IEEE Access
Multidisciplinary ⁝ Rapid Review ⁝ Open Access Journal

Received July 29, 2021, accepted September 5, 2021, date of publication September 14, 2021, date of current version October 4, 2021.

*Digital Object Identifier 10.1109/ACCESS.2021.3112750*

# Encryption and Re-Randomization Techniques for Malware Propagation

**AHSAN RASHEED ABBASI[1], MEHREEN AFZAL[1], WASEEM IQBAL[1], SHYNAR MUSSIRALIYEVA[2], FAWAD KHAN[1], AND AWAIS UR REHMAN[3]**
[1]Department of Information Security, National University of Sciences and Technology, Islamabad 44000, Pakistan
[2]Department of Information Systems, Al-Farabi Kazakh National University, 050040 Almaty, Kazakhstan
[3]Department of Computer Science, University of South Asia, Lahore 54000, Pakistan

Corresponding author: Waseem Iqbal (waseem.iqbal@mcs.edu.pk)

**ABSTRACT** Encryption, which is essential for the protection of sensitive information can also transform any malicious content to illegible form, which can then reside in any network, undetected. Encryption of malicious payload is used by malware authors to mask their code, however, the objective of hiding the malicious code can be further improved by techniques of re-randomization. The concept of re-randomization using asymmetric cryptography has been emerged as a new area of interest for malware designers. Re-randomizing is a technique which can prevent detection of source path of a malware and makes it indistinguishable. This article extends the idea of using asymmetric cryptography for re-randomization and has proposed a novel scheme using Pailliar's asymmetric cryptosystem. Moreover, this research work illustrates the limitations of RSA for malware re-randomization. A comprehensive performance analysis of the re-randomization techniques for various malware payloads is also presented, which can be used for the detection of re-randomized malware effectively.

**INDEX TERMS** Paillier cryptosystem, RSA, ElGamal, homomorphic encryption, malware encryption, re-encryption, environmental keys.

## I. INTRODUCTION

The rapid evolution of internet has enabled individuals and devices from across the borders to connect and interact with each other. This inter-connectivity facilitated humans in numerous ways. However, such an increased reliance of corporate, healthcare, and individuals on the internet for performing even basic activities of life generated the volumes of confidential and personal information [9]. In the pursuit of this sensitive information, adversaries develop sophisticated techniques to exfiltrate the wealth of data from individuals and organizations.

To achieve the purpose of data exfiltration, denial of service or system disruption several types of malware have been designed and many successful attacks can be found in literature [1]–[3], [22]. On the other hand, malware detection techniques are also getting better day by day. In this tug of war, malware designers continuously work on maturing the mechanics of malware. Desired goals of malware designers include extending the life span of malware and its efficient propagation techniques along with the techniques to conceal its malicious behavior. Moreover, malware designers make focused efforts to defeat static [23] and dynamic [26] malware analysis techniques by using anti-sandbox, anti-debug and anti-analysis methods such as dead code insertion, encryption, etc. [5], [6], [8], [19].

Cryptographic techniques are extensively used by malware designers to conceal malware fingerprints. Indistinguishability, an important feature of cryptographic protocols ensures that an adversary does not have an added advantage to determine if the same message is encrypted twice. This remarkable feature exists in most of the probabilistic encryption schemes, which includes the classical ElGamal, and Paillier cryptosystems. Application of such schemes for re-randomization results in evading malware with greater accuracy because malware becomes indistinguishable from other data. Applying this technique successfully can for example, result in failure of intrusion detection system (IDS) or anti-malware for detection of malware. Idea of re-randomization technique based on ElGamal cryptosystem has been presented in [16]. The detail of re-randomization process is discussed in section III.

The associate editor coordinating the review of this manuscript and approving it for publication was Chi-Yuan Chen.

Importance of research in this direction also lies in the fact that to design strategy to detect and prevent malware, techniques used by malware designers need to be well understood. In this article, we have not only given the implementation methodology for ElGamal based re-randomization technique but have also explored the possibility or otherwise in-feasibility of other public key cryptographic techniques for re-randomization. A new scheme using Paillier cryptosystem is proposed along with the simulation on different malware samples. A comparative study of different schemes is also presented. Following are the contributions of this paper:

- This research work proposes and implements the novel scheme for malware encryption and re-randomization based on Paillier cryptosystem.
- We have evaluated and implemented ElGamal based scheme [16] for malware encryption and re-randomization.
- This work explains the infeasibility of classical RSA cryptosystem to re-randomize the malware payload.
- The paper contributions also includes the comprehensive simulation results on popular malware samples.

## II. RELATED WORK

Malicious use of encryption and malicious use of mathematics are evolving fields [7], [11] which originated in Young and Yung's earlier research about the use of public key cryptography for designing an offensive system for money extortion named as cryptovirology [12]. Eric Filiol describes, how encrypting malware payload prevents malware analyst to reverse engineer of binaries [15]. Markus Jakobosson's Asymmetric re-encryption [21] proves the input and output encryption co-related to the exact plaintext, without leakage of an information related to the plaintext to verifier or the server subset of the verifier. In 2004, Golle *et al.* [18] described a new primitive, universal re-encryption based on the ElGamal public key cryptographic algorithm which allows the re-randomization of ciphertext without knowing of the relevant private key. A high tech professional grade virus called Gauss [20] was detected in 2012-13 with an encrypted payload using data from the targeted victim's computer as the decryption key. No analyst can decrypt the payload and determine, what the payload will do until the virus is installed on the system of a targeted victim. Filiol presented the encryption of malicious software [14], and described that it is feasible to stop someone to analyze the software and reversing it, likely with the use Riordan and Schneier [28] keys to encrypt payload. In 2017, H.Galteland and G.Gjosteen worked on malware encryption and re-randomization [16]. They present technique in which malware author encrypts payload using unique key(s) generated from target environmental data. The ciphertext is re-randomized at each new node leads to form an indistinguishable variant of the same malware in the network that infects subsequent machines or devices without the knowledge of private key. The success of their model lies in the fact that different replicated variant of identical malware in the network boost the malware analyst's workload substantially and prevents analyst to defend some nodes in the network. However, they did not give any implementation results on their proposed ElGamal based scheme.

## III. MALWARE ENCRYPTION AND RE-RANDOMIZATION TECHNIQUE

H.Galteland and G.Gjosteen malware encryption scheme is based on malware attack process [16], shown in figure 1, in which the malware author with malware M and the malware source $M_{source}$, infects X initial nodes (in or outside the target network) with distinct variants of his malware. In response the X initial nodes infect subsequent machines in the network by propagating indistinguished copies of same malware.

Each and every direct connection to the malware source helps to increase the malware analyst's probability of finding the malware author's origin, because of that the malware author must perform as several additional infections as feasible and prefer indirect routes to the target node T.

Encrypting the payload protects malicious software code from being reversed and tries to obscure the malware author's intents. The malware encryption and re-randomization scheme [16] encrypt the malware payload using ElGamal public key algorithm [13] on source device. The malware attack model further distributed into malware encryption, decryption and re-randomization process.

The malware encryption process consists of malware and cleartext loader. The malware is the malicious code that need to execute on target system to compromise the opponent node(s). Before the malware attack process begin, The cleartext loader program scans and check the target system for environmental variables, which can be system variables, operating system unique parameters, path variables and other network triggers etc. The cryptographic hash function is used to transform the environmental variables to encryption key(s), instead of storing keys insides the malware payload. Later on, these environmental keys auto decrypt the malware payload on specific target node or network. The cleartext loader security depends on obfuscation scheme used by malware author.

On the target side, once the malware loader executes on a new device, it scans the compromised node's environmental variables and tries to decrypt the malicious payload through the derived keys. If the malware decryption succeeds, the malware payload will execute. Otherwise, malware re-randomization process will be executed.

The malware re-randomization process produces several indistinguishable variants of a malware, rather than replicating identical samples. The homomorphic property of public key encryption scheme is used to re-encrypt the ciphertext. The malware re-encryption scheme is based on universal re-encryption scheme [18] that uses the ElGamal public key algorithm to re-encrypt the ciphertext.

The re-randomization algorithm input an encrypted payload to generate the same malicious code, with some randomly chosen values in order to create a new ciphertext against same plaintext. The homomorphic encryption

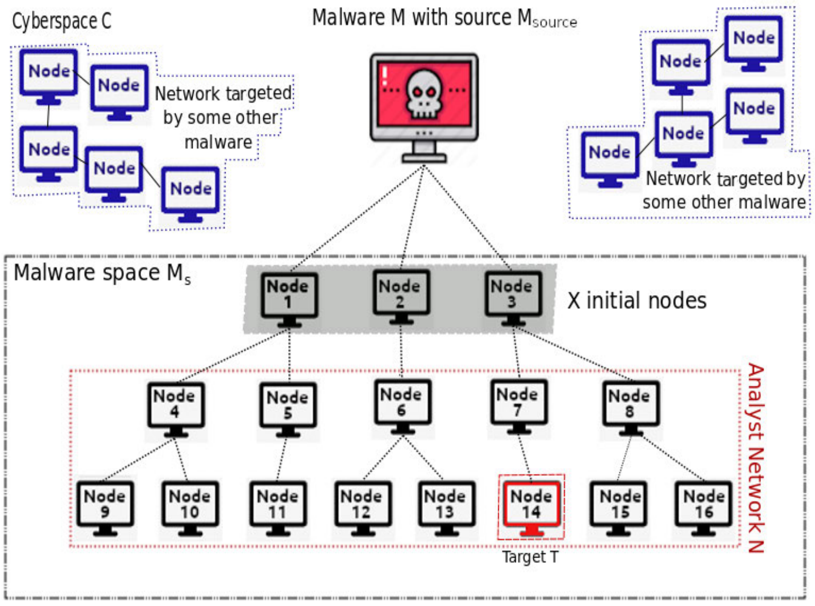

FIGURE 1. Malware propagation model.

property of ElGamal is used to generate different looking sample of same malware to infects X target nodes without any knowledge of private key.

The re-randomization process begins with scanning the target environmental data. The loader program generates the decryption key(s) from the environmental data, to decrypt the encrypted malicious payload and checks for decryption succeed or fail. If the malware decrypted successfully on a node, it will execute the malware. Otherwise, malware decryption fails and it will re-randomization of payload, as shown in figure 1. In this case, the re-randomization algorithm takes uniformly random values (ElGamal random factor) as input to re-encrypt the encrypted malicious payload to generate the indistinguishable sample of same malware without the knowledge of secret key. The re-randomization process uses the homomorphism of ElGamal that allow re-encryption of malicious payload. At last, it will drop the re-randomized payload to subsequent nodes(s) in the network.

From malware analyst prospective, whose responsibility is to protect computers in the network N from any potential threat of malware, and has comprehensive understanding of the environment that he protects. The malware analyst can observer the wider malware space $M_s$, to find the more malware $M$ samples. The malware author initiates with N distinct encrypted samples of a malware to infect X initial nodes on the network. Suppose, the malware analyst collects these initial X malware sample. Analyst's primary objective is to ensure that neither of these X initial variants compromise his devices in the network. To guarantee this, analyst approximately need K decryption for each of his N nodes. Hence, the malware analyst workload $W$ to analyze the malware sample is almost $W = XNK$. With the re-randomization of each encrypted malware sample the work load of the analyst will be $WR$.

## IV. PROPOSED IMPLEMENTATION OF ElGamal-BASED MALWARE ENCRYPTION AND RE-RANDOMIZATION SCHEME

The malware encryption and re-randomization scheme is built on public key cryptosystem ElGamal over group G of primitive order $p$, with generator $g$. The framework of proposed implementation of malware encryption and re-randomization is based on Galteland extended scheme [16]. The implemented scheme consists of four algorithms, the malware encryption algorithm $Enc_m$, the malware decryption algorithm $Dec_m$, the malware re-randomization algorithm $Re\text{-}randDec_m$ and malware re-randomized payload decryption algorithm $Re\text{-}rand\text{-}Dec_m$. The decryption algorithm $Dec_m$ will be executed, if the malicious payload is only encrypted (not re-randomized), otherwise (in case of malware re-randomization) the malware re-randomized payload decryption algorithm $Re\text{-}randDec_m$ will be executed.

### A. MALWARE ENCRYPTION ALGORITHM

The malware encryption algorithm $Enc_m$ [Algorithm 1] takes input a executable file (malware) and transforms it into bit string $m$ (bit stream) to encrypt the files of large size. The bit string is then padded with one 1 bit and length 0's bits. The number of zeros padded which is L, where $L = l*(n+1)+1$, $l$ is the message length and n is number of re-randomization that will perform on encrypted malicious payload. So the plaintext (padded malicious code) $m_{L_m}$ is bit string of length $L_m$.

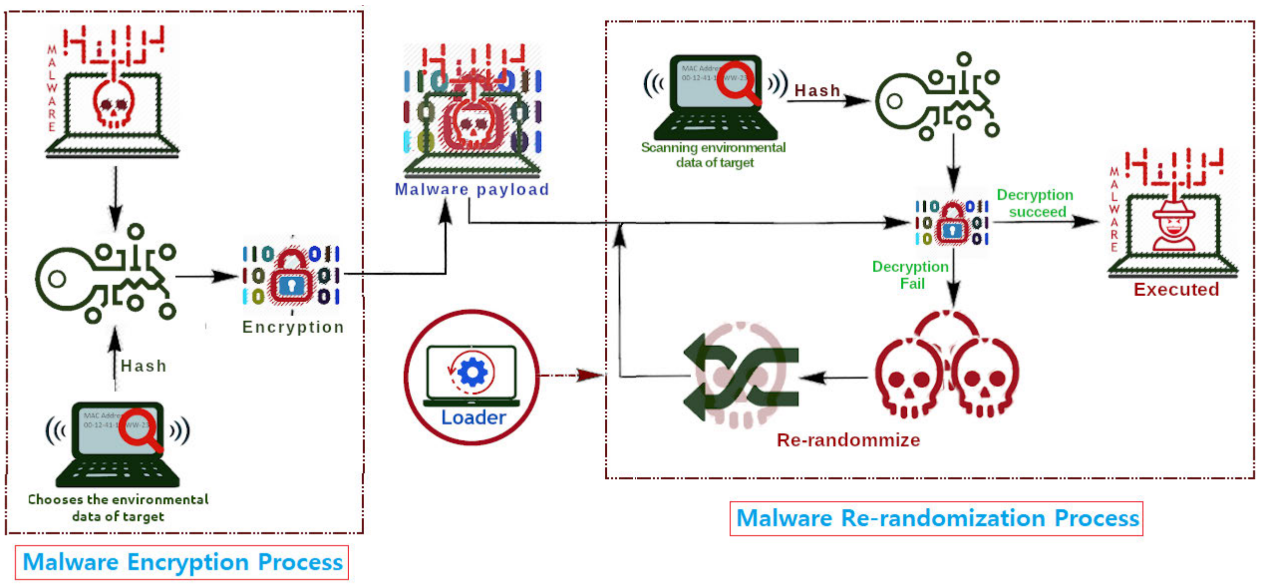
MALWARE
Hash
Encryption
Chooses the environmental data of target
Malware Encryption Process
Malware payload
Loader
Hash
Scanning environmental data of target
Decryption succeed
Decryption Fail
MALWARE
Executed
Re-randommize
Malware Re-randomization Process

**FIGURE 2.** Malware encryption and re-randomization process.

### 1) ENCRYPTION PARAMETERS

The malware encryption uses the standard ElGamal's parameters. Let $G$ represent the underlying group for ElGamal asymmetric cryptosystem with $p$ as order of group $G$ and generator denoted as $g$. The $p$ represents the big prime number and the generator $g$ is chosen at random number, such that $g < p$ and $gcd(p, g) = 1$. The values $r$ and $s$ are chosen as random such that $r, s \in (1, 2, \ldots (p-1))$.

### 2) KEY GENERATION

The plaintext $m$ is encrypted using encryption key $k$, generated from environmental data of the target node, where $k \in Z_p^*$. In our implementation, we use the MAC address of target system and transforms it into the key $k$.

**Algorithm 1** ElGamal Based Malware Encryption: $Enc_m$

```
Input: $m \leftarrow \{0, 1\}^l$
Output: $c$
1: $k = mac \mod p$
2: $m_{Lm} = m||1||0^{(L-1)}$
3: $r, s \xleftarrow{r} Z_p^*$
4: $\gamma \xleftarrow{r} Z_p^*$,
5: $\gamma_{Lm} = SHAKE\text{-}128\,(\gamma)$
6: $(c_1, c_2, c_3, c_4, c_5) = (g^r, g^{kr}, g^s, g^{ks}.\gamma, \gamma_{L_m} \oplus m_{L_m})$
7: $c = c_1||c_2||c_3||c_4||c_5$
8: return $c$
```

### 3) ENCRYPTION FACTOR $\gamma$

The encryption factor $\gamma$ is chosen as random. For encryption process the length of encryption factor $\gamma$ must be equal to $L_m$, where $L_m$ is the length of plain text $m$ with padded bit stream. For every variant, the value $\gamma$ will change as it is chosen as random. Hence, we need to define a way to transforms a random encryption factor $\gamma$ into a string $\gamma_{L_m}$ of arbitrary custom length $L_m$. To transform the encryption factor $\gamma$ into to a string $\gamma_{L_m}$, we employed the SHAKE hash algorithm.

**SHAKE-128**: The SHAKE algorithm [30] belongs to SHA-3 family of XOF (Extendable output functions) algorithms. The XOF is a hash function in which the message digest can be extended to any arbitrary custom length. The SHAKE-128 and SHAKE-256 are two standard SHA-3 XOF's. The suffix "128" and "256" specify the strength of the algorithm instead of indicating the message digest length as in other standard hashing algorithms. The SHAKE-128 and SHAKE-256 were the first XOFs to be standardized by NIST.

In our implementation of malware encryption algorithm, we input the random encryption factor $\gamma$ to SHAKE-128 algorithm and get our required message digest as string $\gamma_{L_m}$ of arbitrary length $L_m$. We have employed the Odzhan C implementation of SHAKE-128 [25] in our implementation. Hence, the SHAKE-128 algorithm is useful to get the desired length output in the encryption process.

### 4) ENCRYPTION

For encryption function [step 6:Algorithm 1], input comprises, ElGamal encryption parameters $p$ and $g$, padded malicious payload (plaintext) $m_{L_m}$, the key $k$, the random factor $r$, $s$ and the encryption factor $\gamma$, $\gamma_{L_m}$. The output of encryption algorithm [Algorithm 1] is ciphertext $c$, which is calculated by concatenating the ciphertext instances $c_1$, $c_2$, $c_3$, $c_4$, and $c_1$.

## B. MALWARE DECRYPTION ALGORITHM

The decryption algorithm $Dec_m$ [Algorithm 2] takes as input the ciphertext C. The decryption process is executed at the target node, so there is a need to auto-decrypt the malicious payload on the target machine using the secret key $k$. As the key $k$ is derived from target node's MAC address of the target node. Hence, we need to extract the target MAC address and calculate the key $k$. To decrypt the encrypted malicious payload, there is a requirement to verify the target node, by checking $c_1^k = c2$, where $c_1 = g^r$ and $c_2 = g^{kr}$. If the verification fails than moves onto re-randomization algorithm to target subsequent node without executing, otherwise, the payload will be decrypted and executed.

**Algorithm 2** ElGamal Based Malware Decryption: *Dec*$_m$

```
Input: $c$
Output: $m$
1: $k = mac \mod p$
2: $\gamma = c_3^{-k}.c_4$
3: $\gamma_{Lm} = SHAKE - 128(\gamma)$
4: $m_{Lm} = c_5 \oplus \gamma_{L_m}$
5: $m = m_{Lm} - (1||0^L)$
6: return $m$
```

To decrypt the malicious payload, the first step is to extract the $\gamma$ from the ciphertext, as the cipher text $c = c_1||c_2||c_3||c_4||c_5$,hence

$$\begin{aligned} c_3^{-k}.c_4 &\implies g^{-ks}.g^{ks} \mod p \\ &\implies \gamma \end{aligned}$$

After determining $\gamma$, $\gamma_{L_m}$ will be evaluated using the XOF SHAKE-128. The length of ciphertext object $c_5$ is equal to the padded plaintext length $L_m$. The length $L_m$ and $\gamma$ is input to SHAKE-128 to get $\gamma_{L_m}$ of length $L_m$.

To decrypt the padded plaintext $m_{L_m}$, XOR the ciphertext object $c_5$ and $\gamma_{L_m}$ To determine the padded bit stream length, search the first '1' from the end of bit stream of $\gamma_{L_m}$ and discard the padded bit from the $\gamma_{L_m}$, to get the original plaintext. As in encryption algorithm, the plaintext is padded with tail of zeros and a single 1. To execute the malware payload, write the decrypted data to a file. Hence, we get the original malicious payload plaintext m. Before executing the malware, there is a need to verify the malicious payload using the standard signature of underlying file format. On successful verification, malware payload will be executed.

## C. MALWARE RE-RANDOMIZATION ALGORITHM

The re-randomization algorithm *Rr-rand*$_m$ [Algorithm 3] takes as input the ciphertext $c$ and the public parameter $p$. Decryption process is executed on the node which prompts the decryption fails or invalid private key $k$ for decryption. In consequence, before targeting the new node, re-encrypt the ciphertext without knowledge of private key $k$. The re-randomization algorithm chooses two new random factors $r'$ and $s'$. The encryption factors $\gamma'$ to re-encrypt the ciphertext $c$, is also chosen at random.

The value of $\beta_{L_\beta}$ with length $L_\beta$ is evaluated as, $\beta_{L_\beta} = c_3||c_4||c_5$. Where the ciphertext $c$ objects, $c_3 = g^s$, $c_4 = g^{ks}.\gamma$ and $c_5 = \gamma_{L_m} \oplus m_{L_m}$. The $\gamma'_{L_\beta}$ will be computed by using the XOF SHAKE128. The output of re-randomization algorithm *Re-rand*$_m$ is ciphertext $c'$.

**Algorithm 3** ElGamal Based Malware Re-Randomization: *Re-rand*$_m$

```
Input: $c, p$
Output: $c'$
1: $\beta_{L_\beta} = c_3||c_4||c_5$
2: $r', s' \overset{r}{\leftarrow} Z_p^*$
3: $\gamma' \overset{r}{\leftarrow} Z_p^*$
4: $\gamma'_{L_\beta} = SHAKE\text{-}128(\gamma')$
5: $(c'_1, c'_2, c'_3, c'_4, c'_5) = (c_1^{r'}, c_2^{r'}, c_1^{s'}, c_2^{s'}.\gamma', \gamma'_{L_\beta} \oplus \beta_{L_\beta})$
6: $c' = (c'_1||c'_2||c'_3||c'_4||c'_5)$
7: return $c'$
```

## D. RE-RANDOMIZED MALICIOUS PAYLOAD DECRYPTION

The re-randomized ciphertext decryption algorithm *Re-randDec*$_m$ [Algorithm 4], is modified form of decryption algorithm *Dec*$_m$ with some inclusion. The *Re-randDec*$_m$ algorithm takes as input the public parameters $p$ and $g$ and the ciphertext $c'$. The secret key $k$ is generated from the target node's MAC address to decrypt the payload. To verify the target node for decryption of payload we check ${c'_1}^k = c'_2$, which is: $(g^{rr'})^k = g^{krr'}$. The verification fails means, the underlaying node is not the target of the malware.

**Algorithm 4** ElGamal Based Re-Randomized Ciphertext Decryption: *Re-randDec*$_m$

```
Input: $c', p$
Output: $m$
1: $k = mac \mod p$
2: $\gamma' = {c'_3}^{-k}.c'_4$
3: $\gamma'_{L_\beta} = SHAKE(MD5(\gamma'))$
4: $\beta'_{L_\beta} = c'_5 \oplus \gamma'_{L_\beta}$
5: $\gamma = c_3^{-k}.c_4$
6: $\gamma_{Lm} = SHAKE - 128(MD5(\gamma))$
7: $m_{Lm} = c_5 \oplus \gamma_{L_m}$
8: $m = m_{Lm} - (1||0^L)$
9: return $m$
```

On verification failure, the decryption process will stop, and re-randomization algorithm will be executed to target the other node(s). On successful verification, the payload will be decrypted and executed. To decrypt the payload, the first step is to determine $\gamma'$ from ciphertext $c'$, as:

$$\begin{aligned} {c'_3}^{-k}.c'_4 &= c_1^{s'(-k)}.c_2^{s'}\gamma' \\ &= g^{rs'(-k)}.g^{krs'}\gamma' \\ &= \gamma' \end{aligned}$$

Now, we can compute the $\gamma'_{L_\beta}$ using the input $\gamma'$ and $L_\beta$ to SHAKE128. The length $L_\beta$ is same as the length of $c'_5$. The $\beta_{L_\beta}$ is computed by XOR the ciphertext $c'_5$ and $\gamma'_{L_\beta}$.

The $\beta_{L_\beta}$ is concatenation of ciphertext's $c$ instances in Rr-rand$_m$ [Algorithm 3], as, $c_3||c_4||c_5$. In other words, $\beta_{L_\beta} = g^s \,||\, g^{ks}.\gamma \,||\, \gamma_{L_m} \oplus m_{L_m}$. We can determine the length of $g^s$ and $g^k s.\gamma$, as they have equal in length with ciphertext $c'$ instances $c'_3$ and $c'_4$ respectively. Hence, we can find $c_3$, $c_4$ and $c_5$ from $\beta_{L_\beta}$. This procedure continues for $n$ iterations, where n is the number of re-randomization performed on encrypted payload.

Next step is to computer $\gamma$, from the ciphertext $c$ instances, $c_3$ and $c_4$ using the secret key $k$, as:

$$\begin{aligned} c_3^{-k}.c_4 &= g^{-ks}.g^{ks} \mod p \\ &= \gamma \end{aligned}$$

Now, we can computer $\gamma_{L_m}$ from $\gamma$ and $c_5$'s length $L_m$ using SHAKE-128. The malicious payload's padded plaintext $m_{L_m}$ is computed by applying XOR operation on $c_5$ and $\gamma_{L_m}$.

The padded bit stream length can be determined by searching the first '1' from the end of bit stream(right to left) of $\gamma_{L_m}$ and discard the padded bit from the $\gamma_{L_m}$ to get the original plaintext $m$. As in encryption algorithm, the plaintext is padded with tail of zeros and exactly a single 1. Write the decrypted data to a file to execute and launch the malware. The re-randomization decryption algorithm needs to execute $n$ time on the encrypted malicious payload, where $n$ is the number of re-randomizations performed on the malicious payload. Before execution of the malware, the standard executable file format signature is employed to validate the malicious payload. Malicious software payload will be executed on successful validation.

## V. LIMITATION OF RSA FOR MALWARE RE-RANDOMIZATION

Partially homomorphic encryption (PHE) [29] helps to keep confidential information protected by enabling only specific mathematical operations on encrypted data to be performed. This implies that for an infinite number of occasions a single operation, either multiplication or addition, could be conducted on the encrypted data. RSA public key cryptosystem is a PHE scheme, which is commonly used to establish secure connections through SSL / TLS. Some other examples of PHE scheme are ElGamal public key cryptosystem (a multiplicative homomorphic scheme) and Paillier cryptosystem (an additive homomorphic scheme). RSA public key cryptosystem supports the multiplicative homomorphic encryption. For malware re-randomization, the requirement is to re-encrypt the encrypted payload or ciphertext without knowledge of secret key. The both ElGamal and Paillier's random factor $r$ helps us to re-encrypt the encrypted malicious payload without revealing the plaintext. In case of RSA, by default the textbook RSA is not semantically secure, as it is deterministic encryption scheme. Although, RSA has homomorphic property but is not sufficient to re-encrypt the ciphertext because there is no random factor, that helps us to re-randomize the ciphertext as in ElGamal and Paillier.

RSA can be either semantically secure or homomorphic, but not both. In practice, before encrypting the plaintext message $m$, RSA can add the randomness (e.g. RSAES-OAEP) [17], which provide semantic security, but completely loses the homomorphic property.

Some other approaches have been used by researchers, to use RSA for re-encryption of ciphertext in proxy re-encryption. Wang *et al.* [31], split the algorithm into two parts to ensure the re-randomization for proxy re-encryption. The first part of the algorithm will use the original private key as well as the fresh public key and create a kind of intermediate key. The second part ensures that the key will then be circulated to untrusted entities which use the intermediate key and the new public key to update their encrypted data to the new key pair. However, in this case, we need to auto-decrypt the malicious payload on the target node by using environmental keys. It is impractical to generate the new key pair to re-encrypt the ciphertext. The reason is, the private key is not appended with malware payload or in text loader. It will be generated on run time from target node. Hence, it is not feasible to use RSA cryptosystem to re-randomize the malware payload.

## VI. PAILLIER-MALWARE ENCRYPTION AND RE-RANDOMIZATION

The proposed scheme is based on the probabilistic public key encryption algorithm, Paillier [27]. The goal is to prevent malware payload from analysis by increasing the malware analyst's workload. Encrypting malware payload obstruct to identify malware author's intentions. The malware re-randomization or re-encryption aid to hide the identity of malware author and make each malware sample indistinguishable. We proposed a framework for malware encryption and re-randomization of malicious payload on the target node without revealing the plaintext or the private key. The secret key is generated from the target environmental data, to encrypt the malicious payload using Paillier encryption algorithm. To execute malicious payload on the target node, there is a need to extract the environmental key and use it to decrypt the malware payload. On successful decryption the malicious payload will execute. Otherwise, re-encrypt the encrypted payload, and transmit it to next node to find the target and execute. The proposed scheme follow the malware propagation model, describe in Section 2. We have implemented the proposed scheme based on Paillier for malware encryption and re-randomization, using C language on Linux platform. For big integer value(greater than 8 bytes), we have used the GMP C library. The proposed scheme includes three main algorithms, malware encryption algorithm, malware re-randomization algorithm and the malware decryption algorithm. The malware encryption and decryption algorithms have sub algorithms, the encryption key generation and decryption key generation algorithms, respectively.

### A. MALWARE ENCRYPTION

The malware encryption process aids the malware writers by encrypting the malicious payload to maximize the workload for analysis. Encrypting the payload restricts the malicious software from being reversed by an analyst and obscures the malware author's desires. Environmental data is collected from the target network and therefore could consist of IP address, directory paths, PATH variables etc. In this work, we use MAC address as environmental variable to generate encryption and decryption keys. The mechanism for malware encryption consists of the payload and the cleartext loader for malware. The malware payload consists of malicious code to be run on target system in order to access the node(s) of the challenger. The loader software scans and tests environmental variables on the target system and defines how these variables can be converted to produce encryption or decryption keys rather than keeping keys within the payload of malware. The encryption process consists of the encryption key generation algorithm and payload encryption algorithm.

#### 1) ENCRYPTION KEY GENERATION ALGORITHM

The encryption key generation algorithm $KeyGen_{Enc}$ [Algorithm 5] is based on Paillier key generation, which computes two prime numbers $p$ and $q$ (of equal size) as a input, where $p$ and $q$ are relatively prime to each other. On the other side, the secret key $\lambda$ of Paillier Cryptosystem [27] depends on $p$ and $q$, and can be computed as, $\lambda = lcm(p-1,\ q-1)$.

**Algorithm 5** Paillier Based Encryption Key Generation: $KeyGen_{Enc}$

```
Input: $mac\_addr$
Output: $pub_k(n, g)$
1: $p \overset{P}{\leftarrow} mac\_addr$
2: $q \overset{P}{\leftarrow} mac\_addr \quad ; q < p$
3: $n = p * q$
4: $g = n + 1$
5: return $n, g$
```

But, in our case it is impractical. Because, for malware propagation, our requirement is to auto-decrypt the malicious payload on the target system. It can be achieved using the key(s) generated from target environmental data. This implies that the decryption key $\lambda$ must be depended on the target environmental data, and can be re-generated.

To overcome this problem, we generate the prime $p$ and $q$, from target environmental data (MAC address of target system), and computes a modulus $n = p * q$. Now, choose a random number $g$ such that $g \in Z^*_{n^2}$ and the order of $g$ is multiple of $n$. Selecting $g = n + 1$, is effective choice and can be easily computed [10]. The output of Encryption Key Generation algorithm $KeyGen_{Enc}$ is public (encryption) key pair $(g, n)$.

#### 2) ENCRYPTION ALGORITHM

The encryption algorithm $\Pi Enc_m$ [Algorithm 6] takes the malware executable file as input and transform it into array of string, as plaintext message $m$. The encryption algorithm $\Pi Enc_m$ uses the public key pair $(g, n)$, generated from key generation algorithm $KeyGen_{Enc}$ to encrypt the plaintext. Select a Paillier's random factor $r$, where $r \in Z^*_{n^2}$.

Paillier cryptosystem allows encrypting integers modulo $n$. Therefore, if input plaintext message $m$ is bigger than $n$, encrypting it will lose most of the plaintext message $m$, only $m \mod n$ is retrieved through decryption. In case of malware encryption, it is likely that $m > n$.

**Algorithm 6** Paillier Based Malware Encryption: $\Pi Enc_m$

```
Input: $m, pub_k(n, g)$
Output: $c$
1: $r \overset{r}{\leftarrow} Z^*_{n^2}$
2: if $m > n$ then
3:     $m = m_1||m_2||\ldots||m_z$ ; $z = len(m)/len(n)$
4:     $c_i = g^{m_i} * r^n \mod n^2 \quad ; i = 1, 2, \ldots, z$
5:     $c = c_1||c_|| \ldots ||c_z$
6: else
7:     $c = g^m * r^n \mod n^2$
8: end if
9: return $c$
```

To encrypt a message bigger than $n$, we break it into $z$ blocks, which encrypt separately. Where z is number of chunks or blocks and each message block $m_i < n$.
We use public key pair $(g, n)$ and random factor $r$ to encrypt message $m$ or message blocks $m_i's$. The output of $Enc_m$ is ciphertext $c$.

### B. MALWARE RE-RANDOMIZATION

Re-randomizing the malicious payload, generates several indistinguishable variants of a malicious software rather than just replicating the same samples. Re-randomization of malware can be achieved using public key cryptographic algorithm's homomorphic property. The inputs of the re-randomization algorithm $\Pi Re\text{-}rand_m$ [Algorithm 7] includes an encrypted payload or cipher text $c$ and public key pair $(n, g)$ to create the exact malicious payload (encrypted), with certain randomly selected values. Paillier's homomorphism and probabilistic property is used to create different looking samples of the same malware.

**Algorithm 7** Paillier Based Malware Re-Randomization: $\Pi Re\text{-}rand_m$

```
Input: $c, pub_k(n, g)$
Output: $c'$
1: $c = c_1||c_|| \ldots ||c_z$
2: $r' \overset{r}{\leftarrow} Z^*_{n^2}$
3: $c'_i = c_i * r'^n \mod n^2$
4: $c' = c'_1||c'_|| \ldots ||c'_z$
5: return $c'$
```

The re-randomization algorithm $\Pi Re\text{-}rand_m$ computes a random factor $r'$, such that $r' \in Z^*_{n^2}$ and re-randomize the ciphertext $c_i$'s, as shown in $\Pi Re\text{-}rand_m$ [Algorithm 7] and generate output re-randomized ciphertext $c'$.

### C. MALWARE DECRYPTION

The malware decryption process begins with scanning the target environmental data. After collecting required environmental data, generates the private key from the acquired data, and decrypt the malicious payload using the obtained private key. The decryption process consists of two algorithms, the private key generation and the decryption of ciphertext.

#### 1) DECRYPTION KEY GENERATION

The decryption key generation algorithm $KeyGen_{Dec}$ [Algorithm 8] output the public key pair $(n, g)$ and private key $\lambda$. The $KeyGen_{Dec}$ uses the MAC address of target system to generate the keys. The public key pair is generated with the same process as in $KeyGen_{Enc}$ [Algorithm 5]. The private key $\lambda$ is generated by using the parameters $p$ and $q$ as shown in $KeyGen_{Dec}$ [Algorithm 8].

**Algorithm 8** Paillier Based Decryption Key Generation: $KeyGen_{Dec}$

```
Input: $mac\_addr$
Output: $pub_k(n, g), pri_k(\lambda)$
1: $p \overset{P}{\leftarrow} mac\_addr$
2: $q \overset{P}{\leftarrow} mac\_addr \quad ; q < p$
3: $n = p * q$
4: $g = n + 1$
5: $\lambda = LCM(p - 1)(q - 1)$
6: return $n, g, \lambda$
```

#### 2) DECRYPTION ALGORITHM

The decryption procedure is same for both encrypted only and re-randomize or re-encrypted payload, so $c' = c$. The decryption algorithm takes as input the ciphertext $c$ and the keys generated from $KeyGen_{Dec}$. The Decryption process shown in $\Pi Dec_m$ [Algorithm 9].

**Algorithm 9** Paillier Based Malware Decryption: $\Pi Dec_m$

```
Input: $c, pub_k(n, g), pri_k(\lambda) \quad //* c == c'$
Output: $m$
1: $c = c_1||c_|| \ldots ||c_z$
2: $m_i = \frac{L(c_i^{\lambda} \mod n^2)}{L(g^{\lambda} \mod n^2)}$
3: where: $L(x) = \frac{(x-1)}{n}, \quad \forall x \in Z_{n^2}^*$
4: $m = m_1||m_2|| \ldots ||m_z$
5: return $m$
```

After decryption of ciphertext instances $c_i$'s, we get the plaintext instances $m_i$'s. We concatenate all the $m_i$'s to get the plaintext message $m$ and write the result in file to verify and executes the malware payload.

In our implementation, the loader program checks for signature of malicious software samples to validate the results of decryption. For windows, the executable (exe) file format starts with $0 \times$ 4D5A (MZ) signature. So, the text loader program will check the.exe file signature to validate the decryption for windows based malware. Otherwise, the underlying malware will belong to the deb or bin file format if malware is based on Linux. The signature of deb file format is $0 \times$ 213C617263683E! (< arch>.) and $0 \times$ 7F454C46 (.ELF) is the signature of the binary executable file. On successful verification of malware payload, the loader program execute the malicious payload.

TABLE 1. Malware samples.

| Malware | Platform | Signature | Size |
|---|---|---|---|
| WannaCry[36] | Windows | MZ | 3.5 (MB) |
| Zeus[35] | Windows | MZ | 252.9(KB) |
| GandCrab[24] | Windows | MZ | 124.4(KB) |
| Kovtreer[32] | Windows | MZ | 431.9 (KB) |
| Wirenet[33] | Linux | .ELF | 64.4(KB) |
| Encoder[34] | Linux | .ELF | 317.5(KB) |
| Dendroid[4] | Android | PK | 942.8 (KB) |

## VII. PERFORMANCE EVALUATION

In this section, we present experimental setup and results of our proposed implementation of ElGamal based scheme and proposed Paillier based malware encryption and re-randomization scheme.

### A. EXPERIMENTAL SETUP

The implementation of both ElGamal and Paillier based malware encryption schemes is on C language for Linux and windows platform. The code has been implemented on Windows 10 (exe) and Ubuntu 16.04(ELF). The Experiments are performed on Intel(R) Xeon(R) CPU E5-1660 v3 with a 3.00 GHz and 16 GB of memory. We import open source OpenSSL-MD5 directive for MD5 hash. We implement indigenous code ElGamal and Pailler for customization and efficiency. The loader program is able to re-randomized the malware payload, regardless of underlying architecture or operating system. The loader program is implemented on Linux and windows platform which can encrypt and re-randomize any type of file i.e., exe, apk, DLL, elf etc. We used some popular Linux, Windows and android malware samples in our experiments, as shown in table 1.

### B. EXPERIMENTAL RESULTS

We evaluated the performance of our proposed implementation of ElGamal based scheme and proposed Paillier based scheme. We analyze and compare performance of encryption, re-randomization and decryption algorithms of both scheme. The time T1, T2, T3 and T4 represent the approximation of processor time (in sec) used by algorithms accordingly to variations in parameters. These parameters are random factor used by encryption algorithm in both schemes. The AVG is the average time of T1, T2, T3 and T4.

Table 2 shows the experimental results and comparison of performance between encryption algorithm of ElGamal based scheme $Enc_m$ [Algorithm 1] and Paillier based scheme $\Pi Enc_m$ [Algorithm 6] for malicious payload encryption of different popular malware sample. Table 2 shows that the

**TABLE 2.** ElGamal vs. Paillier encryption algorithm.

| Malware | ElGamal-based Scheme | | | | | Paillier based Scheme | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | T1 | T2 | T3 | T4 | AVG | T1 | T2 | T3 | T4 | AVG |
| WannaCry[36] | 2.392 | 2.384 | 2.356 | 2.356 | 2.365 | 6.977 | 6.968 | 6.965 | 6.963 | 6.968 |
| Zeus[35] | 0.171 | 0.162 | 0.164 | 0.17 | 0.167 | 0.473 | 0.479 | 0.475 | 0.479 | 0.477 |
| GandCrab[24] | 0.086 | 0.084 | 0.085 | 0.081 | 0.084 | 0.232 | 0.231 | 0.232 | 0.23 | 0.231 |
| Koveter[32] | 0.003 | 0.002 | 0.002 | 0.003 | 0.003 | 0.819 | 0.824 | 0.829 | 0.836 | 0.832 |
| Wirenet[33] | 0.041 | 0.039 | 0.043 | 0.044 | 0.042 | 0.119 | 0.116 | 0.118 | 0.119 | 0.118 |
| Encoder[34] | 0.209 | 0.208 | 0.207 | 0.21 | 0.209 | 0.587 | 0.588 | 0.59 | 0.594 | 0.590 |
| Dendroid[4] | 0.637 | 0.634 | 0.643 | 0.629 | 0.636 | 1.891 | 1.884 | 1.879 | 1.883 | 1.884 |

**TABLE 3.** ElGamal vs. Paillier decryption algorithm.

| Malware | ElGamal-based Scheme | | | | | Paillier based Scheme | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | T1 | T2 | T3 | T4 | AVG | T1 | T2 | T3 | T4 | AVG |
| WannaCry[36] | 7.982 | 16.6 | 2.532 | 13.934 | 10.262 | 4.152 | 4.136 | 4.14 | 4.141 | 4.142 |
| Zeus[35] | 4.572 | 1.001 | 9.98 | 0.721 | 4.068 | 1.384 | 1.379 | 1.38 | 0.192 | 1.084 |
| GandCrab[24] | 3.304 | 5.01 | 12.114 | 2.051 | 5.620 | 1.282 | 1.274 | 1.272 | 1.269 | 1.274 |
| Koveter[32] | 4.163 | 15.69 | 7.354 | 16.696 | 10.976 | 1.532 | 1.54 | 1.531 | 1.536 | 1.535 |
| Wirenet[33] | 2.726 | 8.657 | 2.798 | 12.271 | 6.613 | 1.232 | 1.225 | 1.23 | 1.224 | 1.228 |
| Encoder[34] | 15.416 | 13.982 | 16.68 | 3.579 | 12.414 | 1.437 | 1.434 | 1.425 | 1.424 | 1.430 |
| Dendroid[4] | 8.406 | 0.538 | 6.057 | 1.826 | 4.207 | 1.965 | 1.968 | 1.961 | 1.949 | 1.961 |

**TABLE 4.** ElGamal vs. Paillier re-randomization algorithm.

| Malware | ElGamal-based Scheme | | | | | Paillier based Scheme | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | T1 | T2 | T3 | T4 | AVG | T1 | T2 | T3 | T4 | AVG |
| WannaCry[36] | 1.540 | 1.532 | 1.534 | 1.521 | 1.532 | 1.001 | 1.003 | 1.004 | 1.003 | 1.003 |
| Zeus[35] | 0.105 | 0.117 | 0.113 | 0.111 | 0.112 | 0.072 | 0.072 | 0.072 | 0.071 | 0.072 |
| GandCrab[24] | 0.055 | 0.05 | 0.055 | 0.051 | 0.053 | 0.035 | 0.034 | 0.036 | 0.035 | 0.035 |
| Koveter[32] | 0.002 | 0.001 | 0.002 | 0.002 | 0.002 | 0.127 | 0.123 | 0.123 | 0.123 | 0.124 |
| Wirenet[33] | 0.028 | 0.028 | 0.028 | 0.028 | 0.028 | 0.018 | 0.018 | 0.018 | 0.018 | 0.018 |
| Encoder[34] | 0.137 | 0.134 | 0.137 | 0.137 | 0.136 | 0.091 | 0.091 | 0.091 | 0.089 | 0.091 |
| Dendroid[4] | 0.407 | 0.412 | 0.409 | 0.402 | 0.408 | 0.266 | 0.27 | 0.269 | 0.268 | 0.268 |

performance of ElGamal based malware encryption scheme is better than Paillier based malware encryption scheme. The reason is, the Paillier encryption scheme divide the plaintext $m$ into multiple blocks before encryption and also merge the blocks after applying encryption. However, the encryption algorithm is executed on source node, not on the target node. Hence, the performance of malware decryption, re-randomization, and re-randomized payload decryption is more crucial than the performance of encryption algorithm.

Table 3, shows the performance of decryption algorithm of both ElGamal $Dec_m$ [Algorithm 2] and Paillier $\Pi Dec_m$ [Algorithm 9] based scheme for encrypted malware payload decryption. The Paillier decryption and re-randomized ciphertext decryption algorithm is same. Here, we analyze the performance of $\Pi Dec_m$ [Algorithm 9] with encrypted (only) ciphertext. By analyzing the Table 3, which shows time taken by Paillier based malware decryption scheme is less than time taken by ElGamal based malware decryption scheme to decrypt the encrypted malicious payload. The variation in time T1, T2, T3 and T4 of ElGamal based malware decryption scheme is due to the use of random factor $r$ and $s$ in encryption algorithm.

The encrypted malware payload re-randomization is performed on the target node, to form indistinguishable encrypted malware sample to target subsequent nodes in the network.

Table 5, shows the performance analysis of ElGamal based re-randomized malware decryption algorithm $Re\text{-}randDec_m$ [Algorithm 4] and Paillier based re-randomized malware decryption algorithm $\Pi Dec_m$ [Algorithm 9]. From Table 5, it is observed that the Paillier based re-randomized payload decryption algorithm has faster execution time to decrypt the malicious payload, as compare to ElGamal based re-randomized payload decryption algorithm. The use of random factor $r$, $s$, $r'$ and $s'$ in ElGamal based encryption and re-randomization algorithms causes more fluctuation in the time T1, T2, T3 and T4 of ElGamal based malware re-randomized malicious payload decryption scheme.

In Table 4, the performance results show that the re-randomization time of Paillier based malware re-randomization algorithm $\Pi Re\text{-}rand_m$ [Algorithm 7] is also lower than the re-randomization time of ElGamal based malware re-randomization algorithm $\Pi Rr\text{-}rand_m$ [Algorithm 3].

The performance results show that the Paillier based malware encryption and re-randomization scheme is computationally inexpensive when compared with ElGamal based malware encryption and re-randomization scheme.

**TABLE 5.** ElGamal vs. Paillier re-randomized payload decryption algorithm.

| Malware | ElGamal-based Scheme | | | | | Paillier based Scheme | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | T1 | T2 | T3 | T4 | AVG | T1 | T2 | T3 | T4 | AVG |
| WannaCry[36] | 9.252 | 22.485 | 9.824 | 23.430 | 16.248 | 4.138 | 4.166 | 4.127 | 4.168 | 4.150 |
| Zeus[35] | 8.837 | 14.454 | 15.172 | 14.175 | 13.160 | 1.383 | 1.374 | 1.387 | 1.388 | 1.383 |
| GandCrab[24] | 11.92 | 11.844 | 15.737 | 12.458 | 12.990 | 1.276 | 1.281 | 1.276 | 1.267 | 1.275 |
| Koveter[32] | 8.813 | 31.036 | 20.678 | 20.012 | 20.135 | 1.533 | 1.539 | 1.54 | 1.534 | 1.537 |
| Wirenet[33] | 4.971 | 24.674 | 7.753 | 29.429 | 20.619 | 1.224 | 1.22 | 1.218 | 1.225 | 1.222 |
| Encoder[34] | 24.061 | 21.096 | 16.177 | 7.815 | 17.287 | 1.445 | 1.44 | 1.438 | 1.447 | 1.443 |
| Dendroid[4] | 16.188 | 8.459 | 17.39 | 13.605 | 13.911 | 1.986 | 1.972 | 1.963 | 1.971 | 1.973 |

## VIII. CONCLUSION

This work gives a new re-randomization technique based on Paillier cryptosystem for malware propagation. We have also proposed the implementation of an existing scheme based on ElGamal cryptosystem for malware encryption and re-randomization. A simulation of both schemes on several malware samples is executed and comparison of the results show that, ElGamal based scheme's encryption algorithm is more efficient than Paillier based scheme's encryption. Whereas, the malware decryption, randomization and re-randomized malicious payload decryption algorithms of Paillier based scheme are computationally inexpensive, as compared to ElGamal based scheme. These algorithm needs to be more efficient due to execution on target environment. This makes our proposed Paillier based re-randomization scheme more suitable for malware propagation. Future research should examine the robustness against different attacks and security analysis of both Paillier and ElGamal based schemes.

## REFERENCES

[1] *Behind the scenes of GandCrab's Operation*. Accessed: Jun. 11, 2021. [Online]. Available: https://www.virusbulletin.com/virusbulletin/2020/01/behind-scenes-gandcrabs-operation/

[2] (May 2013). *Cryptolocker Ransomware Infections*. [Online]. Available: https://us-cert.cisa.gov/ncas/alerts/TA13-309A

[3] (Jul. 2018). *Emotet Malware (AA20-280A)*. [Online]. Available: https://us-cert.cisa.gov/ncas/alerts/aa20-280a

[4] Ashishb. (Apr. 2016). *Android-Malware/Tree/Master/Dendroid*. [Online]. Available: https://github.com/ashishb/android-malware/tree/master/Dendroid

[5] C. Barria, D. Cordero, C. Cubillos, and R. Osses, "Obfuscation procedure based in dead code insertion into crypter," in *Proc. 6th Int. Conf. Comput. Commun. Control (ICCCC)*, May 2016, pp. 23–29.

[6] B. Bashari Rad, M. Masrom, and S. Ibrahim, "Camouflage in malware: From encryption to metamorphism," *Int. J. Comput. Sci. Netw. Secur.*, vol. 12, pp. 74–83, Jan. 2012.

[7] P. Beaucamps and E. Filiol, "On the possibility of practically obfuscating programs towards a unified perspective of code protection," *J. Comput. Virol.*, vol. 3, no. 1, pp. 3–21, Mar. 2007.

[8] J.-M. Borello and L. Mé, "Code obfuscation techniques for metamorphic viruses," *J. Comput. Virol.*, vol. 4, no. 3, pp. 211–220, Aug. 2008.

[9] M. Chen, S. Mao, and Y. Liu, "Big data: A survey," *Mobile Netw. Appl.*, vol. 19, no. 2, pp. 171–209, Apr. 2014.

[10] I. Damgård, M. Jurik, and J. B. Nielsen, "A generalization of Paillier's public-key system with applications to electronic voting," *Int. J. Inf. Secur.*, vol. 9, no. 6, pp. 371–385, Dec. 2010.

[11] A. Desnos, "Implementation of K-ary viruses in Python," Hack.lu, Tech. Rep., 2009.

[12] T. Dullien and S. Porst, "REIL: A platform-independent intermediate representation of disassembled code for static code analysis," Tech. Rep., 2009.

[13] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," in *Adv. Cryptol.*, G. R. Blakley and D. Chaum, Eds. Berlin, Germany: Springer, 1985, pp. 10–18.

[14] E. Filiol, "Strong cryptography armoured computer viruses forbidding code analysis: The Bradley virus," INRIA, France, Res. Rep. RR-5250, 2004.

[15] E. Filiol, "Malicious cryptography techniques for unreversable (malicious or not) binaries," 2010, *arXiv:1009.4000*. [Online]. Available: http://arxiv.org/abs/1009.4000

[16] H. Galteland and K. Gjøsteen, "Malware encryption schemes-rerandomizable ciphertexts encrypted using environmental keys," IACR Cryptol. ePrint Arch., Tech. Rep., 2017, p. 1007.

[17] S. Goldwasser, "Probabilistic encryption: Theory and applications (partial information, factoring, pseudo random bit generation)," Ph.D. dissertation, Univ. California, Berkeley, CA USA, 1984.

[18] P. Golle, M. Jakobsson, A. Juels, and P. Syverson, "Universal re-encryption for mixnets," in *Topics in Cryptology–(CT-RSA)*, T. Okamoto, Ed. Berlin, Germany: Springer, 2004, pp. 163–178.

[19] R. Goyal, S. Sharma, S. Bevinakoppa, and P. Watters, "Obfuscation of stuxnet and flame malware," in *Proc. Latest Trends Appl. Inform. Comput., 3rd Int. Conf. Appl. Inform. Comput. Theory (AICT)*, WSEAS, 2012.

[20] GreAT, "Gauss: Abnormal distribution," Kaspersky Lab Global Res. Anal. Team, Moscow, Russia, Tech. Rep., Aug. 2012. [Online]. Available: https://securelist.com/gauss-abnormal-distribution/36620/

[21] M. Jakobsson, "On quorum controlled asymmetric proxy re-encryption," in *Public Key Cryptography*. Berlin, Germany: Springer, 1999, pp. 112–121.

[22] D. J. K. Stevens. (Mar. 2010). *ZeuS Banking Trojan Report*. [Online]. Available: https://www.secureworks.com/research/zeus

[23] A. Moser, C. Kruegel, and E. Kirda, "Limits of static analysis for malware detection," in *Proc. 23rd Annu. Comput. Secur. Appl. Conf. (ACSAC)*, Dec. 2007, pp. 421–430.

[24] Mstfknn. (Nov. 2018). *Gand Crab*. [Online]. Available: https://github.com/mstfknn/malware-sample-library/blob/master/GandCrab/

[25] Odzhan. (Feb. 2019). *Odzhan Shake-128*. GitHub. [Online]. Available: https://github.com/odzhan/tinycrypt/tree/master/stream/shake128

[26] 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, pp. 1–48, Sep. 2019, doi: 10.1145/3329786.

[27] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Advances in Cryptology–(EUROCRYPT)*, J. Stern, Ed. Berlin, Germany: Springer, 1999, pp. 223–238.

[28] J. Riordan and B. Schneier, *Environmental Key Generation Towards Clueless Agents*. Berlin, Germany: Springer, 1998, pp. 15–24.

[29] J. Sen, "Homomorphic encryption: Theory & applications," 2013, *arXiv:1305.5886*. [Online]. Available: http://arxiv.org/abs/1305.5886

[30] N.I.N.I.S. Technology., "Sha-3 standard: Permutation-based hash and extendable-output functions: Fips pub 202," Scotts Valley, CA, USA: CreateSpace Independent Publishing Platform, 2015. [Online]. Available: https://books.google.com.pk/books?id=hCwatAEACAAJ

[31] L. Wang, K. Chen, Y. Long, and X. Mao, "A new RSA-based proxy re-encryption scheme," *J. Comput. Inf. Syst.*, vol. 11, pp. 567–575, Jan. 2015.

[32] Ytisf. *Binaries/Trojan.Kovter*. https://github.com/ytisf/theZoo/blob/master/malware/Binaries/Trojan.Kovter/

[33] Ytisf. *Linux Wirenet*. https://github.com/ytisf/theZoo/tree/master/malware/Binaries/Linux.Wirenet

[34] Ytisf. *Linux.Encoder.1*. https://github.com/ytisf/theZoo/tree/master/malware/Binaries/Linux.Encoder.1

[35] Ytisf. (May 2017). *Wannacry Ransomware*. [Online]. Available: https://github.com/ytisf/theZoo/tree/master/malware/Binaries/Ransomware.WannaCry

[36] Ytisf. (May 2017). *Wannacry Ransomware*. [Online]. Available: https://github.com/ytisf/theZoo/tree/master/malware/Binaries/Ransomware.WannaCry

**AHSAN RASHEED ABBASI** received the master's degree in information security, with specialization in cryptography and cryptanalysis, from the Department of Information Security, Military College of Signals, National University of Sciences and Technology, Islamabad, Pakistan. His research interests include cryptology, cryptanalysis, and malware analysis.

**MEHREEN AFZAL** received the degree in mathematics and the Ph.D. degree in information security from the National University of Sciences and Technology (NUST), Rawalpindi, Pakistan, in 1995 and 2010, respectively. She is currently associated with the Military College of Signals, NUST. Her contributions include research articles on cryptanalysis and design of cryptographic algorithms and protocols. Her research interests include information security and cryptology.

**WASEEM IQBAL** received the bachelor's degree in computer science from the Department of Computer Science, University of Peshawar, in 2008, and the master's degree in information security from the Military College of Signals, NUST, in 2012. He was inducted as a Lecturer at the Department of Information Security, NUST, in May 2012, where he was promoted to an Assistant Professor, in February 2015. He is currently an Academician, a Researcher, a Security Professional, and an Industry Consultant. He is currently enrolled in the Ph.D. Program and is in research phase. His professional services include but not limited to industry consultation, a Workshops Organizer/Resource Person, a Technical Program Committee Member, the Conference Chief Organizer, an Invited Speaker, and a reviewer for several international conferences. He has authored over 45 scientific research articles in prestigious international journals (ISI indexed) and conferences. He is the Principal Advisor for more than eight M.S. students and ten UG projects, out of ten, eight of his UG projects are industry funded projects. He has conducted more the 15 CEH, CHFI, CSCU, and forensics practical hands on workshops for industry and general public. He achieved the merit-based scholarship throughout his bachelor's degree. In recognition of his services, he received the Overall University Best Teacher Award for the year 2014 to 2015.

**SHYNAR MUSSIRALIYEVA** is currently an Associate Professor and the Head of the Department of Information Systems, Al-Farabi Kazakh National University, Almaty, Kazakhstan. Her research interests include information security and computational mathematics.

**FAWAD KHAN** received the B.S. degree in electrical engineering from UET, Peshawar, and the M.S. degree in electrical engineering from CECOS University, in 2010 and 2014, respectively, and the Ph.D. degree from the School of Cyber Engineering, Xidian University, in 2018. He is currently working with the National University of Sciences and Technology, Pakistan. His research interests include cryptography and information security. His professional services include: a Technical Program Committee Member and a Reviewer of several international journals and conferences, including IEEE ACCESS, the IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS, *EURASIP Journal on Information Security* (Springer), and *Neural Computing & Applications* (Springer).

**AWAIS UR REHMAN** received the master's degree in computer science from the University of South Asia, Lahore, Pakistan. His research interests include cybersecurity, vulnerability and exploits, and machine learning.