Securely Obfuscating Re-Encryption

Susan Hohenberger[1,2], Guy N. Rothblum[3⋆],
abhi shelat[2], and Vinod Vaikuntanathan[3⋆⋆]

[1] Johns Hopkins University (susan@cs.jhu.edu)
[2] IBM Zurich Research (sus,abs@zurich.ibm.com)
[3] MIT CSAIL (rothblum,vinodv@mit.edu)

Abstract. We present the first positive obfuscation result for a traditional cryptographic functionality. This positive result stands in contrast to well-known negative impossibility results [BGI+01] for *general obfuscation* and recent negative impossibility and improbability [GK05] results for obfuscation of many cryptographic functionalities.

Whereas other positive obfuscation results in the standard model apply to very simple point functions, our obfuscation result applies to the significantly more complicated and widely-used *re-encryption functionality*. This functionality takes a ciphertext for message m encrypted under Alice's public key and transforms it into a ciphertext for the same message m under Bob's public key.

To overcome impossibility results and to make our results meaningful for cryptographic functionalities, we use a new definition of obfuscation. This new definition incorporates more security-aware provisions.

1 Introduction

A recent line of research in theoretical cryptography aims to understand whether it is possible to *obfuscate* programs so that a program's code becomes unintelligible while its functionality remains unchanged. A general method for obfuscating programs would lead to the solution of many open problems in cryptography.

Unfortunately, Barak, Goldreich, Impagliazzo, Rudich, Sahai, Vadhan and Yang [BGI+01] show that for many notions of obfuscation, a general program obfuscator does not exist—i.e., they exhibit a class of circuits which cannot be obfuscated. A subsequent work of Goldwasser and Kalai [GK05] shows the impossibility and improbability of obfuscating more natural functionalities.

In spite of these negative results for general-purpose obfuscation, there are a few positive obfuscation results for simple functionalities such as point functions. A point function I_x returns 1 on input x and 0 on all other inputs. Canetti [Can97] shows that under a very strong Diffie-Hellman assumption point functions can be obfuscated. Further work of Canetti, Micciancio and Reingold [CMR98], Wee [Wee05] and Dodis and Smith [DS05] relaxes the assumptions

⋆ Research supported by NSF grant CNS-0430450 and NSF grant CFF-0635297.
⋆⋆ Research supported by NSF grant CNS-0430450.

required for obfuscation and considers other (related) functionalities. Despite these positive results, obfuscators for traditional cryptographic functionalities (such as those that deal with encryption) have remained elusive.

Our Results. In this work, we present the first obfuscator for a more traditional cryptographic functionality. Namely, we show that:

Main Theorem 1 (Informal) *Under reasonable bilinear complexity assumptions, there exists an efficient program obfuscator for a family of circuits implementing re-encryption.*

A *re-encryption program* for Alice and Bob takes a ciphertext for a message m encrypted under Alice's public key, and transforms it into a ciphertext for the same message m under Bob's public key. Re-encryption programs have many practical applications such as the iTunes DRM system (albeit, with symmetric keys [Smi05]), secure distributed file servers [AFGH06] and secure email forwarding.

The straightforward method to implement re-encryption is to write a program P which decrypts the input ciphertext using Alice's secret key and then encrypts the resulting message with Bob's public key. When P is run by Alice, this is a good solution.

In the practical applications noted above, however, the re-encryption program is executed by a third-party. When this is the case, the straightforward implementation has serious security problems since P's code may reveal Alice's secret key to the third party. A better solution is to design an obfuscator for the re-encryption program P. That is, we would like that:

> *A third party who has a re-encryption program* learns *no more from the re-encryption program than it does from interaction with a black-box oracle that provides the same functionality.*

As we discuss later in §1.2, several re-encryption schemes have been proposed before [BS97,BBS98,DI03,AFGH06], but none of these prior works satisfy the strong obfuscation requirement informally stated above. Our main technical contribution is the construction of a novel re-encryption scheme which meets this strong notion while remaining surprisingly practical. As a side note, in our construction, ciphertexts that are re-encrypted from Alice to Bob cannot be further re-encrypted from Bob to Carol. This may be a limitation in some scenarios, but it is nonetheless sufficient for the important practical applications noted above.

Our main conceptual contribution is a definition of obfuscation that both sidesteps impossibility results by considering *randomized* functionalities and is more meaningful for cryptographic applications than previous definitions of obfuscation. Let us briefly explain.

1.1 Notion of Secure Obfuscation

The work of [BGI$^+$01] views an obfuscator as a compiler which takes a program (i.e., boolean circuit) P and turns it into an equivalent program that satisfies the *predicate black-box* property: any *predicate* that is computable from the obfuscated program should also be computable from black-box access to the program (see Definition 1).

Secure Obfuscation. Unfortunately, the predicate definition [BGI+01] and subsequent work does not provide a meaningful security guarantee when the obfuscated program is used as part of a larger cryptographic system. Intuitively, while the predicate black-box property gives a quantifiable guarantee that *some* information (namely, predicates) about the program is hidden by the obfuscated circuit, other "non-black-box information" may still leak. Moreover, this leaked information might compromise the security of a cryptographic scheme which uses the obfuscated circuit. For instance, it is completely possible that an obfuscated program for "delegating signatures" both meets the predicate black-box definition and is unforgeable under black-box access to a signature oracle, yet allows an adversary who has the obfuscated program code to forge a signature!

Since many potential applications of obfuscation use obfuscated circuits in larger cryptographic schemes, the definition of obfuscation *should* guarantee that the security of cryptographic schemes is preserved in the following sense.

> *If a cryptographic scheme is "secure" when the adversary is given black-box access to a program, then it remains "secure" when the adversary is given the obfuscated program.*

The most important feature of our new definition of obfuscation is that it preserves security in the above sense, and thus we refer to it as *secure obfuscation.* Informally, our definition requires that if there exists a *non black-box adversary* with access to an obfuscated program who can break the security of a cryptographic scheme, then there exists a *black-box simulator* which breaks the scheme with similar probability using only black-box access to the program. Thus, if the scheme is secure against black-box adversaries, then it is also secure against adversaries with access to obfuscated programs. The definition we give in this work gives the above guarantee for any cryptographic scheme with a *distinguishable attack* property; any scheme where a distinguisher with public information and black-box access to the obfuscated functionality can distinguish whether or not an attacker has broken the scheme. Semantically secure encryption and re-encryption are examples of such schemes.

This new definition of obfuscation can play an important role in the design of cryptographic schemes that use obfuscation. With secure obfuscation, the design of such schemes proceeds in two stages:

1. Specify the functionality of a program (or program family), and prove security of the cryptographic scheme against an adversary given *black-box* access to the program.
2. Construct a secure obfuscator for the program (or program family).

The combination of these two steps guarantees security of the scheme against an adversary that has full access to the obfuscated program. Indeed, for our scheme, we show step (1) in Theorem 3 and step (2) in Theorem 4.

Average-Case Obfuscation. Our new definition only requires obfuscation for a *random* circuit in a family of circuits (as in [Can97,CMR98,Had00,DS05,GK05]). This relaxed requirement remains meaningful for the many cryptographic applications of obfuscation in which the circuit to be obfuscated *is* chosen at random. Normally the random choice of a circuit corresponds to the random selection of cryptographic keys. We call the new definition of security *average-case secure obfuscation*. Combining more security-aware provisions (i.e. giving a distinguisher oracle access to the functionality) with this average-case relaxation was originally suggested by Pass [Pas06].

Obfuscating Probabilistic Circuits. It is not hard to see that *deterministic* functionalities that are not learnable cannot be obfuscated under our definition (see §2.2). In fact, security-preserving definitions considered in previous works ([BGI$^+$01,Wee05]) are only achievable for learnable deterministic functions. An important conceptual contribution of our definition and of this work is showing that these impossibility results disappear when considering obfuscation of *probabilistic* circuits. Furthermore, obfuscation of probabilistic circuits is important because most interesting cryptographic functionalities are probabilistic.

Other work on Obfuscation. Recently, Ostrovsky and Skeith [OS05] consider a different notion of *public-key obfuscation* focused on keyword search. A public-key obfuscator does not maintain the functionality of a program, but rather ensures that the outputs of a public-key obfuscated program are *encryptions* of the original program's outputs. Adida and Wikström [AW05] use a variation of this definition for public mixing. Both of these works differ from our notion of obfuscation in that our notion preserves functionality and explicitly considers black-box versus non-black-box access to a program.

1.2 The Obfuscated Re-Encryption Scheme

Comparison with Prior Work. Mambo and Okamoto [MO97] noted the popularity of re-encryption programs in practical applications and suggested efficiency improvements over the decrypt-and-encrypt approach. Blaze, Bleumer, and Strauss introduced the notion of *proxy re-encryption* [BS97,BBS98] in which the re-encryption program is executed by a third-party *proxy*. In their security notion, the proxy cannot read the messages of either Alice or Bob. The Blaze et al. construction is *bidirectional* (i.e., a program to translate ciphertexts from Alice to Bob can also be used to translate from Bob to Alice) and can be repeatedly applied (i.e., a ciphertext can be re-encrypted from Alice to Bob to Carol, etc.). Ateniese, Fu, Green, and Hohenberger [AFGH06] presented a semantic-security style definition for proxy re-encryption and designed the first *unidirectional* scheme, although their scheme can only be applied once. Ateniese et al. also built a secure distributed storage system using their algorithms.

While these prior works are secure under specialized definitions, they cannot be considered as obfuscations for re-encryption since they leak subtle non-black-box information. On the other hand, the re-encryption definitions of Ateniese et al. [AFGH06] provide some security guarantee with respect to *dependent* auxiliary inputs, which we will not consider in this work. For example, they show that even when Alice has the re-encryption program from Alice to Bob, Bob's semantic security still holds. (Although our definition does not require this, the scheme we present here satisfies this property.)

Overview of the Construction. We now provide intuition behind our construction of an obfuscator for re-encryption (see §4 for the full construction). In a series of attempts, we develop a cryptosystem and an obfuscated re-encryption program which translates ciphertexts under pk_1 to ciphertexts under pk_2. Our starting point is a suitable public key cryptosystem.

Recall the semantically-secure encryption scheme due to Boneh, Boyen, and Shacham [BBS04] as instantiated in a group $\mathbb{G}$ of order q equipped with a bilinear map. The keys in this scheme are generated by selecting a random $h \xleftarrow{r} \mathbb{G}$ and $a, b \xleftarrow{r} \mathbb{Z}_q$, and setting $sk = (a, b, h)$ and $pk = (h^a, h^b, h)$. To encrypt a message $m \in \mathbb{G}$, select two random values $r, s \xleftarrow{r} \mathbb{Z}_q$ and output the ciphertext $C = [h^{ar}, h^{bs}, h^{r+s} \cdot m]$. To decrypt a ciphertext $C = [W, X, Y]$, compute the plaintext $Y/(W^{1/a} \cdot X^{1/b})$. Let $pk_1 = (g^{a_1}, g^{b_1}, g)$ and $pk_2 = (h^{a_2}, h^{b_2}, h)$ be two public keys for this cryptosystem.

The basic (naive) re-encryption program from pk_1 to pk_2 contains (sk_1, pk_2). The program simply decrypts the input using sk_1 and encrypts the resulting message with pk_2. Clearly this program exposes both sk_1 and the underlying plaintext to any third-party executing the re-encryption program.

As a first attempt to obfuscate the basic program, consider the re-encryption program that contains $Z_1 = a_2/a_1$ and $Z_2 = b_2/b_1$ and re-encrypts the ciphertext $[W, X, Y]$ by computing $[W^{Z_1}, X^{Z_2}, Y]$ for pk_2. (On a different cryptosystem, a similar approach was suggested by Blaze et al. [BBS98].) Unfortunately, this re-encryption program leaks non-black-box information (i.e., does not satisfy the virtual black-box property in Def. 2). For example, the program containing (Z_1, Z_2) which translates ciphertexts from Alice to Bob can be transformed into a new program containing (Z_1^{-1}, Z_2^{-1}) which translates ciphertexts from Bob to Alice—a feat which black-box access does not allow.

As a second attempt, consider the re-encryption program containing $Z_1 = h^{a_2/a_1}$ and $Z_2 = h^{b_2/b_1}$. Alice, with $sk_1 = (a_1, b_1, g)$, can compute this program given Bob's public key $pk_2 = (h^{a_2}, h^{b_2}, h)$. (On a different cryptosystem, a similar approach was suggested by Ateniese et al. [AFGH06].) The re-encryption program works as follows: on input a ciphertext $[W, X, Y] = [g^{a_1 r}, g^{b_1 s}, g^{r+s} \cdot m]$ under pk_1, output the ciphertext $[\mathbf{e}(W, Z_1), \mathbf{e}(X, Z_2), \mathbf{e}(Y, h)] = [E, F, G]$ under pk_2. To decrypt $[E, F, G]$, the holder of sk_2 would first compute $Q = G/(E^{1/a_2} \cdot F^{1/b_2})$ and then find and output the message m_i in the message space M such that $\mathbf{e}(m_i, h) = Q$. Of course, to ensure efficient decryption, this limits the size of the message space M to be a polynomial. Notice the encryption scheme now support two "forms" of ciphertexts—an original form and a re-encrypted one, each containing elements from different groups. As a result, a re-encrypted ciphertext cannot be further re-encrypted. The question, though, is whether or not such a program is any closer to being an obfuscation.

To be a secure obfuscation according to our Def. 2, the output of an adversary who is given the obfuscated program must be indistinguishable—even to a distinguisher with oracle access to the re-encryption program—from the output of a simulator given only black-box access to the program. Unfortunately, in the second attempt, knowledge of the public keys $pk_1 = (g^{a_1}, g^{b_1}, g)$ and $pk_2 = (h^{a_2}, h^{b_2}, h)$ easily allows a distinguisher to test whether a program containing (Z_1, Z_2) is a valid re-encryption program for these keys by checking that $\mathbf{e}(g^{a_1}, Z_1) = \mathbf{e}(g, h^{a_2})$ and $\mathbf{e}(g^{b_1}, Z_2) = \mathbf{e}(g, h^{b_2})$. We do not know how to construct a simulator that can output a program which also passes this test.

To bypass this problem, we design our re-encryption program to be a probabilistic function of the keys. More specifically, consider the program containing $(y^{a_2/a_1}, y^{b_2/b_1}, y) = (Z_1, Z_2, Z_3)$ for a randomly selected $y \in \mathbb{G}$. (In the context of point functions, a similar approach was suggested by Canetti [Can97].) Alice can still generate this re-encryption program using only Bob's public key. The re-encryption program becomes: on input $[W, X, Y] = [g^{a_1 r}, g^{b_1 s}, g^{r+s} \cdot m]$ under pk_1, output the ciphertext under pk_2 as $[\mathbf{e}(W, Z_1), \mathbf{e}(X, Z_2), \mathbf{e}(Y, Z_3), Z_3] = [E, F, G, H]$. Decryption works as follows: first compute $Q = G/(E^{1/a_2} \cdot F^{1/b_2})$ and then output message m_i in the message space M such that $\mathbf{e}(m_i, H) = Q$.

This solution has one subtle problem because *all* ciphertexts produced by the obfuscated re-encryption program include $H = y$ as the fourth component, whereas ciphertexts produced by the decrypt-and-encrypt approach contain a fresh random value in that position. Thus, the obfuscated program does not "preserve the functionality" of the original one. This is easily fixed by having the obfuscated program re-randomize its output by choosing $z \xleftarrow{r} \mathbb{Z}_q$ and outputting $[E^z, F^z, G^z, H^z]$. (Note, it is not sufficient that we choose y randomly, since this choice is only made once for all re-encrypted ciphertexts, whereas z is chosen freshly for each re-encryption.)

Even this, however, falls short, because we do not know how to prove this construction is secure. In particular, since the distinguisher has access to a re-encryption oracle, it can query the oracle on the values contained in the obfuscated program! Indeed, in the above scheme, there is a specific (complicated) property of valid obfuscated programs that a distinguisher can test for, and we do not know how to construct a simulator that also passes this test.

In order to overcome this final hurdle, our program re-randomizes the input ciphertext before applying the transformation above. If the public key is (g^a, g^b, g), and the input ciphertext is $C = [W, X, Y]$, our program re-randomizes C by sampling r', s' and computing the ciphertext $[W \cdot (g^a)^{r'}, X \cdot (g^b)^{s'}, Y \cdot g^{r'+s'}]$. Finally, we are able to show this construction meets our obfuscation definition under two reasonable complexity assumptions.

As a final point about our complexity assumptions, because our obfuscation definition only requires average-case obfuscation, we do not have to make the strong complexity assumptions necessary in the constructions of Canetti [Can97] and Wee [Wee05]. Thus, our scheme simultaneously meets a strong theoretical definition while retaining the sensibility associated with standard assumptions and efficient algorithms.

2 Definitions

Barak et al. [BGI+01] required that an obfuscator strip programs of non-black-box information. They formalized this by requiring that any predicate computable from the obfuscated program is also computable from black-box access to it. Goldwasser and Kalai [GK05] gave a stronger definition, guaranteeing security in the presence of (dependent) auxiliary input. A formal definition, which we call *predicate black-box obfuscation* (or predicate obfuscation, for short), follows.

For a family $\mathbf{C}$ of polynomial-size circuits, for a length parameter n let $\mathbf{C}_n$ be the circuits in $\mathbf{C}$ with input length n (i.e. $\mathbf{C} = \{\mathbf{C}_n\}$).

Definition 1 (Predicate Obfuscation [BGI+01,GK05]). *An efficient algorithm* Obf *is a* predicate obfuscator for the family $\mathbf{C} = \{\mathbf{C}_n\}$, *if it has the following properties:*

- *Preserving Functionality: There exists a negligible function* $neg(n)$, *s.t. for all input lengths* n, *for any* $C \in \mathbf{C}_n$*:*

$$\Pr[\exists x \in \{0,1\}^n : (\mathsf{Obf}(C))(x) \neq C(x)] \leq neg(n)$$

  The probability is taken over Obf*'s random coins.*
- *Polynomial Slowdown: There exists a polynomial* $p(n)$ *such that for sufficiently large input lengths* n, *for any* $C \in \mathbf{C}_n$, *the obfuscator* Obf *only enlarges* C *by a factor of* p: $|\mathsf{Obf}(C)| \leq p(|C|)$.
- *Predicate Virtual Black-box: For every polynomial sized adversary circuit* $\mathcal{A}$, *there exists a polynomial size simulator circuit* Sim *and a negligible function* $neg(n)$, *such that for every input length* n, *for every* $C \in \mathbf{C}_n$, *for every predicate* π, *for every auxiliary input* $z \in \{0,1\}^{q(n)}$*:*

$$\left|\Pr[\mathcal{A}(\mathsf{Obf}(C), z) = \pi(C, z)] - \Pr[\mathsf{Sim}^C(1^n, z) = \pi(C, z)]\right| \leq neg(n)$$

  The probability is over the coins of the adversary, the simulator and the obfuscator.

As discussed in §1, the predicate black-box definition does not guarantee *security* when obfuscated circuits are used in cryptographic settings. To address this, we introduce the new notion of *average-case secure obfuscation*:

Definition 2 (Average-Case Secure Obfuscation). *An efficient algorithm* Obf *that takes as input a (probabilistic) circuit and outputs a new (probabilistic) circuit, is an* average-case secure obfuscator *for the family* $\mathbf{C} = \{\mathbf{C}_n\}$, *if it satisfies the following properties:*

- *Preserving Functionality: "With overwhelming probability* $\mathsf{Obf}(C)$ *behaves* almost *identically to* C *on all inputs". There exists a negligible function* $neg(n)$, *such that for any input length* n, *for any* $C \in \mathbf{C}_n$*:*

$$\Pr_{\text{coins of } \mathsf{Obf}}[\exists x \in \{0,1\}^n : \Delta\left((\mathsf{Obf}(C))(x), C(x)\right) \geq neg(n)] \leq neg(n)$$

The distributions $(\mathsf{Obf}(C))(x)$ *and* $C(x)$ *are taken over* $\mathsf{Obf}(C)$*'s and* C*'s random coins respectively.* Δ *denotes statistical* ($L1$) *distance between distributions.*

- *Polynomial Slowdown: (identical to Definition 1)*
- *Average-Case Secure Virtual Black-Box: For any efficient adversary* $\mathcal{A}$*, there exists an efficient simulator* Sim *and a negligible function* $neg(n)$*, such that for every efficient distinguisher* D*, for every input length* n *and for every polynomial-size auxiliary input* z*:*

$$\left| \begin{array}{l} \Pr[C \xleftarrow{r} \mathbf{C}_n : \ \mathsf{D}^C(\mathcal{A}(\mathsf{Obf}(C), z), z) = 1] \\ \ - \Pr[C \xleftarrow{r} \mathbf{C}_n : \ \mathsf{D}^C(\mathsf{Sim}^C(1^n, z), z) = 1] \end{array} \right| \leq neg(n)$$

The probability is over the selection of a random circuit C *from* $\mathbf{C}_n$*, and the coins of the distinguisher, the simulator, the oracle and the obfuscator. Note that entities with black-box access to* C *cannot set* C*'s random tape.*
Note that without loss of generality it is sufficient to require the existence of a simulator for the "dummy" adversary that just outputs its input. This would give an equivalent definition, but it loses some intuitive appeal.

Discussion. Intuitively, Definition 2 guarantees that any attack that a non-black box adversary can mount using the obfuscated circuit, can also be mounted by a black-box simulator with (oracle) access to the functionality. This new definition differs from the predicate definition in several ways. It considers obfuscation of a *random* circuit from a family, and furthermore, the circuit families considered can be *probabilistic* (this allows us to side-step impossibility results, see §2.2). We also follow [GK05] in requiring that the obfuscation be secure in the presence of (independent) auxiliary input, where the auxiliary input is selected first, and then a random circuit is chosen from the family. Note that the average-case secure virtual black-box requirement of our new definition is incomparable to the predicate black-box requirement of [BGI+01]; the latter is weaker in that it only requires that the obfuscator hides *predicates*, but is stronger in that it provides the predicate distinguisher with the actual program (whereas our definition only gives our predicate distinguisher black-box access).

Finally, we emphasize that in this new definition there are *two* important sources of randomness. The first source of randomness is in the circuits being obfuscated, which are probabilistic. The second, more subtle, source of randomness is in the selection of a random circuit C from the family $\mathbf{C}_n$. The average-case secure virtual black-box requirement guarantees security when a circuit is selected from the family by a *specific* distribution (i.e., the uniform distribution—one should think of this as uniformly choosing random keys for a cryptographic scheme). The predicate black-box definition, on the other hand, guarantees security for *every* circuit in the family, or (equivalently) for *every distribution* on circuits. Other work [CMR98,DS05] guarantees security for a large class of distributions on circuits from a family, such as *every* distribution with at least super-logarithmic min-entropy. Our notion of secure obfuscation can be generalized to give security against more general classes of distributions. For clarity, we choose to present the less general definition above.

2.1 Meaningfulness for Security

This section serves as an *informal* discussion of the security guarantee provided by average-case secure obfuscation. As mentioned in §1, the definition of obfuscation should be security-preserving in the following sense: *"If a cryptographic scheme is secure when the adversary is given black-box access to a program, then it remains secure when the adversary is given the obfuscated program."* We claim that for a large class of applications (including re-encryption), average-case secure obfuscation indeed gives this guarantee.

To see this, consider any cryptographic scheme, for which a *distinguisher*, that has only public information (e.g. public keys) and *black-box* access to an obfuscated program, can test whether a given adversary can break a scheme (we call this the *distinguishable attack* property). Many standard cryptographic schemes, such as semantically secure encryption and re-encryption, have this property. For such schemes Definition 2 indeed guarantees that for every adversary that mounts an attack using an obfuscated circuit, there exists a *black-box* simulator that can mount an attack with a similar success probability. Thus, if the scheme is secure against black-box adversaries, it is also secure against non black-box adversaries that are given the obfuscated program.

To illustrate the meaningfulness of the notion of average-case secure obfuscation, we propose to use the following informal argument as a methodology for constructing secure cryptographic schemes:

If a cryptographic scheme has the following three properties:

1. The scheme is secure against black-box adversaries with oracle access to functionality X selected randomly from a family F
2. A distinguisher D with oracle access to X can test whether an adversary $\mathcal{A}$ can break the security guarantee of the scheme (we call this property the *distinguishable attack* property)
3. There exists an average-case secure obfuscator for a family C_F of circuits implementing the functionalities in F,

Then the cryptographic scheme is also secure against adversaries who are given an obfuscation of a circuit selected at random from the family C_F.

As a case study, consider semantically-secure re-encryption (see Def. 3). An attacker is given two relevant public keys and black-box access to a re-encryption oracle. The attacker is successful if it can distinguish the encryptions of two different messages (of its choice) under one of the public keys. As with many cryptographic schemes, re-encryption schemes have Property (1). For Property (2), a distinguisher who is given public keys and oracle access to the re-encryption functionality can indeed *test* whether an adversary has a noticeable chance of mounting a successful attack.[4] Thus, for any re-encryption functionality, assuming that Property (3) holds (i.e. there exists an average-case secure obfuscator for some circuit family computing re-encryption), we conclude that the scheme is

[4] To do this, the distinguisher simply runs the adversary with the public keys, answering the adversary's re-encryption requests using the re-encryption oracle.

also secure against adversaries who are given an obfuscated re-encryption circuit. The predicate definition would *not* let us make such a conclusion.

2.2 Obfuscating Probabilistic Programs

In this section we discuss an impossibility result for average-case secure obfuscation of *deterministic* circuits, and explain how we side-step this impossibility by considering probabilistic circuits. Wee [Wee05] observes that the only deterministic circuits that can be obfuscated under strong security-preserving notions of obfuscation are those that are *learnable*. This result also applies to obfuscating *deterministic* circuits under Definition 2. To see the intuition behind this result, consider a circuit family $\mathbf{C}$, the "empty" adversary who simply outputs the obfuscated circuit $\mathsf{Obf}(C)$ it gets, and a distinguisher (with black-box access to C) that outputs 1 only if whatever circuit it gets agrees with C for random inputs.[5] Because Obf preserves functionality, the above adversary that outputs $\mathsf{Obf}(C)$ will get the distinguisher to accept with all but negligible probability. To make the distinguisher accept with similar probability, the simulator must learn, from black-box access, a circuit that is (at the very least) very close to C on random inputs. Thus random circuits from $\mathbf{C}$ must be learnable from black-box access. In particular, deterministic circuit families that are not learnable cannot be obfuscated under Definition 2.

This impossibility disappears when we consider probabilistic circuit families. This is because the (efficient) distinguisher with black-box access to a probabilistic C and non black-box access to $\mathsf{Obf}(C)$ cannot necessarily distinguish whether the *distributions* that C and $\mathsf{Obf}(C)$ output on a particular input are *statistically* close or far. This is similar to the case of encryption (see Goldwasser and Micali [GM84]), where only randomness can prevent an adversary from recognizing whether two ciphertexts are encryptions of the same bit. Our obfuscation of re-encryption programs uses this observation. In fact, our simulator outputs a "dummy circuit" that has little to do with the circuit being obfuscated, but is still indistinguishable from the true obfuscated circuit.

3 Algebraic Setting and Assumptions

Bilinear Groups. Let $\mathsf{BMsetup}$ be an algorithm that, on input the security parameter 1^k, outputs the parameters for a bilinear map as $(q, g, \mathbb{G}, \mathbb{G}_T, \mathbf{e})$, where $\mathbb{G}, \mathbb{G}_T$ are groups of prime order $q \in \Theta(2^k)$. The efficient mapping $\mathbf{e} : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ is both *bilinear*, i.e., for all $g \in \mathbb{G}$ and $a, b \in \mathbb{Z}_q$, $\mathbf{e}(g^a, g^b) = \mathbf{e}(g, g)^{ab}$, and *non-degenerate*, i.e., if g generates $\mathbb{G}$, then $\mathbf{e}(g, g) \neq 1$.

For simplicity, we present our solution using bilinear maps of the form $\mathbf{e} : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$. Our scheme can also be implemented in the more general setting where $\mathbf{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ and isomorphisms between $\mathbb{G}_1$ and $\mathbb{G}_2$ may not be efficiently computable. Galbraith, Paterson, and Smart [GPS06] provide more information on various implementation options.

[5] Wee considers different distinguishers that check different inputs.

Complexity Assumptions. In this paper, we make the following two complexity assumptions in bilinear groups. When we say two distributions are computationally indistinguishable, we mean with respect to a distinguisher with auxiliary information (which is selected independently of the instance).

Assumption 1 (Strong Diffie Hellman Indistinguishability) *Let $\mathbb{G}$ be a group of order q where q is a k-bit prime, $g \xleftarrow{r} \mathbb{G}$ and $a, b, c, d \xleftarrow{r} \mathbb{Z}_q$. Then the following two distributions are computationally indistinguishable:*

$$\left\{g, g^a, g^b, g^c, g^{abc}\right\}_k \stackrel{c}{\approx} \left\{g, g^a, g^b, g^c, g^d\right\}_k$$

This assumption has not been proposed before, but it is implied by the *Decision 3-party Diffie-Hellman* assumption proposed in [BSW06].

Assumption 2 (Decision Linear [BBS04]) *Let $\mathbb{G}$ be a group of order q where q is a k-bit prime, $f, g, h \xleftarrow{r} \mathbb{G}$ and $a, b, c \xleftarrow{r} \mathbb{Z}_q$. Then the following two distributions are computationally indistinguishable:*

$$\left\{f, g, h, f^a, g^b, h^{a+b}\right\}_k \stackrel{c}{\approx} \left\{f, g, h, f^a, g^b, h^c\right\}_k$$

4 A Special Encryption Scheme and Re-Encryption Functionality

In this section, we describe a special encryption scheme and a *re-encryption functionality* for which we later present a secure obfuscation scheme.

4.1 A Special Encryption Scheme Π

Our special encryption scheme Π is described in Fig. 1. The encryption algorithm supports two forms of ciphertexts and takes an additional input $\beta \in \{0, 1\}$ to choose between them. For the first form, encryption and decryption work as per the Boneh et al. [BBS04] construction. For the second form, the encryption and decryption are novel and relevant for re-encryption. Note that this encryption system also requires the message space M to be a subset of $\mathbb{G}$ which is of size polynomial in k. The semantic security of this scheme will be proven in Thm. 3.

4.2 Re-Encryption Functionality

Recall that obfuscation is with respect to a class of circuits. We now define a special class of re-encryption circuits for the encryption scheme Π which can be easily analyzed.

Let (pk_1, sk_1) and (pk_2, sk_2) be two keys pairs which were generated by running KeyGen on independent random tapes. When given an honestly-generated ciphertext encrypted under pk_1, a *re-encryption circuit* decrypts the ciphertext and then re-encrypts the resulting message under a second public key pk_2. For

Fig. 1. ENCRYPTION SCHEME Π

Common: For a security parameter 1^k, let $(q, g, \mathbb{G}, \mathbb{G}_T, \mathbf{e}) \leftarrow \mathsf{BMsetup}(1^k)$ be a common parameter and let $M \subset \mathbb{G}$ where $|M| = O(\text{poly}(k))$ be the message space.

$\mathsf{KeyGen}(1^k, (q, g, \mathbb{G}, \mathbb{G}_T, \mathbf{e}))$:

1. Randomly select a new generator $h \xleftarrow{r} \mathbb{G}$ and random $a, b \xleftarrow{r} \mathbb{Z}_q$.
2. Output $pk = (h^a, h^b, h)$ and $sk = (a, b, h)$.

$\mathsf{Enc}(pk, \beta, m)$:

1. Parse $pk = (h^a, h^b, h)$.
2. Choose random $r, s \xleftarrow{r} \mathbb{Z}_q$.
3. If $\beta = 0$, output the ciphertext $\big[0,\ (h^a)^r,\ (h^b)^s,\ h^{r+s} \cdot m,\ 0\big]$.
4. If $\beta = 1$, then choose a random $t \xleftarrow{r} \mathbb{G}$, and output the ciphertext $[1,\ \mathbf{e}((h^a)^r, t),\ \mathbf{e}((h^b)^s, t),\ \mathbf{e}(h^{r+s} \cdot m, t),\ t]$.

$\mathsf{Dec}(sk, [s, W, X, Y, Z])$:

1. Parse $sk = (a, b, h)$.
2. If $s = 0$, then output $Y/(W^{1/a} \cdot X^{1/b})$.
3. If $s = 1$, then
   (a) Compute $Q = Y/(W^{1/a} \cdot X^{1/b})$.
   (b) For each $m \in M$, test if $\mathbf{e}(m, Z) = Q$. If so, output m and halt.

technical reasons, we also require the circuit to produce the pairs of public keys for which it transforms ciphertexts.

More formally, the re-encryption circuit F_{sk_1,pk_2}, when run on input $c_1 = \mathsf{Enc}(pk_1, 0, m)$ for any message $m \in M$, computes $m \leftarrow \mathsf{Dec}(sk_1, c_1)$, then computes $c_2 \leftarrow \mathsf{Enc}(pk_2, 1, m)$, and finally outputs c_2. When F_{sk_1,pk_2} is run on input the special symbol `keys`, it outputs the ordered pair of public keys (pk_1, pk_2). For ciphertexts corresponding to messages in not in M, the circuit returns a randomized ciphertext of the second form for the same message. For other ill-formed inputs, it returns $\perp$.

Furthermore, let C_{sk_1,pk_2} be the same as F_{sk_1,pk_2} with the exception that the values sk_1 and pk_2 can be read from the circuit description. This property is easy to achieve by adding a "message" section to the circuit which does not affect the circuit's output, but encodes a message, with say, AND gates encoding a 1 and OR gates encoding a 0. We now define the family of circuits:

$$C_k = \left\{C_{sk_1,pk_2} \mid (pk_1, sk_1) \leftarrow \mathsf{KeyGen}(1^k), (pk_2, sk_2) \leftarrow \mathsf{KeyGen}(1^k)\right\}$$

4.3 Security for Re-Encryption

We generalize the standard notion of indistinguishability [GM84] for encryption schemes by allowing the adversary to have access to a re-encryption oracle. In particular, the following definition captures the notion that "given a ciphertext y and black-box access to a re-encryption circuit, an adversary does not learn any information about the plaintext corresponding to y."

Definition 3 (IND-security with Oracle C_{sk_1,pk_2}). *Let Π be an encryption scheme and let the random variable $\mathsf{IND}_b(\Pi,\mathcal{A},k)$ where $b \in \{0,1\}$, $\mathcal{A} = (\mathcal{A}_1,\mathcal{A}_2)$ and $k \in \mathbb{N}$ denote the result of the following probabilistic experiment:*

$$\begin{array}{l}
\mathsf{IND}_b(\Pi,\mathcal{A},k) \\
\quad (pk_1, sk_1) \leftarrow \mathsf{KeyGen}(1^k),\ (pk_2, sk_2) \leftarrow \mathsf{KeyGen}(1^k) \\
\quad (m_0, m_1, i, \beta, z) \leftarrow \mathcal{A}_1^{C_{sk_1,pk_2}}(1^k) \\
\quad y \leftarrow \mathsf{Enc}(pk_i, \beta, m_b) \\
\quad B \leftarrow \mathcal{A}_2^{C_{sk_1,pk_2}}(y, z) \\
\quad \mathit{Output}\ B
\end{array}$$

Scheme Π is indistinguishable under a chosen-plaintext attack if $\forall$ p.p.t. algorithms $\mathcal{A}$ the following two ensembles are computationally indistinguishable:

$$\Big\{\mathsf{IND}_0(\Pi,\mathcal{A},k)\Big\}_k \overset{c}{\approx} \Big\{\mathsf{IND}_1(\Pi,\mathcal{A},k)\Big\}_k$$

Remark 1. For simplicity, we allow the adversary to pick the key pk_i under which the challenge is encrypted and the form β of the encryption. By a standard hybrid argument, the above definition is equivalent to one in which the adversary is given four encryptions of the challenge message—one per key and per form.

Theorem 3. *The encryption scheme Π (in Fig. 1) is an indistinguishable-secure encryption scheme with respect to oracle C_{sk_1,pk_2} under the Decision Linear assumption in $\mathbb{G}$.*

The proof sketch is given in Appendix A.

5 The Obfuscator for Re-Encryption

In Fig. 2, we describe an obfuscator Obf for the class of re-encryption circuits C_k relative to the encryption scheme Π defined in the previous section.

5.1 Main Result

Theorem 4. *The obfuscator in Fig. 2 is a secure obfuscator for family C_k.*

Proof sketch. Let $pk_1 = (g^{a_1}, g^{b_1}, g)$ and $pk_2 = (h^{a_2}, h^{b_2}, h)$ with appropriately defined secret keys. Let C denote the re-encryption circuit C_{sk_1,pk_2}, and let $R \leftarrow \mathsf{Obf}(C)$ be an obfuscated version of C.

Preserving Functionality. Consider any $C \in C_k$ and let circuit $R \leftarrow \mathsf{Obf}(C)$. We claim that the output distributions of C and R are statistically close (in fact, identical). To see this, we must consider three classes of inputs. First, for any message $m \in M$, observe that

$$\mathsf{Enc}(pk_1, 0, m) = [0,\ g^{a_1 r},\ g^{b_1 s}, g^{r+s} \cdot m,\ 0]$$

Fig. 2. OBFUSCATOR Obf FOR RE-ENCRYPTION CIRCUITS FOR Π

Algorithm Obf, on input a circuit $C_{sk_1,pk_2} \in C_k$,

1. Reads $sk_1 = (a_1, b_1, g)$ and $pk_2 = (h^{a_2}, h^{b_2}, h)$ from the description of C_{sk_1,pk_2}.
2. Selects a random integer $z \xleftarrow{r} \mathbb{Z}_q^*$ and compute the re-encryption tuple $(Z_1, Z_2, Z_3) = ((h^{a_2})^{z/a_1}, (h^{b_2})^{z/b_1}, h^z)$.
3. Constructs and outputs an obfuscated circuit R_{sk_1,pk_2} that contains the values $pk_1, pk_2, Z_1, Z_2, Z_3$ and does the following:
   - On input $\mathtt{keys}$, output $pk_1 = (g^{a_1}, g^{b_1}, g)$ and $pk_2 = (h^{a_2}, h^{b_2}, h)$.
   - On input a 5-tuple $[0, W, X, Y, 0]$ where $W, X, Y \in \mathbb{G}$, then:
     (a) Select input re-randomization values $r, s \xleftarrow{r} \mathbb{Z}_q^*$.
     (b) Re-randomize the input as $W' \leftarrow W \cdot (g^{a_1})^r$, $X' \leftarrow X \cdot (g^{b_1})^s$, and $Y' \leftarrow Y \cdot g^{r+s}$.
     (c) Compute $E \leftarrow \mathbf{e}(W', Z_1)$.
     (d) Compute $F \leftarrow \mathbf{e}(X', Z_2)$.
     (e) Compute $G \leftarrow \mathbf{e}(Y', Z_3)$.
     (f) Select an output re-randomization value $y \xleftarrow{r} \mathbb{Z}_q^*$.
     (g) Output the ciphertext $[1,\ E^y,\ F^y,\ G^y,\ Z_3^y]$.
   - Otherwise return $\perp$.

for a randomly chosen $r, s \xleftarrow{r} \mathbb{Z}_q^*$. When such a ciphertext is fed as input to R, the circuit outputs

$$[1,\ \mathbf{e}(g^{a_1(r+r')}, h^{a_2 z/a_1})^y,\ \mathbf{e}(g^{b_1(s+s')}, h^{b_2 z/b_1})^y,\ \mathbf{e}(g^{r+s+r'+s'} \cdot m, h^z)^y,\ h^{zy}]$$

for randomly chosen $r', s', y \xleftarrow{r} \mathbb{Z}_q^*$. Substituting $\overline{r} = \frac{r+r'}{\ell}, \overline{s} = \frac{s+s'}{\ell}, \overline{t} = h^{yz}$, and ℓ is such that[6] $g^\ell = h$, this 5-tuple can be re-written as

$$[1,\ \mathbf{e}(h^{a_2\overline{r}}, \overline{t}),\ \mathbf{e}(h^{b_2\overline{r}}, \overline{t}),\ \mathbf{e}(h^{\overline{r}+\overline{s}} \cdot m, \overline{t}),\ \overline{t}]$$

which is identically distributed to the output of $\mathsf{Enc}(pk_2, 1, m)$. Second, the same holds for all $m \in \mathbb{G} \backslash M$. Lastly, for $\mathtt{keys}$ and junk input, the outputs are identical.

Polynomial slowdown. This property follows by inspection because the obfuscated circuit computes a few bilinear maps and exponentiations.

Virtual Blackbox. In order to satisfy the virtual black-box property, it suffices to only consider the "dummy" adversary. Thus, we must construct a simulator $\mathsf{Sim}^C(1^k, z)$ such that for all distinguishers D^C which take as input an obfuscated circuit R and auxiliary input z,

$$\left| \Pr[D^C(\mathsf{Obf}(C), z) = 1] - \Pr[D^C(\mathsf{Sim}^C(1^k, z), z) = 1] \right| < neg(k).$$

Let us define the simulator $\mathsf{Sim}^C(1^k, z)$ as follows:

[6] We do not need to compute ℓ explicitly.

1. Query the oracle C on `keys` to get pk_1, pk_2.
2. Sample $Z'_1, Z'_2, Z'_3 \xleftarrow{r} \mathbb{G}$.
3. As in Step (3) of the Obf algorithm, create and output a circuit R' using the values $(pk_1, pk_2, Z'_1, Z'_2, Z'_3)$.

Notice that Sim^C produces a circuit which does not correctly compute the re-encryption function. However, we now show that under appropriate complexity assumptions, no p.p.t. distinguisher D^C will notice.

Towards this goal, notice that the output of $D^C(\mathsf{Obf}(C), z)$ is distributed identically to $\mathsf{Nice}(D^C, k, z)$ and the output of $D^C(\mathsf{Sim}^C(1^k, z))$ is distributed identically to $\mathsf{Junk}(D^C, k, z)$ where

$\mathsf{Nice}(D^C, k, z)$	$\mathsf{Junk}(D^C, k, z)$
$q,\ \mathbb{G} \leftarrow \mathsf{BMsetup}(1^k)$	$q,\ \mathbb{G} \leftarrow \mathsf{BMsetup}(1^k)$
$g, h, r \xleftarrow{r} \mathbb{G}$	$g, h, r \xleftarrow{r} \mathbb{G}$
$a_1, a_2, b_1, b_2 \xleftarrow{r} \mathbb{Z}_q$	$a_1, a_2, b_1, b_2 \xleftarrow{r} \mathbb{Z}_q$
$pk_1 \leftarrow (g^{a_1}, g^{b_1}, g)$	$pk_1 \leftarrow (g^{a_1}, g^{b_1}, g)$
$pk_2 \leftarrow (h^{a_2}, h^{b_2}, h)$	$pk_2 \leftarrow (h^{a_2}, h^{b_2}, h)$
$Z_1 \leftarrow r^{a_2/a_1};\ Z_2 \leftarrow r^{b_2/b_1}$	$Z'_1, Z'_2 \xleftarrow{r} \mathbb{G}$
$b \leftarrow D^C(pk_1, pk_2, Z_1, Z_2, r, z)$	$b \leftarrow D^C(pk_1, pk_2, Z'_1, Z'_2, r, z)$
Output b	Output b

In the above experiments, the oracle C represents the re-encryption oracle for the public keys pk_1 to pk_2 which are chosen in the experiment. There is a slight abuse of notation here; when we write $\mathsf{expt}(D^C, k, z)$ we mean that the distinguisher D has oracle access to C_{sk_1, pk_2} for the keys sk_1, pk_2 chosen in the experiment. The virtual blackbox property follows immediately from the following lemma. $\square$

Lemma 1. *Under the SDHI and Decision Linear assumptions, for all p.p.t. distinguishers D and auxiliary information z, the following two distributions are statistically close.*

$$\left\{\mathsf{Nice}(D^C, k, z)\right\}_k \quad \text{and} \quad \left\{\mathsf{Junk}(D^C, k, z)\right\}_k$$

Proof Outline. We prove this lemma in a series of incremental steps. We begin with a simple indistinguishability problem and incrementally add elements and provide access to various oracles until the experiments are equivalent to their final form in Lemma 1. Let us now start with a claim which relates the SDHI problem to a simple indistinguishability problem: (In all of the following experiments, we implicitly generate $q,\ \mathbb{G} \leftarrow \mathsf{BMsetup}(1^k)$ and each experiment is indexed by k and z although we omit this extra notation when the context is clear.)

Proposition 1. *Under the SDHI assumption,* $\mathsf{Nice}^{(1)}_{k,z} \stackrel{c}{\approx} \mathsf{Junk}^{(1)}_{k,z}$ *where*

$\mathsf{Nice}^{(1)}$*: Proceeds as* Nice *except that the output is* $(g^{a_1}, g, h^{a_2}, h, Z_1, r, z)$.
$\mathsf{Junk}^{(1)}$*: Proceeds as* Junk *except that the output is* $(g^{a_1}, g, h^{a_2}, h, Z'_1, r, z)$.

If there exists a distinguisher D which distinguishes $\mathsf{Nice}^{(1)}$ from $\mathsf{Junk}^{(1)}$ with advantage ε, then there exists an distinguisher D' which solves the SDHI problem with the same advantage (in roughly the same time).

Proof sketch. The algorithm $D'(g, g^a, g^b, g^c, Q, z)$ works as follows:

1. D' chooses a random $w \xleftarrow{r} \mathbb{Z}_q$.
2. D' runs $D(g^w, (g^b)^w, g^a, g, Q, g^c,\ z)$ and echoes the response.

Consider $a_1 = 1/b$, $a_2 = a$ and $r = g^c$. Thus, if $Q = g^{abc}$, then we have $Q = r^{ab} = r^{a_2/a_1}$ in which case the input to D is identically distributed to $\mathsf{Nice}^{(1)}$. Otherwise, Q is equal to r^t for some random t and the input to D is identically distributed to $\mathsf{Junk}^{(1)}$. □

We now extend Proposition 1 to include more input values.

Proposition 2. *Under the SDHI assumption,* $\mathsf{Nice}^{(2)}_{k,z} \overset{c}{\approx} \mathsf{Junk}^{(2)}_{k,z}$ *where*

> $\mathsf{Nice}^{(2)}$*: Same as* Nice *except that the output is* $(pk_1, pk_2, Z_1, Z_2, r, z)$.
> $\mathsf{Junk}^{(2)}$*: Same as* Junk *except that the output is* $(pk_1, pk_2, Z_1', Z_2', r, z)$.

Proof sketch. Consider the hybrid distribution $T^{(2)}$ which is the same as $\mathsf{Nice}^{(2)}$ except that $Z_2' \xleftarrow{r} \mathbb{G}$ and the output is $(pk_1, pk_2, Z_1, Z_2', r, z)$. If $\mathsf{Nice}^{(2)}$ and $\mathsf{Junk}^{(2)}$ are distinguishable with advantage ε, then either $\mathsf{Nice}^{(2)}$ and $T^{(2)}$ or $T^{(2)}$ and $\mathsf{Junk}^{(2)}$ are distinguishable by algorithm D with advantage $\varepsilon/2$. Either case implies a distinguisher for $\mathsf{Nice}^{(1)}$ from $\mathsf{Junk}^{(1)}$. In the later case, this involves picking $b_1, b_2 \in \mathbb{Z}_q$ to form public keys, picking Z_2' randomly, and using the input instance from $\mathsf{Nice}^{(1)}$ (or $\mathsf{Junk}^{(1)}$) to simulate the input distribution for D. The former case does the same, but swaps the role of a_i and b_i. □

Towards the proof of our main theorem, we now extend Prop. 2 by providing the distinguisher with an oracle which returns a five-tuple of random values which works as follows. On input $[0, W, X, Y, 0]$, where $W, X, Y \in \mathbb{G}$, $\mathcal{R}$ selects three random values $E, F, G \xleftarrow{r} \mathbb{G}_T$ and a random value $H \xleftarrow{r} \mathbb{G}$ and returns $[1, E, F, G, H]$. Otherwise, $\mathcal{R}$ returns $\perp$. Intuitively, oracle $\mathcal{R}$ outputs only random values and thus should not help any distinguisher.

Proposition 3. *Under the SDHI assumption,* $\mathsf{Nice}^{(3)}_{k,z} \overset{s}{\approx} \mathsf{Junk}^{(3)}_{k,z}$ *where*

> $\mathsf{Nice}^{(3)}$*: Same as* $\mathsf{Nice}(D^{\mathcal{R}}, k, z)$.
> $\mathsf{Junk}^{(3)}$*: Same as* $\mathsf{Junk}(D^{\mathcal{R}}, k, z)$.
> *(That is, the distinguishers have oracle access to $\mathcal{R}$ instead of C and unlike the $^{(2)}$-experiments which output a tuple, these experiments output a bit.)*

Proof sketch. The oracle $\mathcal{R}$ can be perfectly simulated without any auxiliary information. Thus, for any D^R, there exists another non-oracle distinguisher D' (which internally runs D while perfectly simulating $\mathcal{R}$ to D) whose output distribution is identical to D. Proposition 2 therefore implies that for all distinguishers D^R, $\mathsf{Nice}^{(2)} \stackrel{c}{\approx} \mathsf{Junk}^{(2)}$ which implies $\mathsf{Nice}^{(3)} \stackrel{s}{\approx} \mathsf{Junk}^{(3)}$ (since the later experiment outputs a bit). □

We now return to the first experiments in which the distinguisher has oracle access to the re-encryption circuit C.

Proposition 4. *For any p.p.t. distinguisher D, let*

$$\alpha(k,z) = \mathsf{Adv}\left(\mathsf{Nice}(D^C,k,z),\ \mathsf{Junk}(D^C,k,z)\right)$$
$$\beta(k,z) = \mathsf{Adv}\left(\mathsf{Nice}(D^{\mathcal{R}},k,z),\ \mathsf{Junk}(D^{\mathcal{R}},k,z)\right)$$

be the advantage[7] *that D has in distinguishing* Nice *from* Junk *given either a re-encrypting oracle C or a random oracle $\mathcal{R}$ respectively. There exists a p.p.t. algorithm $\mathcal{A}$ which decides the Decision Linear problem with probability at least $\frac{1}{2} + \frac{1}{4}(\alpha(k,z) - \beta(k,z))$.*

Proof. Without loss of generality, assume that $\alpha > \beta$. (If not, then we flip the way $\mathcal{A}$ guesses in its final step.) The algorithm $\mathcal{A}$ takes as input, a Decision Linear instance $\Gamma = (h_1, h_2, h, h_1^x, h_2^y, Q)$ and auxiliary information z, and:

1. $\mathcal{A}$ samples a challenge bit $c \stackrel{r}{\leftarrow} \{0,1\}$ to pick whether to run Nice or Junk.
2. $\mathcal{A}$ samples integers $a, b, u \stackrel{r}{\leftarrow} \mathbb{Z}_q$ and group elements $g, Z_1', Z_2', Z_3' \stackrel{r}{\leftarrow} \mathbb{G}$.
3. $\mathcal{A}$ sets $pk_1 = (g^a, g^b, g)$ and $pk_2 = (h_1, h_2, h)$ and computes a valid re-encryption tuple (Z_1, Z_2, Z_3) by $Z_1 \leftarrow h_1^{u/a}$, $Z_2 \leftarrow h_2^{u/b}$, and $Z_3 \leftarrow h^u$.
4. If $c = 1$, then $\mathcal{A}$ runs $D^{\mathcal{O}}(pk_1, pk_2, Z_1, Z_2, Z_3, z)$ where $\mathcal{O}$ is defined below.
   If $c = 0$, then $\mathcal{A}$ runs $D^{\mathcal{O}}(pk_1, pk_2, Z_1', Z_2', Z_3', z)$.
   When D queries the oracle $\mathcal{O}$ on input $[0, W, X, Y, 0]$, $\mathcal{A}$ responds as follows:
   (a) Sample input re-randomization values $r, s, t \stackrel{r}{\leftarrow} \mathbb{Z}_q$.
   (b) Re-randomize the input as $W' \leftarrow W \cdot g^{ar}$, $X' \leftarrow X \cdot g^{bs}$, and $Y' \leftarrow Y \cdot g^{r+s}$.
   (c) Compute $E \leftarrow \mathbf{e}(W', Z_1) \cdot \mathbf{e}(g, h_1^{tx})$.
   (d) Compute $F \leftarrow \mathbf{e}(X', Z_2) \cdot \mathbf{e}(g, h_2^{ty})$.
   (e) Compute $G \leftarrow \mathbf{e}(Y', Z_3) \cdot \mathbf{e}(g, Q^t)$.
   (f) Sample output re-randomization value $\ell \stackrel{r}{\leftarrow} \mathbb{Z}_q$.
   (g) Respond with the ciphertext $[1, E^\ell, F^\ell, G^\ell, Z_3^\ell]$.
   Whenever D queries its oracle on input `keys`, $\mathcal{A}$ responds with pk_1 and pk_2, and on all other queries, $\mathcal{A}$ responds with $\perp$.
5. Eventually D outputs $c' \in \{0,1\}$. If $c = c'$, $\mathcal{A}$ outputs 1 (i.e., it guesses that $Q = h^{x+y}$). Else if $c \neq c'$, then $\mathcal{A}$ outputs 0 (i.e., it guesses that $Q \neq h^{x+y}$).

Note that $\mathcal{A}$ almost mimics the real obfuscated program. The difference is that when computing (4c)-(4e), additional terms are multiplied in to the ciphertext. When the Γ instance is a decision linear tuple, then these operations simply

[7] By advantage, we mean the following. Suppose D_0, D_1 are two probability distributions. Then for any adversary $\mathcal{A}$, the *advantage* in distinguishing D_0 from D_1 is defined as: $\mathsf{Adv}_{\mathcal{A}}(D_0, D_1) = |\Pr[x_0 \stackrel{r}{\leftarrow} D_0 : \mathcal{A}(x_0) = 1] - \Pr[x_1 \stackrel{r}{\leftarrow} D_1 : \mathcal{A}(x_1) = 1]|$.

contribute to additional re-randomization of the ciphertext (this does not change the ciphertext distribution). However, if Γ is not a decision linear instance, then these operations make E, F, G a random 3-tuple that is also independent of Z_3. This proof step is essential.

Claim: If Γ is a decision linear instance, then $\Pr[\mathcal{A}(\Gamma) = 1] = \frac{1}{2} + \alpha(k, z)/2$.
Proof of Claim: When $Q = h^{x+y}$, then $\mathcal{A}$ perfectly simulates Nice^C or Junk^C towards the algorithm D. The key point is to recognize that (h_1, h_2, h) can be interpreted as a randomly generated public key since h_1, h_2 can be rewritten as $h_1 = h^{e_1}$ and $h_2 = h^{e_2}$ for some (unknown) e_1, e_2. Since the re-encryption tuple Z_1, Z_2, Z_3 is also a valid re-encryption tuple for $pk_1 \to pk_2$, the input parameters to D in step 4 are identically distributed to the inputs to D in either experiment Nice or Junk. Moreover, the response to an oracle query on `keys` is also identically distributed. All that remains is to show that the responses $\mathcal{A}$ provides to oracle queries on $[0, W, X, Y, 0]$ are also identically distributed. This last point follows by inspection because $Q = h^{x+y}$ and Z_1, Z_2, Z_3 are a valid re-encryption tuple. A simple probability analysis completes the result:

$$\begin{aligned}
\Pr[\mathcal{A}(\Gamma) = 1 | \Gamma \in DL] &= \frac{1}{2}\left(\Pr[\mathsf{Nice}(D^C) = 1] + \Pr[\mathsf{Junk}(D^C) = 0]\right) \\
&= \frac{1}{2}\left(\Pr[\mathsf{Nice}(D^C) = 1] + 1 - \Pr[\mathsf{Junk}(D^C) = 1]\right) \\
&= \frac{1}{2} + \frac{\mathsf{Adv}(\mathsf{Nice}(D^C),\ \mathsf{Junk}(D^C))}{2} \\
&= \frac{1}{2} + \frac{\alpha}{2}
\end{aligned}$$

Claim: If Γ is not a decision linear instance, then $\Pr[\mathcal{A}(\Gamma) = 1] = \frac{1}{2} + \beta(k, z)/2$.
Proof of Claim: This proof is almost identical to the previous one. The only difference is we must show that responses to the oracle queries return four randomly selected group elements. Let us denote by ω, χ, γ, v the values such that $W = g^{\omega}, X = g^{\chi}$, $Y = g^{\gamma}$ and $Q = h^{v}$, and by e_1, e_2 the values such that $h_1 = h^{e_1}$ and $h_2 = h^{e_2}$. Observe that the elements returned by the oracle are

$$\begin{aligned}
E &= [\mathbf{e}(W \cdot g^{ar}, h_1^{u/a}) \cdot \mathbf{e}(g, h_1^{tx})]^{\ell} = \mathbf{e}(g, h)^{\ell e_1[\omega u/a + tx] + rz\ell e_1} \\
F &= [\mathbf{e}(X \cdot g^{bs}, h_2^{u/b}) \cdot \mathbf{e}(g, h_2^{ty})]^{\ell} = \mathbf{e}(g, h)^{\ell e_2[\chi u/b + ty] + sz\ell e_2} \\
G &= [\mathbf{e}(Y \cdot g^{r+s}, h^{u}) \cdot \mathbf{e}(g, Q^{t})]^{\ell} = \mathbf{e}(g, h)^{\ell[(\gamma + r + s)u] + tv\ell} \\
H &= h^{u\ell}
\end{aligned}$$

Since r, s, t, ℓ are fresh independently selected values, then E, F, G, H will also be independent on every invocation of the oracle.

Proof of Lemma 1. By the decision linear assumption and Prop. 4, it follows that $|\alpha(k, z) - \beta(k, z)|$ is negligible. By Prop. 3, $\beta(k, z)$ must be a negligible function, and therefore, so too must $\alpha(k, z)$. This establishes the lemma.

Acknowledgments

We would like to thank Ran Canetti for suggesting the problem and the TCC 2007 anonymous reviewers for their helpful comments.

References

[AFGH06] Giuseppe Ateniese, Kevin Fu, Matthew Green, and Susan Hohenberger. Improved Proxy Re-encryption Schemes with Applications to Secure Distributed Storage. *ACM Trans. on Information and System Security*, 9(1):1–30, February 2006. Previously, in *NDSS*, pages 29-43, 2005.

[AW05] Ben Adida and Douglas Wikström. How to shuffle in public. Cryptology ePrint Archive, Report 2005/394, 2005. `http://eprint.iacr.org/`.

[BBS98] Matt Blaze, Gerrit Bleumer, and Martin Strauss. Divertible protocols and atomic proxy cryptography. In *EUROCRYPT '98*, volume 1403 of LNCS, pages 127–144, 1998.

[BBS04] Dan Boneh, Xavier Boyen, and Hovav Shacham. Short Group Signatures Using Strong Diffie Hellman. In *CRYPTO*, volume 3152, pages 41–55, 2004.

[BGI+01] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In *CRYPTO '01*, volume 2139 of LNCS, pages 1–18, 2001.

[BS97] Matt Blaze and Martin Strauss. Atomic proxy cryptography. Technical report, AT&T Research, 1997.

[BSW06] Dan Boneh, Amit Sahai, and Brent Waters. Fully collusion resistant traitor tracing with short ciphertexts and private keys. In *EUROCRYPT '06*, volume 4004 of LNCS, pages 573–592, 2006.

[Can97] Ran Canetti. Towards realizing random oracles: Hash functions that hide all partial information. In *CRYPTO*, volume 1294, pages 455–469, 1997.

[CMR98] Ran Canetti, Daniele Micciancio, and Omer Reingold. Perfectly one-way probabilistic hash functions (preliminary version). In *STOC*, pages 131–140, 1998.

[DI03] Yevgeniy Dodis and Anca Ivan. Proxy cryptography revisited. In *NDSS*, 2003.

[DS05] Yevgeniy Dodis and Adam Smith. Correcting errors without leaking partial information. In *STOC '05*, pages 654–663, 2005.

[GK05] Shafi Goldwasser and Yael Tauman Kalai. On the impossibility of obfuscation with auxiliary input. In *FOCS '05*, pages 553–562, 2005.

[GM84] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, 1984. Previously, in *STOC*, pages 365-377, 1982.

[GPS06] Steven D. Galbraith, Kenneth G. Paterson, and Nigel P. Smart. Pairings for cryptographers, 2006. Cryptology ePrint Archive: Report 2006/165.

[Had00] Satoshi Hada. Zero-knowledge and code obfuscation. In *ASIACRYPT '00*, volume 1976 of LNCS, pages 443–457, 2000.

[MO97] Masahiro Mambo and Eiji Okamoto. Proxy Cryptosystems: Delegation of the Power to Decrypt Ciphertexts. *IEICE Trans. Fund. Electronics Communications and Computer Science*, E80-A/1:54–63, 1997.

[OS05] Rafail Ostrovsky and William E. Skeith III. Private searching on streaming data. In *CRYPTO*, volume 3621 of LNCS, pages 223–240, 2005.

[Pas06] Rafael Pass, 2006. Personal Communication.
[Smi05] Tony Smith. DVD Jon: buy DRM-less Tracks from Apple iTunes, March 18, 2005. `http://www.theregister.co.uk/2005/03/18/itunes_pymusique`.
[Wee05] Hoeteck Wee. On obfuscating point functions. In *STOC*, pages 523–532, 2005.

A Proof of Security for Encryption Scheme

Proof sketch.[of Thm. 3] Let us first argue that Π is an encryption scheme, i.e., it is perfectly complete. When $\beta = 0$, this follows from the BBS scheme. For the second form of ciphertexts, on input $[1, E, F, G, H]$, the decryption algorithm first computes $Q = \frac{G}{E^{1/a_2} \cdot F^{1/b_2}}$, which by inspection is equal to $\mathbf{e}(m, H)$. The decryption algorithm loops over each (of the polynomially many) $m_i \in M$ and tests whether $\mathbf{e}(m_i, H) = Q$ and therefore eventually recovers m as required.

To argue that the scheme meets the security definition, suppose adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ and distinguisher D has advantage ε in distinguishing $\mathsf{IND}_0(\cdots)$ from $\mathsf{IND}_1(\cdots)$. Then, we construct an adversary $\mathcal{A}'$ that decides the Decision Linear problem with advantage $\varepsilon/4$ as follows. Let $\Gamma = (h_1, h_2, h, h_1^x, h_2^y, Q)$ be a DL instance; $\mathcal{A}'$ works as follows:

1. Sample $a, b, c \xleftarrow{r} \mathbb{Z}_q$.
2. Set $pk_1 = (h_1, h_2, h)$ and $pk_2 = (h_1^{ac}, h_2^{bc}, h^c)$.
3. Run $\mathcal{A}_1^{\mathcal{O}}(1^k)$ to produce a tuple (m_0, m_1, i, β, z).
   When $\mathcal{A}$ queries $[s, W, X, Y, Z]$ to its oracle, respond as follows:
   (a) Return $\perp$ if $s \neq 0$ or $Z \neq 0$, or if $W, X, Y \notin \mathbb{G}$, etc.
   (b) Sample $r \in \mathbb{G}$ and use the valid re-encryption program $Z_1 \leftarrow r^a, Z_2 \leftarrow r^b$ and $Z_3 \leftarrow r$ to compute a response.
4. Sample a bit $t \xleftarrow{r} \{0, 1\}$.
5. Set y to be the ciphertext $[0, h_1^x, h_2^y, Q \cdot m_t, 0]$ if $i = 0$ and $[0, (h_1^x)^{ac}, (h_2^y)^{bc}, Q^c \cdot m_t, 0]$ if $i = 1$. Furthermore, if $\beta = 1$, transform y into a second-form ciphertext (this can be done with public information).
6. Run $B \leftarrow \mathcal{A}_2^{\mathcal{O}}(y, z)$
7. Run $t' \leftarrow D(B)$ and output 1 if $t' = t$ (guess that Γ is a DLA instance) and otherwise output 0.

We argue that when Γ is a DL instance, $\mathcal{A}'$ perfectly simulates the experiment IND_t. When Γ is not an instance, then the encryption y is independent of the message m_t and so the probability that $t' = t$ is exactly $1/2$. The proof of the theorem follows by standard probability manipulation of these two facts. $\square$