Malware encryption schemes – rerandomizable ciphertexts encrypted using environmental keys *

Herman Galteland† and Kristian Gjøsteen

Department of Mathematical Sciences,
NTNU – Norwegian University of Science and Technology, Trondheim
{herman.galteland, kristian.gjosteen}@ntnu.no

October 10, 2017

Abstract

Protecting malware using encryption prevents an analyst, defending some computer(s) in the network, from analyzing the malicious code and identifying the intentions of the malware author [4, 5]. We discuss malware encryption schemes that use environmental encryption keys [11], generated from some computer(s) the malware author intends to attack, and is able to rerandomize [2, 8] ciphertexts, to make each malware sample in the network indistinguishable.

We are interested in hiding the intentions and identity of the malware author, not in hiding the existence of malware.

Keywords. Malicious cryptography, environmental keys, rerandomization, provable security.

1 Introduction

Malware is software maliciously installed on a computer designed to give functionality and behavior desired by the *malware author*, but not by the legitimate computer owner.

Our goal is to study malware propagation and how to protect propagating malware from analysis. We will not study the construction of computer viruses or any other types of malware, but rather at how to construct schemes designed to encrypt malware such that we can hide the intentions and the identity of the malware author.

1.1 Real world examples

BurnEye [13] is a tool designed to defend binary files and is an example on how to protect malware. The tool adds three layers of protection to a file: obfuscation, encryption, and a fingerprint layer. The fingerprint layer ensures that the files can only be executed on a specific computer that has the specifications stated by the fingerprint layer. The encryption layer uses a user-chosen password as the encryption key such that the file can only be executed (or analyzed) by someone with the proper password.

Gauss [10] is an example of sophisticated malware that uses encryption to protect certain payloads. Gauss uses *environmental keys* to decrypt these payloads, where an

*This is the extended version of [7]

†This work is funded by Nasjonal sikkerhetsmyndighet (NSM), www.nsm.stat.no

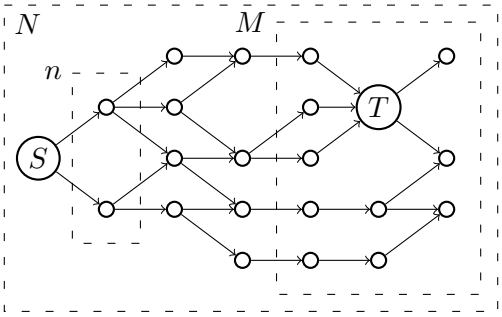

Figure 1: Example of malware propagating into an analyst's network. The source S infects n initial nodes. The analyst protects a network if M nodes. There is at most N malware samples in the universe.

environmental key is generated from locally available data. The malware gathers data on the infected computer and hashes it to create decryption keys, where the string of data that results in the correct key is selected by the malware author. The malicious code can only be executed when the correct key is produced, that is, when the malware infects the intended target. To our knowledge the contents of the encrypted payloads of Gauss are still unknown.

1.2 Malware propagation

Consider a malware author whose objective is to attack a specific location (or locations). The malware author's goals is to hide his intentions and identity. The malware author's opponent is an *analyst* observing and defending a network of nodes, which contains one (or more) of the malware author's target(s). The goal of the analyst is to detect malware targeting any part of the network he is protecting and to discover the intentions and the identity of the malware author. Hiding the mere existence of malware from the analyst is a distinct problem and not one we consider in the current work.

1.2.1 Setup

We use the following model to describe malware propagation, see Figure 1. The source S, the malware author, infects n initial nodes with (different variations of) his malware. Released malware infects subsequent nodes by making similar copies of themselves and propagates throughout the network.

The analyst protects M nodes in the network from any possible malware threat. We assume he has full knowledge of the environment he is protecting. By observing the wider network the analyst can find at most N malware samples.

Every direct link to the malware author increases the analyst's chance to discover the author's identity. So, to avoid identification, the malware author should perform as few initial infections as possible and use indirect paths to the target.

1.2.2 Encrypting the malware payload

The malware author encrypts the malware payload to increase the analyst's workload. Encrypting the payload prevents an analyst reverse engineering the malicious code [5] and hides the intentions of the malware author. We use encryption keys derived from environmental parameters, network triggers, or a combination of these [11]. The environmental

Malware consist of a cleartext loader and an encrypted payload. When malware arrives on a new host the loader is executed and preforms the following steps:

1. The loader scans the host environment and determines the environmental data.
2. The loader hashes the environmental data to produce one or more keys.
3. The loader tries to decrypt the encrypted payload with each key.
4. If the decryption succeeds, the decrypted payload is executed.
5. The malware may also attempt to infect some other host, in which case the encrypted payload is rerandomized before it is transmitted to the new host.

Note that the malware author will certainly use some polymorphic engine and other standard malware techniques in order to provide a basic level of protection for the cleartext loader and the encrypted payload.

Figure 2: The malware attack process.

information is gathered from the target node and could consist of, for example, IP address, PATH variables, and/or any arbitrary time and date. This would either requires information about the target node, be guessed, or be general information (to target several nodes).

An encrypted malware consist of the *encrypted payload*, containing the malicious code, and a cleartext *loader* that gathers environmental parameters to generate decryption keys.

To initialize an encrypted malware the author chooses environmental data identifying the target node(s), hashes the data to create an encryption key, encrypts the payload using the key, and releases the encrypted malware. When malware infects a new node the cleartext loader determines the environmental data of the infected node, hashes the data to derive $K \geq 1$ keys, and attempts to decrypt the payload using the K derived keys. If the decryption is a success can the malicious code be executed. Otherwise the loader creates copies of the malware to infect subsequent nodes.

The malware author can initialize at most n distinct encrypted malware, one for each initial infection, which encrypts the same malicious code. Hence, there is at most n distinct encrypted payloads among the samples collected by the analyst. Each sample is encrypted and has an unknown target. If the analyst wants to guarantee that none of these n samples would attack a node in his network then he needs to do roughly K trial decryptions for each of his M nodes. Hence, the analyst's workload is at most nMK.

1.2.3 Rerandomize the encrypted payload

Instead of making exact copies of the malware we want the loader to *rerandomize* [2,8] the encrypted payload using techniques from asymmetric cryptography. The rerandomization process takes as input an encrypted payload and some uniformly random values to produce a new ciphertext that encrypts the same malicious code. Hence, the loader can produce several different-looking encrypted malware to infect subsequent nodes without knowledge of the secret key. The process is described in Figure 2.

We want the rerandomize process to produce the encrypted payloads such that any two malware samples are indistinguishable. If the analyst is unable to distinguish between malware samples then, essentially, there are N unique variations of the malware in the network. This means that the analyst need to do K trial decryptions for N samples for M different nodes to ensure that none of the malware samples are targeting any of his nodes. This will increase the analyst's workload to NMK.

Since the malware creates different variations of itself the malware author can choose n to be small and, possibly, significantly reducing the risk of unveiling his identity.

Path variation malware consist of a cleartext loader and an encrypted payload. When path variation malware arrives on a new host the loader is executed and preforms the following steps:

1. The loader scans the host environment and determines the environmental data.
2. The loader hashes the environmental data to produce one or more keys and a default key.
3. The loader check each key if they can decrypt the encrypted payload.
   - If a key is found use it to decrypt, if not use the default key to decrypt.
4. The loader checks if the decrypted payload could be executed.
5. The malware may also attempt to infect some other host, in which case the encrypted payload is rerandomized before it is transmitted to the new host.

Note that the malware author will certainly use some polymorphic engine and other standard malware techniques in order to provide a basic level of protection for the cleartext loader and the encrypted payload.

Figure 3: Path variation malware attack process.

1.2.4 Path variation

Instead of using a single encryption key we can choose to use several keys, derived from different nodes in the network, describing a path towards the target, see Figure 4. The last key is derived from environmental data identifying the target node, just as before, and the remaining keys, called *default keys*, are derived from environmental data that is available on all nodes in the path. This requires the malware author to investigate and gather the environmental data of each node in the path towards the target before malware can be encrypted. The difference between the path and the single key variations is that the path scheme will *always* try to decrypt the payload using a key (a default key in most cases).

The path variation can only be decrypted if malware infect the nodes in the correct path toward the target. There is no difference between malware samples in the correct path and malware samples in a wrong path, hence, all samples needs to be treated as if they are both in the path and not in the path by the opponent. The only difference is when the malicious code has been executed and the opponent notices.

If the opponent only wants to know if a node in his network is targeted he has to check all possible paths between a node with a sample to all other nodes and see if the sample decrypt correctly. If the opponent finds the target node, either by analysis or by noticing that the malicious code was executed, then he can trace the correct path back towards the source, by using the algorithms in the malware, and possibly find information that could identify the malware author. However, this requires knowledge of the full network[1], where the opponent only has knowledge over his network. This is an unwanted trait, but it seems to be unavoidable if we also wish to include the rerandomization. Without knowledge of the target node the opponent cannot trace back to the source by looking at the algorithms alone.

1.2.5 Limitations

The limitations of our schemes is that the analyst can always guess, or predict, the target of the malware author. Also, if the malware reaches its target, the payload will be decrypted and executed. If the analyst notices the attack he will be able to deduce the environmental key and thus be able to decrypt the payload. This seems impossible to avoid.

Once an analyst discovers the key used for one sample, he can easily discover all other samples corresponding to that key. However, the malware author will hope that different analysts are unwilling to reveal that they are under attack, they somehow consider this

[1] Unless the source is inside the analyst's network

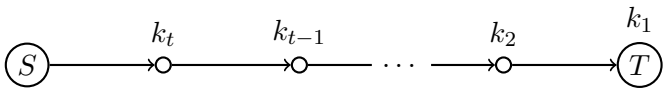

Figure 4: Location of keys on the path towards the target

fact sensitive, and that they therefore do not share discovered keys. This means that one analyst's success may not make all the other analyst's work easier.

For the path variant, if the analyst discover the target node then it is possible to find the source, as discussed previously. If the opponent finds the correct path back to the source then he can discover all the encryption keys. However, this is not enough to determine if another malware sample is encrypted under the same keys, especially if that sample has infected a node not in the path. The analyst has to find the sample's infection path back to the source first, remove any decryption keys used (to restore the sample to its initial condition), and then check if the keys decrypt the payload. This would require knowledge of the full network.

1.2.6 Potential threat

Assuming there is more than one malware author, an analyst can not be certain of whether every new encrypted malware sample corresponds to one he has previously determined is no threat or a genuinely new piece of malware. That is, all malware samples created by different malware authors looks like the same encrypted malware. However, this requires all malware authors to agree on a malware encryption standard; use the same size, use the same loader, and otherwise look the same. This seems unlikely. If malware authors do not agree on a standard then the analysts can use these pieces of information to classify samples.

1.3 Related work

Traditionally cryptography has been developed and used as a defense against attackers. However, it is clear that cryptography can also be of use to the attackers.

Young and Yung where the first to raise the concern about malicious use of cryptography (cryptovirology) [15] and have several works related to malware construction and propagation. First, Young and Yung designed a virus capable of encrypting files on the victim's computer and hold them for ransom [14]. Second, they describe how to utilize a mix network to mix programs and propagate malware [15]. Third, they designed a mobile program that carries a rerandomizable ciphertext, which enables anonymous communication, where the program takes random walks through a network and rerandomize the ciphertext at each node, using system called Feralcore [16].

The mix network and the mobile program, by Young and Yung, use the idea of universal re-encryption, by Golle et al [8], to re-encrypt ciphertexts. The re-encryption process transforms the ciphertexts into a new ciphertext that encrypts the same message and do not require knowledge about the public key. Similar to universal re-encryption is the notion of rerandomization by Canetti et al. [2].

Filiol showed that by encrypting malware payload [4, 5] one can prevent anyone from analyzing the code and reverse engineer it, possibly using the environmental keys of Riordan and Schneier [11] as encryption keys. Similar to Riordan and Schneier's environmental keys, secure triggers [6, 9] are also used to keep certain content private until a particular event occurs.

1.4 Overview

The rest of this paper contains the technical details of our schemes. The general cryptosystem designed to encrypt and rerandomize malware payload is described in Section 2.1.

The basic scheme, in Section 2.2, shows that malware encryption described in the introduction is possible in theory, however, the scheme is not practical because it requires short messages.

The extended scheme, in Section 2.4, is based on the basic scheme and can encrypt longer messages, making it more practical. The basic and extended schemes use the malware attack process described in Figure 2.

The path scheme, in Section 2.6, is the path variant of both the basic and the extended scheme and uses several encryption keys instead of one. The malware attack process for the path variation is described in Figure 3.

For each scheme we show that they are secure using games, where the opponent is asked to distinguish between ciphertexts encrypting the same message and ciphertexts encrypting two different messages. That is, we will simulate whether an analyst is able to distinguish malware samples. The security proof of the basic scheme is in Section 2.3, extended scheme in Section 2.5, and path scheme in Section 2.7. All three proofs are similar.

2 Rerandomizable malware encryption schemes

In this section we present three encryption schemes designed to encrypt and rerandomize malware payload. The first scheme is a basic proof of concept and the second is an extension of the basic scheme capable of encrypting larger payloads. The third scheme is the path variant of the first two. Further, we show that it is hard to distinguish between encrypted payload samples by using games.

As a simplification we denote payload as messages, encrypted payload as ciphertexts, replication of malware as rerandomization of ciphertexts, and environmentally derived keys as keys.

2.1 Preliminary

In each scheme we have an algorithm $\mathcal{E}$ encrypting messages, an algorithm $\mathcal{D}$ decrypting ciphertexts, and an algorithm $\mathcal{R}$ rerandomizing ciphertexts. In the path variant of the extended scheme (in Section 2.6) we add a padding functionality to the rerandomize algorithm and rename it to a padding algorithm $\mathcal{P}$.

Encryption For a message m and a key k the encryption algorithm $\mathcal{E}(k, m)$ outputs a ciphertext c.

Decryption For a ciphertext c and a key k the decryption algorithm $\mathcal{D}(k, c)$ either outputs a message m or a special symbol $\perp$ indicating decryption failure.

Rerandomization For a ciphertext c, encrypting a message m, the rerandomize algorithm $\mathcal{R}(c)$ outputs a ciphertext c' encrypting the same message m.

The output distribution of the rerandomize algorithm should be computationally indistinguishable from the output distribution of the encryption algorithm. That is, it should be hard to determine if two different ciphertexts encrypts the same message or not.

The systems should be correct, we should almost always be able to decrypt all ciphertexts output by the encryption algorithm. Since the output distribution of the encryption and rerandomize algorithms are computationally indistinguishable, ciphertexts output by the rerandomize algorithm will also almost always be correct.

Correctness If c was output from $\mathcal{E}(k, m)$ then $\mathcal{D}(k, c)$ will always output m except with negligible probability.

Rerandomization If c was output by $\mathcal{E}(k, m)$ then the output distribution of $\mathcal{R}(c)$ should be computationally indistinguishable from the output distribution of $\mathcal{E}(k, m)$.

We will not always be able to apply an arbitrary number of rerandomizations to a ciphertext without getting decryption errors, which we will see is the case in Section 2.4 and in Section 2.6.

The security requirements of our cryptosystems reflects the intentions of the malware author. It should be difficult to guess the malware author's target, and it should be hard to determine if two ciphertexts are the encryption of the same message or not.

Key indistinguishability It should be hard to say something about which key a ciphertext has been encrypted under.

Indistinguishability It should be hard to decide if two ciphertexts, encrypted under the same key, decrypts to the same message or not.

2.2 Basic scheme

The basic scheme is based on the ElGamal cryptosystem over a group G of prime order p generated by g. This scheme is essentially the same as the encryption scheme proposed by Golle et al [8].

Encryption For a message $m \in G$ and a key $k \in \mathbb{Z}_p^*$, let $r, s \xleftarrow{r} \mathbb{Z}_p^*$ and output

$$c = (x, y, z, w) = (g^r, g^{kr}, g^s, g^{ks}m).$$

Decryption For a ciphertext $c = (x, y, z, w)$ and a key $k \in \mathbb{Z}_p^*$ check if $x^k = y$. If it is then output

$$m = z^{-k}w.$$

Otherwise output $\perp$.

Rerandomize For a ciphertext $c = (x, y, z, w)$, let $r', s' \xleftarrow{r} \mathbb{Z}_p^*$ and output

$$c' = (x', y', z', w') = (x^{r'}, y^{r'}, zx^{s'}, wy^{s'}).$$

Correctness If $c = (x, y, z, w)$ was output by the encryption algorithm then there exists parameters r, s, k, and a message m such that

$$c = (x, y, z, w) = (g^r, g^{kr}, g^s, g^{ks}m).$$

With input c the rerandomize algorithm will output $c' = (x', y', z', w')$ where

$$\begin{aligned}
x' &= x^{r'} = g^{rr'}, \\
y' &= y^{r'} = g^{krr'}, \\
z' &= zx^{s'} = g^s g^{rs'} = g^{s+rs'}, \\
w' &= wy^{s'} = g^{ks} g^{krs'} m = g^{k(s+rs')} m.
\end{aligned}$$

That is, $c' = (g^{rr'}, g^{krr'}, g^{s+rs'}, g^{k(s+rs')}m)$. Since $r \neq 0$, we get that $s + rs'$ can take any value modulo p except s and all values are equally probable. Hence, the output distribution of the encryption and rerandomize algorithms are computationally indistinguishable and has the same structure, that is,

$$c' = (g^{\hat{r}}, g^{\hat{k}\hat{r}}, g^{\hat{s}}, g^{\hat{k}\hat{s}}m)$$

for some parameters $\hat{r}$, $\hat{s}$, $\hat{k}$, and a message m.

For a ciphertext $c = (x, y, z, w)$ and the correct key k we have that $x^k = (g^r)^k = g^{kr} = y$, which is true for ciphertexts output by both the encryption and rerandomize algorithms. The message m is retrieved by computing

$$z^{-k}w = (g^s)^{-k}g^{ks}m = g^{-ks+ks}m = m.$$

In other words, the decryption algorithm is correct.

Longer messages The limitation of the basic scheme is that the message size is relatively small. One option is to encrypt several messages under the same key. That is, a set of messages $\{m_1, m_2, \ldots, m_n\}$ can be encrypted as

$$(g^r, g^{kr}, g^{s_1}, g^{ks_1}m_1, g^{s_2}, g^{ks_2}m_2, \ldots, g^{s_n}, g^{ks_n}m_n)$$

for some parameters $s_1, s_2, \ldots, s_n$, r, and key k. However, this is not an efficient method. In Section 2.4 we construct the extended scheme where we use techniques from symmetric cryptography to encrypt longer messages more efficiently.

2.3 Security of the basic scheme

The decryption key is derived from environmental data sampled by the loader from the infected computer. From the opponent's perspective the collection of sampled data types can be considered as a probability space of possible decryption keys. We will denote this space by D. If the size of D is large then the opponent is less likely to guess the correct decryption key, where the size of D is determined by the number of different data types and combinations of the gathered data.

We show that the opponent $\mathcal{O}$ is unable to distinguish between ciphertexts and that his advantage is determined by D, that is, the probability of the opponent guessing the correct key. To prove this we use games [12]. In our games we start by simulating Experiment 1 where we ask the opponent to differentiate between the two cases; two ciphertexts encrypting different messages, and two ciphertexts encrypting the same message.

Experiment 1 Given two ciphertext c_1 and c_2, decide if

$$\begin{array}{ll} c_1 = \mathcal{E}(k_1, m_1) \\ c_2 = \mathcal{E}(k_2, m_2) \end{array} \quad \text{or} \quad \begin{array}{ll} c_1 = \mathcal{E}(k_1, m_1) \\ c_2 = \mathcal{R}(c_1) \end{array}$$

for some messages m_1, m_2 and keys k_1, k_2.

We show that the security of the scheme is based on the hardness of the Decisional Diffie-Hellman (DDH) problem [1] in the random oracle model. The DDH problem is to distinguish tuples of the form (g, g^a, g^b, g^{ab}) and tuples of the form (g, g^a, g^b, g^c), for some $a, b, c \in \mathbb{Z}_p^*$. Where the DDH assumption states that the DDH problem is hard to solve.

To create the encryption keys we use an oracle to hash elements drawn from the probability space D. We denote the oracle by H, where it should be impossible to get any information about the input of the oracle by looking at its output.

Algorithm 1 Game 0 simulating Experiment 1

$u_1, u_2 \xleftarrow{r} D$, $k_1 \leftarrow H(u_1)$, $k_2 \leftarrow H(u_2)$, $b \xleftarrow{r} \{0,1\}$
Get m_1, m_2 from $\mathcal{O}$
if $b = 0$ **then**
    $c_1 \leftarrow \mathcal{E}(k_1, m_1)$
    $c_2 \leftarrow \mathcal{E}(k_2, m_2)$
    Send c_1, c_2 to $\mathcal{O}$
if $b = 1$ **then**
    $c_1 \leftarrow \mathcal{E}(k_1, m_1)$
    $c_2 \leftarrow \mathcal{R}(c_1)$
    Send c_1, c_2 to $\mathcal{O}$
Get b' from $\mathcal{O}$

Game 0 Simulate Experiment 1. See Algorithm 1 for the detailed procedure. Let E_0 be the event that $b = b'$ in Game 0.

Game 1 Stop the game if the opponent queries either u_1 or u_2 (guessed the correct key). H outputs $b' \xleftarrow{r} \{0,1\}$. We denote this event by F_1.

Let E_1 be the event that $b = b'$ in Game 1. Unless event F_1 occurs Game 1 behaves just like Game 0. Thus $E_0 \wedge \neg F_1 \iff E_1 \wedge \neg F_1$ and by the difference lemma we have

$$|\Pr[E_0] - \Pr[E_1]| \leq \Pr[F_1].$$

Game 2 Draw $k_1, k_2 \xleftarrow{r} \mathbb{Z}_p^*$ and stop querying the oracle. (The opponent can still query the oracle, hence we need to draw samples from D to check if the opponent is guessing the correct key(s).)

Let E_2 be the event that $b = b'$ in Game 2. The output of H is indistinguishable from uniform samples of $\mathbb{Z}_p^*$, hence, $\Pr[E_1] = \Pr[E_2]$.

Game 3 For uniform $s, s' \xleftarrow{r} \mathbb{Z}_p^*$ and keys k_1, k_2 precompute

$$(x, y, z, w) = (g, g^{k_1}, g^s, g^{k_1 s}), \quad \text{and} \quad (x', y', z', w') = (g, g^{k_2}, g^{s'}, g^{k_2 s'})$$

before receiving message m_1 and m_2.

Let E_3 be the event that $b = b'$ in Game 3. After encrypting both messages, or encrypting one and rerandomizing it, we get that the output of the encryption, and rerandomize, algorithms are exactly the same in Game 2 and Game 3. Thus, $\Pr[E_2] = \Pr[E_3]$.

Game 4 Let $(x, y, z, w) = (g, g^{k_1}, g^s, g^{k_1 s})$ be the first tuple and

$$(x', y', z', w') = (x, x^a y^c, z x^b, w^c z^a y^{cb} x^{ab}) = (g, g^{a+ck_1}, g^{b+s}, g^{(a+ck_1)(b+s)})$$

be the second, for some $a, b, c \xleftarrow{r} \mathbb{Z}_p^*$.

Let E_4 be the event that $b = b'$ in Game 4. Since the tuples of Game 4 results in the same output space as the tuples of Game 3 we get that $\Pr[E_3] = \Pr[E_4]$.

Game 5 The output of the rerandomize algorithm is on the form

$$(g^{rr'}, g^{k_1 rr'}, g^{s+rs'}, g^{k_1(s+rs')} m)$$

for some r, r', s and s', where $s + rs' \neq s$ since none of the variables used in the algorithm can be zero. This gives us a statistical difference of $1/p$ between the output distributions.

Algorithm 2 Input: (x, y, z, w)

$u_1, u_2 \xleftarrow{r} D$, $b \xleftarrow{r} \{0,1\}$
$a, b, c \xleftarrow{r} \mathbb{Z}_p^*$
$(x', y', z', w') = (x, x^a y^c, z x^b, w^c z^a y^{cb} x^{ab})$
Get m_1, m_2 from $\mathcal{O}$
if $b = 0$ **then**
    $r, r' \xleftarrow{r} \mathbb{Z}_p^*$
    $c_1 \leftarrow (x^r, y^r, z, w m_1)$
    $c_2 \leftarrow (x'^{r'}, y'^{r'}, z', w' m_2)$
    Send c_1, c_2 to $\mathcal{O}$
if $b = 1$ **then**
    $r, r', s', \tilde{s} \xleftarrow{r} \mathbb{Z}_p^*$
    $c_1 \leftarrow (x^r, y^r, z, w m_1)$
    $c_2 \leftarrow (x^{rr'}, y^{rr'}, z x^{rs'+\tilde{s}}, w y^{rs'+\tilde{s}} m_1)$
    Send c_1, c_2 to $\mathcal{O}$
Get b' from $\mathcal{O}$

Change the rerandomization algorithm such that the second ciphertext (in case $b = 1$) is computed as

$$(g^{rr'}, g^{k_1 rr'}, g^{s+rs'+\tilde{s}}, g^{k_1(s+rs'+\tilde{s})} m_1),$$

where $\tilde{s} \xleftarrow{r} \mathbb{Z}_p^*$. The new sum $s + rs' + \tilde{s}$ can be any value in $\mathbb{Z}_p^*$ and all values are equally probable.

Let F_5 be the event that $s + rs' + \tilde{s} = s$ and let E_5 be the event that $b = b'$ in Game 5. Unless F_5 occurs, Game 4 and Game 5 behaves the same, that is, $E_4 \wedge \neg F_5 \iff E_5 \wedge \neg F_5$ and by the difference lemma we get that

$$|\Pr[E_4] - \Pr[E_5]| \leq \Pr[F_5] = \frac{1}{p}.$$

Game 6 Change the first tuple into the form $(g, g^{a'}, g^{b'}, g^{c'})$, for uniform elements $a', b', c' \in \mathbb{Z}_p^*$. The second tuple will then look like

$$(g, g^{a+a'c}, g^{b+b'}, g^{ab+ab'+a'bc+cc'}).$$

Let E_6 be the event that $b = b'$ in Game 6. Since we are using uniform elements in the tuples the encryption and rerandomization algorithms are, essentially, one-time pads. Hence, $\Pr[E_6] = 1/2$.

We claim that $|\Pr[E_5] - \Pr[E_6]| = \mathrm{Adv}_{\mathrm{ddh}}^{\text{ind-cpa}}$, where we will use Algorithm 2. The input of the algorithm is a tuple (g, g^a, g^b, g^c), for some a, b, and c, where c can be equal to ab. The algorithm simulates Game 5 if the input is on the form (g, g^a, g^b, g^{ab}) and

$$\Pr[A_2(g, g^a, g^b, g^{ab}) = 1 \mid a, b \xleftarrow{r} \mathbb{Z}_p^*] = \Pr[E_5].$$

If the input is on the form (g, g^a, g^b, g^c) the algorithm proceed as in Game 6 and

$$\Pr[A_2(g, g^a, g^b, g^c) = 1 \mid a, b, c \xleftarrow{r} \mathbb{Z}_p^*] = \Pr[E_6],$$

where the DDH-advantage of Algorithm 2 is equal to $|\Pr[E_5] - \Pr[E_6]|$.

Summary From the games we bound the advantage of the opponent $\mathcal{O}$.

$$\begin{aligned}
\mathrm{Adv}(\mathcal{O}) &= |\Pr[E_0] - 1/2| \\
&= |\Pr[E_0] - \Pr[E_1] + \Pr[E_1] - \Pr[E_2] + \Pr[E_2] - \Pr[E_3] + \Pr[E_3] \\
&\qquad - \Pr[E_4] + \Pr[E_4] - \Pr[E_5] + \Pr[E_5] - \Pr[E_6] + \Pr[E_6] - 1/2| \\
&\leq |\Pr[E_0] - \Pr[E_1]| + |\Pr[E_4] - \Pr[E_5]| + |\Pr[E_5] - \Pr[E_6]| \\
&\leq \Pr[F_1] + \frac{1}{p} + \mathrm{Adv}^{\text{ind-cpa}}_{\text{ddh}}(\mathcal{O}).
\end{aligned}$$

By the DDH assumption the DDH advantage is negligible and, for large enough p, we get that the advantage of our opponent is determined by the probability that $\mathcal{O}$ guesses or predicts the correct key, that is, determined by the probability space D.

2.4 Extended scheme

In the extended scheme we represent messages as bit strings to allow longer messages to be encrypted. This change reduces the number of rerandomizations we can perform on a ciphertext and we need to relax the requirements of the cryptosystem.

Correctness If c was produced by iteratively applying $\mathcal{R}$ to the output of $\mathcal{E}(k, m)$ at most n times, then $\mathcal{D}(k, c)$ will never output $\perp$ and output m except with negligible probability.

We use a pseudorandom function $f : G \to \{0,1\}^N$ mapping group elements to bit strings of length N, for some large $N \in \mathbb{N}$. We let f_L denote the truncation of the output to L bits, for $L < N$. We assume that group elements can be encoded as bit strings of length at most $l/2$. The construction in this section is very similar to the hybrid scheme by Golle et al [8].

Encryption For a message $m \in \{0,1\}^L$ and a key $k \in \mathbb{Z}_p^*$, let $r, s \xleftarrow{r} \mathbb{Z}_p^*$, $\gamma \xleftarrow{r} G$ and output

$$c = g^r || g^{kr} || g^s || g^{ks}\gamma || \left(f_{L+l(n+1)+1}(\gamma) \oplus (m||1||0^{l(n+1)}) \right).$$

Decryption For a ciphertext $c = x||y||b_0'$ and a key $k \in \mathbb{Z}_p^*$ check if $x^k = y$. If not output $\perp$. If it is let $b_0' = z_0||w_0||b_0$ and compute

$$b_1' = f_{|b_0|}(z_0^{-k} w_0) \oplus b_0.$$

If the result b_1' ends in $l' \geq l$ zeros then the message is the result minus the tail of zeros and exactly one 1. Otherwise interpret b_1' as $z_1||w_1||b_1$ and repeat the procedure. If this procedure is repeated $n+1$ times output $\perp$.

Rerandomization For a ciphertext $c = x||y||b_m||b_l$, where b_l is the last l bits. Let $r', s' \xleftarrow{r} \mathbb{Z}_p^*$, $\gamma' \xleftarrow{r} G$, and output

$$c' = x^{r'} || y^{r'} || x^{s'} || y^{s'} \gamma' || \left(f_{|b_m|}(\gamma') \oplus b_m \right).$$

Correctness Before applying the rerandomize algorithm, b_m looks like

$$g^s||g^{ks}\gamma|| \left(f_{L+ln+1}(\gamma) \oplus (m||1||0^{ln})\right)$$

for $s \in \mathbb{Z}_p^*$, key k, and $\gamma \in G$. The l last bits we discard, b_l, is an "encryption" of l zeros. We can therefore only perform n rerandomizations on a ciphertext before we get decryption failure, that is, there are no tail of zeros left for the decryption algorithm to detect.

If $c = x||y||b_0'$ was output from the encryption algorithm, we have that $x^k = g^{kr} = y$. Hence, we can write b_0' as $z||w||b_0$, and compute

$$\begin{aligned} f_{|b_0|}(z^{-k}w) \oplus b_0 &= f_{L+l(n+1)+1}(g^{-ks}g^{ks}\gamma) \oplus f_{L+l(n+1)+1}(\gamma) \oplus (m||1||0^{l(n+1)}) \\ &= (m||1||0^{l(n+1)}). \end{aligned}$$

The result ends with a tail of $l' \geq l$ zeros and the output message is m.

Let c be a ciphertext that was produced by iteratively applying the rerandomize algorithm to the output of $\mathcal{E}(k, m)$ t times, where $1 \leq t \leq n$. Write c as $x||y||b_t'$, where $x = g^{r_1 \cdots r_{t+1}}$, $y = g^{k(r_1 \cdots r_{t+1})}$, and b_t' looks like

$$(g^{r_1 \cdots r_t})^{s'}||(g^{k(r_1 \cdots r_t)})^{s'}\gamma_t|| \left(f_{L+l(n+1-t)+1}(\gamma_t) \oplus b_{t-1}'\right)$$

for $s', r_1, \ldots, r_{t+1} \in \mathbb{Z}_p^*$, key k, and group element $\gamma_t \in G$. Observe that for all $1 \leq t \leq n$ we have that $x^k = y$. Thus we can write $b_t' = z_t||w_t||b_t$ and compute

$$\begin{aligned} f_{|b_t|}(z_t^{-k}w_t) \oplus b_t &= f_{L+l(n+1-t)+1}(g^{-ks'(r_1 \cdots r_t)}g^{ks'(r_1 \cdots r_t)}\gamma_t) \oplus f_{L+l(n+1-t)+1}(\gamma_t) \oplus b_{t-1}' \\ &= b_{t-1}' \end{aligned}$$

where b_{t-1}' does not end with a tail of $l' \geq l$ zeros (except with negligible probability), since the ciphertext is also encrypted once using with the encryption algorithm (in addition to the t rerandomizations). Let $b_{t-1}' = z_{t-1}||w_{t-1}||b_{t-1}$ and repeat the process t more times. In the last iteration we perform the decryption on the bit string $z_0||w_0||b_0$, where b_0 looks like

$$f_{L+l(n+1-t)+1}(\gamma_0) \oplus (m||1||0^{l(n+1-t)}).$$

We know this decrypts to the message m and the decryption algorithm is correct.

2.5 Security of the extended scheme

Similar to the security proof of the basic scheme, we show that the opponent is unable to distinguish between encrypted ciphertexts and that his advantage is determined by D the probability of guessing the correct key. As in the proof of the basic scheme, we use games to simulate Experiment 1.

Game 0 Simulate Experiment 1. The full procedure can be seen in Algorithm 1. Let E_0 be the event that $b = b'$ in Game 0.

Game 1 Similar to the basic Game 1, where we get $|\Pr[E_0] - \Pr[E_1]| \leq \Pr[F_1]$.

Game 2 Similar to the basic Game 2, where we get $\Pr[E_1] = \Pr[E_2]$.

Game 3 Similar to the basic Game 3, where we get $\Pr[E_2] = \Pr[E_3]$.

Algorithm 3 Input: (x, y, z, w)

$u_1, u_2 \xleftarrow{r} D$, $b \xleftarrow{r} \{0,1\}$
$a, b, c \xleftarrow{r} \mathbb{Z}_p^*$
$(x', y', z', w') = (x, x^a y^c, z x^b, w^c z^a y^{cb} x^{ab})$
Get m_1, m_2 from $\mathcal{O}$
if $b = 0$ **then**
    $r, r'\gamma, \gamma' \xleftarrow{r} \mathbb{Z}_p^*$
    $c_1 \leftarrow x^r || y^r || z || w\gamma || \left(f_{L+l(n+1)+1}(\gamma) \oplus (m_1 || 1 || 0^{l(n+1)})\right)$
    $c_2 \leftarrow x'^{r'} || y'^{r'} || z' || w'\gamma' || \left(f_{L+l(n+1)+1}(\gamma') \oplus (m_2 || 1 || 0^{l(n+1)})\right)$
    Send c_1, c_2 to $\mathcal{O}$
if $b = 1$ **then**
    $r, r's', \gamma, \gamma' \xleftarrow{r} \mathbb{Z}_p^*$
    $c_1 \leftarrow x^r || y^r || z || w\gamma || \left(f_{L+l(n+1)+1}(\gamma) \oplus (m_1 || 1 || 0^{l(n+1)})\right)$
    Let $c_1 = x^r || y^r || b_m || b_l$, where b_l is the last l bits
    $c_2 \leftarrow x^{rr'} || y^{rr'} || x^{rs'} || y^{rs'}\gamma' || \left(f_{|b_m|}(\gamma') \oplus b_m\right)$
    Send c_1, c_2 to $\mathcal{O}$
Get b' from $\mathcal{O}$

Game 4 Similar to the basic Game 4, where we get $\Pr[E_3] = \Pr[E_4]$.

Game 5 Similar to the basic Game 6, except we use Algorithm 3 instead. We get that $|\Pr[E_4] - \Pr[E_5]| = \mathrm{Adv}_{\mathrm{ddh}}^{\text{ind-cpa}}$.

Game 6 Sample a function h from a family Γ of all functions from G to $\{0,1\}^N$ instead of using the function f. We denote h_L as the truncation of the output of h to L bits. The pseudorandom function (PRF) advantage is defined to the opponent's ability to distinguishing f from any function h sampled from Γ. The PRF-advantage is negligible if f is pseudorandom.

Let E_6 be the event $b = b'$ in Game 6. We use an arbitrary function h, with a random group element γ, to encrypt the message m, hence, the output ciphertexts of the encryption and rerandomization algorithms can be any random bit string. Thus $\Pr[E_6] = 1/2$.

We claim that $|\Pr[E_5] - \Pr[E_6]|$ is equal to the PRF-advantage, where we use Algorithm 4. The algorithm draws a function h from the family Γ, which may be equal to f. The PRF-advantage is

$$\left|\Pr[A_4(x, y, z, w) = 1 \mid f \leftarrow \Gamma] - \Pr[A_4(x, y, z, w) = 1 \mid h \leftarrow \Gamma]\right|$$

which is equal to $|\Pr[E_5] - \Pr[E_6]|$.

Summary From the games we bound the advantage of the opponent $\mathcal{O}$.

$$\begin{aligned}
\mathrm{Adv}(\mathcal{O}) &= |\Pr[E_0] - 1/2| \\
&= |\Pr[E_0] - \Pr[E_1] + \Pr[E_1] - \Pr[E_2] + \Pr[E_2] - \Pr[E_3] + \Pr[E_3] \\
&\qquad - \Pr[E_4] + \Pr[E_4] - \Pr[E_5] + \Pr[E_5] - \Pr[E_6] + \Pr[E_6] - 1/2| \\
&\leq |\Pr[E_0] - \Pr[E_1]| + |\Pr[E_4] - \Pr[E_5]| + |\Pr[E_5] - \Pr[E_6]| \\
&\leq \Pr[F_1] + \mathrm{Adv}_{\mathrm{ddh}}^{\text{ind-cpa}}(\mathcal{O}) + \mathrm{Adv}_{\mathrm{prf}}(\mathcal{O}).
\end{aligned}$$

Assuming f is pseudorandom the PRF advantage is negligible and by the DDH assumption the DDH-advantage is negligible. Therefore, the advantage of the opponent is determined by the probability that the opponent guesses or predicts the correct key, that is, determined by the probability space D.

Algorithm 4 Input: (x, y, z, w)

$u_1, u_2 \xleftarrow{r} \mathcal{D}$, $b \xleftarrow{r} \{0,1\}$, $h \leftarrow \Gamma$
$a, b, c \xleftarrow{r} \mathbb{Z}_p^*$
$(x', y', z', w') = (x, x^a y^c, z x^b, w^c z^a y^{cb} x^{ab})$
Get m_1, m_2 from $\mathcal{O}$
if $b = 0$ **then**
    $r, r', \gamma, \gamma' \xleftarrow{r} \mathbb{Z}_p^*$
    $c_1 \leftarrow x^r || y^r || z || w\gamma || \left(h_{L+l(n+1)+1}(\gamma) \oplus (m_1 || 1 || 0^{l(n+1)})\right)$
    $c_2 \leftarrow x'^{r'} || y'^{r'} || z' || w'\gamma' || \left(h_{L+l(n+1)+1}(\gamma') \oplus (m_2 || 1 || 0^{l(n+1)})\right)$
    Send c_1, c_2 to $\mathcal{O}$
if $b = 1$ **then**
    $r, r', s', \gamma, \gamma' \xleftarrow{r} \mathbb{Z}_p^*$
    $c_1 \leftarrow x^r || y^r || z || w\gamma || \left(h_{L+l(n+1)+1}(\gamma) \oplus (m_1 || 1 || 0^{l(n+1)})\right)$
    Let $c_1 = x^r || y^r || b_m || b_l$, where b_l is the last l bits
    $c_2 \leftarrow x^{rr'} || y^{rr'} || x^{rs'} || y^{rs'} \gamma' || \left(h_{|b_m|}(\gamma') \oplus b_m\right)$
    Send c_1, c_2 to $\mathcal{O}$
Get b' from $\mathcal{O}$

2.6 Path scheme

The path variation scheme encrypts malware under several keys and encrypted malware will be correctly decrypted if it travels on the correct path in the network toward the target. If a malware sample infects a node not in the path then it can no longer be correctly decrypt, except with negligible probability.

The scheme uses several encryption keys, where each key is generated from environmental data gathered from a node in the path. The malware author selects the path and, hence, needs some environmental information about each nodes in the path to generate the encryption keys.

When encrypted malware infects a node the loader samples local environmental data to generate a set of keys and a *default key*, checks if one of the non-default keys can be used for decryption, if not then it will use the default key in the decryption algorithm. If the node is in the correct path a key will be removed from the encrypted malware. If it is not in the correct path then the attempted decryption introduces a new random value to the ciphertext, which will not be removed on any subsequent node except for a negligible probability.

Malware payload will be encrypted using a set of keys $k_t, \ldots, k_2, k_1$, where $k_t, \ldots, k_2$ are default keys associated to a node in the path and k_1 is a specific key associated to the target node, see Figure 4.

2.6.1 Based on the basic scheme

Creating a path variant of the basic scheme is straight forward. We use the same algorithms except for the following. The encryption algorithm encrypts the malware under the sum of the keys, instead of a single key, and the decryption algorithm will always try to decrypt using a key. The algorithms are as follows.

Encryption For a message $m \in G$ and keys $k_1, k_2, \ldots, k_t \in \mathbb{Z}_p^*$, let $r, s \xleftarrow{r} \mathbb{Z}_p^*$ and output

$$c = (x, y, z, w) = (g^r, g^{r(k_1+k_2+\cdots+k_t)}, g^s, g^{s(k_1+k_2+\cdots+k_t)} m).$$

Decryption For a ciphertext $c = (x, y, z, w)$ check if any of the non-default keys $\hat{k}$ satisfies $x^{\hat{k}} = y$, if so let $k = \hat{k}$ if not let k be the default key. Output

$$c' = (x, y', z, w') = (x, x^{-k} y, z, z^{-k} w).$$

Algorithm 5 Extended path variation attack process.

compute $c_t \leftarrow \mathcal{E}((k_1, \ldots, k_t), m)$
for $i = 1, \ldots, n$ **do**
    compute $c'_t \leftarrow \mathcal{P}(c_t)$ and use it to infect a nodes in the network ▷ must infect a path node
while a node is infected with malware payload c'_i **do** ▷ performed on all infected node
    compute $c_i \leftarrow \mathcal{D}(k, c'_i)$
    if result is executable **then**
        run malware
    else
        compute $c'_{i-1} \leftarrow \mathcal{P}(c_i)$ and use it to infect a new node

Rerandomize For a ciphertext $c = (x, y, z, w)$, let $r', s' \xleftarrow{r} \mathbb{Z}_p^*$ and output

$$c' = (x', y', z', w') = (x^{r'}, y^{r'}, zx^{s'}, wy^{s'}).$$

2.6.2 Based on the extended scheme

The path variant of the extended scheme utilize an onion type encryption [3], where each onion layer is encrypted under one of the keys. This is because we need to completely remove a key from the ciphertext when we use it in a decryption. However, the rerandomize algorithm hides keys inside the ciphertext using a PRF and locks it using a group element, hence, if we encrypt each layer using one key we can remove a layer by only using a key.

Since we encrypt in layers we need a padding algorithm, $\mathcal{P}$, to pad the ciphertexts such they have the same default length, denoted L_D. The scheme pads the ciphertext after an encryption or a decryption. Note that the encryption algorithm no longer add any zeros when encrypting. The padding algorithm also rerandomize the ciphertexts, hence, it replaces the rerandomize algorithm.

Padding For a ciphertext c, encrypting a message m, the padding algorithm $\mathcal{P}(c)$ outputs a ciphertext c', encrypting the same message m, with a defined length.

See Algorithm 5 for the malware algorithm, which is specific for the extended path variation. Note that the decryption algorithm will be correct only if the malware attack process is preformed as showed in the algorithm. Thus we need a specific correctness requirement for the path version of the extended scheme. We also need a requirement for the padding algorithm, since it replaces the rerandomization algorithm.

Correctness If c'_i was output from $\mathcal{P}(\mathcal{E}(k_i, c_{i-1}))$ or $\mathcal{P}(\mathcal{D}(k_{i+1}, c_{i+1}))$ then $\mathcal{D}(k_i, c'_i)$ will always output c_{i-1} except with negligible probability.

Padding If c was output by $\mathcal{E}(k, m)$ then the output distribution of $\mathcal{P}(c)$ should be computationally indistinguishable from the output distribution of $\mathcal{E}(k, m)$.

The algorithms of the extended path scheme are as follows.

Encryption For a message $m \in \{0,1\}^L$, keys $k_1, k_2, \ldots, k_t \in \mathbb{Z}_p^*$, $\{r_i, s_i\}_{i=1}^t \xleftarrow{r} \mathbb{Z}_p^*$, and $\{\gamma_i\}_{i=1}^t \xleftarrow{r} G$. Encrypt the message in layers where

$$c_1 = g^{r_1} || g^{r_1 k_1} || g^{s_1} || g^{s_1 k_1} \gamma_1 || \left(f_L(\gamma_1) \oplus m\right)$$

and

$$c_i = g^{r_i} || g^{r_i k_i} || g^{s_i} || g^{s_i k_i} \gamma_i || \left(f_{L+2(i-1)l}(\gamma_i) \oplus c_{i-1}\right)$$

for $i \in \{2, \ldots, t\}$. Pad c_t such that it has default length L_D.

Padding For a ciphertext c. If $|\,c\,| = L_D$ let $c = x||y||b_m||b_l$, where b_l is the last l bits, otherwise let $c = x||y||b_m$.

$$\mathcal{P}(c) = x^{r'}||y^{r'}||x^{s'}||y^{s'}\gamma'||\left(f_{|b_m|+N+1}(\gamma') \oplus (b_m||1||0^N\right).$$

where $N = L_D - |b_m| - 2l - 1$.[2]

Decryption For a ciphertext $c = x_0||y_0||b_0'$ check if any of the non-default keys $\hat{k}$ satisfies $x_0^{\hat{k}} = y_0$, if so let $k = \hat{k}$ if not let k be the default key. Let $b_0' = z_0||w_0||b_0$ and compute

$$b_1' = f_{|b_0|}(z_0^{-k}w_0) \oplus b_0,$$

interpret b_1' as $z_1||w_1||b_1||1||0^{N'}$, where $N' \in \mathbb{N}$.[3] Compute

$$b_2' = f_{|b_1|}(z_1^{-k}w_1) \oplus b_1$$

and check if b_2' is an executable malware. If it is then the attack was successful, if not pad b_2'.

2.7 Security of the path scheme

2.7.1 Based on the basic scheme

The proof is as the basic proof, given in Section 2.3, except that:

- the probability space D is replaced with $D_1 \times \cdots \times D_t$,
- use vectors of samples $\mathbf{u}_1$ and $\mathbf{u}_2$ instead of u_1 and u_2, and
- use vectors of keys $\mathbf{k}_1$ and $\mathbf{k}_2$ instead of k_1 and k_2.

The opponent's advantage is bounded by

$$\mathrm{Adv}(\mathcal{O}) \leq \Pr[F_1] + \frac{1}{p} + \mathrm{Adv}_{\mathrm{ddh}}^{\text{ind-cpa}}(\mathcal{O}).$$

That is, the advantage is bounded by the opponent's ability to guess or predict the correct keys and is determined by the probability space $D_1 \times \cdots \times D_t$.

2.7.2 Based on the extended scheme

The proof is as the extended proof, given in Section 2.5, except that:

- replace the rerandomize algorithm with the padding algorithm in Experiment 1,
- the probability space D is replaced with $D_1 \times \cdots \times D_t$,
- use vectors of samples $\mathbf{u}_1$ and $\mathbf{u}_2$ instead of u_1 and u_2,
- use vectors of keys $\mathbf{k}_1$ and $\mathbf{k}_2$ instead of k_1 and k_2,
- precompute $2t$ tuples of the form

$$\{(x_j, y_j, z_j, w_j)\}_{j=1}^t = \left\{\left(g, g^{k_{1,j}}, g^{s_j}, g^{s_j k_{1,j}}\right)\right\}_{j=1}^t$$

and

$$\left\{\left(x_j', y_j', z_j', w_j'\right)\right\}_{j=1}^t = \left\{\left(g, g^{k_{2,j}}, g^{s_j'}, g^{s_j' k_{2,j}}\right)\right\}_{j=1}^t$$

instead of precomputing two tuples, and

[2] N is a multiple of l.

[3] N' will be a multiple of l zeros if the correct key is used in the decryption algorithm.

– from the first tuple, (x, y, z, w), create an additional $2t-1$ tuples, instead of one, by uniformly sample $\{a_j, b_j, c_j\}_{j=1}^{2t-1}$ and compute

$$\left\{\left(x, x^{a_j} y^{c_j}, zx^{b_j}, w^{c_j} z^{a_j} y^{c_j b_j} x^{a_j b_j}\right)\right\}_{j=1}^{2t-1}.$$

The opponent's advantage is bounded by

$$\mathrm{Adv}(\mathcal{O}) \leq \Pr[F_1] + \mathrm{Adv}_{\mathrm{ddh}}^{\text{ind-cpa}}(\mathcal{O}) + \mathrm{Adv}_{\mathrm{prf}}(\mathcal{O}).$$

That is, the advantage is bounded by the opponent's ability to guess or predict the correct keys and is determined by the probability space $D_1 \times \cdots \times D_t$.

Acknowledgments

We would like to thank Adam Young for helpful discussions and comments. We would also like to thank the anonymous reviewers for helpful comments.

References

[1] Dan Boneh. *Algorithmic Number Theory: Third International Symposiun, ANTS-III Portland, Oregon, USA, June 21–25, 1998 Proceedings*, chapter The Decision Diffie-Hellman problem, pages 48–63. Springer Berlin Heidelberg, Berlin, Heidelberg, 1998.

[2] Ran Canetti, Hugo Krawczyk, and Jesper Nielsen. Relaxing chosen-ciphertext security. Cryptology ePrint Archive, Report 2003/174, 2003. http://eprint.iacr.org/.

[3] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13*, SSYM'04, pages 21–21, Berkeley, CA, USA, 2004. USENIX Association.

[4] Eric Filiol. Strong Cryptography Armoured Computer Viruses Forbidding Code Analysis: the bradley virus. Research Report RR-5250, INRIA, 2004.

[5] Eric Filiol. Malicious cryptography techniques for unreversable (malicious or not) binaries. *CoRR*, abs/1009.4000, 2010.

[6] Ariel Futoransky, Emiliano Kargieman, Carlos Sarraute, and Ariel Waissbein. Foundations and applications for secure triggers. Cryptology ePrint Archive, Report 2005/284, 2005. http://eprint.iacr.org/.

[7] Herman Galteland and Kristian Gjøsteen. *Malware, Encryption, and Rerandomization – Everything Is Under Attack*, pages 233–251. Springer International Publishing, Cham, 2017.

[8] Philippe Golle, Markus Jakobsson, Ari Juels, and Paul Syverson. *Universal Reencryption for Mixnets*, pages 163–178. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[9] Fritz Hohl. Time limited blackbox security: Protecting mobile agents from malicious hosts, 1998.

[10] Kaspersky Lab Global Research and Analysis Team. Gauss: Abnormal distribution. In-depth research analysis report, KasperSky Lab, August 9th 2012. securelist.com/en/analysis/204792238/gauss_abnormal_distribution.

[11] James Riordan and Bruce Schneier. Environmental key generation towards clueless agents. In Giovanni Vigna, editor, *Mobile Agents and Security*, volume 1419 of *Lecture Notes in Computer Science*, pages 15–24. Springer Berlin Heidelberg, 1998.

[12] Victor Shoup. Sequences of games: a tool for taming complexity in security proofs. Cryptology ePrint Archive, Report 2004/332, 2004.

[13] Ed Skoudis and Lenny Zeltser. *Malware: Fighting Malicious Code*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2003.

[14] Adam Young and Moti Yung. Cryptovirology: extortion-based security threats and countermeasures. In *Security and Privacy, 1996. Proceedings., 1996 IEEE Symposium on*, pages 129–140, May 1996.

[15] Adam Young and Moti Yung. *Malicious Cryptography: Exposing Cryptovirology*. John Wiley & Sons, 2004.

[16] Adam Young and Moti Yung. The drunk motorcyclist protocol for anonymous communication. In *Communications and Network Security (CNS), 2014 IEEE Conference on*, pages 157–165, Oct 2014.