Collusion-Resistant Obfuscation and Functional Re-encryption

Nishanth Chandran[*] UCLA

Melissa Chase[†] Microsoft Research

Vinod Vaikuntanathan[‡] Microsoft Research

Abstract

Program Obfuscation is the problem of transforming a program into one which is functionally equivalent, yet whose inner workings are completely unintelligible to an adversary. Despite its immense cryptographic utility, program obfuscation has proved to be a hard and elusive goal, as evidenced by the wide-ranging impossibility results, starting with the work of Barak *et al.* (CRYPTO 2001). There is a limited, although steadily increasing, set of positive results in this area, including obfuscation of point functions, proximity testing, testing of hyperplane membership, and obfuscating re-encryption programs.

The presence of auxiliary inputs about secrets is a practical and omnipresent concern in cryptography, and the case of program obfuscation is no different. Achieving program obfuscation was hard to begin with; achieving secure obfuscation in the presence of auxiliary information about the program is downright daunting. In particular, virtually no positive results are known in this setting.

In this work, we define a specific form of auxiliary input security, called *collusion-resistant* obfuscation. Informally, we consider a setting where the program to be obfuscated is composed of many "pieces", each one chosen by a different party. The question then is: does the obfuscation remain secure, even if the adversary gets hold of the pieces of the program belonging to a subset of the parties? Thus, the auxiliary input here is simply the various pieces of the program.

Following the work of Hohenberger *et al.* (TCC 2007), we consider the notion of average-case secure obfuscation and define collusion resistance with respect to this notion. We then show how to obfuscate a natural and complex cryptographic functionality called *functional re-encryption*. Informally, the functional re-encryption functionality for a public-key encryption scheme and a policy function F with n possible outputs is one that transforms ("re-encrypts") an encryption of a message m under an "input public key" into an encryption of the same message m under one of the n "output public keys", namely the public key indexed by $F(m)$. We show how to obfuscate functional re-encryption for any policy function F (with a polynomial-size domain) using bilinear maps.

In a nutshell, our result shows how to achieve a meaningful relaxation of the highly useful yet elusive notion of auxiliary input security, for a sophisticated cryptographic functionality.

Keywords: program obfuscation, auxiliary input security, functional re-encryption

[*] `nishanth@cs.ucla.edu`. Part of this work was done while at Microsoft Research, Redmond.

[†] `melissac@microsoft.com`

[‡] `vinodv@alum.mit.edu`

1 Introduction

Informally, a program obfuscator is an algorithm that transforms a program into another, functionally equivalent program whose inner workings are "completely unintelligible". Starting from the formalization of program obfuscation in the work of Barak, Goldreich, Impagliazzo, Rudich, Sahai, Vadhan and Yang [BGI$^+$01], the problem has received considerable attention in the cryptographic community. A method of obfuscating programs is an exceedingly valuable tool, both in theory and practice.

Despite its potential for far-reaching applications, the area of program obfuscation is wrought with impossibility results. The seminal work of Barak et al. [BGI$^+$01] demonstrated a class of circuits which cannot be obfuscated even under a weak notion of obfuscation, thereby diminishing the hope of achieving general-purpose obfuscation. Further impossibility results for obfuscation of more natural functionalities was shown in [GK05, Wee05, HMLS07, BC10]. Positive results for obfuscation, on the other hand, had been largely limited to relatively simple class of functions such as point functions [Can97, CMR98, LPS04, Wee05, GK05, CD08], proximity testing [DS05], encrypted permutations [AW07] and more recently, testing hyperplane membership [CRV10].

Hohenberger et al. [HRSV07] showed how to obfuscate a complex cryptographic functionality called re-encryption [BBS98, AFGH05]. Informally, a re-encryption program associated with two public keys transforms an encryption of a message m under the first of these keys to an encryption of the same message m under the second public key. Hohenberger et al. (and independently, [HMLS07]) also introduce a strong definition of (average-case) secure obfuscation which we will use and build on in this work. Following [HRSV07], Hada [Had10] showed how to securely obfuscate an encrypted signature functionality.

Despite the slow and steady stream of positive results for obfuscation, we seem to have relatively few techniques and paradigms for obfuscation. In particular,

- The key point that enables obfuscation in both [HRSV07] and [Had10] is that they obfuscate functionalities that compute a function "inside a ciphertext". For example, in [HRSV07], this is the decryption function and in [Had10], it is the signature function. Not surprisingly, it has been noted that given a fully homomorphic encryption scheme [RAD78, Gen09], the functionalities of [HRSV07, Had10] can be easily obfuscated. Thus, we would like to *find other paradigms for obfuscating complex functionalities.*

- Both re-encryption and obfuscated signatures can be thought of as access control mechanisms. The catch, though, is that both of them embody an "all-or-nothing" form of access control – for example, in the case of re-encryption, neither the re-encryptor nor the recipient alone can decrypt a ciphertext created by the initiator although together, the two of them can learn the entire contents of the ciphertext. We would like to consider functionalities that capture a *finer grained delegation of access.*

- An issue that is important in both theory and practice is the presence of auxiliary inputs. Most positive results on obfuscation (including [HRSV07, Had10], but also others) do not achieve any form of security against auxiliary inputs that depend on the function being obfuscated. Indeed, this task seems quite hard, as indicated by impossibility results of [GK05] (for some limited positive results against auxiliary inputs, see [BC10]). *Can we achieve obfuscation against a large, meaningful class of auxiliary inputs?*

In this work, we make progress on the above lines of inquiry. Firstly, we (slightly) weaken the definition of secure obfuscation in the presence of auxiliary inputs, and introduce the notion of *collusion resistant obfuscation*. Secondly, we show how to obfuscate a natural and complex cryptographic functionality called *functional re-encryption* in a way that satisfies this notion of security. This functionality captures a finer grained delegation of access, and also protects against collusion between various participating parties.

1.1 Collusion Resistant Obfuscation

Consider the following scenario. A department would like to create a login program that will grant access to several users - say, Alice, Bob, and Carol, who have different passwords. The department would like to obfuscate this program and give it to the server that will run it. Now, we would like to guarantee that this obfuscation remains secure even if, for example, Alice were to collude with the server. One can view Alice's password as being specific auxiliary information that an adversary obtains about the program. Note that this is a restricted form of auxiliary information as we do not allow an adversary to learn, say specific bits of Bob or Carol's passwords. In this work, we are interested in the notion of average-case secure obfuscation (as defined by [HRSV07, HMLS07]) and hence in the above example we assume that all passwords are chosen uniformly at random.

One can generalize the above functionality and obtain a general definition of collusion resistant obfuscation. We would like to obfuscate a function family $\{\mathcal{C}_n\}$ that has the following particular form. Any $C_{\mathcal{K}} \in \mathcal{C}_n$ is parameterized by a set of "secret" keys $\mathcal{K} = \{k_1, k_2, \cdots, k_\ell\}$ (in addition to any other parameters that the circuit might take) that are chosen at random from some specified distribution. Now, define a subset of keys represented through a set of indices $\mathcal{T} \subseteq [\ell]$, where $[\ell]$ denotes the set $\{1, 2, \cdots, \ell\}$. We would like to construct an obfuscation of the circuit, denoted by $\mathtt{Obf}(C_{\mathcal{K}})$, so that $\mathtt{Obf}(C_{\mathcal{K}})$ is a "secure obfuscation" of $C_{\mathcal{K}}$ (in the sense of [HRSV07]) even against an adversary that knows the set of keys $\{k_i\}_{i \in \mathcal{T}}$.

1.2 Functional Re-encryption

Functional re-encryption is an expressive generalization of re-encryption [BBS98, AFGH05]. A functional re-encryption functionality is parameterized by a policy function $F : \mathcal{D} \to [n]$ (i.e, F has domain $\mathcal{D}$ and has n possible outputs) chosen from some class of functions, an input public key pk as well as n output public keys. The functionality receives as input a ciphertext of message m with "identity" id under the input public key pk. [1] It decrypts the ciphertext using the secret key sk to get m and id, and then re-encrypts m under the "appropriate" output public key $\widehat{\mathsf{pk}}_{F(\mathsf{id})}$. Following our desiderata from before, one could think of functional re-encryption as a form of fine-grained delegation of access.

To motivate the functional re-encryption functionality, consider the following scenario: Alice wishes to have her e-mail server "route" her incoming mail to one of a set of n recipients. The

[1] This is a slight generalization of the description given earlier in the abstract where the function F is applied to the entire message. We choose to view the message as an identity on which the function F is applied, and a separate "payload" for conceptual cleanliness.

particular recipient to which the ciphertext should be routed depends on both the contents of the ciphertext – essentially, the identity id – as well as Alice's access policy encoded by her function F. The e-mail server does this by "re-encrypting" the contents of the ciphertext under the appropriate public key. The minimal requirement from such a system is that the "re-encryption mechanism" hide both the message as well as Alice's access policy – it should merely provide a means for the server to do the appropriate routing. [2]

One (not particularly appealing) way for Alice to do this would be to give the e-mail server her secret key and her access policy which lets the server decrypt all incoming messages and figure out where to route them. Unfortunately, this "solution" completely fails our minimum requirement above. Ideally, Alice would like to "obfuscate" the trivial functional re-encryption program above and give it to the server. We show how to *securely obfuscate* functional re-encryption which, informally speaking, guarantees that any "attack" that the server can carry out given the obfuscated functional re-encryption program, it could have carried out given oracle access to the functional re-encryption program (which is no power at all!)

Furthermore, in reality, we could reasonably expect the server to collude with some of the recipients to learn additional information about messages as well as Alice's access policy function F. Clearly, collusion helps the server – he can use a recipient's decryption key together with the re-encryption program to learn the output of F on certain inputs. Our strong notion of collusion-resistant secure obfuscation guarantees that this is the only information that the server could possibly learn by colluding. In this situation, the auxiliary input is the secret keys of the colluding recipients.

Selectively delegating access is indeed the central theme of a recently introduced notion of predicate encryption [KSW08, SSW09]. In fact, (predicate-hiding, public key) predicate encryption schemes can be used to solve Alice's dilemma. This is done by completely ignoring the email server and giving each of the recipients a "little secret key" that is just powerful enough to decrypt the appropriate ciphertexts (dictated by the access policy). Aside from the fact that there are no known public-key predicate hiding encryption schemes (nor even good definitions of them), this solution has two drawbacks – first, there is no way to revoke access from a recipient other than by having Alice choose a fresh key for herself (which could be quite expensive). Second, this solution requires all recipients to be aware of the existence of an access policy, while the solution based on functional re-encryption is completely transparent to the recipients – i.e., they continue using their already registered public keys, and they do not even have to know the existence of the functional re-encryption mechanism.

1.3 Overview of Results and Techniques

Collusion resistant obfuscation. We define the notion of collusion resistant obfuscation that guarantees security against a natural form of auxiliary inputs. This notion of auxiliary input security might be realizable (without random oracles) for many common cryptographic tasks.

Functional Re-encryption. As a starting point, it is easy to see that even functional re-encryption can be obfuscated given a fully homomorphic encryption (FHE) scheme. However,

[2] Of course, since the e-mail server does not know who the recipient is, it either sends the resulting ciphertext to all the recipients or publishes it on a bulletin board from which the intended recipient can then access it.

if the adversary obtains the re-encryption program together with the secret key of any of the recipients (by colluding with the appropriate recipient), it can recover the entire input secret key. That is, the scheme is totally insecure against collusion between the re-encrypter and even a single recipient.

To overcome the drawbacks of the solution for functional re-encryption from FHE, we construct a scheme that is secure even against a re-encrypter that may collude with some subset of the recipients. We show, informally:

Theorem 1 (Informal)**.** *Under the Symmetric External Diffie-Hellman assumption (or, alternatively the decisional linear assumption+the SDHI assumption), there exists an encryption scheme such that for any function $F : \mathcal{D} \to \mathcal{R}$, there is a* collusion-resistant *average-case secure obfuscation of the functional re-encryption program w.r.t. F. The size of the input ciphertext in the encryption scheme $O(|\mathcal{D}| \cdot \mathsf{poly}(\lambda))$, and the size of the output ciphertext is $O(\mathsf{poly}(\lambda))$ (i.e., independent of the domain and the range of F).*

We now present the ideas behind our construction at a very high level. One can think of a functional re-encryption program as a program that must achieve two goals - a) it must "hide" the policy function F, and b) it must also "hide" the input secret key (that it uses to decrypt the input ciphertext). These two goals must simultaneously be achieved while maintaining the right functionality. Informally, the main innovation in our work is a technique to hide the policy function - this combined with techniques from [HRSV07] allows us to achieve both goals simutaneously. We shall now describe this first technique in more detail.

Let $\mathbb{G}, \mathbb{H}, \mathbb{G}_T$ be groups such that there is a bilinear map $\mathsf{e} : \mathbb{G} \times \mathbb{H} \to \mathbb{G}_T$. Let $\mathbf{a}_1, \cdots, \mathbf{a}_d \in \mathbb{Z}_q^d$ be vectors that denote elements in the domain $\mathcal{D}$ of function F and let $\hat{a}_1, \cdots, \hat{a}_n \in \mathbb{Z}_q$ denote elements in the range $\mathcal{R}$ of F. Now consider a function OF that maps elements in $\mathbb{G}^d$ to elements in $\mathbb{G}_T$ in the following way. OF is parameterized by random generators $g \in \mathbb{G}$ and $h \in \mathbb{H}$. Upon input $g^{\mathbf{a}_i}$, OF maps it to $\mathsf{e}(g,h)^{\hat{a}_{F(i)}}$. Informally, we shall now show how to publish a program that achieves the functionality provided by OF, but at the same time hides F.

The program computes a vector $\boldsymbol{\alpha} \in \mathbb{Z}_q^d$ such that the inner product $\langle \mathbf{a}_i, \boldsymbol{\alpha} \rangle = \hat{a}_{F(i)}$ for all i. Note that this is indeed possible as $\boldsymbol{\alpha}$ is a solution to a system of d equations in d variables. The program description simply contains $h^{\boldsymbol{\alpha}}$. On input $g^{\mathbf{a}_i}$, the program computes and outputs $\prod_{j=1}^{d} \mathsf{e}(g^{a_{ij}}, h^{\alpha_j}) = \mathsf{e}(g,h)^{\langle \mathbf{a}_i, \boldsymbol{\alpha} \rangle} = \mathsf{e}(g,h)^{\hat{a}_{F(i)}}$, which is the output as desired.

Unfortunately, this solution does not completely hide the function. Note that if $F(1) = F(2)$ (say), then an adversary can learn this by simply running the above program and checking if the output is the same on both the inputs. To get around this problem, we modify the program in the following way. The program picks random w_i, for all i, and computes two vectors $\boldsymbol{\alpha}, \boldsymbol{\beta} \in \mathbb{Z}_q^d$ such that the inner product $\langle \mathbf{a}_i, \boldsymbol{\alpha} \rangle = w_i \hat{a}_{F(i)}$ and $\langle \mathbf{a}_i, \boldsymbol{\beta} \rangle = w_i$, for all i (in our actual solution we require the R.H.S of the second equation to be $w_i - 1$ instead of w_i, but we will ignore that for now). The program description now contains $h^{\boldsymbol{\alpha}}, h^{\boldsymbol{\beta}}$. On input $g^{\mathbf{a}_i}$, the program computes and outputs $\prod_{j=1}^{d} \mathsf{e}(g^{a_{ij}}, h^{\alpha_j}) = \mathsf{e}(g,h)^{w_i \hat{a}_{F(i)}}$, as well as $\prod_{j=1}^{d} \mathsf{e}(g^{a_{ij}}, h^{\beta_j}) = \mathsf{e}(g,h)^{w_i}$. Now, on two different inputs (of F) that have the same output, the above program outputs elements of the form $(\mathsf{e}(g,h)^{xa}, \mathsf{e}(g,h)^{x})$ and $(\mathsf{e}(g,h)^{ya}, \mathsf{e}(g,h)^{y})$, for random a, x and y. However, these tuples are indistinguishable from random, even given $\mathsf{e}(g,h)$ and $\mathsf{e}(g,h)^a$, (by DDH) and hence an adversary cannot tell if $F(1) = F(2)$. This construction now ensures that F is completely hidden.

Now, note that if we let $\{g^{\mathbf{a}_i}\}, 1 \le i \le d$ be the input public key and $\mathsf{e}(g,h)^{\hat{a}_j}$ be the output public key, then one can potentially use the above construction to build a scheme that converts an encryption of message m under $g^{\mathbf{a}_i}$ to one under $\mathsf{e}(g,h)^{\hat{a}_{F(i)}}$. This is precisely what we do. Our encryption schemes are ElGamal-like, the input encryption key contains a set of vectors $g^{\mathbf{a}_i}, \cdots, g^{\mathbf{a}_d}$, and an input encryption of message m with identity i uses the key $g^{\mathbf{a}_i}$. Finally, in order to obtain a secure obfuscation, we apply techniques from [HRSV07] to re-randomize both input and output ciphertexts.

Obfuscating Functional Re-encryption for Arbitrary Policy Functions? A natural question raised by our result is whether it is possible to achieve collusion-resistant obfuscation of functional re-encryption for *arbitrary* (polynomial-time computable) policy functions F (in particular, functions F with domains of super-polynomial size). We show that this goal is impossible to achieve. In particular, we show that a collusion-resistant obfuscation with respect to a policy function F already contains within it a [BGI+01]-style obfuscation (a so-called "predicate obfuscation") of the policy function F. In some sense, this is not entirely surprising, and corresponds to the intuition that a collusion-resistant obfuscation of functional re-encryption allows computation of the function F [3], and yet hides all internal details of F except the input-output behavior. Together with the impossibility result of [BGI+01] for obfuscating general (families of) functions, this shows that there are classes of (polynomial-time computable) policy functions for which it is impossible to construct collusion-resistant secure obfuscation of functional re-encryption. See Appendix D for a formal statement and proof this result.

The next question to ask is whether there is any non-trivial policy function (with a domain of super-polynomial size) for which this goal can be achieved. We now argue that this requires some new innovation on the question of constructing *public-key* predicate encryption schemes which satisfy a strong security notion called *predicate-hiding*. Predicate encryption schemes were defined by Katz, Sahai and Waters [KSW08], following [SW05, GPSW06] (in particular, the predicate-hiding property was defined in the work of Shi, Shen and Waters [SSW09]). Constructions of predicate encryption schemes (even ones that do not achieve predicate-hiding) are known only for simple classes of functions such as inner products [KSW08]. Moreover, in the public-key setting, we do not know how to achieve (any reasonable definition of) *predicate-hiding*, even for simple functions.

2 Collusion Resistant Secure Obfuscation

2.1 Average-case Secure Obfuscation

Throughout this paper, we will implicitly assume that the adversary (as well as simulator) can obtain arbitrary polynomial-size independent auxiliary input z. We remark that our construction is secure even against this presence of such auxiliary information. We now recall the notion of average-case secure obfuscation introduced in [HRSV07] below.

[3] A collusion-resistant obfuscation of functional re-encryption allows computation of the function F since given an output secret key $\widehat{\mathsf{sk}}_i$ and the re-encryption program, one can test if $F(\mathsf{id}) = i$ for any id in the domain of F. Simply encrypt a random message with identity id, run it through the re-encryption program and decrypt it using $\widehat{\mathsf{sk}}_i$. If this returns the same message that was encrypted, then conclude that $F(\mathsf{id}) = i$.

Definition 1. *An efficient algorithm* $\mathtt{Obf}$ *that takes as input a (probabilistic) circuit* C *from the family* $\{\mathcal{C}_n\}$ *and outputs a new (probabilistic) circuit, is an* average-case secure obfuscator*, if it satisfies the following properties:*

- Preserving functionality: *With overwhelming probability* $\mathtt{Obf}(C)$ *behaves "almost identically" to* C *on all inputs. Formally, there exists a negligible function* $\mathtt{neg}(\lambda)$*, such that for any input length* n *and any* $C \in \mathcal{C}_n$*:*

$$\Pr_{\textit{coins of } \mathtt{Obf}}[\exists x \in \{0,1\}^n : C' \leftarrow \mathtt{Obf}(C);\ \mathtt{SD}(C'(x), C(x)) \geq \mathtt{neg}(\lambda)] \leq \mathtt{neg}(\lambda)$$

where $\mathtt{SD}(\mathcal{X}, \mathcal{Y})$ *denotes the statistical distance between two distributions* $\mathcal{X}$ *and* $\mathcal{Y}$*.*

- Polynomial slowdown: *There exists a polynomial* $p(n)$ *such that for sufficiently large input lengths* n*, for any* $C \in \mathcal{C}_n$*, the obfuscator* $\mathtt{Obf}$ *only enlarges* C *by a factor of* p*. That is,* $|\mathtt{Obf}(C)| \leq p(|C|)$*.*

- Average-case Virtual Black-Boxness: *For any efficient adversary* $\mathcal{A}$*, there exists an efficient simulator* $\mathcal{S}$*, and a negligible function* $\mathtt{neg}(\lambda)$*, such that for every efficient distinguisher* D*, and for every input length* n*:*

$$\left| \Pr[C \leftarrow \mathcal{C}_n : \mathsf{D}^C(\mathcal{A}(\mathtt{Obf}(C))) = 1] - \Pr[C \leftarrow \mathcal{C}_n : \mathsf{D}^C(\mathcal{S}^C(1^\lambda)) = 1] \right| \leq \mathtt{neg}(\lambda)$$

The probability is over the selection of a random circuit C *from* $\mathcal{C}_n$*, and the coins of the distinguisher, the simulator, the oracle, and the obfuscator.*

2.2 Average-case secure obfuscation with collusion

Consider the case where we would like to obfuscate a function family $\{\mathcal{C}_n\}$ that has the following particular form. Any $C_{\mathcal{K}} \in \mathcal{C}_n$ is parameterized by a set of "secret" keys $\mathcal{K} = \{k_1, k_2, \cdots, k_\ell\}$ (in addition to any other parameters that the circuit might take) that are chosen at random from some specified distribution. Now, define a (non-adaptively chosen) subset of keys represented through a set of indices $\mathcal{T} \subseteq [\ell]$, where $[\ell]$ denotes the set $\{1, 2, \cdots, \ell\}$. We would like to construct an obfuscation of the circuit, denoted by $\mathtt{Obf}(C_{\mathcal{K}})$, so that $\mathtt{Obf}(C_{\mathcal{K}})$ is a "secure obfuscation" of $C_{\mathcal{K}}$ (in the sense of [HRSV07]) even against an adversary that knows the set of keys $\{k_i\}_{i \in \mathcal{T}}$.

We accomplish this using a definition that is similar in spirit to the notion of obfuscation against *dependent auxiliary inputs* [GK05]. More precisely, in addition to their usual inputs and oracles, we give both the adversary and the simulator access to a (non-adaptively chosen) subset $\{k_i\}_{i \in \mathcal{T}} \subseteq \mathcal{K}$ of the keys. This can be seen as auxiliary information about the circuit $C_{\mathcal{K}} \leftarrow \mathcal{C}_n$. The formal definition of collusion-resistant secure obfuscation is as follows.

Definition 2. *An efficient algorithm* $\mathtt{Obf}$ *that takes as input a (probabilistic) circuit and outputs a new (probabilistic) circuit, is a* collusion-resistant (average-case) secure obfuscator *for the family* $\{\mathcal{C}_n\}$ *if it satisfies the following properties:*

- "Preserving functionality*" and* "Polynomial Slowdown*", as in Definition 1.*

- Average-case Virtual Black-Boxness against Collusion: *For any efficient adversary $\mathcal{A}$, there exists an efficient simulator $\mathcal{S}$, and a negligible function $\texttt{neg}(\lambda)$, such that for every input length n, every efficient distinguisher D, and any subset $\mathcal{T} \subseteq [\ell]$:*

$$\Big| \Pr[C_{\mathcal{K}} \leftarrow \mathcal{C}_n : \mathsf{D}^{C_{\mathcal{K}}}(\mathcal{A}(\texttt{Obf}(C_{\mathcal{K}}), \{k_i\}_{i\in\mathcal{T}})) = 1] - \Pr[C_{\mathcal{K}} \leftarrow \mathcal{C}_n : \mathsf{D}^{C_{\mathcal{K}}}(\mathcal{S}^{C_{\mathcal{K}}}(1^\lambda, \{k_i\}_{i\in\mathcal{T}})) = 1] \Big| \leq \texttt{neg}(\lambda)$$

The probability is over the selection of a random circuit $C_{\mathcal{K}}$ *from $\mathcal{C}_n$, and the coins of the distinguisher, the simulator, the oracle, and the obfuscator.*

Remarks on the Definition.

Handling Malicious Choice of Keys: An even stronger attack model allows the adversary to obtain an obfuscation of a circuit $C_{\mathcal{K}}$ where some of the keys in $\{k_i\}_{i\in\mathcal{T}}$ are adversarially chosen. Further, one could allow the adversary to select the set $\mathcal{T}$ adaptively, after seeing the public keys and/or the obfuscated program. We postpone a full treatment of these issues to future work.

2.2.1 Securely obfuscating Functional Re-encryption

We would like to obtain a collusion-resistant average-case obfuscator for the functional re-encryption functionality. A Functional Re-encryption (FR) functionality associated to a function $F : \mathcal{D} \to \mathcal{R}$, an input public/secret key pair $(\mathsf{pk}, \mathsf{sk})$, and output public keys $\widehat{\mathsf{pk}}_1, \ldots, \widehat{\mathsf{pk}}_{|\mathcal{R}|}$[4] is a functionality that takes as input a ciphertext $c = \texttt{I-Enc}(\mathsf{pk}, \mathsf{id}, m)$ and re-encrypts m under the output public key $\widehat{\mathsf{pk}}_{F(\mathsf{id})}$. More precisely, we are interested in a family of circuits

$$\mathcal{FR} = \{\mathsf{FR}_{\lambda,F,\mathcal{D},\mathcal{R}} : \lambda > 0 \text{ and } \mathcal{D}, \mathcal{R} \subseteq \{0,1\}^* \text{ and } F : \mathcal{D} \to \mathcal{R}\}$$

where each circuit $C_{\mathsf{pk},\mathsf{sk},\widehat{\mathsf{pk}}_1,\ldots,\widehat{\mathsf{pk}}_{|\mathcal{R}|}} \in \mathsf{FR}_{\lambda,F,\mathcal{D},\mathcal{R}}$ is a *probabilistic circuit* indexed by a key pair $(\mathsf{pk}, \mathsf{sk}) \leftarrow \texttt{I-Gen}(1^\lambda)$, and public keys $(\widehat{\mathsf{pk}}_i, \star) \leftarrow \texttt{O-Gen}(1^\lambda)$, and works as follows:

```
$C_{\mathsf{pk},\mathsf{sk},\widehat{\mathsf{pk}}_1,\ldots,\widehat{\mathsf{pk}}_{|\mathcal{R}|}}$, on input $c$ :
    Computes $(\mathsf{id}, m) \leftarrow \texttt{I-Dec}(\mathsf{sk}, c)$, and outputs $\widehat{c} \leftarrow \texttt{O-Enc}(\widehat{\mathsf{pk}}_{F(\mathsf{id})}, m)$.
    If $\texttt{I-Dec}(\mathsf{sk}, c)$ returns $\perp$ then outputs random elements according to the format of $\widehat{c}$.
$C_{\mathsf{pk},\mathsf{sk},\widehat{\mathsf{pk}}_1,\ldots,\widehat{\mathsf{pk}}_{|\mathcal{R}|}}$, on a special input keys:
    Outputs $\mathsf{pk}, \widehat{\mathsf{pk}}_1, \ldots, \widehat{\mathsf{pk}}_{|\mathcal{R}|}$.
```

In other words, all public keys included in the circuit $C_{\mathsf{pk},\mathsf{sk},\widehat{\mathsf{pk}}_1,\ldots,\widehat{\mathsf{pk}}_{|\mathcal{R}|}}$ are considered public knowledge, and the only pieces of information we are interested in protecting are the input secret key sk and the function F. Also, note that we are interested in guaranteeing security for arbitrarily chosen F and not F chosen at random.

The set of secret keys that will parameterize a functional re-encryption functionality is $\mathcal{K} = \{\widehat{\mathsf{sk}}_1, \cdots, \widehat{\mathsf{sk}}_{|\mathcal{R}|}\}$. The definition of collusion-resistant average-case secure obfuscation guarantees

[4] Without loss of generality, and for simplicity of notation, we will often assume that the domain $\mathcal{D} = \{1, 2, \ldots, d\}$ and the range $\mathcal{R} = \{1, 2, \ldots, n\}$ throughout the paper.

security against an adversary who not only knows the re-encryption program, but also has access to a subset $\{\widehat{\mathsf{sk}}_i\}_{i\in\mathcal{T}} \subseteq \mathcal{K}$ of the output secret keys. This scenario endows the adversary with considerable power and knowledge. For instance,

- The adversary will inevitably be able to decrypt all ciphertexts $c = \texttt{I-Enc}(\mathsf{pk}, \mathsf{id}, m)$, where $F(\mathsf{id}) \in \mathcal{T}$, simply by using the re-encryption program to convert the ciphertext c into an encryption of m under the output public key $\widehat{\mathsf{pk}}_{F(\mathsf{id})}$, and then decrypting it using $\widehat{\mathsf{sk}}_{F(\mathsf{id})}$.
- Moreover, the power to selectively decrypt a subset of the input ciphertexts gives the adversary information about the access policy function F itself. For instance, the adversary can determine if $F(\mathsf{id}) = i$ whenever $i \in \mathcal{T}$.

3 Preliminaries

We let λ be the security parameter throughout this paper. By $\texttt{neg}(\lambda)$ we denote some *negligible* function, namely a function μ such that for all $c > 0$ and all sufficiently large λ, $\mu(\lambda) < 1/\lambda^c$. For two distributions $\mathcal{D}_1$ and $\mathcal{D}_2$, $\mathcal{D}_1 \overset{c}{\approx} \mathcal{D}_2$ means that they are computationally indistinguishable (to be precise, this statement holds for *ensembles* of distributions).

We let $[\ell]$ denote the set $\{1, \cdots, \ell\}$. We denote vectors by bold-face letters, e.g., $\mathbf{a}$. Let $\mathbb{G}$ be a group of prime order q. For a vector $\mathbf{a} = (a_1, a_2, \cdots, a_\ell) \in \mathbb{Z}_q^\ell$ and group element $g \in \mathbb{G}$, we write $g^{\boldsymbol{a}}$ to mean the vector $(g^{a_1}, g^{a_2}, \cdots, g^{a_\ell})$. For two vectors $\boldsymbol{a}$ and $\boldsymbol{b}$ where $\boldsymbol{a}$ and $\boldsymbol{b}$ are either both in $\mathbb{Z}_q^\ell$ or both in $\mathbb{G}^\ell$, we write $\boldsymbol{ab}$ to denote their component-wise product and $\boldsymbol{a}/\boldsymbol{b}$ to denote their component-wise division. In case $\boldsymbol{b} \in \mathbb{Z}_q^\ell$, we let $\boldsymbol{a}^{\boldsymbol{b}}$ denote their component-wise exponentiation. For a vector $\boldsymbol{a}$ and scalar x, $x\boldsymbol{a} = \boldsymbol{ab}$, $\boldsymbol{a}/x = \boldsymbol{a}/\boldsymbol{b}$, and $\boldsymbol{a}^x = \boldsymbol{a}^{\boldsymbol{b}}$, where $\boldsymbol{b} = (x, x, \cdots, x)$ of dimension ℓ.

Assumptions. We assume the existence of families of groups $\{\mathbb{G}^{(\lambda)}\}_{\lambda>0}$, $\{\mathbb{H}^{(\lambda)}\}_{\lambda>0}$ and $\{\mathbb{G}_T^{(\lambda)}\}_{\lambda>0}$ with prime order $q = q(\lambda)$, endowed with a bilinear map $\mathsf{e}_\lambda : \mathbb{G}^{(\lambda)} \times \mathbb{H}^{(\lambda)} \to \mathbb{G}_T^{(\lambda)}$. When clear from the context, we omit the superscript that refers to the security parameter from all these quantities. The mapping is efficiently computable, and is bilinear – namely, for any generators $g \in \mathbb{G}$ and $h \in \mathbb{H}$, and $a, b \in \mathbb{Z}_q$, $\mathsf{e}(g^a, h^b) = \mathsf{e}(g, h)^{ab}$. We also require the bilinear map to be non-degenerate, in the sense that if $g \in \mathbb{G}, h \in \mathbb{H}$ generate $\mathbb{G}$ and $\mathbb{H}$ respectively, then $\mathsf{e}(g, h) \neq 1$.

We assume the *Symmetric External Diffie-Hellman Assumption* (SXDH)), which says that the decisional Diffie-Hellman (DDH) problem is hard in both of the groups $\mathbb{G}$ or $\mathbb{H}$. That is, the following two ensembles are indistinguishable:

$$\Big\{(q, \mathbb{G}, \mathbb{H}, \mathbb{G}_T, \mathsf{e}) \leftarrow \mathsf{BilinSetup}(1^\lambda); g \leftarrow \mathbb{G}; a, b \leftarrow \mathbb{Z}_q \ : \ (q, \mathbb{G}, \mathbb{H}, \mathbb{G}_T, \mathsf{e}, g, g^a, g^b, g^{ab})\Big\} \overset{c}{\approx}$$
$$\Big\{(q, \mathbb{G}, \mathbb{H}, \mathbb{G}_T, \mathsf{e}) \leftarrow \mathsf{BilinSetup}(1^\lambda); g \leftarrow \mathbb{G}; a, b, c \leftarrow \mathbb{Z}_q \ : \ (q, \mathbb{G}, \mathbb{H}, \mathbb{G}_T, \mathsf{e}, g, g^a, g^b, g^c)\Big\}$$

and a similar statement when $g \in \mathbb{G}$ is replaced with $h \in \mathbb{H}$. In contrast, the assumption that DDH is hard in one of the two groups $\mathbb{G}$ or $\mathbb{H}$ is simply called the external Diffie-Hellman assumption (XDH). These assumptions were first proposed and used in various works, including [Ver04, BBS04, Sco02, GS08]. In this work, we use the SXDH assumption. We remark that all our results can be obtained, albeit less efficiently, under a combination of the decisional linear assumption (DLIN) and the strong Diffie-Hellman indistinguishability (SDHI) assumptions. We defer a complete treatment of this extension to the full version.

4 Collusion-Resistant Functional Re-encryption

We are now ready to present our construction of a functional re-encryption scheme from the symmetric external Diffie-Hellman (SXDH) assumption. We first construct our basic encryption schemes in Section 4.1. In Section 4.2, we present the description of a program that implements the functional re-encryption scheme. Finally, in Section 4.3, we prove that our functional re-encryption program satisfies the notion of collusion-resistant average-case secure obfuscation.

4.1 Construction of the Encryption Schemes

A functional re-encryption scheme transforms a ciphertext under an *input public key* into a ciphertext of the same message under one of many *output public keys*. In our construction, the input and the output ciphertexts have different shapes – namely, the input ciphertext lives in the "source group" $\mathbb{G}$ whereas the output ciphertext lives in the "target group" $\mathbb{G}_T$. We now proceed to describe our input and output encryption schemes which are both variants of the ElGamal encryption scheme.

Parameters. The public parameters for both the input and the output encryption scheme consist of the description of three groups $\mathbb{G}$, $\mathbb{H}$ and $\mathbb{G}_T$ of prime order $q = q(\lambda)$, with a bilinear map $e : \mathbb{G} \times \mathbb{H} \to \mathbb{G}_T$. Also included in the public parameters are two generators – $g \in \mathbb{G}$ and $h \in \mathbb{H}$. Let $\mathcal{M} = \mathcal{M}(\lambda) \subseteq \mathbb{G}$ denote the message space of both the input and output encryption schemes. We assume that $|\mathcal{M}|$ is polynomial in λ.

The Input Encryption Scheme. We first construct the input encryption scheme, which is parameterized by $d = d(\lambda)$ which is an upper bound on the size of the domain of the policy function that we intend to support. We will also use a NIZK proof system; we note that [GS08] provides a efficient scheme for the type of statements we use, which is perfectly sound and computationally zero-knowledge based on SXDH.

1. $\texttt{I-Gen}(1^\lambda, 1^d)$: Pick random vectors $\mathbf{a}_1, \cdots, \mathbf{a}_d$ from $\mathbb{Z}_q^d$. We also generate crs, a common reference string (abbreviated CRS) for the NIZK proof system. Output $\mathsf{pk} = (\mathsf{crs}, g, g^{\mathbf{a}_1}, \cdots, g^{\mathbf{a}_d})$, and $\mathsf{sk} = (\mathbf{a}_1, \cdots, \mathbf{a}_d)$. We remark that the public key pk can be viewed as being made up of d public keys $\mathsf{pk}_i = (g, g^{\mathbf{a}_i})$ of a simpler scheme.
2. $\texttt{I-Enc}(\mathsf{pk}, i \in [d], m)$: To encrypt a message $m \in \mathcal{M}$, with "identity" $i \in [d]$, choose random exponents r and r' from $\mathbb{Z}_q^d$, and compute:
   (a) $\mathbf{C} = g^{r\mathbf{a}_i}$; $D = g^r m$, and
   (b) $\mathbf{C}' = g^{r'\mathbf{a}_i}$; $D' = g^{r'}$
   (c) π, a proof that these values are correctly formed, i.e. that they correspond to one of the vectors $g^{\mathbf{a}_i}$ contained in the public key.

Output the ciphertext $(\mathbf{E}, \mathbf{E}', \pi)$ where $\mathbf{E} = (\mathbf{C}, D)$ and $\mathbf{E}' = (\mathbf{C}', D')$. (Looking ahead, we remark that $\mathbf{E}$ looks like an encryption of message m under pk_i, while $\mathbf{E}'$ looks like an encryption of 0 under pk_i. $\mathbf{E}'$ is used only by the re-encryption program for input re-randomization, and is ignored by the decryption algorithm `I-Dec`.)

3. `I-Dec`$(\mathsf{sk}, (\mathbf{E}, \mathbf{E}'))$: If any of the components of the ciphertext $\mathbf{E}'$ is $1_{\mathbb{G}}$ or if the proof π does not verify, output $\perp$.[5] Ignore $\mathbf{E}', \pi$ subsequently, and parse $\mathbf{E}$ as $(\mathbf{C}, D)$. Check that for some $i \in [d]$ and $m \in \mathcal{M}$, $D \cdot (\mathbf{C}^{1/\mathbf{a}_i})^{-1} = (m, \cdots, m)$. If yes, output (i, m). Otherwise output $\perp$.

The Output Encryption scheme. We now describe the output encryption scheme.

1. `O-Gen`(1^λ): Pick $\hat{a} \leftarrow \mathbb{Z}_q$. Let $\widehat{\mathsf{pk}} = h^{\hat{a}}$ and $\widehat{\mathsf{sk}} = \hat{a}$.
2. `O-Enc`$(\widehat{\mathsf{pk}}, m)$: To encrypt a message $m \in \mathcal{M} \subset \mathbb{G}_1$,
   - Choose random numbers $r, s \leftarrow \mathbb{Z}_q$.
   - Compute $\widehat{Y} = (h^{\hat{a}})^r$ and $\widehat{W} = h^r$.
   - Output the ciphertext as $\big[\widehat{F}, \widehat{G}, \widehat{H}\big] := \big[\mathsf{e}(g^s, \widehat{Y}),\ \mathsf{e}(g^s, \widehat{W}) \cdot \mathsf{e}(m, h^s),\ h^s\big]$.
3. `O-Dec`$(\widehat{\mathsf{sk}} = \hat{a}, (\widehat{F}, \widehat{G}, \widehat{H}))$: The decryption algorithm does the following:
   - Compute $\widehat{Q} = \widehat{G} \cdot \widehat{F}^{-1/\hat{a}}$.
   - For each $m \in \mathcal{M}$, test if $\mathsf{e}(m, \widehat{H}) = \widehat{Q}$. If so, output m and halt.

4.2 Obfuscation for Functional Re-encryption

We now describe our scheme for securely obfuscating the functional re-encryption functionality for the input and output encryption schemes described above.

The Functional Re-encryption Key. The obfuscator gets an input *secret key* sk, the n output public keys $\widehat{\mathsf{pk}}_i$, and the description of a function $F : [d] \to [n]$. It outputs a functional re-encryption key which is a description of a program that takes as input a ciphertext of message $m \in \mathcal{M}$ and identity $i \in [d]$ under public key pk, and outputs a ciphertext of m under $\widehat{\mathsf{pk}}_{F(i)}$.

The obfuscator does the following:

1. Pick $z \leftarrow \mathbb{Z}_q$ and $w_i \leftarrow \mathbb{Z}_q$ for all $i \in [d]$ uniformly at random.
2. Solve for $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_d)$ and $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_d)$ such that for all $i \in [d]$:

$$\langle \mathbf{a}_i, \boldsymbol{\alpha} \rangle \ = \ w_i \cdot \hat{a}_{F(i)} \qquad \text{and} \qquad \langle \mathbf{a}_i, \boldsymbol{\beta} \rangle \ = \ w_i - 1$$

The re-encryption key consists of the tuple $(Z, \mathbf{A}, \mathbf{B})$ where $Z = h^z$, $\mathbf{A} = h^{z\boldsymbol{\alpha}}$ and $\mathbf{B} = h^{z\boldsymbol{\beta}}$. We remark that computing the re-encryption key does not require knowledge of the output secret keys.

[5] This "sanity check" is to ensure the security of the re-encryption program. Note that if $(\mathbf{E}, \mathbf{E}')$ is honestly generated, this event happens only with negligible probability.

The Functional Re-encryption Program. Given the functional re-encryption key $(Z, \mathbf{A}, \mathbf{B})$ and an input ciphertext $(\mathbf{E}, \mathbf{E}')$ where $\mathbf{E} = (\mathbf{C}, D)$ and $\mathbf{E}' = (\mathbf{C}', D')$, the functional re-encryption program performs the following steps:

1. *Sanity Check:* If any of the components of the input ciphertext $\mathbf{E}'$ is $1_{\mathbb{G}}$ or if the proof π does not verify, output $(\widehat{F}, \widehat{G}, \widehat{H})$ for random $\widehat{F}, \widehat{G} \in \mathbb{G}_T$ and random $\widehat{H} \in \mathbb{H}$. The sanity check is to ensure that the next step – namely, input re-randomization – randomizes the ciphertext $\mathbf{E}$.

2. *Input Re-Randomization:* Pick a random exponent $t \leftarrow \mathbb{Z}_q$ and compute $\widehat{\mathbf{C}} = \mathbf{C}(\mathbf{C}')^t$ and $\widehat{D} = D(D')^t$.

   Note that the random exponent t is used to re-randomize the encryption of 0, and this re-randomized encryption of 0 is multiplied with the encryption of m to get a re-randomized encryption of m.

3. *The main Re-encryption step:* Write $\widehat{\mathbf{C}} := (\widehat{C}_1, \ldots, \widehat{C}_d)$, $\mathbf{A} := (A_1, \ldots, A_d)$ and $\mathbf{B} := (B_1, \ldots, B_d)$. Compute

$$F = \prod_{j=1}^{d} \mathsf{e}(\widehat{C}_j, A_j) \qquad \text{and} \qquad G = \prod_{j=1}^{d} \mathsf{e}(\widehat{C}_j, B_j) \cdot \mathsf{e}(\widehat{D}, Z)$$

4. *Output Re-randomization:* Pick a random exponent $s \leftarrow \mathbb{Z}_q$ and compute $\widehat{F} = F^s$, $\widehat{G} = G^s$ and $\widehat{H} = H^s$.

   Output the ciphertext $(\widehat{F}, \widehat{G}, \widehat{H})$.

Preserving functionality. Let the input ciphertext be $(\mathbf{C}, D, \mathbf{C}', D', \pi)$. Given that π verifies, we know these values will be of the form $\mathbf{C} = g^{r\mathbf{a}_i}, D = g^r m$ and $\mathbf{C}' = g^{r'\mathbf{a}_i}, D = g^{r'}$. (If π does not verify, then both the functionality and the above program will output random group elements.) Let the re-encryption key be $(Z, \mathbf{A}, \mathbf{B})$ where $Z = h^z$, $\mathbf{A} = h^{z\boldsymbol{\alpha}}$ and $\mathbf{B} = h^{z\boldsymbol{\beta}}$.

- First, the input re-randomization step computes $\widehat{\mathbf{C}} = \mathbf{C}(\mathbf{C}')^t = g^{(r+tr')\mathbf{a}_i} {=} g^{\hat{r}\mathbf{a}_i}$ and $\widehat{D} = D(D')^t = g^{r+tr'}m = g^{\hat{r}}m$, where we defined $\hat{r} \overset{\Delta}{=} r + tr'$.

- Second, the main re-encryption step computes $F = \prod_{j=1}^{d} \mathsf{e}(\widehat{C}_j, A_j) = \mathsf{e}(g, h)^{\hat{r}z\langle \mathbf{a}_i, \boldsymbol{\alpha} \rangle} = \mathsf{e}(g, h)^{\hat{r}zw_i\hat{a}_{F(i)}}$ and

$$\begin{aligned} G &= \prod_{j=1}^{d} \mathsf{e}(\widehat{C}_j, B_j) \cdot \mathsf{e}(\widehat{D}, Z) \\ &= \mathsf{e}(g, h)^{\hat{r}z\langle \mathbf{a}_i, \boldsymbol{\beta} \rangle} \cdot \mathsf{e}(g^{\hat{r}}m, h^z) = \mathsf{e}(g, h)^{\hat{r}z(w_i - 1)} \cdot \mathsf{e}(g^{\hat{r}}, h^z) \cdot \mathsf{e}(m, h^z) = \mathsf{e}(g, h)^{\hat{r}zw_i} \cdot \mathsf{e}(m, h^z) \end{aligned}$$

- After the output re-randomization step (using randomness s), the ciphertext looks like $\widehat{F} = \mathsf{e}(g^{\sigma}, h^{\hat{a}_{F(i)}\rho})$, $G = \mathsf{e}(g^{\sigma}, h^{\rho}) \cdot \mathsf{e}(m, h^{\sigma})$ and $H = h^{\sigma}$, where $\rho = \hat{r}w_i$ and $\sigma = sz$ are both uniformly random in $\mathbb{Z}_q$, even given all the randomness in the input ciphertext. The claim about ρ being uniformly random crucially relies on the "sanity check" step in the re-encryption program.

Thus, the final ciphertext is distributed exactly like the output of $\texttt{O-Enc}(\widehat{\mathsf{pk}}_{F(i)}, m)$.

Semantic security of encryption schemes. We show that the input and output encryption schemes are semantically secure (in particular, the input scheme hides both the message and the "identity") under the DDH assumption over different groups, even given the re-encryption program. We present a detailed proof in Appendix C.

4.3 Proof of Collusion-resistant Secure Obfuscation

We show that our construction is a collusion-resistant average-case secure obfuscator for the functional re-encryption functionality. In order to satisfy collusion-resistance, the encryption as well as the obfuscation scheme have to be modified somewhat. The modifications do not affect the functionality or the security of the scheme, and are merely artifacts that seem necessary to show that our functional re-encryption scheme meets the rigorous demands of being a secure obfuscation.

A necessary modification to the encryption and obfuscation schemes. Consider the case where a corrupt recipient that holds secret key $\widehat{\mathsf{sk}}_j$ colludes with the re-encryption program. Now, essentially, this recipient has access to a program that selectively decrypts *input* ciphertexts that are encrypted with an identity i such that $F(i) = j$. However, the simulator only has oracle access to such a program. Hence, in order to put the simulator on an equal footing with the adversary we need to give the simulator the power to selectively decrypt input ciphertexts. One way to do this is to cheat and give the simulator sk_i (the vector $\mathbf{a}_i$ in our construction) for all i such that $F(i) = j$. (Note that sk_i is a secret key that allows for the selective decryption of ciphertexts with identity i, but not any other ciphertext.). For ease of exposition, we shall for now assume that the simulator obtains sk_i for all i such that $F(i) \in \mathcal{T}$. However, we would not like to resort to this cheat — we show in Appendix B how this can be avoided.

Towards showing that our obfuscation satisfies the collusion-resistant secure obfuscation definition, we first construct a simulator.

Simulator. Let $\mathcal{C} \leftarrow \mathsf{FR}_{\lambda,F,d,n}$ be a functional re-encryption circuit for the function $F : [d] \to [n]$, parameterized by the input keys $(\mathsf{pk}, \mathsf{sk})$ and the output keys $(\widehat{\mathsf{pk}}_j, \widehat{\mathsf{sk}}_j)$ for all $j \in [n]$. Let $\mathcal{T} \subseteq [n]$ be a set of corrupted receivers. We construct a simulator $\mathcal{S}$ that gets as input the secret keys of all the corrupted receivers $\widehat{\mathsf{sk}}_j$ (for $j \in \mathcal{T}$), and has oracle access to the functionality $\mathcal{C}$.

First, consider the case where none of the receivers is corrupted. Then, the simulator works as follows. Recall that the obfuscated re-encryption program consists of the tuple $(h^z, h^{z\boldsymbol{\alpha}}, h^{z\boldsymbol{\beta}})$ where z is uniformly random, and $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are solutions to some linear equations involving the input and output secret keys. The simulator, instead, simply picks $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ uniformly at random (with no relation to the input or the output keys). It then runs the adversary on this "junk functional re-encryption program" (along with the secret keys of the corrupted receivers). Under the SXDH assumption, we manage to show that this is indistinguishable from the obfuscated program that the adversary expects to get (even if the adversary is also given oracle access to the real re-encryption circuit $\mathcal{C}$).

If some of the receivers are corrupted, the simulator cannot choose $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ at random any more. Indeed, since the distinguisher has the corrupted output keys, it can check if the $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ (in the exponent) satisfy the equations involving the corrupted keys, namely $\{\widehat{\mathsf{sk}}_j\}_{j\in\mathcal{T}}$. Thus, the simulator has to choose $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ as *uniformly random solutions to a set of equations that involve the corrupted keys.* It turns out that this can be done efficiently since the simulator knows the keys of the corrupted receivers as well. As mentioned before, for ease of exposition, we shall also provide the simulator with $\{\mathsf{sk}_j\}_{j\in F^{-1}(\mathcal{T})}$. However, we show how this can be removed in Appendix B.

Without further ado, let us present the simulator $\mathcal{S}^{\mathcal{C}}(1^\lambda, \mathcal{T}, \{\widehat{\mathsf{sk}}_i\}_{i\in\mathcal{T}}, \{\mathsf{sk}_j\}_{j\in F^{-1}(\mathcal{T})})$ that works as follows:

1. Query the oracle $\mathcal{C}$ on input the string "`keys`" to get all the public keys, including the input public key $\mathsf{pk} = (g, g^{\mathbf{a}_1}, \cdots, g^{\mathbf{a}_d})$; and the output public keys $\widehat{\mathsf{pk}}_1 = (h, h^{\hat{a}_1}), \cdots, \widehat{\mathsf{pk}}_n = (h, h^{\hat{a}_n})$.

2. Sample random $z, w_1, \ldots, w_d$ from $\mathbb{Z}_q$. Sample random $\boldsymbol{\alpha}, \boldsymbol{\beta}$ from $\mathbb{Z}_q^d$ such that

$$\forall i \text{ s.t. } F(i) \in \mathcal{T}: \qquad \langle \mathbf{a}_i, \boldsymbol{\alpha}\rangle = w_i \hat{a}_{F(i)} \qquad \text{and} \qquad \langle \mathbf{a}_i, \boldsymbol{\beta}\rangle = w_i - 1$$

   Note that this can be done efficiently using the knowledge of the vectors $\mathbf{a}_i$ that we obtained in Step 1, as well as the numbers $\hat{a}_{F(i)}$ which are part of the corrupted secret keys. Compute $Z = h^z$, $\mathbf{A} = h^{z\boldsymbol{\alpha}}$, and $\mathbf{B} = h^{z\boldsymbol{\beta}}$. Output the tuple $(Z, \mathbf{A}, \mathbf{B})$ as the re-encryption key.

We now show that the output of the simulator described above is indistinguishable from an obfuscation of the re-encryption functionality (given in Section 4.2), even to a distinguisher that has the corrupted receivers' secret keys and oracle access to the re-encryption functionality. This proves that the obfuscation scheme we constructed in section 4.2 is a collusion-resistant average-case secure obfuscation satisfying Definition 2. More formally, we show:

Theorem 2. *Assuming SXDH, for any PPT distinguisher* D *and any corrupted set* $\mathcal{T} \subseteq [n]$,

$$\mathsf{D}^{\mathcal{C}}\left[\mathit{Obf}(\mathcal{C}), \{\widehat{\mathsf{sk}}_j\}_{j\in\mathcal{T}}\right] \overset{c}{\approx} \mathsf{D}^{\mathcal{C}}\left[\mathcal{S}^{\mathcal{C}}(1^\lambda, \mathcal{T}, \{\widehat{\mathsf{sk}}_j\}_{j\in\mathcal{T}})\right]$$

where Obf *is the obfuscator, and* $\mathcal{C} \leftarrow \mathsf{FR}_{\lambda,F,d,n}$ *is a uniformly random re-encryption circuit parameterized by* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathit{I\text{-}Gen}(1^\lambda)$ *and* $(\widehat{\mathsf{pk}}_i, \widehat{\mathsf{sk}}_i) \leftarrow \mathit{O\text{-}Gen}(1^\lambda)$.

We now describe a sketch of the proof of this theorem. For the formal proof, see Appendix A.

Proof. (sketch.) At a high level, the proof will go through the following steps:

- **Step** 1**:** For simplicity, let us first consider the case when there is no collusion – that is, neither the distinguisher nor the simulator has access to any of the output secret keys. Later, we will point out the necessary modifications to achieve collusion-resistance.

  We first show (in Lemma 1) that the re-encryption key is indistinguishable from random group elements to any distinguisher D who is given the public keys for the input and output encryption scheme (but *no oracle access*). In other words, we will show that constructing a re-encryption key $(Z, \mathbf{A}, \mathbf{B})$ where $Z = h^z, \mathbf{A} = h^{z\boldsymbol{\alpha}}$ and $\mathbf{B} = h^{z\boldsymbol{\beta}}$ with $\boldsymbol{\alpha}, \boldsymbol{\beta}$ being solutions to the equations

$$\langle \mathbf{a}_i, \boldsymbol{\alpha} \rangle = w_i \hat{a}_{F(i)} \qquad \text{and} \qquad \langle \mathbf{a}_i, \boldsymbol{\beta} \rangle = w_i - 1 \quad \text{for all } i \in [d] \tag{1}$$

is indistinguishable from constructing a re-encryption key with uniformly random $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$. This follows from two ideas – first, under the DDH assumption in group $\mathbb{H}$, it is hard to distinguish between $(h, h^{\boldsymbol{\alpha}}, h^{\boldsymbol{\beta}})$ where $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are solutions to Equations 1, from the case where they are solutions to the same set of equations with the right-hand sides replaced by *uniformly random elements* in $\mathbb{Z}_q^*$. [6] Next, we note that choosing $\boldsymbol{\alpha}, \boldsymbol{\beta}$ as a solution to a set of equations with uniformly random right-hand side is equivalent to simply choosing random $\boldsymbol{\alpha}, \boldsymbol{\beta}$. This completes the first step - in Appendix A we show that this generalizes to the case where $\mathcal{T}$ is non-empty, and the simulator's $\boldsymbol{\alpha}, \boldsymbol{\beta}$ are chosen as a random solution to the resulting underconstrained set of equations.

- **Step 2:** Next, we will provide our distinguisher D with oracle access to a random oracle that simply returns random group elements of the same format as the output ciphertext of the re-encryption program. (The only exception is that, when it receives a ciphertext encrypted under id such that $F(\mathsf{id}) \in \mathcal{T}$, it honestly performs the re-encryption.) We show, in Lemma 2, that the re-encryption key is indistinguishable from random group elements to this distinguisher $\mathsf{D}^{\mathcal{RO}}$ as well.

  This follows from Step 1 fairly easily once we note that the distinguisher in Step 1 could easily simulate this random oracle itself.

- **Step 3:** In Lemma 3, we will provide our distinguisher D with oracle access to either the re-encryption oracle or the random oracle, and argue that D will not be able to determine which oracle it is given, even if it is also given the real re-encryption key.

  The main intuition behind this proof is that, based on SXDH, we can show that honestly generated outputs ciphertexts are indistinguishable from random tuples. This is fairly easy to see: consider public key $h^{\hat{a}}$, and the following tuple $\left[\mathsf{e}(g^s, h^w),\ \mathsf{e}(g^s, h^r) \cdot \mathsf{e}(m, h^s),\ h^s\right]$ for random $\hat{a}, s, r \in \mathbb{Z}_q$. If $w = \hat{a}r$, this is a valid encryption of m, if w is a random element of $\mathbb{Z}_q$, then this is a random tuple from $\mathbb{G}_T \times \mathbb{G}_T \times \mathbb{H}$.

  A fairly straightforward hybrid argument then shows that a real encryption oracle for public keys $\widehat{\mathsf{pk}}_1, \ldots, \widehat{\mathsf{pk}}_n$ is indistinguishable from a random oracle which only produces valid ciphertexts for $\widehat{\mathsf{pk}}_i$ with $i \in \mathcal{T}$ (even when the distinguisher is given $\widehat{\mathsf{sk}}_i$ for $i \in \mathcal{T}$).

  Finally, we note that we can generate a real re-encryption key and perfectly simulate either the real re-encryption oracle or the random re-encryption oracle given only $\widehat{\mathsf{pk}}_1, \ldots, \widehat{\mathsf{pk}}_n$, and either the encryption oracle or the random oracle described above. We conclude that the real re-encryption oracle and random re-encryption oracle are indistinguishable even given the real re-encryption key (and $\widehat{\mathsf{sk}}_i$ for $i \in \mathcal{T}$).

[6] Note that the right-hand sides of Equation 1 are not random as such – for example, consider the case where $F(1) = F(2) = 1$. Then, the right-hand sides of the four equations corresponding to $i = 1$ and $i = 2$ are $w_1\hat{a}_1, w_1 - 1, w_2\hat{a}_1, w_2 - 1$, which are clearly correlated.

- **Step 4:** In Lemma 4, we will again provide our distinguisher D with oracle access to either the re-encryption oracle or the random oracle and argue that it will not be able to determine which oracle it is given, this time when given the simulated re-encryption key instead.

  Again, this follows from Step 3, when we note that the distinguisher in Step 3 could easily ignore the re-encryption key it is given and instead run the simulator to generate a simulated one.

We have argued that the distinguisher has the same behavior given the real re-encryption key and real re-encryption oracle or the real re-encryption key and random oracle (Step 3), that it has the same behavior given the real re-encryption key and random oracle or the simulated re-encryption key and random oracle (Step 2), and that it has the same behavior given the simulated re-encryption key and random oracle or the simulated re-encryption key and real re-encryption oracle (Step 4). Putting everything together, we conclude that the real re-encryption key and simulated re-encryption key are indistinguishable, even given access to the real re-encryption oracle. Thus, we obtain the proof of Theorem 2.

□

References

[AFGH05] Giuseppe Ateniese, Kevin Fu, Matthew Green, and Susan Hohenberger. Improved proxy re-encryption schemes with applications to secure distributed storage. In *NDSS, Proceedings of the Network and Distributed System Security Symposium, NDSS 2005, San Diego, California, USA*, 2005.

[AW07] Ben Adida and Douglas Wikström. How to shuffle in public. In *Theory of Cryptography, 4th Theory of Cryptography Conference, TCC 2007, Amsterdam, The Netherlands, February 21-24, 2007, Proceedings*, pages 555–574, 2007.

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

[BBS04] Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures. In *CRYPTO, Advances in Cryptology - CRYPTO 2004, 24th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 2004, Proceedings*, pages 41–55, 2004.

[BC10] Nir Bitansky and Ran Canetti. On strong simulation and composable point obfuscation. In *CRYPTO, Advances in Cryptology - CRYPTO 2010, 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings*, pages 520–537, 2010.

[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, Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23, 2001, Proceedings*, pages 1–18, 2001.

[Can97] Ran Canetti. Towards realizing random oracles: Hash functions that hide all partial information. In *CRYPTO, Advances in Cryptology - CRYPTO '97, 17th Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 1997, Proceedings*, pages 455–469, 1997.

[CD08] Ran Canetti and Ronny Ramzi Dakdouk. Obfuscating point functions with multibit output. In *EUROCRYPT, Advances in Cryptology - EUROCRYPT 2008, 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Istanbul, Turkey, April 13-17, 2008. Proceedings*, pages 489–508, 2008.

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

[CRV10] Ran Canetti, Guy N. Rothblum, and Mayank Varia. Obfuscation of hyperplane membership. In *TCC, Theory of Cryptography, 7th Theory of Cryptography Conference, TCC 2010, Zurich, Switzerland, February 9-11, 2010. Proceedings*, pages 72–89, 2010.

[DS05] Yevgeniy Dodis and Adam Smith. Correcting errors without leaking partial information. In *STOC, Proceedings of the 37th Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, May 22-24, 2005*, pages 654–663, 2005.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *STOC, Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pages 169–178, 2009.

[GK05] Shafi Goldwasser and Yael Tauman Kalai. On the impossibility of obfuscation with auxiliary input. In *FOCS, 46th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2005), 23-25 October 2005, Pittsburgh, PA, USA, Proceedings*, pages 553–562, 2005.

[GPSW06] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *ACM Conference on Computer and Communications Security*, pages 89–98, 2006.

[GS08] Jens Groth and Amit Sahai. Efficient non-interactive proof systems for bilinear groups. In *EUROCRYPT, Advances in Cryptology - EUROCRYPT 2008, 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Istanbul, Turkey, April 13-17, 2008. Proceedings*, pages 415–432, 2008.

[Had10] Satoshi Hada. Secure obfuscation for encrypted signatures. In *EUROCRYPT, Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, French Riviera, May 30 - June 3, 2010. Proceedings*, pages 92–112, 2010.

[HMLS07] Dennis Hofheinz, John Malone-Lee, and Martijn Stam. Obfuscation for cryptographic purposes. In *TCC, Theory of Cryptography, 4th Theory of Cryptography Conference, TCC 2007, Amsterdam, The Netherlands, February 21-24, 2007, Proceedings*, pages 214–232, 2007.

[HRSV07] Susan Hohenberger, Guy N. Rothblum, Abhi Shelat, and Vinod Vaikuntanathan. Securely obfuscating re-encryption. In *TCC, Theory of Cryptography, 4th Theory of Cryptography Conference, TCC 2007, Amsterdam, The Netherlands, February 21-24, 2007, Proceedings*, pages 233–252, 2007.

[KSW08] Jonathan Katz, Amit Sahai, and Brent Waters. Predicate encryption supporting disjunctions, polynomial equations, and inner products. In *EUROCRYPT, Advances in Cryptology - EUROCRYPT 2008, 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Istanbul, Turkey, April 13-17, 2008. Proceedings*, pages 146–162, 2008.

[LPS04] Ben Lynn, Manoj Prabhakaran, and Amit Sahai. Positive results and techniques for obfuscation. In *EUROCRYPT, Advances in Cryptology - EUROCRYPT 2004, International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004, Proceedings*, pages 20–39, 2004.

[RAD78] R. Rivest, L. Adleman, and M. Dertouzos. On data banks and privacy homomorphisms. In *Foundations of Secure Computation*, pages 169–177. Academic Press, 1978.

[Sco02] Mike Scott. Authenticated ID-based key exchange and remote log-in with insecure token and PIN number. http://eprint.iacr.org/2002/164, 2002.

[SSW09] Emily Shen, Elaine Shi, and Brent Waters. Predicate privacy in encryption systems. In *TCC, Theory of Cryptography, 6th Theory of Cryptography Conference, TCC 2009, San Francisco, CA, USA, March 15-17, 2009. Proceedings*, pages 457–473, 2009.

[SW05] Amit Sahai and Brent Waters. Fuzzy identity-based encryption. In *EUROCRYPT, Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings*, pages 457–473, 2005.

[Ver04] Eric R. Verheul. Evidence that xtr is more secure than supersingular elliptic curve cryptosystems. *J. Cryptology*, 17(4):277–296, 2004.

[Wee05] Hoeteck Wee. On obfuscating point functions. In *STOC, Proceedings of the 37th Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, May 22-24, 2005*, pages 523–532, 2005.

A Formal Proof of Collusion-resistant Secure Obfuscation (Theorem 2)

For any function $F : [d] \to [n]$, subset $\mathcal{T} \in [n]$, security parameter λ, and distinguisher algorithm D, we define two distributions $\mathsf{Real\text{-}C}(\mathsf{D}, 1^\lambda, F, \mathcal{T})$ and $\mathsf{Sim\text{-}C}(\mathsf{D}, 1^\lambda, F, \mathcal{T})$ as follows. Informally, $\mathsf{Real\text{-}C}(\mathsf{D}, 1^\lambda, F, \mathcal{T})$ is the distribution that describes the output of the distinguisher D given the obfuscated functional re-encryption program, and $\mathsf{Sim\text{-}C}(\mathsf{D}, 1^\lambda, F, \mathcal{T})$ is the distribution describing the output of the distinguisher given the simulated program (where in both cases the distinguisher is also given oracle access to the re-encryption circuit). We will show that these two distributions are indistinguishable under the SXDH assumption, which shows that our obfuscation method achieves the notion of collusion-resistant secure obfuscation (Definition 2).

$\mathsf{Real\text{-}C}(\mathsf{D}, 1^\lambda, F, \mathcal{T})$:

1. Choose random g from $\mathbb{G}$ and h from $\mathbb{H}$. Choose random vectors $\mathbf{a}_1, \cdots, \mathbf{a}_d$ from $\mathbb{Z}_q^d$ and exponents $\hat{a}_1, \cdots, \hat{a}_n$ from $\mathbb{Z}_q$. Generate a CRS crs for the NIZK proof system.

2. Set $\mathsf{pk} = (\mathsf{crs}, g, g^{\mathbf{a}_1}, \cdots, g^{\mathbf{a}_d})$, $\mathsf{sk} = (\mathbf{a}_1, \ldots, \mathbf{a}_d)$, and $\widehat{\mathsf{pk}}_1 = (h, h^{\hat{a}_1}), \cdots, \widehat{\mathsf{pk}}_n = (h, h^{\hat{a}_n})$.

3. Let $\mathcal{C} \in \mathcal{FR}_{\lambda,F,[d],[n]}$ be the corresponding circuit, i.e. $\mathcal{C} = C_{\mathsf{pk},\mathsf{sk},\widehat{\mathsf{pk}}_1,\ldots,\widehat{\mathsf{pk}}_n}$.

4. Run the real re-encryption key generation procedure to generate a re encryption key $(Z, \mathbf{A}, \mathbf{B})$. I.e., pick $z, w_1, \ldots, w_d$ at random from $\mathbb{Z}_q$. Choose vectors $\boldsymbol{\alpha} = \alpha_1, \cdots, \alpha_d$ and $\boldsymbol{\beta} = \beta_1, \cdots, \beta_d$ at random from $\mathbb{Z}_q^d$ with the restriction that

$$\langle \mathbf{a}_i, \boldsymbol{\alpha} \rangle = w_i \hat{a}_{F(i)} \qquad \text{and} \qquad \langle \mathbf{a}_i, \boldsymbol{\beta} \rangle = w_i - 1 \quad \text{for all } i \in [d]$$

   Compute $Z = h^z$, $\mathbf{A} = h^{z\boldsymbol{\alpha}}$, and $\mathbf{B} = h^{z\boldsymbol{\beta}}$.

5. Let b be the output $\mathsf{D}^{\mathcal{C}}(\mathsf{pk}, \widehat{\mathsf{pk}}_1, \cdots, \widehat{\mathsf{pk}}_n, (Z, \mathbf{A}, \mathbf{B}), \{\hat{a}_j\}_{j \in \mathcal{T}}, \{\mathbf{a}_{\mathsf{id}}\}_{\mathsf{id} \in F^{-1}(\mathcal{T})})$.

6. Output b.

$\mathsf{Sim\text{-}C}(\mathsf{D}, 1^\lambda, F, \mathcal{T})$

1. Choose random g from $\mathbb{G}$ and h from $\mathbb{H}$. Choose random vectors $\mathbf{a}_1, \cdots, \mathbf{a}_d$ from $\mathbb{Z}_q^d$ and exponents $\hat{a}_1, \cdots, \hat{a}_n$ from $\mathbb{Z}_q$. Generate a CRS crs for the NIZK proof system.

2. Set $\mathsf{pk} = (\mathsf{crs}, g, g^{\mathbf{a}_1}, \cdots, g^{\mathbf{a}_d})$, $\mathsf{sk} = (\mathbf{a}_1, \ldots, \mathbf{a}_d)$, and $\widehat{\mathsf{pk}}_1 = (h, h^{\hat{a}_1}), \cdots, \widehat{\mathsf{pk}}_n = (h, h^{\hat{a}_n})$.

3. Let $\mathcal{C} \in \mathcal{FR}_{\lambda,F,[d],[n]}$ be the corresponding circuit, i.e. $\mathcal{C} = C_{\mathsf{pk},\mathsf{sk},\widehat{\mathsf{pk}}_1,\ldots,\widehat{\mathsf{pk}}_n}$.

4. Run $\mathcal{S}^{\mathcal{C}}(1^\lambda)$ to obtain $(Z, \mathbf{A}, \mathbf{B})$, I.e. pick $z, w_1, \ldots, w_d$ at random from $\mathbb{Z}_q$. Choose vectors $\boldsymbol{\alpha} = \alpha_1, \cdots, \alpha_d$ and $\boldsymbol{\beta} = \beta_1, \cdots, \beta_d$ at random from $\mathbb{Z}_q^d$ with the restriction that

$$\langle \mathbf{a}_i, \boldsymbol{\alpha} \rangle = w_i \hat{a}_{F(i)} \qquad \text{and} \qquad \langle \mathbf{a}_i, \boldsymbol{\beta} \rangle = w_i - 1 \quad \text{for all } i \in F^{-1}(\mathcal{T})$$

   Compute $Z = h^z$, $\mathbf{A} = h^{z\boldsymbol{\alpha}}$, and $\mathbf{B} = h^{z\boldsymbol{\beta}}$.

5. Let b be the output $\mathsf{D}^{\mathcal{C}}(\mathsf{pk}, \widehat{\mathsf{pk}}_1, \cdots, \widehat{\mathsf{pk}}_n, (Z, \mathbf{A}, \mathbf{B}), \{\hat{a}_j\}_{j \in \mathcal{T}}, \{\mathbf{a}_{\mathsf{id}}\}_{\mathsf{id} \in F^{-1}(\mathcal{T})})$.

6. Output b.

Theorem 3. *Assuming SXDH, for all F, $\mathcal{T}$, and PPT D, the distributions $\mathsf{Real\text{-}C}(\mathsf{D}, 1^\lambda, F, \mathcal{T})$ and $\mathsf{Sim\text{-}C}(\mathsf{D}, 1^\lambda, F, \mathcal{T})$ are indistinguishable.*

Proof. We prove this through a series of steps.

Step 1:

We begin by considering the following two distributions:

- $\mathsf{RealInput}(1^\lambda, F, \mathcal{T})$, which proceeds as $\mathsf{Real\text{-}C}(\mathsf{D}, 1^\lambda, F, \mathcal{T})$ except that the output is

$$\left[\mathsf{crs}, g, g^{\mathbf{a}_1}, \cdots, g^{\mathbf{a}_d}, h, h^{\hat{a}_1}, \cdots, h^{\hat{a}_n}, (h^z, h^{z\boldsymbol{\alpha}}, h^{z\boldsymbol{\beta}}), \{\hat{a}_i\}_{i\in\mathcal{T}}, \{\mathbf{a}_{\mathsf{id}}\}_{\mathsf{id}\in F^{-1}(\mathcal{T})}\right]$$

for $\boldsymbol{\alpha}, \boldsymbol{\beta}$ chosen from $\mathbb{Z}_q^d$ such that $\langle \mathbf{a}_i, \boldsymbol{\alpha}\rangle = w_i \hat{a}_{F(i)}$ and $\langle \mathbf{a}_i, \boldsymbol{\beta}\rangle = w_i - 1$ for all $i \in [d]$ (for randomly chosen w_i's).

- $\mathsf{SimInput}(1^\lambda, F, \mathcal{T})$, which proceeds as $\mathsf{Real\text{-}C}(\mathsf{D}, 1^\lambda, F, \mathcal{T})$ except that the output is

$$\left[\mathsf{crs}, g, g^{\mathbf{a}_1}, \cdots, g^{\mathbf{a}_d}, h, h^{\hat{a}_1}, \cdots, h^{\hat{a}_n}, (h^z, h^{z\boldsymbol{\alpha}}, h^{z\boldsymbol{\beta}}), \{\hat{a}_i\}_{i\in\mathcal{T}}, \{\mathbf{a}_{\mathsf{id}}\}_{\mathsf{id}\in F^{-1}(\mathcal{T})}\right]$$

for uniformly random $\boldsymbol{\alpha}, \boldsymbol{\beta} \in \mathbb{Z}_q^d$ subject to the condition that

$$\langle \mathbf{a}_i, \boldsymbol{\alpha}\rangle = w_i \hat{a}_{F(i)} \qquad \text{and} \qquad \langle \mathbf{a}_i, \boldsymbol{\beta}\rangle = w_i - 1 \quad \text{for all } i \text{ such that } F(i) \in \mathcal{T}$$

where, as before, the w_i's are randomly chosen. The difference between the two distributions is that in $\mathsf{SimInput}(1^\lambda, F, \mathcal{T})$, the vectors $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are required to satisfy only a subset of the constraints (namely, ones that correspond to $i \in F^{-1}(\mathcal{T})$). In particular, if the corrupted set $\mathcal{T}$ is empty, then $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are truly random vectors.

Lemma 1. *Assuming SXDH, for all F, $\mathcal{T}$, $\mathsf{RealInput}(1^\lambda, F, \mathcal{T})$ and $\mathsf{SimInput}(1^\lambda, F, \mathcal{T})$ are indistinguishable.*

Proof. Consider the following hybrid distribution, in which $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are chosen to satisfy a random set of equations:

- $\mathsf{HybridInput}(1^\lambda, F, \mathcal{T})$, which proceeds as $\mathsf{Real\text{-}C}(\mathsf{D}, 1^\lambda, F, \mathcal{T})$ except that the output is

$$(\mathsf{crs}, g, g^{\mathbf{a}_1}, \cdots, g^{\mathbf{a}_d}, h, h^{\hat{a}_1}, \cdots, h^{\hat{a}_n}, (h^z, h^{z\boldsymbol{\alpha}}, h^{z\boldsymbol{\beta}}), \{\hat{a}_i\}_{i\in\mathcal{T}}, \{\mathbf{a}_{\mathsf{id}}\}_{\mathsf{id}\in F^{-1}(\mathcal{T})})$$

for $\boldsymbol{\alpha}, \boldsymbol{\beta}$ chosen as a random solution to:

$$\left\{ \begin{array}{cl} \langle \mathbf{a}_i, \boldsymbol{\alpha}\rangle = w_i \hat{a}_{F(i)} & \text{for all } i \in F^{-1}(\mathcal{T}) \\ \langle \mathbf{a}_i, \boldsymbol{\alpha}\rangle = y_i & \text{for all } i \notin F^{-1}(\mathcal{T}) \end{array} \right\} \qquad \text{and} \qquad \langle \mathbf{a}_i, \boldsymbol{\beta}\rangle = w_i - 1 \text{ for all } i \in [d]$$

where the w_i's and y_i's are chosen at random from $\mathbb{Z}_q$.

First, it is easy to see that the distributions $\mathsf{HybridInput}(1^\lambda, F, \mathcal{T})$ and $\mathsf{SimInput}(1^\lambda, F, \mathcal{T})$ are really the same distribution. This is because in both distributions, for every $i \in F^{-1}(\mathcal{T})$, the right-hand side of the two equations – one involving $\boldsymbol{\alpha}$ and the other involving $\boldsymbol{\beta}$ – use the same randomness w_i, whereas for $i \notin F^{-1}(\mathcal{T})$, the right-hand sides are random and independent (in fact, they are independent of $\hat{a}_{F(i)}$ as well).

We now claim that $\mathsf{HybridInput}(1^\lambda, F, \mathcal{T})$ and $\mathsf{RealInput}(1^\lambda, F, \mathcal{T})$ are computationally indistinguishable, assuming SXDH. This finishes the proof of the lemma.

Claim 1. *Assuming SXDH, for all F, $\mathcal{T}$, $\mathsf{HybridInput}(1^\lambda, F, \mathcal{T})$ and $\mathsf{RealInput}(1^\lambda, F, \mathcal{T})$ are indistinguishable.*

Proof. This can be proved via a series of hybrids $\mathtt{H}^t(1^\lambda, F, \mathcal{T})$, for $0 \le t \le d$.

- $\mathtt{H}^t(1^\lambda, F, \mathcal{T})$ proceeds as $\mathsf{Real\text{-}C}(\mathsf{D}, 1^\lambda, F, \mathcal{T})$ except that the output is

  $$(\mathsf{crs}, g, g^{\mathbf{a}_1}, \cdots, g^{\mathbf{a}_d}, h, h^{\hat{a}_1}, \cdots, h^{\hat{a}_n}, (h^z, h^{z\boldsymbol{\alpha}}, h^{z\boldsymbol{\beta}}), \{\hat{a}_i\}_{i\in\mathcal{T}}, \{\mathbf{a}_{\mathsf{id}}\}_{\mathsf{id}\in F^{-1}(\mathcal{T})})$$

  for $\boldsymbol{\alpha}, \boldsymbol{\beta}$ chosen as a random solution to:

  $$\left\{\begin{array}{cc} \langle \mathbf{a}_i, \boldsymbol{\alpha}\rangle = w_i \hat{a}_{F(i)} & \text{for all } i \in F^{-1}(\mathcal{T}) \\ \langle \mathbf{a}_i, \boldsymbol{\alpha}\rangle = w_i \hat{a}_{F(i)} & \text{for all } i > t \\ \langle \mathbf{a}_i, \boldsymbol{\alpha}\rangle = y_i & \text{for all } i \notin F^{-1}(\mathcal{T}) \text{ such that } i \le t \end{array}\right\} \quad \text{and} \quad \langle \mathbf{a}_i, \boldsymbol{\beta}\rangle = w_i{-}1 \text{ for all } i \in [d]$$

  where the w_i's and y_i's are chosen at random from $\mathbb{Z}_q$.

Clearly, $\mathtt{H}^0(1^\lambda, F, \mathcal{T}) = \mathsf{RealInput}(1^\lambda, F, \mathcal{T})$, and $\mathtt{H}^d(\lambda, F, \mathcal{T}) = \mathsf{HybridInput}(1^\lambda, F, \mathcal{T})$. We will argue that for all $t = 1, \cdots, d$, the distributions $\mathtt{H}^{t-1}(\lambda)$ and $\mathtt{H}^t(\lambda)$ are indistinguishable by SXDH.

Note that if $F(t) \in \mathcal{T}$, then the two distributions are identical. For all other cases, suppose there is a PPT adversary $\mathcal{A}$ that can distinguish the two distributions. Then we construct an adversary $\mathcal{B}$ that acts as a distinguisher for SXDH. $\mathcal{B}$ gets as input a tuple $(h, X_1 = h^{x_1}, X_2 = h^{x_2}, X_3 = h^{x_3})$ where either $x_3 = x_1 x_2$ (corresponding to an SXDH instance) or x_3 is uniformly random (corresponding to a non-SXDH instance). $\mathcal{B}$ works as follows:

1. Choose random $\mathbf{a}_1, \cdots, \mathbf{a}_d$ from $\mathbb{Z}_q^d$, and random $\hat{a}_j$ from $\mathbb{Z}_q$ for all $j \ne F(t)$. Generate a CRS crs for the NIZK proof system.

2. Choose random $w_1, \cdots, w_{t-1}, w_{t+1}, \cdots, w_d$ (all w_i's except w_t) and random $y_1, \cdots, y_{t-1}, y_{t+1}, \cdots, y_d$ (all y_i's except y_t) from $\mathbb{Z}_q$.

3. Choose $\mathbf{A} = (A_1, \ldots, A_d) \in \mathbb{H}^d$ and $\mathbf{B} = (B_1, \ldots, B_d) \in \mathbb{H}^d$ as a random solution to:

   $$\left\{\begin{array}{cc} \prod_{j=1}^d A_j^{a_{ij}} = h^{w_i \hat{a}_{F(i)}} & \text{for all } i \in F^{-1}(\mathcal{T}) \\ \prod_{j=1}^d A_j^{a_{ij}} = h^{w_i \hat{a}_{F(i)}} & \text{for all } i > t \\ \prod_{j=1}^d A_j^{a_{ij}} = h^{y_i} & \text{for all } i \notin F^{-1}(\mathcal{T}) \text{ such that } i < t \end{array}\right\} \quad \text{and} \quad \prod_{i=1}^d B_j^{a_{ij}} = h^{w_i - 1} \text{ for all } i \ne t$$

   and

   $$\prod_{j=1}^d A_j^{a_{tj}} = X_3 \qquad \text{and} \qquad \prod_{j=1}^d B_j^{a_{tj}} = X_2 \cdot h^{-1}$$

4. Choose a random $z \leftarrow \mathbb{Z}_q$, and generate the distribution

   $$\left[\mathsf{crs}, g, g^{\mathbf{a}_1}, \cdots, g^{\mathbf{a}_d}, h, h^{\hat{a}_1}, \cdots, h^{\hat{a}_{F(t)-1}}, X_1, h^{\hat{a}_{F(t)+1}}, \cdots, h^{\hat{a}_n}, (h^z, \mathbf{A}^z, \mathbf{B}^z), \{\hat{a}_i\}_{i\in\mathcal{T}}, \{\mathbf{a}_{\mathsf{id}}\}_{\mathsf{id}\in F^{-1}(\mathcal{T})}\right]$$

   Feed this distribution to $\mathcal{A}$ and output whatever $\mathcal{A}$ outputs.

Note that we replaced $h^{\hat{a}_{F(t)}}$ by X_1, thus implicitly setting $\hat{a}_{F(t)} = x_1$. We also set $\prod_{j=1}^{d} B_j^{a_{tj}} = h^{w_t - 1} = X_2 \cdot h^{-1} = h^{x_2 - 1}$, thus implicitly setting $w_t = x_2$. Finally, we also set $\prod_{j=1}^{d} A_j^{a_{tj}} = X_3 = h^{x_3}$. Thus, we implicitly set $y_t = x_3$.

If $X_3 = h^{x_1 x_2}$, then we have perfectly simulated $\mathtt{H}^{t-1}$, and otherwise we have perfectly simulated $\mathtt{H}^t$. Thus a PPT algorithm $\mathcal{A}$ that distinguishes between the hybrids $\mathtt{H}^{t-1}$ and $\mathtt{H}^t$ directly results in $\mathcal{B}$ being able to break SXDH with the same advantage.

□

□

<u>**Step 2:**</u>
Now, we provide the distinguisher with access to an oracle $\mathcal{RO}$ that behaves as follows:

- on input $(\mathbf{C}, D, \mathbf{C}', D', \pi)$, where $\mathbf{C}, \mathbf{C}'$ are from $\mathbb{G}_1^d$ and D, D' are from $\mathbb{G}_1$, it checks whether $\mathtt{I\text{-}Dec}(\mathsf{sk}, (\mathbf{C}, D, \mathbf{C}', D', \pi))$ produces a pair (id, m), where $m \in \mathcal{M}$ and $\mathsf{id} \in F^{-1}(\mathcal{T})$. If so, it behaves as the honest re-encryption functionality would and produces a valid encryption of m under public key $\widehat{\mathsf{pk}}_{F(\mathsf{id})}$. Otherwise it returns $[\widehat{F}, \widehat{G}, \widehat{H}]$, where $\widehat{F}, \widehat{G}$ are chosen at random from $\mathbb{G}_T$ and $\widehat{H}$ is chosen at random from $\mathbb{H}$.

Let $\mathsf{Real\text{-}RO}(\mathsf{D}, 1^\lambda, F, \mathcal{T})$ (resp. $\mathsf{Sim\text{-}RO}(\mathsf{D}, 1^\lambda, F, \mathcal{T})$) be the distribution which proceeds as in $\mathsf{Real\text{-}C}(\mathsf{D}, 1^\lambda, F, \mathcal{T})$ (resp. $\mathsf{Sim\text{-}C}(\mathsf{D}, 1^\lambda, F, \mathcal{T})$), but where the distinguisher is given access to $\mathcal{RO}$ rather than the real re-encryption circuit $\mathcal{C}$. I.e. replace step 5 with:

5. Let b be the output $\mathsf{D}^{\mathcal{RO}}(\mathsf{pk}, \widehat{\mathsf{pk}}_1, \cdots, \widehat{\mathsf{pk}}_n, (Z, \mathbf{A}, \mathbf{B}), \{\hat{a}_j\}_{j \in \mathcal{T}}, \{\mathbf{a}_{\mathsf{id}}\}_{\mathsf{id} \in F^{-1}(\mathcal{T})})$.

We consider the resulting distributions $\mathsf{Real\text{-}RO}(\mathsf{D}, 1^\lambda, F, \mathcal{T})$ and $\mathsf{Sim\text{-}RO}(\mathsf{D}, 1^\lambda, F, \mathcal{T})$, and show the following.

Lemma 2. *Assuming SXDH, for all $F, \mathcal{T}$, and all PPT algorithms D, distributions Real-RO$(\mathsf{D}, 1^\lambda, F, \mathcal{T})$ and Sim-RO$(\mathsf{D}, 1^\lambda, F, \mathcal{T})$ are indistinguishable.*

Proof. Note that the oracle $\mathcal{RO}$ can be perfectly simulated given access to $\{\mathbf{a}_{\mathsf{id}}\}_{\mathsf{id} \in F^{-1}(\mathcal{T})}$. (This is because decryption of a ciphertext $((\mathbf{C}, D), (\mathbf{C}', D'), \pi)$ simply verifies π, and then tests whether $D \cdot (C^{1/\mathbf{a}_i})^{-1}$ produces a valid message for each possible i. To test for $i \in F^{-1}(\mathcal{T})$, the oracle will only need to use the corresponding values of $\mathbf{a}_i$.) Hence, for every distinguisher $\mathsf{D}^{\mathcal{RO}}$, there exists a distinguisher D' that does not use the help of any oracle whose output distribution is identical to $\mathsf{D}^{\mathcal{RO}}$. D' simply simulates $\mathcal{RO}$ and internally runs D. Lemma 1 implies that $\mathsf{RealInput}(1^\lambda, F, \mathcal{T})$ and $\mathsf{SimInput}(1^\lambda, F, \mathcal{T})$ are indistinguishable and hence it also implies the lemma. □

<u>**Step 3:**</u>
We now argue that it doesn't matter whether D is given oracle access to the real re-encryption circuit $\mathcal{C}$ or to the $\mathcal{RO}$ oracle, even when it is also given access to the real obfuscated re-encryption program.

Lemma 3. *Assuming SXDH, for all $F, \mathcal{T}$, and all PPT algorithms D, distributions Real-C$(\mathsf{D}, 1^\lambda, F, \mathcal{T})$ and Real-RO$(\mathsf{D}, 1^\lambda, F, \mathcal{T})$ are indistinguishable.*

Proof. We prove this through a sequence of hybrid experiments that use hybrid oracles $\mathcal{O}_1, \cdots, \mathcal{O}_{n+1}$. In Hybrid experiment $\mathsf{Real\text{-}O}_i(\mathsf{D}, 1^\lambda, F, \mathcal{T})$, we give the distinguisher D, oracle access to hybrid $\mathcal{O}_i$. We now describe the oracle $\mathcal{O}_i$. $\mathcal{O}_i$ does the following: On input ciphertext $(\mathbf{C}, D, \mathbf{C}', D')$ if the ciphertext is an encryption of m with identity id, such that $F(\mathsf{id}) \geq i$, it outputs whatever $\mathcal{RO}$ outputs on the same input and otherwise outputs whatever $\mathcal{C}$ outputs on the same input. Note that $\mathcal{O}_{n+1} = \mathcal{C}$ and $\mathcal{O}_1 = \mathcal{RO}$.

Let $\mathsf{Real\text{-}O}_i(\mathsf{D}, 1^\lambda, F, \mathcal{T})$ be the distribution which proceeds as in $\mathsf{Real\text{-}C}(\mathsf{D}, 1^\lambda, F, \mathcal{T})$, but where the distinguisher is given access to $\mathcal{O}_i$ rather than the real re-encryption circuit $\mathcal{C}$.

We will show, under the SXDH assumption, that $\mathsf{Real\text{-}O}_i(\mathsf{D}, 1^\lambda, F, \mathcal{T})$ is indistinguishable from $\mathsf{Real\text{-}O}_{i+1}(\mathsf{D}, 1^\lambda, F, \mathcal{T})$ for all $1 \leq i \leq n$. Since $\mathcal{O}_{n+1} = \mathcal{C}$ and $\mathcal{O}_1 = \mathcal{RO}$, we have $\mathsf{Real\text{-}O}_{n+1}(\mathsf{D}, 1^\lambda, F, \mathcal{T}) = \mathsf{Real\text{-}C}(\mathsf{D}, 1^\lambda, F, \mathcal{T})$ and $\mathsf{Real\text{-}O}_1(\mathsf{D}, 1^\lambda, F, \mathcal{T}) = \mathsf{Real\text{-}RO}(\mathsf{D}, 1^\lambda, F, \mathcal{T})$, which will prove the lemma.

We now proceed to show that, given a distinguisher such that $\mathsf{Real\text{-}O}_i(\mathsf{D}, 1^\lambda, F, \mathcal{T})$ and $\mathsf{Real\text{-}O}_{i+1}(\mathsf{D}, 1^\lambda, F, \mathcal{T})$ produce noticeably different outputs, we will construct an adversary $\mathcal{A}$ that will break the SXDH assumption. Now, $\mathcal{A}$ takes as input a SXDH instance, which is a tuple $(\bar{h}, \bar{A} = \bar{h}^{\bar{a}}, \bar{R} = \bar{h}^{\bar{r}}, \bar{W})$, and has to decide whether $\bar{W} = \bar{h}^{\bar{a}\bar{r}}$ or $\bar{h}^{\bar{w}}$ for random $\bar{a}, \bar{r}, \bar{w} \in \mathbb{Z}_q$. $\mathcal{A}$ does the following:

1. $\mathcal{A}$ chooses random g from $\mathbb{G}$, and samples $\mathsf{sk} = (\mathbf{a}_1, \cdots, \mathbf{a}_d)$ at random from $\mathbb{Z}_q^d$, as well as $\hat{a}_j$ from $\mathbb{Z}_q$ for all $1 \leq j \leq n, j \neq i$. It generates a CRS crs for the NIZK proof system.

2. $\mathcal{A}$ sets $\mathsf{pk} = (\mathsf{crs}, g, g^{\mathbf{a}_1}, \cdots, g^{\mathbf{a}_d})$, $\widehat{\mathsf{pk}}_i = (\bar{h}, \bar{A})$, and $\widehat{\mathsf{pk}}_j = (\bar{h}, h^{\hat{a}_j})$, for $j \neq i$. $\mathcal{A}$ creates a valid re-encryption key $(Z, \mathbf{A}, \mathbf{B})$.

3. $\mathcal{A}$ runs $\mathsf{D}^{\mathcal{O}}(\mathsf{pk}, \widehat{\mathsf{pk}}_1, \cdots, \widehat{\mathsf{pk}}_n, (Z, \mathbf{A}, \mathbf{B}), \{\hat{a}_j\}_{j \in \mathcal{T}}, \{\mathbf{a}_{\mathsf{id}}\}_{\mathsf{id} \in F^{-1}(\mathcal{T})})$ where $\mathcal{O}$ is defined below.

   When D queries the oracle $\mathcal{O}$ on input $(\mathbf{C}, D, \mathbf{C}', D')$, $\mathcal{A}$ responds as below:

   (a) If input is not of the right format, or if the sanity check fails, then output $(\widehat{F}, \widehat{G}, \widehat{H})$ for random $\widehat{F}, \widehat{G} \in \mathbb{G}_T$, and random $\widehat{H} \in \mathbb{H}$.

   (b) Decrypt ciphertext using sk to obtain message m as well as id. If the decryption algorithm outputs $\perp$, then output a random tuple from $\mathbb{G}_T \times \mathbb{G}_T \times \mathbb{H}$. Otherwise continue as follows.

   (c) If $F(\mathsf{id}) \in \mathcal{T}$, proceed as in the real re-encryption program. If not, proceed as follows:

   (d) If $F(\mathsf{id}) \neq i$, then output whatever $\mathcal{O}_{i+1}$ does on this input.

   (e) If $F(\mathsf{id}) = i$, proceed as follows:

   - Re-randomize input ciphertexts as in the real re-encryption program: Pick a random exponent $t \leftarrow \mathbb{Z}_q$ and compute $\widehat{\mathbf{C}} = \mathbf{C}(\mathbf{C}')^t$ and $\widehat{D} = D(D')^t$.
   - Write $\widehat{\mathbf{C}} := (\widehat{C}_1, \ldots, \widehat{C}_d)$, $\mathbf{A} := (A_1, \ldots, A_d)$ and $\mathbf{B} := (B_1, \ldots, B_d)$. Again, we compute the main re-encryption step as in the real re-encryption program:

   $$F = \prod_{j=1}^{d} \mathsf{e}(\widehat{C}_j, A_j) \qquad \text{and} \qquad G = \prod_{j=1}^{d} \mathsf{e}(\widehat{C}_j, B_j) \cdot \mathsf{e}(\widehat{D}, Z)$$

   - Here we add an additional step: Pick random exponent v from $\mathbb{Z}_q$ and set:

   $$F' = F \cdot \mathsf{e}(g, \bar{W}^v) \qquad \text{and} \qquad G' = G \cdot \mathsf{e}(g, \bar{R}^v)$$

- Finally, we re-randomize the output ciphertext as in the real re-encryption program: Pick a random exponent $s \leftarrow \mathbb{Z}_q$ and compute $\widehat{F} = F'^s$, $\widehat{G} = G'^s$ and $\widehat{H} = H^s$.
- Output ciphertext $(\widehat{F}, \widehat{G}, \widehat{H})$.

4. $\mathcal{A}$ outputs whatever D outputs.

Let us consider the two cases: when the tuple that $\mathcal{A}$ receives is a SXDH instance and when it is a random instance.

Case 1: $\mathcal{A}$ receives an SXDH instance.

Now, when $\bar{W} = \bar{h}^{\bar{a}\bar{r}}$, we show that $\mathcal{A}$ perfectly simulates $\mathsf{Real\text{-}O}_{i+1}(\mathsf{D}, 1^{\lambda}, F, \mathcal{T})$, providing D with a real re-encryption key and oracle access to $\mathcal{O}_{i+1}$. Now, $(\bar{h}, \bar{A})$ can be interpreted as a randomly generated public key for the output encryption scheme. Hence the honest re-encryption key created $(Z, \mathbf{A}, \mathbf{B})$ has the correct distribution, so D receives the right input.

Now, let us consider $\mathcal{A}$'s responses to D's oracle queries. The only thing that $\mathcal{A}$ does differently is in steps 3e and 3d above. When D queries a ciphertext $(\mathbf{C}, D, \mathbf{C}', D')$ that is an encryption of message m with identity id, such that $F(\mathsf{id}) = i$, then $\mathcal{A}$ re-randomizes the ciphertext using elements $\mathsf{e}(g, \bar{W}^v = h^{\bar{a}\bar{r}v})$ and $\mathsf{e}(g, \bar{R}^v = h^{\bar{r}v})$. If the initial ciphertext used randomness r, r', then the result will be identical to running the real re-encryption program, but choosing $t' = t - \bar{r}/r'$ in the input re-randomization step. Thus $\mathcal{A}$ perfectly simulates oracle $\mathcal{O}_{i+1}$ in this case. When D queries a ciphertext $(\mathbf{C}, D, \mathbf{C}', D')$ that is an encryption of message m with identity id, such that $F(\mathsf{id}) \neq i$, $\mathcal{O}$ simply outputs whatever $\mathcal{O}_{i+1}$ does, thus simulating oracle $\mathcal{O}_{i+1}$ in this case as well. We conclude that when $\mathcal{A}$ receives an SXDH instance, it perfectly simulates $\mathsf{Real\text{-}O}_{i+1}(\mathsf{D}, 1^{\lambda}, F, \mathcal{T})$.

Case 2: $\mathcal{A}$ receives a random tuple.

Similarly, when the tuple that $\mathcal{A}$ receives is not a SXDH instance (and is random), then we show that $\mathcal{O}$ perfectly simulates $\mathcal{O}_i$. When D queries a ciphertext $(\mathbf{C}, D, \mathbf{C}', D')$ that is an encryption of message m with identity id, such that $F(\mathsf{id}) \neq i$ (or when the sanity check fails), then $\mathcal{O}$ outputs whatever $\mathcal{O}_{i+1}$ outputs, which is the same as what $\mathcal{O}_i$ outputs on such encryptions. We now consider the case when D queries a ciphertext $(\mathbf{C}, D, \mathbf{C}', D')$ that is an encryption of message m with identity id, such that $F(\mathsf{id}) = i$.

Let us denote $\mathbf{C} = g^{\boldsymbol{\sigma}}$, $D = g^{\nu}$, (similarly, $\mathbf{C}' = g^{\boldsymbol{\sigma}'}$,$D' = g^{\nu'}$), and $\bar{W} = g^{\bar{w}}$. Now, after the input re-encryption step, we obtain $\widehat{\mathbf{C}} = g^{(\boldsymbol{\sigma}+t\boldsymbol{\sigma}')}$, $\widehat{H} = g^{\nu+t\nu'}$. Hence, we have:

- $E \leftarrow \prod_{j=1}^{d} \mathsf{e}(\widehat{C}_j, A_j) = \mathsf{e}(g, h)^{z\sum_{j=1}^{d}\alpha_j(\sigma_j + t\sigma'_j)}$
- $G \leftarrow \mathsf{e}(\widehat{D}, Z) = \mathsf{e}(g, h)^{z(\nu + t\nu')}$

The final output of the oracle consists of terms $(\hat{F}, \hat{G}, \hat{H})$, where

- $\hat{F} = \mathsf{e}(g, h)^{w(\bar{w}vz + \sum_{j=1}^{d}\alpha_j(\sigma_j + t\sigma'_j))}$
- $\hat{G} = \mathsf{e}(g, h)^{w(z(\nu + t\nu') + \bar{r}v)}$.
- $\hat{H} = h^{zw}$.

$\hat{H}$ is uniformly random, since w is chosen at random from $\mathbb{Z}_q$. $\hat{F}, \hat{G}$ are uniformly random since t, v are random, and from the fact that $\nu' \neq 0$ (since the sanity check ensures that $H' \neq 1$). Hence $\mathcal{O}$ perfectly simulates $\mathcal{O}_i$ in this case. We conclude that, for all i, $\mathsf{Real\text{-}O}_i(\mathsf{D}, 1^\lambda, F, \mathcal{T}) \approx \mathsf{Real\text{-}O}_{i+1}(\mathsf{D}, 1^\lambda, F, \mathcal{T})$, and thus $\mathsf{Real\text{-}RO}(\mathsf{D}, 1^\lambda, F, \mathcal{T})$ is indistinguishable from $\mathsf{Real\text{-}C}(\mathsf{D}, 1^\lambda, F, \mathcal{T})$.

Combining everything, we get the proof of Lemma 3. □

Step 4:

Finally, we argue that no distinguisher can distinguisher the real re-encryption oracle from $\mathcal{RO}$, when given the simulated circuit (and the appropriate public and secret keys).

Lemma 4. *Assuming SXDH, for all $F, \mathcal{T}$, and all PPT algorithms D, distributions $\mathsf{Real\text{-}RO}(\mathsf{D}, 1^\lambda, F, \mathcal{T})$ and $\mathsf{Sim\text{-}RO}(\mathsf{D}, 1^\lambda, F, \mathcal{T})$ are indistinguishable.*

This follows trivially from Lemma 3, since the distinguisher there could easily ignore the obfuscated program he is given and run $\mathcal{S}$ to generate a simulated one instead.

Combining Lemmas 1, 2, 3, and 4 concludes the proof of Theorem 3.

□

B A Necessary Modification to the Encryption and Obfuscation Schemes

Let us assume that the corrupt recipient holds secret key $\widehat{\mathsf{sk}}_j$ and that $F(i) = j$. Now clearly, a corrupt recipient (that colludes with the re-encryption program) has the ability to decrypt input ciphertexts that have $\mathsf{id} = i$. That is, the corrupt recipient can simply take $\texttt{I-Enc}(\mathsf{pk}, i, m)$, feed this as input into the re-encryption program, obtain $\texttt{O-Enc}(\widehat{\mathsf{pk}}_j, m)$ as output and then decrypt this to obtain m. So, in order to simulate the re-encryption program correctly, the simulator would need to be able to produce encryptions of message m under $\widehat{\mathsf{pk}}_j$ whenever the input ciphertext has $\mathsf{id} = i = F^{-1}(j)$. This calls for some sort of selective decryption of input ciphertexts. Thus, we would like to ensure that a simulator can selectively decrypt ciphertexts with $\mathsf{id} = i$ such that $F(i) = j$, so that the simulator can ensure that the output ciphertext is correct. However, we would not want to give the simulator the ability to decrypt other input ciphertexts. Luckily for us, the secret key $\mathsf{sk}_i = \mathbf{a}_i$ is exactly such a key that allows for the selective decryption of messages with $\mathsf{id} = i$. In Appendix A, we "cheat" and endow our simulator with additional power by giving it sk_i as well. In this section we shall see how to remove this cheat. We would like to modify our construction so that the simulator, which knows $\widehat{\mathsf{sk}}_j$, can also learn sk_i using just the oracle access to the functional re-encryption functionality.

Our idea is to modify the input encryption scheme in the following manner. We will publish $k_i^* = \texttt{I-Enc}(\mathsf{pk}, i, \mathsf{sk}_i)$ as part of the input public key, so that the simulator can feed k_i^* to the re-encryption oracle, obtain $\texttt{O-Enc}(\widehat{\mathsf{pk}}_j, \mathsf{sk}_i)$ as output, and, using knowledge of $\widehat{\mathsf{sk}}_j$, recover sk_i. At a first glance, it seems that this might require the input encryption scheme to be circular secure. However, we note that this need not be the case. We only need to publish a specific string k_i^*, as part of the public key, that "denotes" an encryption of sk with $\mathsf{id} = i$. Hence, we modify the input encryption scheme so that the public key now includes randomly chosen k_i^* for all $i \in \mathcal{D}$. Furthermore, we modify the input decryption algorithm so that it will now check if the input ciphertext is k_i^* for any $i \in \mathcal{D}$ and if so output sk_i; otherwise the input decryption algorithm works as before. This modification now allows the simulator to learn sk_i, using which the simulator can construct an appropriate re-encryption key and program.

Next, we need to ensure that the real re-encryption program outputs an encryption of sk_i under $\widehat{\mathsf{pk}}_j$ when fed with k_i^* (as the adversary now also has access to a string that denotes an encryption of sk_i with $\mathsf{id} = i$). To do this, we will modify the re-encryption scheme to include $q_i = \texttt{O-Enc}(\widehat{\mathsf{pk}}_{F(i)}, \mathsf{sk}_i)$ for all $i \in \mathcal{D}$ as part of the re-encryption key. Next, we modify the re-encryption program so that now, for all $i \in \mathcal{D}$, on input k_i^*, the program will re-randomize q_i and output this instead. On all other input ciphertexts, the re-encryption program works as before.

We note that the proof in section A implies that the modified scheme is a collusion-resistant secure obfuscation. To see this, note that given $\{\mathsf{sk}_{\mathsf{id}} = \mathbf{a}_{\mathsf{id}}\}_{\mathsf{id} \in F^{-1}(\mathcal{T})}$, one can easily simulate access to an oracle for the modified encryption functionality. (The only difference is that when the adversary queries the oracle on one of the values k_i^* included in the public key, we compute and return $\texttt{O-Enc}(\widehat{\mathsf{pk}}_{F(i)}, \mathsf{sk}_i)$.) Similarly, given access to the modified oracle, one can easily compute sk_i for all $i \in F^{-1}(\mathcal{T})$ simply by calling the oracle to obtain $\texttt{O-Enc}(\widehat{\mathsf{pk}}_{F(i)}, \mathsf{sk}_i)$ and decrypting the result. Thus, any distinguisher that can distinguish the real and simulated game for this modfied re-encryption scheme can distinguish equally well between games $\mathsf{Real\text{-}C}(\mathsf{D}, 1^\lambda, F, \mathcal{T})$ and $\mathsf{Sim\text{-}C}(\mathsf{D}, 1^\lambda, F, \mathcal{T})$, so the proof in section A implies security for the modified scheme.

C Semantic Security of Input and Output Encryption schemes based on SXDH

We show that the input encryption scheme (with public key pk) is semantically secure (i.e., it hides both the message and the "identity") under the SXDH assumption. Furthermore, we show that this holds even if the adversary is given oracle access to a functional re-encryption oracle for randomly chosen output public keys $\widehat{\mathsf{pk}}_1, \ldots, \widehat{\mathsf{pk}}_n$ and an adversarially chosen policy function F, as well as a subset of the corresponding secret keys $\{\widehat{\mathsf{sk}}_j\}_{j \in \mathcal{T}}$ (where the "corrupted set" $\mathcal{T}$ is arbitrary, but independent of the keys). Of course, we cannot guarantee any security if the challenge ciphertext is encrypted using a identity $i \in [d]$ such that $F(i) \in \mathcal{T}$. This is simply because the adversary can use her oracle to re-encrypt the challenge ciphertext under the public key $\widehat{\mathsf{pk}}_{F(i)}$, and then recover the message using the secret key $\widehat{\mathsf{sk}}_{F(i)}$. Thus, informally, we only require that semantic security holds as long as the identity i used in the challenge ciphertext is such that $F(i) \notin \mathcal{T}$.

Conceptually, the proof proceeds in two steps: first, we show that the extra ability that the adversary obtains (in the form of the functional re-encryption oracle and some of the output secret keys) can be simulated using the knowledge of some "relevant parts" of the input secret key sk. More precisely, recall that the input secret key sk is composed of d vectors $\mathbf{a}_i \in \mathbb{Z}_q^d$, one for each $i \in [d]$. We show that the view of an adversary who knows the output secret key $\widehat{\mathsf{sk}}_j$ (and has access to the functional re-encryption oracle) can be simulated using the "keys" $\{\mathbf{a}_i \ : \ F(i) = j\}$.

Thus, it suffices to show that $\texttt{I-Enc}(\mathsf{pk}, i_0, m_0) \overset{c}{\approx} \texttt{I-Enc}(\mathsf{pk}, i_1, m_1)$, even given the vectors $\mathbf{a}_t$ for all $t \in [d] \setminus \{i_0, i_1\}$. This follows easily from the DDH assumption over the group $\mathbb{G}$. Recall that a ciphertext of the identity-message pair (i_b, m_b) under the input encryption scheme consists of the pair $(\mathbf{E}, \mathbf{E}')$ where $\mathbf{E} = (g^{r\mathbf{a}_{i_b}}, g^r m_b)$ for a uniformly random r (and $\mathbf{E}'$ is the corresponding encryption of 0). Now, under the DDH assumption over $\mathbb{G}$, $\mathbf{E}$ looks like a tuple of uniformly random group elements, which hides both the "identity" i_b as well as the message m_b.

Semantic security of the output encryption scheme (with public key $\widehat{\mathsf{pk}}$) follows directly from the DDH assumption in the group $\mathbb{H}$. Note that the presence of a re-encryption oracle (with $\widehat{\mathsf{pk}}$ as one of the output public keys) does not affect the semantic security since the oracle can be simulated using just $\widehat{\mathsf{pk}}$ (and in particular, without any knowledge of $\widehat{\mathsf{sk}}$).

We now present the proofs in detail.

C.1 Security of the Input Encryption Scheme

We first formally define what we mean by the security of the input encryption scheme. Roughly speaking, we would like the input encryption scheme with respect to a public key pk to be semantically secure (i.e., it hides both the "identity" and the message). Furthermore, we would like the semantic security to hold even if the adversary is given access to a functional re-encryption oracle for some (adversarially specified) function $F : [d] \to [n]$, randomly chosen output public keys $\widehat{\mathsf{pk}}_1, \ldots, \widehat{\mathsf{pk}}_n$, as well as a subset of the corresponding secret keys $\{\widehat{\mathsf{sk}}_j\}_{j \in \mathcal{T}}$. Of course, we cannot guarantee any security if the "challenge" id^* is such that $j = F(\mathsf{id}^*) \in \mathcal{T}$. This is simply because the adversary can first get the challenge ciphertext re-encrypted under $\widehat{\mathsf{pk}}_j$ (using the re-encryption oracle) and then use the secret key $\widehat{\mathsf{sk}}_j$ to recover the message. Thus, informally, we require security to hold as long as this event does not happen which leads us to the notion of *collusion-resistant indistinguishability* of encryptions, defined below.

Definition 3 (CR-Indistinguishability of Encryptions)**.** *Let* $\Pi_I = (\texttt{I-Gen}, \texttt{I-Enc}, \texttt{I-Dec})$ *be the input encryption scheme, and* $\Pi_O = (\texttt{O-Gen}, \texttt{O-Enc}, \texttt{O-Dec})$ *be the output encryption scheme. Let the random variable* $\mathsf{CR\text{-}IND}_b(\Pi_I, \Pi_O, F, A, \lambda)$*, where* $F : [d] \to [n]$ *is a function,* $b \in \{0,1\}$*,* $A = (A_1, A_2, A_3)$ *is a tuple of p.p.t. algorithms and* $\lambda \in \mathbb{N}$*, denote the result of the following probabilistic experiment: (Let* $\mathcal{C} := \mathcal{C}_{F,\mathsf{sk},\widehat{\mathsf{pk}}_1,\ldots,\widehat{\mathsf{pk}}_n}$ *denote the functional re-encryption functionality for an input key-pair* $(\mathsf{pk}, \mathsf{sk})$ *and output public keys* $\widehat{\mathsf{pk}}_1, \ldots, \widehat{\mathsf{pk}}_n$*.)*

```
$\mathsf{CR\text{-}IND}_b(\Pi_I, \Pi_O, F, A, \lambda)$ :

  $(\mathcal{T} \subseteq [n], \mathsf{state}_1) \leftarrow A_1(1^\lambda)$
  $(\mathsf{pk}, \mathsf{sk}) \leftarrow \texttt{I-Gen}(1^\lambda, 1^d)$
  $(\widehat{\mathsf{pk}}_j, \widehat{\mathsf{sk}}_j) \leftarrow \texttt{O-Gen}(1^\lambda)$ for all $j \in [n]$;  Let $\mathcal{C} := \mathcal{C}_{F,\mathsf{sk},\widehat{\mathsf{pk}}_1,\ldots,\widehat{\mathsf{pk}}_n}$
  $(m_0, m_1, \mathsf{id}_0, \mathsf{id}_1, \mathsf{state}_2) \leftarrow A_2^{\mathcal{C}}(\mathsf{pk}, \widehat{\mathsf{pk}}_1, \ldots, \widehat{\mathsf{pk}}_n, \{\widehat{\mathsf{sk}}_j\}_{j \in \mathcal{T}}, \mathsf{state}_1)$
        s.t. $|m_0| = |m_1|$ and $\mathsf{id}_0, \mathsf{id}_1 \in [d]$ and $F(\mathsf{id}_0), F(\mathsf{id}_1) \notin \mathcal{T}$
  $y \leftarrow \texttt{I-Enc}_{\mathsf{pk}}(m_b, \mathsf{id}_b)$
  $b' \leftarrow A_3^{\mathcal{C}}(y, \mathsf{state}_2)$
  Output $b'$
```

$(\texttt{I-Gen}, \texttt{I-Enc}, \texttt{I-Dec})$ *satisfies* collusion-resistant indistinguishability *under a chosen-plaintext attack with respect to* $(\texttt{O-Gen}, \texttt{O-Enc}, \texttt{O-Dec})$ *if* $\forall$ *p.p.t. algorithms* $A = (A_1, A_2, A_3)$*, all* $d = d(\lambda) \in \mathbb{N}$*,* $n = n(\lambda) \in \mathbb{N}$*, and all functions* $F : [d] \to [n]$*, the advantage of* A*, defined as below is negligible:*

$$\mathsf{Adv}(\Pi_I, \Pi_O, F, A, \lambda) \stackrel{\Delta}{=} \left| \Pr[\mathsf{CR\text{-}IND}_0(\Pi_I, \Pi_O, F, A, \lambda) = 1] - \Pr[\mathsf{CR\text{-}IND}_1(\Pi_I, \Pi_O, F, A, \lambda) = 1] \right|$$

Note that ultimately we would like to achieve a stronger security guarantee by guaranteeing the indistinguishability of encryptions in the experiment above, even when the adversary A_2 is given the real functional re-encryption program (as opposed to oracle access to the functional re-encryption functionality). However, showing the theorem stated below will suffice for us, as we can combine this together with Theorem 2 to obtain the stronger security guarantee.

Theorem 4. *Under the SXDH assumption, the input encryption scheme* $\Pi_I = (\texttt{I-Gen}, \texttt{I-Enc}, \texttt{I-Dec})$ *when modified as described in appendix B satisfies Definition 3 (collusion-resistant indistinguishability under CPA attacks).*

Proof. Define the following oracle $\mathcal{RO}$ that behaves as follows:

- On input $(\mathbf{C}, D, \mathbf{C}', D', \pi)$, where all the vectors are from $\mathbb{G}_1^d$ and $D, D' \in \mathbb{G}$, it checks whether $\texttt{I-Dec}(\mathsf{sk}, (\mathbf{C}, D, \mathbf{C}', D', \pi))$ produces (id, m) for any $m \in \mathcal{M}$ and $\mathsf{id} \in F^{-1}(\mathcal{T})$. If so, it behaves as the honest re-encryption functionality would and produces a valid encryption of m under public key $\widehat{\mathsf{pk}}_{F(\mathsf{id})}$. Otherwise it returns $[\widehat{E}, \widehat{G}, \widehat{H}]$, where $\widehat{E}, \widehat{G}$ are chosen at random from $\mathbb{G}_T$ and $\widehat{H}$ is chosen at random from $\mathbb{H}$.

- On input k^*_{id} (for one of the k^*_i values included in pk), if $F(\mathsf{id}) \in \mathcal{T}$ it returns $\texttt{O-Enc}(\widehat{\mathsf{pk}}_{F(\mathsf{id})}, \mathsf{sk}_{\mathsf{id}})$.[7] If $F(\mathsf{id}) \notin \mathcal{T}$, it returns $[\widehat{E}, \widehat{G}, \widehat{H}]$, where $\widehat{E}, \widehat{G}$ are chosen at random from $\mathbb{G}_T$ and $\widehat{H}$ is chosen at random from $\mathbb{H}$.

First, consider the following experiments $H^1_b, b = 0, 1$ which is identical to $\mathsf{CR\text{-}IND}_b$, except that adversary A_2 gets oracle access to $\mathcal{RO}$ instead of $\mathcal{C}$. Formally,

```
$H^1_b(\Pi_I, \Pi_O, F, A, \lambda)$ :

  $(\mathcal{T} \subseteq [n], \mathsf{state}_1) \leftarrow A_1(1^\lambda)$
  $(\mathsf{pk}, \mathsf{sk}) \leftarrow \texttt{I-Gen}(1^\lambda, 1^d)$
  $(\widehat{\mathsf{pk}}_j, \widehat{\mathsf{sk}}_j) \leftarrow \texttt{O-Gen}(1^\lambda)$ for all $j \in [n]$;  Let $\mathcal{RO} := \mathcal{RO}_{F,\mathsf{sk},\widehat{\mathsf{pk}}_1,\ldots,\widehat{\mathsf{pk}}_n}$
  $(m_0, m_1, \mathsf{id}_0, \mathsf{id}_1, \mathsf{state}_2) \leftarrow A_2^{\mathcal{RO}}(\mathsf{pk}, \widehat{\mathsf{pk}}_1, \ldots, \widehat{\mathsf{pk}}_n, \{\widehat{\mathsf{sk}}_j\}_{j \in \mathcal{T}}, \mathsf{state}_1)$
        s.t. $|m_0| = |m_1|$ and $\mathsf{id}_0, \mathsf{id}_1 \in [d]$ and $F(\mathsf{id}_0), F(\mathsf{id}_1) \notin \mathcal{T}$
  $y \leftarrow \texttt{I-Enc}_{\mathsf{pk}}(m_b, \mathsf{id}_b)$
  $b' \leftarrow A_3^{\mathcal{RO}}(y, \mathsf{state}_2)$
  Output $b'$
```

Proposition 1. *∀ p.p.t algorithms* $A = (A_1, A_2, A_3)$*, all* $d = d(\lambda) \in \mathbb{N}$*,* $n = n(\lambda) \in \mathbb{N}$*, all functions* $F : [d] \to [n]$*, and all* $b \in \{0, 1\}$*:*

$$\left| \Pr[H^1_b(\Pi_I, \Pi_O, F, A, \lambda) = 1] - \Pr[\mathsf{CR\text{-}IND}_b(\Pi_I, \Pi_O, F, A, \lambda) = 1] \right|$$

is negligible.

[7] Recall that in our scheme, the partial decryption key $\mathsf{sk}_{\mathsf{id}}$ is $\mathbf{a}_{\mathsf{id}}$.

Proof. Lemma 3 shows that the output of $\mathcal{C}$ is indistinguishable from the output of $\mathcal{RO}$ even given the real re-encryption program. The proof of this proposition trivially follows from this lemma and from the semantic security of the output encryption scheme (shown in section C.2). $\square$

Next, consider the following experiments $H_b^2, b = 0, 1$ which is identical to H_b^1, except that adversaries A_2, A_3 get partial decryption keys for the input encryption scheme $\{\mathsf{sk}_i\}_{i \in F^{-1}(\mathcal{T})} = \{\mathbf{a}_i\}_{i \in F^{-1}(\mathcal{T})}$, but do not get any oracle access. Formally,

```
$H_b^2(\Pi_I, \Pi_O, F, A, \lambda)$ :

  $(\mathcal{T} \subseteq [n], \mathsf{state}_1) \leftarrow A_1(1^\lambda)$
  $(\mathsf{pk}, \mathsf{sk}) \leftarrow \texttt{I-Gen}(1^\lambda, 1^d)$
  $(\widehat{\mathsf{pk}}_j, \widehat{\mathsf{sk}}_j) \leftarrow \texttt{O-Gen}(1^\lambda)$ for all $j \in [n]$
  $(m_0, m_1, \mathsf{id}_0, \mathsf{id}_1, \mathsf{state}_2) \leftarrow A_2(\mathsf{pk}, \widehat{\mathsf{pk}}_1, \ldots, \widehat{\mathsf{pk}}_n, \{\widehat{\mathsf{sk}}_j\}_{j \in \mathcal{T}}, \{\mathsf{sk}_i\}_{i \in F^{-1}(\mathcal{T})}, \mathsf{state}_1)$
        s.t. $|m_0| = |m_1|$ and $\mathsf{id}_0, \mathsf{id}_1 \in [d]$ and $F(\mathsf{id}_0), F(\mathsf{id}_1) \notin \mathcal{T}$
  $y \leftarrow \texttt{I-Enc}_{\mathsf{pk}}(m_b, \mathsf{id}_b)$
  $b' \leftarrow A_3(y, \mathsf{state}_2)$
  Output $b'$
```

Proposition 2. *$\forall$ p.p.t algorithms $A = (A_1, A_2, A_3)$, all $d = d(\lambda) \in \mathbb{N}$, $n = n(\lambda) \in \mathbb{N}$, all functions $F : [d] \to [n]$, and all $b \in \{0, 1\}$:*

$$\left| \Pr[H_b^2(\Pi_I, \Pi_O, F, A, \lambda) = 1] - \Pr[H_b^1(\Pi_I, \Pi_O, F, A, \lambda) = 1] \right| = 0$$

.

Proof. The proof of this proposition follows from the fact that (1) the reduction in H_b^2 can perfectly simulate $\mathcal{RO}$ to adversary A_2, by simply decrypting the input ciphertext using all secret keys $\{sk_i\}_{i \in F^{-1}(\mathcal{T})}$ (If some decryption succeeds, then the simulator honestly performs the re-encryption and otherwise returns random group elements like $\mathcal{RO}$ does.), and (2) the reduction in H_b^1 can easily produce sk_i for $i \in F^{-1}(\mathcal{T})$ just by sending a k_i^* query to $\mathcal{RO}$, and decrypting the resulting ciphertext using $\widehat{\mathsf{sk}}_{F(i)}$. $\square$

At this point, we can show that H_0^2 is indistinguishable from H_1^2 by showing that $\texttt{I-Enc}(\mathsf{pk}, \mathsf{id}_0, m_0) \overset{c}{\approx} \texttt{I-Enc}(\mathsf{pk}, \mathsf{id}_1, m_1)$ for any $\mathsf{id}_0, \mathsf{id}_1 \notin F^{-1}(\mathcal{T})$.

Proposition 3. *Assuming SXDH, $\forall$ p.p.t algorithms $A = (A_1, A_2, A_3)$, all $d = d(\lambda) \in \mathbb{N}$, $n = n(\lambda) \in \mathbb{N}$, all functions $F : [d] \to [n]$:*

$$\left| \Pr[H_0^2(\Pi_I, \Pi_O, F, A, \lambda) = 1] - \Pr[H_1^2(\Pi_I, \Pi_O, F, A, \lambda) = 1] \right|$$

is negligible.

Proof. The proof of this proposition is shown via a sequence of intermediary hybrid experiments.

First, consider a hybrid experiment H_b^3, which is identical to H_b^2, except that A_3 gets as input a valid ciphertext y that now contains a simulated zero-knowledge proof, that the ciphertext is valid, instead of the real zero-knowledge proof. Such a simulated zero-knowledge proof can be provided by the simulator using the trapdoor to crs. By the computational zero-knowledge of the proof system, it follows that experiment H_b^3 is indistinguishable from experiment H_b^2.

Next, consider a hybrid experiment H^4, which is identical to H_b^3, except that A_3 gets as input random group elements (as opposed to a valid ciphertext y), along with the simulated zero-knowledge proof. We will now show that H_b^3 is indistinguishable from H^4 for $b = 0, 1$. We shall show that if an adversary $A = (A_1, A_2, A_3)$ can distinguish between H^4 and H_b^3, then we can construct an adversary $\mathcal{B}$ that can break SXDH. To do this, we will go through a sequence of hybrids from H_b^3 to H^4. Note that $y = (\mathbf{C}, D, \mathbf{C}', D')$ where $\mathbf{C}, \mathbf{C}' \in \mathbb{G}^d$ and $D, D' \in \mathbb{G}$. In hybrid W_j (for $0 \le j \le d$), adversary A_3 will be given the first j elements of $\mathbf{C}$ as well as group element D correctly and the remaining $d - j$ elements of C will be random elements. In hybrid V_j (for $0 \le j \le d$), the adversary A_3 will be given random $\mathbf{C}, D$, correctly generated D' and $\mathbf{C}'$ such that the first the first j elements are generated correctly and the remaining $d - j$ are chosen at random. Note that $W_d = H_0^3$, $W_0 = V_d$, and $V_0 = H^4$. We will now show the indistinguishability of W_j and W_{j+1} for any $0 \le j \le d$. (Indistinguishability of V_j and V_{j+1} follows from a very similar argument.)

Assume that adversary $A = (A_1, A_2, A_3)$ can distinguish between W_j and W_{j+1}. $\mathcal{B}$ receives as input a tuple $(\bar{g}, \bar{A} = \bar{g}^{\bar{a}}, \bar{B} = \bar{g}^{\bar{b}}, \bar{C} = \bar{g}^{\bar{c}})$ and has to decide whether $\bar{c} = \bar{a}\bar{b}$ or random. $\mathcal{B}$ receives $(\mathcal{T}, \mathsf{state}_1)$ from A_1. $\mathcal{B}$ sets $g = \bar{g}$, and picks random vectors $\mathsf{sk}_i = \mathbf{a}_i$ for all $i \in \mathcal{T}$. For each $i \notin \mathcal{T}$ $\mathcal{B}$ proceeds as follows: it chooses $a_{ik} \in Z_q$ at random for all $k \neq j$. Then it chooses random ω_i and implicitly sets $a_{ij} = \omega_i \bar{a}$. Finally, it uses these values to compute $g^{\mathbf{a}_i}$ (using $\bar{A}^{\omega_i}$ in place of $g^{a_{ij}}$). Let $\mathsf{pk} = g^{\mathbf{a}_1}, \dots g^{\mathbf{a}_d}$ for the values derived this way.

$\mathcal{B}$ now provides A_2 with $(\mathsf{pk}, \{\widehat{\mathsf{sk}}_j\}_{j \in \mathcal{T}}, \{\mathsf{sk}_i\}_{i \in F^{-1}(\mathcal{T})}, \mathsf{state}_1)$. $\mathcal{B}$ receives $(m_0, m_1, \mathsf{id}_0, \mathsf{id}_1, \mathsf{state}_2)$ from A_2. If $m_0 = \mathsf{sk}_i$ or $m_1 = \mathsf{sk}_i$ for some $i \notin F^{-1}(\mathcal{T})$, $\mathcal{B}$ aborts. (This should happen only with negligible probability, because A is only ever given $g^{\mathsf{sk}_i} = g^{\mathbf{a}_i}$; if A can produce $\mathsf{sk}_i = \mathbf{a}_i$, then A breaks the discrete logarithm assumption.) For all other messages, $\mathcal{B}$ chooses random $r' \in \mathbb{Z}_q$ and random $R_{j+1}, \dots, R_d \in \mathbb{G}$, and creates the ciphertext as $\mathbf{C} = (\bar{B}^{a_{\mathsf{id}_b 1}}, \cdots, \bar{B}^{a_{\mathsf{id}_b (j-1)}}, \bar{C}, R_{j+1}, \cdots, R_d)$, $D = \bar{B}m_0$, $\mathbf{C}' = (\bar{g}^{a_{\mathsf{id}_b 1} r'}, \dots, \bar{g}^{a_{\mathsf{id}_b (j-1)} r'}, \bar{A}^{\omega_i r'}, \bar{g}^{a_{\mathsf{id}_b (j+1)} r'}, \bar{g}^{a_{\mathsf{id}_b d} r'})$, and $D' = \bar{g}^{r'}$. $\mathcal{B}$ sends $(\mathbf{C}, D, \mathbf{C}', D')$ as y to A_3 along with a simulated proof that y is a valid ciphertext. Now, clearly, if the tuple that $\mathcal{B}$ receives is a SXDH tuple, then the ciphertext that A_3 receives is according to W_{j+1} and if the tuple that $\mathcal{B}$ receives is random, then the ciphertext that A_3 receives is according to W_j. Hence, if $A = (A_1, A_2, A_3)$ can distinguish between W_j and W_{j+1}, then $\mathcal{B}$ can distinguish between a SXDH tuple and random tuple. Thus, W_j is indistinguishable from W_{j+1} for all $0 \le j \le d - 1$. Similarly, we can show that each V_j is indistinguishable from V_{j+1}. We conclude, by DDH, H_b^3 is indistinguishable from H^4 for both values of b, so H_0^3 is indistinguishable from H_1^3.

Recall that we argued that it follows from the computational zero-knowledge property of the proof system that hybrid H_b^3 is indistinguishable from hybrid H_b^2. Hence, we have H_0^2 is indistinguishable from H_1^2, thus proving the lemma.

□

Combining Propositions 1, 2 and 3 gives us the proof of Theorem 4.

□

C.2 Security of the Output Encryption Scheme

We show the semantic security of the output encryption scheme assuming SXDH. In other words, we show that for any two messages m_0, m_1, the distributions $\texttt{O-Enc}(\widehat{\mathsf{pk}}, m_0)$ and $\texttt{O-Enc}(\widehat{\mathsf{pk}}, m_1)$ are computationally indistinguishable.[8]

Theorem 5. *Assuming SXDH, for a randomly chosen public key $\widehat{\mathsf{pk}}$, and for any two messages m_0, m_1, the distributions $\texttt{O-Enc}(\widehat{\mathsf{pk}}, m_0)$ and $\texttt{O-Enc}(\widehat{\mathsf{pk}}, m_1)$ are computationally indistinguishable.*

Proof. The semantic security of the output encryption scheme follows directly from the hardness of the DDH assumption in the group $\mathbb{H}$.

More formally, we show that if an adversary $\mathcal{A}$ can distinguish an encryption of m_0 from an encryption of m_1 under $\widehat{\mathsf{pk}}$ with non-negligible probability, then we can construct an adversary $\mathcal{A}'$ that will break the SXDH assumption with advantage ϵ as well. $\mathcal{A}'$ works as follows:

- $\mathcal{A}'$ receives as input a tuple $(h, A = h^{\hat{a}}, R = h^r, W)$ (where h is a random generator of the group $\mathbb{H}$, and $\hat{a}, r$ are random exponents). The goal of $\mathcal{A}'$ is to determine whether $W = h^{\hat{a}r}$ or not.
- $\mathcal{A}'$ picks a random generator g of group $\mathbb{G}$ and sends the public key $\widehat{\mathsf{pk}} = (g, h, h^{\hat{a}})$ to $\mathcal{A}$.
- On receiving two messages m_0 and m_1 from $\mathcal{A}$, $\mathcal{A}'$ flips a bit b at random and picks an exponent s at random from $\mathbb{Z}_q$. $\mathcal{A}'$ sends the ciphertext

  $$C_b := (\mathsf{e}(g^s, W), e(g^s, R) \cdot \mathsf{e}(m_b, h^s), h^s)$$

  as the encryption of m_b to $\mathcal{A}$.
- Now $\mathcal{A}$ replies with a bit b'. $\mathcal{A}'$ simply outputs 1 if $b = b'$ (i.e., guessing that $W = h^{\hat{a}r}$) and outputs a random bit otherwise (i.e., guessing that W is random).

It is easy to see that when W is random, the ciphertext C_b is independent of b and hence the success probability of $\mathcal{A}$ in this case is exactly $\frac{1}{2}$.

In the case when $W = h^{\hat{a}r}$, the ciphertext C_b has the same distribution as $\texttt{O-Enc}(\widehat{\mathsf{pk}}, m_b)$. Hence, the adversary $\mathcal{A}$ has advantage at least ϵ. It is easy to see that $\mathcal{A}'$ succeeds in determining whether $W = h^{\hat{a}r}$ with non-negligible advantage. □

D Functional Re-encryption and General Predicate Obfuscation

In this section, we show a connection between *collusion-resistant* (average-case) secure obfuscation of functional re-encryption and program obfuscation of the family of functions $\mathcal{F}_\lambda = \{F : \mathcal{D}_\lambda \to \mathcal{R}_\lambda\}_{\lambda>0}$ satisfying the predicate obfuscation definition of Barak et al. [BGI+01].

More precisely, we show that if there is an obfuscator for the circuit family $\mathsf{FR}_{\mathcal{F}}$ that achieves the notion of average-case secure obfuscation against collusion, then there is an obfuscator for the

[8] Note that if we show the plain semantic security of the output encryption scheme with public key $\widehat{\mathsf{pk}}$, it automatically follows that the semantic security holds even in the presence of a functional re-encryption oracle for which $\widehat{\mathsf{pk}}$ is an *output public key*. This is simply because the functionality of such a functional re-encryption oracle can be implemented using just $\widehat{\mathsf{pk}}$, and in particular, without any knowledge of the secret key $\widehat{\mathsf{sk}}$.

circuit family $\mathcal{F}$ that achieves the predicate obfuscation definition of Barak et al. [BGI$^+$01]. Since there is no general-purpose obfuscator that satisfies the predicate obfuscation definition [BGI$^+$01], this shows that one cannot have a general purpose obfuscator (satisfying the definition of collusion resistance) for functional re-encryption for *general functions* (with large domain).

Predicate Obfuscation. We define a slightly relaxed notion of predicate obfuscation where the obfuscated program is a probabilistic circuit which is allowed to err with negligible probability on every input. This relaxation, called "approximate functionality", was already considered in [BGI$^+$01]. In particular, their impossibility result holds even with such a relaxation.

Definition 4 (Predicate Obfuscation [BGI$^+$01])**.** *An efficient algorithm $\mathcal{O}$ is a* predicate obfuscator *for the family $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda>0}$, if it has the following properties:*

- *(Approximately) Preserving Functionality: There exists a negligible function $\texttt{neg}(\lambda)$, s.t. for all input lengths λ, for any $C \in \mathcal{C}_\lambda$:*

$$\Pr[(\mathcal{O}(C))(x) \neq C(x)] \leq \texttt{neg}(\lambda)$$

  The probability is taken over a uniformly random choice of $x \in \{0,1\}^\lambda$ and over $\mathcal{O}$'s random coins.

- *Polynomial Slowdown: There exists a polynomial $p(\lambda)$ such that for sufficiently large input lengths λ, for any $C \in \mathcal{C}_\lambda$, the obfuscator $\mathcal{O}$ only enlarges C by a factor of p: $|\mathcal{O}(C)| \leq p(|C|)$.*

- *Predicate Virtual Black-box: For every polynomial sized adversary circuit $\mathcal{A}$, there exists a polynomial size simulator circuit $\mathcal{S}$ and a negligible function $\texttt{neg}(\lambda)$, such that for every input length λ, for every $C \in \mathcal{C}_\lambda$:*

$$\left|\Pr[\mathcal{A}(\mathcal{O}(C)) = 1] - \Pr[\mathcal{S}^C(1^\lambda) = 1]\right| \leq \texttt{neg}(\lambda)$$

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

We show the connection between functional re-encryption for a family of functions $\mathcal{F}$ and predicate obfuscation of $\mathcal{F}$ below:

Theorem 6. *Assume that there is a semantically secure encryption scheme $\mathcal{E} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$, a family of functions $\mathcal{F}_\lambda = \{F : \mathcal{D}_\lambda \to \{0,1\}\}$ and an obfuscator $\mathcal{O}$ for the family $\mathcal{C}_{\lambda,\mathcal{F}} = \left\{C_{F,\mathsf{pk},\mathsf{sk},\mathsf{pk}_0,\mathsf{pk}_1} : F \in \mathcal{F}_\lambda\right\}$ that satisfies the notion of collusion-resistant average-case secure obfuscation. Then, there is a predicate obfuscator for the family $\mathcal{F}_\lambda$ that satisfies the definition above.*

Proof. Let $\mathcal{O}$ be a collusion-resistant average-case secure obfuscator for the family $\mathcal{C}_\mathcal{F}$. The obfuscator $\mathcal{O}_\mathcal{F}$ for the family of functions $\mathcal{F}_\lambda = \left\{F : \mathcal{D}_\lambda \to \{0,1\}\right\}_{\lambda>0}$, on input a function $F \in \mathcal{F}_\lambda$, proceeds as follows:

- Choose an input public/secret key pair $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda)$ and two output public/secret key pairs $(\mathsf{pk}_b, \mathsf{sk}_b) \leftarrow \mathsf{Gen}(1^\lambda)$ (for $b \in \{0,1\}$).
- Run the obfuscator $\mathcal{O}$ on input the functional re-encryption circuit $C_{F,\mathsf{pk},\mathsf{sk},\mathsf{pk}_0,\mathsf{pk}_1}$ to get an obfuscated program Ψ.

- Output $\Psi' = (\Psi, \mathsf{pk}, \mathsf{sk}_0)$ as the obfuscated program for F.

On input $x \in \mathcal{D}_\lambda$, the obfuscated program Ψ' works as follows:

- Choose a uniformly random message $m \leftarrow \mathcal{M}$.
- Compute a ciphertext $c \leftarrow \mathsf{Enc}(\mathsf{pk}, \mathsf{id} = x, m)$ and let $\hat{c} \leftarrow \Psi(c)$ be the output of the obfuscated re-encryption program Ψ on input c.
- Let $m' \leftarrow \mathsf{Dec}(sk_0, \hat{c})$. If $m' = m$, output 0, else output 1.

Since both the obfuscator $\mathcal{O}$ and the obfuscated program Ψ run in time $\mathsf{poly}(|F|, \lambda)$, so does the obfuscator $\mathcal{O}_{\mathcal{F}}$ and the obfuscated program Ψ'. In Claim 2, we show that $\mathcal{O}_{\mathcal{F}}$ indeed computes F correctly (i.e., preserves functionality) and in Claim 3, we show that $\mathcal{O}_{\mathcal{F}}$ satisfies the predicate virtual black-box property.

Claim 2. *The obfuscator $\mathcal{O}_{\mathcal{F}}$ computes F correctly.*

Proof. Fix *any input* $x \in \mathcal{D}$. First, note that by the "preserving functionality" property, the ciphertext $\hat{c}$ computed by the obfuscated program $\mathcal{O}_{\mathcal{F}}$ is statistically close to a uniformly random encryption of m under the output public key $\mathsf{pk}_{F(x)}$.

Secondly, for two uniformly random public/secret key pairs $(\mathsf{pk}_0, \mathsf{sk}_0)$ and $(\mathsf{pk}_1, \mathsf{sk}_1)$, and for any message $m \in \mathcal{M}$ and any $x \in \mathcal{D}$,

$$\Pr[\mathsf{Dec}(\mathsf{sk}_0, \mathsf{Enc}(\mathsf{pk}_1, x, m)) = m] \le 1/|\mathcal{M}| + \mathtt{neg}(\lambda)$$

where the probability is over the coins of the encryption algorithm. In other words, trying to decrypt an encryption of m under pk_1 using the secret key sk_0 will almost never yield the correct answer. Since we can assume without loss of generality that $|\mathcal{M}|$ has superpolynomial size, it follows that for every input x, $\mathcal{O}_{\mathcal{F}}$ computes F correctly with probability $1 - \mathtt{neg}(\lambda)$. □

Claim 3. *The obfuscator $\mathcal{O}_{\mathcal{F}}$ satisfies the virtual black-box property in Definition 4.*

Proof. For every PPT adversary $\mathcal{A}_{\mathcal{F}}$, we construct a simulator $\mathcal{S}_{\mathcal{F}}$ such that

$$\Pr[\Psi' \leftarrow \mathcal{O}_{\mathcal{F}}(F) : \ \mathcal{A}_{\mathcal{F}}(\Psi') = 1] - \Pr[\mathcal{S}_{\mathcal{F}}^{F}(1^\lambda) = 1] \le \mathtt{neg}(\lambda)$$

Consider the (secure obfuscation) adversary $\mathcal{A}$ that simply outputs its input, and a distinguisher D that runs $\mathcal{A}_{\mathcal{F}}$ on its input. Then,

$$\Pr[\mathsf{D}^{\mathsf{FR}_{F,\mathsf{pk},\mathsf{sk},\mathsf{pk}_0,\mathsf{pk}_1}}(\mathcal{A}(\mathcal{O}(\mathsf{FR}_{F,\mathsf{pk},\mathsf{sk},\mathsf{pk}_0,\mathsf{pk}_1}), \mathsf{sk}_0)) = 1] \quad = \quad \Pr[\mathcal{A}_{\mathcal{F}}(\mathcal{O}(\mathsf{FR}_{F,\mathsf{pk},\mathsf{sk},\mathsf{pk}_0,\mathsf{pk}_1}), \mathsf{sk}_0) = 1] \tag{2}$$

Now, by the definition of average-case secure obfuscation, we are guaranteed a simulator $\mathcal{S}$ such that

$$\Pr[\mathsf{D}^{\mathsf{FR}_{F,\mathsf{pk},\mathsf{sk},\mathsf{pk}_0,\mathsf{pk}_1}}(\mathcal{A}(\mathcal{O}(\mathsf{FR}_{F,\mathsf{pk},\mathsf{sk},\mathsf{pk}_0,\mathsf{pk}_1}), \mathsf{sk}_0)) = 1] \quad \approx \quad \Pr[\mathsf{D}^{\mathsf{FR}_{F,\mathsf{pk},\mathsf{sk},\mathsf{pk}_0,\mathsf{pk}_1}}(\mathcal{S}^{\mathsf{FR}_{F,\mathsf{pk},\mathsf{sk},\mathsf{pk}_0,\mathsf{pk}_1}}(1^\lambda, \mathsf{sk}_0)) = 1] \tag{3}$$

Now, the simulator $\mathcal{S}_{\mathcal{F}}$, on input 1^λ and oracle access to F, works as follows:

1. Choose public/secret key pairs $(\mathsf{pk}, \mathsf{sk})$ and $(\mathsf{pk}_0, \mathsf{sk}_0)$, and run $\mathcal{S}$ on input $(1^\lambda, \mathsf{sk}_0)$.

2. Handle $\mathcal{S}$'s oracle queries to $\mathsf{FR}_{F,\mathsf{pk},\mathsf{sk},\mathsf{pk}_0,\mathsf{pk}_1}$ as follows:

    (a) On query a ciphertext c from $\mathcal{S}$, decrypt c using sk and obtain the value of $x \in \mathcal{D}$ and $m \in \mathcal{M}$.

    (b) Next, query the F-oracle with x and learn $F(x) \in \{0,1\}$.

    (c) Construct the ciphertext $\hat{c} = \mathsf{Enc}(\mathsf{pk}_{F(x)}, m)$ and return it to $\mathcal{S}$ as oracle $\mathsf{FR}_{F,\mathsf{pk},\mathsf{sk},\mathsf{pk}_0,\mathsf{pk}_1}$'s response.

3. Finally, run $\mathcal{A}_{\mathcal{F}}$ on whatever $\mathcal{S}$ outputs, and output the resulting bit.

This is a perfect simulation of the view of $\mathcal{S}$, and thus:

$$\Pr[\mathsf{D}^{\mathsf{FR}_{F,\mathsf{pk},\mathsf{sk},\mathsf{pk}_0,\mathsf{pk}_1}}(\mathcal{S}^{\mathsf{FR}_{F,\mathsf{pk},\mathsf{sk},\mathsf{pk}_0,\mathsf{pk}_1}}(1^\lambda, \mathsf{sk}_0)) = 1] \quad = \quad \Pr[\mathcal{S}_{\mathcal{F}}^{F}(1^\lambda) = 1] \tag{4}$$

Putting together the three equations 2, 3 and 4, we get:

$$\Pr[\mathcal{A}_{\mathcal{F}}(\Psi') = 1] \approx \Pr[\mathcal{S}_{\mathcal{F}}^{F}(1^\lambda) = 1]$$

showing that Ψ' is a predicate obfuscation of F. □

Claims 2 and 3, together complete the proof of Theorem 6. □

Remarks

- The proof can be generalized to functions over a larger, yet polynomial-size, range. Recall that the notion of functional re-encryption as defined makes sense only for functions with polynomial size range, and thus, this is the best we can hope for.

- Barak et al. [BGI$^+$01] show impossibility of predicate obfuscation for a family of functions with large, superpolynomial, range. Thus, their result cannot be used directly to rule out the possibility of functional re-encryption for all (polynomial-time computable) function families. However, the result of [BGI$^+$01] can easily be extended to show the impossibility of predicate obfuscation for a family of functions with polynomial range. This result does not contradict our result, as their result only rules out a general obfuscator where the obfuscator runs in time $\mathcal{O}(\text{poly}(|F|))$ for any function F. Note that this is not the case for our positive results, as we restrict the function F to have polynomial sized domain.