Disguisable Symmetric Encryption Schemes for an Anti-forensics Purpose[⋆]

Ning Ding, Dawu Gu, and Zhiqiang Liu

Department of Computer Science and Engineering
Shanghai Jiao Tong University
Shanghai, 200240, China
{dingning,dwgu,ilu_zq}@sjtu.edu.cn

Abstract. In this paper, we propose a new notion of secure disguisable symmetric encryption schemes, which captures the idea that the attacker can decrypt a cipher text he encrypted to different meaningful values when different keys are put to the decryption algorithm. This notion is aimed for the following anti-forensics purpose: the attacker can cheat the forensics investigator by decrypting an encrypted file to a meaningful file other than that one he encrypted, in the case that he is catched by the forensics investigator and ordered to hand over the key for decryption.

We then present a construction of secure disguisable symmetric encryption schemes. Typically, when an attacker uses such encryption schemes, he can achieve the following two goals: if the file he encrypted is an executable malicious file, he can use fake keys to decrypt it to a benign executable file, or if the file he encrypted is a data file which records his malicious activities, he can also use fake keys to decrypt it to an ordinary data file, e.g. a song or novel file.

Keywords: Symmetric Encryption, Obfuscation, Anti-forensics.

1 Introduction

Computer forensics is usually defined as the set of techniques that can be applied to understand if and how a system has been used or abused to commit mischief [8]. The increasing use of forensics techniques has led to the development of "anti-forensics" techniques that can make this process difficult, or impossible [2][7][6]. That is, the goal of anti-forensics techniques is to frustrate forensics investigators and their techniques.

In general, the anti-forensics techniques mainly contains those towards data wiping, data encryption, data steganography and techniques for frustrating forensics software etc. When an attacker performs an attack on a machine (called the target machine), there are much evidence of the attack left in the target machine and his own machine (called the tool machine). The evidence usually includes malicious data, malicious programs etc. used throughout the attack. To frustrate

⋆ This work was supported by the Specialized Research Fund for the Doctoral Program of Higher Education (No. 200802480019).

forensics investigators to gather such evidence, the attacker usually tries to erase these evidence from the target machine and the tool machine after or during the attack. Although erasing the evidence may be the most efficient way to prevent the attacker from being traced by the forensics investigator, the attacker sometimes needs to store some data and malicious programs in the target machine or the tool machine so as to continue the attack later. In this case the attacker may choose to encrypt the evidence and then later decrypt it when needed.

A typical encryption operation for a file (called the plain text) is to first encrypt it and then erase the plain text. Thus after this encrypting operation, it seems that there is only the encrypted file (called the cipher text) in the hard disk and does not exist the plain text. However, some forensics software can recover the seemingly erased file or retrieve the plain text corresponding to a cipher text in the hard disk by making use of the physical properties of hard disks and the vulnerability of the operation systems. Thus, some anti-forensics researchers proposed some techniques on how to really erase or encrypt data such that no copy of the data or plain text still exists in the hard disk. By adopting such anti-forensics techniques, it can be ensured that there exist only encrypted data left in the machine. Thus, if the encryption scheme is secure in cryptographic sense, the forensics investigator cannot find any information on the data if he does not know the private key. Hence it seems that by employing the really erasing techniques and a secure encryption scheme, the attacker could realize secure encryption of malicious data and programs and avoid accusation even if the forensics investigator can gather cipher texts from the target machine or the tool machine since none can find any information from these cipher texts. But is this really true in all cases?

Consider such a case. The attacker uses a secure encryption scheme to encrypt a malicious executable file. But later the forensics investigator catches him and also controls the tool or target machines absolutely. Suppose the forensics investigator can further find the encrypted file of the malicious program by scanning the machine. Then the forensics investigator orders the attacker to hand over the private key so as to decrypt the file to obtain the malicious program. In this case, the attacker cannot hand over a fake key to the investigator since by using this fake key as the decryption key, either the decryption cannot proceed successfully or even if the decryption can proceed successfully, the decrypted file is usually not an executable file. This shows to the investigator that the attacker lies to him. Thus the inquest process will not end unless the attacker hands over the real key. So it can be seen that the secrecy of the cipher text cannot be ensured in this case.

The above discussion shows that ordinary encryption schemes may be insufficient for this anti-forensics purpose even if they possess strong security in cryptographic sense (e.g. IND-CCA2). One method of making the attacker able of cheating the forensics investigator is to let the encrypted file has multiple valid decryptions. Namely, each encryption of an executable file can be decrypted to more than one different executable files. Assuming such encryption schemes exist, in the above case when ordered to hand over the real key, the attacker can hand over one or more fake keys to the forensics investigator and the cipher text can be correspondingly decrypted to one or many benign executable programs, which are not the malicious program. Then the attacker can cheat the investigator that the program encrypted previously would be actually a benign program instead of a malicious program. Thus, the forensics investigator cannot accuse the attacker that he lies to the investigator. We say that an encryption scheme with such security is disguisable (in anti-forensics setting).

It can be seen that the disguisable encryption may be only motivated for the anti-forensics purpose and thus the standard encryption study does not investigate it explicitly and to our knowledge no existing encryption scheme is disguisable. Thus, in this paper we are interested in the question how to construct disguisable encryption schemes and try to provide an answer to this question.

1.1 Our Result

We provide a positive answer to the above question with respect to the symmetric encryption. That is, we first put forward a definition of secure disguisable symmetric encryption which captures the idea that a cipher text generated by the attacker can be decrypted to different meaningful plain texts when using different keys to the decryption algorithm. A bit more precisely, the attacker holds a real key and several fake keys and uses the real key to encrypt a file to output the cipher text. Then if the attacker is controlled by the forensics investigator and ordered to hand over the key to decrypt the cipher text, the attacker can hand over one or more fake keys and claims that these keys include the real one. We also require that the forensics investigator cannot learn any information of the number of all the keys the attacker holds.

Then we present a construction of secure disguisable symmetric encryption schemes. Informally, our result can be described as follows.

Claim 1. There exists a secure disguisable symmetric encryption scheme.

When an attacker encrypted a file using such encryption schemes, he can cheat the forensics investigator later by decrypting the encryption of the malicious file to another file. In particular, if an attacker used a secure disguisable symmetric encryption scheme to encrypt a malicious executable file and later is ordered to decrypt the cipher text, then the attacker can decrypt the cipher text to a benign executable file, or decrypt it to a malicious program other than the real encrypted one which, however, is unrelated to the attack. Or, if the attacker encrypted some data file which records his malicious activities, then later he can decrypt this cipher text to an ordinary data file, such as a song or a novel file. In both cases, the forensics investigator cannot recognize attacker's cheating activities.

For an encryption scheme, all security is lost if the private key is lost. Thus the attacker who uses a disguisable encryption scheme should ensure that the keys (the real one and many fakes ones) can be stored in a secure way. In the last part of this paper, we also provide some discussion on how to securely manage the keys.

1.2 Our Technique

Our construction of disguisable symmetric encryption schemes heavily depends on the the recent result of obfuscating multiple-bit point and set-membership functions proposed by [4]. Loosely speaking, an obfuscation of a program P is a program that computes the same functionality as P computes, but any adversary can only use this functionality and cannot learn anything beyond it, i.e., the adversary cannot reverse-engineering nor understand the code of the obfuscated program. A multiple-bit point function $MBPF_{x,y}$ is the one that on input x outputs y and outputs $\perp$ on all other inputs. As shown by [4], an obfuscation for multiple-bit point functions can be applied to construct a symmetric encryption scheme: The encryption of a message m with key k is letting $O(MBPF_{k,m})$ be the cipher text. To decrypt the cipher text with k is to compute $O(MBPF_{k,m})(k)$, which output is m.

Inspired by [4], we find that an obfuscation for multiple-bit set-membership functions can be used to construct a disguisable symmetric encryption scheme. A multiple-bit set-membership function $MBSF_{(x_1,y_1),(x_2,y_2),\cdots,(x_t,y_t)}$ is the one that on input x_i outputs y_i for. Our idea for constructing a disguisable symmetric encryption scheme is as follows: to encrypt y_1 with the key x_1, we choose $t-1$ more fake keys $x_2, \cdots, x_t$ and arbitrary $y_2, \cdots, y_t$ and let the obfuscation of $MBSF_{(x_1,y_1),(x_2,y_2),\cdots,(x_t,y_t)}$ be the cipher text. Thus the cipher text (also viewed as a program) on input x_i outputs y_i. This means the cipher text can be decrypted to many values. In this paper, we will formally illustrate and extend this basic idea as well as some necessary randomized techniques to construct a secure disguisable symmetric encryption scheme which can possess the required security.

1.3 Outline of This Paper

The rest of this paper is as follows. Section 2 presents the preliminaries. Section 3 presents our result, i.e. the definition and the construction of the disguisable symmetric encryption scheme as well as some discussion of how to securely store and manage keys for an attacker. Section 4 summarizes this paper.

2 Preliminaries

This section contains the notations and definitions used throughout this paper.

2.1 Basic Notions

A function $\mu(\cdot)$, where $\mu : \mathbb{N} \to [0,1]$ is called negligible if $\mu(n) = n^{-\omega(1)}$ (i.e., $\mu(n) < \frac{1}{p(n)}$ for all polynomial $p(\cdot)$ and large enough n's). We will sometimes use neg to denote an unspecified negligible function.

The shorthand "PPT" refers to probabilistic polynomial-time, and we denote by PPT machines non-uniform probabilistic polynomial-time algorithms unless stated explicitly.

We say that two probability ensembles $\{X_n\}_{n\in\mathbb{N}}$ and $\{Y_n\}_{n\in\mathbb{N}}$ are computationally indistinguishable if for every PPT algorithm A, it holds that $|\Pr[A(X_n)=1]-\Pr[A(Y_n)=1]| = \mathsf{neg}(n)$. We will sometimes abuse notation and say that the two random variables X_n and Y_n are computationally indistinguishable when each of them is a part of a probability ensemble such that these ensembles $\{X_n\}_{n\in\mathbb{N}}$ and $\{Y_n\}_{n\in\mathbb{N}}$ are computationally indistinguishable. We will also sometimes drop the index n from a random variable if it can be infer from the context. In most of these cases, the index n will be the security parameter.

2.2 Point Functions, Multi-bit Point and Set-Membership Functions

A point function, $PF_x : \{0,1\}^n \to \{0,1\}$, outputs 1 if and only if its input matches x, i.e., $PF_x(y) = 1$ iff $y = x$, and outputs 0 otherwise. A point function with multiple-bit output, $MBPF_{x,y} : \{0,1\}^n \to \{y, \perp\}$, outputs y if and only if its input matches x, i.e., $MBPF_{x,y}(z) = y$ iff $z = x$, and outputs $\perp$ otherwise. A multiple-bit set-membership function, $MBSF_{(x_1,y_1),\cdots,(x_t,y_t)} : \{0,1\}^n \to \{y_1, \cdots, y_t, \perp\}$ outputs y_i if and only if the input matches x_i and outputs $\perp$ otherwise, where t is at most a polynomial in n.

2.3 Obfuscation

Informally, an obfuscation of a program P is also a program that computes the same functionality as P but its code can hide all information beyond the functionality. That is, the obfuscated program is fully "unintelligent" and any adversary cannot understand nor reverse-engineering it. This paper adopts the definition of obfuscation proposed by [4][3][9].

Definition 1. *Let $\mathbb{F}$ be a family of functions. A uniform PPT O is called an obfuscator of $\mathbb{F}$, if:*

Approximate Functionality: *for any $F \in \mathbb{F}$, $\Pr[\exists x, O(F(x)) \neq F(x)]$ is negligible. Here the probability is taken over the coin tosses of O.*

Polynomial Slowdown: *There exists a polynomial p such that, for any $F \in \mathbb{F}$, $O(F)$ runs in time at most $p(T_F)$, where T_F is the worst-case running-time of F.*

Weak Virtual black-box property: *For every PPT distinguisher A and any polynomial p, there is an (non-uniform) PPT simulator S, such that for any $F \in \mathbb{F}$, $\Pr[A(O(F)) = 1] - \Pr[A(S^F(1^{|F|})) = 1] \leq \frac{1}{p(n)}$.*

The theoretical investigation of obfuscation was initialized by [1]. [4] presented a modular approach to construct an obfuscation for multiple-bit point and set-membership functions based on an obfuscation for point functions [3][5].

2.4 Symmetric Encryption

We recall the standard definitions of a symmetric (i.e. private-key) encryption scheme. We start by presenting the syntax definition as follows:

Definition 2. *(Symmetric encryption scheme). A symmetric or private-key encryption scheme* $\mathsf{SKE} = (G; E; D)$ *consists of three uniform PPT algorithms with the following semantics:*
1. The key generation algorithm G *samples a key* k*. We write* $k \leftarrow G(1^n)$ *where* n *is the security parameter.*
2. The encryption algorithm E *encrypts a message* $m \in \{0,1\}^{poly(n)}$ *and produces a cipher text* C*. We write* $C \leftarrow E(k; m)$*.*
3. The decryption algorithm D *decrypts a cipher text* C *to a message* m*. We write* $m \leftarrow D(k; C)$*. Usually, perfect correctness of the scheme is required, i.e., that* $D(k; E(k; m)) = m$ *for all* $m \in \{0,1\}^{poly(n)}$ *and all possible* k*.*

Security of encryption schemes. The standard security for encryption is computational indistinguishability, i.e., for any two different messages m_1, m_2 with equal bit length, their corresponding cipher texts are computationally indistinguishable.

3 Our Result

In this section we propose the definition and the construction of disguisable symmetric encryption schemes. As shown in Section 1.1, the two typical goals (or motivation) of this kind of encryption schemes is to either let the attacker disguise his malicious program as a benign program, or let the attacker disguise his malicious data as ordinary data.

Although we can present the definition of disguisable symmetric encryption schemes in a general sense without considering the goal it is intended to achieve, we still explicitly contain the goal in its definition to emphasis the motivation of such encryption schemes. In this section we illustrate the definition and construction with respect to the goal of disguising executable files in detail, and omit those counterparts with respect to the goal of disguising data files. Actually, the two definitions and constructions are same if we do not refer to the type of the underlying files.

In Section 3.1, we present the definition of disguisable symmetric encryption schemes and the security requirements. In Section 3.2 we present a construction of disguisable symmetric encryption schemes which can satisfy the required security requirements. In Section 3.3 we provide some discussion on how to securely store and manage the keys in practice.

3.1 Disguisable Symmetric Encryption

In this subsection we present the definition of secure disguisable symmetric encryption as follows.

Definition 3. *A disguisable symmetric encryption scheme* $\mathsf{DSKE} = (G; E; D)$ *(for encryption of executable files) consists of three uniform PPT algorithms with the following semantics:*

1. *The key generation algorithm G on input 1^n, where n is the security parameter, samples a real key k and several fake keys $\mathsf{FakeKey}_1, \cdots, \mathsf{FakeKey}_r$. (The fake keys are also inputs to the encryption algorithm.)*
2. *The encryption algorithm E on input k, an executable file $\mathsf{File} \in \{0,1\}^{poly(n)}$ to be encrypted, together with $\mathsf{FakeKey}_1, \cdots, \mathsf{FakeKey}_r$, produces a cipher text C.*
3. *The (deterministic) decryption algorithm D on input a key and a cipher text C (promised to be the encryption of the executable file File) outputs a plain text which value relies on the key. That is, if the key is k, D's output is File. If the key is any fake one generated previously, D's output is also an executable file other than File. Otherwise, D outputs $\perp$. We require computational correctness of the scheme. That is, for the random keys generated by G and E's internal coins, D works as required except negligible probability.*

We remark that in a different viewpoint, we can view that the very key used in encryption consists of k and all $\mathsf{FakeKey}_i$, and k, $\mathsf{FakeKey}_i$ can be named segments of this key. Thus in this viewpoint our definition essentially means that decryption operation only needs a segment of the key and behaves differently on input different segments of this key. However, since not all these segments are needed to perform correct decryption, i.e., there is no need for the users of such encryption schemes to remember all segments after performing the encryption, we still name k and all $\mathsf{FakeKey}_i$ keys in this paper. We only require computational correctness due to the obfuscation for MBSF functions underlying our construction which can only obtain computational approximate functionality (i.e., no PPT algorithm can output a x such that $O(F(x)) \neq F(x)$ with non-negligible probability).

Security of disguisable symmetric encryption schemes. We say DSKE is secure if the following conditions hold:

1. For any two different executable files $\mathsf{File}_1, \mathsf{File}_2$ with equal bit length, their corresponding cipher texts are computationally indistinguishable.
2. Assuming there is a public upper bound B on r known to everyone, any adversary on input a cipher text can correctly guess the value of r with probability no more than $\frac{1}{B} + \mathsf{neg}(n)$. (This means r should be uniform and independent of the cipher text.)
3. After the user hands over to the adversary $1 \leq r' \leq r$ fake key(s) and claims one of them is the real key and the remainders are fake keys (if $r' \geq 2$), the adversary cannot distinguish the cipher texts of $\mathsf{File}_1, \mathsf{File}_2$ either. Further,the conditional probability that the adversary can correctly guess the value of r is no more than $\frac{1}{B-r'} + \mathsf{neg}(n)$ if $r' < B$. (This means r is still uniform and independent of the cipher text on the occurrence that the adversary obtains the r' fake keys.)

We remark that the first requirement originates from the standard security of encryption, and that the second requirement basically says that the cipher text does not contain any information of r (beyond the public bound B), and that the third requirement says the requirements 1 and 2 still hold even if the adversary obtains some fake keys. In fact the second and third requirements are proposed for the anti-forensics purpose mentioned previously.

3.2 Construction of the Encryption Schemes

In this subsection we present a construction of the desired encryption scheme. Our scheme heavily depends on the current technique of obfuscating multiple-bit set-membership functions presented in [4]. The construction in [4] is modular based on the obfuscation for point functions. As shown by [4], this modularization construction is secure if the underlying obfuscation for point functions satisfies some composability. Actually, the known construction of obfuscation for point function in [3] when using the statistically indistinguishable perfectly one-way hash functions [5] satisfies such composability, which results in that the construction in [4] is a secure obfuscation with computational approximate functionality. We will not review the definitions and constructions of the obfuscation and perfectly one-way hash functions in [5][3] and several composability discussed in [4] here, and refer the readers to the original literature.

We first present a naive scheme in Construction 1 which can illustrate the basic idea how to construct a multiple-bit set-membership function to realize a disguisable symmetric encryption. But the drawback of this scheme is that it cannot possess the desired security. Then we present the final scheme in Construction 2 which can achieve the requirements of secure disguisable encryption schemes.

Construction 1: We construct a naive scheme $\mathsf{DSKE}' = (G; E; D)$ as follows:

1. G: on input 1^n, uniformly sample two n-bit strings independently from $\{0,1\}^n$, denoted k and $\mathsf{FakeKey}$ (note $\Pr[k = \mathsf{FakeKey}] = 2^{-n}$). k is the real symmetric key and $\mathsf{FakeKey}$ is the fake key. (r is 1 herein.)
2. E: on input $k, \mathsf{FakeKey}$ and an executable file $\mathsf{File} \in \{0,1\}^t$, perform the following computation:
   (a) Choose a fixed existing different executable file in the hard disk with bit length t (if its length is less than t, pad some dummy instructions to it to satisfy this requirement), denoted $\mathsf{FakeFile}$, and then compute the following program P.

```
P's description:
input: $x$
1. in the case $x = k$, return File;
2. in the case $x =$ FakeKey, return FakeFile;
3. return $\perp$;
4. end.
```

   (b) Generate a program Q for P. (It differs from the obfuscation in [4] in that it does not use a random permutation on two blocks of Q, i.e. lines 1-3 and lines 4-6.)

That is, let y denote File and y_i denote the ith bit of y. For each i, if $y_i = 1$ E computes a program U_i as an obfuscation of PF_k (point function defined in Section 2.2), using the construction in [3] employing the statistically indistinguishable perfectly one-way hash functions in [5], otherwise E computes U_i as an obfuscation of PF_u where u is a uniformly random n-bit string. Generate a more program U_0 as an obfuscation of PF_k.

Similarly, E adopts the same method to compute t obfuscation according to each bit of $\mathsf{FakeFile}$. Denote by FakeU_i these t obfuscation, $1 \le i \le t$. Generate a more program FakeU_0 as an obfuscation of PF_{FakeKey}.

Q's description:

```
input: $x$
1. in the case $U_0(x) = 1$
2.     for $i = 1$ to $t$ let $y_i \leftarrow U_i(x)$;
3.     return $y$.
4. in the case $\mathsf{FakeU}_0(x) = 1$
5.     for $i = 1$ to $t$ let $y_i \leftarrow \mathsf{FakeU}_i(x)$;
6.     return $y$;
7. return $\perp$.
8. end
```

Q is the cipher text.

3. D: on input a cipher text c and a key key, it views c as a program and executes $c(\mathsf{key})$ to output what c outputs as the corresponding plain text.

It can be seen that P actually computes a multiple-bit set-membership function, defined in Section 2.2, and $Q \leftarrow E(k, \mathsf{File}, \mathsf{FakeKey})$ possesses the computational approximate functionality with P. Thus, except negligible probability, for any File that an attacker wants to encrypt, we have that $D(k, Q) = \mathsf{File}$, $D(\mathsf{Fakekey}, Q) = \mathsf{FakeFile}$. This shows that Definition 3 of disguisable symmetric encryption schemes is satisfied by DSKE'.

Now the next step is to verify if this encryption is secure with respect to the security of the disguisable symmetric encryption schemes. That is, we need to verify if the security requirements are satisfied. However, as we will point out, DSKE' is actually insecure with respect to the security requirements. First, since Q is not a secure obfuscation of P, we cannot establish the indistinguishability of encryption. Second, the secrecy of r cannot be satisfied. Instead, r is fixed as 1 herein. Thus if the forensics investigator knows the attacker adopts DSKE' to encrypt a malicious program and orders the attacker to hand over the two keys, the attacker may choose either to provide both k, $\mathsf{FakeKey}$ or to provide $\mathsf{FakeKey}$ (the attacker claims he only remembers one of the two keys) to the investigator. In the former case, the forensics investigator can immediately grasp the malicious program as well as another fake program. Notice that the execution traces of the two decryptions are not same, i.e. the decryption using the real key always occurs in Lines 2 and 3 of Q, while the one using the fake key occurs in Lines 5 and 6.

Thus the investigator can tell the real malicious program from the other one. In the latter case, the investigator can still judge if the attacker tells him the real key by checking the execution trace of Q. To achieve the security requirements, we should overcome the drawbacks of distinguishability of encryption, exposure of r and execution trace of Q, as the following shows.

We improve the naive scheme by randomizing r over some interval $[1, B]$ for a public constant B and adopt the secure obfuscation for multiple-bit set-membership functions in [4] etc. The construction of the desired encryption scheme is as follows.

Construction 2: The desired encryption scheme $\mathsf{DSKE} = (G; E; D)$ is as follows:

1. G: on input 1^n, uniformly sample $r+1$ n-bit strings independently from $\{0,1\}^n$, denoted k and $\mathsf{FakeKey}_i$ for $1 \leq i \leq r$. k is the real symmetric key and $\mathsf{FakeKey}_i$ for each i is a fake key.
2. E: on input the secret key k, $\mathsf{FakeKey}_1, \cdots, \mathsf{FakeKey}_r$ and an executable file $\mathsf{File} \in \{0,1\}^t$, perform the following computation:
   (a) Choose a fixed existing executable file with bit length t in the hard disk, denoted File'. Let $u_0, \cdots, u_r$ denote $k, \mathsf{FakeKey}_1, \cdots, \mathsf{FakeKey}_r$. Then uniformly and independently choose $B-r$ more strings from $\{0,1\}^n$, denoted $u_{r+1}, \cdots, u_B$ (the probability that at least two elements in $\{u_0, \cdots, u_B\}$ are identical is only $\mathsf{neg}(n)$). Construct two $(B+1)$-cell tables K' and F' satisfying $K'[i] = u_i$ for $0 \leq i \leq B$ and $F'[0] = \mathsf{File}$ and $F'[i] = \mathsf{File}'$ for $1 \leq i \leq Q$.

   (b) Generate the following program P, which has the tables K', F' hardwired.

```
input: $x$
1. for $i = 0$ to $B$ do the following
2.     if $x = K'[i]$, return $F'[i]$;
3. return $\perp$;
4. end.
```

   (c) Adopt the method presented in [4] to obfuscate P.
   That is, choose a random permutation π from $[0, B]$ to itself and let $K[i] = K'[\pi(i)]$ and $F[i] = F'[\pi(i)]$ for all i's. Then obfuscate the multiple-bit point functions $MBPF_{K[i],F[i]}$ for all i's. More concretely, let y_i denote $F[i]$ and $y_{i,j}$ denote the jth bit of y_i. For each j, if $y_{i,j} = 1$ E generates a program $U_{i,j}$ as an obfuscation of $PF_{K[i]}$ (point function), using the construction in [3] employing the statistically indistinguishable perfectly one-way hash functions in [5], otherwise E generates $U_{i,j}$ as an obfuscation of PF_u where u is a uniformly random n-bit string. Generate a more program $U_{i,0}$ as an obfuscation of $PF_{K[i]}$.

   Generate the following program Q, which is an obfuscation of P:

```
input: $x$
```

```
1. for $i = 0$ to $B$ do the following
2.     if $U_{i,0}(x) = 1$
3.         for $j = 1$ to $t$, let $y_{i,j} \leftarrow U_{i,j}(x)$;
4.         return $y_{i,j}$;
5. return $\perp$;
6. end.
```

Q is the cipher text.

3. D: on input a cipher text c and a key key, it views c as a program and executes $c(\mathsf{key})$ to output what c outputs as the corresponding plain text.

Since it is not hard to see that DSKE satisfies Definition 3, we now turn to show that DSKE can achieve the desired security requirements, as the following claims state.

Claim 2. *DSKE satisfies the computational indistinguishability of encryption.*

Proof. This claim follows from the result in [4] which ensures that Q is indeed an obfuscation of P. To prove this claim we need to show that for arbitrary two files f_1 and f_2 with equal bit length, letting Q_1 and Q_2 denote their cipher texts respectively generated by DSKE, Q_1 and Q_2 are indistinguishable. Formally, we need to show that for any PPT distinguisher A and any polynomial p, $|\Pr[A(Q_1) = 1] - \Pr[A(Q_2) = 1]| \leq \frac{1}{p(n)}$.

Let P_1 (resp. P_2) denote the intermediate program generated by the encryption algorithm in encrypting f_1 (resp. f_2) in step (b). Since Q_1 (resp. Q_2) is an obfuscation of P_1 (resp. P_2), by Definition 1 we have that for the polynomial $3p$ there exists a simulator S satisfying $|\Pr[A(Q_i) = 1] - \Pr[A(S^{P_i}(1^{|P_i|}) = 1]| \leq \frac{1}{3p(n)}$ for $i = 1, 2$.

As $|\Pr[A(Q_1) = 1] - \Pr[A(Q_2) = 1]| \leq |\Pr[A(Q_1) = 1] - \Pr[A(S^{P_1}(1^{|P_1|})) = 1]| + |\Pr[A(Q_2) = 1] - \Pr[A(S^{P_2}(1^{|P_2|})) = 1]| + |\Pr[A(S^{P_1}(1^{|P_1|})) = 1] - \Pr[A(S^{P_2}(1^{|P_2|})) = 1]|$, to show $|\Pr[A(Q_1) = 1] - \Pr[A(Q_2) = 1]| \leq \frac{1}{p(n)}$, it suffices to show $|\Pr[A(S^{P_1}(1^{|P_1|})) = 1] - \Pr[A(S^{P_2}(1^{|P_2|})) = 1]| = \mathsf{neg}(n)$.

Let bad_1 (resp. bad_2) denote the event that in the computation of $A(S^{P_1}(1^{|P_1|}))$ (resp. $A(S^{P_2}(1^{|P_2|})))$, S queries the oracle with an arbitrary one of the $B + 1$ keys stored in table K.

It can be seen that on the occurrence of $\neg\mathsf{bad}_i$, the oracle P_i always responds $\perp$ to S in the respective computation for $i = 1, 2$. This results in that $\Pr[A(S^{P_1}) = 1|\neg\mathsf{bad}_1] = \Pr[A(S^{P_2}) = 1|\neg\mathsf{bad}_2]$. Further, since the $r + 1$ keys in each computation are chosen uniformly, the probability that at least one of S's queries to its oracle equals one of the keys is $O(\frac{\text{poly}(n)}{2^n})$, which is a negligible quantity, since S at most proposes polynomial queries. This means $\Pr[\mathsf{bad}_i] = \mathsf{neg}(n)$ for $i = 1, 2$.

Since $\Pr[\neg\mathsf{bad}_i] = 1 - \mathsf{neg}(n)$, $\Pr[A(S^{P_i}) = 1|\neg\mathsf{bad}_i] = \frac{\Pr[A(S^{P_i})=1,\neg\mathsf{bad}_i]}{\Pr[\neg\mathsf{bad}_i]} = \Pr[A(S^{P_i}) = 1] + \mathsf{neg}(n)$ or $\Pr[A(S^{P_i}) = 1] - \mathsf{neg}(n)$. Thus we have $|\Pr[A(S^{P_1}) = 1] - \Pr[A(S^{P_2}) = 1]| = \mathsf{neg}(n)$. So this claim follows as previously stated. □

Now we need to show that any adversary on input a cipher text can hardly obtain some information of r (beyond the public bound B).

Claim 3. *For any PPT adversary A, A on input a cipher text Q can correctly guess r with probability no more than $\frac{1}{B} + \mathsf{neg}(n)$.*

Proof. Since A's goal is to guess r (which was determined at the moment of generating Q), we can w.l.o.g. assume A's output is in $[1, B] \cup \{\perp\}$, where $\perp$ denotes the case that A outputs a value which is outside $[1, B]$ and thus viewed meaningless.

Then, we construct B PPT algorithms $A_1, \cdots, A_B$ with the following descriptions: A_i on input Q executes $A(Q)$ and finally outputs 1 if A outputs i and outputs 0 otherwise, $1 \leq i \leq B$. It can be seen each A_i can be viewed as a distinguisher and thus for any polynomial p there is a simulator S_i for A_i satisfying that $|\Pr[A_i(Q) = 1] - \Pr[A_i(S_i^P(1^{|P|})) = 1]| \leq \frac{1}{p(n)}$. Namely, $|\Pr[A(Q) = i] - \Pr[A(S_i^P(1^{|P|})) = i]| \leq \frac{1}{p(n)}$ for each i. Thus for random r, $|\Pr[A(Q) = r] - \Pr[A(S_r^P(1^{|P|})) = r]| \leq \frac{1}{p(n)}$.

Let good_i denote the event that S_i does not query its oracle with any one of the $r+1$ keys for each i. On the occurrence of good_i, the oracle P always responds $\perp$ to S_i and thus the computation of $A(S_i^P)$ is independent of the $r+1$ keys hidden in P. For the same reasons stated in the previous proof, $\Pr[A(S_i^P) = r|\mathsf{good}_i] = \frac{1}{B}$ and $\Pr[\mathsf{good}_i] = 1 - \mathsf{neg}(n)$. Thus it can be concluded $\Pr[A(S_i^P) = r] \leq \frac{1}{B} + \mathsf{neg}(n)$ for all i's. Thus for random r, $\Pr[A(S_r^P) = r] \leq \frac{1}{B} + \mathsf{neg}(n)$. Hence combining this with the result in the previous paragraph, we have for any p $\Pr[A(Q) = r] \leq \frac{1}{B} + \mathsf{neg}(n) + \frac{1}{p(n)}$. Thus $\Pr[A(Q) = r] \leq \frac{1}{B} + \mathsf{neg}(n)$. □

When the attacker is catched by the forensics investigator, and ordered to hand over the real key and all fake keys, he is supposed to provide r' fake keys and tries to convince the investigator that what he encrypted is an ordinary executable file. After obtaining these r' keys, the forensics investigator can verify if these keys are valid. Since Q outputs $\perp$ on input any other strings, we can assume that the attacker always hands over the valid fake keys, or else the investigator will no end the inquest until the r' keys the attacker provides are valid. Then we turn to show that the cipher texts of two plain texts with equal bit length are still indistinguishable.

Claim 4. *DSKE satisfies the computational indistinguishability of encryption, even if the adversary obtains $1 \leq r' \leq r$ valid fake keys.*

Proof. Assume an arbitrary A obtains a cipher text Q (Q_1 or Q_2) and r' fake keys. Since on input the r' fake keys as well as their decryptions and the subprogram in Q which consists of the obfuscated multi-bit point functions corresponding to those unexposed keys, denoted Q' (Q'_1 or Q'_2), A can generate a cipher text which is identically distributed to Q, it suffices to show that for any outcome of the r' fake keys as well as their decryptions, A', which is A with them hardwired, cannot tell Q'_1 from Q'_2. Notice that Q' is also an obfuscated multi-bit set-membership function. Then adopting the analogous method in the proof of Claim 2, we have for any polynomial p, $|\Pr[A'(Q'_1) = 1] - \Pr[A'(Q'_2) = 1]| \leq \frac{1}{p(n)}$. Details omitted. □

Lastly, we need to show that after the adversary obtains $1 \leq r' \leq r$ valid fake keys where $r' < B$, it can correctly guess r with probability nearly $\frac{1}{B-r'}$, as the following claim states.

Claim 5. *For any PPT adversary A, A on input a cipher text Q can correctly guess r with probability no more than $\frac{1}{B-r'} + \mathsf{neg}(n)$ on the occurrence that the adversary obtains $1 \leq r' \leq r$ valid fake keys for $r' < B$.*

Proof. The proof is almost the same as the one of Claim 3. Notice that there are $B - r'$ possible values left for r and for any outcome of the r' fake keys and their decryptions, A with them hardwired can be also viewed as an adversary, and Q' (referred to the previous proof) is an obfuscated multi-bit set-membership function. The remainder proof is analogous. □

Thus, we have shown that DSKE satisfies all the required security requirements of disguisable symmetric encryption.

Since for an encryption scheme all security is lost if the key is lost, to put it into practice we need to discuss the issue of securely storing and management of these keys, which will be shown in the next subsection.

3.3 Management of the Keys

Since all the keys are generated at random, these keys cannot be remembered by human's mind. Actually, by the requirement of the underlying obfuscation method presented in [4], the min-entropy of a key should be at least super-logarithmic and the available construction in [5] requires the min-entropy should be at least n^{ε}. By the algorithm of key generation in our scheme, this requirement can be satisfied.

If an attacker has the strong ability to remember the random keys with n^{ε} min-entropy, he can view these keys as the rememberable passwords and keeps all keys in his mind. Thus there is no need for him to store the keys (passwords) and manage them. But actually, we think that it is still hard for human's mind to remember several random strings with such min-entropy. On the other hand, keys or passwords generated by human's mind are of course not random enough and thus cannot ensure the security of the encryption schemes.

The above discussion shows that a secure management of keys should be introduced for attackers. The first attempt towards this goal is to store each key into a file and the attacker remembers the names of these files. When he needs to use the real key, he retrieves it from some file and then executes the encryption or decryption. When the encryption or decryption operation finishes, he should wipe all the information in the hard disk which records the read/write operation of this file. However, this attempt cannot eliminate the risk that the forensics investigator can scan the hard disk to gather all these files and obtain all keys.

Another solution is to use the obfuscation for multiple-bit set-membership functions one more time, as the construction 2 illustrates. That is, the attacker can arbitrarily choose r human-made passwords which can be easily remembered by himself. Let each password correspond to a key (the real one or a fake one). Then he constructs a program PWD which on each password outputs the corresponding key. It can be seen that the program PWD also computes a multiple-bit set-membership function, similar to the program P in Construction 2. Then obfuscate PWD using the similar way.

However, it should be emphasized that to achieve the theoretical security guarantee by this obfuscation the passwords should be random with min-entropy n^{ε}. In general the human-made rememberable ones cannot satisfy this requirement, or else we could directly replace the keys in Construction 2 by these passwords. So this solution only has a heuristic security guarantee that no forensics investigator can reverse-engineering nor understand PWD even if he obtains all its code.

The third solution is to let the attacker store the keys into a hardware device. However, we think putting all keys in a device is quite insecure since if the attacker is catched and ordered to hand over keys, he has to hand over this device and thus all the keys may expose to the investigator.

Actually, we think that it is the two assumptions that result in that we cannot provide a solution with a theoretical security guarantee. The two assumptions are that the human's mind cannot remember random strings with min-entropy n^{ε} and that the forensics investigator can always gather any file he desires from the attacker's machine or related devices. Thus to find a scheme for secure management of keys with a theoretical guarantee, we maybe need to relax at least one of the assumptions.

We suggest a solution by adopting such relaxation. Our relaxation is that we assume that the attacker has the ability to store at least a random string with such min-entropy in a secure way. For instance, this secure way may be to divide the string into several segments and store the different segments in his mind, the secret place in the hard disk and other auxiliary secure devices respectively. Under this assumption, the attacker can store the real key in this secure way and store some fake keys in different secret places in the hard disk using one or many solutions presented above or combining different solutions in storing these fake keys.

4 Conclusions

We now summarize our result as follows. To apply the disguisable symmetric encryption scheme, an attacker needs to perform the following ordered operations.

First, he runs the key generation algorithm to obtain a real key and several fake keys according to Construction 2. Second, he adopts a secure way to store the real key as well as storing some fake keys in his hard disk. Third, erase all possible information generated in the first and second steps. Fourth, prepare a benign executable file which is of the same length with the malicious program (resp. the data file) he wants to encrypt. Fifth, the attacker can encrypt the malicious program (resp. the data file) if needed. By Construction 2, the encryption is secure, i.e. indistinguishable.

If the attacker is catched by the forensics investigator and ordered to hand over keys to decrypt the cipher text of the malicious program (resp. the data file), he provides several fake keys to the investigator and claims that one of them is the real key and others are fake. Since all decryption are valid and the investigator has no idea of the number of the keys, the investigator cannot distinguish if the attacker lies to him.

References

1. Barak, B., Goldreich, O., Impagliazzo, R., Rudich, S., Sahai, A., Vadhan, S.P., Yang, K.: On the (Im)possibility of obfuscating programs. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 1–18. Springer, Heidelberg (2001)
2. Berghel, H.: Hiding Data, Forensics, and Anti-forensics. Commun. ACM 50(4), 15–20 (2007)
3. Canetti, R.: Towards realizing random oracles: Hash functions that hide all partial information. In: Kaliski Jr., B.S. (ed.) CRYPTO 1997. LNCS, vol. 1294, pp. 455–469. Springer, Heidelberg (1997)
4. Canetti, R., Dakdouk, R.R.: Obfuscating point functions with multibit output. In: Smart, N.P. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 489–508. Springer, Heidelberg (2008)
5. Canetti, R., Micciancio, D., Reingold, O.: Perfectly One-way Probabilistic Hash Functions. In: The 30th ACM Symposium on Theory of Computing, pp. 131–140. ACM, New York (1998)
6. Garfinkel, S.: Anti-forensics: Techniques, Detection and Countermeasures. In: The 2nd International Conference on i-Warfare and Security (ICIW), ACI, pp. 8–9 (2007)
7. Cabrera, J.B.D., Lewis, L., Mehara, R.: Detection and Classification of Intrusion and Faults Using Sequences of System Calls. ACM SIGMOD Record 30, 25–34 (2001)
8. Mohay, G.M., Anderson, A., Collie, B., McKemmish, R.D., de Vel, O.: Computer and Intrusion Forensics. Artech House, Inc., Norwood (2003)
9. Wee, H.: On Obfuscating Point Functions. In: The 37th ACM Symposium on Theory of Computing, pp. 523–532. ACM, New York (2005)