turns data into something called Cipher text that's hard for attackers to understand [15].

1.4 Anti-Forensic

In the current digital world, almost everything relies on digital data. This data is found in various fields, showing how technology is part of them. Digital forensics grew due to situations like cybercrimes and terrorism, where digital evidence holds important information. People, companies, and governments all use digital data in many parts of life.[16],[17]. Anti-forensics techniques like APFS can make digital investigations harder. This is because it uses a different way to store data than older systems like HFS+. This makes it tough for investigators to get to and study encrypted data. Using cloud services more also creates issues. Investigators might not be able to reach physical devices or get data from providers due to legal or technical reasons. Not understanding how things work in reverse can lead to losing evidence, as many apps and systems are made to stop forensic analysis.

1.4.1 File Deletion Technique

This method entails the removal of files from a computer's file system, which can be achieved through conventional delete commands or specialized tools designed to overwrite the deleted data. The objective behind file deletion is to create challenges or render it nearly impossible for investigators to recover the data that has been deleted.

1.4.2 Disk Wiping

A technique used to overwrite data on a hard disk, making it difficult or impossible to recover the original data. Disk wiping tools use various algorithms to overwrite the data multiple times to ensure that it cannot be recovered.

1.4.3 Encryption

Encryption is a technique used to convert data into an unreadable form using a secret key. If the key is not known, the data cannot be read. Attackers may use encryption to hide sensitive data, making it difficult for investigators to find or access it.

1.4.4 Steganography

A method for concealing data within other files, including images or audio files. Investigators will have a difficult time locating the secret data because it is difficult to see or detect.

1.4.5 Obfuscation

A technique for concealing code or data by making it challenging to read or comprehend. Obfuscation is a technique attacker use to conceal harmful code or data, making it challenging for researchers to decipher or reverse-engineer it.

1.4.6 Antivirus Evasion

Antivirus evasion tactics are utilized to circumvent protection provided by antivirus programs. In order to evade detection by antivirus software, attackers may utilize techniques such as polymorphism, encryption, or obfuscation.

Hussein et al., 2020 [18] discusses the presence of malicious software on a computer, highlighting the challenges investigators may face in locating or comprehending files, programs, or connections in such instances. The use of rootkits can obscure information, and attackers also employ network tunneling to conceal data within alternative methods of transmission. This strategy aids them in circumventing firewalls and evading detection by network monitoring programs. Hassan , 2019 [19] notes that individuals with malicious intent employ tactics to evade detection or scrutiny.

Digital forensics, a field dedicated to examining digital evidence for legal purposes, addresses this issue. Its primary objective is the comprehensive investigation of evidence to present it in a court setting. Professionals in digital forensics utilize tools to uncover, preserve, and analyze evidence, subsequently drawing conclusions and communicating their findings to others.

1.5 Reverse Engineering

Fontana, 2022 [20] discuss the concept of reverse engineering, likening it to the process of disassembling something to gain insights into its functioning. This practice is applicable to various domains such as software, hardware, military technology, and biological processes. The central objective is to analyze machine code and reverse it to reconstruct the original source code. Individuals engage in this activity to comprehend the underlying mechanisms, revive outdated technologies, assess security, and gain a competitive edge over rivals.

The authors, discusses the utilization of reverse engineering as a method to comprehend the fundamental concepts and principles of software. This means looking at what goes into the software and what comes out. People also test it to find problems or weak points in security. Reverse engineering means studying a system to know its parts and how they connect. Remember, the point isn't to copy the program, but to study its code, how it works, and what it does[21] [22]. Holger and Hausi [23] assert that reverse engineering involves extracting information from the source code of software, facilitating a clearer understanding and opportunities for enhancement. This process entails acquiring information, analyzing it, and rendering it visible. Methods for accomplishing this task include identifying fundamental components, examining their interconnections, and pinpointing valuable elements within the software.

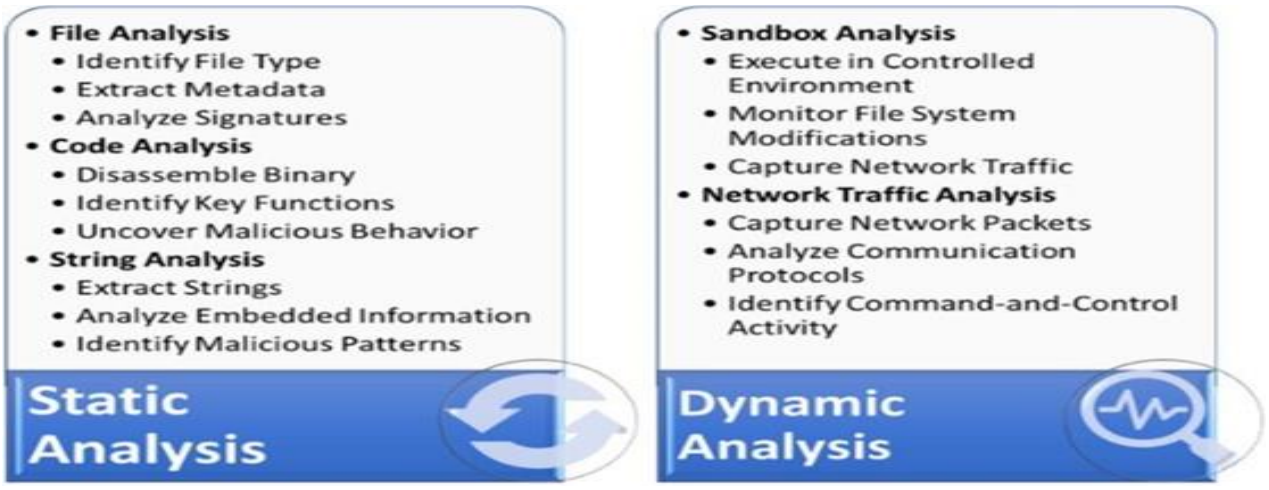
Table 1.3. outlines software reverse engineering techniques, including disassembly, debugging, DE compilation, dynamic analysis, static analysis, binary analysis, and fuzzing. These techniques help analyze software systems for security assessment and enhancement, identifying vulnerabilities and defects, and enhancing software performance.

Table 1.3 Various Techniques Used in the Field of Software Analysis

Technique	Description
Disassembly	Conversion of compiled code to human-readable assembly code.
Debugging	Identification and fixing of errors or defects in software systems.
DE Compilation	Conversion of compiled code to high-level programming language.
Dynamic Analysis	Analysis of software systems while they are running.
Static Analysis	Analysis of software systems without executing them.

Binary Analysis	Analysis of compiled code at the binary level.
Fuzzing	Sending large amounts of data to a software system to identify vulnerabilities or defects.

The practice of examining a software system to extract design and implementation details without having access to the source code is known as software reverse engineering, or SRE. This is frequently employed for debugging, malware analysis, security audits, and other purposes [24] . Here's a detailed look into various reverse engineering techniques: Show in Figure 1.3



• File Analysis
• Identify File Type
• Extract Metadata
• Analyze Signatures
• Code Analysis
• Disassemble Binary
• Identify Key Functions
• Uncover Malicious Behavior
• String Analysis
• Extract Strings
• Analyze Embedded Information
• Identify Malicious Patterns
Static Analysis
• Sandbox Analysis
• Execute in Controlled Environment
• Monitor File System Modifications
• Capture Network Traffic
• Network Traffic Analysis
• Capture Network Packets
• Analyze Communication Protocols
• Identify Command-and-Control Activity
Dynamic Analysis

Figure 1.3 Static and Dynamic Analysis

1.5.1 Static Analysis

Static analysis involves examining the software without actually executing the program. It provides a way to inspect the code for patterns, structures, and possible malicious signatures.

Tools:

- **IDA Pro**: A disassembler and debugger that can handle a variety of formats across different architectures and operating systems.
- **Ghidra**: An open-source software reverse engineering suite developed by the NSA, which includes a disassembler among other tools.
- **Hex-Rays**: A decompiler that converts executable programs into a higher level, readable code closer to the original source code.

Techniques:

- **Disassembly**: Converting binary code to assembly code which is more readable and understandable to humans.
- **Decompilation**: Attempting to revert assembly code back into a higher-level language code (like C or C++).
- **Pattern Recognition**: Identifying common constructs like loops, if-else conditions, switch statements, etc.

1.5.2 Dynamic Analysis

Dynamic analysis involves running the program and observing its behavior during execution, which is useful for understanding real-time operation and interaction with the environment.

Tools:

- **OllyDbg**: A dynamic debugger particularly used for Windows binary applications.
- **x64dbg**: An open-source x64/x32 debugger for windows.
- **GDB**: The GNU Project Debugger for various programming languages, including C, C++, and Fortran.

Techniques:

- **Breakpoints**: Used to pause the program at certain points to examine the state of the application.
- **Step Execution**: Allows step-by-step execution to observe changes and behaviors in specific code segments.
- **Memory Dumping**: Extracting data from the software's memory space during runtime to analyze current states and data structures.

1.5.3 Symbolic Execution

Symbolic execution is an advanced method where the inputs to the software are treated as symbolic variables instead of concrete values. This technique is useful for discovering what parts of code can lead to bugs or vulnerabilities.

Tools:

- **KLEE**: A symbolic virtual machine built on top of the LLVM compiler infrastructure.
- **angr**: A python framework for analyzing binaries, capable of symbolic execution, taint analysis, and more.

Techniques:

- **Constraint Solving**: Determining the conditions under which certain parts of the code are executed.
- **Path Exploration**: Exploring various paths through the code based on different input conditions.

1.5.4 Fuzzing

Fuzzing is a technique where random data is inputted into the software to induce errors and uncover vulnerabilities.

Tools:

- **AFL (American Fuzzy Lop)**: A security-oriented fuzzer that employs a variety of mutation techniques.

- **Boofuzz**: A network protocol fuzzing tool.

Techniques:

- **Mutation-Based Fuzzing**: Modifying existing data inputs in minor ways to explore new execution paths.
- **Generation-Based Fuzzing**: Creating inputs from scratch based on models of input data structures.

1.5.5 Decompiling to High-Level Languages

While earlier mentioned as part of static analysis, decompiling to more abstract languages like C# or Java (in the case of .NET and JVM applications, respectively) uses specialized tools:

Tools:

- **dotPeek**: A .NET decompiler.
- **JD-GUI**: A standalone graphical utility that displays Java sources from CLASS files.

Reverse engineering can be a complex and technically demanding field, requiring knowledge of low-level programming, operating systems, computer architectures, and often, cybersecurity principles. These techniques and tools provide a comprehensive foundation for delving into software reverse engineering, but mastering them requires significant practice and continued learning [25]. The process of reverse engineering generally involves a series of critical steps as depicted in the Figure 1.4.

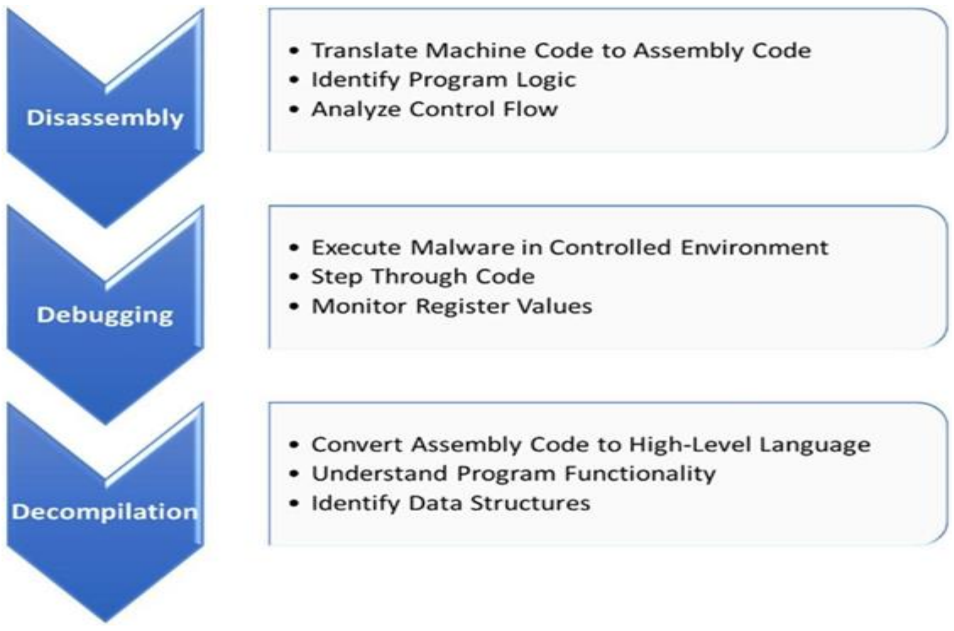

Disassembly
• Translate Machine Code to Assembly Code
• Identify Program Logic
• Analyze Control Flow
Debugging
• Execute Malware in Controlled Environment
• Step Through Code
• Monitor Register Values
Decompilation
• Convert Assembly Code to High-Level Language
• Understand Program Functionality
• Identify Data Structures

Figure 1.4 Reverse Engineering Steps

The table 1.4,1.5,1.6 provides a comprehensive overview of the technical aspects of disassembly in reverse engineering, highlighting the systematic approach required to understand and potentially improve the system being analyzed.

Table 1.4 Hardware Disassembly

Step	**Details**	**Tools/Techniques**
Preparation and Tools	Gather necessary disassembly and diagnostic tools.	Screwdrivers, pliers, tweezers, soldering iron, multimeter, oscilloscope
Static Analysis	Visual inspection and circuit analysis.	Service manuals, schematics, datasheets
De-soldering and Removal	Carefully remove components without causing damage.	Soldering iron, de-soldering tools

Circuit Reverse Engineering	Reconstruct the circuit diagram and test component functionalities.	Oscilloscope, multimeter
Integration Understanding	Analyze how different parts interact and signal processing.	Signal analyzer, logic analyzer

Table 1.5 Software Disassembly

Step	**Details**	**Tools/Techniques**
Preparation and Tools	Setup disassemblers, debuggers, and anti-tamper detection.	Disassemblers (IDA Pro, Ghidra), debuggers (OllyDbg, x64dbg)
Static Analysis	Examine code to identify key patterns and structures.	Disassemblers, hex editors
Dynamic Analysis	Monitor program execution and inspect memory operations.	Debuggers, memory inspection tools
Decompilation	Convert machine code back to a higher-level language to understand algorithm implementations.	Decompilers, static analysis tools
Patching and Modification	Modify code as needed and verify that changes do not introduce new issues.	Patching tools, testing frameworks

Table 1.6 Ethical and Legal Consideration

Consideration	**Details**
Legal and Ethical Usage	Reverse engineering should be done within legal frameworks and respect copyright and licensing laws.
Security and Privacy	Ensure that reverse engineering practices do not compromise data security or violate privacy rights.

Debugging in the context of software reverse engineering involves identifying, analyzing, and fixing bugs or defects in software after examining its code, especially when the source code is not available. This process can be quite complex and involves a variety of tools and techniques to deconstruct and trace software behaviors [26]. Below is a table 1.7 outlining the key technical concepts associated with debugging during software reverse engineering:

Table 1.7 Debugging

Concept	**Details**	**Tools/Techniques**
Breakpoints	Used to pause program execution at a specific point, allowing the examination of the context.	Debuggers (e.g., GDB, x64dbg)
Step Execution	Execute code one instruction or line at a time to observe changes in program state.	Debuggers (e.g., IDA Pro's debugging feature)
Register Inspection	Examine the contents of CPU registers to track how data is processed and manipulated.	Debuggers, CPU monitoring tools
Memory Dumps	Capture and analyze regions of memory used by the software to understand data handling.	Memory viewers (e.g., HxD, WinDbg)
Variable Watching	Track changes to specific variables to see how their values change throughout execution.	Integrated in most debuggers
Call Stack Analysis	Analyze call stacks to trace function calls and returns, providing insights into execution flow.	Debuggers, Stack tracing tools
Exception Handling	Monitor how the software handles runtime errors and exceptions.	Debuggers, Exception monitoring
Conditional Debugging	Set conditions for breakpoints to pause execution only when certain criteria are met.	Advanced debuggers
Log Analysis	Review logs generated by the application or system to identify abnormal or unexpected events.	Log viewers, Custom scripts for log parsing

Reverse Execution	Execute instructions in reverse to understand what led to a current state or error.	Reverse debuggers (e.g., RR - Record and Replay framework)
Symbolic Information	Use symbols to get meaningful names and context for memory addresses and code regions.	Debuggers with symbol server integration

Additional Concepts and Tools

- **Disassemblers and Decompilers:** Tools like IDA Pro, Ghidra, and Radare2 are crucial for converting executable code into a human-readable format (assembly code or higher-level languages), aiding in understanding what the program does without running it.
- **Dynamic Analysis Tools:** Used to analyze the behavior of a program while it is running, often through instrumentation.
- **Static Analysis Tools:** Analyze the code without executing it, often used to spot potential errors or vulnerable patterns in the code.

These concepts and tools collectively enable software reverse engineers to dissect a program's operation in great detail, effectively allowing them to debug, understand, and often repurpose software for which they don't have the original source code.

Dynamic analysis in software reverse engineering is the practice of analyzing a program by executing it in real-time, observing its behavior and its interactions with various systems. This method is particularly useful for understanding complex software behavior, detecting hidden functionality, or identifying potential vulnerabilities that only manifest during execution. Below is a table 1.8 outlining the key technical concepts associated with dynamic analysis in software reverse engineering:

Table 1.8 Dynamic Analysis

Concept	**Details**	**Tools/Techniques**
Runtime Tracing	Monitors and logs system calls, function calls, and other runtime events to analyze program behavior.	Debuggers, Tracers (e.g., Strace, DTrace)

Memory Profiling	Analyzes how a program uses memory during execution, identifying memory leaks and other issues.	Profilers (e.g., Valgrind, Memory Profiler)
Network Monitoring	Captures network traffic generated by the program to analyze its communication patterns and protocols.	Network analyzers (e.g., Wireshark, tcpdump)
Input Fuzzing	Inputs random, malformed, or unexpected data into the program to test its robustness and error handling.	Fuzzers (e.g., AFL, Burp Suite)
API Hooking	Intercepts and monitors API calls to understand library or system interactions and potentially modify them.	Hooking tools (e.g., Frida, Microsoft Detours)
Performance Analysis	Measures the execution time and resource usage of programs to identify performance bottlenecks.	Performance profilers, Benchmarking tools
Instrumentation	Injects additional code into the program to gather more information during its execution.	Dynamic Binary Instrumentation tools (e.g., PIN, DynamoRIO)
Environment Simulation	Simulates different operating environments to observe how the program behaves under varied conditions.	Virtual machines, Emulators
Behavioral Analysis	Observes and logs the behavior of the software to build a model of its normal	Behavioral detection systems, Anomaly detection tools

	operation and anomaly detection.	
Reverse Debugging	Allows the execution of a program to be reversed, making it easier to understand the cause of bugs.	Reverse debuggers (e.g., RR - Record and Replay framework)

Additional Tools and Methods

- **Snapshot and Restore:** Takes snapshots of the running state of an application, allowing testers to restore to a specific state quickly and repeatedly.
- **Taint Analysis:** Tracks the flow of sensitive data through the program to detect security vulnerabilities such as buffer overflows and injections.

Dynamic analysis is crucial for a comprehensive understanding of how software operates in a real-world scenario, which is especially beneficial for debugging complex issues, enhancing security, and improving performance. These methods allow reverse engineers to see beyond the static code and delve into the practical implications and operations of a software system under actual running conditions.

One goal of reverse engineering is to make big pictures of how software is built. This helps manage the hard parts at the bottom. A common use of this is to make new documents for software that doesn't have complete or lost documentation [27]. The article talks about how understanding complex software is hard and can have mistaken if done by hand. But using special tools and methods like software visualization, code checking, adding notes, and changing can make things better and more correct. Reverse engineering, which uses compiler technology, can also help find connections that might not be easy to see in the code. The way software is designed, like with C++ language, is important for understanding and keeping it working well. These tools also help keep software good by helping test parts, making changes safe, and setting rules for how things talk to each other [28]. The research discussed how methods and procedures are reused by organizing them in patterns of inheritance, even though this might make the code less neat. What sets C++ apart is that these approaches are built into the language itself and can be understood by looking at the source code without executing it. In contrast to older languages like COBOL that rely on specific labels and numbers for organization, C++ doesn't follow that approach. The key concept is to convey ideas and solutions clearly and straightforwardly [29].

Fazlida et al., [28] investigated how humans understanding is really important, as using these parts wrongly can make software complex and hard to manage. They found nine types of issues that can happen during the development of software through a list of mistakes in object-oriented software [28]. According to the authors, lacking sufficient knowledge can impact the kinds of bugs that occur. Software engineering focuses on understanding programming languages and the process of creating software, while reverse engineering examines programs written in a particular language. SRE (Software Reverse Engineering) employs Cypress and Stamp to gather information about design and execution [30]. The research discovers that because there's so much data and code, this method is filled with challenges, similar to issues the old system faced when being considered for reengineering.

Bennett 1995, [31] has discussed about "the huge software systems," he means the old systems we need but aren't sure how to manage in our organization. The authors mention that most old systems usually lack sufficient, precise, and up-to-date documentation. This research utilizes these challenges and presents automated or partially automated methods to recover a system's components design [30],[32],[33].

Envision the system's metadata and database structure by employing the principles of software program visualization theory, as suggested by [34], [35], [36]. The article emphasizes that reverse engineering involves professionals analyzing malware behavior by comparing hard drive contents before and after malware introduction. This practice allows for easier documentation, maintenance, and modification of software at a more accessible level [37]. Additionally, this approach is valuable for creating a secure system and a system that addresses vulnerabilities effectively[38]. strongly emphasizes that vulnerability analysis and reverse engineering play a pivotal role in the progress and ongoing development of cybersecurity investigations and advancements [39].

1.6 VeraCrypt Software

VeraCrypt, a free and open-source program, can encrypt entire drives or encrypted volumes effectively. It functions smoothly, encrypts data well, and is easily accessible, making it a practical and reasonable choice. Stemming from the TrueCrypt project, which ended in 2014, VeraCrypt carries on its legacy. When an authorized user wants to access encrypted data on a computer's storage, VeraCrypt decrypts it seamlessly. The article thoroughly covers the overall features of the TrueCrypt software, explains its encryption methods involving hashing, encryption algorithms, and file system filter drivers [40]. The article discusses the availability of open-source security tools and highlights a gap in their adoption by end users. It specifically focuses on encryption-based open-source cybersecurity tools and aims to provide guidance for individuals interested in exploring these tools in the digital world [41].

The authors enlightened that disk encryption protects data on devices such as hard drives and flash cards when not in use. TrueCrypt stands out for its complete encryption, reliability, efficiency, and ease of use. It maintains data security by encrypting and only allowing authorized decryption. Encryption involves methods like hashing, algorithms, and file system filters [40]. VeraCrypt, released in 2016, is a sophisticated encryption application with several key features [42]. For more information about VeraCrypt, you can visit their official GitHub repository at https://github.com/veracrypt/VeraCrypt. These features include:

- Creation of virtual encrypted disks that can be mounted as real disks.
- Encryption of whole partitions or storage devices like USB flash drives or disks.
- Parallelization and pipelining techniques ensure that data can be read and written at a fast rate, comparable to unencrypted drives.
- Encryption can be accelerated using hardware capabilities available in modern processors.
- Provides features like plausible deniability, including hidden volumes through steganography and hidden operating systems.

1.6.1 Volume Encryption

VeraCrypt offers two types of volumes: file-hosted volumes and partition/device-hosted volumes.

- **File-Hosted Volumes**

This type of volume is a regular file that can be located on any storing medium. It is an independent file that can be easily moved or copied as needed [7].

- **Partition or Device-Hosted Volume**

With this type of volume, VeraCrypt encrypts an entire hard disk partition. It is also possible to encrypt various devices such as USB flash drives, solid state drives (SSDs), or external disks [9].

1.6.2 System Encryption

VeraCrypt uses on-the-fly encryption, also known as transparent encryption, to encrypt the system partition or the whole system drive where Windows is installed[43]. Encryption and decryption of files happen automatically in the background after the initial setup by the user. System encryption ensures the highest level of security because all files created on the system partition are permanently encrypted. This includes temporary files, logs, and sensitive files containing private personal information that Windows generates and stores. By employing system encryption, these files are always protected. The process involves a Pre-Boot Authentication (PBA) mechanism, which is managed by the VeraCrypt boatload.

1.6.3 Plausible Deniability

In situations where a user is compelled to disclose their password by an adversary, VeraCrypt offers two types of plausible deniability. A VeraCrypt partition or device appears as a cluster of random data until it is decrypted. This can be explained as the partition being completely wiped by overwriting it with random data, creating a plausible scenario. However, it is crucial to prevent any data leaks for this to be effective.

The second challenge is that when using system encryption, an unencrypted VeraCrypt bootloader is employed, which can be easily identified. To overcome this, the hidden operating system feature can be utilized. A hidden volume is established within the available space of a separate VeraCrypt volume. Even when the outer volume is accessible, the concealed volume remains indiscernible, because the free space of a volume consists of random data, just like the seemingly random data in VeraCrypt's unencrypted volume [43]. It is important to use completely different passwords for the two volumes. The hidden volume functions similarly to a normal one as shown in Figure 1.5. When mounting the volume, the password used determines whether the visible or hidden volume will be accessed.

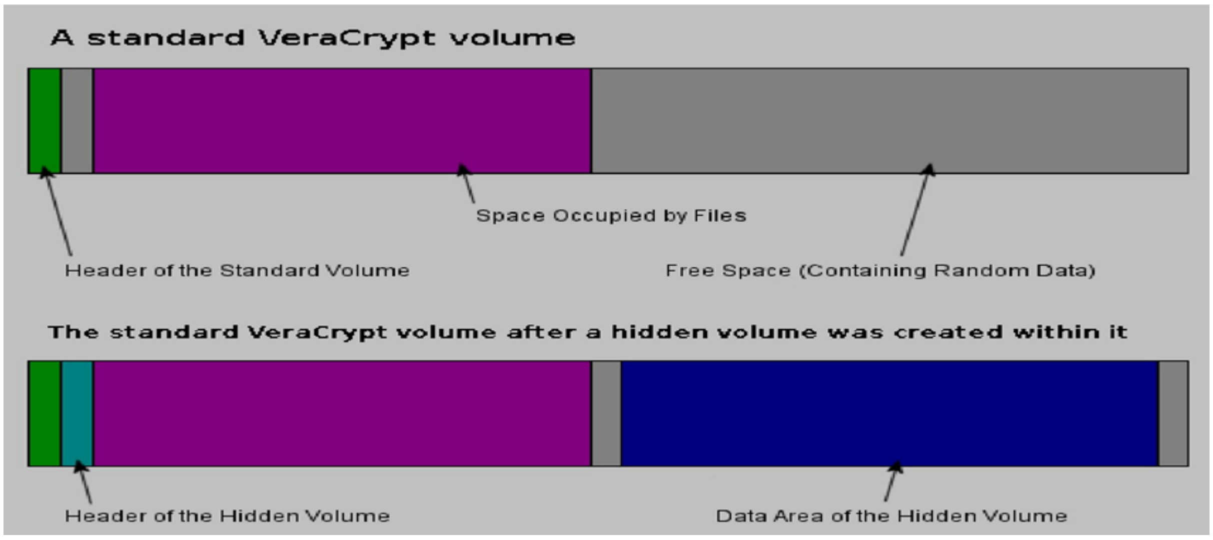

A standard VeraCrypt volume
Space Occupied by Files
Header of the Standard Volume
Free Space (Containing Random Data)
The standard VeraCrypt volume after a hidden volume was created within it
Header of the Hidden Volume
Data Area of the Hidden Volume

Figure 1.5 Hidden Volume [42]

A hidden operating system operates in a similar manner to a decoy operating system. During setup, a decoy operating system is created to serve as a cover. This is necessary because when system encryption is enabled, the VeraCrypt bootloader, which cannot be hidden, is generated for pre-boot authentication (PBA). Consequently, it becomes impossible to prove the absence of an encrypted operating system stored on the disk. The choice between the decoy system and hidden system depends entirely on the password entered during PBA.

Using the decoy system as frequently as possible is crucial. If someone gains access to the decoy system, they could scrutinize its usage. If the system's uptime is suspiciously low, the existence of another system might be questioned. Meanwhile, when the hidden operating system is in use, it appears to be installed on the same partition as the decoy system, even though it actually resides on a separate artition. Every action performed is transparently redirected to the hidden volume where the hidden operating system is located. Neither the system nor applications are aware that data is being written to a different location [10].

While using the decoy system, it is not possible to accidentally overwrite data in the outer encrypted volume since it resides behind the decoy system partition. However, it is possible to overwrite the area where the hidden operating system is located while accessing the outer volume. To address this scenario, VeraCrypt offers a protection mechanism for the hidden volumes (hidden operating system). When activated, this feature prevents any write operations to the hidden volume area. If such an action occurs, VeraCrypt reports an "invalid parameter" error to the system, blocking write operations to both the hidden and outer volumes until they are dismounted.

In general, Plausible deniability is a concept in data encryption that allows users to conceal the existence of encrypted information or activities. It serves as a protective layer, allowing individuals to deny the presence of specific data even when subjected to external scrutiny or coercion. Key encryption tools like TrueCrypt and VeraCrypt use plausible deniability mechanisms such as concealed volumes, hidden operating systems, dual passwords, decoy files, and steganography. These techniques help users maintain the security of their data even when exposed to external scrutiny or coercion [44].

1.6.4 Optimization and Acceleration

Parallelization in VeraCrypt involves dividing data processing into multiple instances running simultaneously. This approach helps accelerate the encryption and decryption of data by breaking down each packet into smaller chunks. The number of chunks depends on the CPU's threads, and the speed increase is directly proportional to the number of threads available [43].

Pipelining is another technique used by VeraCrypt to ensure efficient access to stored data, making it appear as if the data is not encrypted. This asynchronous processing eliminates the need to wait for the decryption of a file located in VeraCrypt's encrypted volume. Instead, the data decryption occurs in parallel, allowing for faster access. It's important to note that pipelining is supported only in the Windows version of VeraCrypt When it comes to hardware acceleration, VeraCrypt takes advantage of it if the CPU supports it. Specifically, hardware-accelerated Advanced Encryption Standard (AES) is utilized by VeraCrypt when the CPU supports AES-NI instructions. This means that if the CPU has the necessary capabilities, VeraCrypt automatically employs hardware acceleration for faster encryption and decryption processes.

1.6.5 Hashing and Encryption

In VeraCrypt, there are different encryption algorithms available for selection, such as AES, Camellia, Kuznyechik, Serpent, Twofish, and combinations of these. These algorithms utilize a key size of 256 bits and a block size of 128. To secure the data, VeraCrypt utilizes the XTS mode of operation, which improves data integrity and minimizes the risk of tampering. Hashing is the process of creating a unique cryptographic representation, called a hash value, of plaintext. It is used to ensure the integrity of communication or files. There are various types of hash algorithms, such as MD5, SHA1, SHA256, and CRC32. The Secure Hashing Algorithm (SHA) is preferred at 256-bits or above for cryptographic functions, with SHA384 and SHA512 functions also available [7].

When selecting a password, users can also choose from various hash algorithms such as RIPEMD-160, SHA-256, SHA-512, Whirlpool, and Stribog. These hash algorithms are used to process the password and ensure its security during the encryption process.

Table 1.9 Comparison Among Different Types of Encryption Software

Encryption Software	Type	Platform
VeraCrypt	Full-disk encryption and file encryption	Windows, Linux, Mac
BitLocker	Full-disk encryption	Windows
GPG (GNU Privacy Guard)	File encryption and email encryption	Windows, Linux, Mac
AxCrypt	File encryption	Windows
TrueCrypt	Discontinued	Windows, Linux, Mac
ProtonMail	Email encryption	Web, iOS, Android

Table 1.9 provides a rundown of encryption tools like VeraCrypt, BitLocker, GPG, AxCrypt, TrueCrypt, and ProtonMail. These tools can secure your entire disk or individual files on Windows, Linux, and Mac systems.

VeraCrypt has a variety of benefits over competitors, including the following:

- It is a flexible encryption tool with powerful encryption algorithms and cross-platform compatibility that enables users to build hidden volumes inside encrypted volumes.

- It gives users plausible deniability and user-friendliness, making it simple to encrypt and decrypt data.
- High data security is provided by VeraCrypt's safe and user-friendly encryption mechanism.
- Strong encryption algorithms provide high data security by allowing users to construct hidden volumes inside encrypted volumes.
- In order to guarantee the security of user data, VeraCrypt is dependable and safe encryption software that provides a wide range of encryption techniques and features [19].

1.6.6 Components of VeraCrypt

User Interface: VeraCrypt has a graphical user interface (GUI) that makes it easy for users to encrypt their files and folders. The user interface includes a wizard that guides users through the process of creating encrypted containers and volumes.

Encryption Algorithms: VeraCrypt supports several encryption algorithms, including AES (Advanced Encryption Standard), Serpent, and Twofish. These algorithms ensure that data is protected from unauthorized access and cannot be easily decrypted.

Encryption Modes: VeraCrypt supports several encryption modes, including XTS (XOR-Encrypt-XOR), CBC (Cipher Block Chaining), and LRW (Liskov-Rivest-Wagner). These modes provide different levels of security and performance, depending on the user's needs. In VeraCrypt, there are several hash algorithms employed, including SHA-256, SHA-512, RIPEMD-160, and Whirlpool. These algorithms play a crucial role in generating digital fingerprints or signatures of data, ensuring that the data remains unaltered and free from tampering [43].

Key Files: VeraCrypt supports the use of key files as an additional layer of security. Key files are small files that are used to unlock encrypted volumes. They can be any file, including images, audio files, or documents. VeraCrypt requires users to set secure passwords in order to safeguard their data. Passwords must contain a mix of letters, numbers, and symbols and be at least 20 characters long.

Volume Creation: VeraCrypt allows users to create encrypted volumes, which are virtual drives that can be mounted and accessed like a regular drive. Volumes can be created on a hard drive, USB drive, or other storage devices.

Hidden Volumes: VeraCrypt supports the creation of hidden volumes, which are encrypted volumes within encrypted volumes. Hidden volumes can be used to store sensitive data, and if someone forces the user to reveal the password, they can disclose the password for the outer volume, which will only show non-sensitive data.

VeraCrypt possesses the capability to encrypt the complete system disk, encompassing both the operating system and all associated files. This feature provides the maximum level of security, as all data on the system drive is encrypted and cannot be accessed without the password. Overall, VeraCrypt is a robust encryption software that provides multiple layers of security to protect sensitive data shown in Figure 1.6. Its components work together to ensure that data is encrypted, authenticated, and protected from unauthorized access [45].

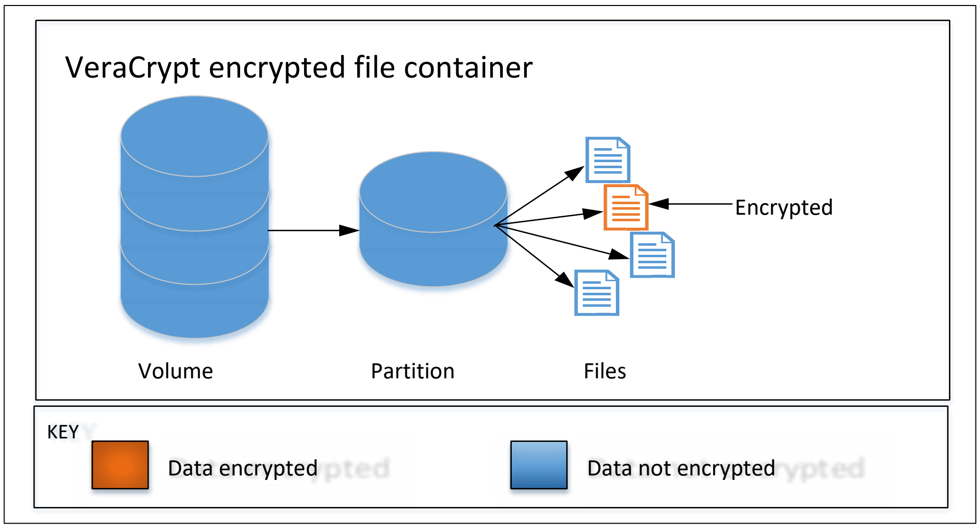


Figure 1.6 Vera Crypt Coded File Container

1.6.7 Programming Comprehension

The structure of a program is what allows humans to manage software, which is inherently large and complex. To make it more manageable, we break it down into smaller parts, with each part representing a "unit" in the program. This helps us create a mental image of the program. A similar process occurs during reverse engineering. Reverse engineers must reconstruct the map of the different components that make up a program. Understanding the programmed structure is the first step in comprehending software assets and is essential for software development. It is needed for tasks such as defect detection and repair, code organization and optimization, adapting to different platforms, utilizing reusable components, integrating old and new technologies, and adding new features. Furthermore, this understanding should be shared within the team and documented for training new team members due to personnel changes. The Structured and object-oriented analysis approaches are still relevant as software moves from initial development to maintenance and evolution [46].

The knowledge required to model an existing system and its surrounding environment bears many similarities with the knowledge required to model the real world and create a new software system. Hierarchies, for instance, are utilized in both situations to create abstractions and lessen complexity. For the purpose of reengineering an existing system to be more object-oriented, object-oriented structure models may be helpful. Understanding the architecture of existing software might benefit from understanding the ideas and patterns used in software engineering.

1.7 IDA Pro Software

IDA Pro is a popular disassembler and debugger software developed by Hex-Rays for reverse engineering and analysis of computer software binaries. It supports various processor architectures and performs advanced code analysis [47]. The software features a multi-view graphical user interface with dockable windows, interactive code navigation, cross-referencing, and code patching capabilities. It also includes a built-in scripting language called IDC, which allows users to automate tasks and perform batch processing. IDA Pro also includes a powerful debugger that supports various techniques and can be attached to running processes. It also offers a rich ecosystem of plugins and extensions, and an optional Hex-Rays Decompiler module. IDA Pro is widely used for tasks like vulnerability analysis, malware analysis, software protection, and code auditing. In the research I used IDA Pro to examine the veracrypt Encryption Software.To use IDA Pro, follow these steps

- Install the software and follow the installation instructions.
- Open a binary file and start IDA Pro.
- Initialize the analysis by determining the binary's architecture, entry points, and metadata.
- Navigate the interface using the graph view and text view, functions window, and strings window.
- Basic analysis includes disassembling, decompiling, searching, and cross-referencing.
- Advanced analysis includes breakpoints, debugging, plugins, renaming, commenting, patching binaries, and saving and exporting data.
- Work with patches and modifications by directly modifying the binary.
- Save and export data for reporting or collaborative purposes.

The feature of this software is depicted in Table 1.10.

Table 1.10 Features of IDA Pro Software

Feature	Details
Analysis Capabilities	IDA Pro analyzes executable files, disassembles, decompiles, and represents code.
Decompile	IDA Pro features built-in decompile for binary code conversion.
Interactive Interface	IDA Pro offers interactive interface for code navigation, data structures, function calls, and modification.
Scripting	IDA Pro supports scripting languages for automating tasks and creating plugins.
Debugger Integration	IDA Pro integrates with debuggers for real-time code analysis.
Collaborative Analysis	IDA Pro allows multiple users to collaborate on the analysis of a binary file, with each user able to make annotations, add comments, and share their analysis with others.
Plugin Architecture	IDA Pro's plugin architecture enables custom functionality development.

Table 1.10 emphasizes that IDA Pro is a robust tool for examination and understanding. software. It helps with tasks like taking apart executable files, turning complex code into simpler language, and collaborating on analysis. IDA Pro also works with scripting, links to external debuggers, and allows people to build their own tools using plugins.

1.8 Problem Statement

The problem addressed in this thesis is the design and development of a Framework for examining anti-forensics disk encryption tools using software reverse engineering principles. The objective is to develop a protocol that can be used to evaluate the encryption tool's performance in hiding data and thwarting forensic examination. This procedure ought to be able to spot any flaws or vulnerabilities in the technology that forensic investigators might use to recover data. Locating the encryption tool for investigation is the initial step. This tool should be widely used and recognized for its capacity to hide data from forensic experts. Conduct static analysis, which involves studying the encryption tool's design and code without running it.

The main goal of this framework is to offer a comprehensive assessment of how well the anti-forensic disk encryption tool works and to identify any weaknesses or openings that forensic investigators could use. Security analysts, forensic investigators, and experts in digital forensics can use this protocol to assess and improve the effectiveness of disk encryption solutions.

1.9 Research Motivation

For forensic investigators, a big question is: How can we make sure we've got all the information? It's important to give a report that's fair and neutral. But when data is hidden, encrypted, or unknown, it's hard to be fair because hidden stuff stops us from analyzing everything properly. Using disk encryption tech is making investigations harder and slower. People doing illegal things use encryption to hide evidence. Denning and Baugh, 1997 [48] Presented various cases and situations where obtaining data in a decrypted format was unattainable. There are numerous occurrences where retrieving non-encrypted data has proven to be exceptionally challenging. Matthew et al., 2010 [49] highlight effectiveness of the attack's contributions is validated by assessing the proposed techniques within the analytical framework. Theoretical insights and practical evaluations are intended to provide guidance for the continuous enhancement of digital forensic strategies.

Singh, 2000 [50] has illustrated the historical precedence of confidential communication, instances such as the use of substitution ciphers by notable figures like Julius Caesar and the imprisoned Mary Queen of Scots, as well as secret writing during the Greek-Persian Wars in 480 B.C., exemplify the imperative need for privacy in correspondence. In each case, the communicators aimed to ensure that their messages remained confidential and comprehensible solely to those possessing the knowledge to decipher them back into a readable form. The stakes were high, as any unintended revelation of the message's content to an unauthorized party could lead to severe consequences. An unfortunate case in point is Mary Queen of Scots, who faced the grave consequence of treason and execution when someone successfully decoded her messages, subsequently using them against her.

Research proposal addresses the growing challenge of anti-forensic disk encryption tools in digital forensics, focusing on a comprehensive approach using software reverse engineering principles to develop a protocol for examining these tools. This holistic approach acknowledges the limitations of conventional methods and the need for innovative strategies to counteract emerging anti-digital forensic tools. The objectives are to advance digital forensic techniques, enhance investigator capabilities, provide practical tools and guidelines, and ensure investigative integrity by countering encryption-based obfuscation techniques. This approach could potentially reshape digital investigations and contribute significantly to the ongoing battle against digital evidence concealment.

1.10 Aims and Objectives

Goals and objectives are essential for establishing precise goals and ensuring that efforts are focused on achieving specific outcomes. They are widely used in various fields. The primary objective of this research was to develop a protocol for anti-forensic disk encryption tool examination using principles of software reverse engineering.

- **To study the impact of anti-forensic techniques on Forensic Investigations.**
  The study of anti-forensic techniques in digital forensics and cybersecurity is crucial for investigators to recover and analyze digital evidence. It involves research objectives, literature review, data collection, testing scenarios, analysis, compliance with legal and ethical considerations, real-world case studies, recommendations, and publication.

- **To propose a protocol for examination of the said tools with focus on reverse engineering.**
  The protocol for examining reverse engineering tools involves a systematic approach, including scope definition, tool selection, documentation, legal compliance, analysis, testing, reporting, recommendations, and presentation.

- **To perform software reverse engineering on popular disk encryption software using industry tool IDA pro disassembler**.
  Reverse engineering disc encryption software using the IDA Pro disassembler is a complex and potentially sensitive task. It involves static and dynamic analysis, identifying encryption algorithms, obfuscation techniques, vulnerability assessment, documentation, testing, reporting, and recommendations.

- **To compare and validate the proposed and existent disk encryption software structures so as to enable forensic investigation.**
  The study validates disk encryption software structures for forensic investigation, ensuring legal compliance, and comparing algorithms, key management, and security features. Future work should focus on improving encryption software structures.

1.11 Scope of the Research Work

An encryption tool code and organizational structure are examined during a static analysis without the programmed being run. This study, which involves looking at the encryption tool source code to find any potential security vulnerabilities or weaknesses, is typically carried out using specialized software tools. The following tasks can be included in the scope of work for a static analysis of an encryption tool:

- Reviewing the source code of the encryption tool is the first step in a static analysis. This entails checking the source code for flaws, weaknesses, and other problems that can affect the tool's security.
- Code flow evaluation- To find any potential security holes or flaws in the code, the static analysis tool can run a code flow analysis. This analysis might point out parts of the code that an attacker might use to access sensitive data without authorization.
- Flow analysis of data to find any potential problems with data security, the static analysis tool can also carry out a data flow analysis. This analysis can point up places in the code where sensitive information, such encryption keys or other private information, could be leaked.
- Static analysis of an encryption tool is, in general, a crucial step in verifying the tool's security and dependability. Before the tool is used, it can assist in locating potential security holes and flaws in the code, reducing the chance of data breaches and other security lapses.

1.12 Research Contribution

A substantial addition to the field of computer forensics can be made by designing and creating a protocol for anti-forensic disc encryption tool analysis utilizing the concepts of software reverse engineering. In order to make it difficult or impossible for forensic investigators to extract data from encrypted discs, anti-forensic tools are used. On the other side, reverse engineering is the act of dissecting software to learn how it functions and how it might be altered.

Designing and developing a methodology for anti-forensic disc encryption tool analysis using the principles of software reverse engineering will significantly advance the area of computer forensics. Anti-forensic technologies are employed to render data extraction from encrypted discs for forensic investigators challenging or impossible. Reverse engineering, on the other hand, is the process of analyzing software to determine how it works and how it might be modified.

The protocol can help academics better comprehend anti-forensic methods and software reverse engineering in addition to its many practical uses. The development of such a procedure can provide information about the conception, use, and analysis of anti-forensic techniques. It can also influence the creation of forensic investigative methods and equipment in the future. In conclusion, employing software reverse engineering concepts to create and develop a protocol for anti-forensics disc encryption tool analysis can significantly advance the field of computer forensics and increase the efficacy of investigations.

1.13 Outline of Thesis

Chapter 1: Introduction

This chapter provides an overview of the research topic, introduces the significance of digital forensics, anti-forensics, encryption techniques, reverse engineering technology detection tools, and presents the problem statement. It outlines the objectives of the study, discusses the scope and limitations, and provides an overview of the thesis organization.

Chapter 2: Literature Review

In this chapter, a comprehensive review of existing literature related to digital forensics, anti-forensics, reverse engineering, and other security-related detection is presented. The chapter also discusses the strengths, limitations, and gaps in the current research, setting the foundation for the proposed study.

Chapter 3: Methodology

This chapter details the methodology used in the research. It describes the process of achieving the set goals and objectives.

Chapter 4: Results and Discussion

The research findings are revealed in this chapter, which also examines them. Performances of various encryption software are evaluated and compared as part of this evaluation.

Chapter 5: Conclusion and Future Directions

This chapter summarizes the main findings of the study and presents the conclusions drawn from the research. It discusses the contributions made by the study and addresses the fulfilment of the objectives; the chapter also suggests potential areas for future research.

1.14 Summary of Chapter

In this research, a technique for deciphering anti-forensic disc encryption methods utilizing the concepts of software reverse engineering is proposed. This procedure aims to aid forensic investigators in better comprehending the inner workings of these instruments and in locating any potential flaws or vulnerabilities. The process entails a number of phases, including identifying the encryption tool being used, obtaining the encrypted data as well as the tool itself, reverse engineering the instrument using debuggers and disassemblers, and spotting any potential flaws or vulnerabilities in the tool's design. The report also covers the difficulties in deciphering between legitimate and malicious uses of anti-forensic disc encryption systems, including the use of obfuscation techniques. Additional details on anti-forensic tactics and software reverse engineering. Without having access to the source code, a software system is examined to determine how it functions. This method is known as software reverse engineering. This method is frequently used by security researchers to find vulnerabilities in software, software engineers to learn how existing software functions, and hackers to take advantage of those vulnerabilities. In order to make it difficult or impossible for investigators to obtain the evidence, anti-forensic tactics are procedures used to conceal or obfuscate digital evidence. Disk encryption, steganography, and file wiping are a few examples of anti-forensic methods that can be employed for both legitimate and illicit objectives. It's crucial to remember that using anti-forensic methods to conceal evidence of criminal activities is unlawful in and of itself. Overall, the technique outlined in the study offers forensic investigators a methodical way to examine anti-forensic disc encryption methods and find any potential flaws that could be exploited during forensic investigation.

CHAPTER II
LITERATURE REVIEW

In the last chapter, we discussed the research and gave a brief summary of important information from previous studies, with a special focus on things like disc encryption, digital forensics, anti-forensic techniques, and reverse engineering methods. Including a literature review is very important in any research project because it helps in gathering information and past research about the topic. It involves carefully looking at and evaluating the work that has been done before on the subject. The literature review looks at what we know about the topic right now and tries to find gaps, trends, differences, and patterns in the existing research. Therefore, in this chapter the brief study of existing encryption software and research are discussed.

The examination of anti-forensic disk encryption tools and the use of software reverse engineering can be relevant to various stakeholders, including law enforcement agencies, forensic investigators, security researchers, software developers, and academics. A broad literature review can cater to this diverse group of potential readers.

2.1 Encryption

VeraCrypt is a secure encryption system that supports various encryption algorithms, including AES, Serpent, and Twofish, with a default of AES with a 256-bit key length. It uses a key derivation function (KDF) to derive encryption keys from user-provided passwords or keyfiles, based on the PBKDF2-HMAC algorithm. Users can create encrypted volumes as files or partitions, with random data generated during the creation process. VeraCrypt performs encryption and decryption on-the-fly, ensuring transparency and seamless access to encrypted data. It supports the creation of hidden volumes and operating system encryption, making it difficult to detect the operating system. It also includes security features like RAM encryption and pre-boot authentication.

2.1.1 Popular Encryption Software

In this section, the details of the existing popular disk encryption software is discussed named ProtonMail, GNU Privacy Guard and AxCrypt.

2.1.2 ProtonMail

An encrypted email service that focuses on providing users with a high level of privacy and security for their email communications. It was developed by a team of researchers at CERN (the

European Organization for Nuclear Research) in Switzerland, and it was launched in 2014. ProtonMail is notable for its end-to-end encryption, which means that only the sender and the recipient of an email can read its contents. Following are some key features and functions of ProtonMail.

- **End-to-End Encryption**

ProtonMail keeps your emails super secure by using end-to-end encryption. This means your messages get turned into secret code on your device and can only be turned back into regular emails on the receiver's device. Even ProtonMail's servers can't peek at your email content.

- **Zero-Access Encryption**

ProtonMail employs zero-access encryption, which prevents the service provider (ProtonMail) from accessing the contents of users' emails. This is achieved by encrypting the data on the user's device before it's sent to the server, and only the user holds the decryption key.

- **Open Source**

ProtonMail has made its encryption algorithms and codebase publicly available as open source, enabling the security community to conduct audits and confirm the validity of the service's security assertions.

- **Anonymous Sign-Up**

ProtonMail provides an anonymous signup process with a username and password, ensuring privacy. It also offers encrypted metadata and accepts anonymous payment methods like Bitcoin and cash, but users should exercise caution.

- **Self-Destructing Emails**

ProtonMail allows users to send self-destructing emails, which are automatically deleted from the recipient's inbox after a certain period of time.

- **Two-Factor Authentication (2FA)**

ProtonMail enhances user account security by enabling two-factor authentication (2FA), requiring users to provide a password and a code generated by a smartphone app or SMS, providing a second layer of protection against unauthorized access even if the password is discovered.

- **Cross-Platform Support**

ProtonMail offers secure email management through web browsers and mobile apps, providing a user-friendly interface on desktop and laptop computers. It offers specialized apps for iOS and

Android platforms, integrating end-to-end encryption and optional two-factor authentication for improved security.

- **Custom Domains**

ProtonMail enables users to create custom domains for email accounts, enhancing personal and professional image, maintaining brand consistency, and providing a personalized email experience.

- **Free and Paid Plans**

ProtonMail offers both free and paid subscription plans. While the free plan provides basic features, the paid plans offer additional storage, custom domains, and advanced features. PGP Support ProtonMail supports PGP (Pretty Good Privacy) encryption, allowing users to send encrypted emails to non-ProtonMail recipients.

- **Secure Contacts**

ProtonMail provides privacy and security by encrypting email recipient contact details, safeguarding sensitive information and preventing unauthorized access, ensuring only account holder and decryption key holders can access it.

- **Secure Attachments**

ProtonMail's encryption technology ensures secure attachment sharing, protecting sensitive information and supporting various file types and sizes, making it a reliable option for secure file sharing.

- **Accessible Interface**

ProtonMail is a user-friendly encrypted email service, offering privacy and security through end-to-end encryption and zero-access methods. It's popular among individuals and organizations seeking a secure alternative to traditional email services, but users should be cautious about personal login credentials and encryption keys.

- **Features and Limitation of ProtonMail**

ProtonMail is an email service that focuses on secure and private communication, offering strong encryption, user privacy features, and commitment to data protection. Key features include end-to-end encryption, zero-access encryption, open-source encryption, anonymous sign-up, self-destructing messages, Two-factor authentication, PGP compatibility, custom domains, simplified encryption, cross-platform support, and secure data centers. However, ProtonMail has limitations such as a learning curve, limited features in the free plan, dependence on ProtonMail or another email service that supports OpenPGP encryption, password recovery challenges, and no traditional POP3 support. The free plan has limitations on storage, sending limits, and advanced features, while the paid plans offer advanced features and increased storage. Additionally, ProtonMail may have limited third-party integration due to its emphasis on security and privacy. Custom domains are only available in paid plans, which may be a limitation for users seeking this feature in the free plan. Overall, ProtonMail is a popular choice for individuals and organizations seeking secure and private email communication.

2.1.3 GNU Privacy Guard

GPG often referred to simply as GnuPG, is a free and open-source software application that provides encryption, decryption, digital signatures, and key management services for securing communications and data. It is based on the OpenPGP (Pretty Good Privacy) standard and is designed to provide privacy and data integrity in email communication, file sharing, and other forms of digital communication. Here's a comprehensive overview of GPG's features, components, and applications.

- **Encryption and Decryption**

GPG allows users to encrypt messages and files to ensure that only authorized recipients can read the content. Encryption is performed using recipient's public keys, and decryption requires the recipient's private key.

- **Digital Signatures**

GPG lets you make digital signatures for messages and files. These signatures prove who sent the stuff and make sure it didn't change while traveling. To check the signature, you use the sender's public key, and the signature itself comes from their private key.

- **Key Management**

GPG manages encryption keys and digital signatures keys. Users can generate their own key pairs (public and private keys), share their public keys with others, and keep their private keys secure. The key pair is used for encryption and digital signatures.

- **Key Servers**: GPG supports the use of key servers, which are online repositories where users can upload and download public keys. This allows others to easily find and use your public key for encryption and verification.

- **GPG's trust** model allows users to establish a level of trust for public keys. Users can mark keys as trusted, untrusted, or unknown. This helps in determining the reliability of a key owner's identity.
- **Web of Trust:** GPG's Web of Trust is a decentralized model of trust verification. Users can sign each other's keys to vouch for their authenticity. This helps in building a network of trust within the GPG community.
- **Symmetric and Asymmetric Encryption:** GPG can do two types of encryption: one where you use the same key for both hiding and revealing information (symmetric), and another where you use a pair of keys (one public and one private) for hiding and revealing information (asymmetric)

2.1.4 AxCrypt

A file encryption software that focuses on providing easy-to-use encryption for individual files and folders. It's designed to protect sensitive data by encrypting it with strong encryption algorithms. AxCrypt is available for both Windows and macOS platforms. Here are the details and descriptions of AxCrypt:

A Key Features

- **File Encryption:** AxCrypt allows you to encrypt individual files or entire folders. Encrypted files are protected with strong encryption algorithms, making it difficult for unauthorized users to access the content.
- **Strong Encryption Algorithms:** AxCrypt uses AES-256 encryption, which is considered one of the most secure encryption standards available. This ensures that your encrypted data remains secure against various attacks.
- **User-Friendly Interface:** AxCrypt provides an easy-to-use interface that integrates with the Windows Explorer context menu. You can right-click on a file or folder and choose to encrypt it using AxCrypt.
- **Passphrase-Based Encryption:** AxCrypt uses a passphrase for encryption and decryption. You create a strong passphrase that only you know, which serves as the encryption key. This passphrase is not stored anywhere, ensuring your data's security.
- **Key Sharing:** You can share encrypted files with others. Recipients need to know the passphrase to decrypt and access the content.

- **Automatic Encryption:** AxCrypt can automatically encrypt files as they are added to a designated folder. This helps ensure that new files are protected without manual intervention.
- **Cloud Storage Integration:** AxCrypt supports integration with cloud storage services like Dropbox, Google Drive, and Microsoft OneDrive. Encrypted files can be uploaded to these services securely.
- **Decryption on the Fly:** Encrypted files can be decrypted on the fly when you open them. You don't need to manually decrypt the file before opening it; AxCrypt handles this process seamlessly.
- **Secure File Deletion:** AxCrypt includes a secure file deletion feature that ensures deleted files cannot be easily recovered by using data recovery tools.
- **Multiple Platform Support:** AxCrypt is available for both Windows and macOS platforms. This allows you to encrypt and decrypt files across different devices.

B Usage

- **Single File Encryption:** Right-click on a file, select the "AxCrypt" option from the context menu, and choose "Encrypt." You'll be prompted to enter a passphrase. The file will be encrypted, and you'll need the passphrase to decrypt and access it.
- **Decryption:** Double-click on an encrypted file. AxCrypt will prompt you to enter the passphrase, and then it will decrypt and open the file with the associated application.
- **Automatic Encryption:** You can set up folders for automatic encryption. Any files added to these folders will be automatically encrypted using the passphrase.
- **Sharing Encrypted Files:** To share encrypted files, you can provide the recipient with the encrypted file and share the passphrase through a secure channel. The recipient needs both the file and the passphrase to decrypt the content.

C Limitations

- **Individual File Encryption:** AxCrypt primarily focuses on individual file encryption. If you're looking for full-disk encryption or system-level encryption, you might need to consider other solutions.
- **Passphrase Management**: The security of your encrypted files depends on the strength of your passphrase. You need to ensure you create a strong passphrase and keep it secure.
- **Dependent on Passphrase**: If you forget your passphrase, you won't be able to access your encrypted files. AxCrypt doesn't have a password recovery mechanism.

- **Manual Encryption**: While AxCrypt offers automatic encryption for designated folders, it doesn't automatically encrypt all files on your system. You need to use the context menu or the AxCrypt application to encrypt files.
- **Windows and macOS Only**: While AxCrypt covers the major desktop platforms (Windows and macOS), it might not be suitable for users who need encryption on other platforms.

2.2 Related Literature on Encryption

This section offers an in-depth examination of prior research related to encryption, organized by the year of publication. Bursac et al.'s 2017 [25] research offers a comprehensive analysis of four open-source encryption tools, offering valuable insights for data protection. However, it has limitations in tool selection, static analysis conditions, and user experience evaluation. R. Singore, 2024 [51] review provides a comprehensive understanding of forensic cybercrime and computer forensics, focusing on digital forensics and various types. It highlights the interdisciplinary nature of the field and identifies future directions, but acknowledges limitations.

Tan et al. (2020) [52] offers a valuable contribution to the understanding of BitLocker encryption and its security implications. It offers key insights into the encryption mechanism, decryption processes, and potential vulnerabilities, while proposing practical recommendations for secure usage. However, the limitations, such as the limited scope, rapid evolution of the field, lack of empirical evaluation, and potential bias, should be considered when interpreting and applying the findings.

C. Gastel 2019 [53] study on SSD firmware reveals systemic weaknesses in hardware encryption, highlighting security risks and the need for regular firmware updates and industry best practices for data encryption. Bhushan et al.,2022 [54] explores Bitlocker encryption, a crucial tool in digital forensics, and its forensic implications. It discusses its encryption process, decryption methods, and future trends. The review emphasizes the need for ongoing research and collaboration between academia, industry, and forensic communities to overcome encryption challenges and navigate the complexities of modern encryption technologies.

Evangelin et al., 2013 [55] emphasizes the importance of reverse engineering in software re-engineering, suggesting a static analysis approach for Java bytecode reverse engineering, despite its limitations. S Signgore, 2024 [51] Examine Forensic Cybercrime and the Role of Computer Forensics" provides an in-depth analysis of forensic cybercrime and its role in investigating digital crimes. It highlights the importance of advanced forensic approaches and the potential of emerging trends like artificial intelligence and machine `learning.

Wali et al., 2024 [56] emphasizes the significance of information privacy and confidentiality in digital security, highlighting the limitations of symmetric encryption methods and the need for secure encryption and user verification schemes. It introduces digital signatures, cryptographic techniques, fostering trust and non-repudiation. Maletic et al.,2004 [56] Supporting Source Code Difference Analysis" introduces meta-differencing, an XML representation of source code, and the srcML format for analyzing syntactic changes between versions. They demonstrate how srcML and srcDiff can be used with standard XML tools.

2.3 Digital Forensics and Anti-Forensic

This section discusses the detailed review of digital forensics and anti-forensics according to the year of publication. In 2002, the author Jonah [1] explored the evolution of codes and ciphers, their impact on history, and the intellectual competition between creators and breakers. It portrays codes as a relentless battle for existence, evolving or adapting to new forms. In the same year 2002, Wolfe [57] discussed the alternative methods for investigating computer cases where suspects refuse cooperation and encryption keys, emphasizing the importance of investigators' methods and strategies in determining case success or limitations.

Nguyen, 2004 [58] investigated the quality of cryptographic implementations in software systems, focusing on GNU Privacy Guard (GPG). It reveals critical flaws in GPG v1.2.3, including a vulnerability to GPG-generated ElGamal signatures. The study emphasized the need for continuous evaluation and auditing of cryptographic software, especially in secure email applications.

Geiger, 2006 [59] have developed the techniques to examines 13 commercial counter-forensic tools in digital forensics, identifying operational weaknesses and filesystem fingerprints, and introduces 19 tools and Aperio for automated detection. Hosgor et al., 2006 [60] discovered the rise of digital crime, such as identity theft and online piracy, necessitates the use of digital forensics to combat these crimes, however, cybercriminals have developed anti-forensic techniques, highlighting the need for robust countermeasures to maintain the integrity of digital crime investigations. Cappaert et al., 2006 [61] discussed the use of encryption techniques to improve software security and prevent tampering attacks, introducing code encryption and integrating code dependencies into a static call graph.

Canfora and Massimiliano, 2007 [62] examined four different algorithms and analyzed their memory construction rate, key size, CPU utilization time, and encryption speed to understand the resource consumption and the time taken by each algorithm to complete its task. Further investigation is required to assess the power usage of these algorithms.

Cappaert et al., 2008 [63] has introduced a method of partial encryption that relies on code encryption. in this approach, the binary codes are divided into smaller segments and each segment is encrypted. When the code is executed, the encrypted segments are decrypted by the users. This way, the partial encryption addresses the limitations of decrypting the entire binary code at once, as only the necessary segments of the code are decrypted during runtime. Salama et al., 2008 [64] evaluates six encryption algorithms used in internet and network applications: AES, DES, 3DES, RC2, Blowfish, and RC6. It compares their effectiveness under various settings and data block sizes, the findings highlight the impact of encryption algorithm choice on security and performance.

Alomari et al., 2009 [65] evaluated encryption algorithms and modes of operation for securing storage devices, focusing on disk drives. It evaluates the XTS mode of operation, an IEEE-approved standard, and compares it with other encryption algorithms. The findings help designers make informed decisions about disk encryption, balancing security and performance. Cohen, 2009 [66] prioritized the growing demand faced by criminal justice agencies to investigate crimes perpetrated through the Internet or other electronic platforms, this necessitates the development of resources and procedures aimed at effectively searching, locating, and preserving electronic evidence.

Casey et al., 2011 [67] has investigated the impact of full disk encryption (FDE) on digital investigations, highlighting its potential to obstruct access to crucial legal evidence. Yadav et al., 2011 [68] discovered the internet has boosted digital forensics, utilizing commercial and open-source tools to investigate and solve digital crimes, the review classifies digital crimes, categorizes tools, and proposes a simplified investigation model for efficiency, useful for law enforcement and forensic experts. Pimenidis , 2011 [69] focused on the challenges faced in investigating and prosecuting electronic crimes due to the presence of artifacts on computer systems. Computer criminals employ countermeasure techniques to hinder investigations, resulting in costly and time-consuming processes, the paper delves into the problem of anti-forensics in computer forensic investigations.

Mouton and Venter, 2011 [70] demonstrated the practicality of Digital Forensic Readiness (DFR) in wireless sensor networks, reducing DFI costs. Our research focuses on cloud environments and extensive evidence analysis. Alharbi et al., 2011 [71] focused on a method called "proactive and reactive functional process," which seeks to examine techniques that could evade forensic investigations and make live investigations easier. This method is created by looking at what we already know and is meant to automate all the steps of the proactive part. It will be used to build and run proactive and reactive systems using a special language and automatic code generation. The method addresses two big challenges: predicting attacks ahead of time and making the proactive part better by adding a feedback loop for improvement.

Janet, 2012 [72] discovered involves finding solutions for handling encrypted data, most of the research focuses on ways to either keep or obtain decryption keys. Forensic investigators are always faced with the obstacle of sophisticated encryption, which is why this issue needs to be stressed. As per Principle 1 of the Good Practice Guide for Computer-Based Electronic Evidence, investigators have to determine whether to take a copy of volatile memory as encryption modifies the memory state.

Adedayo and Shao, 2013 [73] examined the role of data encryption in information security and its impact on digital forensics. It examines the TrueCrypt encryption solution and the need for innovative technologies to balance information security and digital forensics, the ultimate goal is to strike a balance between encryption's role and digital forensics' pursuit. Sharjeel et al., 2013 [74] examined how modern trends, advancements in technology, and strategies to hide evidence (anti-forensics) affect digital forensics. It also looks at how experts in digital forensics are responding to these challenges.

Andrew Davies, 2014 [75] have delved into the difficulties encountered by forensic investigators when it comes to identifying concealed volumes and operating systems within computer hard disk data. These challenges are primarily attributed to the intricate characteristics of TrueCrypt and the constant evolution of digital security technologies. Dealing with encryption is still super important for successful investigations. The articles share important information about how encryption investigations work, including both cases where they worked and cases where they didn't. It gives useful tips and shows examples of successful attempts to break encryption, encouraging forensic investigators to be proactive in their digital investigations. Understanding encryption is still very important for successful investigations [76] [67].

Balducci and Devlin, 2015 [45] enlightened that the objective of data forensics in the context of TrueCrypt is to unlock an encrypted container and obtain its contents in a readable format. This container may take the form of a disk partition, a single file volume, or even an entire disk. Researchers anticipate that by scrutinizing TrueCrypt's weaknesses on multiple fronts, they can uncover a method for decrypting it. A thorough examination of source code revealed only four minor vulnerabilities.

Casey, 2016 [77] highlighted the blurred distinction between technical and investigative tasks in digital evidence, leading to confusion and erroneous judgments, resulting in court restrictions and the establishment of digital forensic units. Victor and Ray, 2016 [78] discovered that digital forensics primarily centers on the identification and examination of cybercrimes, encompassing various subdomains that include database, Internet of Things (IoT), malware, network, drone, cloud, wireless, data, and mobile forensics. Vincze et al., 2016 [79] emphasized that the 21st century has witnessed significant progress in secure computing and network technologies, which, in turn, have posed fresh challenges for the field of digital forensics. Traditional approaches have become insufficient due to the intricacy of digital information and the prevalent use of encryption and anonymization tools. To enhance the efficacy of digital forensics, they suggest continuous training, the establishment of standardized procedures, and increased investment in research and development.

Louiza et al., 2017 [9] has discovered full disc encryption (FDE), a process that encrypts every sector of a disc volume without additional storage. It provides a formal delineation of security concepts, particularly in chosen-plaintext and chosen-ciphertext attacks. The review classifies various modes of FDE operation and introduces a new concept called a "diversifier" that reduces input/output operations per second by 4%. This innovative approach has the potential to revolutionize full disc encryption and information security.

Awotunde et al., 2017 [80] discussed cryptography, which involves using codes to protect sensitive data by encrypting and decrypting it. However, this process comes with a trade-off in terms of resource usage. David et al., 2017 [46] presented tests if a binary file header is compatible with the PZ-PE file format specification, using real-life scenarios and targeted attacks, future research will explore data sections or opcodes sections to expand detection criteria. Jusas et al., 2017 [81] provided evaluations and investigations on digital triage, focusing on live and post-mortem triage, mobile device triage, and triage tools, highlighting its potential to solve case backlogs in digital forensics but highlighting challenges for developers. Muhamad, 2017 [82] provided an overview of the Advanced Encryption Standard (AES) algorithm, its key features, encryption process, and strengths, comparing it to other encryption algorithms like DES, 3DES, and Blowfish, and comparing them to gain a comprehensive understanding of AES's unique capabilities.

Dogan, 2018 [83] introduced a new data hiding algorithm using pixel pairs and chaotic maps, utilizing the modulo function. The algorithm's effectiveness is evaluated through embedding secret data into images, revealing high visual quality, efficient running time, data security, and high payload capacity, promising widespread adoption. Arewa, 2018 [84] explored Nigeria's cybercrime challenges, focusing on national laws and digital forensics. It uses a library-based approach to analyze materials, highlighting the need for specialized skills and digital protocols for detection and evidence preservation. It recommends legal and institutional mechanisms to combat cybercrimes.

Arshadet al., 2018 [85] explored the digital forensics plays a crucial role in criminal investigations, but its credibility in legal proceedings is marred by complexity and ever-changing legislation. The article explores emerging technologies and recommends the establishment of best practices and testing methodologies to bolster the trustworthiness of electronic evidence. Vadlamudi et al., 2018 [3] discovered that cloud computing is crucial for modern life, but cybercriminals are developing anti-forensic techniques. This paper analyses challenges in cloud forensic investigations, contributing to efforts to enhance methodologies and stay ahead of threats. Meijer and Gastel, 2018 [86] examined how numerous SEDs from various vendors implemented complete disc encryption. These companies cover almost half of the SSDs now available for purchase. We discovered that the drives we looked at contain serious security flaws. Many times, the drive's contents can be recovered without the use of a password or secret key, completely getting around the encryption.

Hassan, 2019 [87] the primary objective of this article is to outline the various anti-forensic attack types and tactics, as well as the difficulties that forensic investigators may run into in the field of digital forensics. Another goal is to show how to defend against these attacks while also taking into consideration their limitations. Patel, 2019 [88] examined the performance of cryptographic algorithms like AES, DES, and Blowfish in encryption and decryption of data files, analyzing execution time and memory utilization. It aims to help users select the right algorithm for their specific needs and consider time and memory constraints, enhancing data security in diverse IT environments.

Christnatalis and Mahmud, 2019 [89] enhanced digital signature security in administrative documents using a hybrid cryptographic approach combining RSA, AES, and Blowfish encryption methods, the Kohonen Self-Organizing Map (SOM) method is used for signature recognition, ensuring the chosen signature image matches user-entered keys and confirming ownership accurately. Jitha and Kumar, 2019 [90] discussed the importance of encrypting SMS messages and using strong cryptographic algorithms like RSA and AES for secure key management and message encryption. It concludes that these measures are crucial for maintaining the security of SMS communication in the digital age.

The research emphasizes the importance of having effective on-site techniques for the detection and protection of encrypted data. It underscores the need for prosecutors to consider Full

Disc Encryption (FDE) issues when obtaining search warrants. Additionally, it delves into strategies for collecting physical evidence from crime scenes and the protocols for extracting digital evidence from operational computer systems in the context of on-site investigations [91],[92].

Wani et al., 2020 [93] has provided an in-depth understanding of file system anti-forensics, a crucial aspect of digital forensics. It explains various techniques and tools used, empowering investigators to counteract obfuscation methods and enhance their digital evidence analysis skills. Majed et al., 2020 [87] investigated the aim of this effort, which is to clarify diverse classes of anti-forensics attacks and techniques, and provide methods to counter them. It is essential to establish strong security measures for data backups and logs and conduct a thorough examination of system traces to identify the presence of anti-forensics attacks.

The utilization of anti-forensic techniques by criminals to conceal their sources of evidence can pose significant challenges for investigators when attempting to uncover evidence related to criminal activities. Anti-forensic techniques are predominantly employed to obfuscate evidence, and within the realm of cloud investigations, they serve as a means to establish reliability and uniformity. The significance of anti-forensics within the cloud environment is well-documented [16], [94],[95].

Shekhanin et al., 2020 [96] highlighted the importance of examining both file system fragmentation and statistical characteristics in the context of steganography detection. Its primary goal is to pinpoint unusual attributes within the file system and offer valuable insights into its functioning. As a result, the research contributes significantly to the advancement of steganoanalysis methods, which are of paramount importance in fields such as digital forensics and cybersecurity. Moreover, it sets the stage for the development of novel approaches in the realm of steganoanalysis. Reza et al., 2020 [97] assessed digital forensics' current state, highlighting challenges like data volume, storage formats, encryption, and legal considerations, suggesting future research for efficient tools and standardized procedures**.**

Kavrestad, 2020 [5] described fundamentals of digital forensics" is a practical guide to digital forensics, written by an experienced examiner with law enforcement and academia backgrounds. It covers essential principles, methods, and constraints, focusing on their application in crime investigations, the book collaborates with the Swedish Police and emphasizes a holistic approach. Sassani et al., **2020** [98] evaluated symmetric encryption algorithms across HDD, SSHD, and SSD-based NAND Multi-Level Cell (MLC) flash memory using two storage encryption software solutions. Results show AES algorithm outperforms Serpent and Twofish in random read and write speeds on HDD, but Twofish is slightly faster in sequential reading on both SSHD and SSD-based NAND MLC flash memory.

Data encryption on disk drives can be accomplished via either file system encryption or full disk encryption. In the case of file system encryption, each individual file is encrypted using its distinct key. This approach offers protection for most files stored on the disk drive, but it doesn't extend to data located outside the file system. In contrast, full disk encryption employs a single symmetric key to secure all the data on the entire disk drive. This encompasses not just files within the file system but also encompasses hidden files, swap files, file metadata, temporary files and caches, registry files, and boot sector data. Full disk encryption guarantees the safeguarding of data in all regions of the disk drive [99] [40] [100].

Mohammed, 2021 [101] have discussed digital forensics is crucial in addressing cybercrimes and security breaches in the digital era. Key trends include cloud, social media, and IoT forensics. Challenges include maintaining data integrity and personnel. Opportunities include USB forensics, intrusion detection systems, and AI integration. Digital forensics is essential for safeguarding against cybercrimes and addressing challenges. Al-Dhaqm et al., 2021 [102] proposed a metamodel to unify investigative practices and enhance the effectiveness and reliability of digital forensic procedures, addressing ambiguities and redundancies in traditional methods.

Anzuoni, 2022 [103] discussed Plausible deniability is a key cryptographic security feature protecting encrypted data from coercive adversaries, various schemes have been developed to address this, but some lack adequate deniability guarantees or have performance drawbacks, the Shufflecake scheme, introduced in this review, balances performance and security, offering 99.6% disk efficiency and minimal slowdown compared to regular encryption tools . Abdul Rehman et al., 2022 [104] explored cybercrimes, highlighting the use of digital artifacts to identify and prosecute cybercriminals, they highlight the role of forensic agencies and law enforcement in digital forensic toolkits, highlighting research gaps and proposing future research to enhance forensic experts' capabilities.

Stoykova et al., 2022 [105] highlighted the significance of reverse engineering in digital forensics for law enforcement, highlighting legal challenges, gaps, and inadequate regulation. It recommends further development of digital forensic regulation to address legal complexities and uncertainties, improving law enforcement agencies' ability to acquire and interpret digital evidence while adhering to legal and ethical standards. Paul and Viswanathan et al., 2022 [106] assessed digital forensics tools by examining their reliability, scalability, and precision, this evaluation helps investigators choose appropriate tools for diverse fields and tackle privacy issues.

Jean et al., 2022 [107] described cyberattacks are becoming more sophisticated, causing significant losses and requiring innovative solutions, cybercriminals use anti-forensics techniques to conceal their tracks, making detection more difficult, traditional security solutions are insufficient, and forensics investigators need advanced tools to combat these attacks the review explores various methodologies, tools, and challenges.

Alotaibi et al., 2023 [108] Proposed a thorough framework for examining Unmanned Aerial Vehicles (UAVs) in forensic scenarios, filling a void in drone forensics by encompassing four stages: Preparation, Data Collection, Analysis, and Documentation. David et al.,2023 [7] investigated and examined the relationship between encryption, energy consumption, and security in wireless environments. It highlights the need for energy-efficient secure protocols and the benefits of implementing encryption schemes as software in Wireless LANs. the research methodology evaluates various symmetric and asymmetric key encryption algorithms.

2.4 Reverse Engineering Techniques

In this section the brief concept and approach of the existing research in the field of reverse engineering techniques has been discussed. Perry, 1987 [109] introduced the semantic interconnection model, providing insights into program development, modifications, and version compatibility, and offers tools to comprehend system construction and evolution over time. Müller et al., 1992 [110] introduced a reverse engineering environment that uses spatial and visual information in graphical representations of software systems. The interactive graph editor enables engineers to interact with system data, construct subsystem structures, and interpret software design, enhancing data quality and decision-making in software engineering.

Muller et al.,1993 [111] presented a reverse engineering method for creating abstract representations of a subject system, utilizing the Rigi system for interactive construction and analysis of subsystem hierarchies, identifying interconnected components. Collberg et al., 1998 [112] discussed how language-based tools can be used to change programs from source code to functional versions that are difficult to reverse engineer. This helps in lowering the barriers for reverse engineering.

Kazman et al.,2001 [113] has discussed the architecture of reconstruction, which is a crucial process in software engineering, obtaining the "as-built" architecture of an implemented system from an existing legacy system using specialized tools. It documents the existing architecture, validates conformance, facilitates evolution, supports reuse, enhances product lines, and establishes best practices.

Stroulia and Systa, 2002 [114] introduced dynamic reverse engineering, focusing on understanding system dynamics for processes and utilization in legacy systems, highlighting challenges in comprehensive program comprehension. Collberg, 2002 [115] discussed three methods to safeguard software from piracy, reverse engineering, and tampering: software watermarking, code obfuscation, and tamper-proofing.

Eisenbarth et al., 2003 [116] introduced a partially automated method for understanding feature linkage to source code, combining dynamic and static analyses to identify necessary computational units. Marcus et al.,2004 [117] explored the concept of reverse engineering, a process that entails extracting source code from software binaries or executables with the intention of circumventing protective measures. This technique is commonly employed by individuals referred to as "crackers" who seek to discern and surpass security mechanisms through a range of tools and methods. The specific applications of reverse engineering can diverge based on the distinct goals and motivations of these individuals.

Chikofsky and Elam, 2005 [118] discovered the reverse engineering is a common practice where individuals analyze source code, whether it is their own code created in the past or code authored by others. This process involves examining the code to gain insights, learn from it, and enhance one's understanding, ultimately resulting in new discoveries and knowledge. Tore et al., 2005 [119] emphasized that source code analysis is valuable for extracting architecture, reverse engineering, and reengineering of software programs, it helps with understanding, maintaining, and reusing programs. Additionally, it is predicted that there will be over 1 trillion lines of code in software, demonstrating the growing significance of source code manipulation and analysis. Chikofsky and Eilam, 2005 [118] provided a thorough rundown on software reverse engineering, covering the process, tools, malware reversing, piracy, and copy protection, making it a valuable resource for those involved**.**

Aggarwal and Jalote, 2006 [120] examined the dynamic analysis, which involves running the target application or IoT device firmware with test cases to observe its behavior during runtime, this approach helps validate and verify the results obtained from static analysis techniques, by dynamically analyzing the target, researchers can gain real-time insights into its functioning, ensuring the accuracy of the findings obtained through static analysis.

Binkley, 2007 [121] emphasized that software reverse engineering first arose as a solution to tackle challenges related to software maintenance, the IEEE-12191 standard acknowledges reverse engineering as a vital technology for managing systems where the source code serves as the only dependable representation. From its inception, reverse engineering has demonstrated its efficacy in resolving diverse issues within the realm of software engineering. Song et al., 2007

[122] investigated how the dynamic decryption and encryption of code can protect data against static analysis, allowing polymorphic viruses and shell code to avoid detection by security tools.

Czeskis et al., 2008 [123] highlighted the differences between a deniable file system and an encrypted file system and looks at TrueCrypt software's capacity to provide a deniable file system, which allows plausible deniability. The structure of files and folders in an encrypted file system is still visible, but the actual content of the files is still inaccessible without the right decryption keys. David et al., 2008 [124] explored the automation of reverse engineering (RE) using digital imaging and computer vision. It explains the concept of RE, which involves creating computer-aided design (CAD) models from pre-existing objects and components. The chapter discusses the conventional approach to RE, which uses coordinate measuring machines (CMMs), and explores digital imaging and computer vision as alternatives. It highlights the potential of laser-based range scanners for data acquisition and introduces a detailed processing pipeline for creating CAD models. The chapter emphasizes the importance of various parameters in modeling accuracy and equips readers with the necessary knowledge.

Larsen, 2008 [125] highlighted the use of debuggers for file reverse-engineering, especially in malware analysis, and raises awareness about their accessibility. However, it lacks in-depth guidance on their usage due to their high level of expertise. Altheide et al., 2008 [126] proposed a reliable method for forensic examinations on hard drives with whole-disk encryption software, eliminating the need for unauthorized or proprietary software.

McGrath et al., 2009 [127] has introduced a methodology to extract probative value from encrypted files, focusing on hybrid cryptosystems. It also aims to pinpoint the original plaintext file, revealing the elusive content. This is particularly relevant in cases involving encrypted child pornography images and terrorism information, addressing "Cui Bono?" can expand digital investigations and enhance the investigative process. Song et al., 2009 [122] has delivered an overview of dynamic code decryption and encryption, which involves self-modifying or self-generating code. Encryption is utilized to maintain data confidentiality, particularly in binary code, where it acts as a defense against static analysis. Polymorphic viruses and polymorphic shell code employ diverse encryption techniques to enhance their protection.

Edward et al., 2010 [128] has discovered the protocol reverse engineering (PRE), which is used for communication analysis to approximate protocol definitions, contrasting with software reverse engineering, which focuses on source code or program implementation. PRE-can also infer communication using software reverse engineering, but requires specific tools and analysis processes. Entity analysis techniques are also widely used. Zhiqiang et al., 2010 [129] has introduced a new reverse engineering technique called REWARDS, which automatically uncovers program data structures from binary executables. It uses dynamic analysis, backward type resolution, and in-memory data structure layout, making it a powerful tool for memory image forensics and binary fuzzing.

Srinivasan et al., 2010 [130] has discussed the main concern in detecting pirated copies and protecting software programmers and content from manipulation, using proactive and post-compromise detection strategies to identify attackers before and after attacks**.** Singh and Singh, 2010 [131] introduced a revolutionary data security method that uses floating-point values for encryption. To strengthen the encryption, a novel strategy was used that generated different integers for words that appeared again and lengthened keys. Mark and Cipresso, 2010 [132] emphasized that software reverse engineering is a challenging and ongoing aspect of computer security, with the main goal of uncovering higher-level program concepts. The insights obtained from reverse engineering are valuable for various tasks, including memory forensics, malware analysis and generation, comprehension of network protocols and cryptographic techniques, digital rights management, and auditing program binaries.

Stamp and Cipresso, 2010 [132] discovers reverse engineering concepts, their universality across platforms, and their importance in detecting viruses and enhancing software security, offering potential benefits in commercial and institutional IT settings. Müller et al., 2010 [133] highlighted the significance of reverse engineering techniques in enhancing software engineers' program knowledge, as it aids in understanding complex software systems and their inner workings.

Treude et al., 2011 [134] has primarily focused on software analysis, which involves examining software artifacts for reverse engineering purposes. The analysis can be done using static, dynamic, or historical approaches, which involve studying execution traces, developer changes, and versioning systems, binkley's survey on source code analysis offers a thorough and comprehensive overview of this entire process. Steven et al., 2011 [135] has introduced techniques for reverse engineering feature models, focusing on identifying parent features and automating the recovery of essential constructs, the approach is applied to Linux and eCos kernels, and FreeBSD, a project without a feature model, the heuristic consistently ranks the correct parent feature among the top results, streamlining the information needed for modelers, reducing the number of options from thousands to five or fewer**.** Treude et al., 2011 [134] conducted a study on the repercussions of cybercrime, their goals, and countermeasures. They focused on security reverse engineering, highlighting five essential procedures: dissecting assembly code, recording discoveries, sharing knowledge, expressing work, and communicating findings.

Canfora et al., 2011 [136] has discussed as the reverse engineering is a rapidly evolving field that involves studying software components, source code, and communication data formats, requiring systematic approaches, education, empirical evidence, and guidelines. However, reverse engineering is often employed for alternative intentions. In essence, when one comprehends the application's structure or gains access to the source code, the opportunity arises to modify it and utilize it for personal or self-serving objectives. Reverse engineering is a common avenue for pursuing such alternative intentions [124], [137]. Antunes et al., 2011 [138] developed a novel method and tool are introduced for creating protocol specifications from network traces, utilizing typical client-server interactions without relying on implementation-specific details.

Treude et al., 2011 [134] scrutinized reverse engineering techniques used by software engineers to combat security attacks, highlighting the unique challenges faced by government organizations and suggesting improvements for tool and process enhancement. The research paper of Jozwiak et al., 2011 [139] aimed to identify encrypted files in digital forensics investigations using various techniques like file signatures, extensions, metadata, and indicators. Canfora et al., 2011 [140] the research highlighted the importance of reverse engineering in software development, particularly for outdated systems with inadequate documentation, highlighting its achievements, significance, and challenges.

Holger et al., 2012 [141] emphasized the necessity for the enhancement of reverse engineering methods and tools to cope with more extensive codebases, address the unique attributes of industrial code, and provide solutions tailored to specific domains, the research endeavors to create a standardized framework for methodically modeling, scrutinizing, constructing, testing, and upholding embedded software. Sasirekha and Hemalatha, 2012 [142] examined the existing software protection methodologies and offers recommendations to improve their effectiveness in safeguarding software from various threats for economic advancement. Chlumecký, 2012 [143] investigated into the challenges of preserving legacy systems, highlighting the need for cost-effective solutions and the preservation of critical institutional knowledge. It also provides a roadmap for organizations to transition to modern, object-oriented architectures, offering valuable insights for streamlining the process.

Tao et al., 2012 [144] explored the significance of understanding code changes in software development, focusing on Microsoft's context. It highlights inadequacies in current practices, such as obtaining completeness, consistency, and risks, and emphasizes breaking down changes into sub-changes. Rekhis and Boudriga, 2012 [145] has proposed a theoretical approach to digital investigations that addresses anti-forensic attacks, it outlines an investigation process, introduces state-based logic, deploys security solutions, and generates plausible scenarios from tampered evidence, the proposed inference system mitigates the impact of these attacks, and a case study demonstrates its feasibility.

Apoorva and Kumar, 2013 [146] has given suggestions to compares AES, Twofish, CAST-256, and Blowfish algorithms for symmetric key cryptography, analyzing their performance under different data loads, speed, block size, and key size**.** Balogun and Ying Zhu, 2013 [73] has discussed the truecrypt software, which provides plausible deniability, complicating digital forensic investigations of encrypted disk drives, a backdoor technology is needed, implementing restrictions to prevent misuse. Jóźwiak et al., 2013 [147] has provided both theoretical and practical methods for identifying and scrutinizing concealed volumes created using cryptographic utilities, applicable to both known and unknown passwords.

Thongkamwitoon, 2014 [148] proposed the digital forensic techniques to trace image origins through multiple rounds of capturing and reproduction. It introduces a theoretical framework and an algorithm for detecting recaptured images, with 99% accuracy rate. Balci et al., 2014 [149] provided an in-depth understanding of malware analysis techniques, including reverse engineering, static and dynamic methods, open-source toolkits, and Yara rules for identifying threats and victims. Zhang and Jia Fan,2014 [150] provided an in-depth examination of TrueCrypt's data encryption mechanisms, elucidating the mathematical processes and procedural steps employed to secure information effectively. additionally, the research delves into techniques that investigators can employ to interact with TrueCrypt-protected data, accompanied by suggestions for enhancing its security, represents a pioneering effort, offering insights into the deciphering of TrueCrypt passwords, and it is readily accessible for all interested readers.

Gonzalez et al., 2015 [151] highlighted the usability challenges faced by reverse engineers in software reverse engineering, emphasizing the importance of prioritizing usability in tool design for effective vulnerability discovery and malware analysis and addressing growing demand in these critical domains. Narayan et al., 2015 [152] provides a classification of tools for automatically reverse engineering protocols, dividing them into two categories: those focusing on the finite state machine and those prioritizing the protocol's structure. Kaplan et al., 2015 [153] explored the reverse engineering is a method of examining and dismantling a system to understand its creation and assembly, often used in turbine manufacturing to enhance design.

Lim et al., 2016 [154] has introduced a novel protection scheme for Android applications to protect against dynamic reverse engineering. The scheme checks if the device is rooted and undergoing debugging, halting execution if these conditions are met. It effectively detects evasion techniques and is implemented as a standalone executable, making it applicable to applications without source code access. Cheng et al., 2016 [155] provided the future learn, an educational platform, can help mobile device users improve their cybersecurity knowledge to reduce risks like financial losses and social engineering.

Narayan et al.,2016 [156] has presented a framework for automatic protocol reverse engineering tools, categorizing them based on finite state machine and protocol format approaches. It highlights different methodologies and areas of application, highlighting the importance of understanding these protocols for improved network security, intrusion detection, and communication efficiency. Diamantopoulos et al., 2016 [157] has discovered and recommended a system that evaluates software component reuse, combining functional compatibility with quality metrics. It assesses component suitability and assigns reusability scores, enhancing project quality and accelerating development processes.

Canfora and Penta, 2017 [121] explored the software reverse engineering, a rapidly growing field that uses automated tools for in-depth analysis, understanding legacy systems, ensuring online security, and facilitating software maintenance, with future research focusing on novel methodologies and ethical considerations. Panda, 2017 [12] analysed the performance of four symmetric key cryptographic ciphers, DES, Triple DES, AES, and Blowfish algorithms, in data security, it emphasizes the importance of securing information in the digital age and the dual nature of cryptographic algorithms, the analysis uses Java for analysis, focusing on encryption and decryption times.

Landman, 2017 [158] sighted software renovation, focusing on domain model recovery, reverse engineering techniques, and static analysis tools' impact on Java code analyses. Kedziora et al., 2017 [159] explored the security requirements for creating a Deniable File System (DFS) and assesses the efficacy of TrueCrypt disk encryption software. The study's findings indicate that the deniability of a TrueCrypt DFS can be undermined by the Windows Vista operating system, Microsoft Word, and Google Desktop.

Mushtaq et al., 2017 [160] has presented a systematic literature review examines multilingual source code analysis, analysing of 56 papers from 3,820 research papers, it identifies 46 research issues in 13 software engineering domains and highlights emerging trends in static source code analysis, program comprehension, refactoring, reverse engineering, and cross-language link detection. Zuck et al., 2017 [161] the discovered VeraCrypt's encryption and data retrieval in data forensics, highlighting its robust encryption of user passwords and key files, effective file volume identification, and practical password recovery methods. Kedziora et al., 2017 [159], their research explored VeraCrypt's security measures for concealed operating systems, utilizing cross-drive analysis to detect hidden OS volumes and estimate their size.

Senthivel et al., 2018 [162] has disclosed that reverse engineering has outlined six enduring goals with a strong emphasis on enhancing its efficacy, it suggests that for software developers to guarantee reliability and advance their work, a broader and more strategic incorporation of reverse engineering is essential, which includes making deliberate enhancements. Werner et al., 2018 [163] developed a reverse engineering method for understanding integrated circuits, utilizing image processing to reconstruct architecture and simplify gates, it introduces hierarchical netlists for efficient identification and segregation of functional modules, achieving 85%-95% accuracy in cryptographic cores.

Sija et al., 2018 [164] has reviewed 39 methods, procedures, and tools, including state machines, protocol formats, and finite protocol state machines and formats. The effectiveness of these methods depends on their chosen inputs and the appropriate reverse engineering input format. Gobel and Baier, 2018 [165] discovered clandestine data hiding techniques within the ext4 filesystem, focusing on capacity and detection rating metrics, contributing to ongoing discourse on countermeasures in digital forensics.

Zhang et al., 2019 [166] explored TrueCrypt's encryption forensics, focusing on its principles and practical data processing procedures, it provides insights into investigating encrypted volumes, aiming to equip investigators with the necessary knowledge and tools for real-world scenarios, the study concludes that TrueCrypt's encryption software is a powerful shield against intrusion. Shwartz et al., 2018 [167] has examined 16 IoT devices' security status and evaluate affordable black-box techniques, including software and fault injection methods, to recover passwords and firmware.

Meir et al., 2019 [168] investigated the influence of computer utilization in illicit endeavors such as bank fraud and identity theft, with a particular emphasis on security reverse engineering. They pinpointed five essential procedures but highlighted the scarcity of research and constraints in current toolkits. Ain et al., 2019 [169] discovered and reviewed categorizes of 54 studies into six approaches for code clone detection (CCD), highlighting the need for novel methods to address bugs and maintenance costs in software development. Jarkovič, 2019 [170] the hypothesis provides an overview of open-source storage encryption solutions and keyword spotting (KWS) systems, focusing on their core functionalities and attributes. It aims to enhance VeraCrypt with KWS using Mycroft Precise, establishing bidirectional communication and testing, and dismounting mounted volumes when a user-defined keyword is detected.

Dayalan, 2019 [21] discovered reverse engineering, a method that enhances software representations and abstraction by identifying system components and their interrelations, emphasizing domain knowledge and domain bridging in programming processes. Zhang et al., 2019 [166] investigated the TrueCrypt encryption system, specifically from the standpoint of data forensics, it places emphasis on the system's security and susceptibility to data recovery methods, the research explores aspects such as obscuring user passwords, effective encryption mechanisms, and strategies for password retrieval, underscoring the intricate nature of data recovery in this context..

Rahmawati and Dewi, 2020 [171] discussed the generating random numbers using true-random and pseudo-random generators, practical examples, and understanding patterns without specific values, while also providing examples. Votipka et al., 2020 [172] discovered the cognitive processes of 16 reverse engineers, revealing their thought processes and decision-making in software security tasks. It identifies three phases: overview, sub-component scanning, and focused experimentation. The study proposes five interaction design guidelines to enhance the effectiveness of reverse engineering tools. Henry and Peterson, 2020 [173] explored the challenges faced by reverse engineers in security-intensive environments, including limited file sharing, time constraints, and lack of cognitive support tools.

In 2021 year, a novel framework has been introduced by Liu et al., [174] for the manual reverse engineering of communication protocols within embedded Linux IoT systems. This framework has proven its efficacy through the successful reverse engineering of the communication protocol used in the WeMo smart plug. Furthermore, it is adaptable for application in embedded Linux file systems, whether they are read-only or writable. Sharma et al., 2021 [41] endeavored to investigate recent progress in the realm of cybersecurity while evaluating the merits, drawbacks, and obstacles associated with the suggested techniques. The primary focus lies in the exploration of novel forms of cyber threats, established security structures, evolving patterns, recent innovations within the cybersecurity domain, along with the diverse security risks and difficulties. This extensive examination conducted within this study is anticipated to provide significant insights for researchers in the fields of IT and cybersecurity.

Liu et al., 2021 [175] presented a specialized manual reverse engineering framework tailored for the identification of communication protocols in embedded Linux IoT systems, the effectiveness of this framework is evident in its successful application to various IoT systems, the methodology is comprehensively showcased through a detailed examination of the reverse engineering process applied to the communication protocol of the WeMo smart plug, this process involves several key steps, such as extracting the firmware from the flash memory, conducting both static and dynamic analyses, and scrutinizing network traffic patterns.

Mattei et al., 2022 [176], they has explored the underutilization of automated tools in software reverse engineering due to usability concerns. It reveals that most tools offer limited interaction and usability support, with dynamic tools lacking user-friendly interfaces and static tools restricting customization options. Stoykova et al., 2022 [105] discussed the manual nature of reverse engineering in software forensics, its role in identifying code modifications, exposing violations, and its significance in file systems for tool testing, evidence collection, and criminal investigations, while also discussing emerging legal and technological challenges.

Long and Zhaoxin, 2023 [177] introduced Deep Encrypted Traffic Detection (DETD), a novel framework that uses a parallel small-scale autoencoder and an L1 regularization-based feature selection algorithm to extract local traffic features from encrypted data. DETD significantly enhances feature extraction efficiency, achieving anomaly detection rates as high as 99.99%, surpassing other neural network-based algorithms. Zakariyya et al., 2023 [178] discussed the rise of cybercrimes and the development of anti-forensic tools, particularly software reverse engineering. It introduces a framework for evaluating VeraCrypt, an open-source disk encryption software, and its practical utility in real-world scenarios. The framework contributes to fortifying digital forensic capabilities.

Bibi et al., 2023 [179] highlightd that one of the most challenging tasks is finding the right software component from the repository, evaluating and assessing these components depends on various factors, as they are influenced by different technological requirements, goals, and business considerations. Nurgaliyev and Wang, 2023 [180] explored the reverse engineering, a process that deconstructs product source code to extract valuable data, offering practical solutions for enhancing cybersecurity and information security.

The articles examined TrueCrypt, an open-source software, and its use of various hash algorithms to generate key-files, enhancing security measures.[40] [67]. The article discussed the Ever-Changing Disk (ECD), a storage system designed to conceal sensitive information within public data. It complies with plausible deniability characteristics, offers protection against snapshot attacks, and conceals write actions as regular operations. Users can regulate data migration rate [161]. Arif Islam and Kumar, 2023 [181] has sighted the role of reverse engineering in combating cyberattacks, presenting methodologies to improve malware identification and detection systems, enabling swift responses. Hassan et al., 2023 [182] revealed that reverse engineering plays a crucial role in comprehending and improving outdated systems. It encompasses the utilization of diverse techniques, including transformational methodologies, theories related to program comprehension, and technologies specific to reverse engineering.

The research's highlights the need for improved tools to address source code infringement cases and the growth of litigation and plagiarism, while highlighting the advancements in software forensics tools. [183],[184].

Table 2.1 Comparative Analysis of the Related Works

Authors/References	Year	Contribution	Key Features	Limitation
Chikofsky et al., [106]	2005	Presents a comprehensive taxonomy of reverse engineering and recovery techniques	Provides a structured framework for understanding reverse engineering techniques.	The taxonomy is outdated, and some of the techniques are no longer used.
Song et al., [38]	2007	Highlighted the evolution of reverse engineering from legacy systems to software related problems	Provides an overview of the history of reverse engineering	The study focused on the evolution of reverse engineering, and the findings may not be relevant to the current state of the field
Cappaert et al., [50]	2008	Discussed steganography as a technique for hiding secret messages in digital audio, images, and video files	Provides an overview of steganography and its applications	The study focused on steganography in digital forensics, and the findings may not be generalizable to other applications of steganography
Müller [99]	1993	Investigate how the Rigi project helps in reverse engineering by creating mental models from abstractions that match the maintainers' understanding	Offers insights into how reverse engineering can be used to improve software comprehension	The study focused on the Rigi project, and the findings may not be generalizable to other reverse engineering tools and techniques
Singh and Singh [119]	2010	Introduced a novel encryption technique using floating-point numbers to enhance data security	Offers a promising encryption technique that could be used to protect digital data	The technique has not been evaluated in real-world scenarios, and its effectiveness may be limited
Fazlida et al., [24]	2013	Reviewed reverse engineering theories and tools	Provides a comprehensive overview of reverse engineering techniques	Some of the tools and techniques are outdated
Shao and Balogun [60]	2013	Explored gaps in digital forensic methodology and raised concerns about trade secrets and vulnerability disclosure	Provides insights into the challenges of digital forensic investigations	The study focused on gaps in digital forensic methodology, and the findings may not be generalizable to other challenges faced by digital forensic investigators
Jozwiak et al., [135]	2013	The article offers theoretical and practical insights into detecting and analyzing hidden volumes created using cryptographic tools, a crucial aspect of digital forensics and cybersecurity.	The article proposals a comprehensive understanding of cryptographic concealment techniques, focusing on both theoretical and practical aspects of detecting hidden volumes,	The article's narrow focus on FAT32 file systems may restrict its applicability to other file systems, as it may not thoroughly discuss its application beyond FAT32.

			contributing to effective forensic methodologies.	
Zhang et al., [138]	2014	Research aids users forgetting their TrueCrypt password and provides valuable insights for computer forensic analysts investigating criminal activities involving TrueCrypt encryption.	The research provides a comprehensive analysis of TrueCrypt encryption, examining its components such as cryptographic algorithms, encryption modes, key derivation processes, and password verification mechanism.	Article highlights the need for ethical considerations and responsible disclosure practices when presenting password cracking details, considering potential misuse and potential misuse of such information.
Turkanovic et al., [11]	2014	The method introduces a revolutionary encryption concept that utilizes public and private keys.	Security system relies on the computational challenge of factoring the published divisor, n.	Size of prime numbers significantly impacts the computational overhead for encryption and decryption operations.
Casey [64]	2016	Emphasizes the significant role of digital evidence in investigations, highlighting its value in evidence evaluation, probative question resolution, and integrity,	Highlights the crucial role of digital evidence in all stages of forensic investigations, including lead generation, crime scene handling, and probative question resolution.	The work may not offer a comprehensive understanding of their intricacies.
Harriss et al., [10]	2016	Presented an extended, granular taxonomy for anti-forensics	Provides a structured framework for understanding ant forensics techniques	Can be complex and difficult to use
Vinze [66]	2016	offering a comprehensive understanding of the evolving electronic device crime landscape.	Cybercrime presents unique challenges compared to traditional crimes, including vast electronic devices	The lack of consensus on terminology for electronic device crimes can cause confusion and communication challenges within the field.
Shwartz et al., [155]	2018	The article provides a comprehensive security analysis of 16 popular IoT devices, addressing the growing concern of security vulnerabilities in these widely used products.	The research provides recommendations for enhancing IoT device security, offering practical solutions without compromising cost or usability, without unduly impacting manufacturers and users.	The research has limitations in generalizability and should be acknowledged for broader device range considerations.
Global and Baier [155]	2018	Highlighted data hiding within the filesystem layer as a significant anti-forensics' technique	Can be used to detect hidden data	Requires specialized knowledge and tools

Vadlamudi et al. [3]	2018	The article acknowledges the rapid rise in technology usage, particularly cloud services, and highlights the need to address forensic challenges in cloud environments.	Focus is on enhancing data security and integrity in the cloud, emphasizing the vital role of cloud forensic processes in investigating criminal activities and ensuring trustworthiness.	Acknowledges the challenges in identifying evidence sources due to anti-forensic techniques, emphasizing the need for comprehensive understanding and counteraction in cloud environments.
Ahmad et al. [34]	2018	Develop optimized image encryption using particle swarm optimization and chaotic map for secure image communication	Offers a promising image encryption technique that could be used to protect digital images	The technique has not been evaluated in real-world scenarios, and its effectiveness may be limited
Ankita et el., [12]	2017	The article underscores the significance of high-speed security algorithms in both wired and wireless communication	The design employs an iterative looping approach, enhancing the efficiency of the AES algorithm implementation, and its block and key size of 128 bits enhances overall performance.	The proposed implementation is dependent on FPGA technology, which may limit its scalability and adaptability to various hardware platforms.
Goodison et al. [2]	2019	Increasing significance of digital evidence in crime solving and court case preparation underscores the transformative role of technology in law enforcement and the criminal justice system.	Law enforcement attendees acknowledged the extensive evidence analyzed by examiners and the challenges in securing funding and staffing for necessary support.	Highlights the need for adequate resources, including technology and personnel, to effectively implement digital evidence processing solutions in the digital age.
Hassan [18]	2019	The article defines digital forensics, a branch of forensic science, as a method used to collect, analyze, and present digital evidence to solve crimes.	Highlighting the importance of scientific knowledge, methodology, and rigor in ensuring the reliability and credibility of the investigative process.	Rapid advancement of technology presents new challenges and complexities in digital forensic investigations, necessitating continuous adaptation and updates to methodologies and tools.
Sari et al. [6]	2019	The research underscores the necessity of internet-transmitted images to prevent theft and emphasizes the need for effective security measures to safeguard sensitive data.	The system employs Triple-DES encryption and a selective bit approach, enhancing the security of image data and preventing unauthorized access during transmission.	The research's evaluation of the proposed scheme may be limited by its use of six testing images, which could benefit from a more diverse set.
M Dayalan [20]	2019	Discussed the process of reverse engineering, which ranges from code-	Offers a holistic view of reverse engineering	complex and time-consuming

		centric approaches to domain knowledge-centric approaches		
Shekhanin et al., [84]	2020	Emphasizes the need for robust security measures in diverse storage devices to protect sensitive data from unauthorized access, regardless of the storage medium used.	Comprehensive evaluation evaluates encryption algorithms' performance across various storage devices, including HDD, SSHD, and SSD-based NAND MLC flash memory, providing a nuanced understanding of their performance.	The article evaluates AES, Serpent, and Twofish symmetric encryption algorithms, but excludes other relevant algorithms for specific use cases or security requirements.
Mohammad, [89]	2021	Highlight Digital forensics is a crucial advancement in cybersecurity, utilizing investigative techniques to counter cybercrimes and protect user privacy and security.	Comprehensive analysis of digital forensics trends, including cloud, social media, and IoT forensics, reflects the multifaceted nature of digital investigations in contemporary cybersecurity challenges	Discusses threats to digital forensics but lacks detailed exploration of specific scenarios or case studies, requiring a more in-depth analysis of real-world examples.
Al-Dhaqm et al., [77]	2021	Proposed a metamodeling approach as a potential solution to address the lack of standard practices in digital forensics	Provides a roadmap for future research in digital forensics	The study was conceptual in nature, and the effectiveness of the metamodeling approach has not been empirically evaluated
Abdul Rahman et al., [92]	2022	The study emphasizes the importance of analyzing digital artifacts left on target devices after cyberattacks to identify and understand cybercriminals, enabling effective investigation.	Provides a comprehensive overview of various computer forensic domains, encompassing a wide range of scenarios and contexts in which digital forensics is utilized.	Comparative analysis of forensic toolkits may be limited due to its lack of depth, but providing more detailed information could improve its practical utility.
Sharma et al., [37]	2021	Identifies and addresses the existing gap in the adoption of open source security tools by end users. understanding to enhance digital workflow security.	Review explores and compares various open source encryption-based cybersecurity tools, providing a comprehensive overview for cybersecurity professionals, academia, hackers, and business institution	Acknowledges the adoption gap in open source security tools, but lacks a thorough analysis of the specific reasons behind this gap, requiring further exploration.

Stoykova et al., [93]	2022	Discussed the legal and technical questions surrounding file system reverse engineering for law enforcement	Provides an overview of the legal and ethical issues surrounding reverse engineering	The study focused on file system reverse engineering for law enforcement, and the findings may not be generalizable to other types of reverse engineering
Long and Zhang [123]	2023	The article addresses the increasing threat of network attacks utilizing encrypted communication, emphasizing the importance of anomaly detection in encrypted traffic for network reliability.	The experimental results demonstrate DETD's superiority over other neural network anomaly detection algorithms, making it a competitive and high-performing solution in encrypted traffic anomaly detection.	The DETD framework's applicability to diverse network environments and attack scenarios could be enhanced by considering potential limitations or adaptations required for different contexts.
Alotaibi et al., [75]	2023	The research addresses the gap in drone forensics by proposing a conceptual model to address the lack of a comprehensive investigation framework for Unmanned Aerial Vehicles.	The model introduces a structured four-stage investigation process, ensuring a systematic and organized approach to forensic investigation, guiding practitioners through each essential phase.	The study may not fully address practical implementation challenges, resource requirements, and model feasibility in various operational settings.
Hassan et al., [169]	2024	The article explores the evolving cybercrime landscape and the development of anti-forensic tools, focusing on software reverse engineering techniques and VeraCrypt as a case study	Highlights the importance of anti-forensic tools, especially those for encryption, in understanding the challenges faced by digital forensic practitioners and developing effective countermeasures.	The study primarily examines anti-forensic encryption tools, focusing on VeraCrypt, but may not encompass the full range of available tools, necessitating a broader exploration.

2.5 Research Gaps

In previous research on anti-forensic tools, several gaps have been identified. Research gaps in anti-forensic disk encryption tools include evaluating their impact on digital forensic investigation, integrating software reverse engineering, developing countermeasures, developing automated tools, evaluating steganographic techniques, examining hardware-based anti-forensic measures, and understanding usability and user experience. Addressing these gaps can significantly contribute to digital forensics and software reverse engineering principles Here are some possible areas of research that need to be looked into:

- Lack of comprehensive protocols: There may be a lack of well-defined and standardized protocols specifically designed for examining anti-forensics disk encryption tools. Existing protocols may not fully address the unique challenges and complexities posed by these tools.
- Limited understanding of reverse engineering techniques: The principles and methods of software reverse engineering as they relate to anti-forensics disc encryption systems may need to be better understood. This could involve exploring advanced reverse engineering methodologies and tools that can effectively analyses and dissect the functionality of these tools.
- Addressing evolving anti-forensics techniques: Anti-forensics techniques and tools are continuously evolving, making it essential to keep pace with new developments. Research could focus on identifying emerging anti-forensics techniques and devising strategies to detect and counter them within the context of disk encryption tools.

2.6 Limitations of Existing Anti-Forensics Disc Encryption Tool

The following are some of the drawbacks of establishing and creating a protocol for an anti-forensics disc encryption tool examination, utilizing software reverse engineering principles: Restricted testing: Because the tools are made to avoid discovery, verifying the protocol's effectiveness can be difficult. Determining whether the tools have been found and successfully decoded can be difficult. Technology development: New anti-forensic tools are always being created as a result of technological advancements. Regular protocol updates are required to stay up with these changes'-forensics tools are frequently used to conceal unlawful activities, therefore designing a strategy for looking at them involves ethical questions. It is crucial to make sure that the protocol is only applied in morally and legally acceptable ways. Among the available disk encryption VeraCrypt systems, I conducted research only on Microsoft’s Windows.

The protocol for examining anti-forensics disc encryption tools through software reverse engineering faces limitations such as restricted testing, technological advancements, ethical considerations, and a limited scope of research. The protocol's effectiveness is challenging due to the tools' evasion of detection. Regular updates and adaptations are needed to keep up with these changes and maintain its relevance. Therefore, it's crucial to approach the development and application of this protocol with critical awareness.

CHAPTER III
METHODOLOGY

Research methodology refers to the systematic and logical approach that researchers use to conduct research. The methodology is a critical aspect of any research project since it outlines the strategies that the researcher will use to gather, analyses, and interpret data. Also describe how the research's objectives will be achieved in this chapter. The method employed in this study aims to establish a comprehensive framework (shown in Fig. 3.7) for the purpose of software reverse engineering, with a primary focus on VeraCrypt. In order to gain knowledge about the software's individual components, behavior, vulnerabilities, and interactions, reverse engineering comprises a thorough investigation of the software's architecture, operation, and internal mechanisms. The following list of specific steps used by the framework in the reverse.

3.1 Debugging Tools

Software Reverse Engineers use debuggers to change how a program runs, so they can understand what the program is doing while it's running. These tools also allow engineers to control parts of the program's memory while it's running. This helps them get a clearer picture of how the software works and its impact on a system or network. The x64dbg Windbg is one of the debugging tools. To complete the debugging process, the user must know about the creating a model, network analysis tools and check-test in some cases.

Creating a Model

The information collected is simplified into a simple idea that shows how each part fits in the bigger picture. The aim is to take specific details from the original and turn them into a basic model that can be used to make other things or systems. In software reverse engineering, this might be something like a picture that shows how data moves or a chart that shows the structure.

Network Analysis Tools

Engineers can use tools to study how a program talks to other computers over a network. They can see the connections the program is attempting to establish and the information it's trying to send (like Wireshark).

Check and Test

Make sure the simple idea you created accurately represents the real thing by reviewing and testing it in different situations. In software, you can use testing to do this. Once you have tested it, you can use the idea to improve the real thing.

3.2 IDA Pro (Interactive Disassembler)

IDA Pro is one of the most sophisticated disassemblers in the market and is widely used in reverse engineering. It converts binary programs into assembly code that can be read and interpreted by humans. This is crucial for analyzing proprietary or obfuscated code that does not come with source access. Following are the key features of IDA Pro:

- **Multi-processor Disassembly**: Supports a wide range of processors and executable formats, making it incredibly versatile for various platforms and architectures.
- **Graphical Visualization**: Offers a graphical view of the assembly code, showing control flow and program structure in a more intuitive format than plain text.
- **Debugging Capabilities**: Includes a debugger that can attach to running processes or analyze static binaries. This allows for dynamic analysis in addition to static.
- **Extensibility**: Users can write custom scripts in Python or IDC (IDA's own scripting language) to automate tasks or perform specific analyses.

Debuggers

Debuggers are tools that allow programmers to execute a program step-by-step, pause it, inspect variables, and understand the flow of execution at a granular level. They are crucial for dynamic analysis in reverse engineering. Following are the key features of Debuggers:

- **Breakpoints**: Allows setting breakpoints at specific instructions or conditions, halting execution at crucial points to examine the state of the application.
- **Step Execution**: Enables step-by-step execution of code, allowing detailed observation of how changes in the state affect the running program.
- **Memory Inspection**: Provides tools to inspect and modify memory and registers, which is crucial for understanding how data and operations affect the program.

Disassemblers

Disassemblers are tools that translate machine code back into assembly code. They are similar to IDA Pro but are often less feature-rich and might not include debugging capabilities. Following are the key Features:

- **Static Analysis**: They work by analyzing executable files without executing them, which is safer when dealing with potentially malicious code.

- **Annotation Capabilities**: Most disassemblers allow the user to annotate the disassembled code, which is essential for keeping track of what various parts of the program are suspected to do.

3.2.1 Reverse Engineering Process

- **Loading the Binary**:

The binary is loaded into IDA Pro or another disassembler. IDA Pro parses the binary, identifies the processor architecture, and disassembles the binary to assembly code.

- **Analysis**:

The disassembled code is analyzed to understand what it does. This involves recognizing function calls, variables, API calls, control structures, etc.

- **Debugging**:

Using a debugger, the binary can then be run in a controlled environment where its behavior can be monitored. Debuggers help verify hypotheses made during the static analysis phase.

- **Modification and Reassembly**:

After understanding and possibly modifying the code, tools can reassemble the code back into a binary form, if necessary, for further testing or modification.

Table 3.1 provides an overview of the major contributions, principal features, and limitations of IDA Pro, debuggers, and disassemblers in the realm of reverse engineering. Each tool serves a distinct function in the reverse engineering workflow, encompassing tasks such as loading and disassembling the binary, debugging, and potentially altering the code for additional examination and testing.

Table 3.1 Comparisons Reverse Engineering Tools

Tools	Contribution	Key features	Limitations
IDA Pro	Converts binary programs into assembly code, crucial for analyzing proprietary or obfuscated code without source access.	Multi-processor Disassembly - Graphical Visualization - Debugging Capabilities - Extensibility	May have a steep learning curve; relatively high cost compared to some alternatives
Debuggers	Allow for dynamic analysis of programs by executing a program step-by-step to inspect variables and understand the flow of execution.	Breakpoints - Step Execution - Memory Inspection	Can require considerable expertise to use effectively; specific debuggers may have limitations on certain platforms

Disassemblers	Translate machine code back into assembly code for static analysis. Similar to IDA Pro but often less feature-rich and might not include debugging capabilities	Static Analysis - Annotation Capabilities	Limited in functionality compared to more integrated tools like IDA Pro; generally, no debugging capabilities.

The figure 3.1 depicts the reverse engineering process of a software system, which involves automated or manual analysis, information storage, design and implementation analysis, structural diagram extraction, and traceability matrices, enabling better maintenance and understanding.

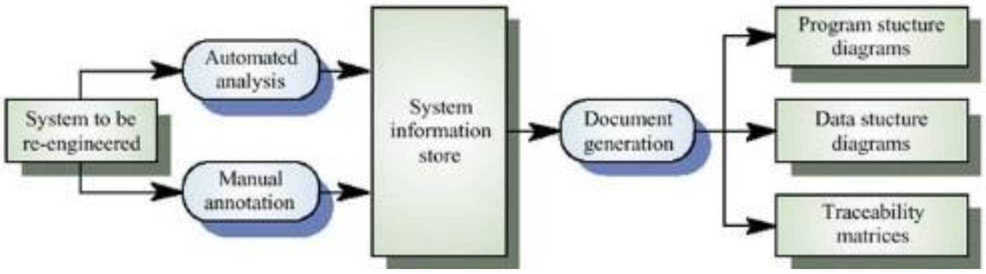
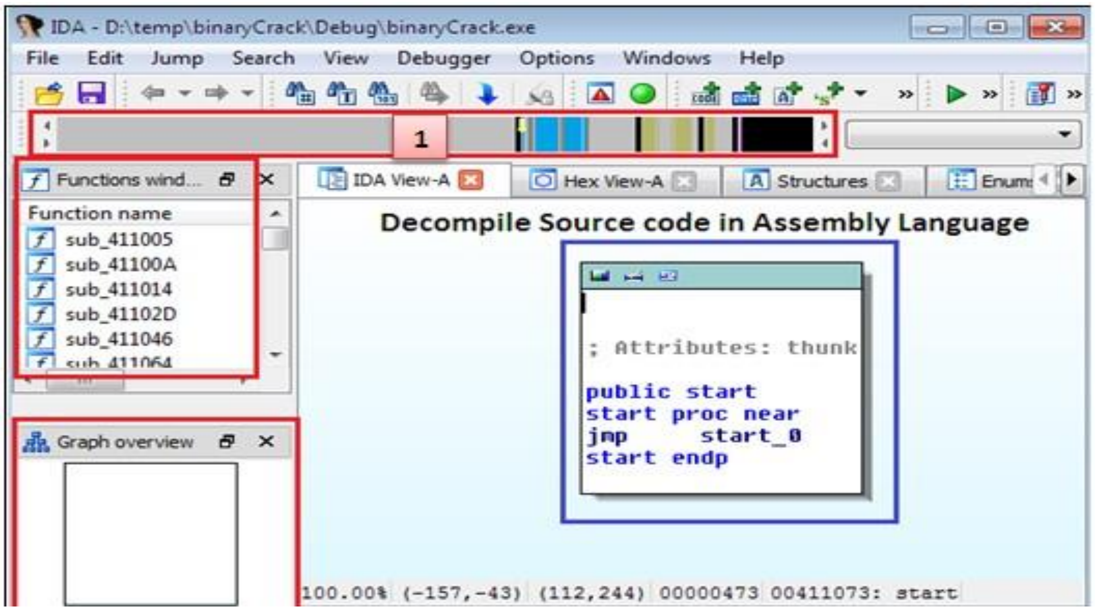


Figure 3.1 Reverse Engineering Process

The image shows in figure 3.2 a software reverse engineering decompiles tool interface, likely used by developers, security researchers, or malware analysts to analyze compiled binaries, executable files, or libraries, displaying decompiled code in assembly language format.



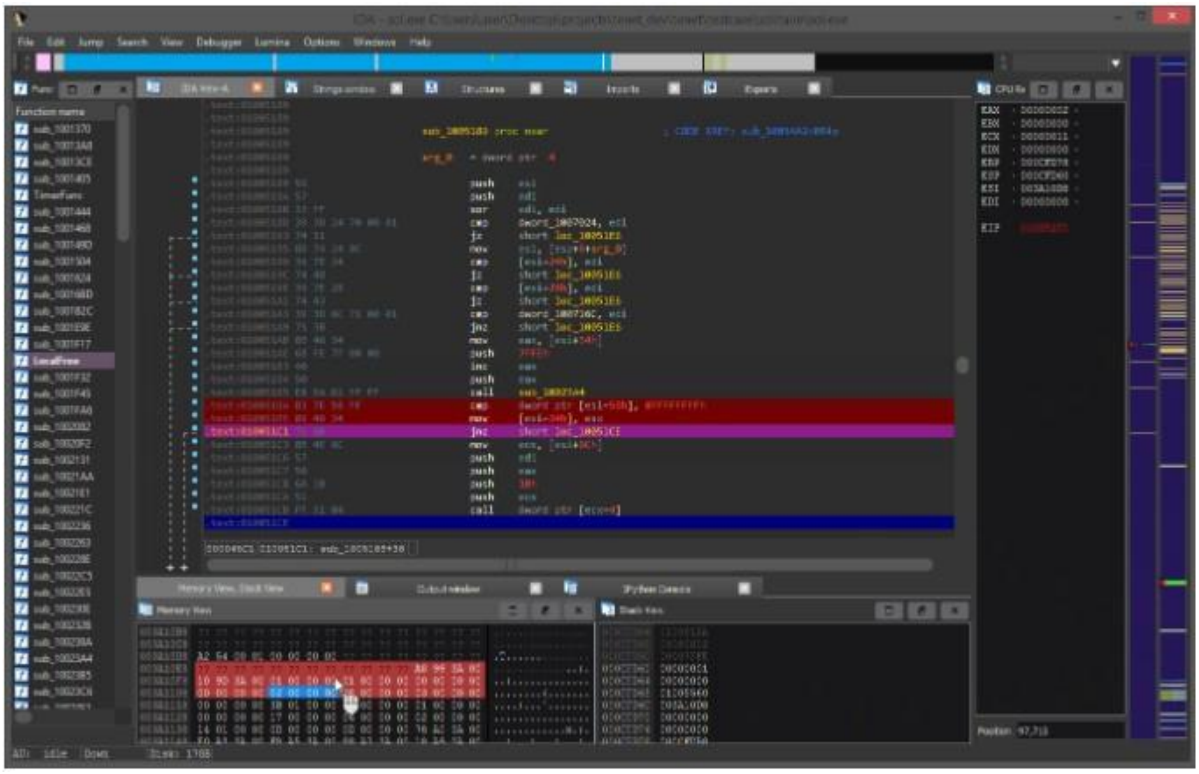
```
sub_411005
sub_41100A
sub_411014
sub_41102D
sub_411046
sub_411064
```

```
; Attributes: thunk

public start
start proc near
jmp     start_0
start endp
```

```
100.00% (-157,-43) (112,244) 00000473 00411073: start
```

Figure 3.2 Decompile Source Code

The figure 3.3, depicts a software analysis application, resembling an IDA Pro interface, used by developers, reverse engineers, and security researchers to examine code execution and understand VeraCrypt program structural components.

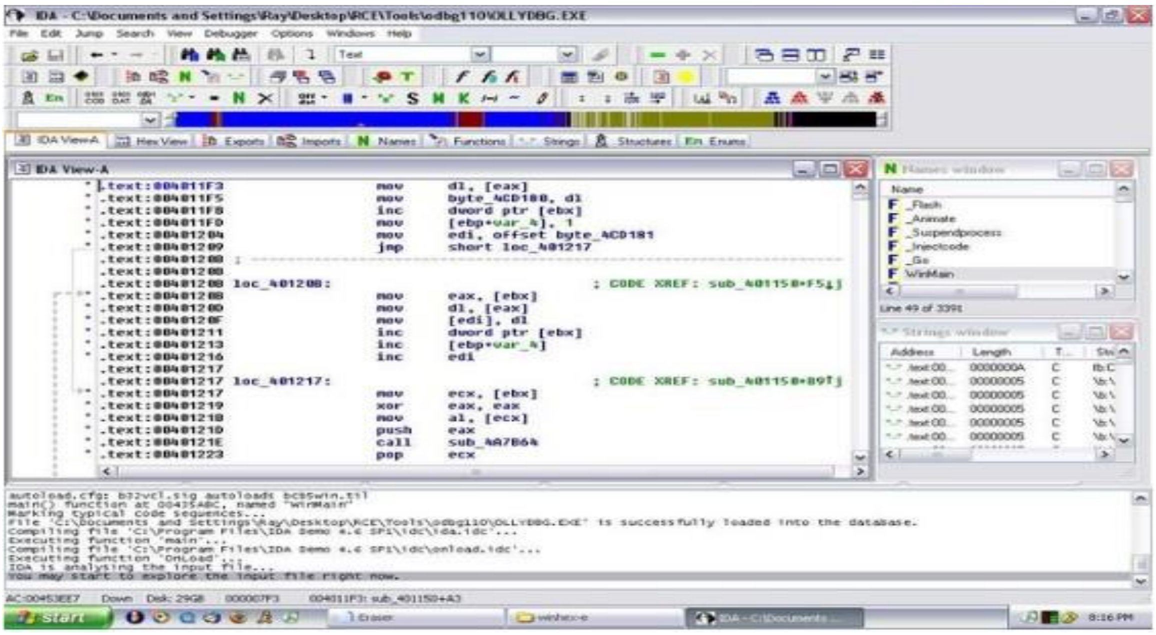


```
sub_10051B3 proc near                ; CODE XREF: [illegible]
arg_0  = dword ptr  4
       push    esi
       push    edi
       xor     edi, edi
       cmp     dword_1007024, edi
       jz      short loc_10051E5
       mov     esi, [esp+8+arg_0]
       cmp     [esi+[illegible]], edi
       jz      short loc_10051E5
       cmp     [esi+[illegible]], edi
       jz      short loc_10051E5
       cmp     dword_100716C, edi
       jnz     short loc_10051E5
       mov     eax, [esi+54h]
       push    [illegible]
       inc     eax
       push    eax
       call    sub_10027A4
       cmp     dword ptr [esi+58h], [illegible]
       mov     [esi+34h], eax
       jnz     short loc_10051CE
       mov     ecx, [esi+0Ch]
       push    edi
       push    eax
       push    30h
       push    ecx
       call    dword ptr [ecx+4]
```

Figure 3.3 Software Analysis Application

The Figure 3.4 Screen short, of IDA Pro disassembler and Hackman Disassembler are two tools with different capabilities. The disassembler performs minimal code analysis and has a similar interface to BORG's. Hackman Debugger has limited debugging capabilities, but can set breakpoints on predetermined events, display DOS and PE headers, and view binary executable program files.

The image displays in figure 3.5 software engineering tool employed by developers, security researchers, and malware analysts for code analysis. It features a left pane dedicated to code exploration, showcasing source code, instructions, and functions specific to each part of VeraCrypt Encryption software

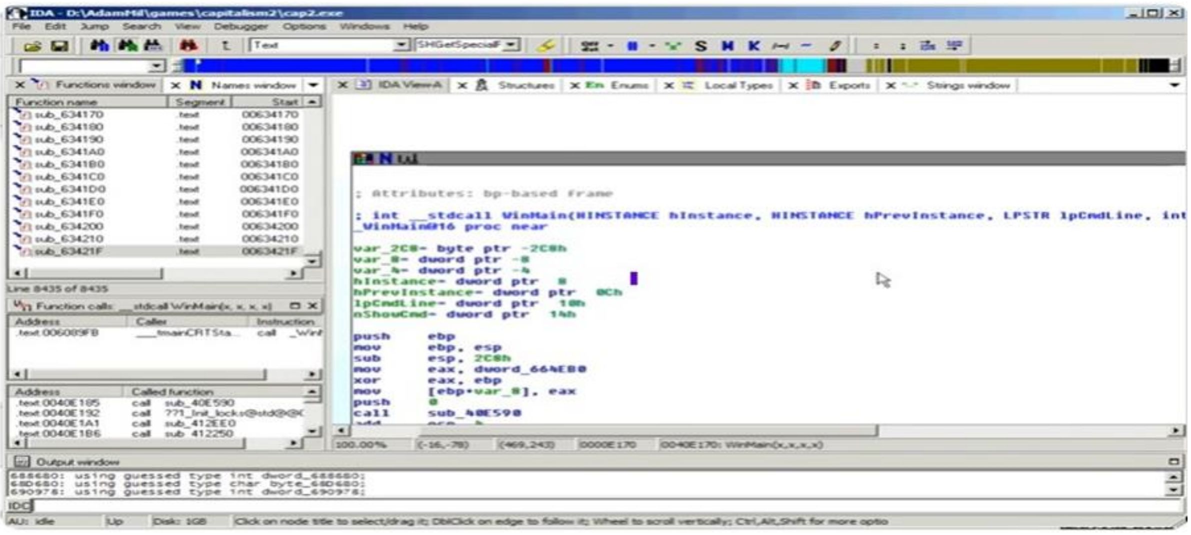

IDA - C:\Documents and Settings\Ray\Desktop\RCE\Tools\odbg110\OLLYDBG.EXE

```
.text:004011F3                 mov     dl, [eax]
.text:004011F5                 mov     byte_4CD180, dl
.text:004011FB                 inc     dword ptr [ebx]
.text:004011FD                 mov     [ebp+var_4], 1
.text:00401204                 mov     edi, offset byte_4CD181
.text:00401209                 jmp     short loc_401217
.text:0040120B ; ---------------------------------------------------------------
.text:0040120B
.text:0040120B loc_40120B:                             ; CODE XREF: sub_401150+F5↓j
.text:0040120B                 mov     eax, [ebx]
.text:0040120D                 mov     dl, [eax]
.text:0040120F                 mov     [edi], dl
.text:00401211                 inc     dword ptr [ebx]
.text:00401213                 inc     [ebp+var_4]
.text:00401216                 inc     edi
.text:00401217
.text:00401217 loc_401217:                             ; CODE XREF: sub_401150+B9↑j
.text:00401217                 mov     ecx, [ebx]
.text:00401219                 xor     eax, eax
.text:0040121B                 mov     al, [ecx]
.text:0040121D                 push    eax
.text:0040121E                 call    sub_4A7B64
.text:00401223                 pop     ecx
```

```
autoload.cfg: b32vcl.sig autoloads bcb5win.til
main() function at 00435ABC, named "WinMain"
Marking typical code sequences...
File 'C:\Documents and Settings\Ray\Desktop\RCE\Tools\odbg110\OLLYDBG.EXE' is successfully loaded into the database.
Compiling file 'C:\Program Files\IDA Demo 4.6 SP1\idc\ida.idc'...
Executing function 'main'...
Compiling file 'C:\Program Files\IDA Demo 4.6 SP1\idc\onload.idc'...
Executing function 'OnLoad'...
IDA is analysing the input file...
You may start to explore the input file right now.
```

AC:00453EE7 Down Disk: 29GB 000007F3 004011F3: sub_401150+A3

Figure 3.4 IDA Pro Disassembly


IDA - D:\AdamHil\games\capitalism2\cap2.exe

Function name	Segment	Start
sub_634170	.text	00634170
sub_634180	.text	00634180
sub_634190	.text	00634190
sub_6341A0	.text	006341A0
sub_6341B0	.text	006341B0
sub_6341C0	.text	006341C0
sub_6341D0	.text	006341D0
sub_6341E0	.text	006341E0
sub_6341F0	.text	006341F0
sub_634200	.text	00634200
sub_634210	.text	00634210
sub_63421F	.text	0063421F

Line 8435 of 8435

Function calls: __stdcall WinMain(x, x, x, x)

Address	Caller	Instruction
.text:006089FB	___tmainCRTSta...	call _Win

Address	Called function
.text:0040E185	call sub_40E590
.text:0040E192	call ??1_Init_locks@std@@(
.text:0040E1A1	call sub_412EE0
.text:0040E1B6	call sub_412250

```
; Attributes: bp-based frame

; int __stdcall WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int
_WinMain@16 proc near

var_2C8= byte ptr -2C8h
var_8= dword ptr -8
var_4= dword ptr -4
hInstance= dword ptr  8
hPrevInstance= dword ptr  0Ch
lpCmdLine= dword ptr  10h
nShowCmd= dword ptr  14h

push    ebp
mov     ebp, esp
sub     esp, 2C8h
mov     eax, dword_664EB0
xor     eax, ebp
mov     [ebp+var_8], eax
push    0
call    sub_40E590
```

100.00% (-16,-78) (469,243) 0000E170 0040E170: WinMain(x,x,x,x)

```
688680: using guessed type int dword_688680;
68D680: using guessed type char byte_68D680;
690978: using guessed type int dword_690978;
```

AU: idle Up Disk: 1GB Click on node title to select/drag it; DblClick on edge to follow it; Wheel to scroll vertically; Ctrl,Alt,Shift for more optio

Figure 3.5 IDA Pro Debugging Code Binary

3.3 Framework of Software Reverse Engineering

Software reverse engineering framework is a method of analyzing a software system to understand its design, functionality, and operation. It involves static, dynamic, static, and dynamic analysis, identifying functions, algorithms, and data structures, and addressing obfuscation and protection mechanisms. Documentation and reporting are crucial, but unauthorized reverse engineering may violate intellectual property laws.

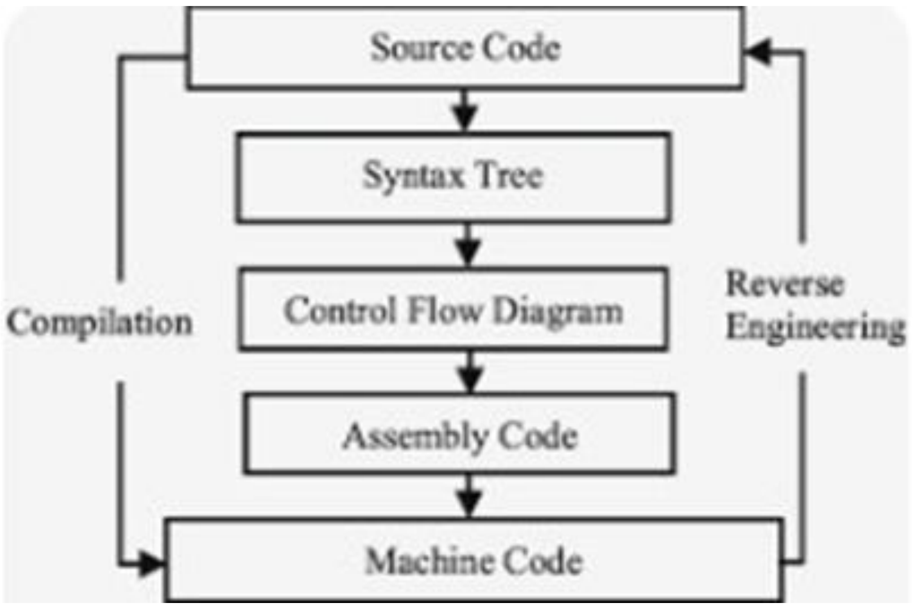


Figure 3.6 Flowchart of Reverse Engineering Process

This flowchart in figure 3.6 illustrates the compilation and reverse engineering process in software development, from source code parsing to machine code translation. It shows the stages of the process, from syntax tree to control flow diagram, and the relationship between abstraction or Components levels.

To initiate the reverse engineering process for an object, it's essential to start by examining and extracting information related to its design. This initial step involves a thorough analysis to understand how the various components of the object are structured and interconnected. In the context of software reverse engineering, this often requires acquiring the source code and any pertinent design documentation for in-depth analysis. Additionally, tools such as disassemblers may be employed to deconstruct the software into its constituent parts, facilitating a closer inspection of its inner working components.

3.3.1 System Structuring

The first stage of the process involves meticulously preparing the target system, VeraCrypt, for a thorough analysis. The use of IDA Pro, a powerful tool famous for its debugging and disassembly features, is used to do this. In order to offer a safe space where each software component can be meticulously scrutinized and severely evaluated, the VeraCrypt programmed is loaded into IDA Pro. The software's architecture, components, and execution flow can all be carefully examined thanks to this controlled environment. This rigorous and organized approach provides the framework for further investigations and makes it easier to have a thorough understanding of how the software performs internally.

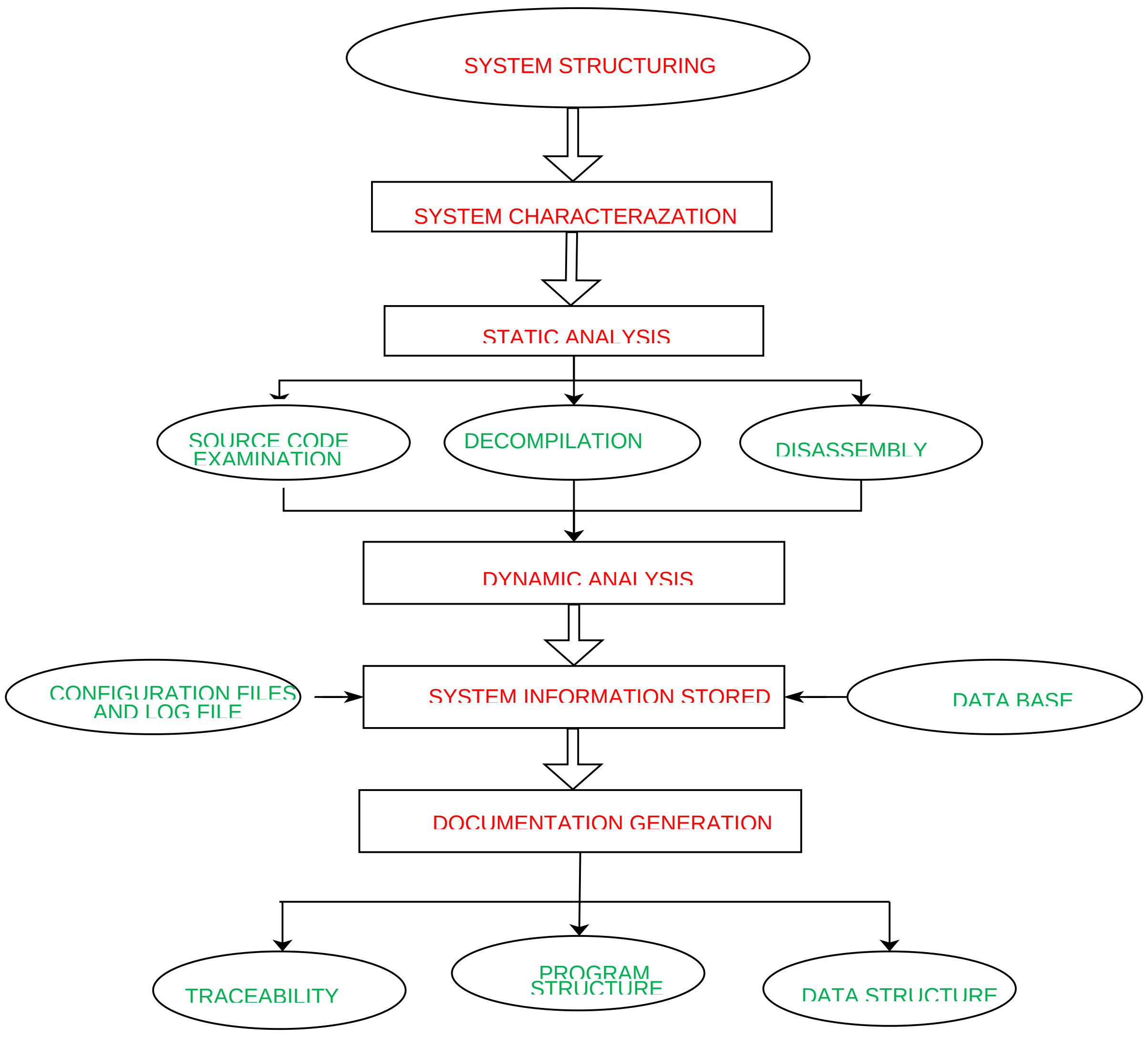


Figure 3.7 Proposed Software Reverse Engineering Framework

3.3.2 System characterization

Identifying the system's parts, such as modules, functions, and data structures, is the next stage in characterizing it. Finding the system's access points, exit points, and communication routes is also a part of this process. Using this data, a high-level diagram of the design and operation of the system is produced.

3.3.3 Static Analysis

In this stage, the system is systematically examined to find any potential weaknesses. The evaluation entails a close examination of the system's code in an isolated state. Finding typical security flaws such format string vulnerabilities, buffer overflows, and SQL injection point vulnerabilities is the main objective. In addition, this assessment includes the identification of dubious code patterns such as code obfuscation and anti-debugging methods. One of the most important techniques in reverse engineering is static analysis, which involves analyzing the software's binary code in great detail without running it. This method offers deep insights into the reasoning behind the code, its structure, and any possible weaknesses. The source code examination, decompilation, and disassembly phases within the static analysis process all share the common goal.

3.3.4 Source Code Examination

When the software's source code is available, the framework thoroughly examines it to learn more about the functionality and design of the programmed. This data is an important reference for the binary code's later static analysis.

3.3.5 De-Compilation

In cases where the source code is unavailable, the framework endeavors to decompile the binary code. This process aims to transform the binary code into a human-readable version, facilitating the identification of potential vulnerabilities and enhancing the understanding of how the software functions.

3.3.6 Disassembly

The framework breaks down the binary code when neither the source code nor the decompiled code is available. This procedure produces a low-level representation of the code, which is sometimes difficult to work with but is frequently necessary to recognize possible security flaws and understand how the product functions internally.

3.3.7 Dynamic Analysis

The framework subsequently does a dynamic analysis of the system when the static analysis is finished. In this phase, the system is run, and its behavior is monitored closely in real-time. A debugger is used to do this, which makes it possible to carefully review the code line by line. While the program is running, the debugger also makes it possible to examine register values and variable values. In addition, data flow within the system memory is traced using a memory analysis tool. For the purpose of locating potential vulnerabilities that would not have been discovered during the earlier static analysis phase, the insights obtained from this dynamic analysis are essential.

3.3.8 System Information Store Analysis

A storage device or database inside the VeraCrypt programmed is thoroughly explored and examined in the fifth step, System Information Store Analysis. The configuration, settings, and interactions between the software and the underlying system are all contained in this repository. Reverse engineers examine this repository to learn how the code communicates with hardware, utilizes system resources, and interacts with external dependencies. The purpose of this research is to comprehend the runtime environment of the software, system-level interactions, and potential weak points.

3.3.9 Configuration Files

The framework carefully examines the software's configuration files to learn more about its default settings and possible customization options. This data can be useful in spotting potential weaknesses like poor encryption keys or default passwords.

3.3.10 Log Files

To understand the functioning of the software and its interactions with users, the framework examines log files. This information can be valuable in identifying errors or unusual behaviors that may indicate potential vulnerabilities.

3.3.11 Databases

The framework delves into the software's databases to understand its mechanisms for storing and retrieving data. This information can be essential for identifying potential vulnerabilities, particularly unauthorized access to sensitive data. By analyzing these aspects during this step, the framework aims to uncover potential weaknesses in the software's configuration, usage, and data handling that may not have been apparent in earlier stages of analysis.

3.3.12 Document Generation

The creation of documentation is crucial for gathering and structuring the knowledge obtained throughout the reverse engineering project. The framework is tasked with generating a comprehensive report that consolidates the results, assessments, and observations from each preceding phase. This document is an invaluable resource that guarantees traceability and makes it possible for upcoming researchers or developers to understand the rationale behind choices made, flaws found, and enhancements suggested. Moreover, across the entire reverse engineering process, this documentation promotes efficient cooperation and knowledge transfer, enabling well-informed decision-making. It serves as a storehouse of important data, improving the general understanding and application of the learned material.

3.3.13 Utilization of Steps and Algorithms

Algorithm Function figure 3.1 outlines the mentioned steps in the sequence provided. The systematic and comprehensive reverse engineering of a specific software system, particularly VeraCrypt, can be accomplished by following the methodology expounded in this study. The framework offered is made up of a number of meticulously planned processes that gradually delve into the complex details of the design and functionality of the software. Additionally, this methodology is an invaluable resource for reverse engineering professionals who want to fully examine and comprehend intricate software systems like VeraCrypt. The framework provides researchers, developers, and cybersecurity specialists with a comprehensive collection of tools to identify vulnerabilities, advance understanding, and strengthen the general security and dependability of software systems by fusing a structured approach with a variety of analytical methodologies.

Table 3. 2 Shows the Steps and Function of Algorithms

Step	Description
Step 1: System Structuring	• Identify system components. • Determine entry and exit points • Identify communication channels
Step 2: System Characterization	• Identify system modules • Recognize system functions • Identify data structures in use
Step 3: Static Analysis	• Identify security vulnerabilities • Recognize suspicious code patterns
Step 4: Dynamic Analysis	• Debug the VeraCrypt file. • Track the flow of data during execution
Step 5: System Information Store Analysis	• Examine configuration files • Review log files for insights • Scrutinize databases used within the system
Step 6: Document Generation	• Generate comprehensive documentation • Present findings from the analysis processes

The methodology centers on comprehending the arrangement of a system through the extraction of its fundamental elements. Our objective is to pinpoint subsystems within the system and establish hierarchical frameworks to depict them. Additionally, we ascertain the exact interfaces connecting these subsystems. Employing this reverse engineering approach enables us to discern the structural characteristics of the system. Consequently, we can scrutinize the effects of local modifications more efficiently by assessing the resources exchanged between subsystems at various levels of granularity.

The following phases are involved in formulating a protocol for the examination of tools with a concentration on reverse engineering: This involves identifying the tools that will be examined and the specific aspects of reverse engineering that will be focused on. Identifying the tools and techniques to be used: Once the scope and objectives have been defined, the next step is to identify the tools and techniques that will be used to achieve the objectives.

This may involve using specialized software tools, manual techniques, or a combination of both. Understand the principles of software reverse engineering: Gain a comprehensive understanding of software reverse engineering techniques and tools commonly used to analyze and dissect software systems. This involves studying reverse engineering methodologies, code analysis, debugging, and disassembly techniques. Shown in table 3.2.

- Software reverse engineering involves dissecting a software system to learn how it functions and glean information from it. IDA Pro disassembler is one popular tool used for this purpose. The IDA Pro disassembler can be used with the following methods to undertake software reverse engineering on well-known disc encryption software.
- Obtain the disc encryption program's binary executable file, which needs to be reverse engineered. This can be accomplished by purchasing the programmed directly from the seller or by downloading it from the internet.
- Open the IDA Pro disassembler and load the binary executable file. IDA Pro is an effective programmed for examining binary files and can be used to convert binary data into assembly code.
- Following the binary analysis and creation of an interactive disassembly listing by IDA Pro, the file will be loaded. The binary code will be displayed in assembly language in this listing.
- Examine the disassembly listing to determine the software's organizational structure. Search the code for significant system calls, functions, and other key areas. Try to comprehend how the code works.
- Use the numerous IDA Pro analysis tools to help you better comprehend the code. These tools can be used to locate function calls, code structures, and other crucial data.
- Use the data from the analysis to find software flaws or to develop an updated version with more capabilities.

In addition, there are other tools available for software reverse engineering besides IDA Pro. For this, you may also utilize other programmed like Ghidra, and Hopper [185][186][187]. Furthermore, advancements in reverse engineering technology are anticipated in the future, Finally, the methods described above can be used to reverse engineer software using the IDA Pro disassembler. Alternative techniques and technologies that are out there for this purpose.

The initial step in the majority of reversing cases is to identify the application's component structure and the precise duties of each component. One then often chooses a module of interest and investigates the specifics of its execution from it. Reverse engineering is the process of disassembling anything and looking at its parts, functionalities, and code to analyses and understand how it works. It is frequently employed in the fields of electrical, mechanical, and software engineering. Reverse engineering processes can vary depending on the thing being examined, but essentially all consist of:

- **Gathering Information**: The initial phase involves collecting comprehensive details about the object under analysis. This encompasses obtaining documentation, user manuals, source code, hardware specifications, and any other pertinent information available.
- **Disassembly**: The next step is to take the object apart, either physically or virtually, depending on the type of object being analyzed. In software engineering, this can involve disassembling the compiled code to examine the machine instructions. In mechanical engineering, this can involve disassembling the parts of a machine to examine their structure and function.
- **Analysis**: Once the object is disassembled, the next step is to analyzed its components, functions, and code. This can involve examining the circuitry of a device, analyzing the algorithms used in software, or examining the materials and manufacturing techniques used in a mechanical object.
- **Reconstruction**: After analyzing the object, the next step is to reconstruct it, either physically or virtually. In software engineering, this can involve reverse engineering the code to create a new version with the same functionality. In mechanical engineering, this can involve designing new parts to replace damaged or worn components.
- **Testing**: Once the object is reconstructed, it must be tested to ensure that it works as expected. This can involve functional testing, stress testing, and other types of testing to ensure that the object functions correctly.
- **Documentation**: Finally, the results of the reverse engineering process should be documented in detail. This can include drawings, schematics, code listings, and other documentation to help others understand how the object works and how it was reverse-engineered.

Overall, the reverse engineering methodology is a complex process that requires a deep understanding of the object being analyzed as well as knowledge of various tools and techniques for disassembly, analysis, and reconstruction. It is often used to understand and improve existing products or to create new products that are similar to existing ones.

In this research, the methodology used a system-level reverse engineering technique. System-level reversing methods assist in figuring out the program's overall structure and occasionally even help identify key areas of interest. Using code-level reversing approaches, you can move on to more in-depth study once you have a rough understanding of the program's layout and have identified its areas of particular interest. In order to gather information, examine programmed executables, monitor programmed input and output, and other tasks, system-level reversing includes executing a number of tools on the programmed and accessing a number of operating system services. The operating system is where much of this information is sourced from. because the operating system is required to be involved in every contact a programmed has with the outside world by definition. Operating systems can be used during reversing sessions to gather a plethora of information about the target programmed under investigation, which is why reversers need to be familiar with them.

Software reverse engineering involves analyzing a software system to understand its inner workings and components details. Encapsulation is a key concept in software engineering that helps to keep software components isolated from each other, reducing the complexity of the system and making it easier to maintain and modify. The following sections describe various techniques for implementing encapsulation in software components at different levels of granularity.

The procedure is the most fundamental software unit in terms of code structure. A procedure is a section of code that may be called by other parts of the programmed and typically has a clear function. Optionally, procedures can take the caller's input data and return data to the caller. In any programming language, procedures are the type of encapsulation that is employed the most frequently. The division of a programmed into objects is the next logical step that replaces procedures. The method of building an object-based programmed differs greatly from the process of designing a standard procedure-based programmed. Many people believe that this method, known as object-oriented design (OOD), is the most widely used and successful method of software design currently accessible. An object is a software component that is related with both data and code, according to the OOD methodology. The code could be a group of instructions that relate to the object and have the ability to change its data. The data is a component of the object and is often private, which means that only object code and not outsiders can access it.

3.3 Modules

The module is a program's main building block. Modules are just binary files that contain discrete portions of an executable programmed (essentially the component boxes from our previous discussion). A programmed can be created by combining two different sorts of modules: Both static and dynamic libraries are used.

Static libraries: Static libraries are collections of source-code files that are assembled to represent specific programmed components. Static libraries logically reflect a feature or a section of programmed functionality. A static library is frequently an external, third-party library that enhances the functionality of the product being built rather than being an essential component of it. A program's static libraries are added as it is being constructed, and they are then included in the program's binary files. When we look at the programmed at a low level while reversing, they are hard to figure out and are so late.

3.4 Procedure of Encapsulation

Procedures can be encapsulated to give an equivalent amount of separation and abstract at a finer level of granularity. An independent piece of code called a process completes a given task and outputs the outcome. Procedures are encapsulated when their internals are hidden and a clear interface is provided for dealing with them. This interface outlines the procedure's inputs, outputs, and any prerequisites or requirements that must be fulfilled.

Processes can be contained using function pointers and opaque data types in C and C++. A function can be supplied as a parameter to other functions through the use of a function pointer, which is a variable that stores the address of the function. An opaque data type is a structure that is completely defined but whose underlying details are hidden from other system components.

Procedures can be implemented in Java and Python using classes and methods. A method is a function that is related to a specific class, and a class acts as a blueprint for creating objects. according to their visibility and access limitations, methods can be designated as public, private, or protected. Digital investigation involves collecting, analyzing, and preserving electronic evidence to respond to cyber incidents. It includes forensic imaging, network traffic analysis, timeline analysis, file and memory analysis, pattern recognition, and interpretation. Findings are presented, expert testimony provided, mitigation and remediation recommendations offered as shown in Figure 3.8.

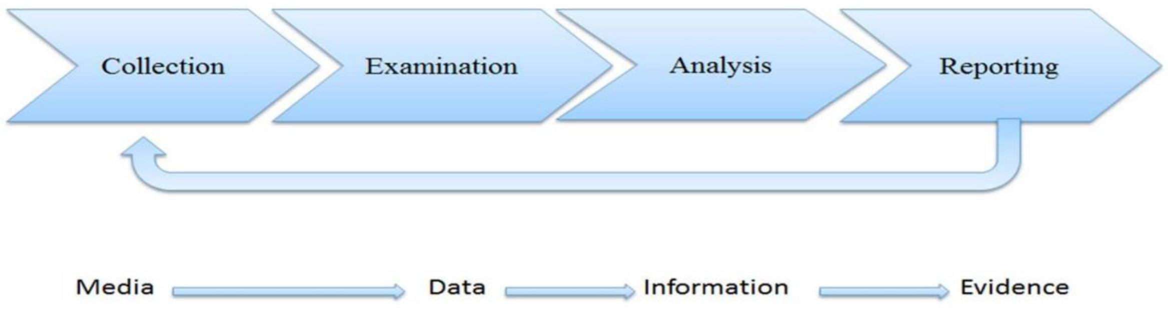


Figure 3.8 Process of Digital Investigation

A liberated open-source programmed called VeraCrypt is capable of encrypting whole drives or encrypted volumes. It operates, encrypts, and is available wonderfully, making it optional and reasonably acceptable [23]. Utilizing VeraCrypt, non-hidden encrypted volumes can be safely hidden. According to the Research, VeraCrypt contains a large number of modules. The summary of VeraCrypt's sensitive modules is shown in Table 3.2.

Table 3.3 VeraCrypt's Sensitive Modules

Sensitive Module	Description
Key derivation function (KDF)	Generates a strong and unique key based on the user's input password
Encryption and decryption modules	Encrypts and decrypts the data on the fly using algorithms like AES, Serpent, and Two fish
Random number generator	Generates random numbers used in the encryption process, using a combination of entropy sources
Hashing modules	Uses various hashing algorithms like SHA-256, SHA-512, and Whirlpool for integrity checking and password verification
Boot loader	Loads the VeraCrypt driver during the boot process to access encrypted data
Driver modules	Kernel-level driver modules intercept read and write requests to encrypted volume, ensuring data integrity

Containers for VeraCrypt data storage. VeraCrypt operates by encrypting and decrypting data in real-time within a designated VeraCrypt volume. These volumes appear and function to the user as regular hard drives or USB storage devices when accessed or mounted within the operating system. The user remains unaware of the automatic encryption or decryption process that occurs in the background. VeraCrypt offers support for various symmetric encryption techniques, allowing the user to choose the desired algorithm When creating a new storage space in VeraCrypt, You can view the list of algorithms that are supported in VeraCrypt 7.1.a. [8].

3.5 Summary of Chapter

The summary of this Chapter is to establish a comprehensive framework for the reverse engineering of software, with a specific focus on VeraCrypt. The methodology entails a systematic exploration of the software's architecture, functionality, and internal mechanisms, incorporating debugging tools, model creation, network analysis tools, and testing. The framework encompasses both static and dynamic analyses, aiming to identify functions, algorithms, and data structures. Documentation and reporting play a crucial role, as unauthorized reverse engineering may infringe upon intellectual property laws. The process involves examining system structure, conducting static analysis, scrutinizing source code, de-compiling, performing dynamic analysis, and analyzing log files to comprehend the system's operations and pinpoint potential vulnerabilities.

CHAPTER IV
RESULTS AND DISCUSSION

When analyzing software, engineers need to disassemble and sometimes decompile the software program. By converting the binary instructions into readable code, they can understand what the program does and how it affects different systems. This knowledge helps engineers develop solutions to mitigate the program's harmful effects. Reverse engineering (RE) uses various tools to determine the purpose and operation of a program. Through RE, engineers can identify the vulnerabilities the software exploits. Despite attempts by malware writers to hide their tracks, RE can extract information such as the program's production date, accessed resources, encryption keys, and file metadata. Engineers use different tools, including disassemblers, to break down the malware code into its components for analysis. This process helps in understanding how the program is designed and structured. Below is a description of one of the main tools used in malware reverse engineering:

Disassemblers (e.g., IDA Pro): These tools convert the program into assembly code, which provides a detailed view of the program's instructions. Decompiles can also be used to convert binary code into native code, although they may not be available for all architectures.

Debuggers (e.g., x64dbg Windbg, GDB): Debuggers allow reversers to modify the program's execution and observe its behavior while it is running. They also provide control over the program's memory, helping engineers analyze its impact on a system or network.

4.1 Experimental Process

In the experimental evaluation of the proposed framework, the implementation was carried out in C++ using the Visual Studio platform. The system configuration used for the experiments is detailed below:

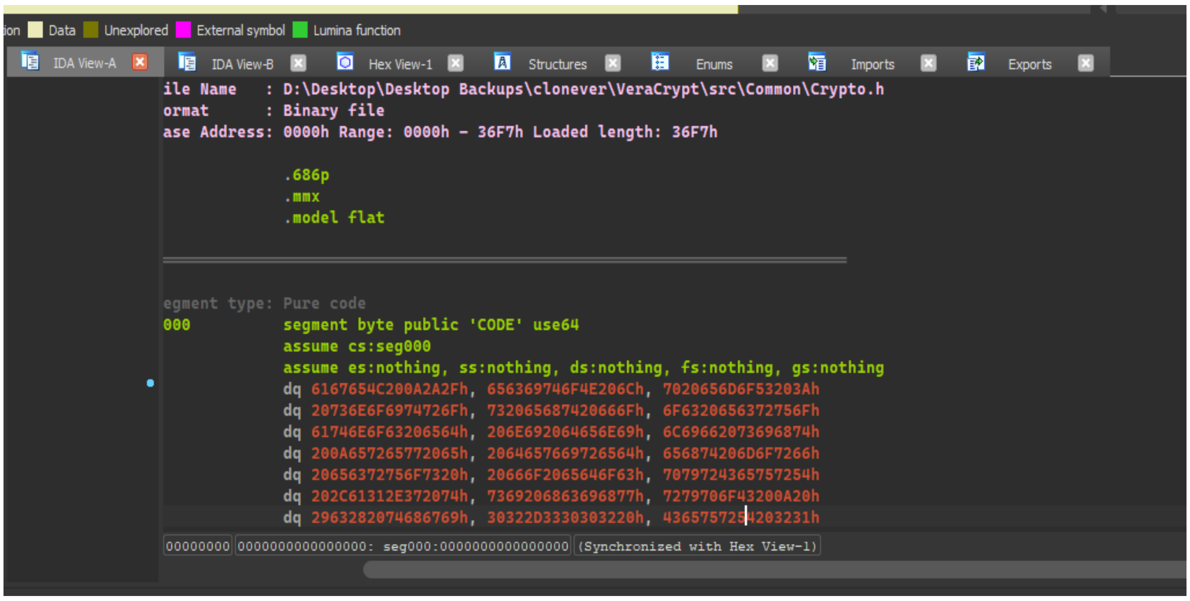
The system configuration used for this research is as follows:

- Processor: Intel Core i5 3217U
- Processor Frequency: 1.8 GHz
- Memory (RAM): 8 GB
- Operating System: Windows 10 Home Edition

IDA Pro is considered the primary tool of choice during the system structuring phase. Reverse engineers hold IDA Pro in high regard because of its sophisticated disassembly capabilities and wide range of features, which significantly facilitate understanding and analyzing complex binary code [40].

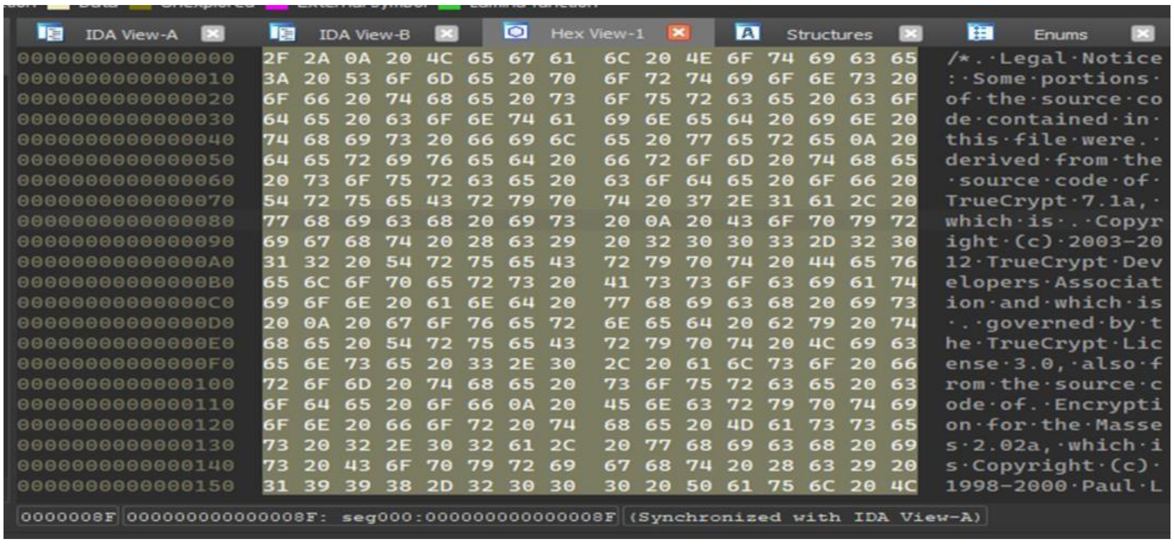
4.2 Snapshots of IDA View and Hex View

- **IDA View**: This view provides a comprehensive disassembly of the binary code, enabling in-depth analysis of the program's structure and functionality as shown in Figure 4.1.
- **Hex View**: The Hex View offers a hexadecimal representation of the binary, allowing for a detailed examination of the binary data as shown in Figure 4.2 and others. Snapshots of the results are in Appendix A.



```
ile Name    : D:\Desktop\Desktop Backups\clonever\VeraCrypt\src\Common\Crypto.h
ormat       : Binary file
ase Address: 0000h Range: 0000h - 36F7h Loaded length: 36F7h

                .686p
                .mmx
                .model flat


egment type: Pure code
000             segment byte public 'CODE' use64
                assume cs:seg000
                assume es:nothing, ss:nothing, ds:nothing, fs:nothing, gs:nothing
                dq 6167654C200A2A2Fh, 656369746F4E206Ch, 7020656D6F53203Ah
                dq 20736E6F6974726Fh, 732065687420666Fh, 6F6320656372756Fh
                dq 61746E6F63206564h, 206E692064656E69h, 6C69662073696874h
                dq 200A657265772065h, 2064657669726564h, 656874206D6F7266h
                dq 20656372756F7320h, 20666F2065646F63h, 7079724365757254h
                dq 202C61312E372074h, 7369206863696877h, 7279706F43200A20h
                dq 2963282074686769h, 30322D3330303220h, 4365757254203231h
00000000 0000000000000000: seg000:0000000000000000 (Synchronized with Hex View-1)
```

Figure 4.1 IDA View of VeraCrypt



```
0000000000000000 2F 2A 0A 20 4C 65 67 61  6C 20 4E 6F 74 69 63 65 /*.·Legal·Notice
0000000000000010 3A 20 53 6F 6D 65 20 70  6F 72 74 69 6F 6E 73 20 :·Some·portions·
0000000000000020 6F 66 20 74 68 65 20 73  6F 75 72 63 65 20 63 6F of·the·source·co
0000000000000030 64 65 20 63 6F 6E 74 61  69 6E 65 64 20 69 6E 20 de·contained·in·
0000000000000040 74 68 69 73 20 66 69 6C  65 20 77 65 72 65 0A 20 this·file·were.·
0000000000000050 64 65 72 69 76 65 64 20  66 72 6F 6D 20 74 68 65 derived·from·the
0000000000000060 20 73 6F 75 72 63 65 20  63 6F 64 65 20 6F 66 20 ·source·code·of·
0000000000000070 54 72 75 65 43 72 79 70  74 20 37 2E 31 61 2C 20 TrueCrypt·7.1a,·
0000000000000080 77 68 69 63 68 20 69 73  20 0A 20 43 6F 70 79 72 which·is·.·Copyr
0000000000000090 69 67 68 74 20 28 63 29  20 32 30 30 33 2D 32 30 ight·(c)·2003-20
00000000000000A0 31 32 20 54 72 75 65 43  72 79 70 74 20 44 65 76 12·TrueCrypt·Dev
00000000000000B0 65 6C 6F 70 65 72 73 20  41 73 73 6F 63 69 61 74 elopers·Associat
00000000000000C0 69 6F 6E 20 61 6E 64 20  77 68 69 63 68 20 69 73 ion·and·which·is
00000000000000D0 20 0A 20 67 6F 76 65 72  6E 65 64 20 62 79 20 74 ·.·governed·by·t
00000000000000E0 68 65 20 54 72 75 65 43  72 79 70 74 20 4C 69 63 he·TrueCrypt·Lic
00000000000000F0 65 6E 73 65 20 33 2E 30  2C 20 61 6C 73 6F 20 66 ense·3.0,·also·f
0000000000000100 72 6F 6D 20 74 68 65 20  73 6F 75 72 63 65 20 63 rom·the·source·c
0000000000000110 6F 64 65 20 6F 66 0A 20  45 6E 63 72 79 70 74 69 ode·of.·Encrypti
0000000000000120 6F 6E 20 66 6F 72 20 74  68 65 20 4D 61 73 73 65 on·for·the·Masse
0000000000000130 73 20 32 2E 30 32 61 2C  20 77 68 69 63 68 20 69 s·2.02a,·which·i
0000000000000140 73 20 43 6F 70 79 72 69  67 68 74 20 28 63 29 20 s·Copyright·(c)·
0000000000000150 31 39 39 38 2D 32 30 30  30 20 50 61 75 6C 20 4C 1998-2000·Paul·L
0000008F 000000000000008F: seg000:000000000000008F (Synchronized with IDA View-A)
```

Figure 4.2 Hex View of VeraCrypt

The utilization of IDA Pro, as demonstrated in the experiment, plays a critical role in the systematic structuring of the system for further analysis. This tool's advanced capabilities aid researchers in dissecting and comprehending the intricacies of binary code, which is foundational in the proposed framework's experimental assessment.

4.3 Experimental Evaluation

After completing the system structuring phase, the subsequent steps of the framework were executed, and an assessment of the framework's performance was conducted. This assessment focused on two key aspects: accuracy and time consumption.

Accuracy, within this context, is defined as the percentage of modules and functions successfully identified within the architecture of VeraCrypt during the execution of the framework. This metric provides insights into how effectively the framework can analyze and understand the software's components and functionality.

4.3.1 Accuracy Evaluation

The accuracy of the framework was assessed by varying the file size of VeraCrypt, ranging from 1000 to 5000, and the results (as depicted in Fig. 4.3) demonstrate a consistent increase in accuracy as the file size expands. This pattern can be attributed to the framework's access to a greater volume of data to investigate and analyze. Consequently, as the file size becomes larger, the framework's accuracy also proportionally improves. In simpler terms, a larger file size offers the framework more information to work with, leading to a more precise identification of modules and functions within VeraCrypt.

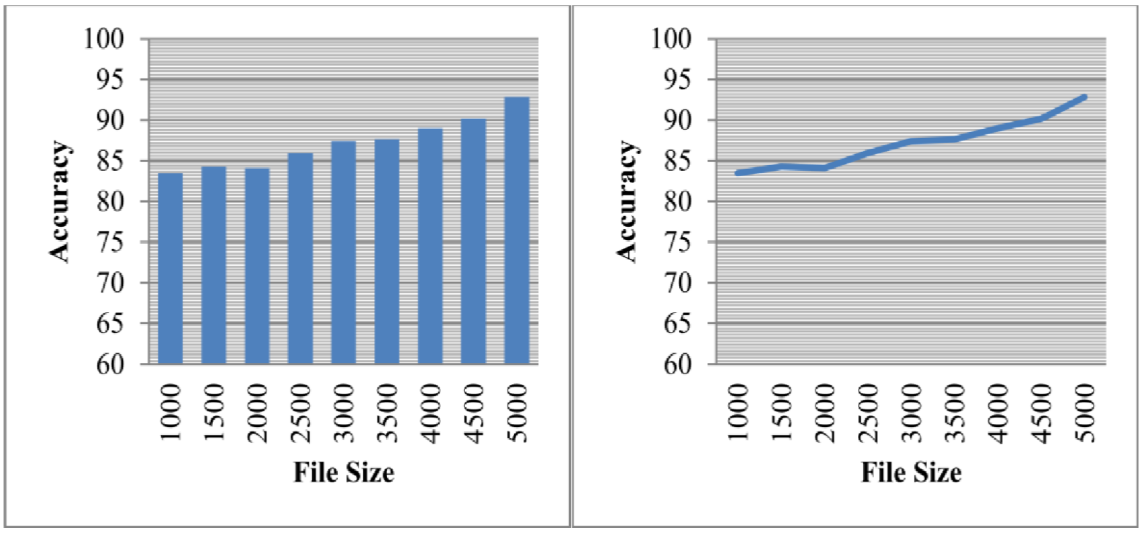


Figure 4.3 Accuracy vs Time

4.3.2 Time Consumption Evaluation

Along with evaluating the accuracy, the framework's evaluation of the structured VeraCrypt as it was handled by IDA also measured the time it took to complete. The results show that time consumption maintains as file size grows (Figure 4.4). This phenomenon can be attributed to the fact that as a file size increases, the framework gets more capable of handling instances that are similar. As a result, this adaptation quickens the process and provides a time consumption that is more consistent and predictable.

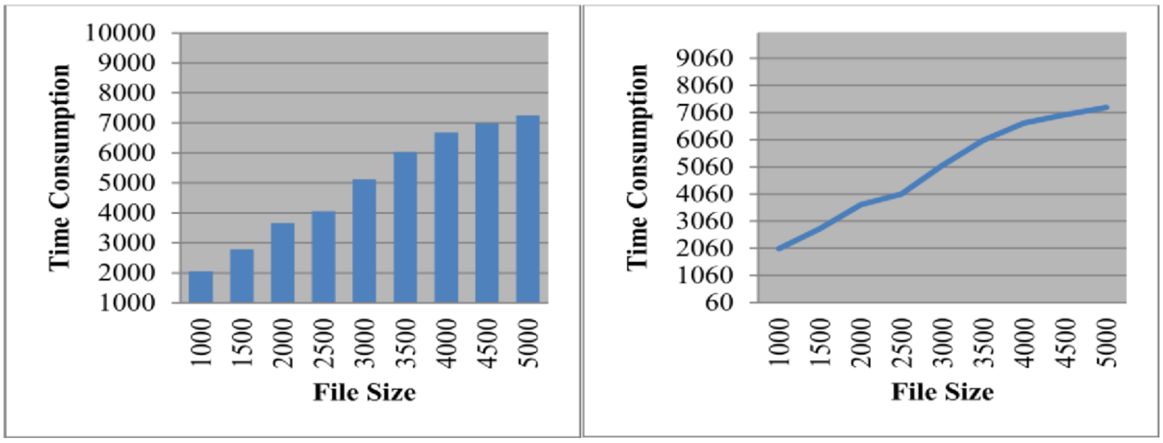


Figure 4.4 Consumption Time vs File Size

To delve further into the findings, the experimental evaluation of the framework highlights a notable correlation between increasing file sizes and improved accuracy. As the size of files grows, the framework exhibits a capacity to handle the augmented complexity of VeraCrypt files while maintaining a consistent level of accuracy. Notably, the stabilization of time consumption underlines the framework's ability to efficiently process larger files without sacrificing its effectiveness in terms of both accuracy and time efficiency.

The identified trend indicates that the framework is resilient and adaptable, demonstrating its scalability when dealing with more extensive datasets. The favorable correlation between increased file size and enhanced accuracy is promising for real-world applications, affirming the framework's sustained reliability and effectiveness as it addresses the growing complexity of the scrutinized VeraCrypt files. The consistent time consumption stabilization further emphasizes the efficiency of the framework, validating its capacity to handle complex tasks with larger files without compromising the overall efficacy of the analysis.

Through extensive experiments, the performance of the framework was rigorously evaluated. The results showed that as the size of the files being analyzed increased, the accuracy of the framework also increased proportionally. Furthermore, As VeraCrypt's file size expanded, the framework constantly maintained a stable level of time consumption, demonstrating its effectiveness in handling more extensive analysis. In conclusion, the suggested framework offers a thorough and practical method for assessing anti-forensic encryption solutions, utilizing VeraCrypt as a case study. This framework offers a viable response to the developing problems caused by anti-forensic approaches by utilizing the powers of software reverse engineering and sophisticated disassembly tools like IDA Pro. The representation of source code flow is shown in Table 4.1.

Table 4.1 Representation of the VeraCrypt Source Code Flow

Step	Description
Initialization	Initialize variables, libraries, and data structures.
User Interface	Handle user input and trigger corresponding actions.
Volume Management	Create, mount, and manage encrypted volumes.
Encryption/Decryption	Perform encryption and decryption operations on data using cryptographic algorithms.
File System Integration	Integrate with the underlying file system for transparent encryption and decryption of files.
Key Management	Generate, store, and retrieve encryption keys.
Error Handling and Logging	Handle errors, provide feedback to the user, and log important events.

It's important to emphasize that the information presented in this table offers a condensed overview of the VeraCrypt source code flow. The practical implementation, however, may encompass a more intricate landscape involving detailed components and interactions.

4.4 Validation Discussion

The framework's accuracy and time efficiency are validated in the Accuracy Evaluation and Time Consumption Evaluation sections. The framework's accuracy improves consistently with larger file sizes, attributed to its access to more data. This enables it to make more precise identifications. The framework's consistent time consumption despite increasing file sizes is impressive, demonstrating its ability to handle larger instances efficiently. This scalability is particularly useful when dealing with large and complex VeraCrypt volumes in real-world scenarios. The findings are promising for practical applications, demonstrating the framework's scalability and robustness, making it a valuable asset for real-world VeraCrypt analysis tasks.

4.4 Description of Data Storage

This section presents a general description of a characteristic solid driver storage media device. It highlights that the way data is perceived by the operating system differs from how it is physically stored on the storage device. In other words, the operating system's representation of data is abstracted from the actual physical storage mechanism employed by the device.

4.4.1 Hard Drive Storage System

In a typical hard disk drive (HDD), multiple rounded platters are stacked together around a central spindle. These platters have a magnetic coating on their surfaces. A movable arm with a head attached to it is used to read and write magnetic information on the platters. Before the HDD is used, it undergoes a low-level plant format process that produces concentric circles known as tracks on the platter surfaces. Individually track is divided into smaller units named sectors, with a typical sector having the capacity to store 512 bytes of data.

In the past, a sector on a hard disk drive was identified using a grouping of its track address (cylinder address "C"), the head address "H" indicating the reading head, and the sector address "S" indicating its location within a track. This addressing scheme was referred to as CHS (Cylinder-Head-Sector) address. However, as hard disk drive capacities increased beyond the limitations imposed by older BIOS systems and the original ATA interface specification, new addressing schemes emerged and replaced the CHS method.

Currently, the most recent published standard is ATA/ATAPI 8, which utilizes a logical block addressing scheme. In this scheme, each sector is sequentially allocated a unique 48-bit number, providing a more efficient and scalable approach to addressing sectors on modern hard drives.

Unlike hard disk drives that rely on the ATA controller and are subject to size restrictions imposed by machine BIOS, SCSI (Small Computer Systems Interface) drives have not encountered the same limitations. SCSI drives also use a method called logical block addressing, which can be either 32-bit or 64-bit, to access and locate sectors on the hard disk drive. This allows for larger storage capacities and greater flexibility compared to ATA-based drives.

4.4.2 Volumes Capacity

Volume refers to a group of sectors used for storing data by an operating system or application. It can include all sectors on a single storage device or multiple devices combined in a RAID setup. RAID volumes offer various levels of fault tolerance, performance, and scalability. During forensic investigations, it is crucial to capture the entire volume, even if it spans across multiple disks. Failing to do so can result in incomplete analysis and missing data. In the context of the thesis, only volumes within a single storage device were considered to focus on detecting TrueCrypt data rather than reconstructing the underlying volume[188], [189].

4.4.3 Partitions of Volume Capacity

A partition refers to a section of a volume that consists of consecutive sectors. It can either cover the entire volume or a smaller portion of it. Partitioning a volume serves various purposes. Firstly, different partitions can contain separate file systems, which determine how data is organized, stowed, and accessed, including factors like folder size, access controls, and naming conventions. Additionally, partitions simplify the process of disaster recovery by separating application data from user data. Users can easily back up their files by focusing on the specific partition that holds their data, instead of going through multiple application files and folders, it is more efficient to search within a partition. Operating systems can assign different partitions for specific purposes, one common usage of partitions is to use one for storing swap memory pages on disk as part of managing virtual memory. To make a partition ready for an operating system to store and retrieve data, it needs to be prepared by using a software application that is launched from bootable media. This application sets up the necessary data structures to establish the file system, a process often known as formatting [188].

4.4.4 System File Storage

A file system is a structured method of storing and retrieving data on a storage device. It organizes data into files and allows for the organization of multiple files into folders or directories. The file system plays a crucial role in determining how data is stored and accessed by a computer, including important information known as metadata. This metadata includes details like the file name, access permissions (who can access the file), file size, location within the storage device, timestamps indicating when the file was last accessed or created, and other essential information needed for efficient data management.

4.5 Components Dedicated to Storage

Files are assigned space within a file system using a specific unit of allocation, which can vary depending on the file system being used. For instance, in the FAT file system, this unit is known as a cluster, while in the Ext file system, it is referred to as a block. In a given file system, all allocation units have the same size and are determined by a multiple of the sector size. To illustrate this concept, let's consider an example where the sector size is 512 bytes and the desired number of sectors to allocate is 8. In this scenario, the allocation unit of the file system would be 8 multiplied by 512 bytes, resulting in a size of 4,096 bytes. Consequently, the minimum file size in this file system would be 4 kilobytes (KB).

In certain file systems, you can choose the allocation unit size when formatting the storage device. It is crucial to select the right size to avoid wasting disk space. If the unit size is too large, it may lead to inefficient use of disk space. Conversely, if the unit size is too small, managing multiple units per file can create additional overhead and potentially cause performance problems. The unallocated space within the partition, along with the allocation units that are not currently assigned to files, is collectively known as unallocated space [30].

Files in a file system are stored in specific storage areas called allocation units, which have different names depending on the file system used, such as clusters in FAT or blocks in Ext. Each allocation unit in the file system has the same size, determined by multiplying the sector size. For example, if the sector size is 512 bytes and the allocation unit is set to 8 sectors, each allocation unit would be 8 x 512 bytes, resulting in a size of 4,096 bytes. Therefore, the minimum file size in this file system would be 4 kilobytes (KB). Generally, the allocation unit size can be specified during the formatting process of the file system.

It is crucial to select an appropriate allocation unit size. If the allocation unit size is too large, it may lead to inefficient usage of diskette space. Conversely, if the division component size is too insignificant, managing multiple division units per file becomes more complex and can result in reduced file system performance. The unallocated space within the partition, including the allocation units not currently assigned to any file, is collectively referred to as unallocated space [190]. The unallocated space within a file system may still comprehend data from files that have been cancelled but not yet overwritten. The possibility of retrieving such data depends on factors such as the level of activity within the file system and the specific allocation and unallocated space reuse strategies employed by the file system.

If the file system is frequently used and experiences a high level of activity, there is a greater chance that the unallocated space will be quickly reused for new files, reducing the likelihood of recovering deleted data. However, if the file system is less busy and remains relatively unchanged over time, there is a higher possibility of recovering deleted data from the unallocated space.

The effectiveness of data retrieval also depends on the file system's allocation and unallocated space reuse strategies. These strategies determine how quickly the file system reuses unallocated space for new data. Some file systems may prioritize immediate reuse of unallocated space to optimize storage efficiency, while others may employ strategies that delay reuse, increasing the chances of data recovery from deleted files.

It is important to note that data recovery from unallocated space is not guaranteed and relies on various factors. To increase the chances of successful data recovery, specialized forensic techniques and tools may be employed to analyses the file system and retrieve deleted data from the unallocated space.

4.5.1 Slack Space

Inefficient use of disk space can be caused by two primary factors: small files not fully utilizing a single allocation unit and large files leaving unused space within the last allocation unit. This inefficiency arises when the file scope does not align precisely with the division unit size, leading to unused space that cannot be utilized by other files within the allocation unit. This unused portion is known as file slack space. Consequently, disk space is wasted, reducing the overall efficiency of storage.

Slack space can also exist at the volume level and within individual partitions. This means that not only can file slack space occur within a partition, but there can also be unused space within the entire volume. Managing and minimizing slack space is important for optimizing disk utilization and maximizing available storage capacity [69]. Volume slack space refers to sectors in a volume that are not assigned to any specific partition. Similarly, partition slack space refers to the unused space within a partition when its size is not a perfect multiple of the sector size Both volume and partition slack spaces are significant areas to examine because they might contain leftover data from previous information stored on the disk. As these spaces are not allocated by the file system, any hidden data within them remains unaffected by normal operating system actions and could potentially be retrieved during forensic analysis [191]. Although VeraCrypt does not utilize slack space for storing its multiple storage containers, the analysis of slack space remains relevant in a forensic context. Examining slack space can provide valuable insights, as the presence of unallocated space between VeraCrypt volumes may be observed. This observation can be significant for statistical analysis and can contribute to the overall investigation process. Therefore, considering slack space in relation to VeraCrypt volumes can still be a valid approach in forensic analysis.

4.6 VeraCrypt Data Storage Containers

VeraCrypt employs real-time encoding and decoding of data stored within its volumes. When a VeraCrypt volume is opened or mounted, it appears and functions like a regular solid disk or USB storing device to the user. The encryption and decryption processes occur seamlessly and are transparent to the user. VeraCrypt offers several symmetric encryption algorithms, allowing users to choose their preferred algorithm when producing a new capacity. The maintained algorithms for VeraCrypt 7.1.a can be found in Table 4.2 , which has been sourced from the VeraCrypt documentation [192].

Table 4.2 Symmetric Encryption Algorithms Supported by VeraCrypt 7.1a

Symmetric Encryption Algorithms	**Key size (bits)**	**Block size (bits)**	**Mode of operation**
AES	256	128	XTS
Serpent	256	128	XTS
Twofish	256	128	XTS
AES+Twofish	[256] + [256]	128	XTS
AES+Twofish+Serpent	[256]+[256]+[256]	128	XTS
Serpent+AES	[256] +[256]	128	XTS
Serpent+Twofish+AES	[256]+[256]+[256]	128	XTS
Twofish+Serpent	[256]+[256]	128	XTS

VeraCrypt uses the term "VeraCrypt volume" to refer to the encrypted data that can be accessed through VeraCrypt. This differs from the previous discussion where the term "volume" referred to the storing medium. To prevent misunderstanding, we will use the term "VeraCrypt volume" when discussing the encrypted data available through VeraCrypt, while the term "volume" will continue to denote a storing capacity as well-defined in the earlier segment.

4.7 Encrypted VeraCrypt Volumes

VeraCrypt provides two types of encrypted storage options. The first type is called coded file containers, which are encrypted files that can be seen within the file system. You can think of them as regular files that contain encrypted data. The structure of an encrypted file container is depicted in Figure 4.5.

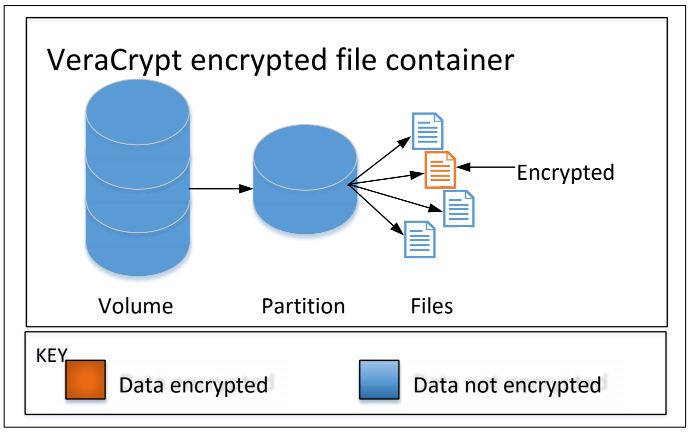


Figure 4.5 VeraCrypt Coded Folder Container

The second type of VeraCrypt volumes Figure 4.6 involves the encryption of either an entire partition or a device, such as a hard disk partition or the complete volume of a storage device. The corresponding figure illustrates this type of volume. While in prior literature, both types are commonly denoted as "outer" volumes, this thesis adopts the term "standard VeraCrypt volume" to specifically characterize this category of encrypted volumes.

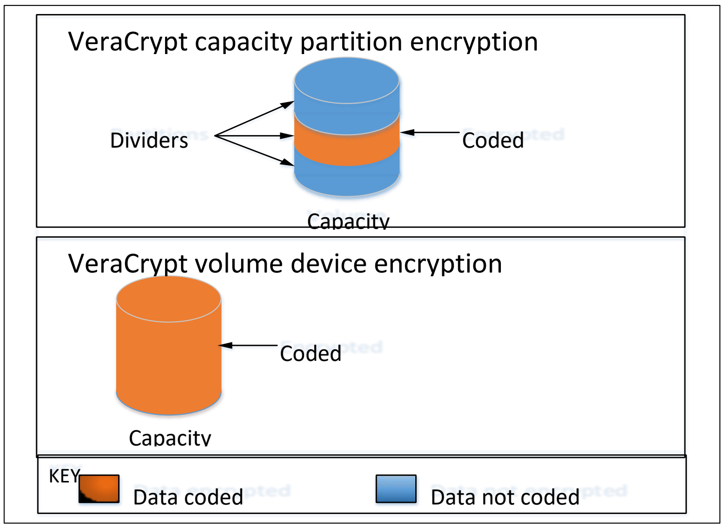

VeraCrypt capacity partition encryption
Dividers
Coded
Capacity
VeraCrypt volume device encryption
Coded
Capacity
KEY
Data coded
Data not coded

Figure 4.6 VeraCrypt Divider and Device Encryption

VeraCrypt categorizes volumes in the second category depending on whether they contain operating system start-up files. Volumes that include these files are called system partitions or system devices. The aim of this thesis is to identify volumes at the partition or device level, which may include system partitions or devices. Furthermore, VeraCrypt provides the option to create hidden volumes and hidden operating systems. These encrypted volumes are hidden within the regular VeraCrypt volume and can be present in any of the aforementioned categories.

4.7.1 VeraCrypt Hidden Volume

Each of the mentioned programs has the potential to include a hidden encrypted VeraCrypt volume, referred to as a hidden volume. This hidden volume is located underground the unallocated free space of a standard VeraCrypt volume, which is not allocated to files. This concept is described in the VeraCrypt documentation [193]. The article underscores the importance of showcasing a diagram that depicts the structure of a typical VeraCrypt volume alongside a concealed volume. Figure 4.3. derived from the article, offers this illustration. It emphasizes that while utilizing the regular VeraCrypt volume, it is purposely challenging to discern the presence of a hidden volume within it [192]. The volume assemblies depicted in Figure 4.5 can be present within the regions labelled as "Data encrypted" in both Figure 4.6 and Figure 4.7.

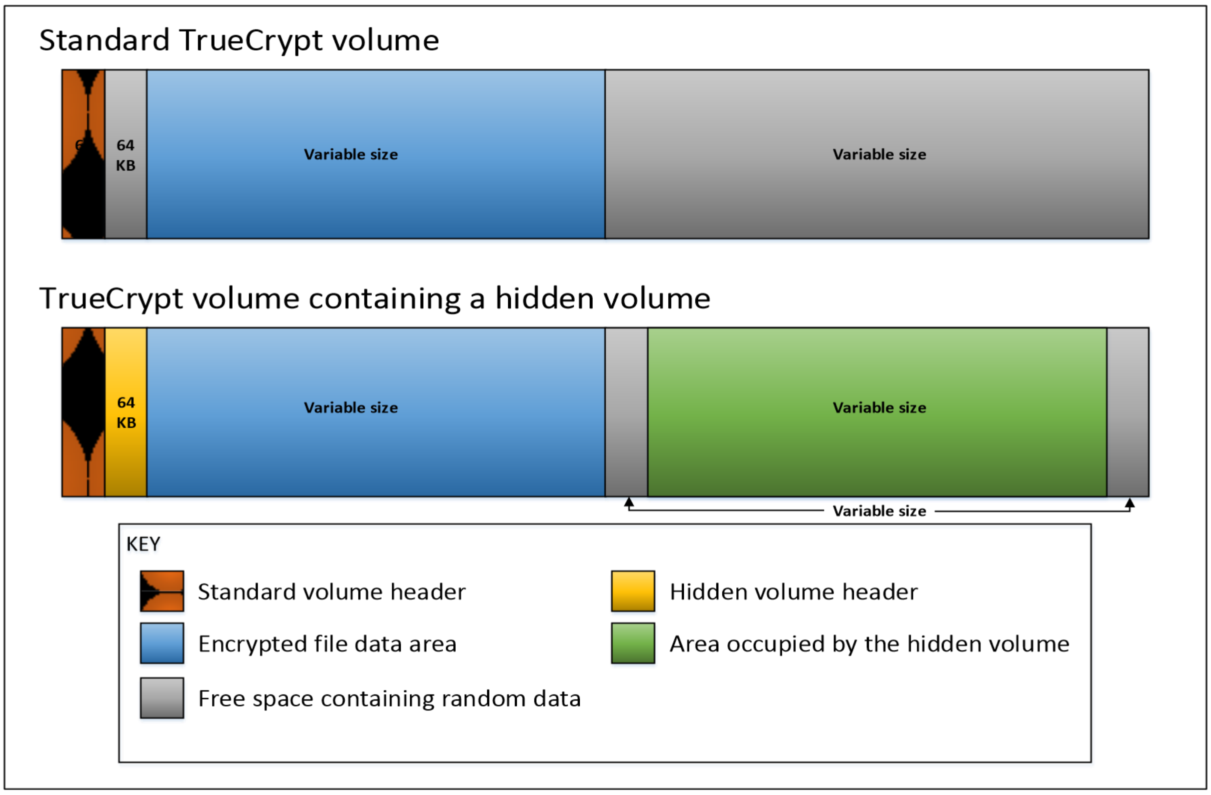


Figure 4.7 VeraCrypt Volumes

Users are required to input a password (unless password caching is enabled) in order to access either a hidden or normal VeraCrypt volume. Which volume, hidden or standard, is mounted depends on the password that the user enters. VeraCrypt first tries to use the password that was entered to mount the standard volume. It then attempts to mount the hidden volume with the same password if the previous attempt fails. Which VeraCrypt volume is accessible depends only on the password the user enters. Consequently, it is essential that the passwords for the hidden volume and the ordinary VeraCrypt volume differ from one another. Without the right password to access the concealed volume, its existence remains undetectable as it appears to contain only random data. These hidden volumes are commonly known as deniable file systems. Unlike standard VeraCrypt volumes stored in files, hidden volumes do not exhibit any identifiable structure within the file system.

To prevent the data of the hidden volume from being overwritten by filling the standard TrueCrypt volume, users can select the option "protect hidden volume against damage caused by writing to outer volume" when growing the standard TrueCrypt volume. This option safeguards the concealed volume from accidental harm [194].When the user selects the "protect hidden volume against damage caused by writing to outer volume" option and enters the password for the hidden volume, the header of the hidden volume is decrypted to access its size information. It's worth mentioning that these guard settings are not saved after dismounting the standard TrueCrypt volume. Therefore, the user must enable this option again when they mount the volume in the future to ensure the continued protection of the hidden volume [195]. The author categorizes VeraCrypt as a "dynamic deniable encryption scheme," similar to the definition provided by Oler and El Fray [196]. The article focuses on deniable file systems, where the dynamic aspect of VeraCrypt is discussed. The author highlights VeraCrypt's distinctive feature of enabling modifications to the data within the hidden area, which distinguishes it from other deniable encryption schemes that are typically static and do not allow such modifications [123], [32].

The thesis explores a method to identify hidden volume areas when multiple copies of the standard VeraCrypt volume exist at different times. However, this approach is not relevant to the current thesis, which focuses on a single instance of the volume. Detecting hidden volumes within a single instance is recognized as a particularly difficult task for forensic investigators. The work of Hargreaves et al. demonstrated the implementation of Karstens' technique in this context [34]. They utilized multiple volume copies to deduce the hidden volume's whereabouts. A cautionary statement found in various types of the VeraCrypt certification, starting from at least version 5.1a, explicitly alerts users to these risks.

In VeraCrypt, the diagram shown in the figure illustrates the layout of a disk that has been formatted with this software. The first part of the diagram represents a common scenario where the disk contains a single volume known as the Standard Volume. While VeraCrypt primarily functions as disk encryption software, it also offers an optional feature of plausible deniability. The initial operation performed by VeraCrypt involves filling the disk with random bytes, which is a typical practice for regular disk encryption tools. In the first section of the disk, there is the encrypted header of the Standard Volume (referred to as Header 1 in the diagram) and an empty slot that contains random bytes generated during the initialization process. This is followed by the section that holds the actual encrypted data of the Standard Volume (Volume 1 in the diagram), and then additional empty space that also contains random bytes from the initialization process.

VeraCrypt also offers the option to "embed" a Hidden Volume (Volume 2 in Figure 4.6) within the empty space left by the Standard Volume. This mechanism provides plausible deniability.

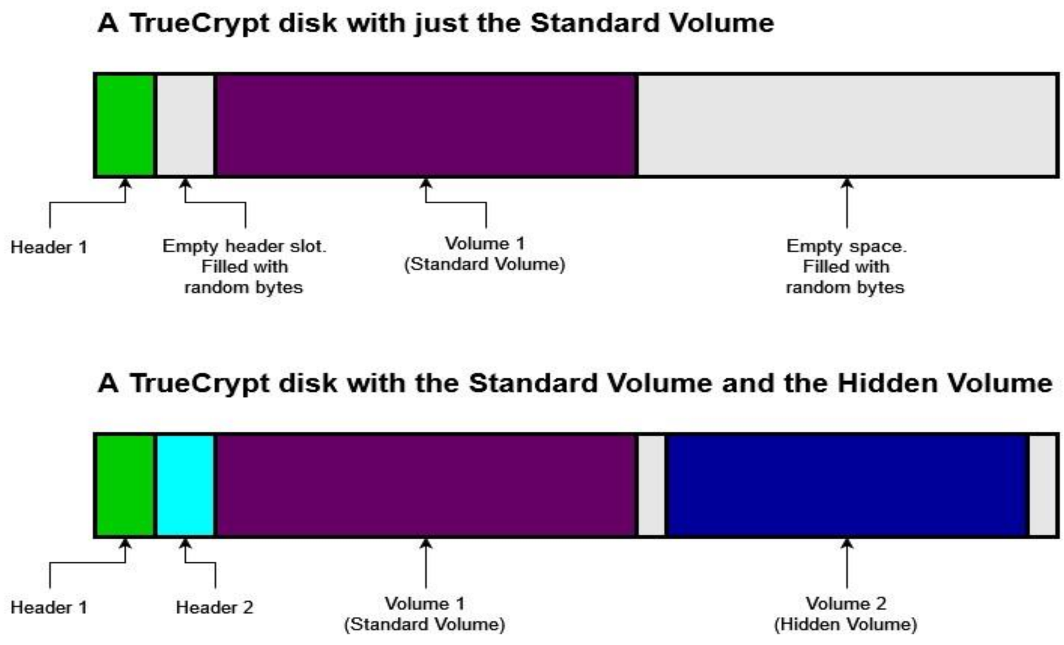

A TrueCrypt disk with just the Standard Volume
Header 1
Empty header slot. Filled with random bytes
Volume 1 (Standard Volume)
Empty space. Filled with random bytes
A TrueCrypt disk with the Standard Volume and the Hidden Volume
Header 1
Header 2
Volume 1 (Standard Volume)
Volume 2 (Hidden Volume)

Figure 4.8 VeraCrypt Disk Layout

Disc layout the layout of a VeraCrypt-formatted disc is shown in Figure 4.8 The first part illustrates the case of the disc only containing one volume, the Standard Volume; this was indeed a very common scenario since VeraCrypt was primarily disc encryption software (plausible deniability was just an optional feature). The first initialization operation performed by VeraCrypt is to fill the disc with random bytes; incidentally, this is also the case for regular disc encryption tools. The first part of the disc contains the encrypted header of the Standard Volume (Header 1 in the picture) and an empty slot filled with random bytes (remaining from the initialization procedure). Then comes the actual encrypted data section of the Standard Volume (Volume 1 in the picture), followed by some empty space, also filled with random bytes (also coming from the initialization procedure). VeraCrypt optionally allows you to "embed" a hidden volume (Volume 2 in the picture) in the empty space left by the standard volume; this is the mechanism providing plausible deniability. Its encrypted header (referred to as Header 2 in the diagram) is placed in the vacant space left after Header 1. The Standard Volume and the Hidden Volume are encrypted using two different passwords.

In the context of plausible deniability, some relevant terms include the synonyms for the Standard Volume, which are the outer volume (as it encompasses the Hidden Volume within its empty space) and the decoy volume (as it contains data intended to mislead authorities). The term "decoy volume" is more general and applies to volumes that are to be surrendered to authorities, not just in the case of VeraCrypt but also in other schemes where volumes are not necessarily nested within each other. contained in the file License.txt included in VeraCrypt binary and source code distribution packages.

4.7.2 VeraCrypt Volume's Encryption Analysis

An essential element in the volume building process is data encryption. In this section, they describe the various encryption techniques and how Veracrypt uses them. The VeraCrypt volume layout was reexamined using the TCanalyzer program, identifying the true volume layout and their header information. Both a hidden volume header and a normal VeraCrypt volume header were present in the partition. Random data was contained in the backup header if there was no hidden disc. With this understanding , the data within the standard VeraCrypt volume was selectively overwritten, making the standard volume unmount able within VeraCrypt, while the hidden volume remained accessible [123]. The Table 4.3 provides details related to the salt values (64 bytes each) associated with different types of VeraCrypt volume headers.

4.7.3 Integrated Cryptographic Algorithms

Veracrypt didn't just use these algorithms individually; it also proposed combining them in different ways. There is a total of five combinations: AES-Twofish, Serpent-AES, Twofish-Serpent, AES-Twofish-Serpent, and Serpent-Twofish-AES. For example, AES-Twofish implies that data blocks are encrypted first with Twofish and then with AES. In all three of these algorithm combinations, the data block size is 128 bits, and the key size is 256 bits.

Additionally, during the key derivation process, users have the option to choose from three hash algorithms: Ripemd-160, SHA-512, and Whirlpool.

Table 4.3 Volume Headers Salt

Volume Header Type	Salt 64 bytes
Standard	e7 48 80 88 ef 71 bf d6 0b 01 3e b7 c2 b9 7a 26 8c 62 e9 f5 33 b7 91 1b 64 30 02 61 31 1a e7 69 55 89 e8 ea a4 1a 3f 38 d4 4d 71 89 3c ac 20 de 9c 5a 39 39 52 61 59 53 86 32 aa 88 fb d5 2b b4
Hidden	6b 29 20 db 47 cd 4a b5 2a 3f 25 2f 19 25 ed 14 47 04 00 04 7f 8f 84 b4 ad e0 8e 7e 24 a4 d4 d7 85 5c c2 6d 92 f7 6d 1b 76 f2 03 48 40 6b 25 a9 96 d9 78 5f c1 ec 35 3a 03 1d 87 f4 4c 24 6c 4c
Standard Backup	a2 a1 5a 34 ea 61 f6 6f b0 9e 9f 46 01 09 83 49 83 3c 08 ae 38 0a a9 d5 6b c7 18 f3 ef 0d c6 ea 62 e0 77 61 6a 1b b1 d8 ec d1 4a fa 1f 0a e3 8c cf 21 53 96 4c de b7 9e b9 e1 f6 61 b0 01 d8 8c
Hidden Backup	30 52 4f 71 3f 5f 40 a8 85 32 f3 82 c0 85 b7 26 b0 7b ad d1 e3 fe d4 f3 73 91 37 39 35 7d 0b a8 d4 d2 46 52 a9 50 28 75 2d 1b 7d 8a 47 6d f3 f7 c2 72 53 13 e5 a8 0b 7e 7d ae 5f 34 55 b4 da 48

4.7.4 Encryption Operation Mode: XTS

XTS mode is a type of encryption mode inspired by XEX mode but with a small change: it uses two separate keys instead of just one, unlike XEX. XTS mode is particularly good for safeguarding data on storage devices and has received approval from organizations like NIST and IEEE [197] [198]. In XTS mode encryption, it can be explained like this: Ciphertext (C) is obtained by performing the following operations:

- Take the plaintext block (P) and XOR it with the result of two encryption operations.
- The first encryption operation (E) is done with the primary key (k1) and a value derived from the block index (i) and a primitive element (α).
- The second encryption operation (E) is done with the secondary key (k2) and the same value derived from the block index (i) and primitive element (α).

The keys used here are 256 bits long. The ⊕ symbol represents the exclusive OR (XOR) operation, and the ⊗ symbol represents polynomial multiplication modulo a specific polynomial (x^128 + x^7 + x^2 + x + 1). α is a primitive element in a finite field (GF (2^128)). In this context, "i" represents the block index within a data unit, and "n" represents the data unit index within the scope of k1. Both "i" and "n" start counting from 0. It's worth noting that in VeraCrypt, the data unit size is 512 bytes, and each block is 16 bytes, so the maximum value for "i" is 31.

4.8 Components and Applications

Components and Application plays a pivotal role in securing both data and communication, encompassing elements such as algorithms, key management, authentication, digital certificates, random number generators, hash functions, and compliance across diverse domains. The subsequent sections delve into the specifics of these aspects.

4.8.1 Command-Line Interface (CLI)

GPG provides a command-line interface that allows users to perform encryption, decryption, signing, and key management tasks. Users can integrate GPG into scripts and workflows.

4.8.2 Graphical User Interfaces

While GPG primarily offers a command-line interface, there are several third-party graphical user interfaces available that provide an easier way to perform GPG operations.

4.8.3 Email Encryption

GPG is commonly used to encrypt and sign emails. Users can encrypt the content of their emails to ensure that only the intended recipient can read it. Digital signatures verify the sender's identity and the integrity of the message.

4.8.4 File Encryption

GPG can encrypt and decrypt files and directories. This is useful for securing sensitive documents and data stored on local systems.

4.8.5 Secure Communication

GNU Privacy Guard (GPG) is a cryptographic tool that encrypts and digitally signs messages, ensuring their integrity and unreadable even if intercepted. It uses a public-key cryptosystem, is integrated with instant messaging clients, and is widely used for email encryption.

4.8.6 Code Signing

GPG is a software tool that digitally signs code, ensuring its authenticity and integrity. It uses a web of trust model to distribute and verify public keys, which can be uploaded to key servers or distributed through trusted channels. GPG promotes transparency, integrity, and trust within the software development community and end-users.

4.8.7 Password Management

Certain password managers employ GPG (GNU Privacy Guard) to encrypt and fortify the user's password database. This encryption method adds an extra layer of security by utilizing public-key cryptography. With GPG, the password manager encrypts the sensitive data using the recipient's public key, ensuring that only the intended recipient, typically the user themselves, can decrypt and access the stored passwords. This approach enhances security, as even if the password database were to be compromised, the encrypted data would remain unintelligible without the corresponding private key.

4.8.8 Key Generation

Users generate a key pair (public and private keys). The private key is kept secret and never shared, while the public key can be freely distributed.

4.8.9 Encryption

When encrypting a message or file, the sender employs the recipient's public key, a component of asymmetric encryption. This key is freely shareable and is used specifically for encryption. Once the message or file is encrypted with the recipient's public key, it becomes inaccessible to anyone without the corresponding private key, which is known only to the recipient. This ensures that only the intended recipient possesses the capability to decrypt and access the content, adding a robust layer of security to communications and file sharing.

4.8.10 Digital Signing

The sender uses their private key to generate a unique digital signature for a message or file, confirming its authenticity. The recipient then uses their public key to verify the signature, confirming the content's authenticity and integrity. This process ensures the authenticity of electronic communications and file exchanges.

4.8.11 Key Distribution

Users have the flexibility to disseminate their public keys through a range of methods, which may encompass traditional means like email, specialized platforms such as key servers designed for cryptographic key management, or even personal or professional websites dedicated to showcasing such information.

4.8.12 Trust Establishment

Users have the opportunity to verify keys and foster trust through various avenues. One method involves utilizing the Web of Trust model, where users establish connections with others they trust, creating a network where key authenticity can be verified through these trusted relationships. Alternatively, users can directly verify fingerprints, comparing them through secure channels like in-person meetings or encrypted communication to ensure the integrity of the keys.

4.8.13 Secure Email Communication

GPG can be used to encrypt and sign emails, ensuring that only authorized recipients can read the content and that the sender's identity is verified.

4.8.14 File Sharing

GPG (GNU Privacy Guard) is a secure file encryption solution that uses asymmetric encryption, ensuring only the intended recipient can decrypt and access the content. This method protects sensitive information from unauthorized access and maintains confidentiality, reducing the risk of data breaches or disclosures. It can be shared via various online channels.

4.8.15 Software Distribution

Developers can sign software releases with their GPG key, allowing users to verify the authenticity of downloaded software.

4.8.16 Digital Documentation

GPG can be used to sign digital documentation, ensuring that the content remains unchanged and is attributed to the correct author.

4.8.17 Protection Against Surveillance

GPG can protect against surveillance and eavesdropping by ensuring that only authorized parties can access the encrypted content.

4.8.18 Limitations Usability

GPG, a command-line encryption tool, can be challenging for beginners due to its command-line interface. However, with proper training and resources, it can be a powerful tool for secure communication and data protection.

4.8.19 Key Management

Key management is crucial in cryptographic systems like GPG, especially for private keys. Loss can cause irreversible data loss. Secure backups and multiple locations protect keys, preventing breaches and identity theft.

4.9 Comparative Analysis of Different Encryption Software

Comparative analysis of encryption software involves evaluating security features, ease of use, performance, platform compatibility, regulatory compliance, auditability, community support, cost, and recent vulnerabilities. Key factors include encryption algorithms, key management, authentication, user interface, scalability, industry standards, logging, auditing, community support as shown in Table. 4.4.

Table 4.4 Comparison Analysis of Different Encryption Software with VeraCrypt

FEATURES	PROTONMAIL	GNU PRIVACY GUARD (GPG)	AXCRYPT	VERACRYPT
Email Encryption	Yes	Yes	No	No
File Encryption	No	Yes	Yes	Yes
Open Source	Partial	Yes	Yes	Yes
Platform	Web, Mobile	Multi-platform	Windows, Mac,	Multi-platform
Compatibility	Desktop	Multi-platform	Android	Multi-platform
Public Key Encryption	Yes	Yes	No	No
Ease of Use	User-Friendly	Moderate	User-Friendly	User-Friendly
Security Features	End to End Encryption	PGP Encryption	AES Encryption	AES Encryption
Full Disk Encryption	No	No	No	Yes (Encrypts entire disk or partitions)
File and Folder Encryption	No	No	No	Yes (Encrypts individual File and folder
Open Source	Partial (Client-side code is open)	Yes	Partial (Some components are open source)	Yes
Performance	Web-based, may have latency	Depends on usage; command-line interface	User-friendly, may impact large files	Depends on usage; generally efficient
Authentication Methods	Password, Two-Factor Authentication	Password, Key file, Smart Card, Subkey	Password, Key file	Password key files
Hidden Volumes	No	Yes	No	Yes
Password Keyfiles	No	Yes	Yes	Yes

Table 4.4 provides a clear and concise comparison of key features and others among ProtonMail, GNU Privacy Guard (GPG), AxCrypt, and VeraCrypt. It's a helpful way to quickly assess the strengths and weaknesses of each encryption tool.

Table 4.5 Comparison with Software Reverse Engineering

Technique	**Description**	**comparison with Software Reverse Engineering**
Formal Verification	Uses mathematical methods to prove correctness of algorithms with respect to a formal specification.	Comprehensive understanding by combining static and dynamic information - Ability to overcome limitations of individual techniques
Symbolic Execution	Executes programs with symbolic inputs rather than real data.	Direct access to source code - Ability to analyze all possible execution paths - Integration with development tools and processes
Fuzz Testing	Automated technique feeding pseudo-random inputs to a program.	Ability to analyze binary executables directly - Low-level understanding of program behavior, data structures, and algorithms - Analysis of closed-source or proprietary software - Analysis of legacy systems or applications without source code
Code Auditing	Manual line-by-line inspection of code to find flaws.	Effective for finding source code errors, requires source code access and might not reveal third-party binary issues, which reverse engineering can analyze

The table 4.5 describe Formal verification, symbolic execution, fuzz testing, and code auditing are methods used to analyze software. Formal verification uses mathematical methods to prove algorithm correctness, while symbolic execution uses symbolic inputs. Fuzz testing is an automated technique that analyzes binary executables without source code, providing a low-level understanding of program behavior. Code auditing is a manual inspection of code to find flaws, but requires source code access. These methods differ from software reverse engineering, which has direct access to source code.

Software reverse engineering offers a unique advantage when examining binary executables, especially in scenarios where source code is unavailable, obfuscated, or when dealing with closed-source or proprietary software. Reverse engineering techniques can provide a low-level understanding of the program's behavior, data structures, and algorithms, enabling analyses that may not be possible with other techniques.

4.11 Summary of Chapter

Establishing a robust Web of Trust can be challenging due to the decentralized nature and the need for personal verification. In summary, GPG is a powerful tool for ensuring privacy, data integrity, and secure communication in the digital world. It is widely used by individuals, organizations, and developers who value data security and want to maintain control over their digital communication and information.

AxCrypt is a user-friendly file encryption tool that provides strong encryption for individual files and folders. It's a suitable choice for individuals who want to add an extra layer of security to their sensitive data without the complexity of full-disk encryption. However, users should be aware of its limitations and consider their encryption needs before using it.

CHAPTER V

CONCLUSION AND FUTURE DIRECTIONS

A novel innovative framework is designed to bolster the skills of forensic investigators in the analysis and comprehension of tools for encrypting discs with anti-forensic features. Its goal is to actively contribute to the continual endeavors to combat digital crime and safeguard digital evidence.

5.1 Conclusion

The thesis emphasizes the critical role of reverse engineering in software analysis, particularly in understanding the functionality and behavior of a program. Reverse engineering is a process that involves dissecting and sometimes decompiling software programmers to convert binary instructions into readable code, allowing engineers to comprehend the programmer’s actions and their impact on different systems. This knowledge is essential for developing solutions to mitigate any harmful effects caused by the programmed.

To understand and analyze Encryption software, engineers need to disassemble and sometimes decompile the software program. This process involves converting the binary instructions of the program into readable code mnemonics or higher-level structures. By doing this, engineers can figure out what the program does and which systems it affects. This knowledge helps engineers create solutions to mitigate the harmful effects of the program.

Reverse engineering (RE) uses different tools to uncover the purpose and functioning of a program. Through RE, engineers can identify the vulnerabilities that the malicious software aims to exploit. Even though those who create malware often try to mislead, RE can unveil information such as the program's creation time, accessed embedded resources, encryption keys, and various elements like file headers and metadata.

To achieve this understanding, reverse engineering utilizes a variety of tools and techniques. These tools help engineers identify vulnerabilities present in the software and uncover hidden information within the programmer’s code. By doing so, reverse engineering assists in enhancing software security and identifying potential threats or malicious activities.

The study highlights the emerging techno-legal challenges associated with using reverse engineering for law enforcement purposes. These challenges include uncertainties about the legality of the tools and methods used for evidence acquisition as well as concerns about complying with intellectual property and confidential information protection obligations. Additionally, the paper points out the existence of gaps between legal provisions and actual practices concerning the disclosure and peer review of sensitive digital forensic methodologies, the protection of trade secrets during investigations, and governmental vulnerability disclosure.

The Research provides valuable insights into the VeraCrypt source code flow using a table representation. The table outlines the different steps involved in the functioning of VeraCrypt, such as initialization, user interface handling, volume management, encryption and decryption operations, file system integration, key management, error handling, and logging. This representation aids in understanding the internal workings of VeraCrypt, a popular open-source encryption software.

VeraCrypt used to be a really famous free software that could keep your computer files safe. not only encrypts stuff faster than similar programs like Bitlocker, File Vault, and PGP, but it also does more things when it comes to protecting data. It can create regular encrypted storage, secret hidden storage, lock up whole parts of a computer, and even keep the whole operating system safe.

Furthermore, the paper discusses the characteristics of data storage on a typical hard disc drive (HDD). It explains the concept of logical block addressing, which is a method used to access and locate sectors on modern hard drives. The Research emphasizes that the operating system's representation of data is abstracted from the actual physical storage mechanism employed by the device.

The study also delves into the details of VeraCrypt encrypted volumes, which are an essential part of the software's functionality. It distinguishes between coded file containers, which are encrypted files within the file system, and volumes that encrypt complete partitions or devices. The paper elaborates on the concept of hidden volumes, which are concealed within standard VeraCrypt volumes and can only be accessed using different passwords. This "plausible deniability" feature allows users to keep sensitive data hidden from authorities by presenting a decoy volume with a different password.

Moreover, the paper highlights VeraCrypt's dynamic, deniable encryption scheme. Unlike other deniable encryption schemes that are static and do not allow modifications to hidden areas, VeraCrypt permits changes to data within hidden volumes. This feature adds an extra layer of complexity to forensic investigations as the existence of hidden volumes becomes challenging to determine.

This thesis emphasizes the significance of reverse engineering in software analysis and its role in understanding programmed functionality and vulnerabilities. It raises awareness about the emerging challenges of using reverse engineering for law enforcement purposes and proposes recommendations for addressing them. The detailed exploration of VeraCrypt's source code flow and encrypted volumes provides valuable insights into the inner workings of this encryption software. Overall, the paper contributes to our understanding of reverse engineering's crucial role in cybersecurity and digital forensic investigations.

5.2 Future Directions

The thesis highlights the importance of dealing with challenges related to reverse engineering in digital forensic investigations by setting up clear rules and best practices. Here are the suggested recommendations:

Create Digital Forensic Regulations: We need clear rules that govern the use of reverse engineering tools and methods in law enforcement investigations. These rules should ensure that reverse engineering is done ethically and legally, addressing issues like evidence collection, intellectual property, and confidentiality.

Improve Disclosure and Peer-Review Practices: To ensure transparency and reliability in digital forensic methods, experts should share their techniques openly for review. This will help validate the methods used and increase trust in the findings.

Strengthen Governmental Vulnerability Disclosure: Governments and regulators should responsibly share vulnerabilities found through reverse engineering with relevant parties like software developers and cybersecurity experts. This will lead to prompt security updates and better protection against cyber threats.

Study Slack Space Implications: Further research is needed on the impact of slack space in storage. Understanding leftover data can improve data recovery and management in forensic analysis. Better Identification and Handling of Hidden Volumes: Efforts should be made to improve identifying and managing hidden volumes within encrypted containers. Collaboration between forensic experts and software developers can help develop techniques to distinguish between standard and hidden volumes for more accurate investigations.

Foster Collaboration Among Stakeholders: Collaboration between law enforcement, forensic experts, and regulators is crucial in addressing complex challenges related to reverse engineering. Open dialogue and cooperation can lead to balanced and effective solutions that follow both legal and ethical principles.

By following these recommendations, law enforcement and forensic experts can navigate the evolving landscape of reverse engineering and digital forensics responsibly, ensuring data security and fairness in legal proceedings.

The thesis explores the role of reverse engineering in software analysis and its implications for cybersecurity and digital forensic investigations. Key recommendations include creating digital forensic regulations, improving disclosure and peer-review practices, strengthening.

government vulnerability disclosure, studying slack space implications, improving the identification and handling of hidden volumes, and fostering collaboration among stakeholders. These recommendations aim to create more ethical, legally compliant, and effective investigations, enhancing data security and fairness in legal proceedings. The research also emphasizes the importance of slack space in storage, better identification and handling of hidden volumes, and fostering collaboration among stakeholders. By following these recommendations, the future of reverse engineering in digital forensic investigations can be more ethical and transparent.

The thesis highlights the imperative of addressing challenges associated with reverse engineering in digital forensic investigations by advocating for the establishment of clear guidelines and best practices. The following recommendations are proposed:

- **Create Digital Forensic Regulations:**
    - Formulate explicit regulations governing the ethical and legal use of reverse engineering tools in law enforcement investigations. These regulations should address concerns related to evidence collection, intellectual property, and confidentiality.

- **Improve Disclosure and Peer-Review Practices:**
    - Encourage experts to openly share their digital forensic techniques for thorough peer review. This transparency fosters trust in the methods employed and enhances the credibility of investigative findings.

- **Strengthen Governmental Vulnerability Disclosure:**
    - Advocate for responsible sharing of vulnerabilities discovered through reverse engineering by governments and regulators with relevant stakeholders, including software developers and cybersecurity experts. This collaboration can lead to swift security updates and improved defense against cyber threats.

- **Study Slack Space Implications**

    - Conduct further research on the implications of slack space in storage. A comprehensive understanding of residual data can significantly enhance data recovery and management in forensic analysis.

- **Better Identification and Handling of Hidden Volumes**
    - Enhance efforts to identify and manage hidden volumes within encrypted containers. Collaboration between forensic experts and software developers is essential to devise techniques that distinguish between standard and hidden volumes, thereby facilitating more accurate investigations.

- **Foster Collaboration Among Stakeholders:**
    - Emphasize the importance of collaboration between law enforcement, forensic experts, and regulators to effectively address the intricate challenges associated with reverse engineering. Open dialogue and cooperation can yield balanced and impactful solutions adhering to both legal and ethical principles.

By adhering to these recommendations, law enforcement and forensic experts can navigate the dynamic landscape of reverse engineering and digital forensics responsibly, ensuring data security and fairness in legal proceedings. This approach promotes ethical and transparent practices, addressing key aspects such as slack space, hidden volumes, and collaborative efforts among stakeholders in digital forensic investigations.

REFERENCES

[1] J. Weber, “Secure Hash Standard Federal,” *Fed. Inf. Process. Stand. Publ. 180-2,* vol. 2, 2002, [Online]. Available: http://csrc.nist.gov/publications/PubsFIPS.html#fips180-4.

[2] U. S. C. J. System, S. E. Goodison, R. C. Davis, and B. A. Jackson, “Digital Evidence and the US Criminal justice system,” *Crimanal justices J.*, vol. 5, 2019.

[3] D. Vadlamudi, K. Thirupathi Rao, P. Vidyullatha, and B. RajasekharReddy, “Analysis on digital forensics challenges and anti-forensics techniques in cloud computing,” *Int. J. Eng. Technol.*, vol. 7, no. 2.7 Special Issue 7, pp. 1072–1075, 2018, doi: 10.14419/ijet.v7i2.7.12230.

[4] S. Singh, *The Code Book the secret history of codes and code-breaking:* 2010.

[5] J. Kävrestad, *Fundamentals of Digital Forensics,* 2nd ed. Cham: Springer International Publishing, 2020. doi: 10.1007/978-3-030-38954-3.

[6] C. A. Sari, E. H. Rachmawanto, and E. J. Kusuma, “Good Performance Images Encryption Using Selective Bit T-Des on Inverted Lsb Steganography,” *J. Ilmu Komput. dan Inf.*, vol. 12, no. 1, p. 41, 2019, doi: 10.21609/jiki.v12i1.646.

[7] D. Chesnut *et al.*, “Data Encryption and Hashing Primer,” *NAACCR Data Secur. Confidentiality Workgr.*, no. January, pp. 1–19, 2023.

[8] C. D. Hosmer, “Examining the Encryption Threat,” *Int. J. Digit. Evid.*, vol. 2, no. 3, pp. 1–11, 2004.

[9] L. Khati *et al.*, *Full Disk Encryption : Bridging Theory and Practice To cite this version : HAL Id : hal-01403418 Full Disk Encryption : Bridging Theory and Practice*. 2017.

[10] F. Hou, N. Xiao, F. Liu, and H. He, “Secure disk with authenticated encryption and IV verification,” *5th Int. Conf. Inf. Assur. Secur. IAS 2009*, vol. 2, pp. 41–44, 2009, doi: 10.1109/IAS.2009.48.

[11] K. B. Harriss, Dr Lydia, “Digital Forensics and Crime,” *POSTnote 520 Digit. Forensics Crime,* no. 520, pp. 1–5, 2016, [Online]. Available: http://researchbriefings.files.parliament.uk/documents/POST-PN-0520/POST-PN-0520.pdf

[12] M. Turkanović and M. Hölbl, “A Method for Obtaining Digital Signatures and Public- Key

Cryptosystems," *Wirel. Pers. Commun.*, vol. 77, no. 2, pp. 907–922, 2014, doi: 10.1007/s11277-013-1543-8.

[13] S. C. W. Ankita Jadhav, Prajakta Choudhari, Tejaswini Gherade, "Implementation of Advanced Encryption Standard (AES) on FPGA," *nternational J. Eng. Sci. Comput. May 2017*, vol. 7, no. 5, pp. 12313–12317, 2017.

[14] S. Chandra, S. Paira, S. S. Alam, and G. Sanyal, "A comparative survey of symmetric and asymmetric key cryptography," *2014 Int. Conf. Electron. Commun. Comput. Eng. ICECCE 2014*, pp. 83–93, 2014, doi: 10.1109/ICECCE.2014.7086640.

[15] M. Panda, "Performance Evaluation of Symmetric Encryption Algorithms for Information Security," *Int. J. Adv. Res. Trends Eng. Technol.*, vol. 4, no. 11, pp. 37–41, 2017.

[16] M. Al Fahdi, N. L. Clarke, and S. M. Furnell, "Challenges to digital forensics: A survey of researchers & practitioners attitudes and opinions," *2013 Inf. Secur. South Africa - Proc. ISSA 2013 Conf.*, pp. 1–8, 2013, doi: 10.1109/ISSA.2013.6641058.

[17] R. I. and Z. H. Yunus Yusoff, "COMMON PHASES OF COMPUTER FORENSICS INVESTIGATION MODELS," *Int. J. Comput. Sci. Inf. Technol.*, vol. 3, no. 3, pp. 17–31, 2011, doi: 10.1155/2016/3068347.

[18] H. Majed, H. N. Noura, and A. Chehab, "Overview of Digital Forensics and Anti-Forensics Techniques," *8th Int. Symp. Digit. Forensics Secur. ISDFS 2020*, no. May, 2020, doi: 10.1109/ISDFS49300.2020.9116399.

[19] N. A. Hassan, "Antiforensics Techniques How," *Digit. Forensics Basics*, no. 8, pp. 291–310, 2019, doi: 10.1007/978-1-4842-3838-7.

[20] I. Fontana, "The human (in)security trap: how European border(ing) practices condemn migrants to vulnerability," *Int. Polit.*, vol. 59, no. 3, pp. 465–484, 2022, doi: 10.1057/s41311-020-00268-y.

[21] M. Dayalan, "Understanding Software Systems Using Reverse Engineering Technology," *Object-Oriented Technol. Database Softw. Syst.*, vol. 6, no. 4, pp. 240–252, 2019, doi: 10.1142/9789812831163_0016.

[22] A. Van Deursen and E. Burd, "Software reverse engineering," *J. Syst. Softw.*, vol. 77, no. 3, pp. 209–211, 2005, doi: 10.1016/j.jss.2004.03.031.

[23] H. M. Kienle and H. A. Mu, “The Tools Perspective on Software Reverse Engineering : Requirements , Construction , and Evaluation,” vol. 79, no. 10, pp. 189–290, 2010, doi: 10.1016/S0065-2458(10)79005-7.

[24] B. Tenovo, P. Mursanto, and H. B. Santoso, “Reverse Engineering Software Tools Based on a Comprehension System,” *Proc. - 2017 7th World Eng. Educ. Forum, WEEF 2017- Conjunction with 7th Reg. Conf. Eng. Educ. Res. High. Educ. 2017, RCEE RHEd 2017, 1st Int. STEAM Educ. Conf. STEAMEC 201*, pp. 202–209, 2018, doi: 10.1109/WEEF.2017.8467039.

[25] P. P. Dudenhofer, “Modeling and Automating the Cyber Reverse Engineering Cognitive Process,” *Colloquim Inf. Syst. Secur.*, 2019.

[26] B. Zhang, “Research Summary of Anti-debugging Technology,” *J. Phys. Conf. Ser.*, vol. 1744, no. 4, 2021, doi: 10.1088/1742-6596/1744/4/042186.

[27] C. Bradley and B. Currie, “Advances in the Field of Reverse Engineering,” *Comput. Aided. Des. Appl.*, vol. 2, no. 5, pp. 697–706, 2005, doi: 10.1080/16864360.2005.10739029.

[28] N. F. M. Sani, N. A. M. Ariffin, and R. Atan, “Design of object-oriented debugger model by using unified modeling language,” *J. Comput. Sci.*, vol. 9, no. 1, pp. 16–29, 2013, doi: 10.3844/jcssp.2013.16.29.

[29] M. E. Rosenbaum, N. L. Nisly, K. J. Ferguson, and E. W. Kligman, *Academic physicians and complementary and alternative medicine: An institutional survey*, vol. 17, no. 1. 2002. doi: 10.1177/106286060201700102.

[30] I. D. Baxter and M. Mehlich, “Reverse engineering is reverse forward engineering,” *Sci. Comput. Program. 9 0167-6423/00/$ - see Front matter c? 2000 Elsevier Sci. B.V. All rights Reserv.*, vol. 36, pp. 131–147, 2000.

[31] Keith Bennett, “IEEE,” *Leg. Syst. Coping with Success.*, 1995.

[32] J. V. Harrison and A. Berglas, “Data flow analysis within the ITOC information system design recovery tool,” *Proc. IEEE Int. Autom. Softw. Eng. Conf. ASE*, pp. 227–236, 1997, doi: 10.1109/ase.1997.632843.

[33] J. Sadiq and T. Waheed, “Reverse engineering & design recovery: An evaluation of design recovery techniques,” *Proc. - Int. Conf. Comput. Networks Inf. Technol.*, pp. 325–332, 2011, doi: 10.1109/ICCNIT.2011.6020889.

[34] S. Diehl, *Software visualization, Visualizing the Structure, Behaviour, and Evolution of Software*, no. March. Germany: Springer Singapore, 2007. doi: 10.1145/1062455.1062634.

[35] T. Ball and S. G. Eick, "Software visualization in the large," *Computer (Long. Beach. Calif).*, vol. 29, no. 4, pp. 33–43, 1996, doi: 10.1109/2.488299.

[36] B. A. Price, R. M. Baecker, and I. S. Small, "A principled taxonomy of software visualization," *J. Vis. Lang. Comput.*, vol. 4, no. 3, pp. 211–266, 1993, doi: 10.1006/jvlc.1993.1015.

[37] C. Raibulet, F. Arcelli Fontana, and M. Zanoni, "Model-driven reverse engineering approaches: A systematic literature review," *IEEE Access*, vol. 5, no. c, pp. 14516–14542, 2017, doi: 10.1109/ACCESS.2017.2733518.

[38] M. Ahmad, M. Z. Alam, Z. Umayya, S. Khan, and F. Ahmad, "An image encryption approach using particle swarm optimization and chaotic map," *Int. J. Inf. Technol.*, vol. 10, no. 3, pp. 247–255, 2018, doi: 10.1007/s41870-018-0099-y.

[39] P. S. Antón and R. H. Anderson, *Finding and Fixing Vulnerabilities in Information Systems: The Vulnerability Assessment and Mitigation Methodology*. 2022. doi: 10.7249/mr1601.

[40] Q. X. Miao, "Research and analysis on Encryption Principle of TrueCrypt software system," *2nd Int. Conf. Inf. Sci. Eng. ICISE2010 - Proc.*, pp. 1409–1412, 2010, doi: 10.1109/ICISE.2010.5691392.

[41] R. Sharma, S. Dangi, and P. Mishra, "A Comprehensive Review on Encryption based Open Source Cyber Security Tools," *Proc. IEEE Int. Conf. Signal Process. Control*, vol. 2021-Octob, pp. 614–619, 2021, doi: 10.1109/ISPCC53510.2021.9609369.

[42] G. Knight, "Encrypt data using VeraCrypt," *London Sch. Hyg. Trop. Med.*, vol. 44, no. 0, 2017, [Online]. Available: www.lshtm.ac.uk/library/

[43] V. Information, *Truecrypt: User Guide, Version 7.1a*, 2nd ed. US: Truecrypt, 2012. [Online]. Available: www.truecrypt.org

[44] E. Casey, G. Fellows, M. Geiger, and G. Stellatos, "The growing impact of full disk encryption on digital forensics," *Digit. Investig.*, vol. 8, no. 2, pp. 129–134, 2011, doi: 10.1016/j.diin.2011.09.005.

[45] A. Balducci, S. Devlin, and T. Ritter, "Open Crypto Audit Project - TrueCrypt

Cryptographic Review," *iSECpartners*, no. Security Assesment, 2015.

[46] B. David, E. Filiol, and K. Gallienne, "Structural analysis of binary executable headers for malware detection optimization," *J. Comput. Virol. Hacking Tech.*, vol. 13, no. 2, pp. 87–93, 2017, doi: 10.1007/s11416-016-0274-2.

[47] E. Summary and I. D. a Pro, "Executive Summary : IDA Pro – at the cornerstone of IT security What is IDA Pro ? How is IDA Pro useful ? Who are IDA Pro users ?," *PC Mag.*, 2009.

[48] D. E. W. E. B. Denning, "Encryption and Evolving Technologies," in *National Strategy Information Center's US Working Group on Organized Crime (WGOC)*, 1997, no. July, pp. 1–23.

[49] Anton Benjamin Vickerman, "Sentencing Remarks of His Honour Judge John Evans R -v- Anton Benjamin Vickerman," *Judic. Engl. wales*, no. August, 2012.

[50] S. Singh, "The code book : the secret history of codes and code-breaking cryptography / Simon Singh." 2000. [Online]. Available: http://search.ebscohost.com/login.aspx?direct=true&db=cat02326a&AN=usl.848099&site=eds-live

[51] R. Singrore, "A Study To Examine Forensic Cybercrime and the Role of Computer Forensics," *Int. J. Innov. Res. Technol. Sci.*, vol. 12, no. 2, 2024.

[52] C. Tan, L. Zhang, and L. Bao, "A Deep Exploration of BitLocker Encryption and Security Analysis," *Int. Conf. Commun. Technol. Proceedings, ICCT*, vol. 2020-Octob, pp. 1070–1074, 2020, doi: 10.1109/ICCT50939.2020.9295908.

[53] C. M. and B. van Gastel, "Self-encrypting deception: weaknesses in the encryption of solid state drives," *IEEE Access*, 2019.

[54] B. D. Ghode and D. S. K. J. Akhlesh Kumar, "Forensic Investigation Utilizing RAM Capture to Decrypt Bitlocker Volumes: A Case Study.," vol. 9, no. 8, pp. 65–72, 2022, [Online]. Available: https://d1wqtxts1xzle7.cloudfront.net/100750049/JETIR2208208-libre.pdf?1680755791=&response-content-disposition=inline%3B+filename%3DForensic_Investigation_Utilizing_RAM_Cap.pdf&Expires=1710779072&Signature=c~YIQ7aM9ieplPNdZ141kMmb6cOZ3MCRw4hyyaRM5OnNes48

[55] D. Evangelin, J. Jelsteen, J. A. Pushparani, and J. N. S. Jebastin, “Importance of Software Re-Engineering Process and Program Based Analysis in Reverse Engineering Process,” vol. 1, no. 1, pp. 11–18, 2013.

[56] A. Wali, H. Ravichandran, and S. Das, “A 2D Cryptographic Hash Function Incorporating Homomorphic Encryption for Secure Digital Signatures,” *Adv. Mater.*, vol. 2400661, pp. 1–15, 2024, doi: 10.1002/adma.202400661.

[57] Henry B Wolfe, “Encountering Encrypted Evidence (potential),” *Proc. 2002 InSITE Conf.*, no. June, 2002, doi: 10.28945/2590.

[58] P. Q. Nguyen, “Can we trust cryptographic software? Cryptographic flaws in GNU privacy guard v1.2.3,” *Int. Assoc. Cryptologic Res.*, vol. 3027, pp. 555–570, 2004, doi: 10.1007/978-3-540-24676-3_33.

[59] E. C. Hosgor, *Detection and Mitigation of Anti-Forensics Using Forensic Tools*, no. 1–83. 2006. [Online]. Available: https://apps.dtic.mil/sti/citations/AD1069614%0Ahttps://apps.dtic.mil/sti/pdfs/AD1069614.pdf

[60] S. Yadav, K. Ahmad, and J. Shekhar, “Analysis of digital forensic tools and investigation process,” *Commun. Comput. Inf. Sci.*, vol. 169 CCIS, pp. 435–441, 2011, doi: 10.1007/978-3-642-22577-2_59.

[61] J. Cappaert, N. Kisserli, D. Schellekens, and B. Preneel, “Self-encrypting code to protect against analysis and tampering,” in *Benelux Workshop on Information and System Security*, 2006, p. 14. [Online]. Available: http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Self-encrypting+Code+to+Protect+Against+Analysis+and+Tampering#0

[62] J. B. Awotunde, A. O. Ameen, I. D. Oladipo, A. R. Tomori, and M. Abdulraheem, “Evaluation of four encryption algorithms for viability, reliability and performance estimation,” *Niger. J. Technol. Dev.*, vol. 13, no. 2, p. 74, 2017, doi: 10.4314/njtd.v13i2.5.

[63] J. Cappaert, B. Preneel, B. Anckaert, M. Madou, and K. De Bosschere, “Towards tamper resistant code encryption: Practice and experience,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 4991 LNCS, pp. 86–100, 2008, doi: 10.1007/978-3-540-79104-1_7.

[64] H. M. A. K. Diaa Salama Abdul. Elminaam and and M. M. Hadhoud, “Performance evaluation of various symmetric encryption algorithms,” *Int. J. Comput. Sci. Netw. Secur.*, vol. 8, no. 12, pp. 280–286, 2008, doi: 10.1109/PDGC.2014.7030724.

[65] M. A. Alomari, K. Samsudin, and A. R. Ramli, “A study on encryption algorithms and modes for disk encryption,” *2009 Int. Conf. Signal Process. Syst. ICSPS 2009*, pp. 793–797, 2009, doi: 10.1109/ICSPS.2009.118.

[66] J. Kizza and F. Migga Kizza, “Digital Evidence and Computer Crime,” in *Securing the Information Infrastructure*, 2011. doi: 10.4018/9781599043791.ch015.

[67] E. Casey, G. Fellows, M. Geiger, and G. Stellatos, “The growing impact of full disk encryption on digital forensics,” *Digit. Investig.*, vol. 8, no. 2, pp. 129–134, 2011, doi: 10.1016/j.diin.2011.09.005.

[68] F. Cohen, “Two models of digital forensic examination,” in *4th International Workshop on Systematic Approaches to Digital Forensic Engineering, SADFE 2009*, 2009, vol. 1, no. 3, pp. 42–53. doi: 10.1109/SADFE.2009.8.

[69] E. Pimenidis, “Computer Anti-forensics Methods and Their Impact on Computer Forensic Investigation Computer Anti-forensics Methods and Their Impact on,” no. August 2016, pp. 145–155, 2016, doi: 10.1007/978-3-642-04062-7.

[70] F. Mouton and H. S. Venter, “A prototype for achieving digital forensic readiness on wireless sensor networks,” *IEEE AFRICON Conf.*, no. September 2011, 2011, doi: 10.1109/AFRCON.2011.6072117.

[71] S. Alharbi, J. Weber-Jahnke, and I. Traore, “The proactive and reactive digital forensics investigation process: A systematic literature review,” *International Journal of Security and its Applications*, vol. 5, no. 4. pp. 59–72, 2011.

[72] W. Janet, “ACPO Good Practice Guide for Digital Evidence,” *Pet. Econ.*, vol. 71, no. 10, p. 41, 2012.

[73] A. M. and S. Ying, “Privacy Impacts of Data Encryption on the Efficiency of Digital Forensics Technology,” *Int. J. Adv. Comput. Sci. Appl.*, vol. 4, no. 5, pp. 36–40, 2013, doi: 10.14569/ijacsa.2013.040506.

[74] M. S. Zareen, A. Waqar, and B. Aslam, “Digital forensics: Latest challenges and response,” *Conf. Proc. - 2013 2nd Natl. Conf. Inf. Assur. NCIA 2013*, pp. 21–29, 2013, doi:

10.1109/NCIA.2013.6725320.

[75] E. Casey, “Practical approaches to recovering encrypted digital evidence,” *Proc. Digit. Forensic Res. Conf. DFRWS 2002 USA*, vol. 1, no. 3, 2002.

[76] A. Davies, *Detecting hidden volumes and operating systems*, no. September. 2014.

[77] E. Casey, “Differentiating the phases of digital investigations,” *Digit. Investig.*, vol. 19, pp. A1–A3, 2016, doi: 10.1016/j.diin.2016.11.001.

[78] V. R. Kebande and I. Ray, “A generic digital forensic investigation framework for Internet of Things (IoT),” *Proc. - 2016 IEEE 4th Int. Conf. Futur. Internet Things Cloud, FiCloud 2016*, pp. 356–362, 2016, doi: 10.1109/FiCloud.2016.57.

[79] E. A. Vincze, “Challenges in digital forensics,” *Police Pract. Res.*, vol. 17, no. 2, pp. 183–194, 2016, doi: 10.1080/15614263.2015.1128163.

[80] M. Computing and U. S. Kumar, “SMS Security System Using Encryption Techniques,” *Int. J. Comput. Sci. Mob. Comput.*, vol. 8, no. 5, pp. 132–142, 2019.

[81] V. Jusas, D. Birvinskas, and E. Gahramanov, “Methods and tools of digital triage in forensic context: Survey and future directions,” *Symmetry (Basel).*, vol. 9, no. 4, pp. 1–20, 2017, doi: 10.3390/sym9040049.

[82] M. A. Ako, “Advanced Encryption Standard (AES) Algorithm to Encrypt and Decrypt Data,” *Cryptogr. Netw. Secur.*, no. June, 2017, [Online]. Available: https://www.researchgate.net/publication/317615794

[83] S. Dogan, “A New Approach for Data Hiding based on Pixel Pairs and Chaotic Map,” *Int. J. Comput. Netw. Inf. Secur.*, vol. 10, no. 1, pp. 1–9, 2018, doi: 10.5815/ijcnis.2018.01.01.

[84] A. Arewa, “Borderless crimes and digital forensic: Nigerian perspectives,” *J. Financ. Crime*, vol. 25, no. 2, pp. 619–631, 2018, doi: 10.1108/JFC-12-2016-0079.

[85] H. Arshad, A. Bin Jantan, and O. I. Abiodun, “Digital forensics: Review of issues in scientific validation of digital evidence,” *J. Inf. Process. Syst.*, vol. 14, no. 2, pp. 346–376, 2018, doi: 10.3745/JIPS.03.0095.

[86] C. Meijer and B. Van Gastel, “Advisory on Solid State Disks (SSDs) / Digital Security , Radboud University / 5-11-2018 Research results,” pp. 1–2, 2018.

[87] H. Majed, H. N. Noura, and A. Chehab, “Overview of Digital Forensics and Anti-Forensics

Techniques," *8th Int. Symp. Digit. Forensics Secur. ISDFS 2020*, no. June, 2020, doi: 10.1109/ISDFS49300.2020.9116399.

[88] K. Patel, "Performance analysis of AES, DES and Blowfish cryptographic algorithms on small and large data files," *Int. J. Inf. Technol.*, vol. 11, no. 4, pp. 813–819, 2019, doi: 10.1007/s41870-018-0271-4.

[89] M. Geiger, "Counter-forensic tools: Analysis and data recovery," *18th FIRST Conf.*, vol. 65, p. 140, 2006, [Online]. Available: http://72.3.219.184/conference/2006/papers/geiger-matthew-papers.pdf

[90] E. Casey, "Digital Evidence and Computer Crime, Third Edition," p. 840, 2011.

[91] K. Kumar and P. Kaur, "A Generalized Process of Reverse Engineering in Software Protection & Security," *Int. J. Comput. Sci. Mob. Comput.*, vol. 4, no. 5, pp. 534–544, 2015.

[92] N. Niknejad, W. Ismail, I. Ghani, B. Nazari, M. Bahari, and A. R. B. C. Hussin, "Understanding Service-Oriented Architecture (SOA): A systematic literature review and directions for further investigation," *Inf. Syst.*, vol. 91, p. 101491, 2020, doi: 10.1016/j.is.2020.101491.

[93] M. A. Wani, A. AlZahrani, and W. A. Bhat, "File system anti-forensics – types, techniques and tools," *Comput. Fraud Secur.*, vol. 2020, no. 3, pp. 14–19, 2020, doi: 10.1016/S1361-3723(20)30030-0.

[94] K. Dahbur, "Proceedings of the 2nd International Conference on Intelligent Semantic Web-Services and Applications, ISWSA'11," *ACM International Conference Proceeding Series*, vol. 2. 2011.

[95] V. R. Kebande and H. S. Venter, "A Functional Architecture for Cloud Forensic Readiness Large-scale Potential Evidence Analysis A Functional Architecture for Cloud Forensic Readiness Large-scale Potential Digital Evidence Analysis .," no. July, 2015, doi: 10.13140/RG.2.1.1052.1440.

[96] K. Shekhanin, A. Kuznetsov, V. Krasnobayev, and O. Smirnov, "Detecting hidden information in fat," *Int. J. Comput. Netw. Inf. Secur.*, vol. 12, no. 3, pp. 33–43, 2020, doi: 10.5815/ijcnis.2020.03.04.

[97] R. Montasari, R. Hill, S. Parkinson, P. Peltola, A. Hosseinian-Far, and A. Daneshkhah, "Digital Forensics," *Int. J. Organ. Collect. Intell.*, vol. 10, no. 2, pp. 37–53, 2020, doi:

10.4018/ijoci.2020040103.

[98] B. A. Sassani, M. Alkorbi, N. Jamil, M. A. Naeem, and F. Mirza, “Evaluating Encryption Algorithms for Sensitive Data Using Different Storage Devices,” *Scientific Programming*, vol. 2020. 2020. doi: 10.1155/2020/6132312.

[99] K. Scarfone, M. Sexton, and M. Souppaya, “Guide to Storage Encryption Technologies for End User Devices Recommendations of the National Institute of Standards and Technology,” pp. 1–40, 2007.

[100] S. L. Garfinkel, “Digital forensics research: The next 10 years,” *Digit. Investig.*, vol. 7, no. SUPPL., pp. S64–S73, 2010, doi: 10.1016/j.diin.2010.05.009.

[101] Mohammed I Alghamdi, “Digital Forensics in Cyber Security—Recent Trends, Threats, and Opportunities,” *Cybersecurity Threat. with New Perspect.*, vol. 11, no. New Edition, pp. 1–173, 2021, doi: DOI: 10.5772/intechopen.94452.

[102] A. Al-Dhaqm *et al.*, “Digital Forensics Subdomains: The State of the Art and Future Directions,” *IEEE Access*, vol. 9, pp. 152476–152502, 2021, doi: 10.1109/ACCESS.2021.3124262.

[103] Elia Anzuoni, “Hidden Filesystem Design and Improvement,” pp. 1–65, 2022.

[104] Mamoun, Z. Jalil, K. Kifayat, and T. R. Gadekallu, “A Comprehensive Survey on Computer Forensics: State-of-the-Art, Tools, Techniques, Challenges, and Future Directions,” *IEEE Access*, vol. 10, pp. 11065–11089, 2022, doi: 10.1109/ACCESS.2022.3142508.

[105] R. Stoykova, R. Nordvik, M. Ahmed, K. Franke, S. Axelsson, and F. Toolan, “Legal and technical questions of file system reverse engineering,” *Comput. Law Secur. Rev.*, vol. 46, 2022, doi: 10.1016/j.clsr.2022.105725.

[106] Paul Joseph and P Viswanathan, “A Comprehensive Survey and Analysis on Multi-Domain Digital Forensic Tools, Techniques and Issues,” *Res. Sq. ,Vellore Inst. Technol.*, pp. 0–25, 2022, doi: : https://doi.org/10.21203/rs.3.

[107] J. P. A. Yaacoub, H. N. Noura, O. Salman, and A. Chehab, “Advanced digital forensics and anti-digital forensics for IoT systems: Techniques, limitations and recommendations,” *Internet Things (Netherlands)arXiv2103.17028v1 [cs.CR] 31 Mar 2021 Digit.*, vol. 19, 2022, doi: 10.1016/j.iot.2022.100544.

[108] Y. D. A.-O. Alotaibi, Fahad, Arafat Al-Dhaqm, “A Conceptual Digital Forensic Investigation Model Applicable to the Drone Forensics Field,” vol. 13, no. 5, pp. 11608–11615, 2023.

[109] D. E. Perry, “Software Interconnection Models.,” *Proc. - Int. Conf. Softw. Eng.*, pp. 61–69, 1987.

[110] H. A. Müller, S. R. Tilley, M. A. Orgun, B. D. Corrie, and N. H. Madhavji, “A reverse engineering environment based on spatial & visual software interconnection models,” *Proc. 5th ACM SIGSOFT Symp. Softw. Dev. Environ. SDE 1992*, pp. 88–98, 1992, doi: 10.1145/142868.143755.

[111] H. A. Müller, M. A. Orgun, S. R. Tilley, and J. S. Uhl, “A reverse-engineering approach to subsystem structure identification,” *J. Softw. Maint. Res. Pract.*, vol. 5, no. 4, pp. 181–204, 1993, doi: 10.1002/smr.4360050402.

[112] C. Collberg, C. Thomborson, and D. Low, “Breaking abstractions and unstructuring data structures,” *Proc. IEEE Int. Conf. Comput. Lang.*, pp. 28–38, 1998, doi: 10.1109/ICCL.1998.674154.

[113] L. O. Brien, “Architecture Reconstruction Guidelines,” *Softw. Eng. Inst.*, vol. 3, no. August, pp. 1–41, 2001, doi: F19628-00-C003.

[114] E. Stroulia and T. Systä, “Dynamic analysis for reverse engineering and program understanding,” *ACM SIGAPP Appl. Comput. Rev.*, vol. 10, no. 1, pp. 8–17, 2002, doi: 10.1145/568235.568237.

[115] C. S. Collberg, C. Thomborson, and S. Member, “ObfuscationÐTools for Software Protection,” *Computer (Long. Beach. Calif).*, vol. 28, no. 8, pp. 735–746, 2002, [Online]. Available: http://citeseer.nj.nec.com/collberg02watermarking.html

[116] T. Eisenbarth, R. Koschke, and D. Simon, “Locating features in source code,” *IEEE Trans. Softw. Eng.*, vol. 29, no. 3, pp. 210–224, 2003, doi: 10.1109/TSE.2003.1183929.

[117] A. Marcus, A. Sergeyev, V. Rajlieh, and J. I. Maletic, “An information retrieval approach to concept location in source code,” *Proc. - Work. Conf. Reverse Eng. WCRE*, pp. 214–223, 2004, doi: 10.1109/WCRE.2004.10.

[118] E. Eilam and E. Chikofsky, *Reversing Secrets of reverse engineering*, vol. 5, no. 3. Wilely Publishing, 2005. [Online]. Available:

https://books.google.co.za/books?hl=en&lr=&id=_78HnPPRU_oC&oi=fnd&pg=PT7&dq =malware+reverse+engineering+&ots=EN4KNrvUZk&sig=UUmKGI69pv7CR1Nh25__ Z4V3Zs4#v=onepage&q=malware reverse engineering&f=false

[119] T. Dybå, B. A. Kitchenham, and M. Jorgensen, "Evidence-based software engineering for practitioners," *IEEE Softw.*, vol. 22, no. 1, pp. 58–65, 2005, doi: 10.1109/MS.2005.6.

[120] A. Aggarwal and P. Jalote, "Integrating static and dynamic analysis for detecting vulnerabilities," *Proc. - Int. Comput. Softw. Appl. Conf.*, vol. 1, pp. 343–350, 2006, doi: 10.1109/COMPSAC.2006.55.

[121] G. Canfora and M. Di Penta, "New frontiers of reverse engineering," *FoSE 2007 Futur. Softw. Eng.*, no. June, pp. 326–341, 2007, doi: 10.1109/FOSE.2007.15.

[122] Y. Song, M. E. Locasto, A. Stavrou, A. D. Keromytis, and S. J. Stolfo, "On the infeasibility of modeling polymorphic shellcode," *Proc. ACM Conf. Comput. Commun. Secur.*, pp. 541–551, 2007, doi: 10.1145/1315245.1315312.

[123] A. Czeskis, D. J. St Hilaire, K. Koscher, S. D. Gribble, T. Kohno, and B. Schneier, "Defeating encrypted and deniable file systems: TrueCrypt v5.1a and the case of the tattling OS and applications," in *HotSec 2008 - 3rd USENIX Workshop on Hot Topics in Security*, 2008, pp. 1–7.

[124] D. Page, A. Koschan, and M. Abidi, "Methodologies and Techniques for Reverse Engineering–The Potential for Automation with 3-D Laser Scanners," *Springer Ser. Adv. Manuf.*, no. September 2014, pp. 11–32, 2008, doi: 10.1007/978-1-84628-856-2_2.

[125] J. Larsen, *Rverser Engineering with IDA Pro*, Third Edit. US: Elsevier, 2008.

[126] C. Altheide, C. Merloni, and S. Zanero, "A methodology for the repeatable forensic analysis of encrypted drives," *Proc. 1st Eur. Work. Syst. Secur. EUROSEC'08*, no. March 2008, pp. 22–26, 2008, doi: 10.1145/1355284.1355289.

[127] N. McGrath, P. Gladyshev, T. Kechadi, and J. Carthy, "Investigating encrypted material," *Lect. Notes Inst. Comput. Sci. Soc. Telecommun. Eng.*, vol. 8 LNICST, pp. 29–35, 2009, doi: 10.1007/978-3-642-02312-5_4.

[128] E. J. Schwartz and T. Avgerinos, "All you ever wanted to know about dynamic taint analysis forward symbolic execution (but might have been afraid to ask)," vol. 6, no. 5, pp. 1–5, 2010.

[129] Z. Lin, X. Zhang, D. Xu, and W. Lafayette, "REWARDS: Automatic Reverse Engineering of Data Structures from Binary Execution," *Proc. 17th Annu. Netw. Distrib. Syst. Secur. Symp.*, pp. 1–20, 2010, [Online]. Available: http://www.isoc.org/isoc/conferences/ndss/10/pdf/23.pdf

[130] R. Srinivasan, P. Dasgupta, V. Iyer, A. Kanitkar, S. Sanjeev, and J. Lodhia, "A multi-factor approach to securing software on client computing platforms," *Proc. - Soc. 2010 2nd IEEE Int. Conf. Soc. Comput. PASSAT 2010 2nd IEEE Int. Conf. Privacy, Secur. Risk Trust*, pp. 993–998, 2010, doi: 10.1109/SocialCom.2010.147.

[131] D. S. and A. Singh, "An Effective Technique for Data Security in Modern Cryptosystem," *Int. J. Inf. Technol.*, vol. 2, no. 1, pp. 189–194, 2010.

[132] M. Stamp and T. Cipresso, *An introduction to software reverse engineering*, no. March. Springer Paris, 2010.

[133] H. A. Müller, K. Wong, and S. R. Tilley, "Understanding Software Systems Using Reverse Engineering Technology," *Object-Oriented Technol. Database Softw. Syst.*, pp. 240–252, 2000, doi: 10.1142/9789812831163_0016.

[134] C. Treude, F. F. Filho, M. A. Storey, and M. Salois, "An exploratory study of software reverse engineering in a security context," *Proc. - Work. Conf. Reverse Eng. WCRE*, pp. 184–188, 2011, doi: 10.1109/WCRE.2011.30.

[135] S. She, R. Lotufo, T. Berger, A. Wąsowski, and K. Czarnecki, "Reverse engineering feature models," *Proc. - Int. Conf. Softw. Eng.*, pp. 461–470, 2011, doi: 10.1145/1985793.1985856.

[136] G. Canfora, M. Di Penta, and L. Cerulo, "Achievements and challenges in software reverse engineering," *Commun. ACM*, vol. 54, no. 4, pp. 142–151, 2011, doi: 10.1145/1924421.1924451.

[137] H. Wang, Y. Wang, T. Taleb, and X. Jiang, "Editorial: Special issue on security and privacy in network computing," *World Wide Web*, vol. 23, no. 2, pp. 951–957, 2020, doi: 10.1007/s11280-019-00704-x.

[138] J. Antunes, N. Neves, and P. Verissimo, "Reverse engineering of protocols from network traces," *Proc. - Work. Conf. Reverse Eng. WCRE IEEE*, pp. 169–178, 2011, doi: 10.1109/WCRE.2011.28.

[139] I. Jozwiak, M. Kedziora, and A. Melinska, "Theoretical and practical aspects of encrypted

containers detection - Digital forensics approach," *Adv. Intell. Soft Comput.*, vol. 97, pp. 75–85, 2011, doi: 10.1007/978-3-642-21393-9_6.

[140] A. Mathematics, "Software Reverse Engineering: Achievements and Challenges," *IEEE Stand. Softw. Maint.*, pp. 1–23, 2016.

[141] H. M., J. Kraft, and H. A., "Software Reverse Engineering in the Domain of Complex Embedded Systems," in *Reverse Engineering - Recent Advances and Applications*, no. May, 2012. doi: 10.5772/33586.

[142] S. N, "A Thorough Investigation on Software Protection Techniques against Various Attacks," *Bonfring Int. J. Softw. Eng. Soft Comput.*, vol. 2, no. 3, pp. 10–15, 2012, doi: 10.9756/bijsesc.10030.

[143] M. Chlumecký, "Methods and Approaches for Reverse Engineering," *J. Digit. Forensics, Secur.*, vol. 4, no. 3, pp. 1–7, 2012.

[144] Y. Tao, Y. Dang, T. Xie, D. Zhang, and S. Kim, "How do software engineers understand code changes? - An exploratory study in industry," *Proc. ACM SIGSOFT 20th Int. Symp. Found. Softw. Eng. FSE 2012*, 2012, doi: 10.1145/2393596.2393656.

[145] S. Rekhis and N. Boudriga, "A system for formal digital forensic investigation aware of anti-forensic attacks," in *IEEE Transactions on Information Forensics and Security*, 2012, vol. 7, no. 2, pp. 635–650. doi: 10.1109/TIFS.2011.2176117.

[146] Apoorva and Y. Kumar, "Comparative Study of Different Symmetric Key Cryptography Algorithms," *Int. J. Appl. or Innov. Eng. Manag.*, vol. 2, no. 7, pp. 204–206, 2013.

[147] I. Jóźwiak, M. Kedziora, and A. Melińska, "Methods for detecting and analyzing hidden FAT32 volumes created with the use of cryptographic tools," *Adv. Intell. Syst. Comput.*, vol. 224, pp. 237–244, 2013, doi: 10.1007/978-3-319-00945-2_21.

[148] T. Thongkamwitoon, "Digital forensic techniques for the reverse engineering of image acquisition chains," pp. 1–120, 2014, [Online]. Available: http://www.commsp.ee.ic.ac.uk/~pld/group/PhDThesis_Thirapiroon14.pdf

[149] J. V. Ahmet Balci , Dan Ungureanu, "Properties of AdeABC and AdeIJK efflux systems of Acinetobacter baumannii compared with those of the AcrAB-TolC system of Escherichia coli," *Antimicrob. Agents Chemother.*, vol. 58, no. 12, pp. 7250–7257, 2014, doi: 10.1128/AAC.03728-14.

[150] L. Zhang, Y. Zhou, and J. Fan, “The forensic analysis of encrypted Truecrypt volumes,” *PIC 2014 - Proc. 2014 IEEE Int. Conf. Prog. Informatics Comput.*, pp. 405–409, 2014, doi: 10.1109/PIC.2014.6972366.

[151] H. Gonzalez, A. A. Kadir, N. Stakhanova, N. Alzahrani, and A. A. Ghorbani, “Exploring reverse engineering symptoms in android apps,” *Proc. 8th Eur. Work. Syst. Secur. EuroSec 2015*, 2015, doi: 10.1145/2751323.2751330.

[152] J. Narayan, S. K. Shukla, and T. C. Clancy, “A survey of automatic protocol reverse engineering tools,” *ACM Comput. Surv.*, vol. 48, no. 3, 2015, doi: 10.1145/2840724.

[153] A. Kaplan, H. Cetinturk, K. Celebioglu, and S. Aradag, “Reverse Engineering Design of a Hydraulic Turbine Runner,” vol. II, pp. 4–7, 2015.

[154] K. Lim, Y. Jeong, S. J. Cho, M. Park, and S. Han, “An android application protection scheme against dynamic reverse engineering attacks,” *J. Wirel. Mob. Networks, Ubiquitous Comput. Dependable Appl.*, vol. 7, no. 3, pp. 40–52, 2016.

[155] Y.-L. Cheng *et al.*, “Comparative Study of Information Security in Mobile Operating Systems: Android and Apple iOS,” *Intech*, vol. 11, no. tourism, p. 13, 2016, [Online]. Available: https://www.intechopen.com/books/advanced-biometric-technologies/liveness-detection-in-biometrics

[156] J. Narayan, S. K. Shukla, and T. C. Clancy, “A survey of automatic protocol reverse engineering tools,” *ACM Computing Surveys*, vol. 48, no. 3. 2015. doi: 10.1145/2840724.

[157] T. Diamantopoulos, K. Thomopoulos, and A. Symeonidis, “Reusability-aware Recommendations of Source Code Components,” pp. 488–491, 2016, doi: 10.1145/2901739.2903492.

[158] D. Version, *Reverse Engineering Source Code:*, Third Edit. Singapoore: Springer Singapore, 2017. [Online]. Available: http://dare.uva.nl

[159] M. Kedziora, Y. W. Chow, and W. Susilo, “Defeating plausible deniability of veracrypt hidden operating systems,” *Commun. Comput. Inf. Sci.*, vol. 719, pp. 3–13, 2017, doi: 10.1007/978-981-10-5421-1_1.

[160] Z. Mushtaq, G. Rasool, and B. Shehzad, “Multilingual Source Code Analysis: A Systematic Literature Review,” *IEEE Access*, vol. 5, pp. 11307–11336, 2017, doi: 10.1109/ACCESS.2017.2710421.

[161] A. Zuck, U. Shriki, D. E. Porter, and D. Tsafrir, “Preserving Hidden Data with an Ever-Changing Disk,” *Proc. Work. Hot Top. Oper. Syst. - HOTOS*, vol. Part F1293, pp. 50–55, 2017, doi: 10.1145/3102980.3102989.

[162] S. Senthivel, S. Dhungana, H. Yoo, I. Ahmed, and V. Roussev, “Denial of Engineering Operations A acks in Industrial Control Systems,” pp. 319–329, 2018.

[163] M. Werner, B. Lippmann, J. Baehr, and H. Grab, “Reverse engineering of cryptographic cores by structural interpretation through graph analysis,” *2018 IEEE 3rd Int. Verif. Secur. Work. IVSW 2018*, pp. 13–18, 2018, doi: 10.1109/IVSW.2018.8494896.

[164] B. D. Sija, Y. Goo, K. Shim, H. Hasanova, and M. Kim, “A Survey of Automatic Protocol Reverse Engineering Approaches , Methods , and Tools on the Inputs and Outputs View,” vol. 2018, 2018.

[165] T. Göbel and H. Baier, “Anti-forensic capacity and detection rating of hidden data in the ext4 filesystem,” *IFIP Adv. Inf. Commun. Technol.*, vol. 532, pp. 87–110, 2018, doi: 10.1007/978-3-319-99277-8_6.

[166] C. T. Lijun Zhang, Xiaoyan Deng, “An Extensive Analysis of truecrypt Encryption Forensics,” pp. 1–6, 2019, doi: 10.1145/3331453.3361328.

[167] O. Shwartz, Y. Mathov, M. Bohadana, Y. Elovici, and Y. Oren, “Reverse Engineering IoT Devices: Effective Techniques and Methods,” *IEEE Internet Things J.*, vol. 5, no. 6, pp. 4965–4976, 2018, doi: 10.1109/JIOT.2018.2875240.

[168] O. Or-Meir, N. Nissim, Y. Elovici, and L. Rokach, “Dynamic malware analysis in the modern era—A state of the art survey,” *ACM Comput. Surv.*, vol. 52, no. 5, 2019, doi: 10.1145/3329786.

[169] Q. U. Ain, W. H. Butt, M. W. Anwar, F. Azam, and B. Maqbool, “A Systematic Review on Code Clone Detection,” *IEEE Access*, vol. 7, pp. 86121–86144, 2019, doi: 10.1109/ACCESS.2019.2918202.

[170] F. O. F. Informatics, “Encrypted disk unmount using personalized audio instruction,” Masaryk University Faculty of Informatics, 2019.

[171] A. S. Rahmawati and R. P. Dewi, “An evaluation and comparison of random.org and some commonly used generators.,” *Pengaruh Pengguna. Pasta Labu Kuning (Cucurbita Moschata) Untuk Substitusi Tepung Terigu Dengan Penambahan Tepung Angkak Dalam*

Pembuatan Mie Kering, pp. 274–282, 2020.

[172] D. Votipka, S. M. Rabin, K. Micinski, J. S. Foster, and M. M. Mazurek, "An observational investigation of reverse engineers' processes," *Proc. 29th USENIX Secur. Symp.*, pp. 1875–1892, 2020.

[173] W. C. Henry and G. L. Peterson, "Exploring Provenance Needs in Software Reverse Engineering," *Dep. Electr. Comput. Eng. Air Force Inst. Technol. Wright-Patterson AFB, USA*, pp. 57–65, 2020, [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9133693

[174] K. Liu *et al.*, "On Manually Reverse Engineering Communication Protocols of Linux-Based IoT Systems," *IEEE Internet Things J.*, vol. 8, no. 8, pp. 6815–6827, 2021, doi: 10.1109/JIOT.2020.3036232.

[175] K. Liu, M. Yang, Z. Ling, H. Yan, and Y. Zhang, "Protocols of Linux-Based IoT Systems," vol. 8, no. 8, pp. 6815–6827, 2021.

[176] H. Rathore and M. Sewak, "Malware Analysis and Detection," *ACM Int. Conf. Proceeding Ser.*, vol. 10, no. 2, pp. 2–5, 2022, doi: 10.1145/3564121.3564809.

[177] G. Long and Z. Zhang, "Deep Encrypted Traffic Detection: An Anomaly Detection Framework for Encryption Traffic Based on Parallel Automatic Feature Extraction," *Comput. Intell. Neurosci.*, vol. 2023, pp. 1–12, 2023, doi: 10.1155/2023/3316642.

[178] Z. H. Abdullahi, S. K. Singh, and M. Hasan, "Software Reverse Engineering Techniques for Evaluating Anti-Forensic Encryption Tools: A Framework Development and Analysis," *Int. J. Intell. Syst. Appl. Eng.*, vol. 12, no. 1s, pp. 620–632, 2024.

[179] N. Bibi, T. Rana, A. Maqbool, F. Afzal, A. Akgül, and M. De la Sen, "An Intelligent Platform for Software Component Mining and Retrieval," *Sensors (Basel).*, vol. 23, no. 1, pp. 1–24, 2023, doi: 10.3390/s23010525.

[180] A. Nurgaliyev and H. Wang, "Analysis of reverse engineering," *2023 15th Int. Conf. Adv. Comput. Intell. ICACI 2023*, pp. 1–7, 2023, doi: 10.1109/ICACI58115.2023.10146175.

[181] S. Arif and M. M. Kumar, "Cyber Safety Analysis Using Reverse Engineering Syed," *Int. J. Res. Publ. Rev. J. homepage www.ijrpr.com ISSN 2582-7421 Cyber*, vol. 4, no. 1, pp. 399–403, 2023.

[182] Z. H. Abdullahi and S. K. Singh, “A Conceptual and Technical Perspective of Reverse Engineering In Digital forensic,” in *2nd International Conference on Recent Development in Engineering ,Sciences, and Management Goverment Engineering College Bharatpur, Rjasthan,* 2023, no. April. [Online]. Available: ISBN; 978-9391535-43-8

[183] Infosec, “Computer Forensics: Media & File System Forensics [Updated 2019] - Infosec Resources.” p. https://resources.infosecinstitute.com/category/co, 2019. [Online]. Available: https://resources.infosecinstitute.com/topic/computer-forensics-media-file-system-forensics/

[184] A. Guide, “Comodo Forensic Analysis Comodo Forensic Analysis - Admin Guide,” *Comodo Secur. Solut.*, vol. 2.o.12018, 2020.

[185] B. Lutkevich, “What is Reverse-engineering? How Does It Work?,” *TechTarget*. 2021. [Online]. Available: https://searchsoftwarequality.techtarget.com/definition/reverse-engineering

[186] San Francisco, “The Evolution of Reverse Engineering_ From Manual Reconstruction to Automated Disassembling _ Apriorit,” in *RSA*, 2021, p. [Online]. Available: https://searchsoftwarequality.

[187] Bruce Morey, “Reverse Engineering Proliferates_ New Applications Emerge as Technology Improves _ SME Media,” in *SME Media*, 2021, p. [Online]. Available: https://searchsoftwarequality.

[188] B. Carrier, “File Systemnsic Analysis,” *Pearson Educ. Inc.*, vol. 4, p. 511, 2005.

[189] D. A. Ford, R. J. T. Morris, and A. E. Bell, “Redundant Arrays of Independent Libraries (RAIL): The StarFish tertiary storage system,” *Parallel Comput.*, vol. 24, no. 1, pp. 45–64, 1998, doi: 10.1016/s0167-8191(97)00116-6.

[190] Y. K. Gupta, “Systematic Digital Forensic Investigation Model,” *Int. J. Comput. Sci. Secur. (IJCSS),* vol. 5, no. January 2011, 2016.

[191] G. C. Kessler, “Anti-forensics and the digital investigator,” *Proc. 5th Aust. Digit. Forensics Conf.*, pp. 1–7, 2007.

[192] T. Foundation, “TrueCrypt User Guide,” *System*, vol. 6, no. 5, pp. 1–110, 2014.

[193] L. Zhang, Y. Zhou, and J. Fan, “The forensic analysis of encrypted Truecrypt volumes,”

PIC 2014 - Proc. 2014 IEEE Int. Conf. Prog. Informatics Comput., pp. 405–409, 2014, doi: 10.1109/PIC.2014.6972366.

[194] A. Jain and G. S. Chhabra, “Anti-forensics techniques: An analytical review,” *2014 7th Int. Conf. Contemp. Comput. IC3 2014*, no. August 2014, pp. 412–418, 2014, doi: 10.1109/IC3.2014.6897209.

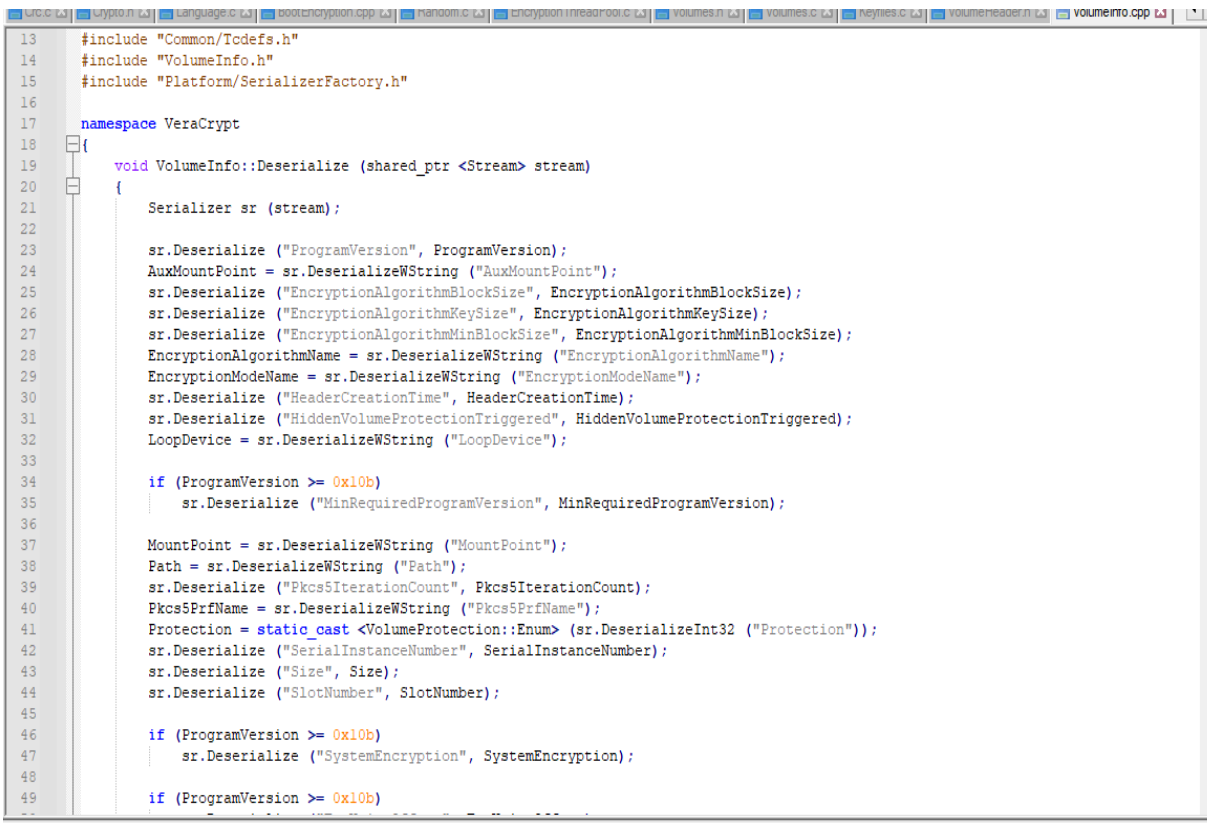
[195] H. Berghel, “Hiding Data, Forensics, and Anti-Forensics,” *Commun. ACM*, vol. 50, no. 4, pp. 15–20, 2007.

[196] Y. Desmedt, “Deniable Encryption,” *Encycl. Cryptogr. Secur.*, pp. 322–323, 2011, doi: 10.1007/978-1-4419-5906-5_316.

[197] P. Rogaway, “Efficient instantiations of tweakable blockciphers and refinements to modes OCB and PMAC,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 3329, pp. 16–31, 2004, doi: 10.1007/978-3-540-30539-2_2.

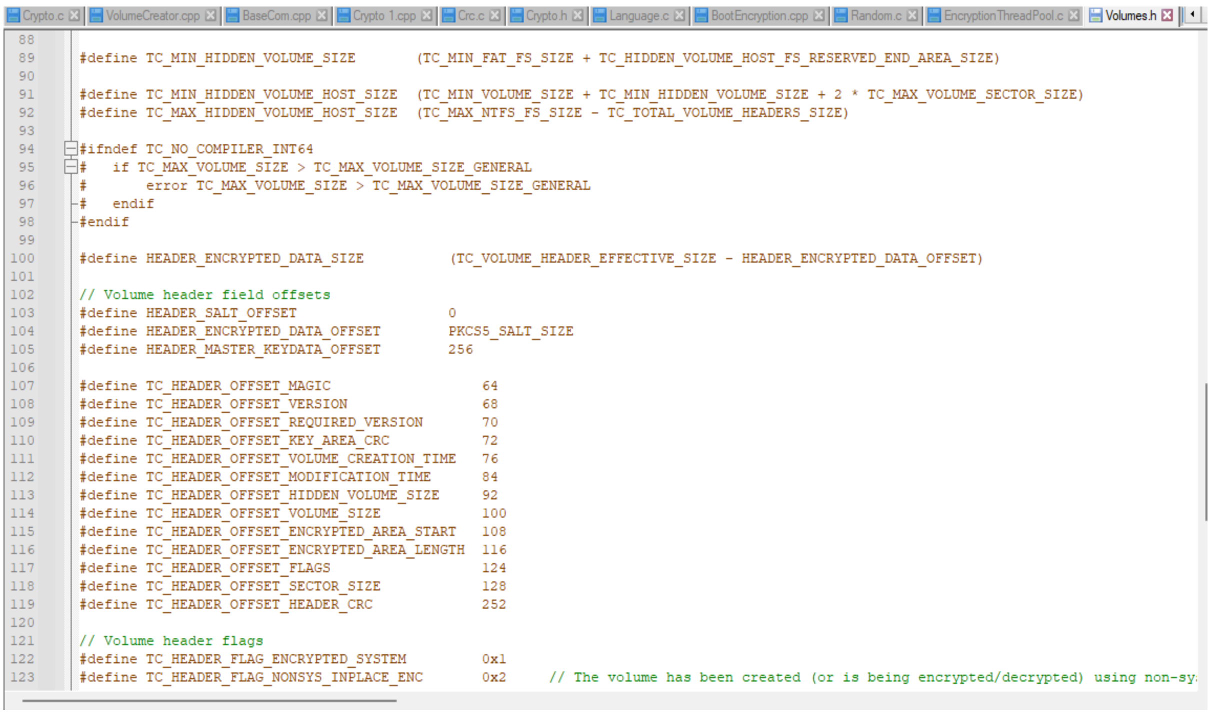
[198] M. Dworkin, “Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication,” *NIST Spec. Publ. 800-38B*, p. 25, 2005, [Online]. Available: http://csrc.nist.gov/publications/nistpubs/800-38B/SP_800-38B.pdf

APPENDEX A



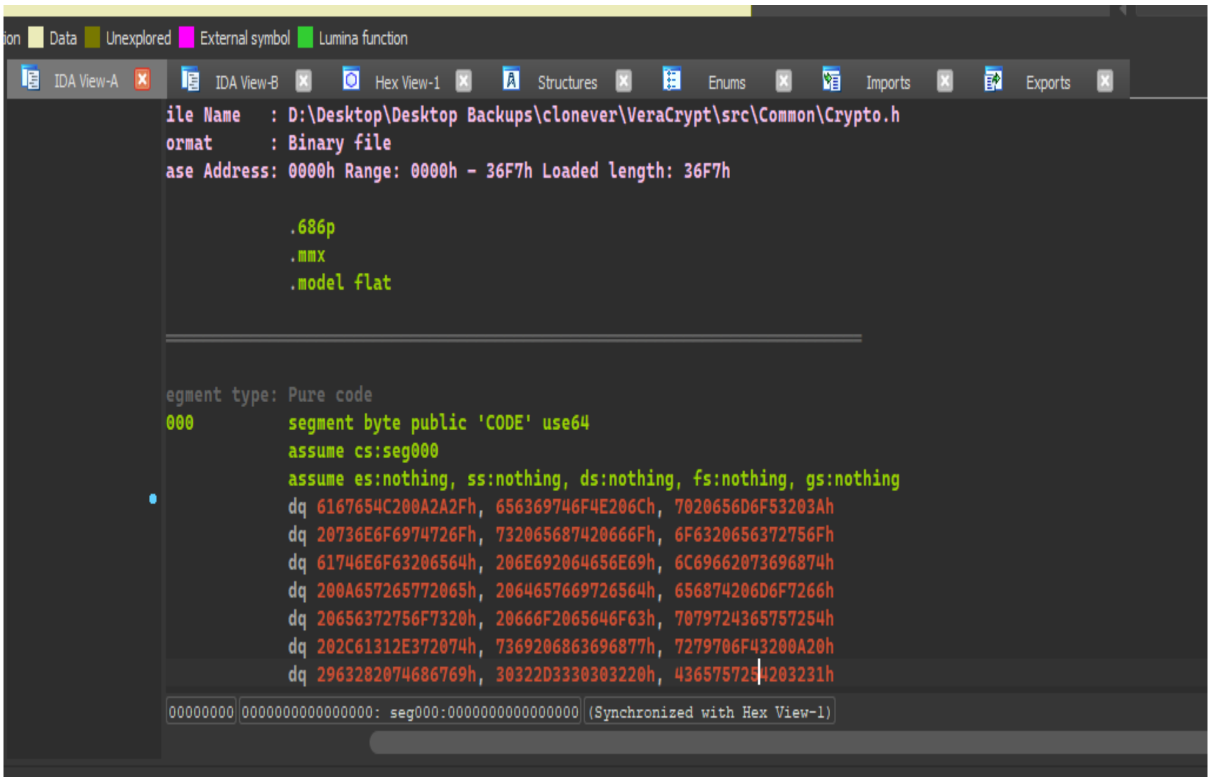
```
#include "Common/Tcdefs.h"
#include "VolumeInfo.h"
#include "Platform/SerializerFactory.h"

namespace VeraCrypt
{
    void VolumeInfo::Deserialize (shared_ptr <Stream> stream)
    {
        Serializer sr (stream);

        sr.Deserialize ("ProgramVersion", ProgramVersion);
        AuxMountPoint = sr.DeserializeWString ("AuxMountPoint");
        sr.Deserialize ("EncryptionAlgorithmBlockSize", EncryptionAlgorithmBlockSize);
        sr.Deserialize ("EncryptionAlgorithmKeySize", EncryptionAlgorithmKeySize);
        sr.Deserialize ("EncryptionAlgorithmMinBlockSize", EncryptionAlgorithmMinBlockSize);
        EncryptionAlgorithmName = sr.DeserializeWString ("EncryptionAlgorithmName");
        EncryptionModeName = sr.DeserializeWString ("EncryptionModeName");
        sr.Deserialize ("HeaderCreationTime", HeaderCreationTime);
        sr.Deserialize ("HiddenVolumeProtectionTriggered", HiddenVolumeProtectionTriggered);
        LoopDevice = sr.DeserializeWString ("LoopDevice");

        if (ProgramVersion >= 0x10b)
            sr.Deserialize ("MinRequiredProgramVersion", MinRequiredProgramVersion);

        MountPoint = sr.DeserializeWString ("MountPoint");
        Path = sr.DeserializeWString ("Path");
        sr.Deserialize ("Pkcs5IterationCount", Pkcs5IterationCount);
        Pkcs5PrfName = sr.DeserializeWString ("Pkcs5PrfName");
        Protection = static_cast <VolumeProtection::Enum> (sr.DeserializeInt32 ("Protection"));
        sr.Deserialize ("SerialInstanceNumber", SerialInstanceNumber);
        sr.Deserialize ("Size", Size);
        sr.Deserialize ("SlotNumber", SlotNumber);

        if (ProgramVersion >= 0x10b)
            sr.Deserialize ("SystemEncryption", SystemEncryption);

        if (ProgramVersion >= 0x10b)
```

Figure A 1 VeraCrypt Volume Information



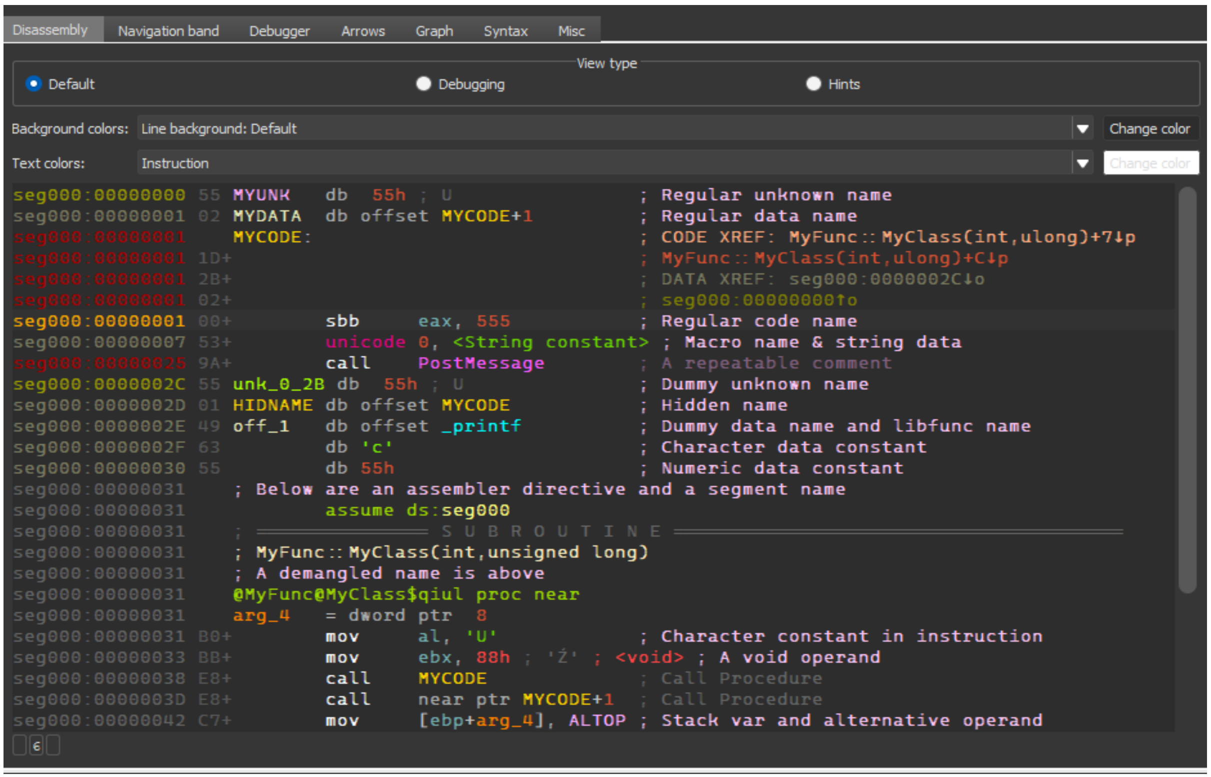
```

#define TC_MIN_HIDDEN_VOLUME_SIZE       (TC_MIN_FAT_FS_SIZE + TC_HIDDEN_VOLUME_HOST_FS_RESERVED_END_AREA_SIZE)

#define TC_MIN_HIDDEN_VOLUME_HOST_SIZE  (TC_MIN_VOLUME_SIZE + TC_MIN_HIDDEN_VOLUME_SIZE + 2 * TC_MAX_VOLUME_SECTOR_SIZE)
#define TC_MAX_HIDDEN_VOLUME_HOST_SIZE  (TC_MAX_NTFS_FS_SIZE - TC_TOTAL_VOLUME_HEADERS_SIZE)

#ifndef TC_NO_COMPILER_INT64
#   if TC_MAX_VOLUME_SIZE > TC_MAX_VOLUME_SIZE_GENERAL
#       error TC_MAX_VOLUME_SIZE > TC_MAX_VOLUME_SIZE_GENERAL
#   endif
#endif

#define HEADER_ENCRYPTED_DATA_SIZE          (TC_VOLUME_HEADER_EFFECTIVE_SIZE - HEADER_ENCRYPTED_DATA_OFFSET)

// Volume header field offsets
#define HEADER_SALT_OFFSET                  0
#define HEADER_ENCRYPTED_DATA_OFFSET        PKCS5_SALT_SIZE
#define HEADER_MASTER_KEYDATA_OFFSET        256

#define TC_HEADER_OFFSET_MAGIC                  64
#define TC_HEADER_OFFSET_VERSION                68
#define TC_HEADER_OFFSET_REQUIRED_VERSION       70
#define TC_HEADER_OFFSET_KEY_AREA_CRC           72
#define TC_HEADER_OFFSET_VOLUME_CREATION_TIME   76
#define TC_HEADER_OFFSET_MODIFICATION_TIME      84
#define TC_HEADER_OFFSET_HIDDEN_VOLUME_SIZE     92
#define TC_HEADER_OFFSET_VOLUME_SIZE            100
#define TC_HEADER_OFFSET_ENCRYPTED_AREA_START   108
#define TC_HEADER_OFFSET_ENCRYPTED_AREA_LENGTH  116
#define TC_HEADER_OFFSET_FLAGS                  124
#define TC_HEADER_OFFSET_SECTOR_SIZE            128
#define TC_HEADER_OFFSET_HEADER_CRC             252

// Volume header flags
#define TC_HEADER_FLAG_ENCRYPTED_SYSTEM         0x1
#define TC_HEADER_FLAG_NONSYS_INPLACE_ENC       0x2     // The volume has been created (or is being encrypted/decrypted) using non-sy
```

Figure A 2, VeraCrypt Header Encrypted Data Size Information



```
ile Name   : D:\Desktop\Desktop Backups\clonever\VeraCrypt\src\Common\Crypto.h
ormat      : Binary file
ase Address: 0000h Range: 0000h - 36F7h Loaded length: 36F7h

                .686p
                .mmx
                .model flat


egment type: Pure code
000             segment byte public 'CODE' use64
                assume cs:seg000
                assume es:nothing, ss:nothing, ds:nothing, fs:nothing, gs:nothing
                dq 6167654C200A2A2Fh, 656369746F4E206Ch, 7020656D6F53203Ah
                dq 20736E6F6974726Fh, 732065687420666Fh, 6F6320656372756Fh
                dq 61746E6F63206564h, 206E692064656E69h, 6C69662073696874h
                dq 200A657265772065h, 2064657669726564h, 656874206D6F7266h
                dq 20656372756F7320h, 20666F2065646F63h, 7079724365757254h
                dq 202C61312E372074h, 7369206863696877h, 7279706F43200A20h
                dq 2963282074686769h, 30322D3330303220h, 4365757254203231h
00000000 0000000000000000: seg000:0000000000000000 (Synchronized with Hex View-1)
```

Figure A 3 VeraCrypt Hex View



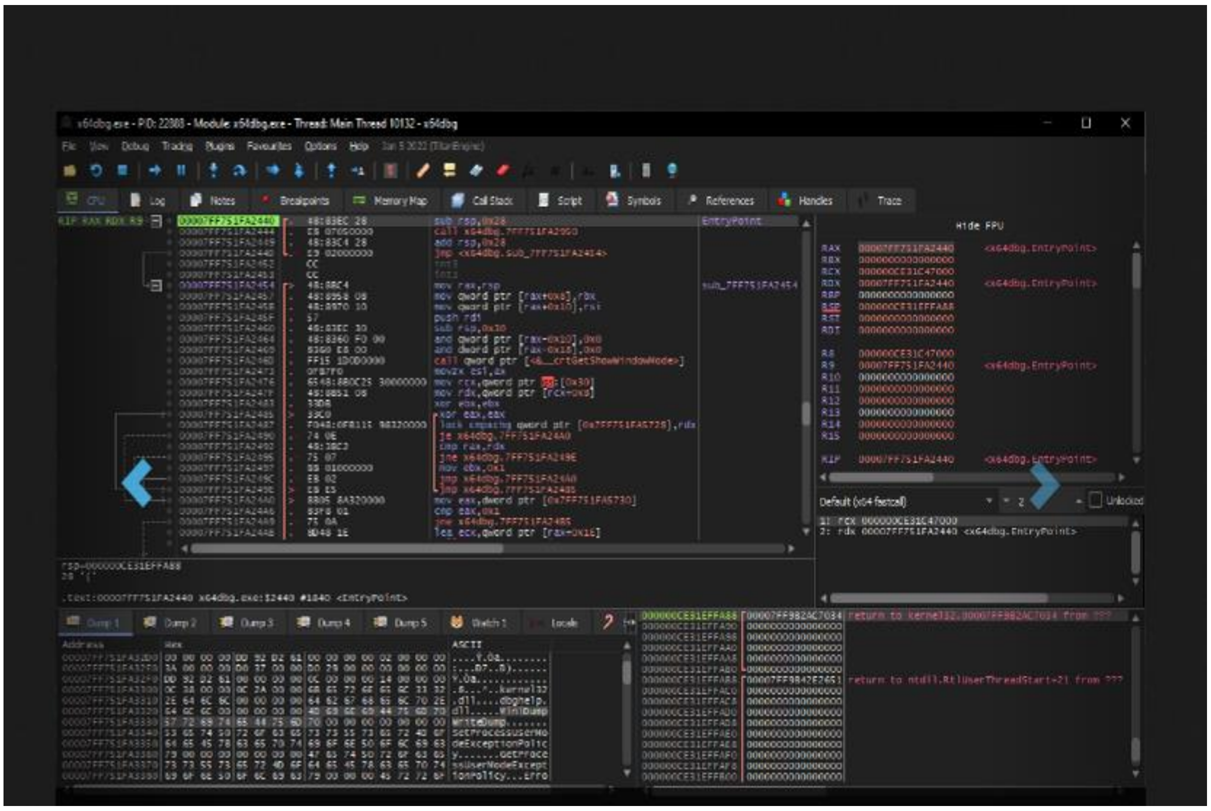
View type: [✓ Default] Debugging Hints

Background colors: Line background: Default — Change color

Text colors: Instruction — Change color

```
seg000:00000000 55 MYUNK   db  55h ; U                 ; Regular unknown name
seg000:00000001 02 MYDATA  db offset MYCODE+1          ; Regular data name
seg000:00000001    MYCODE:                             ; CODE XREF: MyFunc::MyClass(int,ulong)+7↓p
seg000:00000001 1D+                                    ; MyFunc::MyClass(int,ulong)+C↓p
seg000:00000001 2B+                                    ; DATA XREF: seg000:0000002C↓o
seg000:00000001 02+                                    ; seg000:00000000↑o
seg000:00000001 00+        sbb     eax, 555            ; Regular code name
seg000:00000007 53+        unicode 0, <String constant> ; Macro name & string data
seg000:00000025 9A+        call    PostMessage         ; A repeatable comment
seg000:0000002C 55 unk_0_2B db  55h ; U                ; Dummy unknown name
seg000:0000002D 01 HIDNAME db offset MYCODE            ; Hidden name
seg000:0000002E 49 off_1   db offset _printf           ; Dummy data name and libfunc name
seg000:0000002F 63         db 'c'                      ; Character data constant
seg000:00000030 55         db 55h                      ; Numeric data constant
seg000:00000031    ; Below are an assembler directive and a segment name
seg000:00000031            assume ds:seg000
seg000:00000031    ; =============== S U B R O U T I N E =======================================
seg000:00000031    ; MyFunc::MyClass(int,unsigned long)
seg000:00000031    ; A demangled name is above
seg000:00000031    @MyFunc@MyClass$qiul proc near
seg000:00000031    arg_4    = dword ptr  8
seg000:00000031 B0+        mov     al, 'U'             ; Character constant in instruction
seg000:00000033 BB+        mov     ebx, 88h ; 'Ź' ; <void> ; A void operand
seg000:00000038 E8+        call    MYCODE              ; Call Procedure
seg000:0000003D E8+        call    near ptr MYCODE+1   ; Call Procedure
seg000:00000042 C7+        mov     [ebp+arg_4], ALTOP  ; Stack var and alternative operand
```

Figure A 4 VeraCrypt IDA View


x64dbg.exe - PID: 22888 - Module: x64dbg.exe - Thread: Main Thread 10132 - x64dbg
File View Debug Tracing Plugins Favourites Options Help Jan 5 2022 (TitanEngine)
CPU Log Notes Breakpoints Memory Map Call Stack Script Symbols References Handles Trace
Hide FPU
Default (x64 fastcall)
Unlocked
Dump 1 Dump 2 Dump 3 Dump 4 Dump 5 Watch 1 Locals

```
00007FF751FA2440  48:83EC 28            sub rsp,0x28                                   EntryPoint
00007FF751FA2444  E8 07050000           call x64dbg.7FF751FA2950
00007FF751FA2449  48:83C4 28            add rsp,0x28
00007FF751FA244D  E9 02000000           jmp <x64dbg.sub_7FF751FA2454>
00007FF751FA2452  CC                    int3
00007FF751FA2453  CC                    int3
00007FF751FA2454  48:8BC4               mov rax,rsp                                    sub_7FF751FA2454
00007FF751FA2457  48:8958 08            mov qword ptr [rax+0x8],rbx
00007FF751FA245B  48:8970 10            mov qword ptr [rax+0x10],rsi
00007FF751FA245F  57                    push rdi
00007FF751FA2460  48:83EC 30            sub rsp,0x30
00007FF751FA2464  48:8360 F0 00         and qword ptr [rax-0x10],0x0
00007FF751FA2469  8360 E8 00            and dword ptr [rax-0x18],0x0
00007FF751FA246D  FF15 1D0D0000         call qword ptr [<&__crtGetShowWindowMode>]
00007FF751FA2473  0FB7F0                movzx esi,ax
00007FF751FA2476  65:48:8B0C25 30000000 mov rcx,qword ptr gs:[0x30]
00007FF751FA247F  48:8B51 08            mov rdx,qword ptr [rcx+0x8]
00007FF751FA2483  33DB                  xor ebx,ebx
00007FF751FA2485  33C0                  xor eax,eax
00007FF751FA2487  F0:48:0FB115 98320000 lock cmpxchg qword ptr [0x7FF751FA5728],rdx
00007FF751FA2490  74 0E                 je x64dbg.7FF751FA24A0
00007FF751FA2492  48:3BC2               cmp rax,rdx
00007FF751FA2495  75 07                 jne x64dbg.7FF751FA249E
00007FF751FA2497  BB 01000000           mov ebx,0x1
00007FF751FA249C  EB 02                 jmp x64dbg.7FF751FA24A0
00007FF751FA249E  EB E5                 jmp x64dbg.7FF751FA2485
00007FF751FA24A0  8B05 8A320000         mov eax,dword ptr [0x7FF751FA5730]
00007FF751FA24A6  83F8 01               cmp eax,0x1
00007FF751FA24A9  75 0A                 jne x64dbg.7FF751FA24B5
00007FF751FA24AB  8D48 1E               lea ecx,qword ptr [rax+0x1E]

rsp=000000CE31EFFA88
28 '('

.text:00007FF751FA2440 x64dbg.exe:$2440 #1840 <EntryPoint>
```

```
RAX  00007FF751FA2440   <x64dbg.EntryPoint>
RBX  0000000000000000
RCX  000000CE31C47000
RDX  00007FF751FA2440   <x64dbg.EntryPoint>
RBP  0000000000000000
RSP  000000CE31EFFA88
RSI  0000000000000000
RDI  0000000000000000

R8   000000CE31C47000
R9   00007FF751FA2440   <x64dbg.EntryPoint>
R10  0000000000000000
R11  0000000000000000
R12  0000000000000000
R13  0000000000000000
R14  0000000000000000
R15  0000000000000000

RIP  00007FF751FA2440   <x64dbg.EntryPoint>

1: rcx 000000CE31C47000
2: rdx 00007FF751FA2440 <x64dbg.EntryPoint>
```

```
Address           Hex                                              ASCII
00007FF751FA32D0  00 00 00 00 D0 92 D2 61 00 00 00 00 02 00 00 00  ....Ò.Òa........
00007FF751FA32E0  3A 00 00 00 D0 37 00 00 D0 29 00 00 00 00 00 00  :...Ð7..Ð)......
00007FF751FA32F0  DD 92 D2 61 00 00 00 00 0C 00 00 00 14 00 00 00  Ý.Òa............
00007FF751FA3300  0C 38 00 00 0C 2A 00 00 6B 65 72 6E 65 6C 33 32  .8...*..kernel32
00007FF751FA3310  2E 64 6C 6C 00 00 00 00 64 62 67 68 65 6C 70 2E  .dll....dbghelp.
00007FF751FA3320  64 6C 6C 00 00 00 00 00 4D 69 6E 69 44 75 6D 70  dll.....MiniDump
00007FF751FA3330  57 72 69 74 65 44 75 6D 70 00 00 00 00 00 00 00  WriteDump.......
00007FF751FA3340  53 65 74 50 72 6F 63 65 73 73 55 73 65 72 4D 6F  SetProcessUserMo
00007FF751FA3350  64 65 45 78 63 65 70 74 69 6F 6E 50 6F 6C 69 63  deExceptionPolic
00007FF751FA3360  79 00 00 00 00 00 00 00 47 65 74 50 72 6F 63 65  y.......GetProce
00007FF751FA3370  73 73 55 73 65 72 4D 6F 64 65 45 78 63 65 70 74  ssUserModeExcept
00007FF751FA3380  69 6F 6E 50 6F 6C 69 63 79 00 00 00 45 72 72 6F  ionPolicy...Erro
```

```
000000CE31EFFA88  00007FF982AC7034  return to kernel32.00007FF982AC7034 from ???
000000CE31EFFA90  0000000000000000
000000CE31EFFA98  0000000000000000
000000CE31EFFAA0  0000000000000000
000000CE31EFFAA8  0000000000000000
000000CE31EFFAB0  0000000000000000
000000CE31EFFAB8  00007FF9842E2651  return to ntdll.RtlUserThreadStart+21 from ???
000000CE31EFFAC0  0000000000000000
000000CE31EFFAC8  0000000000000000
000000CE31EFFAD0  0000000000000000
000000CE31EFFAD8  0000000000000000
000000CE31EFFAE0  0000000000000000
000000CE31EFFAE8  0000000000000000
000000CE31EFFAF0  0000000000000000
000000CE31EFFAF8  0000000000000000
000000CE31EFFB00  0000000000000000
```

Figure A 5 x64dbg Windbg Tool

APPENDEX B

AxCrypt.

```
*/

#ifndef TC_HEADER_Volume_Volume

#define TC_HEADER_Volume_Volume

#include "Platform/Platform.h"
#include "Platform/StringConverter.h"
#include "EncryptionAlgorithm.h"
#include "EncryptionMode.h"
#include "Keyfile.h"
#include "VolumePassword.h"
#include "VolumeException.h"
#include "VolumeLayout.h"
namespace VeraCrypt
{
	class VolumePath
	{
	public:
		VolumePath () { }
		VolumePath (const wstring &path) { Data = path; }
		VolumePath (const FilesystemPath &path) { Data = path; }
		bool operator== (const VolumePath &other) const { return Data == other.Data; }
		bool operator!= (const VolumePath &other) const { return Data != other.Data; }
		operator FilesystemPath () const { return FilesystemPath (Data); }
		operator string () const { return StringConverter::ToSingle (Data); }
		operator wstring () const { return Data; }
		bool IsDevice () const { return FilesystemPath (Data).IsBlockDevice() || FilesystemPath
(Data).IsCharacterDevice(); }
		bool IsEmpty () const { return Data.empty(); }

		wstring GetExtension () const
		{
			if (Data.empty() || (Data.size() == 1))
```

```
				return L"";
			else
			{
				size_t pos = Data.find_last_of (L'.');
				if (pos == string::npos)
					return L"";
				return Data.substr (pos + 1);
			}
		}

		bool HasTrueCryptExtension () const
		{
			wstring sExt = GetExtension ();
			if ((sExt.size () == 2)
				&& (sExt[0] == L't' || sExt[0] == L'T')
				&& (sExt[1] == L'c' || sExt[1] == L'C')
				)
			{
				return true;
			}
			else
				return false;
		}
	protected:
		wstring Data;
	};
	typedef list <VolumePath> VolumePathList;
	struct VolumeHostType
	{
		enum Enum
		{
			Unknown,
			File,
			Device
		};
	};
	struct VolumeProtection
	{
		enum Enum
		{
			None,
			ReadOnly,
			HiddenVolumeReadOnly
		};
	};
	class Volume
	{
	public:
		Volume ();
		virtual ~Volume ();
		void Close ();
		shared_ptr <EncryptionAlgorithm> GetEncryptionAlgorithm () const;
		shared_ptr <EncryptionMode> GetEncryptionMode () const;
		shared_ptr <File> GetFile () const { return VolumeFile; }
		shared_ptr <VolumeHeader> GetHeader () const { return Header; }
		uint64 GetHeaderCreationTime () const { return Header->GetHeaderCreationTime(); }
		uint64 GetHostSize () const { return VolumeHostSize; }
		shared_ptr <VolumeLayout> GetLayout () const { return Layout; }
		VolumePath GetPath () const { return VolumeFile->GetPath(); }
		VolumeProtection::Enum GetProtectionType () const { return Protection; }
		shared_ptr <Pkcs5Kdf> GetPkcs5Kdf () const {return Header->GetPkcs5Kdf(); }
		uint32 GetSaltSize () const {return Header->GetSaltSize(); }
		size_t GetSectorSize () const {return SectorSize; }
		uint64 GetSize () const {return VolumeDataSize; }
		uint64 GetEncryptedSize () const {return EncryptedDataSize; }
		uint64 GetTopWriteOffset () const {return TopWriteOffset; }
		uint64 GetTotalDataRead () const {return TotalDataRead; }
		uint64 GetTotalDataWritten () const { return TotalDataWritten; }
		VolumeType::Enum GetType () const { return Type; }
		bool GetTrueCryptMode() const { return TrueCryptMode; }
		int GetPim() const { return Pim;}
		uint64 GetVolumeCreationTime () const {return Header->GetVolumeCreationTime(); }
		bool IsHiddenVolumeProtectionTriggered333 () const {return
HiddenVolumeProtectionTriggered; }
```

```
		bool IsInSystemEncryptionScope () const { return SystemEncryption; }
		void Open (const VolumePath &volumePath, bool preserveTimestamps, shared_ptr
<VolumePassword> password, int pim, shared_ptr <Pkcs5Kdf> kdf, bool truecryptMode, shared_ptr
<KeyfileList> keyfiles, VolumeProtection::Enum protection = VolumeProtection::None, shared_ptr
<VolumePassword> protectionPassword = shared_ptr <VolumePassword> (), int protectionPim = 0,
shared_ptr <Pkcs5Kdf> protectionKdf = shared_ptr <Pkcs5Kdf> (),shared_ptr <KeyfileList>
protectionKeyfiles = shared_ptr <KeyfileList> (), bool sharedAccessAllowed = false, VolumeType::Enum
volumeType = VolumeType::Unknown, bool useBackupHeaders = false, bool
partitionInSystemEncryptionScope = false);
		void Open (shared_ptr <File> volumeFile, shared_ptr <VolumePassword> password, int
pim, shared_ptr <Pkcs5Kdf> kdf, bool truecryptMode, shared_ptr <KeyfileList> keyfiles,
VolumeProtection::Enum protection = VolumeProtection::None, shared_ptr <VolumePassword>
protectionPassword = shared_ptr <VolumePassword> (), int protectionPim = 0, shared_ptr <Pkcs5Kdf>
protectionKdf = shared_ptr <Pkcs5Kdf> (), shared_ptr <KeyfileList> protectionKeyfiles = shared_ptr
<KeyfileList> (), VolumeType::Enum volumeType = VolumeType::Unknown, bool useBackupHeaders =
false, bool partitionInSystemEncryptionScope = false);
		void ReadSectors (const BufferPtr &buffer, uint64 byteOffset);
		void ReEncryptHeader (bool backupHeader, const ConstBufferPtr &newSalt, const
ConstBufferPtr &newHeaderKey, shared_ptr <Pkcs5Kdf> newPkcs5Kdf);
		void WriteSectors (const ConstBufferPtr &buffer, uint64 byteOffset);
		bool IsEncryptionNotCompleted () const {return EncryptionNotCompleted; }
	protected:
		void CheckProtectedRange (uint64 writeHostOffset, uint64 writeLength);
		void ValidateState () const;
		shared_ptr <EncryptionAlgorithm> EA;
		shared_ptr <VolumeHeader> Header;
		bool HiddenVolumeProtectionTriggered;
		shared_ptr <VolumeLayout> Layout;
		uint64 ProtectedRangeStart;
		uint64 ProtectedRangeEnd;
		VolumeProtection::Enum Protection;
		size_t SectorSize;
		bool SystemEncryption;
		VolumeType::Enum Type;
		shared_ptr <File> VolumeFile;
		uint64 VolumeHostSize;
		uint64 VolumeDataOffset;
```

```
			uint64 VolumeDataSize;
			uint64 EncryptedDataSize;
			uint64 TopWriteOffset;
			uint64 TotalDataRead;
			uint64 TotalDataWritten;
			bool TrueCryptMode;
			int Pim;
			bool EncryptionNotCompleted;
		private:
			Volume (const Volume &);
			Volume &operator= (const Volume &);
		};
}
#endif // TC_HEADER_Volume_Volume
```

APPENDEX C

Publication in Journal

1. Z. H. Abdullahi, S. K. Singh, and M. Hasan, “Software Reverse Engineering Techniques for Evaluating Anti-Forensic Encryption Tools: A Framework Development and Analysis,” *Int. J. Intell. Syst. Appl. Eng.*, vol. 12, no. 1s, pp. 620–632, 2024. [Scopus Indexed]

Publication in Conferences

1. Z. H. Abdullahi and S. K. Singh, “A Conceptual and Technical Perspective of Reverse Engineering In Digital forensic,” in 2nd International Conference on Recent Development in Engineering ,Sciences, and Management Goverment Engineering College Bharatpur, Rjasthan, 2023, no. April. [Online]. Available: ISBN; 978-9391535-43-8
2. The impact of Anti-forensic Techniques on Forensic Investigation Challenges, International Conference of Computer Science Engineering and Emerging Technologies (ICCS-2022) Taylor and Francis Group London. ISBN 978-1-032-52199-2 Booth100 6th International Joint Conference On Computing Sciences (Iccs-2022) [Scopus Indexed]
3. Development of Evaluation Protocol for Anti-Forensics Encryption Tool Using Software Reverse Engineering Techniques, International Conference of Computer Science Engineering and Emerging Technologies (ICCS-2022) Taylor and Francis Group London. ISBN 978-1-032-52199-2 Booth100 6th International Joint Conference On Computing Sciences (Iccs-2022) [Scopus Indexed]
4. An Overview of Anti-forensic Techniques and their Impact on Digital Forensic Analysis, 4th International online multidisciplinary Research Conference, at Osmania University Centre for international Program, Osmania University Campus Hyderabad, ISBN 978-93-86171-283