



FUD Crypter Development by Combining Symmetric & Asymmetric Cryptographic Algorithms

¹Manan Patel, ¹Rushi Padhiyar, ¹Priyanshu Negi, ²Nidhi Patel, ³Milind Purswani

^{1,2}Computer Science and Engineering, Parul University Vadodara, India

¹Student, ²Assistant Professor

³Security Engineer, Amazon San Jose, USA.

Abstract - Fully Undetectable (FUD) Crypter's, pivotal for eluding antivirus detection, necessitate advanced techniques. This study explores encryption, code obfuscation and anti-virus evasion techniques implemented in FUD Crypter development. It plays a crucial role by bypassing vigilant antivirus software. As the arms race between malware creators and security defenders intensifies, the development of FUD Crypter's demands the integration of advanced techniques. This study rigorously examines the efficacy of our FUD Crypter's design and different mechanisms engaged in the process, subjecting them to comprehensive evaluations against a spectrum of antivirus solutions. Additionally, the research delves into the intricacies of conversion processes, limitations, and the normalization characteristics of some well-known malwares. It further scrutinizes the attributes of Python-based malware through the lens of normalization curves, encompassing both size and execution time metrics. Empirical analyses contain extensive Python malware datasets and evaluations across prominent platforms. The culmination of these insights underscores the development of a robust and holistic antivirus evasion strategy.

Keywords - Encryption, Code Obfuscation, Antivirus Evasion, Malware Analysis, Cryptography, Red Team, Malware Authors, Reverse Engineering, Signature-based Detection, Dynamic Analysis, Polymorphic Malwares, Malware Detection

I. INTRODUCTION

In the ever-evolving landscape of cybersecurity, the perpetual struggle between malicious actors and defenders continues to push the boundaries of innovation. Amid this intricate dance, Fully Undetectable (FUD) Crypter's emergence has reshaped the dynamics of malware evasion. These ingenious tools serve as the vanguard for concealing the true nature of software programs from the prying eyes of increasingly sophisticated antivirus systems. As digital threats grow in complexity, the development of FUD Crypter's stands as a testament to the ingenuity required to navigate the adversarial terrain.

Cryptographic methodology, with its roots traced back to ancient times, serves as the foundation for modern cybersecurity strategies. Encryption and decryption, core cryptographic functions, play a pivotal role in safeguarding information by transforming plain text messages into unreadable ciphertext. This cryptographic dance ensures secure communication channels, shielding sensitive data from unauthorized access. Symmetric and asymmetric cryptography are two pillars of this field, where symmetric cryptography employs a shared secret key for encryption and decryption [3]. In contrast, asymmetric cryptography operates on a pair of public and private keys.

FUD Crypter's are not merely clandestine instruments; they embody a fusion of advanced techniques that blur the lines between legitimate software and concealed threats. This research delves into one of those techniques and dissects their inner workings to unveil the intricate craftsmanship that underpins their effectiveness [9]. In Crypters, Encryption is the first line of defence, enabling secure transmission while shrouding critical components in layers of cryptographic complexity. In parallel, code obfuscation techniques distort software logic, frustrating reverse engineering endeavors and amplifying the crypter's resilience.

However, the realm of FUD Crypter development transcends these individual techniques, encompassing a holistic perspective that integrates algorithmic elegance with empirical validation. This research traverses the intricacies of unique methodologies and explores the nuances of the conversion process, recognizing inherent limitations and sketching the boundaries of what's achievable. Furthermore, it brings forth normalization trends and median attributes within the landscape of general malware and Python-based malicious software. The study crafts a comprehensive understanding of malware intricacies by weaving these empirical threads.

Furthermore, this paper affords an in-depth exposé of the practical architecture and operational dynamics of a FUD Crypter. The exploration delves into the orchestration of symmetric and asymmetric encryption algorithms [3], unveiling the mechanisms that facilitate the crypter's veil of invisibility. The core of crypter lies in the dynamic generation and encapsulation of cryptographic keys, similar to how lock and key works together. This intricate interplay orchestrates a web of encryption and encapsulation that seeks to obscure the payload's true essence. By seamlessly intertwining encryption with encapsulation, the crypter crafts a labyrinthine puzzle that traditional antivirus systems grapple to decipher.

In summary, this research sheds light on the cutting-edge world of FUD Crypters and the intricate techniques behind them. It illuminates the evolving landscape of cybersecurity, where innovation is the key to staying ahead in the perpetual battle between attackers and defenders.

II. METHODOLOGY

Cryptography [3] is a vital aspect of modern communication systems, ensuring secure and private transfer of information. Two essential techniques are used to achieve this: symmetric and asymmetric key cryptography. Both techniques offer unique advantages to the cryptographic projects they are used in, making them essential tools in cryptography [3][32]

Symmetric key cryptography, also known as secret-key cryptography, encrypts and decrypts with a single key [32]. This efficient and fast approach makes it ideal for high-speed encryption and decryption applications, such as data storage and wireless communication. One widely used symmetric key cryptography algorithm is Advanced Encryption Standard (AES), known for its speed, efficiency, and security.

Asymmetric key cryptography, often known as public-key cryptography, uses two keys: one public and one private. The public key is available to anyone, while the private key is kept secret. This approach is more secure than symmetric key cryptography, as it eliminates the need to share the secret key, which could be intercepted. Asymmetric key cryptography is ideal for secure communication and authentication applications, such as digital signatures and key exchange. One of the most often used asymmetric key cryptography schemes algorithms is Rivest-Shamir-Adleman (RSA), known for its high security and versatility [32].

While symmetric key cryptography algorithms like AES [3] are faster and more efficient, they require a secure channel to transmit the key. Asymmetric key cryptography algorithms like RSA are slower but eliminate the need for a secure channel. Both approaches have distinct benefits, and the decision of which algorithm to utilize is determined by the project's specific security requirements [3].

Advanced Encryption Standard (AES) [3] is a widely used encryption algorithm that provides confidentiality and authenticity over an insecure communication channel. The minimum length of an AES cipher key is 128 bits, providing 2^{128} possible keys, making brute-force attacks infeasible. AES operates on a 128-bit block of plaintext at a time and can use key sizes of 128, 192, or 256 bits [30]

In AES [3], each byte is considered an element of a finite field of characteristic 2 with 8 terms, also known as Galois field, $GF(2^8)$. $GF(2^8)$ is a field of 256 elements, and each element can be represented by a polynomial of degree 7 with coefficients of either 0 or 1 [30]

Rivest-Shamir-Adleman (RSA) [19] is a widely used public-key encryption algorithm that provides confidentiality and authenticity over an insecure communication channel. RSA is based on the mathematical principles of big prime numbers, and it encrypts and decrypts using a public key and a private key. RSA is a popular asymmetric encryption method, which means it uses separate keys for encryption and decoding [3]

2.3 Why are we integrating AES and RSA?

The combination of AES (Advanced Encryption Standard) and RSA (Rivest-Shamir-Adleman) cryptographic algorithms within Fully Un-Detectable (FUD) crypter's presents a formidable technique to heighten the robustness of payloads against detection and analysis by antivirus software. In this context, employing AES encryption is a foundational step to obscure the payload's primary content, leaving it indecipherable to organizations without the associated decryption key. This crucial phase effectively evades detection by antivirus systems that rely on known viral signatures because the payload is concealed through encryption [21].

1. Encryption for Payload Concealment:

Integrating AES encryption into FUD crypter's adds a solid layer of payload hiding. This approach relies on scrambling the payload using AES encryption, rendering it indecipherable to unauthorized entities lacking the appropriate decryption key. This has a double effect: first, known viral signatures within the payload remain invisible to antivirus software, bypassing their typical signature-based detection algorithms. Second, AES encryption aids to maintaining the integrity of the payload from malicious modification, increasing its resilience against discovery.

2. Payload Obfuscation:

FUD crypter's widely deploy payload obfuscation methods to conceal the payload code, rendering it challenging for antivirus software to detect and analyze. The joint employment of AES and RSA [19] further augments this obfuscation. Alterations made to the payload's code, driven by obfuscation, break its signature, effectively escaping pattern-based identification by antivirus engines. Consequently, the crypter's payload eludes the typical heuristic analysis approaches utilized by antivirus software, offering a substantial obstacle to detection.

3. Signature Avoidance using RSA:

Including RSA-based digital signatures is a fundamental part of the FUD crypter's complexity. This multilayered approach endows the recipient with the capacity to validate the payload's validity. The RSA digital signature is a unique cryptographic fingerprint closely related to the payload's content and the signer's private key [25] As a result, any tampering or illegal modifications to the payload may be promptly identified through the invalidation of the RSA signature. This works as an

effective deterrent against harmful alterations and offers an extra degree of complexity to the antivirus software's inspection procedure.

4. *Strengthening Evasion Capabilities:*

By interweaving AES encryption, payload obfuscation, and RSA digital signatures, FUD crypter's significantly boost their evasion skills against antivirus software. The payload's contents remain unavailable through AES encryption, hindering the discovery of recognized harmful patterns. Concurrently, payload obfuscation techniques disturb heuristic analysis, complicating signature-based detection. RSA signatures, in turn, confirm the payload's validity while functioning as a deterrent to manipulation. Despite the problematic nature of these combined approaches, it is vital to recognize the dynamic world of cybersecurity, where fresh strategies are continually emerging on both fronts — evasion and detection.

2.4 Workflow Description:

This section includes a Python script leveraging a clever fusion of Advanced Encryption Standard (AES) and Rivest-Shamir-Adleman (RSA) algorithms to permit safe and undetectable payload execution. The script's workflow may be briefly summarized as follows:

1. *RSA Key Pair Generation:*

The script starts by producing an RSA key pair including a private key and its associated public key. The private key stays confidential, while the public key encrypts payloads to establish a secure communication channel.

2. *AES Key Generation:*

An AES key, necessary for payload encryption, is created via a cryptographically safe random number generator. This 256-bit AES key lays the foundation for effective encryption.

3. *Payload Encryption using AES:*

Payload material, taken from a given file, undergoes AES encryption. This procedure involves constructing a random Initialization Vector (IV) to boost security. The payload is padded using PKCS7 padding, and the AES key is applied in CFB mode to create encrypted data. The encrypted payload is kept as a concatenation of the IV and encrypted content.

4. *AES Key Encryption with RSA:*

The AES key's security is encrypted with the recipient's (public) RSA key. The RSA OAEP padding algorithm guarantees safe encryption and decoding improving overall security. Since the private key is generated dynamically and not saved to a file or any external storage, it will be lost once the script execution is complete. This approach can be appropriate for scenarios where you want to keep the private key in memory only for the duration of a specific operation and do not need to persistently store it.

5. *Stub File Generation:*

A dynamic stub file is constructed to contain the encrypted AES key, RSA public key, and encrypted payload. This stub file becomes the platform for safe payload execution. The RSA private key is securely encoded within the stub file, ensuring decryption is restricted to authorized organizations.

The private key is generated using the `generate_rsa_key_pair` function but is not stored persistently in this code. The public key is extracted from the generated RSA key pair and stored as a PEM-encoded string in the `'stub.py'` file (`public_key_pem`).

6. *Existing Stub File Check:*

The script validates the presence of preceding stub files. If identified, choices to delete or rename the file are presented to preserve a clean execution environment & maintaining system hygiene.

7. *Execution of Encrypted Payload:*

During execution, the stub file performs AES key decryption using the RSA private key, followed by payload decryption and execution.

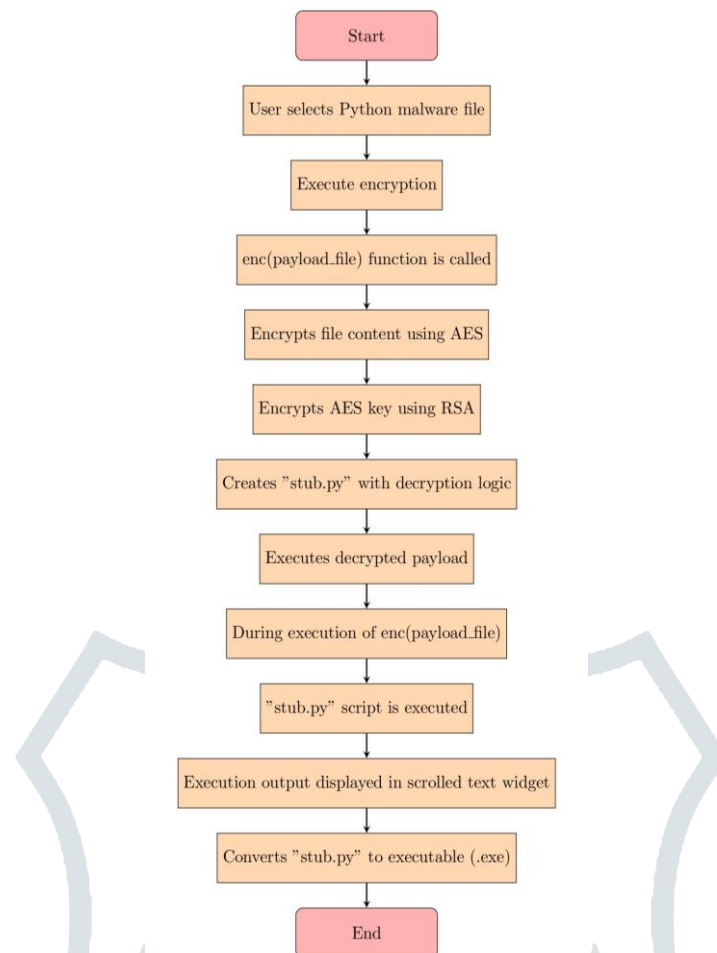


Fig.1 Workflow of the conversion steps

Furthermore, an essential aspect worth highlighting is the consolidation of these steps into a single, self-contained file. This approach streamlines the deployment and execution of the FUD Crypter, making it more convenient for users. By packaging all the necessary components and functionalities within a single file, the crypter becomes a self-contained solution for secure and undetectable payload execution.

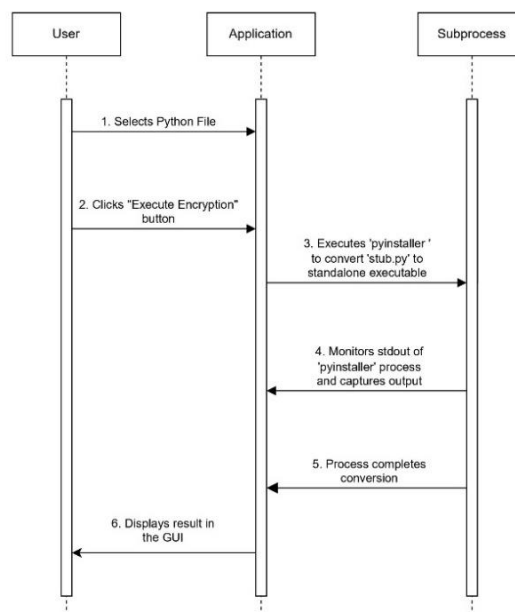


Fig.2 Sequence Diagram explaining the conversion steps

III. IMPLEMENTATION

This section provides a detailed description of the practical implementation of the proposed encryption method. The encryption process combines symmetric (AES) and asymmetric [3] (RSA) encryption techniques to ensure data confidentiality and integrity. Each step is explained thoroughly.

3.1 Key Generation

This section discusses the generation of encryption keys, a crucial aspect of our encryption process.

3.1.1. RSA Key Pair Generation

RSA encryption necessitates a key pair comprising a private key for decryption and a public key for encryption. We use the `generate_rsa_key_pair()` function for this purpose:

```
def generate_rsa_key_pair():
    private_key =
    rsa.generate_private_key(
        public_exponent=65537,
        key_size=2048
    )
    public_key =
    private_key.public_key()
    return private_key, public_key
```

The `generate_rsa_key_pair()` function utilizes the `rsa.generate_private_key()` method from the cryptography library to create a private key size of 2048 bits. It also derives a corresponding public key.

3.1.2. AES Key Generation

AES encryption relies on a symmetric key for both encryption and decryption. A random 256-bit AES key is generated using the `generate_aes_key()` function:

```
def generate_aes_key():
    return os.urandom(32)
```

The `generate_aes_key()` function generates a random 256-bit AES key using the `os.urandom()` function. This key will be used for symmetric encryption of the payload data.

3.2. Payload Encryption:

This section delves into the encryption of the payload data using the AES encryption algorithm.

3.2.1. AES Encryption

To secure the payload data, the AES encryption algorithm is employed. The `encrypt_aes(aes_key, data)` function handles this operation:

```
def encrypt_aes(aes_key, data):
    iv = os.urandom(16) #
    Initialization vector for AES
    cipher =
    Cipher(algorithms.AES(aes_key),
    modes.CFB(iv))
    encryptor = cipher.encryptor()
```

The `encrypt_aes()` function accepts the AES key and the payload data as input. It generates a random initialization vector (IV) using `os.urandom()` to enhance security.

The AES encryption algorithm, available in the cryptography library, encrypts the data in CFB (Cipher Feedback) mode. This mode ensures that even slight changes in the encrypted data will significantly alter the decrypted result.

```
padder =
sym_padding.PKCS7(128).padder()
padded_data =
padder.update(data.encode('utf-8')) +
padder.finalize()

    encrypted_data =
    encryptor.update(padded_data) +
    encryptor.finalize()
    return iv + encrypted_data
```

Proper padding (PKCS7) is applied to the data to ensure integrity during encryption and decryption.

3.3. Key Exchange

To establish a secure channel for data decryption, the AES Encryption key is encrypted using the recipient's public RSA key.

3.3.1. RSA Encryption of AES Key

Encrypting the AES key using the recipient's public RSA key is pivotal in ensuring secure communication. The following code accomplishes this:

```
def enc(payload_file):
    private_key, public_key =
generate_rsa_key_pair()
    aes_key = generate_aes_key()

    with open(payload_file) as f:
        contents = f.read()

    encrypted_contents_aes =
encrypt_aes(aes_key, contents)
    encrypted_contents_rsa =
encrypt_rsa(public_key, aes_key)
```

The `encrypt_rsa()` function takes the public RSA and AES keys as input parameters.

Inside the function, the `public_key.encrypt()` method from the cryptography library is used to encrypt the AES key. This encryption step ensures that only authorized parties possessing the corresponding private RSA key can decrypt the payload data.

3.4. Payload File Generation

This subsection covers the creation of a new Python script file named "stub.py" and the inclusion of both the encrypted AES key and the payload within this file. Additionally, `stub.py` includes code for decrypting the AES key and payload using the private RSA key

The code has the process of writing the encrypted AES key and payload data in hexadecimal format to the `stub.py` file.

`stub.py` is a self-contained script that includes all the necessary code for decryption. Upon execution, it decrypts the AES key and the payload data for further processing.

3.5. Usage

This section provides detailed information on how to utilize the implemented encryption method.

3.5.1. Running the Script

To encrypt a payload file, execute the script and provide the path to the payload file when prompted:

```
python crypter.py
```

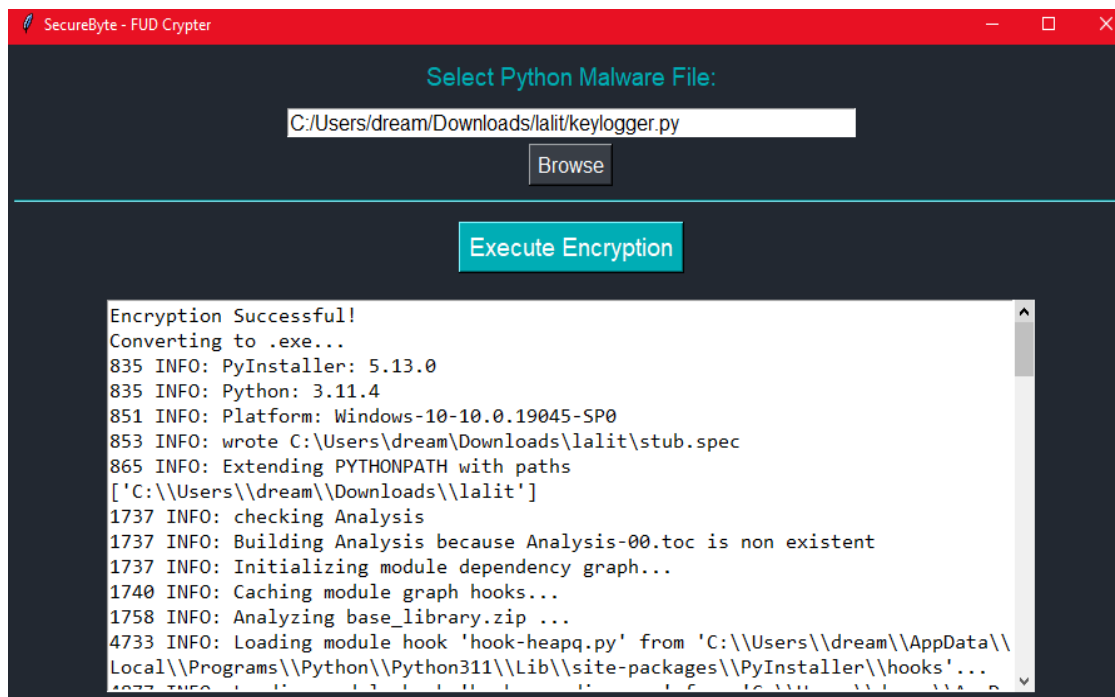


Fig.3. GUI of our FUD Crypter

The user is expected to run the *crypter.py* script. This script initiates the encryption process and prompts the user to specify the path to the payload file. The script then performs the encryption steps as described in the previous sections.

3.5.2. Secure Key Distribution

The recipient must have the corresponding public RSA key to ensure secure communication. This key is essential for decrypting the AES key and accessing the payload securely. Secure key distribution is crucial for maintaining the confidentiality and integrity of data.

3.6. Security Considerations

The security of this encryption method relies on proper key management, secure distribution of the public RSA key, and adherence to best practices for data handling. Protecting the private RSA key and ensuring its secure storage are paramount to the system's overall security.

IV. RESULTS

In this section, we present the detailed results of our research, which primarily revolves around assessing the effectiveness of our crypter on Python malware samples. Our analysis encompasses several critical aspects, including establishing a normalization curve for Python malwares, our testing procedure, execution time recordings, the controlled testing environment using Windows 10 Sandbox, and a comprehensive comparison of detection rates before and after applying our crypter on various platform such as VirusTotal[35], HybridAnalysis[34], KleenScan[36], MetaDefender [37]

4.1 Normalization Curve for Python Malwares:

We carefully constructed a normalization curve for Python malware to initiate our investigation which serves as a foundational reference point, elucidating the distribution of malware sizes under normal conditions when our crypter is not actively employed. X-Axis represents individual Python malware samples, with each point corresponding to a specific malware sample and Y-Axis represents the sizes of the Python malware files, measured in a chosen unit (e.g., bytes or kilobytes). We obtained valuable insights into the baseline variance in Python malware sizes by calculating the median of the malware sizes in this state. This information is vital for evaluating the impact of our crypter on the subsequent sizes of encrypted malware samples.

We used the data from May 28, 2022 to July 20, 2023 [33]. We wanted recent and relevant samples for our research. We tested these malware samples using our crypter. We wanted to see how long each malware took to run during testing. By doing it this way, we could examine the malware in a comprehensive and practical way.

Our analysis of 65 Python based malware revealed a diverse distribution of file sizes. The normalization curve plot visually depicted the distribution, showing that a significant portion of the malware falls within a certain range. The median size of approximately **9.586 KB** provides a valuable reference point for understanding the typical size of Python format malware.

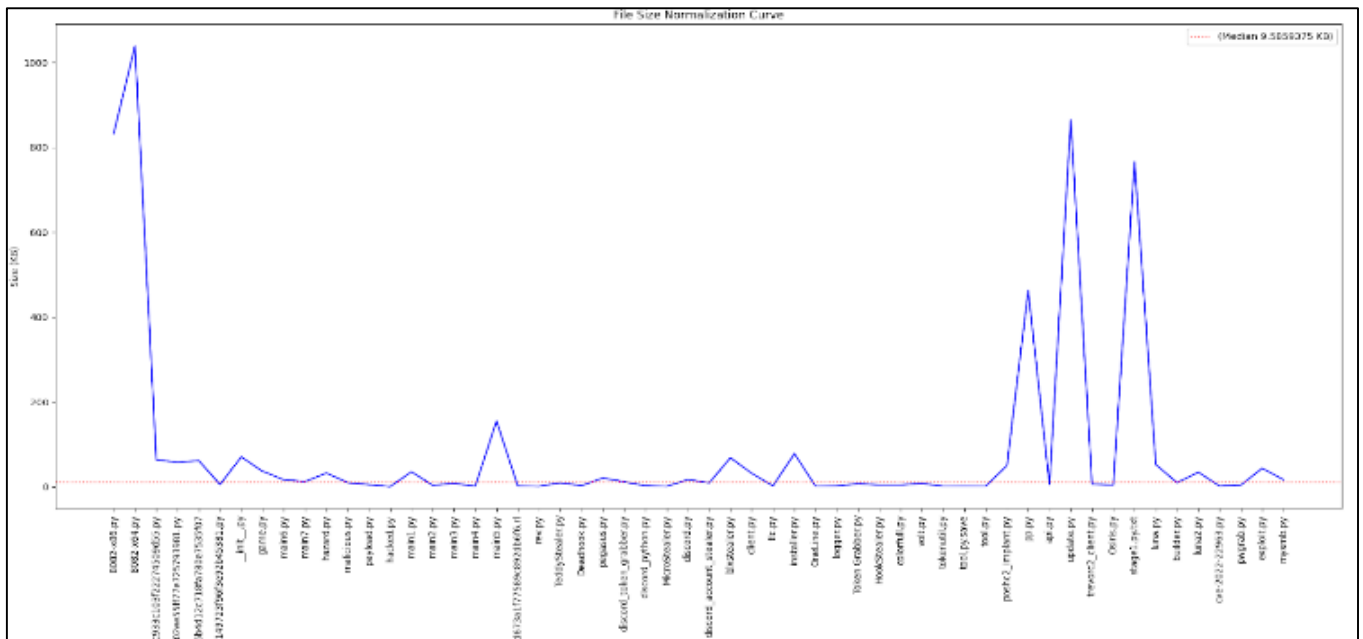


Fig. 4. Normalization Graph of Python Malware based on their sizes.

4.2 Testing Procedure:

To rigorously evaluate the performance of our crypter, we subjected it to a battery of tests against a carefully curated set of Python malware samples. This sample set was meticulously selected to include the most recent entries from Malware Bazaar [33], thereby representing authentic, real-world threats to cybersecurity. It is important to emphasize that selecting contemporary malware samples enhances the relevance and applicability of our research findings in the context of the ever-evolving threat landscape. Ensuring the authenticity of the test samples was of paramount importance. We meticulously validated the integrity and legitimacy of each Python malware sample within our test set.

We gathered necessary data-points about various aspects of the crypter's performance throughout the testing phase such as runtime behavior, alterations in binary characteristics, and the crypter's effectiveness in evading detection by prominent antivirus and intrusion detection systems. Subsequently, we subjected this data to comprehensive statistical analysis, enabling us to quantify and interpret the results rigorously. Our rigorous evaluation criteria facilitated a holistic assessment of the crypter's strengths and weaknesses.

4.3 Execution Time Recordings:

During the testing phase, our research methodology emphasized the meticulous recording and analysis of the execution times of our crypter for each Python malware sample. This aspect of our evaluation process was conducted with utmost precision and attention to detail.

4.3.1 Detailed Time Measurements:

To accurately gauge the efficiency of our crypter, we recorded the execution times of the crypter for processing each malware sample within our test set. These measurements were not limited to overall execution times but delved into granular details. We captured the time the crypter took from the initiation to the successful completion of the encryption process. This level of granularity allowed us to understand the crypter's performance on a per-sample basis, offering insights into its consistency and reliability.

4.3.2 Range of Execution Times:

Our recorded execution times they have exhibited a notable range, from a minimum of 0.11 seconds to a maximum of 0.37 seconds. This wide range of time measurements enabled us to comprehensively assess the crypter's responsiveness and efficiency across various Python malware samples. It is important to note that these times reflect the duration required for the crypter to obfuscate the code while maintaining the malware's core functionality.

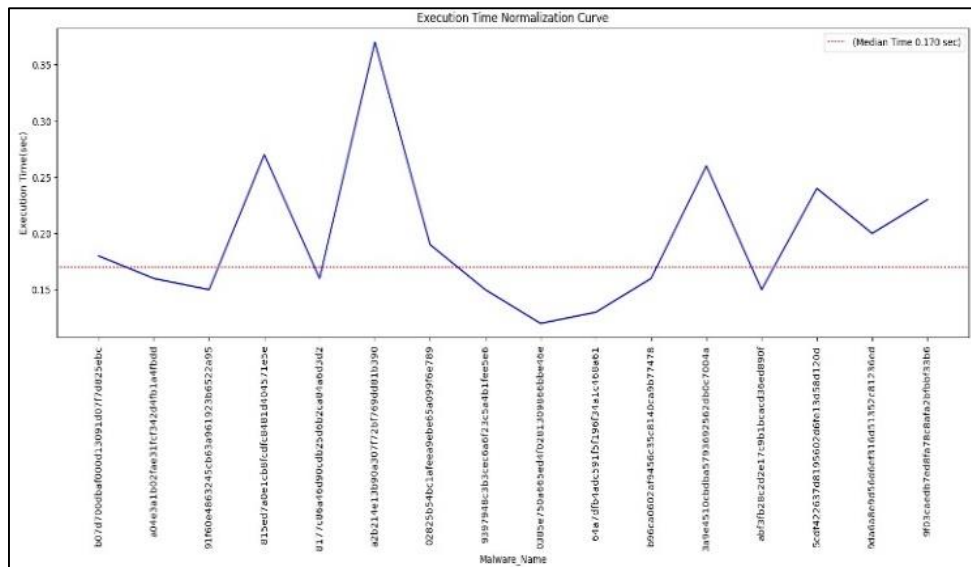


Fig. 5. Normalization Curve of Python Malware based on Execution Time

4.3.3 Efficiency and System Impact:

The precision in execution time tracking underscored the efficiency of our crypter in processing Python malware. Importantly, the recorded execution times also highlighted the minimal impact of the crypter on overall system performance. By maintaining swift encryption processes within such short timeframes, our crypter demonstrated its practicality and suitability for real-world cybersecurity scenarios, where minimizing performance degradation is critical.

4.4 Windows 10 Sandbox Utilization:

A crucial aspect of our research design is the controlled environment in which we conducted our tests. All evaluations occurred within a Windows 10 Sandbox provided by CrowdStrike Falcon Sandbox [38] [34], ensuring a consistent and controlled setting for the normal Python malware samples. Windows 10 Sandbox is a lightweight, temporary, and disposable virtualized environment specifically designed for running untrusted or potentially harmful applications in isolation [34].

By employing this environment, we aimed to eliminate the influence of extraneous variables that might affect the results. While the specific configurations and optimizations of the Falcon Windows 10 Sandbox [38] environment is not publicly disclosed in detail, it is reasonable to assume that they have implemented security and isolation measures to ensure accurate and controlled malware analysis. These measures may include network isolation, security software configurations, and possibly custom settings designed to enhance the analysis process and protect the host system. It allowed us to conduct experiments with high confidence that the observed outcomes were solely attributed to the interaction between the Python malware samples and our crypter.

The controlled environment also allowed us to clearly distinguish between the behavior of normal Python malware samples and their crypter-encrypted counterparts, thus facilitating a robust analysis of the crypter's impact on malware detection and system behavior.

4.5 Detection Rates Comparison:

Our research culminated in a comprehensive comparison of detection rates before and after the application of our crypter. This aspect of our study yielded significant findings that shed light on the crypter's efficacy.

Before: Initially, all 10 malware samples were detectable by most security systems, including but not limited to VirusTotal [35], HybridAnalysis [34], KleanScan [36], MetaDefender [37].

MDS Hash (Malwares)	Virus Total	Hybrid Analysis - AV	Detection Hybrid Analysis - Threat Score	MetaDefender	KleanScan
a2b214e13b90a307f72bf769dd81b390	19/60	20%	100/100	4/26	8/40
91f60e4863245cb63a961923b6522a95	19/60	24%	100/100	7/26	9/40
815ed7a0e1cb8fcd8c481d404571e5e	20/60	15%	100/100	4/26	12/40
271e93707156e7d56318251f6f906c1f	19/60	14%	100/100	3/26	12/40
02825b54bc1afeea9e8e5a09f6e789	18/60	14%	100/100	3/26	12/40
64a7dfb4adc591f5f196f34a1c468a61	24/60	3%	44/100	1/26	2/40
5cdf422637d8195602d5fe13d58d120d	21/60	2%	100/100	1/26	10/40
3a9e4510cbdba5793692562db0c7004a	25/59	9%	100/100	2/26	12/40
9f03caedb7ed8fa78c8afa2fbfbf33b6	29/59	41%	100/100	NA	10/40
abf3fb28c2d2e17c9b1bcacd36ed890f	22/60	14%	100/100	3/26	7/40

Fig. 6. Malwares Detection by different platform before using FUD Crypter

After: Following the application of our crypter, a remarkable reduction in detection rates was observed. In particular, VirusTotal's [35] detection rate plummeted to 0%, indicating that none of the samples managed to trigger detection alerts. On the other hand, Hybrid Analysis [34] was only able to detect 2 out of the 10 samples due to the dynamic nature of the hybrid analysis [34], showcasing a notable inconsistency between the pre- and post-encryption data sets.

MDS Hash (Malwares)	Virus Total	Hybrid Analysis - AV Detection	Hybrid Analysis - Threat Score	MetaDefender	KleenScan
a2b214e13b90a307f72bf769dd81b390	0/60	Clean	0/100	Clean	Clean
91f60e4863245cb63a961923b6522a95	0/60	Clean	0/100	Clean	Clean
815ed7a0e1cb8fcd8481d404571e5e	0/60	Clean	0/100	Clean	Clean
271e93707156e7d56318251f6f906c1f	0/60	Clean	0/100	Clean	Clean
02825b54bc1afeea9ebe65a099f6e789	0/60	Clean	0/100	Clean	Clean
64a7dfb4dc591f5f196f34a1c468a61	0/60	Clean	0/100	Clean	Clean
5cdf422637d8195602d6fe13d58d120d	0/60	Clean	0/100	Clean	Clean
3a9e4510cbdba5793692562db0c7004a	0/59	Clean	0/100	Clean	Clean
9f03caedb7ed8fa78c8afa2bfbbf33b6	0/59	Clean	0/100	Clean	Clean
abf3fb28c2d2e17c9b1bcacd36ed890f	0/59	Clean	0/100	Clean	Clean

Fig. 7. Malwares Detection by different platform After using FUD Crypter

VirusTotal[35], HybridAnalysis[34], KleenScan[36], MetaDefender [37] aggregates results from a wide range of antivirus engines, each with its own set of signatures and heuristics. These engines use signature-based detection and behavioral analysis to identify known malware and suspicious behavior. When you submit a file to these platforms, it scans the file using multiple antivirus engines and provides a report that includes detection results from each engine. If a file is detected as malicious by any of the antivirus engines used by these platforms, it will be reported as a positive detection. Conversely, if no antivirus engine detects the file as malicious, the detection rate is reported as 0%.

On the other hand, Hybrid Analysis employs a dynamic analysis approach, focusing on the runtime behavior of files and executables. It monitors the file's behavior within a controlled environment and analyzes its actions. Dynamic analysis platforms like Hybrid Analysis can detect malware based on behavioral characteristics, such as suspicious system interactions, file modifications, and network activity. The effectiveness of dynamic analysis platforms like Hybrid Analysis may vary based on the malware's ability to evade behavioral detection mechanisms [34].

The malware samples we tested had behaviors that were highly indicative of malicious intent but did not exhibit easily recognizable signatures, VirusTotal[35], HybridAnalysis [34], KleenScan [36] and MetaDefender [37] has static signature-based engines that might not have been able to detect them after encryption and obfuscation, resulting in a 0% detection rate on VirusTotal. On the other hand, Hybrid Analysis's dynamic analysis capabilities may have allowed it to detect certain behavioral anomalies or patterns associated with the encrypted and obfuscated malware, resulting in a detection rate of 2 out of 10 samples.

In summation, our research yields compelling evidence regarding the profound impact of our crypter on Python malware detection rates. Before the crypter's application, all 10 malware samples were easily discernible by various security systems. However, after undergoing obfuscation [22] and encryption, these samples managed to evade detection by most security systems, with detection rate plummeting to 0%.

Our research is a stark reminder of the potency of our crypter's encryption and obfuscation techniques in circumventing traditional security measures. It also raises pertinent questions about the adequacy of existing security systems in identifying and mitigating sophisticated malware that employs obfuscation methods. The implications of our findings underscore the imperative need for advanced and adaptive cybersecurity solutions to effectively combat the evolving threat landscape posed by obfuscated Python malware. These solutions should be equipped to tackle the challenges of such evasive threats and safeguard the integrity of computer systems and data in an increasingly complex digital ecosystem.

V. LIMITATIONS

In the development of a Fully Undetectable (FUD) Crypter, we used the power of Python to craft a versatile and covert payload. Throughout this process, we encountered several limitations that warrant thorough consideration. These limitations primarily revolve around the accurate conversion of various file formats, including '.exe', '.bin', '.msi', and '.bat' files into Python payloads. The implications of these limitations extend to the overall functionality of the Crypter, ultimately affecting its efficacy in obfuscating payloads and enhancing security.

5.1 Conversion Accuracy Challenges:

One fundamental restriction is the accuracy with which multiple file formats may be converted into Python payloads. When handled purely by Python, the conversion procedure may cause inaccuracies and inconsistencies. These errors emerge as unnecessary data, often known as "junk values," within the converted payloads. Such disparities can be linked to Python's possible difficulty in deciphering the low-level system processes and complexities unique to each original file format.

These inconsistencies are a major source of concern since they threaten the Crypter's integrity. Payloads containing garbage values are prone to unexpected behavior or failure during execution. As a result, these mistakes not only jeopardize the Crypter's capacity to remain undetected, but also increase the chance of detection.

5.2 Impact on Functionality:

The limitations associated with conversion accuracy have a direct and substantial impact on the functionality of the Python payload. When conversions are not executed with precision, the resulting payloads may become mangled or non-functional. This compromise severely impairs the Crypter's primary objective of achieving full undetectability.

Mangled or non-functional payloads jeopardize the Crypter's role in enhancing security and evading detection. The consequences of these inaccuracies are twofold: First, there is an increased likelihood of payload execution failures, rendering the Crypter ineffective in its core purpose. Second, payloads may display erratic and unpredictable behavior, further raising the risk of detection by security mechanisms.

VI. CONCLUSION

The development of Fully Undetectable (FUD) Crypters is a beacon of innovation and resilience. Our research has explored the intricate realm of FUD Crypter development, emphasizing the integration of advanced techniques encompassing encryption, code obfuscation [22]. These elements form the bedrock of modern malware evasion strategies, enabling covert execution and eluding vigilant antivirus software.

At its core, our study has showcased the extraordinary impact of combining AES and RSA [19] cryptographic algorithms within FUD Crypters. This union of cryptographic prowess has fundamentally reshaped the landscape of malware evasion. With its payload concealment capabilities, the application of AES encryption serves as the first line of defense, rendering malware virtually invisible to signature-based antivirus detection. Simultaneously, code obfuscation [22] techniques introduce a layer of complexity that challenges heuristic analysis, further bolstering evasion. Integrating RSA digital signatures adds a unique layer of validation and deterrent against unauthorized alterations to the payload.

Our research journey ventured beyond the confines of cryptographic techniques, delving into the nuances of Python-based malware and the establishment of a normalization curve. This curve illuminated the typical distribution of malware sizes, offering valuable reference points for understanding the impact of our Crypter on encrypted payloads. Furthermore, meticulous execution time recordings underscored the efficiency of our Crypter while affirming its minimal impact on system performance.

Operating within the controlled environment of Windows 10 Sandbox [34], our research rigorously compared detection rates before and after the application of our Crypter. The results were unequivocal: a remarkable reduction in detection rates, with VirusTotal's detection rate plummeting to 0%, exemplified the Crypter's efficacy in circumventing traditional security measures.

Ultimately, our research signifies a pivotal milestone in the ongoing battle between cybersecurity defenders and threat actors. Integrating AES and RSA within our Crypter has elevated malware evasion to unprecedented heights and underscored the need for adaptive cybersecurity solutions capable of addressing the evolving threat landscape. The implications of our findings extend beyond the boundaries of this research, highlighting the critical importance of innovation, adaptability, and collaboration in safeguarding the digital ecosystem.

VII. ACKNOWLEDGMENTS

We want to express our sincere appreciation to all the individuals and organizations who have contributed to completing this research paper.

First and foremost, we would like to extend our heartfelt gratitude to our research advisor, **Nidhi Patel**, and our mentor, **Milind Purswani**. Their guidance, support, and invaluable insights throughout the research have been instrumental in shaping this paper's direction, scope, and quality. Their expertise and mentorship have greatly contributed to our understanding and development in FUD Crypter Development by Combining Symmetric & Asymmetric Cryptographic Algorithms

We would also like to acknowledge the researchers and authors whose scholarly works and publications formed the foundation of this research. Their significant contributions to the FUD Crypter Development by Combining Symmetric & Asymmetric Cryptographic Algorithms have been paramount, and their insights have greatly enriched the content of this paper.

Furthermore, we thank the reviewers and editors who provided valuable feedback and suggestions during the review process. Their constructive criticism and guidance have played a pivotal role in improving the quality and clarity of this paper.

Additionally, we would like to express our appreciation to our colleagues and peers for their engaging discussions, insightful perspectives, and unwavering support throughout the development of this research paper. Their contributions and collaboration have been invaluable in shaping our understanding and refining the content.

Thank you all for your dedication and commitment to advancing the FUD Crypter Development by Combining Symmetric & Asymmetric Cryptographic Algorithms.

VIII. REFERENCES

- [1] C Intila et al 2019 IOP Conf. Ser.: Mater. Sci. Eng. 482 012016
- [2] A., Harsha & Patil, Basavaraj. (2016). A Review: Security of Data in Cloud Storage using ECC Algorithm. Bonfring International Journal of Software Engineering and Soft Computing. 10.9756/BIJSESC.8262.
- [3] Faiqa Maqsood, Muhammad Ahmed, Muhammad Mumtaz Ali and Munam Ali Shah, "Cryptography: A Comparative Analysis for Modern Techniques" International Journal of Advanced Computer Science and Applications(IJACSA),8(6),2017.<http://dx.doi.org/10.14569/IJACSA.2017.080659>

- [4] S. A. Jaju and S. S. Chowhan, "A Modified RSA algorithm to enhance security for digital signature," 2015 International Conference and Workshop on Computing and Communication (IEMCON), Vancouver, BC, Canada, 2015, doi: 10.1109/IEMCON.2015.7344493.
- [5] A. Aminuddin, "Android Assets Protection Using RSA and AES Cryptography to Prevent App Piracy," 2020 3rd International Conference on Information and Communications Technology (ICOIACT), Yogyakarta, Indonesia, 2020, doi: 10.1109/ICOIACT50329.2020.9331988.
- [6] H. Dibas and K. E. Sabri, "A comprehensive performance empirical study of the symmetric algorithms: AES, 3DES, Blowfish and Twofish," 2021 International Conference on Information Technology (ICIT), Amman, Jordan, 2021, doi: 10.1109/ICIT52682.2021.9491644.
- [7] E. Vidhya, S. Sivabalan and R. Rathipriya, "Hybrid Key Generation for RSA and ECC," 2019 International Conference on Communication and Electronics Systems (ICCES), Coimbatore, India, 2019, doi: 10.1109/ICCES45898.2019.9002197.
- [8] Aseel Hamoud Hamza and Saif M. Kh. Al-Alak 2021 J. Phys.: Conf. Ser. 1804 012096 DOI 10.1088/1742-6596/1804/1/012096
- [9] Y. Ali and A. Hameed, "Cloud Crypter for bypassing Antivirus," 2019 15th International Conference on Emerging Technologies (ICET), Peshawar, Pakistan, 2019, doi: 10.1109/ICET48972.2019.8994615.
- [10] C. Barria, D. Cordero, C. Cubillos and R. Osses, "Obfuscation procedure based in dead code insertion into crypter," 2016 6th International Conference on Computers Communications and Control (ICCCC), Oradea, Romania, 2016, doi: 10.1109/ICCCC.2016.7496733.
- [11] Institute of Informatics, Silesian University of Technology, 44-100 Gliwice, Poland Appl. Sci. 2023, 13(8), 5083; <https://doi.org/10.3390/app13085083>
- [12] O. Sukwong, H. Kim and J. Hoe, "Commercial Antivirus Software Effectiveness: An Empirical Study" in Computer, vol. 44, no. 03, 011. <https://doi.ieeecomputersociety.org/10.1109/MC.2010.187>
- [13] F. -H. Hsu, M. -H. Wu, C. -K. Tso, C. -H. Hsu and C. -W. Chen, "Antivirus Software Shield Against Antivirus Terminators," in IEEE Transactions on Information Forensics and Security, vol. 7, no. 5, Oct. 2012, doi: 10.1109/TIFS.2012.2206028.
- [14] A. Fashakh and A. Abdu Ibrahim, "A Proposed Secure and Efficient Big Data (Hadoop) Security Mechanism using Encryption Algorithm," 2023 5th International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA), Istanbul, Turkiye, 2023, doi: 10.1109/HORA58378.2023.10155777.
- [15] M. Vandenwauver, J. Claessens, W. Moreau, C. Vaduva and R. Maier, "Why enterprises need more than firewalls and intrusion detection systems," Proceedings. IEEE 8th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE'99), Stanford, CA, USA, 1999, doi: 10.1109/ENABL.1999.805191.
- [16] Huseyin Cavusoglu, Birendra Mishra, Srinivasan Raghunathan, (2005) The Value of Intrusion Detection Systems in Information Technology Security Architecture. Information Systems Research 16(1):28-46. <https://doi.org/10.1287/isre.1050.0041>
- [17] A Study of Encryption Algorithms AES, DES and RSA for Security By Dr. Prerna Mahajan & Abhishek Sachdeva - IITM, India.
- [18] A Comparative Analysis of Cryptographic Algorithms: AES & RSA and Hybrid Algorithm for Encryption and Decryption By Sa'idu Sani, Prashansa Taneja & Shreya Kalta.
- [19] An Improved Cryptographic Scheme Using AES & RSA Algorithms for Maximum Security In File Encryption and Decryption By Sa'idu Sani
- [20] A Comparative Analysis of AES and RSA Algorithms By Shaili Singhal & Dr. Niraj Singhal
- [21] A Comparative Study of the Performance and Security Issues of AES and RSA Cryptography By Abdullah Al Hasib - Norwegian University of Science and Technology
- [22] Assessment of Source Code Obfuscation Techniques By Alessio Viticchie , Leonardo Regano, Marco Torchiano , Cataldo Basile , Mariano Ceccato , Paolo Tonella and Roberto Tiella
- [23] Code Obfuscation against Static and Dynamic Reverse Engineering By Sebastian Schrittwieser - Vienna University of Technology, Austria and Stefan Katzenbeisser – Darmstadt University of Technology, Germany
- [24] Bypass Antivirus Dynamic Analysis Limitations of the AV model and how to exploit them By Emeric Nasi
- [25] Bypassing Antivirus Detection with Encryption By Tasiopoulos Vasileios
- [26] Bypassing Antivirus and Antivirus Vulnerabilities By Aryan Jadon - San Jose State University
- [27] Review of Viruses and Antivirus Patterns By Muchelule Yusuf, Wanjala & Neyole, Misiko Jacob, Yusuf Muchelule.
- [28] Study of an Anti-Virus Framework By Ming Zhang - School of Information Engineering, Wuhan University of Technology, Wuhan, China & Wei Chen – Linyi University, Shandong, China
- [29] Advancement in Antivirus By Priya Upadhyay, Ayushi Saxena, Ankuj Singh, Pradeep Sharma - Student at Vivekananda College of Technology & Management, Aligarh, India
- [30] Abdullah, Ako. (2017). Advanced Encryption Standard (AES) Algorithm to Encrypt and Decrypt Data. Insights Gained from Constructing a Large Scale Dynamic Analysis Platform by Cody Miller et al. Published in Digital Investigation, 2017. <https://doi.org/10.1016/j.diin.2017.06.007>.
- [31] Singh, Gurpreet & Kinger, Supriya. (2013). A Study of Encryption Algorithms (RSA, DES, 3DES and AES) for Information Security. International Journal of Computer Applications. 67. 33-38. 10.5120/11507-7224.
- [32] Bazaar, M. Malware Bazaar Dataset. Available online: <https://bazaar.abuse.ch/>
- [33] Free Automated Malware Analysis Service - powered by Falcon Sandbox. (n.d.). <https://www.hybrid-analysis.com/>
- [34] VirusTotal. (n.d.). VirusTotal. <https://www.virustotal.com/gui/>
- [35] Kleenscan.com. (n.d.). Kleenscan. <https://www.kleenscan.com/index>
- [36] MetaDefender Cloud | Advanced threat prevention and detection. (n.d.). MetaDefender Cloud. <https://metadefender.opswat.com/>
- [37] The CrowdStrike Falcon® platform. (2023, September 21). crowdstrike.com. <https://www.crowdstrike.com/falcon-platform>