Re-encryption, functional re-encryption, and multi-hop re-encryption: A framework for achieving obfuscation-based security and instantiations from lattices

Nishanth Chandran[1], Melissa Chase[1], Feng-Hao Liu[2⋆], Ryo Nishimaki[3], and Keita Xagawa[3]

[1] Microsoft Research {nichandr,melissac}@microsoft.com
[2] University of Maryland fenghao@cs.umd.edu
[3] NTT Secure Platform Laboratories {nishimaki.ryo,xagawa.keita}@lab.ntt.co.jp

Abstract. In this work we define multiple relaxations to the definition of correctness in secure obfuscation. While still remaining meaningful, these relaxations provide ways to obfuscate many primitives in a more direct and efficient way. In particular, we first show how to construct a secure obfuscator for the re-encryption primitive from the Decisional Learning with Errors (DLWE) assumption, without going through fully homomorphic encryption. This can be viewed as a meaningful way to trade correctness for efficiency.
Next, we show how our tools can be used to construct secure obfuscators for the functional re-encryption and multi-hop unidirectional re-encryption primitives. In the former case, we improve upon the efficiency of the only previously known construction that satisfies the stronger notion of collusion-resistant obfuscation (due to Chandran *et al.* - TCC 2012) and obtain a construction with input ciphertexts of *constant* length. In the latter case, we provide the first known obfuscation-based definition and construction; additionally, our scheme is the first scheme where the size of the ciphertexts does not grow with every hop.

1 Introduction

Program obfuscation. Informally, an obfuscator [6] is an algorithm that converts a program into another program that has the same behavior but is "completely unintelligible", in that it reveals no information besides what can be learned from observing the input/output behavior. Most previous works have focused on impossibility results or constructions for extremely simple programs. In one of the first works on obfuscating more complex cryptographic functionalities, Hohenberger *et al.* [25] showed how to securely obfuscate the re-encryption functionality. The re-encryption functionality (introduced by [7] and more formally defined in [5]) is parameterized by two public keys for an encryption scheme. It takes as input a ciphertext of message m under the first public key,

⋆ This work was done while the author was an intern at Microsoft Research.

and outputs a ciphertext of the same message m under the second public key. Re-encryption has many applications, ranging from secure distributed file servers, to outsourced filtering of encrypted spam, to the iTunes DRM system.

Why secure obfuscation for re-encryption? The use of obfuscation-based definitions for re-encryption is particularly appealing for many reasons. First, secure obfuscation results in a definition of security for re-encryption that is much stronger than several previous definitions. It simultaneously captures many game-based properties defined in earlier formalizations of re-encryption and guarantees that the proxy cannot learn *anything* beyond what is revealed by the input-output behavior of the re-encryption functionality (which it must inherently learn). Second, note that if we have a protocol that is secure when making use of an "ideal" re-encryption functionality, then the security of the system will be preserved when the untrusted proxy is given a program that is a secure obfuscation of the same functionality. Finally, the secure obfuscation definition for re-encryption is clean and easy to use, which is particularly relevant for a primitive such as re-encryption for which multiple variants of security definitions have been studied. Additionally, it also makes it easier to define security for the more complex functionalities that we consider, such as multi-hop and functional re-encryption. In light of these advantages, and given the widespread applications of proxy re-encryption, obtaining efficient constructions that satisfy the definition of secure obfuscation is very important from both a theoretical and a practical perspective.

Why re-encryption? Beyond the direct applications mentioned above, studying re-encryption may help advance the more general study of obfuscation. One of the few areas in obfuscation which has seen positive results is the case where the output of the program is encrypted [22, 13]. Since re-encryption is one of the simplest such functionalities, it makes a good starting place for further study.

Constructing re-encryption schemes. Hohenberger *et al.* [25] (and also independently Hofheinz *et al.* [24]) introduced the notion of *average-case secure obfuscation*, which has been the standard definition of obfuscation in these works; it captures the idea that the obfuscated program reveals nothing to an adversary when the associated encryption key is chosen at random and unknown to the adversary. The work of Hohenberger *et al.* [25] showed how to securely obfuscate the re-encryption functionality under this definition assuming a bilinear pairing. In the interest of basing primitives on a variety of assumptions, it is natural to ask: *can we construct a secure obfuscator for the re-encryption functionality based on other types of assumptions?* In addition, their scheme has the limitation that the input and output encryption schemes are different, in other words, the program takes as input ciphertexts under one encryption scheme and outputs ciphertexts under not just a different key but a different scheme. While this may

be alright in certain scenarios, many applications (e.g. multi-hop re-encryption) require input and output schemes to have the same structure to allow for cascading, i.e. taking a re-encrypted ciphertext and re-encrypting it again.

As noted in [15] the re-encryption functionality can be securely realized given any fully homomorphic encryption (FHE) scheme [28, 15]; the re-encryption key is simply $K_{\mathsf{pk}\to\widehat{\mathsf{pk}}} = \mathsf{Enc}_{\widehat{\mathsf{pk}}}(\mathsf{sk})$ and the re-encryption program, on input $c = \mathsf{Enc}_{\mathsf{pk}}(m)$, computes $\mathsf{Enc}_{\widehat{\mathsf{pk}}}(c)$ and then $\mathsf{Eval}_{\mathsf{evk}}(f, c, K_{\mathsf{pk}\to\widehat{\mathsf{pk}}})$, where f is the decryption circuit, to obtain $\mathsf{Enc}_{\widehat{\mathsf{pk}}}(m)$. (This can be generalized to achieve essentially any functionality with encrypted output.) We know constructions of FHE based on a variety of lattice-based assumptions [15, 16, 11, 10, 17, 9, 8], so this might give lattice-based constructions for re-encryption.

There are however two issues with this approach: First, FHE is a very strong primitive, and despite significant progress, it is still very expensive; ideally constructing a simple functionality like re-encryption should not require such heavyweight tools. More importantly, by the definition of correctness of program obfuscation, a secure obfuscator for the re-encryption functionality must output ciphertexts that have a distribution that is statistically close to the distribution output by the ideal re-encryption circuit *for all inputs*. In particular, this statistical closeness must hold even for invalid ciphertexts. The only way we know to achieve such a distribution is through bootstrapping [15], which is the most computationally expensive part of the FHE constructions (and not included the more efficient somewhat homomorphic encryption (SHE) schemes).

Challenges in lattice based constructions. Thus one might ask, *what about simpler lattice-based constructions?* More concretely, can we achieve an obfuscation-based notion of re-encryption without bootstrapping? Under previous obfuscation definitions, this seems very challenging, and, interestingly, the challenge arises not from the security requirements (VBB obfuscation), but from the correctness property (referred to as *preserving functionality*). Intuitively, the issue is as follows: a well-formed ciphertext is formed by encoding the message and then adding a small amount of random noise; this is what would be produced by the unobfuscated program, and an obfuscation which preserves functionality would have to produce ciphertexts that are similarly distributed. This means that no matter what ciphertext the adversary chooses as input (even an invalid ciphertext formed by adding a lot of noise), the obfuscated program must either recognize that the ciphertext is invalid, or output ciphertexts with small, independently generated noise. The only way we know to do this is to use bootstrapping, which essentially runs the decryption algorithm under a layer of encryption and thus can detect poorly formed ciphertexts or remove the noise from the input ciphertext and produce an output ciphertext with fresh small noise. How-

ever, as mentioned above, bootstrapping is very expensive, thus we would like to consider meaningful notions that can be achieved with simpler techniques.

Our contributions. Definitionally, our first contribution is to examine different weaker notions of correctness. We propose two new definitions, which are relaxations of the standard notion of preserving functionality. We then evaluate the implications of these definitions, focusing for concreteness on re-encryption primitives. Next, we consider how to construct schemes satisfying these weaker definitions. We define two tools, which we call blurring and key-switching, essentially formalizing a number of techniques that were used in various FHE constructions. While these techniques are not new, we provide general definitions, independent of any specific instantiation, thus allowing them to be used abstractly as tools in general constructions. Finally, we consider two additional re-encryption primitives, functional re-encryption and multi-hop re-encryption, and use our new tools and definitions to solve several previously open problems.

1.1 Our results and techniques

Relaxing correctness in secure obfuscation. We define two relaxations of correctness for the definition of secure obfuscation that allow more efficient constructions of re-encryption (and other) schemes. The first relaxation, informally, guarantees only that the output distribution of the obfuscated program and the ideal functionality are statistically close on so called “well-formed" inputs (i.e. a subset of all the inputs to the functionality). The security property (average case VBB) is still the standard notion (and is guaranteed on all inputs); such a relaxation of the correctness can be viewed as a form of “correctness in the semi-honest setting", in that correctness is guaranteed whenever the adversary selects inputs to the obfuscated program honestly. The next relaxation guarantees that the output of the obfuscator on well-formed inputs is correct with respect to some algorithm. (For example, they might both decrypt to the same value in case of a decryption algorithm.) Finally, we consider a correctness guarantee that says that the output distribution of the obfuscated program is computationally indistinguishable from that of the ideal functionality. (We might for example consider an obfuscator which satisfies this computational correctness over all inputs, and additionally satisfies one of the above notions on the set of well formed inputs.) We view these three relaxations to correctness of the secure obfuscation definition as important contributions of this work and believe they maybe applicable to other functionalties beyond re-encryption. Finally, we emphasize that these are relaxations only to the *correctness* of the scheme. We still maintain the guarantee that the obfuscated program reveals no more than what can be computed given black box access to the functionality.

Abstractions for two lattice-based techniques. Our next contribution is to abstract out two mechanisms that we need for re-encryption from the previous works of [15, 10], and implement these mechanisms with several instantiations. In particular, we provide abstractions for (1) key-switching and (2) blurring. These two mechanisms are designed to be used together: the key-switching mechanism is used to transform a ciphertext $\mathsf{Enc}_{\mathsf{pk}}(m)$ into another ciphertext $\mathsf{Enc}_{\widehat{\mathsf{pk}}}(m)$. However the output distribution of this mechanism might be different from a fresh ciphertext of message m under public key $\widehat{\mathsf{pk}}$; the blurring mechanism is used to smooth out this difference. We define two variants: a strong blurring and a weak blurring mechanism. At a high level, using strong blurring helps us achieve the first relaxation of correctness; weak blurring enables us to achieve the second and third relaxations.

We then proceed to show how to implement the key switching mechanism as well as the strong and weak blurring mechanisms using: a) Regev's encryption scheme [27], and b) the dual Regev encryption scheme [19].

We remark here that while the notions of key switching and blurring are not new, we provide a formal definition of the properties that we require from these two notions. To the best of our knowledge, this is the first such definition of these notions and we hope it will help these techniques to find other applications.

Contribution to lattice-based schemes and secure obfuscation. The problems we encounter in satisfying the obfuscation-based definitions of security seem to be fundamental to most lattice-based schemes; we hope that our relaxations will also help lead to lattice-based obfuscations for other functionalities.

1.2 Applications of our results

We apply our tools to construct schemes for re-encryption and two useful variants: functional re-encryption, and multi-hop unidirectional re-encryption.

Re-encryption. We first show that using any fully homomorphic encryption scheme with a strong blurring mechanism, one can obtain a secure obfuscation of a re-encryption scheme that satisfies the standard definition of correctness (i.e., the output is statistically close to the ideal functionality on *all* inputs).

Given that FHE is overkill, we provide direct, more efficient, constructions based on the Decisional Learning with Errors (DLWE) assumption [27] via the realizations of key switching and blurring mentioned above. With strong blurring, the correctness of this scheme is guaranteed on all well-formed inputs (i.e., the output of our re-encryption program is statistically close to $\mathsf{Enc}_{\widehat{\mathsf{pk}}}(m)$ for all honestly generated ciphertexts $c = \mathsf{Enc}_{\mathsf{pk}}(m)$). With weak blurring, we obtain a secure obfuscation whose output distribution (on *all* ciphertexts) is computationally indistinguishable from that of the re-encryption functionality. (Moreover, re-encryptions of honestly generated ciphertexts still decrypt correctly.)

All the above constructions provide a tradeoff between using (less efficient but powerful) FHE to achieve the strongest definition of correctness and using efficient specific lattice-based schemes to achieve slightly weaker notions of correctness. Again, all these constructions satisfy a strong obfuscation-based notion of security (average case VBB [25] and collusion resistance [13]).

Functional re-encryption and collusion-resistant obfuscation. Once we construct the basic re-encryption schemes, we turn our attention towards a more complex primitive, known as functional re-encryption, which incorporates access control into the re-encryption functionality. The work of Chandran, Chase, and Vaikuntanathan [13] introduced this primitive and showed an obfuscation-based result. Informally, a program implementing functional re-encryption is parameterized by an input public key pk, n output public keys $\widehat{\mathsf{pk}}_1, \cdots, \widehat{\mathsf{pk}}_n$, and an access policy $F : [D] \to [n]$. The program takes as input a ciphertext of message m with tag $i \in [D]$ under input public key pk and outputs a ciphertext of the same message m under $\widehat{\mathsf{pk}}_{F(i)}$. Functional re-encryption can be used to implement a server that forwards a user Alice's email to other recipients, depending on the tag (or the content) of the email, but at the same time hides the message and the access policy from the server. Chandran *et al.* also introduced the notion of collusion-resistant obfuscation in the context of functional re-encryption, which, informally, guarantees that the obfuscated program remains secure even when the server can collude with some of the recipients. They gave a pairing-based construction of functional re-encryption (for access policies with poly-size domain) satisfying collusion-resistant obfuscation.

Using our framework, we obtain constructions with varied levels of correctness and efficiency, similar to the tradeoffs in our constructions of the basic re-encryption primitive. All of our constructions satisfy the strong security definition of the collusion-resistant obfuscation. We remark that in our schemes the size of the input ciphertext is constant (as opposed to the construction of [13], in which the size of the input ciphertext is $\mathcal{O}(D)$). Our output ciphertext, on the other hand is of size $\mathcal{O}(n)$ (as opposed to constant in [13]); however, each of the n recipients still only needs to receive a constant size block of that ciphertext.

Multi-hop unidirectional proxy re-encryption. Traditionally, most re-encryption schemes are single-hop, in the sense that the ciphertext produced by the re-encryption process is of a different form and cannot be re-encrypted again. The exception are a few schemes beginning with [7] which are multi-hop, but bi-directional, which means that any re-encryption key which allows re-encryption from Alice to Bob also allows re-encryption from Bob to Alice (and thus both secret keys are necessary to generate the re-encryption key). In many settings however, this is not desirable - intuitively, Bob should not need to trust Alice in order for Alice to be able to forward her mail to Bob. Thus, it seems desirable

to have a scheme which allows the output of the re-encryption process to be re-encrypted again, but which does not require this kind of trust. That is the problem we consider here (referred to from here on as multi-hop re-encryption).

In this work, we present the first obfuscation based definitions and constructions for multi-hop unidirectional proxy re-encryption schemes. We remark that the problem of constructing multi-hop re-encryption schemes was first posed in [5]; a major drawback of previous schemes [21, 14] is that the ciphertext size grows linearly with the number of re-encryptions. Here we construct L-hop re-encryption schemes (where a ciphertexts can be re-encrypted up to L times) in which ciphertexts do not grow with re-encryption.[4]

Our results also translate to the ideal lattice setting based on the ring-LWE assumption [26]. For simplicity, we focus here on the general lattice setting.

2 Definitions for Obfuscation

In this section, we present our relaxed definitions of correctness in average-case secure obfuscation. We first recall the definition of average-case secure obfuscation with collusion as defined by Chandran *et al.* [13] and present the relaxed definitions of correctness with respect to this definition. As the Chandran et al definition is a generalization of the average case obfuscation definition by Hohenberger et al [25], similar relaxations can also be applied in that setting.

Informally, average-case obfuscation guarantees that obfuscation hides the program as long as it is chosen at random from a given family; resistance against collusion addresses the case where we would like these obfuscation guarantees to hold even when some types of information about the program being obfuscated may be available to the adversary. (This for example captures the case where the adversary in a re-encryption scheme holds both the obfuscated re-encryption program and some of the decryption keys.)

More formally, we consider families $\{C_\lambda\}$ that have the following form. Any $C_\mathcal{K} \in C_\lambda$ is parameterized by a set of "secret" keys $\mathcal{K} = \{k_1, k_2, \cdots, k_\ell\}$ (potentially in addition to any other parameters) 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 $\mathsf{Obf}(C_\mathcal{K})$, so that $\mathsf{Obf}(C_\mathcal{K})$ is a "secure obfuscation" of $C_\mathcal{K}$ (in the sense of [25]) even against an adversary that knows the set of keys $\{k_i\}_{i\in\mathcal{T}}$. More precisely, in addition to their usual inputs and oracles, [13] give both the adversary

[4] L-hop *re*-encryption does not follow from i-hop encryption [18]: the latter allow users to evaluate multiple functions sequentially and homomorphically only under *one* public key.

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 $C_{\mathcal{K}} \leftarrow C_\lambda$.

Finally, we modify the definition to allow some parts of the circuit to be hidden in a worst case sense. This was addressed in [13] for the case of functional re-encryption by adding an additional definition saying that an obfuscation is secure with respect to a class of functions $\mathbb{F}$ if there exists a simulator Sim which satisfies the collusion resistant average-case black box property for all $f \in \mathbb{F}$. It seems more natural and more general to incorporate this directly into the definition of secure obfuscation, so that is the approach we will take here. The formal definition of collusion-resistant secure obfuscation is as follows.

Definition 2.1 (Average-case Obfuscation with Collusion). *Let $\{C_\lambda\}$ be a family of circuits $C_{\mathcal{K},w}$ indexed by values from the sets $\mathbb{K}(\lambda)$ and $\mathbb{W}(\lambda)$, where each $\mathcal{K} \in \mathbb{K}$ is of the form $(k_1, \ldots, k_\ell)$. A PPT algorithm Obf that takes as input a (probabilistic) circuit and outputs a new (probabilistic) circuit is a collusion-resistant secure obfuscator for the family $\{C_\lambda\}$ in the average-case over $\mathbb{K}$ and in the worst case over $\mathbb{W}$, if it satisfies the following properties:*

Preserving functionality: *There exists a negligible function $\mathsf{ngl}(\cdot)$ such that for any input length λ and any $C \in C_\lambda$:*

$\Pr[\exists x \in \{0,1\}^\lambda : C' \leftarrow \mathsf{Obf}(C); \Delta(C'(x), C(x)) \geq \mathsf{ngl}(\lambda)] < \mathsf{ngl}(\lambda)$,
where $\Delta(\cdot,\cdot)$ denotes statistical distance, and the probability is taken over the random coins of Obf.

Polynomial slowdown: *There exists a polynomial $p(\lambda)$ such that for sufficiently large λ, for any $C \in C_\lambda$, $|\mathsf{Obf}(C)| \leq p(|C|)$.*

Average case virtual black-boxness (ACVBB) against collusion with worst-case hiding over $\mathbb{W}$: *For any $w \in \mathbb{W}(\lambda)$, let $C_{\lambda,w}$ be the set of circuits $C_{\mathcal{K},w} \in C_\lambda$. (When w is fixed, we specify a circuit in $C_{\lambda,w}$ by $C_{\mathcal{K}}$.) There exists a PPT simulator Sim and a negligible function $\mathsf{ngl}(\cdot)$ such that for all PPT distinguishers D, all sufficiently long input lengths λ, all $w \in \mathbb{W}(\lambda)$, and all subsets $\mathcal{T} \in [\ell]$:*

$$\left| \begin{array}{l} \Pr[C_{\mathcal{K}} \leftarrow C_{\lambda,w} : D^{C_{\mathcal{K}}}(\mathsf{Obf}(C_{\mathcal{K}}), \{k_i\}_{i\in\mathcal{T}}) = 1] \\ -\Pr[C_{\mathcal{K}} \leftarrow C_{\lambda,w} : D^{C_{\mathcal{K}}}(\mathsf{Sim}^{C_{\mathcal{K}}}(1^\lambda, \{k_i\}_{i\in\mathcal{T}}), \{k_i\}_{i\in\mathcal{T}}) = 1] \end{array} \right| < \mathsf{ngl}(\lambda).$$

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

Note that in the case where we do not wish to consider collusion-resistance, one can simply use the same definition as above where $\mathcal{T}$ is the empty set and $\ell = 1$. Our relaxations of correctness in secure obfuscation, which we will discuss below, apply to the non-collusion case as well.

Relaxed correctness in secure obfuscation. We next proceed to show how we can relax the "preserving functionality" notion defined above. This will enable

us to obtain more efficient constructions for various functionalities related to re-encryption. We shall relax this notion in three different ways: the first relaxation informally guarantees that the output distribution of the obfuscated program and the ideal functionality are statistically close only on a subset of all the inputs to the functionality; the second relaxation informally guarantees that on a subset of all the inputs to the functionality, and for some algorithm Dec (this algorithm would typically be a decryption algorithm), the output of Dec applied to the output of the program and the output of Dec applied to the output of the ideal functionality results in the same value; the third relaxation informally guarantees that the output of the program and the output of the functionality, on a subset of all the inputs, are computationally indistinguishable to all PPT adversaries (typically, this subset is parameterized by the set of corrupted parties in the system and this captures the idea that on inputs where the ideal functionality produces encryptions under honest parties' keys, the adversary shouldn't be able to distinguish the output of the obfuscated program from the ideal program). Note that, in most cases, this third property only makes sense in combination with one of previous two relaxations, because we do want some guarantee that the obfuscated program works as expected; in our re-encryption case, for example, we can combine this with the second relaxed correctness, to ensure that the program's output is indistinguishable from random encryptions, and at the same time honestly generated and re-encrypted ciphertexts decrypt correctly.

Definition 2.2 (Relaxed Average-case Obfuscation with Collusion). *For an obfuscation algorithm* Obf *which satisfies the polynomial slowdown and average-case collusion resistant virtual black-boxness properties as in Definition 2.1, we define the following relaxations of the correctness property:*

Preserving functionality with respect to Π: *Let Π be a set of pairs $(\mathcal{K}, x)$ where $\mathcal{K}$ is an index for the circuit and x is an input. The obfuscated circuit is guaranteed to agree with the original circuit only on input pairs in the subset Π. That is, there exists a negligible function $\mathsf{ngl}(\cdot)$ such that for any input length λ and any $C_{\mathcal{K}} \in C_\lambda$, and every x such that $(\mathcal{K}, x) \in \Pi$:* $\Pr[C'_{\mathcal{K}} \leftarrow \mathsf{Obf}(C_{\mathcal{K}}); \Delta(C'_{\mathcal{K}}(x), C_{\mathcal{K}}(x)) \geq \mathsf{ngl}(\lambda)] < \mathsf{ngl}(\lambda)$, *where the probability is over the random coins of* Obf. *For inputs outside Π, there is no guarantee for the output of $C'_{\mathcal{K}}(x)$. When Π is the set of all possible inputs, this corresponds to the standard notion of "preserving functionality" (Definition 2.1).*

Preserving $\mathcal{D}ec$ correctness with respect to Π: *Let Π be a set of pairs $(\mathcal{K}, x)$ where $\mathcal{K}$ is an index for the circuit and x is an input, and $\mathcal{D}ec(\cdot, \cdot)$ be some algorithm. The obfuscated circuit is guaranteed to agree with the original circuit only on input pairs in the subset Π, under the algorithm $\mathcal{D}ec$. That is, for all $(\mathcal{K}, x) \in \Pi$, for all $C'_{\mathcal{K}} \leftarrow \mathsf{Obf}(C_{\mathcal{K}})$, we require that* $\Pr[y \leftarrow C_{\mathcal{K}}(x), y' \leftarrow C'_{\mathcal{K}}(x) : \mathcal{D}ec(\mathcal{K}, y) = \mathcal{D}ec(\mathcal{K}, y')] = 1 - \mathsf{ngl}(\lambda)$ *for some negligible* ngl.

Computationally preserving functionality with respect to $\Pi_{\bar{\mathcal{T}}}$: *Let $\bar{\mathcal{T}}$ be a set in $[\ell]$ (usually the set $\mathcal{T} = [\ell] \setminus \bar{\mathcal{T}}$), and let $\Pi_{\bar{\mathcal{T}}}$ be a subset (potentially dependent on $\mathcal{T}$) of pairs $(\mathcal{K}, x)$ where $\mathcal{K} = (k_1, \ldots, k_\ell)$ is an index for circuit and x is an input. For any pair of circuits C, C', denote by $O_{\mathcal{K},C,C'}(\cdot)$ the program that on input x, outputs $C'(x)$ if $(\mathcal{K}, x) \in \Pi_{\bar{\mathcal{T}}}$ and $C(x)$ otherwise. Then for all PPT adversaries A, we require that:*

$$\left| \begin{array}{r} \Pr\left[C_{\mathcal{K}} \leftarrow C_\lambda, C'_{\mathcal{K}} \leftarrow \mathsf{Obf}(C_{\mathcal{K}}) : A^{O_{\mathcal{K},C_{\mathcal{K}},C'_{\mathcal{K}}}(\cdot)}(\{k_i\}_{i\in\mathcal{T}}) = 1\right] \\ - \Pr\left[C_{\mathcal{K}} \leftarrow C_\lambda : A^{C_{\mathcal{K}}(\cdot)}(\{k_i\}_{i\in\mathcal{T}}) = 1\right] \end{array} \right| < \mathsf{ngl}(\lambda).$$

3 Our Framework and Instantiations

In this section, we define and construct several new tools which will be useful for our applications. First we present two abstract properties, and argue that we can implement them trivially with FHE. Then we show they can be achieved much more efficiently for the Regev [27] and dual Regev [19] encryption schemes.

3.1 Notions of Key-Switching and Blurring

Key switching. Let $\Sigma = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be a semantically secure encryption scheme. The first property we consider is the existence of a key-switching mechanism. Here we formalize a property based on an idea from Brakerski and Vaikuntanathan [10]: briefly, a key-switching mechanism allows one to directly convert ciphertexts encrypted under one public key to ciphertexts encrypted under a second public key. More formally our definition is as follows:

Definition 3.1 (Key-Switching Mechanism). *A* key-switching mechanism *for an encryption scheme $\Sigma = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ consists of two algorithms:*

$-\mathsf{SwGen}(\mathsf{pk}, \mathsf{sk}, \widehat{\mathsf{pk}})$: Let $(\mathsf{pk}, \mathsf{sk})$ be a pair of "source" keys output by Gen, and $\widehat{\mathsf{pk}}$ be a "target" public key ($(\widehat{\mathsf{pk}}, \cdot)$ is output by Gen). The algorithm takes $(\mathsf{pk}, \mathsf{sk}, \widehat{\mathsf{pk}})$ as input, and outputs a switch-key $K_{\mathsf{pk}\to\widehat{\mathsf{pk}}}$ that can transform ciphertexts encrypted under pk to ciphertexts encrypted under $\widehat{\mathsf{pk}}$.

$-\mathsf{Sw}(K_{\mathsf{pk}\to\widehat{\mathsf{pk}}}, c)$: The algorithm takes a switch-key $K_{\mathsf{pk}\to\widehat{\mathsf{pk}}}$ and a ciphertext c as input, and outputs a ciphertext $\hat{c}$.

The key-switching mechanism is correct *if for all $(\mathsf{pk}, \mathsf{sk}), (\widehat{\mathsf{pk}}, \widehat{\mathsf{sk}}) \leftarrow \mathsf{Gen}(1^\lambda)$, for all $K_{\mathsf{pk}\to\widehat{\mathsf{pk}}} \leftarrow \mathsf{SwGen}(\mathsf{pk}, \mathsf{sk}, \widehat{\mathsf{pk}})$, for all $m \in \{0,1\}$ and for all $c \leftarrow \mathsf{Enc}_{\mathsf{pk}}(m)$, $c' \leftarrow \mathsf{Sw}(K_{\mathsf{pk}\to\widehat{\mathsf{pk}}}, c)$, it holds that $\mathsf{Dec}_{\widehat{\mathsf{sk}}}(c') = m$. More generally, the key-switching mechanism is* correct on set $\Pi = \{(\mathsf{pk}, \mathsf{sk}, c)\}$ *if for all $(\widehat{\mathsf{pk}}, \widehat{\mathsf{sk}}) \leftarrow \mathsf{Gen}(1^\lambda)$, for all $K_{\mathsf{pk}\to\widehat{\mathsf{pk}}} \leftarrow \mathsf{SwGen}(\mathsf{pk}, \mathsf{sk}, \widehat{\mathsf{pk}})$, and for all $c' \leftarrow \mathsf{Sw}(K_{\mathsf{pk}\to\widehat{\mathsf{pk}}}, c)$, it holds that $\mathsf{Dec}_{\widehat{\mathsf{sk}}}(c') = \mathsf{Dec}_{\mathsf{sk}}(c)$.*

Remark 3.2. The idea of a key-switching mechanism was introduced by Brakerski and Vaikuntanathan [10] to construct fully homomorphic encryption schemes. They used an approach where the SwGen algorithm is given pk, sk and then samples $\widehat{\mathsf{pk}}, \widehat{\mathsf{sk}}$ on its own, and outputs a switch-key that allows one to transform ciphertexts under pk to $\widehat{\mathsf{pk}}$. This suffices for the construction of fully homomorphic encryption. However, for our applications we require the switch-key generation algorithm to take the source keys and the target public key as input, and to output the switch-key without knowing the secret key $\widehat{\mathsf{sk}}$.

To make key-switching an interesting notion, we need some property guaranteeing at the very least that the switch-key does not allow the holder to decrypt messages. We require something stronger, essentially that the switch-key reveals nothing at all about the input public key to anyone who does not hold either of the secret keys. We capture this with a simulation based definition:

Definition 3.3 (Security of Key-Switching Mechanism). *We say the Key-Switching Mechanism is secure if there exists a simulated key generation algorithm* $\mathsf{SimSwGen}(\widehat{\mathsf{pk}})$ *that only takes as input the target public key (and not the source keys) and can output a switch-key such that for any PPT adversary the following two distributions are indistinguishable:*

$$\{(\mathsf{pk},\mathsf{sk}), (\widehat{\mathsf{pk}},\widehat{\mathsf{sk}}) \leftarrow \mathsf{Gen}(1^\lambda); K_{\mathsf{pk}\to\widehat{\mathsf{pk}}} \leftarrow \mathsf{SwGen}(\mathsf{pk},\mathsf{sk},\widehat{\mathsf{pk}}) : (\mathsf{pk}, \widehat{\mathsf{pk}}, K_{\mathsf{pk}\to\widehat{\mathsf{pk}}})\}$$

$$\{(\mathsf{pk},\mathsf{sk}), (\widehat{\mathsf{pk}},\widehat{\mathsf{sk}}) \leftarrow \mathsf{Gen}(1^\lambda); K_{\mathsf{pk}\to\widehat{\mathsf{pk}}} \leftarrow \mathsf{SimSwGen}(\widehat{\mathsf{pk}}) : (\mathsf{pk}, \widehat{\mathsf{pk}}, K_{\mathsf{pk}\to\widehat{\mathsf{pk}}})\}$$

Blurring. The second property that we consider is what we call a blurring mechanism. At a high level, the goal is to take a ciphertext and produce a new unrelated-looking ciphertext that encrypts the same message. This kind of re-randomization is hard to achieve in lattice-based constructions, so we relax this restriction somewhat and consider definitions in which guarantees only hold for a restricted set of ciphertexts, or against computationally bounded adversaries.

Informally, weak blurring says that if we take any string c and blur it, then this is indistinguishable from the string produced by taking a new ciphertext of some (perhaps different) message and blurring it, even given the ciphertext c (but not the secret key). This is true for all strings c and not just "well-formed" (or honestly generated) ciphertexts. Furthermore, the blurred ciphertext and c will still decrypt to the same message for the "well-formed" ciphertexts c. Strong blurring, on the other hand, additionally says that if we take a "well-formed" ciphertext c and blur it, then this is indistinguishable from the string produced

by taking a new ciphertext of the same message and blurring it, even given the secret key sk and the ciphertext c. (This follows from statistical closeness of the two distributions). More formally, we define these properties as follows:

Definition 3.4 (Blurring). *Given an encryption scheme $\Sigma = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$, we consider the following two blurring properties: Let Π be a set of public-key, secret-key, ciphertext tuples, i.e. $(\mathsf{pk}, \mathsf{sk}, c)$. Let $\mathsf{Blur}(\mathsf{pk}, c)$ be an algorithm which takes as input a public key and a ciphertext and produces a new ciphertext $\tilde{c}$. Then we can consider the following two properties:*

Weak Blurring: *We say* Blur *is a weak blurring mechanism where the correctness holds for Π if the following two properties hold.*

(1) Hiding: for any PPT adversary A, the following are indistinguishable.
Experiment 0: $\mathsf{pk} \leftarrow \mathsf{Gen}(1^\lambda), c \leftarrow A(\mathsf{pk})$*, output* $(\mathsf{pk}, c, \mathsf{Blur}(\mathsf{pk}, c))$.
Experiment 1: $\mathsf{pk} \leftarrow \mathsf{Gen}(1^\lambda), c \leftarrow A(\mathsf{pk})$*, output* $(\mathsf{pk}, c, \mathsf{Enc}_{\mathsf{pk}}(0))$.

(2) Correctness: There exists negligible ngl *such that, for all* $(\mathsf{pk}, \mathsf{sk}, c) \in \Pi$*,*
$$\Pr[\hat{c} \leftarrow \mathsf{Blur}(\mathsf{pk}, c) : \mathsf{Dec}_{\mathsf{sk}}(\hat{c})) = \mathsf{Dec}_{\mathsf{sk}}(c)] = 1 - \mathsf{ngl}(\lambda).$$

Strong Blurring: *We say* Blur *is a strong blurring mechanism with respect to Π if it is a weak blurring mechanism where correctness holds for Π with the following additional property: For every $(\mathsf{pk}, \mathsf{sk}, c) \in \Pi$, let $m = \mathsf{Dec}_{\mathsf{sk}}(c)$; then we require that $\Delta\big((c, \mathsf{Blur}(\mathsf{pk}, c)), (c, \mathsf{Blur}(\mathsf{pk}, \mathsf{Enc}_{\mathsf{pk}}(m)))\big) < \mathsf{ngl}(\lambda)$.*

We note that many existing works consider similar definitions of re-randomization[5] . Strong blurring where Π is the set of all ciphertexts and valid key pairs would be equivalent to the definition in [23]. Weak blurring where Π is the set of all ciphertexts and valid key pairs is very similar in spirit to the definition of semantic security for universal re-encryption presented in [20].

One direct application of such a blurring mechanism is to achieve function privacy for any fully homomorphic encryption (FHE) scheme for which we can blur the ciphertexts produced by the evaluation algorithm. (See the full version.)

Implementations using Function Private FHE. We note that both of these properties can be achieved easily given an appropriate FHE scheme. Given a key private and function private FHE, we can construct a key-switching mechanism by evaluating the decryption circuit as discussed in the introduction. We can build strong blurring with respect to *all inputs* similarly. (See the full version.)

As a consequence, we can use the lattice-based FHE by Brakerski [8] (based on Regev's encryption) and our blurring mechanism for Regev-based schemes (see the next section) to implement an encryption scheme that has: (1) key

[5] We remark here, that our blurring technique is similar in spirit to the smudging technique proposed by Asharov *et al.* [4]. However, we abstract out the technique and formally define "blurring," independent of any specific encryption construction.

switching, (2) strong blurring with respect to *all inputs*, and, (3) key privacy. In our constructions of functional re-encryption and multi-hop re-encryption, this approach gives the strongest obfuscation results, at the cost of efficiency.

3.2 Implementations using Regev's Encryption Scheme

Recall that Regev's encryption scheme has the following structure: $\mathsf{pk} = (A, b)$ where $A \in \mathbb{Z}_q^{N \times n}$, $b \in \mathbb{Z}_q^N$, and $\mathsf{sk} = s \in \mathbb{Z}_q^n$ where $b = A \cdot s + e$ for some noise vector e, sampled from some distribution χ^N where χ is B-bounded. The encryption has the following structure: $c = (c_1, c_2) = r^\top \cdot (A, b) + (0^n, m \cdot \lfloor q/2 \rfloor)$ where r is a random vector in $\{0,1\}^N$. For details, see [27]. In what follows, let $\Sigma = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be Regev's encryption scheme.

Key-switching mechanism. As discussed in Remark 3.2, the requirements on key-switching in the context of re-encryption are slightly different from those in the FHE application. Thus, the construction from [10] does not work directly. We now show how we can modify that scheme to obtain a key-switching algorithm which does satisfy our requirements. Consider the following algorithms:

$\mathsf{SwGen}(\mathsf{pk}, \mathsf{sk}, \widehat{\mathsf{pk}})$: Parse $\mathsf{sk} = s \in \mathbb{Z}_q^n$. For $i \in [n], \tau \in [\lceil \log q \rceil]$, compute $K_{i,\tau} \leftarrow \mathsf{Enc}_{\widehat{\mathsf{pk}}}(0) + (0^n, s_i \cdot 2^\tau)$, where s_i denotes the i-th component of the vector s. Output $K_{\mathsf{pk} \to \widehat{\mathsf{pk}}} = \{K_{i,\tau}\}_{i \in [n], \tau \in [\lceil \log q \rceil]}$.

$\mathsf{SimSwGen}(\widehat{\mathsf{pk}})$: Let n, q be the parameters from $\widehat{\mathsf{pk}}$. For $i \in [n], \tau \in [\lceil \log q \rceil]$, compute $K_{i,\tau} \leftarrow \mathsf{Enc}_{\widehat{\mathsf{pk}}}(0)$, and output $K_{\mathsf{pk} \to \widehat{\mathsf{pk}}} = \{K_{i,\tau}\}_{i \in [n], \tau \in [\lceil \log q \rceil]}$.

$\mathsf{Sw}(K_{\mathsf{pk} \to \widehat{\mathsf{pk}}}, c)$: first parse $c = (c_1, c_2) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$, and $K_{\mathsf{pk} \to \widehat{\mathsf{pk}}} = \{K_{i,\tau}\}_{i \in [n], \tau \in [\lceil \log q \rceil]}$. Denote by $c_{1,i}$ the i-th component of c_1, and denote the bit-decomposition of $c_{1,i}$ as $\{c_{1,i,\tau}\}_{\tau \in [\lceil \log q \rceil]}$, i.e. $c_{1,i} = \sum_{\tau \in [\lceil \log q \rceil]} c_{1,i,\tau} 2^\tau$, where each $c_{1,i,\tau} \in \{0,1\}$. Then output $\hat{c} = (0^n, c_2) + \sum_{i,\tau} c_{1,i,\tau} \cdot K_{i,\tau}$

The above construction has the same structure as the one in [10], so the correctness and security follow from the DLWE assumption. (See the full version.)

Blurring mechanism. Consider the following two blurring algorithms:

$\mathsf{SBlur}(\mathsf{pk}, c; E)$, where $E \in \mathbb{Z}$ is a parameter hardcoded into the algorithm defining an appropriate error distribution (we will consider SBlur with different values for E): Parse $\mathsf{pk} = (A, b) \in \mathbb{Z}_q^{N \times n} \times \mathbb{Z}_q^N$, sample $f \leftarrow [-E, E] \cap \mathbb{Z}$, and output $c + \mathsf{Enc}_{\mathsf{pk}}(0) + (0^n, f)$.

$\mathsf{WBlur}(\mathsf{pk}, c)$: Output $c + \mathsf{Enc}_{\mathsf{pk}}(0)$.

Our idea for weak blurring is simple. We just add an encryption of 0 to the ciphertext. Since the distribution of $\mathsf{Enc}_{\mathsf{pk}}(0)$ is pseudo-random for Regev's encryption scheme, doing this computationally blurs the output. Also, Regev's

encryption scheme is additively homomorphic (with a small blow up of noise), so this preserves the correctness of decryption.

For strong blurring, our idea is to blur the randomness as well. We recall that the ciphertext c has the form $(u, u^\top \cdot s) + (0^n, m \cdot [q/2]) + (0^n, z)$ where $z = r^\top \cdot e$ is the error term and $u = r^\top \cdot A$. Adding an encryption of 0 will blur our the first term $(u, u^\top \cdot s)$ (by a leftover hash lemma argument). The additional error e will blur out the last term z. For $E \cdot \lambda^{\omega(1)} < q/4$, decryption will still be correct. This idea also allows us to blur a subset sum of polynomially many ciphertexts. Thus, we can blur the ciphertexts after the key switching algorithm above. For a detailed analysis of weak and strong blurring see the full version.

3.3 Implementations using the dual Regev encryption scheme

In this section, we present another implementation of these mechanisms using the dual Regev encryption scheme. We remark that the dual Regev scheme appeared in [19], but we make a slight modification to the ciphertext and secret key that allows us to implement a key-switching mechanism.

The dual Regev encryption scheme we use here has the following structure: $\mathsf{pk} = (A, u)$ where $A \in \mathbb{Z}_q^{n \times N}$ and $u \in \mathbb{Z}_q^n$ are uniformly random, and $\mathsf{sk} = S \in \mathbb{Z}_q^{N \times N}$ such that S is a short basis of $\Lambda^\perp(A)$. The encryption has the following structure: $c = (c_1, c_2) = s^\top \cdot (A, u) + e^\top + (0^N, m \cdot [q/2])$ where $s \leftarrow \chi^n$, $e \leftarrow \chi^{N+1}$ are noise vectors, sampled (independently) from some B-bounded distribution χ. For details see the full version. In what follows, let $\Sigma = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be the dual Regev encryption scheme.

Key-switching mechanism. Consider the following algorithms:

$\mathsf{SwGen}(\mathsf{pk}, \mathsf{sk}, \widehat{\mathsf{pk}})$: Parse $\mathsf{pk} = (A, u_A) \in \mathbb{Z}_q^{n \times N} \times \mathbb{Z}_q^n$, $\widehat{\mathsf{pk}} = (B, u_B) \in \mathbb{Z}_q^{n' \times N'} \times \mathbb{Z}_q^{n'}$, and $\mathsf{sk} = S \in \mathbb{Z}_q^{N \times N}$. First sample short noise matrices $V \leftarrow \chi^{n' \times n'}$, $X \leftarrow \chi^{n' \times (N'+1)}$. Let $(\tilde{B}, \tilde{u}_B) = V \cdot (B, u_B) + X$. Then sample some short $Z \in \mathbb{Z}_q^{N \times N'}, z \in \mathbb{Z}_q^N$ such that $A \cdot (Z, z) = (\tilde{B}, \tilde{u}_B - u_A)$. This can be done by using the sampling algorithm $\mathsf{SampleD}(S, A, \cdot, \sigma_z)$ at each column of the matrix $(\tilde{B}, \tilde{u}_B - u_A)$, together with the secret key S, and parameter $\sigma_z = \omega(\sqrt{n \log n \log q})$. Finally output $K_{\mathsf{pk} \to \widehat{\mathsf{pk}}} = (Z, z)$.

$\mathsf{SimSwGen}(\widehat{\mathsf{pk}})$: Let n, q be the parameters from $\widehat{\mathsf{pk}}$ and $\sigma_z = \omega(\sqrt{n \log n \log q})$ be an additional parameter of the encryption. Output (Z, z) chosen by taking $N' + 1$ independent samples from $D_{\mathbb{Z}^N, \sigma_z}$ (a discrete Gaussian on $\mathbb{Z}^N$ with parameter σ_z).

$\mathsf{Sw}(K_{\mathsf{pk} \to \widehat{\mathsf{pk}}}, c)$: first parse $c = (c_1, c_2) \in \mathbb{Z}_q^N \times \mathbb{Z}_q$, and $K_{\mathsf{pk} \to \widehat{\mathsf{pk}}} = (Z, z) \in \mathbb{Z}_q^{N \times N'} \times \mathbb{Z}_q^N$. Output $\hat{c} = (c_1, c_2) \cdot \left(\begin{smallmatrix} Z & z \\ 0 & 1 \end{smallmatrix}\right)$.

The correctness of the construction follows by a direct examination. Take an encryption of 0 for example: let $c = \mathsf{Enc}(0) = s^\top(A, u_A) + e^\top$. If we apply the switch key algorithm, we get a transformed ciphertext:

$$s^\top(A \cdot Z, A \cdot z + u_A) + e^\top = s^\top(\tilde{B}, \tilde{u}_B) + e^\top = s^\top V \cdot (B, u_B) + s^\top \cdot X + e^\top$$

Since V, X and s are short, we can view $s^\top V$ as another short $s'^\top$, and $s^\top \cdot X + e^\top$ as a slightly larger error $e'^\top$. Thus, this transformed ciphertext can be decrypted correctly.

The security argument is slightly trickier. First we observe that the matrix $(\tilde{B}, \tilde{u}_B)$ is computationally indistinguishable from a uniformly random matrix. This is because the security of DLWE holds even if the secret is sampled from the noise distribution $\chi^{n'}$ as shown by Applebaum et al. [3]. Thus, the distribution (Z, z) such that $A \cdot (Z, z) = (\tilde{B}, \tilde{u}_B)$ is computationally indistinguishable from the distribution (Z', z') such that $A \cdot (Z', z') = (U, u)$ where (U, u) is a uniformly random matrix. As shown by Gentry et al. [19], (Z, z) can be sampled (up to a negligible statistical distance) by the $\mathsf{SampleD}(A, S, \cdot, \sigma_z)$ as above, and (Z', z') is just the discrete Gaussian on $\mathbb{Z}^N$ with parameter σ_z. Thus, the security holds. For formal statements and proofs see the full version.

Blurring mechanism. Consider the following two blurring algorithms:

$\mathsf{SBlur}(\mathsf{pk}, c; E)$, where $E \in \mathbb{Z}$ is a parameter hardcoded into the algorithm: First parse $\mathsf{pk} = (A, u) \in \mathbb{Z}_q^{n \times N} \times \mathbb{Z}_q^n$. Then sample $p \leftarrow (E \cdot \chi)^n$, and $e \leftarrow (E \cdot \chi)^{N+1}$, and output $c + p^\top \cdot (A, u) + e^\top$.

$\mathsf{WBlur}(\mathsf{pk}, c)$: Output $c + \mathsf{Enc}_{\mathsf{pk}}(0)$.

Our idea for weak blurring is simple. We just add an encryption of 0. Since $\mathsf{Enc}_{\mathsf{pk}}(0)$ is pseudo-random, it will computationally blur the output. Also, the dual Regev encryption scheme is additively homomorphic (with a small blow up of noise), so it won't hurt the correctness.

For strong blurring, we need to blur the randomness as well. Recall that the ciphertext is of the form $s^\top \cdot (A, u) + e^\top$ for $s \leftarrow \chi^n, e \leftarrow \chi^{N+1}$ where χ is a B-bounded distribution. Suppose the distribution has the following property: $(E \cdot \chi)^N$ is statistically close to $y + (E \cdot \chi)^N$ for any $y \in \mathbb{Z}_q^N$ such that $\|y\|_\infty \leq B$. Then we can simply use $E \cdot \chi$ to blur the randomness. In fact, if χ is the Gaussian distribution (as it is in our setting) and the parameters satisfy $E \geq B \cdot \lambda^{\omega(1)}$, then this property can be achieved. See the full version for details.

Remark 3.5. We also propose an alternative implementation for key-switching in the dual Regev encryption scheme. The key observation is that the key-switching mechanism in Regev's encryption scheme as described in Section 3.2 can be easily adapted to the dual Regev scheme. Since the dual Regev scheme has the same structure for the decryption algorithm, (i.e. it computes the inner

product of a ciphertext and a secret key, as Regev's scheme does for its decryption algorithm), a key-switching mechanism can be obtained in the same way. On the other hand, we will keep the same the blurring mechanism as above.

Remark 3.6. The dual Regev encryption scheme can be extended to a variety of identity-based encryption (IBE) and hierarchical identity-based encryption (HIBE) schemes as shown in [19, 12, 1, 2]. We further observe that our constructions of key switching and blurring in the dual Regev scheme can be naturally extended to these dual Regev based (H)IBE schemes.

4 Applications of our tools

In this section, we sketch how we can use the tools developed in the previous section to construct secure obfuscators for various re-encryption based primitives. More detailed descriptions appear in the full version. For each primitive we first define an ideal circuit family whose obfuscation would give a solution to the problem, and then we show how to obfuscate it.

4.1 Obfuscating Re-encryption

We first construct a simple re-encryption scheme. In re-encryption a user Alice with public key pk wants to allow an untrusted server to translate ciphertexts encrypted under her public key into ciphertexts encrypting the same message under the public key $\widehat{\mathsf{pk}}$ of another user Bob. She generates a re-encryption program, which the server can use to perform the translation without decrypting.

The ideal re-encryption circuit family. Each circuit $C_{\mathsf{pk},\mathsf{sk},\widehat{\mathsf{pk}}}$ is parameterized by a source key pair $(\mathsf{pk}, \mathsf{sk})$, and a target public key $\widehat{\mathsf{pk}}$. On input ciphertext c, it decrypts using sk, encrypts the result under $\widehat{\mathsf{pk}}$, and outputs the resulting $\hat{c}$.

Obfuscating re-encryption. Intuitively, if Alice could obfuscate the above circuit, then she could give the resulting program to the server. The program would have the same functionality, so it would allow the server to correctly translate ciphertexts from pk to $\widehat{\mathsf{pk}}$. At the same time it would reveal no more information than if the server had access to a trusted party who would compute re-encryption for it; in particular, this means the program would not help the server at all in decrypting messages as long as it doesn't know Bob's secret key. (If Bob and the server collude, they can of course decrypt any messages encrypted for Alice, but this is inherent in the functionality of re-encryption.)

We build on an encryption scheme with a key-switching mechanism and a blurring mechanism. To obfuscate $C_{\mathsf{pk},\mathsf{sk},\widehat{\mathsf{pk}}}$, Obf (1) computes the re-encryption

key as $K_{\mathsf{pk}\to\widehat{\mathsf{pk}}} \leftarrow \mathsf{SwGen}(\mathsf{pk}, \mathsf{sk}, \widehat{\mathsf{pk}})$, and (2) generates the description of a re-encryption program that has the re-encryption key $K_{\mathsf{pk}\to\widehat{\mathsf{pk}}}$ hardcoded and on input ciphertext c computes and outputs $\hat{c} \leftarrow \mathsf{Blur}(\widehat{\mathsf{pk}}, \mathsf{Sw}(K_{\mathsf{pk}\to\widehat{\mathsf{pk}}}, c))$.

Theorem (informal) The above scheme satisfies ACVBB for the re-encryption functionality. With weak or strong blurring (resp.), it preserves functionality with respect to Π, or computationally preserves functionality and preserves Dec-correctness for Π, where Π is the set of honestly generated ciphertexts.

Interpreting the correctness guarantees. First, we note that in many scenarios, a scheme which satisfies Dec-correctness on the set Π of honestly-generated ciphertexts may be sufficient. Essentially, this says that whenever the server applies the re-encryption program to an honest ciphertext, the result will be another ciphertext which will decrypt to the correct message.

A scheme which computationally preserves functionality with respect to the set of all ciphertexts essentially guarantees that for any party without Bob's secret key, the output of the re-encryption program looks like a fresh random encryption. In particular, any party who eavesdrops on ciphertexts sent to the server and on the resulting ciphertexts sent to Bob will not be able to link each re-encrypted ciphertext to the original ciphertext from which it was formed.

Statistically preserving functionality with respect to the set Π of honestly-generated ciphertexts means that when the re-encryption program is applied to an honest ciphertext, *even Bob* can't distinguish the result from a freshly generated encryption. For example, if the server collects a set of ciphertexts, shuffles them, and then sends them all to Bob, even if Bob saw the original ciphertexts as they were sent to the server, he won't be able to link them to the ciphertexts he receives. This might be useful in privacy applications, e.g. if we want to guarantee that Bob can't tell who uploaded a particular message.

Finally, the standard definition of preserving functionality guarantees that the recipient Bob can't distinguish the output of the re-encryption from a fresh encryption, even if the initial ciphertext was not well formed.

4.2 Obfuscating Functional re-encryption

Functional re-encryption, introduced by [13], extends the re-encryption to allow Alice to include an access policy when forming the re-encryption key, after which the server (without learning the access policy), can convert any ciphertext encrypted under Alice's public key into a ciphertext encrypted for the appropriate recipient (depending on the message and the access policy).

As in [13], we consider a message space in which each message consists of a short tag and a potentially longer message, and specify the policy function by defining a function F which maps tags to the appropriate recipients. For now,

we consider the simple case where each tag is mapped to a different recipient. (The general case results in a larger re-encryption key; see the full version.)

The ideal circuit family. Each circuit $C_{\mathsf{pk},\mathsf{sk},\widehat{\mathsf{pk}}_1,\ldots\widehat{\mathsf{pk}}_n,F}$ is parameterized by an input key pair, a list of n output public keys, and the function F. On input ciphertext c, it decrypts c to obtain tag i and message m, then for each recipient j, if $F(i) = j$ it encrypts m under $\widehat{\mathsf{pk}}_j$, and otherwise it encrypts $\perp$ under $\widehat{\mathsf{pk}}_j$. It outputs the resulting list of ciphertexts. In our application above, the server could then forward each ciphertext to the appropriate recipient, but only the one for which $F(i) = j$ will decrypt to anything meaningful. (This circuit is somewhat different from the one in [13]; for a discussion, see the full version.)

Obfuscating functional re-encryption. Again, if we could obfuscate this functionality, we would obtain a program that Alice could safely give the server that would allow it to perform the re-encryption without learning anything about the messages. Furthermore, if we guarantee that our obfuscation worst-case hides the class of policy functions F then we know that the server will learn nothing about Alice's access policy; if the obfuscation is collusion resistant then these guarantees hold even if the server colludes with some subset of the recipients.

We build on a key-private encryption scheme with key-switching and blurring mechanisms. Roughly, Alice's public key consists of a public key pk_i for every possible i, and encryption of (m, i) for Alice computes $\Sigma.\mathsf{Enc}_{\mathsf{pk}_i}(m)$. The recipients use Σ directly. To obfuscate $C_{\mathsf{pk},\mathsf{sk},\widehat{\mathsf{pk}}_1,\ldots,\widehat{\mathsf{pk}}_n,F}$, Obf (1) computes a switch-key $K_{i\to F(i)} \leftarrow \mathsf{SwGen}(\mathsf{pk}_i, \widehat{\mathsf{pk}}_{F(i)})$ for each i (all these keys together, sorted based on $F(i)$, make up the re-encryption key rk_F), and (2) generates the description of a re-encryption program that has this rk_F hardcoded and, on input ciphertext c, computes $\hat{c}_j \leftarrow \mathsf{Blur}(\widehat{\mathsf{pk}}_j, \mathsf{Sw}(K_{F^{-1}(j)\to j}, c))$ for each $j \in [n]$ and outputs the list of ciphertexts $\hat{c}_1, \ldots, \hat{c}_n$.

Theorem (informal) This scheme satisfies collusion-resistant ACVBB with worst-case case hiding for F, and correctness depending on the blurring used.

4.3 Obfuscating multi-hop re-encryption

In multi-hop re-encryption, there are n users, each with his own key pair. Any of these users can choose to allow their messages to be re-encrypted to other users. We describe these choices with a directed graph, where each vertex corresponds to a user, and an edge from i to j in G means user i wants to allow re-encryption from ciphertexts under his public key (pk_i) to ciphertexts under pk_j. L-hop re-encryption allows each ciphertext to be re-encrypted L times. (Formally, we also assume each ciphertext reveals how many times it has been re-encrypted. We omit this below for simplicity; see the full version for details.)

The ideal circuit family for G. Each circuit $C_{\mathsf{pk}_1,\mathsf{sk}_1,\ldots,\mathsf{pk}_n,\mathsf{sk}_n}$ is parameterized by n key pairs $(\mathsf{pk}_i, \mathsf{sk}_i)$. On input i, j and a ciphertext c, if $(i, j) \in G$ it decrypts c using sk_i, then encrypts the result under pk_j, and outputs the resulting $\hat{c}$; otherwise it outputs an encryption of $\perp$ under pk_j.

The Obfuscation. If we could design many separate re-encryption programs which together form an obfuscation of the above functionality, we would obtain programs that each user could safely give the server that would allow it to perform the re-encryption without learning anything about the messages. Moreover, since this circuit distinguishes between an edge from i to j and an edge from j to i (G is a directed graph), the obfuscation would give a unidirectional re-encryption scheme.

We build on a key private encryption scheme with a key-switching mechanism and a blurring mechanism. To form a program using $(\mathsf{pk}_i, \mathsf{sk}_i), \mathsf{pk}_j$ (for $(i, j) \in G$), user i will (1) compute a re-encryption key $K_{\mathsf{pk}_i \to \mathsf{pk}_j} \leftarrow \mathsf{SwGen}(\mathsf{pk}_i, \mathsf{sk}_i, \mathsf{pk}_j)$, and (2) generate the description of a re-encryption program that has the re-encryption key $K_{\mathsf{pk}_i \to \mathsf{pk}_j}$ hardcoded and on input ciphertext c computes and outputs $\hat{c} \leftarrow \mathsf{Blur}(\mathsf{pk}_j, \mathsf{Sw}(K_{\mathsf{pk}_i \to \mathsf{pk}_j}, c))$.

Theorem (informal) The combined programs satisfy collusion-resistant ACVBB, where correctness depends on the blurring algorithm used.

References

[1] Agrawal, S., Boneh, D., and Boyen, X. Efficient lattice (H)IBE in the standard model. In *Eurocrypt* (2010), H. Gilbert, Ed., vol. 6110 of *LNCS*, Springer, pp. 553–572.

[2] Agrawal, S., Boneh, D., and Boyen, X. Lattice basis delegation in fixed dimension and shorter-ciphertext hierarchical IBE. In *Crypto* (2010), T. Rabin, Ed., vol. 6223 of *LNCS*, Springer, pp. 98–115.

[3] Applebaum, B., Cash, D., Peikert, C., and Sahai, A. Fast cryptographic primitives and circular-secure encryption based on hard learning problems. In *Crypto* (2009), S. Halevi, Ed., vol. 5677 of *LNCS*, Springer, pp. 595–618.

[4] Asharov, G., Jain, A., López-Alt, A., Tromer, E., Vaikuntanathan, V., and Wichs, D. Multiparty computation with low communication, computation and interaction via threshold FHE. In *Eurocrypt* (2012), D. Pointcheval and T. Johansson, Eds., vol. 7237 of *LNCS*, Springer, pp. 483–501.

[5] Ateniese, G., Fu, K., Green, M., and Hohenberger, S. Improved proxy re-encryption schemes with applications to secure distributed storage. In *NDSS* (2005), The Internet Society.

[6] Barak, B., Goldreich, O., Impagliazzo, R., Rudich, S., Sahai, A., Vadhan, S. P., and Yang, K. On the (im)possibility of obfuscating programs. In *Crypto* (2001), J. Kilian, Ed., vol. 2139 of *LNCS*, Springer, pp. 1–18.

[7] Blaze, M., Bleumer, G., and Strauss, M. Divertible protocols and atomic proxy cryptography. In *Eurocrypt* (1998), K. Nyberg, Ed., vol. 1403 of *LNCS*, Springer, pp. 127–144.

[8] Brakerski, Z. Fully homomorphic encryption without modulus switching from classical GapSVP. In *Crypto* (2012), R. Safavi-Naini and R. Canetti, Eds., vol. 7417 of *LNCS*, Springer, pp. 868–886.

[9] Brakerski, Z., Gentry, C., and Vaikuntanathan, V. (leveled) fully homomorphic encryption without bootstrapping. In *ITCS* (2012), S. Goldwasser, Ed., ACM, pp. 309–325.

[10] Brakerski, Z., and Vaikuntanathan, V. Efficient fully homomorphic encryption from (standard) LWE. In *FOCS* (2011), R. Ostrovsky, Ed., IEEE, pp. 97–106.

[11] Brakerski, Z., and Vaikuntanathan, V. Fully homomorphic encryption from ring-lwe and security for key dependent messages. In *Crypto* (2011), P. Rogaway, Ed., vol. 6841 of *LNCS*, Springer, pp. 505–524.

[12] Cash, D., Hofheinz, D., Kiltz, E., and Peikert, C. Bonsai trees, or how to delegate a lattice basis. *Journal of Cryptology 25*, 4 (October 2012), 601–639.

[13] Chandran, N., Chase, M., and Vaikuntanathan, V. Functional re-encryption and collusion-resistant obfuscation. In *TCC* (2012), R. Cramer, Ed., vol. 7194 of *LNCS*, Springer, pp. 404–421.

[14] Chu, C.-K., and Tzeng, W.-G. Identity-based proxy re-encryption without random oracles. In *ISC* (2007), J. A. Garay, A. K. Lenstra, M. Mambo, and R. Peralta, Eds., vol. 4779 of *LNCS*, Springer, pp. 189–202.

[15] Gentry, C. Fully homomorphic encryption using ideal lattices. In *STOC* (2009), M. Mitzenmacher, Ed., ACM, pp. 169–178.

[16] Gentry, C. Toward basing fully homomorphic encryption on worst-case hardness. In *Crypto* (2010), T. Rabin, Ed., vol. 6223 of *LNCS*, Springer, pp. 116–137.

[17] Gentry, C., and Halevi, S. Fully homomorphic encryption without squashing using depth-3 arithmetic circuits. In *FOCS* (2011), R. Ostrovsky, Ed., IEEE, pp. 107–109.

[18] Gentry, C., Halevi, S., and Vaikuntanathan, V. i-hop homomorphic encryption and rerandomizable Yao circuits. In *Crypto* (2010), T. Rabin, Ed., vol. 6223 of *LNCS*, Springer, pp. 155–172.

[19] Gentry, C., Peikert, C., and Vaikuntanathan, V. Trapdoors for hard lattices and new cryptographic constructions. In *STOC* (2008), C. Dwork, Ed., ACM, pp. 197–206.

[20] Golle, P., Jakobsson, M., Juels, A., and Syverson, P. F. Universal re-encryption for mixnets. In *CT-RSA* (2004), T. Okamoto, Ed., vol. 2964 of *LNCS*, Springer, pp. 163–178.

[21] Green, M., and Ateniese, G. Identity-based proxy re-encryption. In *ACNS* (2007), J. Katz and M. Yung, Eds., vol. 4521 of *LNCS*, Springer, pp. 288–306.

[22] Hada, S. Secure obfuscation for encrypted signatures. In *Eurocrypt* (2010), H. Gilbert, Ed., vol. 6110 of *LNCS*, Springer, pp. 92–112.

[23] Hemenway, B., Libert, B., Ostrovsky, R., and Vergnaud, D. Lossy encryption: Constructions from general assumptions and efficient selective opening chosen ciphertext security. In *Asiacrypt* (2011), D. H. Lee and X. Wang, Eds., vol. 7073 of *LNCS*, Springer, pp. 70–88.

[24] Hofheinz, D., Malone-Lee, J., and Stam, M. Obfuscation for cryptographic purposes. In *TCC* (2007), S. P. Vadhan, Ed., vol. 4392 of *LNCS*, Springer, pp. 214–232.

[25] Hohenberger, S., Rothblum, G. N., Shelat, A., and Vaikuntanathan, V. Securely obfuscating re-encryption. In *TCC* (2007), vol. 4392 of *LNCS*, Springer, pp. 233–252.

[26] Lyubashevsky, V., Peikert, C., and Regev, O. On ideal lattices and learning with errors over rings. In *Eurocrypt* (2010), H. Gilbert, Ed., vol. 6110 of *LNCS*, Springer, pp. 1–23.

[27] Regev, O. On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM 56*, 6 (2009), Article 34. A preliminary version appeared in *STOC 2005*, 2005.

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