Functional Re-encryption and Collusion-Resistant Obfuscation

Nishanth Chandran[1⋆], Melissa Chase[1], and Vinod Vaikuntanathan[2⋆⋆]

[1] Microsoft Research
[2] University of Toronto

Abstract. We introduce a natural cryptographic functionality called *functional re-encryption*. Informally, this functionality, for a public-key encryption scheme and a function F with n possible outputs, transforms ("re-encrypts") an encryption of a message m under an "input public key" pk into an encryption of the same message m under one of the n "output public keys", namely the public key indexed by $F(m)$.

In many settings, one might require that the program implementing the functional re-encryption functionality should reveal nothing about both the input secret key sk as well as the function F. As an example, consider a user Alice who wants her email server to share her incoming mail with one of a set of n recipients according to an access policy specified by her function F, but who wants to keep this access policy private from the server. Furthermore, in this setting, we would ideally obtain an even stronger guarantee: that this information remains hidden even when some of the n recipients may be corrupted.

To formalize these issues, we introduce the notion of *collusion-resistant obfuscation* and define this notion with respect to average-case secure obfuscation (Hohenberger *et al.* - TCC 2007). We then provide a construction of a functional re-encryption scheme for any function F with a polynomial-size domain and show that it satisfies this notion of collusion-resistant obfuscation. We note that collusion-resistant security can be viewed as a special case of dependent auxiliary input security (a setting where virtually no positive results are known), and this notion may be of independent interest.

Finally, we show that collusion-resistant obfuscation of functional re-encryption for a function F gives a way to obfuscate F in the sense of Barak *et al.* (CRYPTO 2001), indicating that this task is impossible for arbitrary (polynomial-time computable) functions F.

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

⋆ part of this work was done while this author was at UCLA.
⋆⋆ part of this work was done while this author was at Microsoft Research.

work of Barak, Goldreich, Impagliazzo, Rudich, Sahai, Vadhan and Yang [3], 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.* [3] 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 were shown in [14, 27, 18, 4]. Positive results for obfuscation, on the other hand, have been largely limited to relatively simple classes of functions such as point functions [7, 9, 21, 27, 14, 8], proximity testing [12], encrypted permutations [1] and more recently, testing hyperplane membership [10].

In one of the few exceptions to this trend, Hohenberger *et al.* [19] showed how to obfuscate a complex cryptographic functionality called re-encryption [5, 2]. 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, [18]) also introduce a strong definition of (average-case) secure obfuscation which we will use and build on in this work. Following [19], Hada [17] showed how to securely obfuscate an encrypted signature functionality.

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

- The key point that enables obfuscation in both [19] and [17] is that they obfuscate functionalities that compute a function "inside a ciphertext". For example, in [19], this is the decryption function and in [17], it is the signature function. Not surprisingly, it has been noted that given a fully homomorphic encryption scheme [22, 13], the functionalities of [19, 17] 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 [19, 17], 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 [14] (for some limited positive results against auxiliary inputs, see [4]). *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 relax (somewhat) 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 [19, 18]) 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 $\{\mathcal{C}_\lambda\}$ that has the following form. Any $C_\mathcal{K} \in \mathcal{C}_\lambda$ 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 $\mathcal{T} \subseteq [\ell]$. ($[\ell]$ 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 [19]) 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 [5, 2]. 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, and n output public keys. The functionality receives as input a ciphertext of message m with "identity" id under the input public key pk. [3] It decrypts the ciphertext using the secret key sk to get m and id, and then re-encrypts m under the "appropriate" output public

[3] 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.

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 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 and Alice's access policy – it should merely provide a means for the server to do the appropriate routing. [4]

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; this lets the server decrypt all incoming messages and determine where to route them. Unfortunately, this "solution" completely fails this minimum requirement. 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, could also be carried out given only 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 or about 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. If we consider the auxiliary input to be the secret keys of the colluding recipients, then our strong notion of collusion-resistant secure obfuscation guarantees that this is the only information that the server could possibly learn by colluding.

Selectively delegating access is also the central theme of a recently introduced notion of predicate encryption [20, 25] (which can be viewed as attribute based encryption in which ciphertexts hide their attributes). In fact, (predicate-hiding, public key) predicate encryption schemes can potentially 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 invisible to the recipients – they

[4] 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.

continue using their already registered public keys, and they do not even have to know of 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 which 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. We show, informally:

Theorem 1 (Informal). *Under the Symmetric External Diffie-Hellman assumption there exists an encryption scheme such that for any function $F : \mathcal{D} \to \mathcal{R}$ with polynomial-sized domain $\mathcal{D}$, 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 is $O(|\mathcal{D}| \cdot \mathsf{poly}(\lambda))$, 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 [19] allows us to achieve both goals simultaneously. 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 $\boldsymbol{a}_1, \cdots, \boldsymbol{a}_d \in \mathbb{Z}_q^d$ 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 $h \in \mathbb{H}$. On input $g^{\boldsymbol{a}_i}$, OF outputs $\mathsf{e}(g,h)^{\hat{a}_{F(i)}}$. We shall now informally sketch 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 $\langle \boldsymbol{a}_i, \boldsymbol{\alpha} \rangle = \hat{a}_{F(i)}$ 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 $h^{\boldsymbol{\alpha}}$. (This can be computed given only $h^{\hat{a}_{F(i)}}$ and $\boldsymbol{a}_i$ for all i, so we do not actually need the recipient secret keys $\hat{a}_{F(i)}$.) On input $g^{\boldsymbol{a}_i}$, the program computes and outputs $\prod_{j=1}^{d} \mathsf{e}(g^{a_{ij}}, h^{\alpha_j}) = \mathsf{e}(g,h)^{\langle \boldsymbol{a}_i, \boldsymbol{\alpha} \rangle} = \mathsf{e}(g,h)^{\hat{a}_{F(i)}}$, 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 $\langle \boldsymbol{a}_i, \boldsymbol{\alpha} \rangle = w_i \hat{a}_{F(i)}$ and $\langle \boldsymbol{a}_i, \boldsymbol{\beta} \rangle = w_i$, 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 $g^{\boldsymbol{a}_i}$, 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 $\{g^{\boldsymbol{a}_i}\}, 1 \leq i \leq d$ 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 $g^{\boldsymbol{a}_i}$ 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 $g^{\boldsymbol{a}_i}, \cdots, g^{\boldsymbol{a}_d}$, and an input encryption of message m with identity i uses the key $g^{\boldsymbol{a}_i}$. Finally, in order to obtain a secure obfuscation, we apply techniques from [19] to re-randomize the 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 [3]-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 [5], and yet hides all internal details of F except the input-output behavior. Together with the impossibility result of [3] 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 the full version of the paper [11] for a formal statement and proof of 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 informally argue that this may require 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 [20], following [23, 15] (in particular, the predicate-hiding property was defined in the work of Shi, Shen and Waters [25]). Constructions of predicate encryption schemes (even ones that

[5] 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 $F(\mathsf{id}) = i$.

do not achieve predicate-hiding) are known only for simple classes of functions such as inner products [20]. Moreover, in the public-key setting, we do not know how to achieve (any reasonable definition of) *predicate-hiding*, even for simple functions. Since collusion-resistant obfuscation of functional re-encryption seems to have the same flavor in functionality as predicate-hiding public-key predicate encryption, advancements in the class of policy functions that these primitives can handle seem to be correlated.

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 the presence of such auxiliary information. We now recall the notion of average-case secure obfuscation introduced in [19] below.

Definition 1. *An efficient algorithm* $\mathtt{Obf}$ *that takes as input a (probabilistic) circuit* C *from the family* $\{\mathcal{C}_\lambda\}$ *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* λ *and any* $C \in \mathcal{C}_\lambda$*:*

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

  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* $p(\lambda)$ *such that for sufficiently large input lengths* λ*, for any* $C \in \mathcal{C}_\lambda$*, 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: *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* λ*:*

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

  The probability is over the selection of a random circuit C *from* $\mathcal{C}_\lambda$*, and the coins of the distinguisher, the simulator, the oracle, and the obfuscator.* [6]

2.2 Average-case secure obfuscation with collusion

Consider the case where we would like to obfuscate a function family $\{\mathcal{C}_\lambda\}$ that has the following particular form. Any $C_\mathcal{K} \in \mathcal{C}_\lambda$ is parameterized by a set of

[6] This is the definition in [19] but with a dummy adversary. The authors of that paper note that this is equivalent to the definition they give.

"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 [19]) 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* [14]. 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 $C_\mathcal{K} \leftarrow \mathcal{C}_\lambda$. 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* $\{\mathcal{C}_\lambda\}$ *if it satisfies the following properties:*

- *"*Preserving functionality*" and "*Polynomial Slowdown*", as in Definition 1.*
- Average-case Virtual Black-Boxness against Collusion: *There exists an efficient simulator* $\mathcal{S}$*, and a negligible function* $\mathtt{neg}(\lambda)$*, such that for every input length* λ*, every efficient distinguisher* D*, and any subset* $\mathcal{T} \subseteq [\ell]$*:*

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

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

Remarks on the Definition. 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. Furthermore, 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.

2.3 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 function $F : \mathcal{D} \to \mathcal{R}$, input public/secret key pair $(\mathsf{pk}, \mathsf{sk})$, and output public keys $\widehat{\mathsf{pk}}_1, \ldots, \widehat{\mathsf{pk}}_{|\mathcal{R}|}$[7] is a functionality that takes as input a ciphertext $c = \mathtt{I\text{-}Enc}(\mathsf{pk}, \mathsf{id}, m)$ and re-encrypts m under the output public key $\widehat{\mathsf{pk}}_{F(\mathsf{id})}$. More precisely, for a given function $F : \mathcal{D} \to \mathcal{R}$, we are interested in the family of circuits $\mathcal{FR}_{F,\mathcal{D},\mathcal{R}} = \{\mathsf{FR}_{\lambda,F,\mathcal{D},\mathcal{R}}\}_{\lambda>0}$ where each circuit

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

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

$C_{\mathsf{pk},\mathsf{sk},\widehat{\mathsf{pk}}_1,\ldots,\widehat{\mathsf{pk}}_{|\mathcal{R}|}}$, on input c :

Computes $(\mathsf{id},m) \leftarrow \mathtt{I\text{-}Dec}(\mathsf{sk},c)$, and outputs $\widehat{c} \leftarrow \mathtt{O\text{-}Enc}(\widehat{\mathsf{pk}}_{F(\mathsf{id})},m)$.

If $\mathtt{I\text{-}Dec}(\mathsf{sk},c)$ returns $\perp$ then outputs random elements in 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 $\mathsf{pk},\widehat{\mathsf{pk}}_1,\ldots,\widehat{\mathsf{pk}}_{|\mathcal{R}|}$.

Now, for a class of functions, $\mathcal{F}$, we say that a re-encryption program *securely obfuscates re-encryption for* $\mathcal{F}$, if there exists a simulator $\mathcal{S}$ that satisfies the collusion resistant obfuscation property w.r.t. $\mathcal{FR}_{F,\mathcal{D},\mathcal{R}}$ for all $F : \mathcal{D} \to \mathcal{R} \in \mathcal{F}$.

In other words, all public keys in the circuit $C_{\mathsf{pk},\mathsf{sk},\widehat{\mathsf{pk}}_1,\ldots,\widehat{\mathsf{pk}}_{|\mathcal{R}|}}$ are considered public knowledge; 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, 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 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 $c = \mathtt{I\text{-}Enc}(\mathsf{pk},\mathsf{id},m)$, 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 $i \in \mathcal{T}$.
- Finally, we remark that the definition of obfuscation for functional re-encryption by itself does not guarantee the semantic security of the input and output encryption schemes. We define these separately and prove the security of the encryption schemes (even in the presence of the re-encryption program). In more detail, we will require the semantic security of the input encryption scheme, on messages encrypted with an identity id^*, whenever $F(\mathsf{id}^*) \notin \mathcal{T}$, even when the adversary is given access to a re-encryption oracle. We will similarly require that the input ciphertext hides the identity id^*, under which the message is encrypted. The security of the output encryption scheme will be that of standard semantic security. Since we wish to hide everything about the function F, we will also require the output encryption scheme to be key private; i.e., an encryption under public key $\widehat{\mathsf{pk}}_i$ will be indistinguishable from an encryption under public key $\widehat{\mathsf{pk}}_j$. For formal definitions and proofs of these properties, see the full version [11].

3 Preliminaries

We let λ be the security parameter throughout this paper. By $\mathtt{neg}(\lambda)$ 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 $\mathcal{D}_2$, $\mathcal{D}_1 \stackrel{c}{\approx} \mathcal{D}_2$ 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., $\boldsymbol{a}$. Let $\mathbb{G}$ be a group of prime order q. For a vector $\boldsymbol{a} = (a_1, a_2, \cdots, a_\ell) \in \mathbb{Z}_q^\ell$ 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, $x\boldsymbol{a} = \boldsymbol{a}\boldsymbol{b}, \boldsymbol{a}/x = \boldsymbol{a}/\boldsymbol{b}$, 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 $\mathbb{G}$ and $\mathbb{H}$. That is, when $(q, \mathbb{G}, \mathbb{H}, \mathbb{G}_T, \mathsf{e}) \leftarrow \mathsf{BilinSetup}(1^\lambda); g \leftarrow \mathbb{G}; a, b, c \leftarrow \mathbb{Z}_q$, the following two ensembles are indistinguishable:

$$\{(q, \mathbb{G}, \mathbb{H}, \mathbb{G}_T, \mathsf{e}, g, g^a, g^b, g^{ab})\} \stackrel{c}{\approx} \{(q, \mathbb{G}, \mathbb{H}, \mathbb{G}_T, \mathsf{e}, g, g^a, g^b, g^c)\}$$

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 [26, 6, 24, 16]. In this work, we use the SXDH assumption.

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 describe 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 construction of our output encryption scheme requires this to be the case; however, one can encrypt longer messages by breaking the message into smaller blocks and encrypting the blocks separately.

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 [16] provides an efficient scheme for the type of statements we use, which is perfectly sound and computationally zero-knowledge based on SXDH. We remark that, while the semantic security of the input encryption scheme does not require this NIZK proof, the obfuscation guarantee provided by our construction relies on it; if, for example the adversary were to provide an invalid ciphertext as input to the re-encryption program (e.g. by combining 2 valid ciphertexts with different i's), the program might output some group elements that are distinguishable from random to an adversary that possesses some of the recipient secret keys.

The input encryption scheme is as follows:

1. $\texttt{I-Gen}(1^\lambda, 1^d)$: Pick random vectors $\boldsymbol{a}_1, \cdots, \boldsymbol{a}_d$ from $\mathbb{Z}_q^d$ that are linearly independent. We also generate crs, a common reference string (abbreviated CRS) for the NIZK proof system. Output $\mathsf{pk} = (\mathsf{crs}, g, g^{\boldsymbol{a}_1}, \cdots, g^{\boldsymbol{a}_d})$, and $\mathsf{sk} = (\boldsymbol{a}_1, \cdots, \boldsymbol{a}_d)$. We remark that the public key pk can be viewed as being made up of d public keys $\mathsf{pk}_i = (g, g^{\boldsymbol{a}_i})$ 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 $\mathbb{Z}_q$, and compute:
   (a) $\mathbf{C} = g^{r\boldsymbol{a}_i}$; $D = g^r m$, and
   (b) $\mathbf{C}' = g^{r'\boldsymbol{a}_i}$; $D' = g^{r'}$
   (c) π, a proof that these values are correctly formed, i.e. that they correspond to one of the vectors $g^{\boldsymbol{a}_i}$ 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 $1_\mathbb{G}$ under pk_i. $\mathbf{E}'$ is primarily used by the re-encryption program for input re-randomization, and is not

required if the encryption scheme is used stand-alone without the functional re-encryption program.)

3. $\texttt{I-Dec}(\mathsf{sk}, (\mathbf{E}, \mathbf{E}'))$: If any of the components of the ciphertext $\mathbf{E}'$ is $1_{\mathbb{G}}$ or if the proof π does not verify, output $\perp$.[8] Ignore $\mathbf{E}', \pi$ subsequently, and parse $\mathbf{E}$ as $(\mathbf{C}, D)$. Check that for some $i \in [d]$ and $m \in \mathcal{M}$, $D \cdot (\mathbf{C}^{1/\boldsymbol{a}_i})^{-1} = (m, \cdots, m)$. If yes, output (i, m). Otherwise output $\perp$.

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

1. $\texttt{O-Gen}(1^\lambda)$: Pick $\hat{a} \leftarrow \mathbb{Z}_q$. Let $\widehat{\mathsf{pk}} = h^{\hat{a}}$ and $\widehat{\mathsf{sk}} = \hat{a}$.
2. $\texttt{O-Enc}(\widehat{\mathsf{pk}}, m)$: To encrypt a message $m \in \mathcal{M} \subset \mathbb{G}$,
   - Choose random number $r \leftarrow \mathbb{Z}_q$.
   - Compute $\widehat{Y} = (h^{\hat{a}})^r$ and $\widehat{W} = h^r$.
   - Output the ciphertext as $\left[\widehat{F}, \widehat{G}\right] := \left[\mathsf{e}(g, \widehat{Y}),\ \mathsf{e}(g, \widehat{W}) \cdot \mathsf{e}(m, h)\right]$.
3. $\texttt{O-Dec}(\widehat{\mathsf{sk}} = \hat{a}, (\widehat{F}, \widehat{G}))$: 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 $\mathsf{e}(m, h) = \widehat{Q}$. If so, output m and halt. (Note that if $\mathsf{e}(m, h)$ are precomputed for all $m \in \mathcal{M}$, then this step can be implemented with a table lookup.)

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 $w_i \leftarrow \mathbb{Z}_q$ 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 $i \in [d]$:

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

The re-encryption key consists of the tuple $(\mathbf{A}, \mathbf{B})$ where $\mathbf{A} = h^{\boldsymbol{\alpha}}$ and $\mathbf{B} = h^{\boldsymbol{\beta}}$. We remark that computing the re-encryption key does not require knowledge of the output secret keys. To compute $h^{\boldsymbol{\alpha}}$, one can take the output public keys $h^{\hat{a}_1}, \cdots, h^{\hat{a}_d}$, and with the knowledge of the input secret keys $\{\boldsymbol{a}_1, \cdots, \boldsymbol{a}_d\}$ and random values $w_1, \ldots, w_d$, one can solve the set of equations $h^{\langle \boldsymbol{a}_i, \boldsymbol{\alpha} \rangle} = h^{w_i \cdot \hat{a}_{F(i)}}$, for all $i \in [d]$, to obtain $h^{\boldsymbol{\alpha}}$. $h^{\boldsymbol{\beta}}$ can be computed in a similar manner.

[8] This "sanity check" is to ensure the correctness 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 $(\mathbf{A}, \mathbf{B})$ 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 $(\widehat{F}, \widehat{G})$ for random $\widehat{F}, \widehat{G} \in \mathbb{G}_T$. 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 $1_{\mathbb{G}}$, and this re-randomized encryption of $1_{\mathbb{G}}$ 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

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

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

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 $\mathbf{C} = g^{r\boldsymbol{a}_i}, D = g^r m$ and $\mathbf{C}' = g^{r'\boldsymbol{a}_i}, D = g^{r'}$. (If π does not verify, then both the functionality and the above program will output random group elements.) Let the re-encryption key be $(\mathbf{A}, \mathbf{B})$ where $\mathbf{A} = h^{\boldsymbol{\alpha}}$ and $\mathbf{B} = h^{\boldsymbol{\beta}}$.

- First, the input re-randomization step computes $\widehat{\mathbf{C}} = \mathbf{C}(\mathbf{C}')^t = g^{(r+tr')\boldsymbol{a}_i} = g^{\hat{r}\boldsymbol{a}_i}$ and $\widehat{D} = D(D')^t = g^{r+tr'} m = g^{\hat{r}} m$, where we defined $\hat{r} \triangleq r + tr'$.
- Second, the main re-encryption step computes $\widehat{F} = \prod_{j=1}^{d} \mathsf{e}(\widehat{C}_j, A_j) = \mathsf{e}(g,h)^{\hat{r}\langle \boldsymbol{a}_i, \boldsymbol{\alpha}\rangle} = \mathsf{e}(g,h)^{\hat{r} w_i \hat{a}_{F(i)}}$ and

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

- Now the ciphertext looks like $\widehat{F} = \mathsf{e}(g, h^{\hat{a}_{F(i)}\rho})$, $\widehat{G} = \mathsf{e}(g, h^{\rho}) \cdot \mathsf{e}(m, h)$, where $\rho = \hat{r} w_i$ is 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 (in particular, since $r' \neq 0$).

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

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", and the output scheme is also key-private) under the DDH assumption over different groups, even given the re-encryption program. We present a detailed proof in the full version [11].

Remark. Note that if $d = n = 1$, then our construction (with the removal of certain now unnecessary parts, such as the NIZK proof) reduces to something very similar to that of [19]. Also, note that if the function F were to have larger (super-polynomial) domain, then our solution would satisfy the property of polynomial slowdown only if F were represented as a truth table. If F has large domain but a concise representation, then this property no longer holds.

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 and must yet produce a "fake" re-encryption program, that on input a ciphertext of message m with identity id, outputs a correct ciphertext of m under $\widehat{\mathsf{pk}}_j$. Hence, in order to put the simulator on an equal footing with the adversary we need to give the simulator the power to produce an explicit program which can selectively decrypt input ciphertexts. One way to do this is to cheat and give the simulator the vector $\boldsymbol{a}_i$, for all i such that $F(i) = j$, in our construction, which we will refer to as sk_i. (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 the full version [11] how this can be avoided. In other words, we first show the security of our scheme in the modified model where the simulator obtains the sk_i values for all i such that $F(i) \in \mathcal{T}$. Next (in the full version [11]), we show that if the scheme is secure in this model, then it can be easily transformed into a scheme that is secure in the standard model where the simulator, like the real world adversary, only gets $\widehat{\mathsf{sk}}_j$ values for $j \in \mathcal{T}$. We will now focus on proving the former statement. Towards showing that our obfuscation satisfies the collusion-resistant secure obfuscation definition in the model where the simulator obtains the sk_i values for all i such that $F(i) \in \mathcal{T}$, 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 $\widehat{\mathsf{sk}}_j$ of all the corrupted receivers (where $j \in \mathcal{T}$) and the secret keys sk_i such that $F(i) \in \mathcal{T}$, 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 $(h, h^{\boldsymbol{\alpha}}, h^{\boldsymbol{\beta}})$ where 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.

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 $\mathsf{pk} = (g, g^{\boldsymbol{a}_1}, \cdots, g^{\boldsymbol{a}_d})$; 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 $w_1, \ldots, w_d$ from $\mathbb{Z}_q$. Sample random $\boldsymbol{\alpha}, \boldsymbol{\beta}$ from $\mathbb{Z}_q^d$ such that
$$\forall i \text{ s.t. } F(i) \in \mathcal{T}: \qquad \langle \boldsymbol{a}_i, \boldsymbol{\alpha} \rangle = w_i \hat{a}_{F(i)} \qquad \text{and} \qquad \langle \boldsymbol{a}_i, \boldsymbol{\beta} \rangle = w_i - 1$$
Note that this can be done efficiently using the knowledge of the vectors $\boldsymbol{a}_i$ that we obtained in $\{\mathsf{sk}_j\}_{j\in F^{-1}(\mathcal{T})}$, as well as the $\hat{a}_{F(i)}$ values which are part of the corrupted secret keys. Compute $\mathbf{A} = h^{\boldsymbol{\alpha}}$, and $\mathbf{B} = h^{\boldsymbol{\beta}}$. Output the tuple $(\mathbf{A}, \mathbf{B})$ 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. *Under SXDH, for any ppt distinguisher* D *and corrupt set* $\mathcal{T} \subseteq [n]$,
$$\mathsf{D}^{\mathcal{C}}\left[\mathcal{O}\mathfrak{bf}(\mathcal{C}), \mathcal{T}, \{\widehat{\mathsf{sk}}_j\}_{j\in\mathcal{T}}, \{\mathsf{sk}_j\}_{j\in F^{-1}(\mathcal{T})}\right] \overset{c}{\approx}$$
$$\mathsf{D}^{\mathcal{C}}\left[\mathcal{S}^{\mathcal{C}}(1^\lambda, \mathcal{T}, \{\widehat{\mathsf{sk}}_j\}_{j\in\mathcal{T}}, \{\mathsf{sk}_j\}_{j\in F^{-1}(\mathcal{T})}), \{\widehat{\mathsf{sk}}_j\}_{j\in\mathcal{T}}, \{\mathsf{sk}_j\}_{j\in F^{-1}(\mathcal{T})}\right]$$

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

From the above theorem, our main theorem (which we stated informally as Theorem 1) follows after making the necessary modifications to the construction outlined earlier. We now describe a sketch of the proof of Theorem 2. For the formal proof, see the full version [11].

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 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 $(\mathbf{A}, \mathbf{B})$ where $\mathbf{A} = h^{\boldsymbol{\alpha}}$ and $\mathbf{B} = h^{\boldsymbol{\beta}}$ with $\boldsymbol{\alpha}, \boldsymbol{\beta}$ being solutions to the equations
  $$\langle \boldsymbol{a}_i, \boldsymbol{\alpha} \rangle = w_i \hat{a}_{F(i)} \qquad \text{and} \qquad \langle \boldsymbol{a}_i, \boldsymbol{\beta} \rangle = w_i - 1 \quad \text{for all } i \in [d] \tag{1}$$
  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 $\mathbb{Z}_q^*$. [9] 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 the full version [11] 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 then show 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:** Next, 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.

[9] 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.

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 $\left[\mathsf{e}(g, h^w),\ \mathsf{e}(g, h^r) \cdot \mathsf{e}(m, h)\right]$ for random $\hat{a}, r \in \mathbb{Z}_q$. 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 $\mathbb{G}_T \times \mathbb{G}_T$.

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 $i \in \mathcal{T}$).

Now, 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 $i \in \mathcal{T}$).

– **Step 4:** Finally, 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.

Acknowledgements We wish to thank Markulf Kohlweiss for suggesting the use of the SXDH assumption which simplified our construction.

References

1. B. Adida and D. Wikström. How to shuffle in public. In *TCC '07*, pp 555–574.
2. G. Ateniese, K. Fu, M. Green, and S. Hohenberger. Improved proxy re-encryption schemes with applications to secure distributed storage. In *NDSS '05*.
3. B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. Vadhan, and K. Yang. On the (im)possibility of obfuscating programs. In *Crypto '01*, pp 1–18.
4. N. Bitansky and R. Canetti. On strong simulation and composable point obfuscation. In *Crypto '10*, pp 520–537.

5. M. Blaze, G. Bleumer, and M. Strauss. Divertible protocols and atomic proxy Cryptography. In *Eurocrypt '98*, pp 127–144.
6. D. Boneh, X. Boyen, and H. Shacham. Short group signatures. In *Crypto '04*, pp 41–55.
7. R. Canetti. Towards realizing random oracles: Hash functions that hide all partial information. In *Crypto '97*, pp 455–469.
8. R. Canetti and R. Dakdouk. Obfuscating point functions with multibit output. In *Eurocrypt '08*, pp 489–508.
9. R. Canetti, D. Micciancio, and O. Reingold. Perfectly one-way probabilistic hash functions. In *STOC '98*, pp 131–140.
10. R. Canetti, G. Rothblum, and M. Varia. Obfuscation of hyperplane membership. In *TCC '10*, pp 72–89.
11. N. Chandran, M. Chase, and V. Vaikuntanathan. Collusion Resistant Obfuscation and Functional Re-encryption. IACR Eprint Archive, http://eprint.iacr.org/2011/337.
12. Y. Dodis and A. Smith. Correcting errors without leaking partial information. In *STOC '05*, pp 654–663.
13. C. Gentry. Fully homomorphic encryption using ideal lattices. In *STOC '09*, pp 169–178.
14. S. Goldwasser and Y. Kalai. On the impossibility of obfuscation with auxiliary input. In *FOCS '05*, pp 553–562.
15. V. Goyal, O. Pandey, A. Sahai, and B. Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *CCS '06*, pp 89–98.
16. J. Groth and A. Sahai. Efficient non-interactive proof systems for bilinear groups. In *Eurocrypt '08*, pp 415–432.
17. S. Hada. Secure obfuscation for encrypted signatures. In *Eurocrypt '10*, pp 92–112.
18. D. Hofheinz, J. Malone-Lee, and M. Stam. Obfuscation for Cryptographic purposes. In *TCC '07*, pp 214–232.
19. S. Hohenberger, G. Rothblum, A. Shelat, and V. Vaikuntanathan. Securely obfuscating re-encryption. In *TCC '07*, pp 233–252.
20. J. Katz, A. Sahai, and B. Waters. Predicate encryption supporting disjunctions, polynomial equations, and inner products. In *Eurocrypt '08*, pp 146–162.
21. B. Lynn, M. Prabhakaran, and A. Sahai. Positive results and techniques for obfuscation. In *Eurocrypt '04*, pp 20–39.
22. R. Rivest, L. Adleman, and M. Dertouzos. On data banks and privacy homomorphisms. In *Foundations of Secure Computation*, pp 169–177. Academic Press, 1978.
23. A. Sahai and B. Waters. Fuzzy identity-based encryption. In *Eurocrypt '05*, pp 457–473.
24. M. Scott. Authenticated ID-based key exchange and remote log-in with insecure token and PIN number. http://eprint.iacr.org/2002/164, 2002.
25. E. Shen, E. Shi, and B. Waters. Predicate privacy in encryption systems. In *TCC '09*, pp 457–473.
26. E. Verheul. Evidence that xtr is more secure than supersingular elliptic curve Cryptosystems. *J. Cryptology*, 17(4):277–296, 2004.
27. H. Wee. On obfuscating point functions. In *STOC 2005*, pp 523–532.