Malware, Encryption, and Rerandomization – Everything is Under Attack*

Herman Galteland and Kristian Gjøsteen**

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

Abstract. A malware author constructing malware wishes to infect a specific location in the network. The author will then infect n initial nodes with n different variations of his malicious code. The malware continues to infect subsequent nodes in the network by making similar copies of itself. An analyst defending M nodes in the network observes N infected nodes with some malware and wants to know if any sample is targeting any of his nodes. To reduce his work, the analyst need only look at unique malware samples. We show that by encrypting the malware payload and using rerandomization to replicate malware, we can make the N observed malware samples distinct and increase the analyst's work factor substantially.

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 other types of malware, but rather how to construct a scheme designed to encrypt malware such that we can hide the intentions of the malware author.

1.1 Real World Examples

BurnEye [11] is a tool designed to protect binary files and is an example on how to protect malware. The tool adds three protective layers to

* The final publication is available at Springer via http://dx.doi.org/978-3-319-61273-7_12

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

a file: obfuscation, encryption, and a fingerprint layer. The latter layer ensures that the file can only be run 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 [8] is an example of sophisticated malware that uses encryption to protect certain payloads. Gauss uses *environmental keys* to decrypt the payload, where an environmental key is a key that is generated from locally available data. The malware gathers local 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 some specific location(s). The malware author's goals is to hide his intentions and identity. The malware author's adversary is an *analyst* observing and defending some network containing 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. He also wants 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.

We use the following model to describe malware propagation (see also Figure 1). The source S, the malware author, infects n initial nodes with (different variations of) his malware and they, in turn, will infect subsequent nodes in the network by making similar copies of themselves. Every direct link to the malware author increases the analyst's chances of discovering the malware author's identity, so to avoid identification, the malware author should perform as few initial infections as possible and use indirect paths to his intended target.

The analyst's job is to defend M nodes in the network from any possible malware threat and he has full knowledge of the environment he is protecting. By observing the wider network the analyst can find N malware samples.

The malware author will encrypt the malware payload to make the analyst's job harder. Encrypting the payload prevents reverse engineering of the malware code [4] and hides the intentions of the author. Since

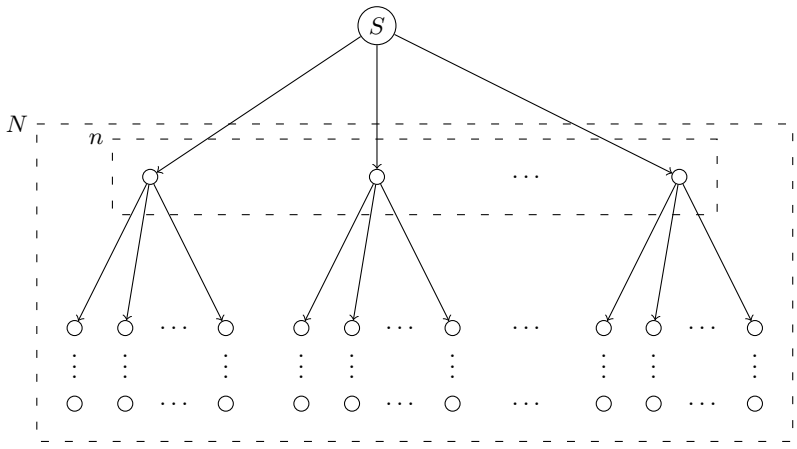

Fig. 1. Illustration of the malware infection paths

obfuscation is hard, we use encryption keys derived from environmental parameters, network triggers, or a combination of these [9]. Thus, the malware will have an encrypted payload, containing the malicious code, and a cleartext loader which gathers environmental parameters to generate decryption keys.

To generate the malware the author chooses environmental data corresponding to the intended target computer, hashes that data to create a secret key, and encrypts the payload using the key. The malware is then ready to be released. When the malware arrives on a new computer, the cleartext loader will determine the environmental data of the infected computer, hash the data to derive $L > 1$ keys, and try to decrypt the payload using the L derived keys. If the decryption is a success under one of the derived keys, the code will be executed. The cleartext loader makes copies of the malware and infect subsequent nodes.

Since only the malware author knows the secret key, only the malware author can create encryptions of the payload. This means that under our propagation model, there are at most n distinct encrypted payloads among the samples collected by the analyst. Each sample is encrypted and has a unknown target. If the analyst wants to be sure that none of these samples would attack the analyst's network, the analyst needs to do roughly L trial decryptions for each of his M nodes, which means that his work factor is nML.

Instead of making exact copies of the malware to replicate it, we want the loader to rerandomize [2, 6] the encrypted payload using techniques

When the malware, in the form of a cleartext loader and an encrypted payload, arrives on a new host, the cleartext loader is executed and performs the following steps:

1. The encrypted payload is rerandomized before it is stored on the host.
2. The loader scans the host environment and determines the environmental data.
3. The loader hashes the environmental data to produce one or more keys.
4. The loader tries to decrypt the encrypted payload with each key.
5. If the decryption succeeds, the decrypted payload is executed.
6. 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 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.

Fig. 2. The malware attack process.

from asymmetric cryptography. The rerandomization process takes as input an encrypted payload and some random numbers and produces a new encrypted payload that encrypts the same malicious code. Hence, the loader can produce several different-looking malware payloads to infect subsequent nodes, without knowledge of the secret key. This process is described in Figure 2.

To fully utilize the rerandomization process, we want it to produce the 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 L trial decryptions of N samples for M different nodes to ensure that none of the malware samples are targeting any of his nodes. This will increase the workload to NML. Since the malware creates new different variations of itself, the malware author can now choose n to be small and possibly significantly reducing the risk of detection.

Now, imagine a world filled with hundreds of different types of malware: all are encrypted, use rerandomization, are of the same size, use the same loader, and otherwise look the same. Every new malware sample an analyst would discover needs to be analyzed. The analyst cannot be certain of whether a new sample corresponds to one he has previously determined is no threat, or a genuinely new piece of malware. Note that this requires the various malware authors to agree on standard payload sizes and a standard loader. If they do not, then the analyst can use these pieces of information to classify samples.

The limitations with our scheme 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.

Another limitation is that 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 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.

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) [13] and have several works related to malware construction and propagation: A virus capable of encrypting files on the victim's computer and hold them for ransom [12]. A mobile program that carries a rerandomizable ciphertext, which enables anonymous communication, where the program takes random walks through a network in a system called Feralcore, and, at each node, the ciphertext is rerandomized [14]. Utilizing a mix network to mix programs and propagate malware [13].

The mix network and the mobile program mentioned use the idea of universal re-encryption, by Golle et al. [6], to re-encrypt ciphertexts. The 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 [3, 4] one can prevent anyone from analyzing the code and reverse engineer it, possibly using the environmental keys of Riordan and Schneier [9] as the encryption key. Similar to Riordan and Schneier, secure triggers [5, 7] are used to keep certain content private until some particular event occurs.

1.4 Overview

In Section 2.1 we describe the cryptosystem designed to encrypt and rerandomize malware payload. In Section 2.2 we construct a basic scheme

based on ElGamal. In Section 2.3 we show that the basic scheme is secure by using games, where the adversary is asked to distinguish between ciphertexts encrypting the same message and ciphertexts encrypting two different messages. That is, we will simulate whether the analyst is able to distinguish malware samples. In Section 2.4 we construct an extended scheme based on the basic scheme, which is capable of encrypting longer messages. In Section 2.5 we show that the extended scheme is secure using games. The procedure is similar to the security proof of the basic scheme.

2 Rerandomizable Encryption

We will in this section describe and construct the encryption scheme designed to encrypt and rerandomize malware payload. We will construct two (similar) example schemes, as proofs of concept, and show that it is hard to distinguish between encrypted payload samples.

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

2.1 Preliminary

In our scheme we have an algorithm $\mathcal{E}$ encrypting messages, an algorithm $\mathcal{D}$ decrypting ciphertexts, and an algorithm $\mathcal{R}$ rerandomizing ciphertexts.

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 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.

We want the output distribution of the rerandomize algorithm to 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. We also want the system to be correct, that is, 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.

The security requirements of our cryptosystem 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

We will construct a basic scheme based on the ElGamal cryptosystem over a group G of prime order p generated by g. The basic scheme is essentially the same as the encryption scheme proposed by Golle et al [6]. The algorithms of the scheme is the following.

Encryption For a message $m \in G$ and a key $k \in \{1, \ldots, p-1\}$ pick $r, s \in \{1, \ldots, p-1\}$ uniformly 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 $k \in \{1, \ldots, p-1\}$ check if $x^k = y$. If not, output a symbol indicating decryption failure. If it is, output

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

Rerandomize For a ciphertext $c = (x, y, z, w)$ pick $r', s' \in \{1, \ldots, p-1\}$ uniformly and output

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

Note that if $c = (x, y, z, w)$ was output by the encryption algorithm then there exists parameters r, s, and 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 a $c' = (x', y', z', w')$ 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, we get that the output distribution of the encryption and rerandomize algorithms are computationally indistinguishable. Note that the ciphertext c' has the same form as a ciphertext output by the encryption algorithm, that is, $(g^{\hat{r}}, g^{\hat{k}\hat{r}}, g^{\hat{s}}, g^{\hat{k}\hat{s}}m)$, for some parameters $\hat{r}$, $\hat{s}$, and $\hat{k}$, and message m.

We can now show the correctness of the decryption algorithm. Note that for all ciphertexts $c = (x, y, z, w)$ we have that $x^k = (g^r)^k = g^{kr} = y$, which is true for ciphertexts output by both the encryption and rerandomize algorithms. We can therefore retrieve the message m by computing

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

Thus the decryption algorithm is correct.

It is possible to extend the basic scheme by encrypting several messages under the same key. For a set of messages $m_1, m_2, \ldots, m_n$, we can encrypt them 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 a key k, and variables $s_1, s_2, \ldots, s_n$, and r. This is not a very efficient method, and we will in Section 2.4 construct a different extended scheme by using techniques from symmetric cryptography. In the next section we will show that the basic scheme is secure.

2.3 Security of the Basic Scheme

We will in this section use games to show that the basic scheme is secure given that it is hard to guess which environmental key the ciphertexts are encrypted under.

The key encrypting the malware payload is derived from environmental parameters sampled by the loader. From the adversary's perspective, the collection of sampled parameter 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 adversary is less likely to guess the correct decryption key, where the size of D is determined by, most notably, the number of different parameters the loader is gathering.

We want to show that the adversary is unable to distinguish between ciphertexts and that his advantage is determined by D, that is, the probability of the adversary guessing the correct key. To do so, we will use a sequence of games [10]. In our games we will start with simulating an experiment where we ask the adversary to differentiate between two cases; ciphertexts encrypting different messages, and ciphertexts encrypting the same message.

Experiment Given two ciphertext c_1, and c_2, decide either

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

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

We can show that the security of the scheme can be based on the hardness of the Decisional Diffie-Hellman (DDH) problem [1] in the random oracle model. The DDH problem is to distinguish between 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 $a, b, c \in \{1, \ldots, p-1\}$. Where the DDH assumption states that the DDH problem is hard to solve.

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

Game 0 In the first game we will follow the experiment. If $b = 0$, we will encrypt the two given messages under two different keys. If $b = 1$, we will encrypt only one of the messages and rerandomize the resulting ciphertext. In both cases, we send the ciphertexts to the adversary, who

Game 0:

$u_1, u_2 \leftarrow D$, $k_1 \leftarrow H(u_1)$, $k_2 \leftarrow H(u_2)$, $b \xleftarrow{r} \{0,1\}$

Get m_1, m_2 from A

If $b = 0$ do:

$r, r', s, s' \xleftarrow{r} \{1, \ldots, p-1\}$,

$c_1 \leftarrow (x, y, z, w) = (g^r, g^{k_1 r}, g^s, g^{k_1 s} m_1)$

$c_2 \leftarrow (x', y', z', w') = (g^{r'}, g^{k_2 r'}, g^{s'}, g^{k_2 s'} m_2)$

Send c_1, c_2 to A

If $b = 1$ do:

$r, r', s, s' \xleftarrow{r} \{1, \ldots, p-1\}$,

$c_1 \leftarrow (x, y, z, w) = (g^r, g^{k_1 r}, g^s, g^{k_1 s} m_1)$

$c_2 \leftarrow (x', y', z', w') = (x^{r'}, y^{r'}, z x^{s'}, w y^{s'})$

Send c_1, c_2 to A

Get b' from A

Fig. 3. Game 0 of the basic scheme

replies with a bit b' and the game ends. The full procedure of Game 0 can be seen in Figure 3.

Let E_0 denote the event that $b = b'$ in Game 0.

Game 1 We will stop the game if the adversary guesses one of the keys correctly. If the adversary gives either u_1 or u_2 in one of its queries the oracle will: flip a coin, $b' \xleftarrow{r} \{0,1\}$, output b', and stop the game. We will denote this event by F_1.

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

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

Game 2 Since the adversary can no longer use the oracle to get any information about the keys without stopping the game, we are essentially drawing our keys randomly from a set. That is, we draw $k_1, k_2 \xleftarrow{r} \{1, \ldots, p-1\}$ uniformly and we will no longer query the hashing oracle. Note that since the adversary can still query the hashing oracle, we still need to draw samples from the space D to check if the adversary is guessing the keys correctly.

Let E_2 denote the event that $b = b'$ in Game 2. Since the adversary can no longer get any information about the environmental keys from the

hashing oracle without stopping the game, the keys used are, essentially, some random group elements. Hence, $\Pr[E_2] = \Pr[E_1]$.

Game 3 We change how we compute the tuples such that we the encryption algorithm do not require the keys as input. To do so we will precompute the tuples before we receive the messages. That is, for some uniform $s, s' \in \{1, \ldots, p-1\}$ and keys k_1, k_2, we will compute

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

before we receive the messages m_1 and m_2.

In the case $b = 0$, we will encrypt the two messages using the precomputed tuples. That is, we will pick a random element per message, r and r', and compute

$$c_1 = (x^r, y^r, z, wm_1) = (g^r, g^{k_1 r}, g^s, g^{k_1 s} m_1),$$
$$c_2 = (x'^{r'}, y'^{r'}, z', w' m_2) = (g^{r'}, g^{k_2 r'}, g^{s'}, g^{k_2 s'} m_2).$$

In the case $b = 1$, we will encrypt one message and rerandomize the computed ciphertext. To encrypt m_1 we pick a random element r and compute

$$c_1 = (x^r, y^r, z, wm_1) = (g^r, g^{k_1 r}, g^s, g^{k_1 s} m_1).$$

To rerandomize $c_1 = (\hat{x}, \hat{y}, \hat{z}, \hat{w})$ we draw some uniform element r' and s', as usual, and compute

$$\begin{aligned} c_2 = (\hat{x}^{r'}, \hat{y}^{r'}, \hat{z}\hat{x}^{s'}, \hat{w}\hat{y}^{s'}) &= (x^{rr'}, y^{rr'}, zx^{rs'}, wy^{rs'} m_1) \\ &= (g^{rr'}, g^{k_1 rr'}, g^{s+rs'}, g^{k_1(s+rs')} m_1). \end{aligned}$$

Let E_3 denote the event that $b = b'$ in Game 3. The output distribution of the encryption algorithm in Game 2 and in Game 3 are exactly the same, similarly for the rerandomization algorithm. Therefore, we have that $\Pr[E_3] = \Pr[E_2]$.

Game 4 We will change the way we create the second tuple, which we use to encrypt the second message in the case $b = 0$. Now we will only make one tuple for the first key k_1 and use the first tuple to create the

second. Let $(x, y, z, w) = (g, g^{k_1}, g^s, g^{k_1 s})$ be the first tuple, the second tuple will then be

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

for some uniformly sampled $a, b, c \in \{1, \ldots, p-1\}$. Note that the second tuple will still be computed before we receive the messages.

Let E_4 be the event that $b = b'$ in Game 4. Since the new tuple results in the same output space when it is used for encrypting messages we get that $\Pr[E_4] = \Pr[E_3]$.

Game 5 We will in this game change the rerandomize algorithm. The output of the encryption and the rerandomize algorithm can be seen as two vectors, (x, y) and (z, w). For the encryption algorithm the first vector will always stay in the subgroup of $G \times G$ generated by (g, g^{k_1}), for a key k_1, and the second will always stay in the same coset of this subgroup. The first vector in the output of the rerandomize algorithm also stay the subgroup $G \times G$ generated by (g, g^{k_1}), however the second does not stay in the same coset. That is, the output of the rerandomize algorithm looks like

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

for some r, r', s and s', where the sum of $s + rs'$ cannot be equal to s since none of the variables used in the algorithm can be zero. Therefore, there is a statistical difference of $1/p$ between the output distributions. We will instead compute the rerandomization of the first ciphertext (in the case $b = 1$) as

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

where $\tilde{s}$ is a uniform element in $\{1, \ldots, p-1\}$. The new sum $s + rs' + \tilde{s}$ can now be any value in $\{1, \ldots, p-1\}$, 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

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

Algorithm $B((x, y, z, w))$:

$u_1, u_2 \xleftarrow{r} \mathcal{D}$, $b \xleftarrow{r} \{0, 1\}$
$a, b, c \xleftarrow{r} \{1, \ldots, p-1\}$
$(x', y', z', w') = (x, x^a y^c, z x^b, w^c z^a y^{cb} x^{ab})$
Get m_1, m_2 from A

If $b = 0$ do:
$r, r' \xleftarrow{r} \{1, \ldots, p-1\}$
$c_1 \leftarrow (x^r, y^r, z, w m_1)$
$c_2 \leftarrow (x'^{r'}, y'^{r'}, z', w' m_2)$
Send c_1, c_2 to A

If $b = 1$ do:
$r, r', s', \tilde{s} \xleftarrow{r} \{1, \ldots, p-1\}$
$c_1 \leftarrow (x^r, y^r, z, w m_1)$
$c_2 \leftarrow (x^{rr'}, y^{rr'}, z x^{rs'+\tilde{s}}, w y^{rs'+\tilde{s}} m_1)$
Send c_1, c_2 to A

Get b' from A

Fig. 4. Algorithm B

Game 6 In the last game, we will turn the first tuple into the form $(g, g^{a'}, g^{b'}, g^{c'})$, for some uniform elements $a', b', c' \in \{1, \ldots, p-1\}$. The second tuple will then look like

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

Both the encryption and rerandomization algorithms will output ciphertexts which can result in any group element when decrypted.

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

To show the connection to the previous game we will use algorithm B, see Figure 4. We claim that $|\Pr[E_5] - \Pr[E_6]| = \mathrm{Adv}_{\mathrm{ddh}}^{\text{ind-cpa}}$, the DDH-advantage (with respect to indistinguishably under chosen plaintext attack, that is, semantic security). The input to the algorithm B is a tuple (x, y, z, w) which looks like (g, g^a, g^b, g^c), for some a, b, and c, where c can be equal to ab. Therefore, the algorithm will simulate Game 5 and Game 6 depending on its input. When the input is on the form (g, g^a, g^b, g^{ab}), the algorithm will proceed just as in Game 5, and therefore

$$\Pr[B(g, g^a, g^b, g^{ab}) = 1 \mid a, b \xleftarrow{r} \{1, \ldots, p-1\}] = \Pr[E_5].$$

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

$$\Pr[B(g, g^a, g^b, g^c) = 1 \mid a, b, c \xleftarrow{r} \{1, \ldots, p-1\}] = \Pr[E_6],$$

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

Recap We can now use the results from the games to bound the advantage of the adversary.

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

By the DDH assumption the DDH advantage is negligible. Therefore, for a large enough p, we get that the advantage of our adversary is determined by the probability that A guesses or predicts the correct key, that is, determined by the probability space D.

2.4 Extended Scheme

We will extend the basic scheme to longer messages by representing them as bit strings. This change will also reduce the number of rerandomizations we can perform on a ciphertext. Therefore, we need to relax the requirements of the cryptosystem slightly. The construction in this section is very similar to the hybrid scheme by Golle et al [6].

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 the failure symbol and output m except with negligible probability.

We will require 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 will assume that group elements can be encoded as bit strings of length $l/2$.

Encryption For a message $m \in \{0,1\}^L$ and a key $k \in \{1,\ldots,p-1\}$ pick $r, s \in \{1,\ldots,p-1\}$ and $\gamma \in G$ uniformly, 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 $k \in \{1,\ldots,p-1\}$ check if $x^k = y$. If not, output a symbol representing decryption failure. 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. If the result does not end with a tail of l' zeros, then interpret b'_1 as $z_1||w_1||b_1$ and repeat the procedure. If the decryption algorithm is repeated $n+1$ times, output a symbol representing decryption failure.

Rerandomization For a ciphertext $c = x||y||b_\alpha||b_\beta$, where b_β is the last l bits, pick $r', s' \in \{1,\ldots,p-1\}$ and $\gamma' \in G$ uniformly, output

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

Note that, before applying the rerandomize algorithm, b_α looks like

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

for some $s \in \{1,\ldots,p-1\}$, key k, and $\gamma \in G$. The l last bits we discard, i.e., b_β, 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. However, we get that the length of the ciphertext is preserved.

We will now show the correctness of the decryption algorithm. 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

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

Since the result ends with a tail of $l' \geq l$ zeros 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 some $s', r_1, \ldots, r_{t+1} \in \{1, \ldots, p-1\}$, key k, and group element $\gamma_t \in G$. Observe that for all $1 \leq t \leq n$, we have that $x^k = y$. Hence, we can write $b'_t = z_t||w_t||b_t$ and compute

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

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). Therefore, 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 will perform the decryption on a bit string which looks like $z_0||w_0||b_0$, where we now have that b_0 looks like

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

which we know decrypts to the message m. That is, the decryption algorithm is correct.

2.5 Security of the Extended Scheme

We will in this section show that the adversary is unable to distinguish between encrypted ciphertexts and that his advantage is determined by D, that is, the probability of the adversary guessing the correct key. As in the proof of the basic scheme, we will use games to simulate the same experiment. Since the two example schemes are similar the games will be too.

Game 0 In the first game we will simulate the experiment. We ask the adversary to differentiate between ciphertexts encrypting two different messages and ciphertexts encrypting the same message. The full procedure of the game can be seen in Figure 5.

Let E_0 be the event that $b = b'$ in Game 0.

Game 1 We will stop the game if the adversary guesses one of the keys correctly. If the adversary sends either u_0 or u_1 in one of its queries, the oracle will flip a coin, $b' \xleftarrow{r} \{0, 1\}$, output b', and stop the game. We will denote this event by F_1.

Game 0:

$u_1, u_2 \xleftarrow{r} D$, $k_1 \leftarrow H(u_1)$, $k_2 \leftarrow H(u_2)$, $b \xleftarrow{r} \{0,1\}$

Get m_1, m_2 from A

If $b = 0$ do:

$r, r', s, s', \gamma, \gamma' \xleftarrow{r} \{1, \ldots, p-1\}$

$c_1 \leftarrow g^r || g^{k_1 r} || g^s || g^{k_1 s} \gamma || \left(f_{L+l(n+1)+1}(\gamma) \oplus (m_1 || 1 || 0^{l(n+1)}) \right)$

$c_2 \leftarrow g^{r'} || g^{k_2 r'} || g^{s'} || g^{k_2 s'} \gamma' || \left(f_{L+l(n+1)+1}(\gamma') \oplus (m_2 || 1 || 0^{l(n+1)}) \right)$

Send c_1, c_2 to A

If $b = 1$ do:

$r, r', s, s', \gamma, \gamma' \xleftarrow{r} \{1, \ldots, p-1\}$

$c_1 \leftarrow g^r || g^{k_1 r} || g^s || g^{k_1 s} \gamma || \left(f_{L+l(n+1)+1}(\gamma) \oplus (m_1 || 1 || 0^{l(n+1)}) \right)$

Write c_1 as $x||y||b_\alpha||b_\beta$, where b_β is the last l bits

$c_2 \leftarrow x^{r'} || y^{r'} || x^{s'} || y^{s'} \gamma' || \left(f_{|b_\alpha|}(\gamma') \oplus b_\alpha \right)$

Send c_1, c_2 to A

Get b' from A

Fig. 5. Game 0 of the extended scheme

Let E_1 denote the event that $b = b'$ in Game 1. If the event F_1 does not occur then Game 0 and Game 1 are equal. That is, $E_0 \wedge \neg F_1 \iff E_1 \wedge \neg F_1$ and by the difference lemma we get that

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

Game 2 Since the adversary can no longer use the oracle to get any information about the keys without stopping the game, we are essentially drawing our keys at random from a set. That is, we draw $k_1, k_2 \xleftarrow{r} \{1, \ldots, p-1\}$ uniformly, and we will no longer query the hashing oracle. Note that since the adversary can still query the hashing oracle, we still need to draw samples from the space D to check if the adversary is guessing the correct keys.

Let E_2 denote the event that $b = b'$ in Game 2. Since the adversary can no longer get any information about the environmental keys from the hashing oracle without stopping the game, the keys used are, essentially, some random group elements. Hence, $\Pr[E_2] = \Pr[E_1]$.

Game 3 We change how we compute the tuples such that we the encryption algorithm do not require the keys as input. We will therefore

precompute the tuples before we receive the messages. That is, for some uniform $s, s' \in \{1, \ldots, p-1\}$ and key k_1, k_2, we will compute

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

before we receive the messages m_1 and m_2.

In the case $b = 0$, we will encrypt the two messages using the precomputed tuples. That is, we will pick r, r', γ and γ' uniformly, and compute

$$\begin{aligned}
c_1 &= x^r||y^r||z||w\gamma|| \left(f_{L+l(n+1)+1}(\gamma) \oplus (m_1||1||0^{l(n+1)})\right) \\
&= g^r||g^{k_1 r}||g^s||g^{k_1 s}\gamma|| \left(f_{L+l(n+1)+1}(\gamma) \oplus (m_1||1||0^{l(n+1)})\right), \\
c_2 &= x'^{r'}||y'^{r'}||z'||w'\gamma'|| \left(f_{L+l(n+1)+1}(\gamma') \oplus (m_2||1||0^{l(n+1)})\right) \\
&= g^{r'}||g^{k_2 r'}||g^{s'}||g^{k_2 s'}\gamma'|| \left(f_{L+l(n+1)+1}(\gamma') \oplus (m_2||1||0^{l(n+1)})\right).
\end{aligned}$$

In the case $b = 1$, we will encrypt one message and rerandomize the computed ciphertext. That is, to encrypt, we pick r and γ uniformly, and compute

$$\begin{aligned}
c_1 &= x^r||y^r||z||w\gamma|| \left(f_{L+l(n+1)+1}(\gamma) \oplus (m_1||1||0^{l(n+1)})\right) \\
&= g^r||g^{k_1 r}||g^s||g^{k_1 s}\gamma|| \left(f_{L+l(n+1)+1}(\gamma) \oplus (m_1||1||0^{l(n+1)})\right).
\end{aligned}$$

To rerandomize let $c_1 = x^r||y^r||b_\alpha||b_\beta$, where b_β is the last l bits, pick two element r', s' and a group element γ', as usual, and compute

$$\begin{aligned}
c_2 &= x^{rr'}||y^{rr'}||x^{rs'}||y^{rs'}\gamma'|| \left(f_{|b_\alpha|}(\gamma') \oplus b_\alpha\right) \\
&= g^{rr'}||g^{k_1 rr'}||g^{rs'}||g^{k_1 rs'}\gamma'|| \left(f_{|b_\alpha|}(\gamma') \oplus b_\alpha\right).
\end{aligned}$$

Let E_3 denote the event that $b = b'$ in Game 3. The output distribution of the encryption algorithm in Game 2 and in Game 3 are exactly the same, similarly for the rerandomization algorithm. Therefore, we have that $\Pr[E_3] = \Pr[E_2]$.

Game 4 We will only make one tuple and use it to create the second one. The first tuple will be $(x, y, z, w) = (g, g^{k_1}, g^s, g^{k_1 s})$ and the second tuple looks will then be

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

Algorithm $B((x, y, z, w))$:

$u_1, u_2 \xleftarrow{r} \mathcal{D}$, $b \xleftarrow{r} \{0, 1\}$

$a, b, c \xleftarrow{r} \{1, \ldots, p-1\}$

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

Get m_1, m_2 from A

If $b = 0$ do:

$r, r'\gamma, \gamma' \xleftarrow{r} \{1, \ldots, p-1\}$

$c_1 \leftarrow x^r || y^r || z || w\gamma || \left(f_{L+l(n+1)+1}(\gamma) \oplus (m_1 || 1 || 0^{l(n+1)}) \right)$

$c_2 \leftarrow x'^{r'} || y'^{r'} || z' || w'\gamma' || \left(f_{L+l(n+1)+1}(\gamma') \oplus (m_2 || 1 || 0^{l(n+1)}) \right)$

Send c_1, c_2 to A

If $b = 1$ do:

$r, r's', \gamma, \gamma' \xleftarrow{r} \{1, \ldots, p-1\}$

$c_1 \leftarrow x^r || y^r || z || w\gamma || \left(f_{L+l(n+1)+1}(\gamma) \oplus (m_1 || 1 || 0^{l(n+1)}) \right)$

Write c_1 as $x^r || y^r || b_\alpha || b_\beta$, where b_β is the last l bits

$c_2 \leftarrow x^{rr'} || y^{rr'} || x^{rs'} || y^{rs'}\gamma' || \left(f_{|b_\alpha|}(\gamma') \oplus b_\alpha \right)$

Send c_1, c_2 to A

Get b' from A

Fig. 6. Algorithm B

for some uniformly sampled $a, b, c \in \{1, \ldots, p-1\}$.

Let E_4 be the event $b = b'$ in Game 4. Since the new tuple results in the same output space when it is used for encrypting messages we get that $\Pr[E_4] = \Pr[E_3]$.

Game 5 We will now make the first tuple to have the form $(g, g^{a'}, g^{b'}, g^{c'})$, for some uniform $a', b', c' \in \{1, \ldots, p-1\}$, where the second tuple will look like

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

Let E_5 be the event $b = b'$ in Game 5. We will use algorithm B, see Figure 6, to show that $|\Pr[E_4] - \Pr[E_5]|$ is equal to the DDH-advantage. If the input of the algorithm is a tuple on the form (g, g^a, g^b, g^{ab}), then the algorithm proceed as in Game 4. If the tuple is on the form (g, g^a, g^b, g^c), then the algorithm proceed as in Game 5. Therefore, the DDH-advantage is equal to $|\Pr[E_4] - \Pr[E_5]|$.

Algorithm $B'((x, y, z, w))$:

$u_1, u_2 \xleftarrow{r} \mathcal{D}$, $b \xleftarrow{r} \{0,1\}$, $h \leftarrow \Gamma$

$a, b, c \xleftarrow{r} \{1, \ldots, p-1\}$

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

Get m_1, m_2 from A

If $b = 0$ do:

$r, r', \gamma, \gamma' \xleftarrow{r} \{1, \ldots, p-1\}$

$c_1 \leftarrow x^r || y^r || z || w\gamma || \left(h_{L+l(n+1)+1}(\gamma) \oplus (m_1 || 1 || 0^{l(n+1)}) \right)$

$c_2 \leftarrow x'^{r'} || y'^{r'} || z' || w'\gamma' || \left(h_{L+l(n+1)+1}(\gamma') \oplus (m_2 || 1 || 0^{l(n+1)}) \right)$

Send c_1, c_2 to A

If $b = 1$ do:

$r, r', s', \gamma, \gamma' \xleftarrow{r} \{1, \ldots, p-1\}$

$c_1 \leftarrow x^r || y^r || z || w\gamma || \left(h_{L+l(n+1)+1}(\gamma) \oplus (m_1 || 1 || 0^{l(n+1)}) \right)$

Write c_1 as $x^r || y^r || b_\alpha || b_\beta$, where b_β is the last l bits

$c_2 \leftarrow x^{rr'} || y^{rr'} || x^{rs'} || y^{rs'} \gamma' || \left(h_{|b_\alpha|}(\gamma') \oplus b_\alpha \right)$

Send c_1, c_2 to A

Get b' from A

Fig. 7. Algorithm B'

Game 6 In the last game, we will sample a function h from a family Γ of all functions from G to $\{0,1\}^N$ instead of using the function f. We want to show that the pseudorandom function (PRF) f can reliably hide the message. The PRF-advantage of an efficient adversary is defined by his ability to distinguishing the function f from any function h sampled from Γ. The PRF-advantage of the adversary is negligible assuming the function f is pseudorandom. Just like for f, we will denote h_L as the truncation of the output to L bits.

Let E_6 be the event $b = b'$ in Game 6. From Game 5 we have that the new tuples looks like $(g, g^{a'}, g^{b'}, g^{c'})$, for some random variables a', b', and c'. Hence, we will not be able to retrieve γ when we try to decrypt the ciphertext encrypting it. Since we are now using any function h, with the random group element γ, to encrypt the message m we are essentially XOR-ing a random bit string to the message. Therefore, the output ciphertexts of the encryption and rerandomization algorithms can be any random bit string and we get that $\Pr[E_6] = 1/2$.

By using the algorithm B', as seen in Figure 7, we can show that the difference between Game 5 and Game 6 is equal to the PRF-advantage. The algorithm draws a function h from the family Γ, which may be equal

to f. Hence, we get that the PRF-advantage is

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

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

Recap We can now use the results from the games to bound the advantage of the adversary.

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

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

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. Eric Filiol. Strong Cryptography Armoured Computer Viruses Forbidding Code Analysis: the bradley virus. Research Report RR-5250, INRIA, 2004.
4. Eric Filiol. Malicious cryptography techniques for unreversable (malicious or not) binaries. *CoRR*, abs/1009.4000, 2010.
5. 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/.

6. Philippe Golle, Markus Jakobsson, Ari Juels, and Paul Syverson. *Universal Reencryption for Mixnets*, pages 163–178. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.
7. Fritz Hohl. Time limited blackbox security: Protecting mobile agents from malicious hosts, 1998.
8. 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.
9. 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.
10. Victor Shoup. Sequences of games: a tool for taming complexity in security proofs. Cryptology ePrint Archive, Report 2004/332, 2004.
11. Ed Skoudis and Lenny Zeltser. *Malware: Fighting Malicious Code*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2003.
12. 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.
13. Adam Young and Moti Yung. *Malicious Cryptography: Exposing Cryptovirology*. John Wiley & Sons, 2004.
14. 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.