On Protecting Cryptographic Keys Against Continual Leakage

Ali Juma Yevgeniy Vahlis*

University of Toronto
{ajuma,evahlis}@cs.toronto.edu

April 13, 2010

Abstract

Side-channel attacks have often proven to have a devastating effect on the security of cryptographic schemes. In this paper, we address the problem of storing cryptographic keys and computing on them in a manner that preserves security even when the adversary is able to obtain information leakage during the computation on the key.

Using the recently achieved fully homomorphic encryption, we show how to encapsulate a key and repeatedly evaluate arbitrary functions on it so that no adversary can gain any useful information from a large class of side-channel attacks. We work in the model of Micali and Reyzin, assuming that only the active part of memory during computation leaks information. Similarly to previous works, our construction makes use of a single "leak-free" hardware token that samples from a globally-fixed distribution that does not depend on the key.

Our construction is the first general compiler to achieve resilience against polytime leakage functions without performing any leak-free computation on the underlying secret key. Furthermore, the amount of computation our construction must perform does not grow with the amount of leakage the adversary is able to obtain; instead, it suffices to make a stronger assumption about the security of the fully homomorphic encryption.

1 Introduction

Leakage resilient cryptographic constructions – constructions that remain secure even when internal state information leaks to the adversary – have received much recent interest. Traditionally, security models have treated such internal state information as perfectly hidden from the adversary. However, the development of various side-channel attacks has made it clear that this traditional view is inconsistent with physical reality. In a side-channel attack, an adversary obtains information about the internal state of a device by measuring such things as power consumption, computation time, and emitted radiation.

Cryptographic primitives with long term keys, such as encryption and signature schemes, are often targeted by such attacks. An adversary observing information leakage from computation on the key can potentially accumulate enough data over time to compromise the security of the scheme. Consequently, storing keys and computing on them in adversarial environments has been an important goal both in theory and practice. Indeed, many operating systems provide cryptographic facilities that allow programs to access keys only through designated functions, such as signing and encrypting. Smart cards provide a similar

*Supported by the Natural Sciences and Engineering Research Council of Canada (NSERC).

interface in hardware. In both cases, the goal is to limit any adversary to interacting with the scheme through designated channels. Nevertheless, information leakage through physical side-channels is often sufficient to overcome such barriers and break the scheme.

In this paper, we propose an approach for protecting cryptographic keys and computing on them repeatedly in a manner that preserves the secrecy of the key even when information about the state of the device continuously leaks to the adversary. Towards this goal, we define a new primitive called a *key proxy*, which encapsulates a key K and provides a structured way of evaluating arbitrary functions on K. This allows, for example, the conversion of any pseudorandom function, signature scheme, or public key encryption scheme into a leakage resilient variant of itself. Our construction withstands a bounded amount of leakage per invocation (where an invocation occurs each time a function is evaluated on K), but the total amount of leakage is unbounded. Previously, only stream ciphers, signature schemes, and identification scheme have been made resilient to an unbounded total amount of leakage.

For our construction, we make use of the recently achieved fully homomorphic encryption [12, 27], and an additional "leak-free" component. We describe two ways of instantiating this component, and in both cases the component samples from a globally fixed distribution that does not depend on K.

Leakage resilient cryptography. The problem of executing code in an adversarial environment has always been on the minds of cryptographers. Still, most cryptographic schemes are designed assuming that the hardware on which they will be implemented is a black box device, and information is accessible to the adversary only through external communication channels. Goldreich and Ostrovsky [15] consider the problem of protecting software from malicious users, and define the concept of an oblivious RAM – a CPU that is capable of evaluating encrypted programs using a constant amount of leak-free memory and an unbounded amount of memory that is fully visible to the adversary. The oblivious RAM is initialized with a secret key, which is used to decrypt encrypted instructions, execute them, and re-encrypt the output. The encrypted state of the program is stored in the clear. Oblivious RAMs provide the strong security guarantee that even if an adversary can keep track of the memory locations accessed by the computation, she is still unable to gain any additional information about the program over what would normally be revealed through black box access.

Since the work of Goldreich *et al*, the focus in leakage resilient cryptography has been steadily shifting towards allowing the adversary ever-growing freedom in observing the *computation* of cryptographic primitives. Ishai, Sahai, and Wagner [18] introduce "private circuits" – a generic compiler that transforms any circuit into one that is resilient to probing attacks. In a probing attack, the adversary selects a subset (of some fixed size) of the wires of the circuit and obtains the values of these wires. Goldwasser, Kalai, and Rothblum [17] define one-time programs – programs that come with small secure hardware tokens, and can be executed a bounded number of times without revealing anything but the output, even if the adversary observes the entire computation. The secure tokens are the hardware equivalent of oblivious transfer – each token stores two keys and reveals one of them upon request, while the second key is erased.

Micali and Reyzin [21] outline a framework for defining and analyzing cryptographic security against adversaries that perform side channel attacks. They introduce an axiom: only computation leaks information. That is, at any point during the execution of an algorithm, only the part of memory that is actively computed on may leak information. This allows for convenient modeling of leakage: an algorithm is described as a sequence of procedures and the set of variables that is accessed by the procedure. The adversary may then obtain leakage separately from the contents of each set of variables as they are accessed during the execution of the algorithm. The only-computation-leaks model (OCL) has since been used to obtain stream ciphers [9, 22] and signature schemes [10] that remain secure even if the adversary obtains leakage from the

active state each time the primitive is used, and the total amount of leakage is unbounded. We refer to such leakage as "continuous leakage" for the rest of the paper.

Faust *et al* [11] propose an alternative restriction on side-channel adversaries: restricting the computational power of the leakage function but allowing leakage on the entire state. Faust *et al* describe a circuit transformation that immunizes any circuit against leakage functions that can be described as AC^0 circuits[1]. The transformed circuit can leak information from the entire set of wires at each invocation, and makes use of a polynomial number of leak-free components that generate samples from a fixed distribution that does not depend on the computation of the circuit. We make use of a similar leak-free component, although the distribution generated by our component is significantly more complex than the one in [11] due to the fact that we must defend against leakage functions that are not restricted to circuits of small depth.

Very recently, specific leakage resilient cryptographic primitives have been constructed under even more general continuous leakage models. Dodis, Haralambiev, Lopez-Alt, and Wichs [7] have constructed several primitives, including signature schemes and authenticated key agreement protocols, that remain secure even if the entire state (and not just the active part) leaks information continuously, assuming that there is a secure (leak-free) update procedure that can be performed on the key. The public key of the scheme remains fixed throughout the lifetime of the system. Brakerski, Kalai, Katz, and Vaikuntanathan [3] construct a public key encryption scheme that allows continuous (length bounded) leakage on the entire state, and does not require a leak-free key update procedure. As in our work, both above works provide protection against leakage that can be described by arbitrary polynomial-time computable functions with sufficiently short output.

In addition to the recent work on cryptographic constructions that are resilient to continuous leakage, there has been significant progress on obtaining resilience to "memory attacks" – side channel attacks where the adversary obtains a bounded amount of information about the memory contents of the device throughout its lifetime. Perhaps due to the bounded nature of this type of leakage, constructions secure against memory attacks tend to be quite efficient and do not require the algorithm to maintain a state. Akavia, Goldwasser, and Vaikuntanathan [1] show that the public key encryption scheme of Regev [23] and the identity based encryption scheme of Gentry, Peikert, and Vaikuntanathan [13] remain secure as long as the adversary does not obtain more than $n/polylog(n)$ bits of information about the private key. Alwen, Dodis, and Wichs [2] construct identification schemes, signature schemes, and authenticated key agreement so that the primitive is resilient to an arbitrary but bounded amount of leakage. Naor and Segev [24] construct a public key encryption scheme based on any hash proof system [4]. Their scheme is quite efficient, and remains secure even if the adversary learns $n - o(n)$ bits of information about the private key. Katz and Vaikuntanathan [20] construct a signature scheme that tolerates a loss of up to $n - n^{\varepsilon}$ bits of information for every ε.

Finally, a separate line of research [8, 6] describes private and public key encryption schemes that remain secure even if the adversary obtains a sufficiently hard to invert function of the secret key.

On leak-free components. When constructing leakage resilient cryptographic primitives, one has to take care in the nature and amount of components that are assumed not to leak any information. It is preferable, but may not always be possible, to avoid such components altogether. For example, one can protect any functionality against leakage given an arbitrary number of leak-free gates that can decrypt a ciphertext, perform a logical operation on the plaintext, and re-encrypt the result. Such a component can be used to evaluate the circuit F on K gate by gate, keeping all intermediate values encrypted, and thereby rendering leakage useless. However, building such leak-free components may be as difficult as constructing a leak-free computer and forgetting all about side-channels. Consequently, the focus of research in this area has

[1] AC^0 circuits have constant depth and unbounded fan-in

always been to reduce the power and amount of computation that is assumed to be a-priori insulated from side-channel attacks.

Our construction uses a leak-free component that produces random encryptions of some fixed message (in our case – $\bar{0}$) under a given public key in the fully homomorphic encryption scheme. More specifically, our construction can be instantiated with one of the following leak-free components: an input-less randomized component that produces tuples of the form (pub, pri, C, C') where C and C' are encryptions of $\bar{0}$, or a randomized component that given *pub* produces two random encryptions of $\bar{0}$.

In both cases above, the computation performed by the component does not depend on K or the function F that is evaluated on it. Indeed, most side-channel attacks exploit the adversary's ability to feed inputs to the device and then collect measurements from the side-channels. Our components are not influenced by any adversarially chosen inputs, which rules out a large class of side-channel attacks. Perhaps more importantly, this allows for rigorous testing of the device in a controlled environment. To obtain an accurate simulation of the component's behavior in practice, the designer of the component simply needs to feed it a sequence of random bits.

Faust *et al* [11] use a similar type of component – one that generates strings from a fixed distribution. The distribution generated by their component is much simpler than ours. However, this simplicity comes with a price – the construction provides protection against leakage functions that can be described as AC^0 circuits. Such circuits cannot, for example, compute linear functions, which are very common in side-channel attacks. Furthermore, in contrast to previous general compilers that achieve leakage resilience, we use only one leak-free component, regardless of the size of the circuit that is evaluated on K or the amount of information leakage per invocation. Thus, our construction does not require the number of leak-free components to grow with the amount of leakage.

Our contributions. We study the problem of computing on a cryptographic key in an environment that leaks information each time a computation is performed. We show that in the OCL model with a single leak-free randomized token, a cryptographic key can be protected in a manner that allows repeated computation on it while making sure that the adversary gains no information from side-channel information leakage.

More precisely, we propose a tool which we call a *key proxy* – a stateful cryptographic primitive that is initialized once with a key K, and then given any circuit F computes $F(K)$. Any leakage obtained by an adversary from the computation of the key proxy can be computed given just F and $F(K)$. Using any *fully homomorphic encryption* (FHE) we construct a key proxy with the following properties:

Resilience to adaptive polynomial time leakage. During each invocation of the key proxy, we allow the adversary to adaptively select leakage functions that are modeled as arbitrary circuits with a sufficiently short output. The exact amount of round leakage that our construction can withstand depends on the level of security of the underlying FHE. Assuming the most basic security for the FHE (i.e. against polynomial time adversaries) permits security against $O(\log n)$ bits of leakage each time a function is evaluated on K. More generally, given a $2^{l(n)}$-secure FHE, our construction can withstand roughly $l(n)$ bits of leakage per invocation.

Independent complexity. The starting point of leakage resilient cryptography is that *computation leaks information*. It does not require a large leap of faith to suspect that *more* computation leaks more information. In fact, to the best of our knowledge, this is indeed the case for many side-channel attacks in practice. The amount of computation performed by our key proxy construction does not depend on the amount of leakage that the adversary is assumed to obtain per invocation. Instead, to get resilience to larger amounts of leakage, a stronger assumption about the security of the underlying fully homomorphic encryption is used. This allows us to avoid a circular dependency where, in order to obtain resilience to larger amounts of

leakage, one must build a more complex device, which in turn leaks more information.

One-time programs with efficient refresh. The one-time programs of [17] can be implemented without leak-free one-time memory tokens by storing the contents of the tokens in memory, and then accessing only the needed values during computation. The one-time programs can then be refreshed occasionally in a secure environment to allow continuous use. Currently, the refresh procedure performs as much computation as the evaluation of the program that it protects. If one is willing to trade resilience against complete exposure of the active memory (achieved by [17]) for resilience length bounded leakage then by pre-computing the outputs of the leak-free tokens in our construction and storing them in memory we obtain one-time programs with an update procedure of fixed complexity that does not depend on the protected program.

Our approach. The underlying building block for our construction is fully homomorphic encryption. An FHE is a public key encryption scheme that allows computation on encrypted data. That is, given a ciphertext with corresponding plaintext M, the public key, and a circuit F, there is an efficient algorithm that computes an encryption of $F(M)$.

For our construction, we partition the state of the key proxy into two parts, A and B (or equivalently two devices). Given a key K, the key proxy is initialized as follows. An FHE key pair $(\mathit{pri}, \mathit{pub})$ is generated and is stored in memory A. Then, a random encryption C of K under *pub* is computed and is stored in memory B. To evaluate a function F (described as a circuit) on K, the following actions are performed. First, a new pair of keys $(\mathit{pri}', \mathit{pub}')$ is generated and stored in memory A, and an encryption $C_{\mathit{pri}} = \mathsf{Enc}_{\mathit{pub}'}(\mathit{pri})$ of the old private key is written to a public channel. Then, computing on memory B and the public channel, the following two ciphertexts are generated homomorphically from C and C_{pri}: an encryption C_{res} of $F(K)$ and a fresh encryption C_{key} of K. Note that both C_{res} and C_{key} are encryptions under the new public key pub'. The ciphertext C_{res} is then sent back to memory A where it is decrypted, and $F(K)$ is returned as the output of the program. This basic approach is described in Figure 1.

It is clear that without leakage, the above construction is secure. Of course, the main difficulty is showing that leakage does not provide the adversary with any useful information. Below we provide an informal description of two main technical issues that arise.

Leakage on private keys. It is easy to see that without refreshing the encryption C of K, a leakage adversary will eventually learn all of K by gradually leaking all of C and *pri* and then simply decrypting. Therefore, it is clear that an update procedure is necessary. The algorithm described in Figure 1 performs such an update: After each invocation, memory A contains a freshly generated private key and memory B contains an encryption of K under the corresponding public key. However, we cannot directly claim that this refreshing procedure provides the necessary level of security. The main difficulty stems from the fact that the adversary obtains leakage on the private key in memory A both before and after she obtains leakage on the encryption C of K under the corresponding public key. In particular, if the adversary could obtain the entire ciphertext C, she would be able to hardcode it into the second leakage function that is applied to the private key. The leakage function would then decrypt C and leak bits of information about K.

This requires us to make use of the fact that the adversary obtains only a bounded amount of leakage on the ciphertext C, and never sees it completely. We argue that any leakage function that provides enough information about the ciphertext in order to later learn something about the plaintext given the private key, essentially acts as a distinguisher and can be used to break the semantic security of the FHE.

Memory A		Memory B
Contents of memory: pri_i		Contents of memory: $C = \mathsf{Enc}_{pub_i}(K)$, Input: circuit F
Compute $(pri_{i+1}, pub_{i+1}) = \mathsf{KeyGen}(1^n)$ Encrypt $C_{pri} = \mathsf{Enc}_{pub_{i+1}}(pri_i)$ Set memory to pri_{i+1}	$\xrightarrow{pub_{i+1}, C_{pri}}$	
		Homomorphically compute using C, C_{pri}: $C_{\mathsf{res}} = \mathsf{Enc}_{pub_{i+1}}(F(K))$ and $C_{\mathsf{key}} = \mathsf{Enc}_{pub_{i+1}}(K)$ Set memory to C_{key}
	$\xleftarrow{C_{\mathsf{res}}}$	
Compute $Y = \mathsf{Dec}_{pri_{i+1}}(C_{\mathsf{res}})$ Return Y		

Figure 1: Informal description of construction

History carrying FHE. Fully homomorphic encryption on its own does not guarantee exactly the security properties that we need. The main issue is that ciphertexts produced by homomorphic encryption may carry information about the homomorphic computation that was performed to obtain them. For instance, it is possible that the ciphertext C_{res} is actually first decrypted to a string of the form $(F(K), K)$ and then the decryption algorithm ignores the second element in the pair. In this case, the adversarial leakage function is clearly not forced to follow the honest decryption algorithm and can make use of the intermediate values of the decryption process to leak information about K. Similarly, the ciphertext C_{key} may contain information about the function F that was evaluated on K. For some applications, such as encryption where F encodes in plain text the message to be encrypted, this is undesirable since the adversary may use future leakage functions to gain information about the message.

Fortunately, the homomorphic encryption schemes of Gentry [12] and of van Dijk *et al* [27] have the following additional property: given any encryption C of a message M and a random encryption C' of M', the ciphertext $C+C'$, where the addition is performed over the appropriate group of ciphertexts, is a random encryption of $M+M'$. Consequently, to address the issue described above, we randomize both C_{res} and C_{key} by adding random encryptions of zero to both ciphertexts. In order to make use of the property described above the encryptions of zero need to be generated without leakage; otherwise, the leaked information maintains a correlation between the randomized ciphertext and the history of the computation that was used to produce the original ciphertext.

In our construction, the encryptions of zero can either be generated in a leak-free manner together with the private and public key and written to the appropriate locations in memory (*pri* to memory A, and *pub* and the encryptions of zero to memory B), or sampled separately given the public key and written to memory B. The main advantage of the first solution is that the leak-free component that generates the tuple $(pri, pub, \mathsf{Enc}_{pub}(0), \mathsf{Enc}_{pub}(0))$ has no inputs, and in fact can be implemented in practice as a token that keeps outputting stored tuples that were computed in advance. The second solution is more useful in a setting where the key proxy is implemented on two separate devices that are connected by a public channel,

such as the scenario where the key proxy is used to defend against cold boot attacks.

We note that in the FHE schemes of [12] and [27], C' has to be generated in a special way in order to have enough noise to annihilate any dependence between $C + C'$ and the computation history of C. For simplicity of exposition we ignore this distinction, and instead remark that the randomization procedures of both FHE schemes satisfy the properties needed for our construction.

Function privacy in key proxies. In the above description of key proxies, we require that the leakage obtained by the adversary can be simulated given just F and $F(K)$. However, in some applications, such as private key encryption, the function F itself also needs to be hidden. In the case of encryption, F contains the message M, so an adversary can break semantic security simply by leaking information about F, ignoring K completely. This raises a subtle modeling issue: the message M must exist somewhere as plaintext, and if the adversary obtains leakage on that computation, she will trivially break semantic security. Therefore, irrespective of the definition of leakage resilient key proxies, semantic security cannot be achieved when every invocation of every algorithm leaks information.

There are several ways in which this issue can be addressed. One solution is to weaken the definition of semantic security by requiring that the plaintexts have high pseudo-entropy[2] given the leakage obtained by the adversary. We avoid this approach both because it leads to complex definitions, and because it does not seem to have a clear advantage over the following much cleaner solution. Instead, we allow the adversary to obtain leakage both before and after the challenge ciphertext is generated, but not on the computation of the challenge ciphertext itself. This essentially means that while leakage can compromise individual encryptions, the long-term key remains safe. Under this restriction, our definition of key proxies provides the needed level of security. This approach is consistent with previous definitions of leakage resilient semantic security (see e.g. [9, 24, 8, 6]), and allows us to avoid additional complexity in our definition. This is desirable especially given the fact that for some applications of key proxies, such as signature schemes, function privacy is not necessary.

We mention briefly that another option is to define a leakage model for private key encryption which allows the encryption algorithm to perform some leak-free pre-processing that is independent of the key. Then, the encryptor can generate an encrypted version of the circuit F, which can be safely given to the adversary without compromising security.

Organization. In Section 3, we describe the computational and leakage models that we use, and define a leakage resilient key proxy. In Section 4, we provide our main construction, and analyze its security. In Section 5, we describe several variants of our model and construction, and provide several applications of leakage resilient key proxies.

2 Preliminaries

Notation. We write PPT to denote Probabilistic Polynomial Time. When we wish to fix the random bits of a PPT algorithm M to a particular value, we write $M(x; r)$ to denote running M on input x and randomness r. We write $time_n(M)$ to denote the running time of algorithm M on security parameter n. We use $x \in_R S$ to denote the fact that x is sampled according to a distribution S. Similarly, when describing an algorithm we may write $x \leftarrow_R S$ to denote the action of sampling an element from S and storing it in a variable x.

[2] A distribution has pseudo-entropy $\geq k$ if it is computationally indistinguishable from some distribution with min-entropy $\geq k$.

It is common in cryptography to describe probabilistic experiments that test the ability of an adversary to break a primitive. Given such an experiment Exp, and an adversary A, we write $A \leftrightarrows \mathsf{Exp}$ to denote the random variable representing outcome of Exp when run with the adversary A.

2.1 Fully Homomorphic Encryption

The main tool in our construction is a fully homomorphic public key encryption (FHE) system. Intuitively, such a system has the usual semantic security properties of a public key encryption (PKE) scheme, but in addition, can perform arbitrary computation on encrypted data. The outcome of this computation is of course also encrypted. The first construction of FHE was given by Gentry in [12], and is based on ideal lattices. Recently another construction was proposed by van Dijk *et al* [27].

We do not go into the details of the FHE constructions, but rather present the result with respect to an arbitrary FHE with an additional randomization property, which is satisfied by both constructions.

Definition 2.1. Let $\mathsf{FHE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{EncEval}, \mathsf{Add}, \mathsf{Subtract})$ be a tuple of PPT algorithms, and let $l : \mathbb{N} \to \mathbb{N}$. We say that HPKE is an $l(n)$-*secure fully homomorphic public key encryption scheme* if the following conditions hold:

1. The triple $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ is a public key encryption scheme. We assume without loss of generality that the private key is always the random bits of KeyGen.
2. The algorithm $\mathsf{EncEval}(\mathit{pub}, \boldsymbol{C}, F)$, where *pub* is a public key, $\boldsymbol{C} = (C_1, \ldots, C_n)$ is a vector of ciphertexts with plaintexts $(m_1, \ldots, m_n)$, and F is a circuit on n inputs, outputs a string C' which is a valid encryption of $F(m_1, \ldots, m_n)$.
3. The algorithms Add and $\mathsf{Subtract}$ have the following properties:
   (a) For all *pri*, for $\mathit{pub} = \mathsf{KeyGen}(\mathit{pri})$, for all messages M_1 and M_2, for a random encryption C_1 of M_1 under *pub* and for every encryption C_2 of M_2 under *pub*, $\mathsf{Add}(\mathit{pub}, C_1, C_2)$ is distributed identically to $\mathsf{Enc}_{\mathit{pub}}(M_1 + M_2)$, and $\mathsf{Subtract}(\mathit{pub}, C_1, C_2)$ is distributed identically to $\mathsf{Enc}_{\mathit{pub}}(M_1 - M_2)$.
   (b) For all ciphertexts C_1 and C_2, $\mathsf{Add}(\mathit{pub}, \mathsf{Subtract}(\mathit{pub}, C_2, C_1), C_1) = C_2$. That is, subtracting a ciphertext is the inverse of adding it.
4. For every probabilistic adversary A running in time at most $l(n)$, the advantage of A in breaking the semantic security of FHE is at most $1/l(n)$.

Remark 2.2. The algorithms Add and $\mathsf{Subtract}$ may be implemented as addition and subtraction over the space of ciphertexts, though we do not require this. In some fully homomorphic encryption schemes, Add and $\mathsf{Subtract}$ may not achieve the exact requirement of step 3 above. Specifically, Add and $\mathsf{Subtract}$ may produce an encryption that cannot be computed on homomorphically using $\mathsf{EncEval}$. We note that this is not a problem for our construction since we only use $\mathsf{EncEval}$ on encryptions of *pri*, which are ephemeral and never the output of Add or $\mathsf{Subtract}$. We avoid formalizing this issue to improve exposition.

3 Models and Definitions

In this section, we present the definition of a leakage resilient key proxy (LRKP). We start with a syntactic description of the primitive, and then describe the security experiment and the leakage model.

Stateful Algorithms. Due to the continuous nature of side-channel attacks, it is necessary for an LRKP to maintain a state in order to achieve security. We model stateful algorithms by considering algorithms with a special input and output structure. A stateful randomized algorithm takes as input a triple $(x; R, S)$ where x is the query to the algorithm, R is a random string, and S is a state (when R is clear from context we omit it, and denote the input by $(x; S)$). It then outputs (y, S_{new}) where y is the reply to the query, and S_{new} is the new state.

Definition 3.1. A *key proxy* is a pair $KP = (\mathsf{KPInit}, \mathsf{KPEval})$, where KPInit is an algorithm, and KPEval is a stateful algorithm. For fixed $c \in \mathbb{N}$ and for all $n \in \mathbb{N}$, $K \in \{0,1\}^{n^c}$, $\mathsf{KPInit}(1^n, K)$ outputs an initial state S. For every circuit $F : \{0,1\}^{|K|} \to \{0,1\}^n$, and random coins R, the stateful algorithm $\mathsf{KPEval}(1^n, F; R, S)$ outputs $F(K)$.

We now describe the security experiment of LRKPs. This experiment is parameterized by the leakage structure on a single invocation of the KPEval algorithm. However, for clarity we start with the description of the general experiment, and then provide details on the leakage that occurs at each invocation. We model the the leakage resilience of a key proxy by requiring the leaked information to be simulatable. That is, we require the existence of a simulator Sim that, given F and $F(K)$, can simulate the leakage and messages obtained by the adversary during the computation of $\mathsf{KPEval}(1^n, F; R, S)$. No efficient adversary should be able to tell whether she is getting actual leakage and messages, or interacting with a simulator. We now describe the real and ideal security experiments:

Let $KP = (\mathsf{KPInit}, \mathsf{KPEval})$ be a key proxy. Let A and Sim be PPT algorithms, $n \in \mathbb{N}$, and consider the following two experiments:

$\mathsf{ExpReal}$ **(Real Interaction).** The interaction of the adversary with the key proxy proceeds as follows:

1. A key K is chosen by the adversary, and $\mathsf{KPInit}(1^n, K)$ is used to generate an initial state S.
2. The adversary repeats the following steps an arbitrary number of times:
   (a) The adversary submits a circuit F, which is evaluated on K by KPEval. During the computation, the adversary acts as a single invocation leakage adversary (described below in Definition 3.4) for KPEval.
   (b) At the end of the computation of KPEval, the adversary is given $F(K)$.
3. After the adversary is done making queries, it outputs a bit b.

$\mathsf{ExpIdeal}$ **(Ideal Interaction).** The interaction of the adversary with simulated leakage proceeds as follows:

1. The adversary submits a key K, which is not revealed to the simulator.
2. The adversary then repeats the following steps an arbitrary number of times:
   (a) The adversary submits a circuit F, and Sim is given F and $F(K)$. The adversary then acts as a single invocation leakage adversary according to Definition 3.4, except that the leakage functions are submitted to the simulator, which returns simulated leakage values and messages.
   (b) Eventually the adversary stops submitting leakage functions, and is given $F(K)$.
3. After the adversary is done making queries, it outputs a bit b.

Definition 3.2. We say that KP is a *Leakage Resilient Key Proxy* if for every PPT A there exists a PPT S and a negligible function $neg(\cdot)$ such that

$$|\Pr[(A \leftrightarrows \mathsf{ExpReal}) = 1] - \Pr[(A \leftrightarrows \mathsf{ExpIdeal}) = 1]| \leq neg(n)$$

The above definition describes the security of an LRKP relative to some unspecified procedure which allows the adversary to obtain leakage during each invocation of KPEval. The exact procedure for a single-invocation leakage depends on the leakage model and on the structure of the implementation of KPEval. Below we formalize the structure of our solution, and describe the leakage obtained by the adversary during a single invocation of KPEval.

Our construction of KPEval is described as a protocol between two parties EvalA and EvalB that leak information separately, and where the messages between EvalA and EvalB are public. In this format, our construction requires two flows between the parties: one from EvalA to EvalB and one from EvalB to EvalA. The following definition formalizes this structure.

Definition 3.3. A 2-round *split state key proxy* is a key proxy $KP = (\mathsf{KPInit}, \mathsf{KPEval})$ such that the state S is represented as a pair $S = (\mathsf{MemA}, \mathsf{MemB}) \in (\{0,1\}^{n^d})^2$ for some fixed $d \in \mathbb{N}$, and the algorithm KPEval is described as four algorithms $(\mathsf{LeakFree}, \mathsf{EvalA}_1, \mathsf{EvalB}, \mathsf{EvalA}_2)$, each running in time polynomial in n, where

1. LeakFree is given randomness RandLF, and outputs strings OutLF_A and OutLF_B.
2. EvalA_1 takes as input MemA, OutLF_A, and randomness RandA, and outputs an updated state $\mathsf{MemA}' \in \{0,1\}^{n^d}$ and a message M_{AB} to EvalB.
3. EvalB takes as input MemB, randomness RandB, OutLF_B, the message M_{AB}, and a circuit $F : \{0,1\}^{|K|} \to \{0,1\}^n$ of arbitrary size. It then outputs an updated state $\mathsf{MemB}' \in \{0,1\}^{n^d}$ and a message M_{BA} to EvalA.
4. EvalA_2 takes as input MemA', the message M_{BA} and outputs an updated state MemA'' and the result $F(K)$.

The output of KPEval is $F(K)$, and the updated state is $(\mathsf{MemA}'', \mathsf{MemB}')$.

Recall that our construction requires a leak-free component. This leak-free component is modeled by algorithm LeakFree above. A crucial point here is that LeakFree receives only randomness as input, and, in particular, receives neither F nor the saved state $(\mathsf{MemA}, \mathsf{MemB})$ as inputs; therefore, regardless of the actual construction, the above definition prevents LeakFree from carrying out the evaluation of F on K, which would make the construction trivial.

We are now ready to describe the leakage structure on a single invocation of a 2-round split state key proxy. The leakage model we use, commonly known as "only computation leaks information" (OCL), lets the adversary obtain leakage only on the active part of memory during each computation.

Definition 3.4. Let $l : \mathbb{N} \to \mathbb{N}$ and let *KP* be a 2-round split state key proxy. A single invocation leakage adversary in the only-computation-leaks model chooses a circuit f_1, sees $f_1(\mathsf{MemA}, \mathsf{OutLF}_A, \mathsf{RandA})$ and M_{AB}, chooses circuit f_2, sees $f_2(\mathsf{MemB}, \mathsf{OutLF}_B, \mathsf{RandB})$ and M_{BA}, chooses a circuit f_3, and sees $f_3(\mathsf{MemA}')$. The adversary is l-bounded if for all n the range of f_1, f_2, f_3 is $\{0,1\}^{l(n)}$.

Note that in the above definition, the leakage functions can compute any internal values that appear during the computations of EvalA_1, EvalB, and EvalA_2. This means, for example, that it is unnecessary to explicitly provide M_{AB} to f_1 or M_{BA} to f_2.

History freeness. In Definition 3.2 we allow information about the functions F_i that are evaluated on K to leak to the adversary. In particular, it is possible that during some invocation j the adversary can obtain through leakage information about some previously queried function F_i. In the introduction we mentioned that leakage resilient variants of some applications, such as private key encryption, are defined to allow leakage both before and after the generation of the challenge ciphertext, but not on the challenge itself. However, if the state of LRKP keeps a history of some of the functions that were applied to K, then by leaking on it after the challenge was computed, the adversary may be able to break the semantic security of the encryption. We note that the above definition is sufficient as-is to obtain security in the presence of what we call "lunch-time leakage" attacks. That is, if the adversary obtains leakage only before the challenge ciphertext is generated, but not after, then leakage on the history of the computation does not help the adversary to break security.

To address the above issue, and allow full leakage in applications such as encryption, we introduce an additional information theoretic property that requires that the state of the LRKP is distributed identically after all sequences of functions that are evaluated on K. This property is satisfied by our construction, and prevents the above mentioned "history attack".

Definition 3.5. An LRKP $(\mathsf{KPInit}, \mathsf{KPEval})$ is called *history free* if for all $n \in \mathbb{N}$, $K \in \{0,1\}^{poly(n)}$, there exists a distribution D over the states of the LRKP such that for all $j \in \mathbb{N}$, all sequences of functions $F_1, \ldots, F_j : \{0,1\}^{|K|} \to \{0,1\}^n$, and all sequences of random tapes $R_0, \ldots, R_{j-1}$, the random variable $\{S_{j+1} | S_1, \ldots, S_j\}$ over R_j is distributed according to D, where $S_1 = \mathsf{KPInit}(1^n, K; R_0)$ and S_i is the updated state after $\mathsf{KPEval}(1^n, F_{i-1}; R_i, S_{i-1})$.

4 Leakage Resilient Key Proxies From Homomorphic Encryption

Given a fully homomorphic public key encryption scheme $\mathsf{FHE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{EncEval}, \mathsf{Add}, \mathsf{Subtract})$ we construct a leakage-resilient 2-round split state key proxy $\mathsf{LRKP} = (\mathsf{KPInit}, \mathsf{KPEval})$.

$\mathsf{KPInit}(1^n, K)$**:** The algorithm $\mathsf{KPInit}(1^n, K)$ first runs $\mathsf{KeyGen}(1^n)$ to obtain a public-private key pair $(\mathit{pub}_1, \mathit{pri}_1)$ for the FHE. It then generates a ciphertext $C_{\mathsf{key}} = \mathsf{Enc}_{\mathit{pub}_1}(K)$ and assigns $\mathsf{MemA} \leftarrow \mathit{pri}_1$ and $\mathsf{MemB} \leftarrow C_{\mathsf{key}}$. The output is an initial state that consists of two parts $(\mathsf{MemA}, \mathsf{MemB})$.

$\mathsf{KPEval}(1^n, F; (\mathsf{MemA}, \mathsf{MemB}))$**:** The algorithm KPEval consists of four subroutines: $\langle \mathsf{LeakFree}, \mathsf{EvalA}_1, \mathsf{EvalB}, \mathsf{EvalA}_2 \rangle$ that are used as follows: on input circuit F first generate $(\mathsf{OutLF}_A, \mathsf{OutLF}_B) \leftarrow_{\mathrm{R}} \mathsf{LeakFree}(1^n)$. Then, follow the protocol described in Figure 2 by computing

$$(M_{AB}, \mathsf{MemA}') \leftarrow_{\mathrm{R}} \mathsf{EvalA}_1(\mathsf{MemA}, \mathsf{OutLF}_A);$$
$$(M_{BA}, \mathsf{MemB}') \leftarrow_{\mathrm{R}} \mathsf{EvalB}(\mathsf{MemB}, \mathsf{OutLF}_B, M_{AB});$$
$$Y \leftarrow \mathsf{EvalA}_2(\mathsf{MemA}', M_{BA})$$

The final state after one evaluation of KPEval is $(\mathsf{MemA}', \mathsf{MemB}')$, and the output is Y.

We now describe the subroutines $\langle \mathsf{LeakFree}, \mathsf{EvalA}_1, \mathsf{EvalB}, \mathsf{EvalA}_2 \rangle$ of KPEval:

$\mathsf{LeakFree}(1^n)$**:** Parse randomness as $(\mathit{pri}_{i+1}, r^i_{LF1}, r^i_{LF2})$, and compute

$$\mathit{pub}_{i+1} = \mathsf{KeyGen}(1^n, \mathit{pri}_{i+1});\ C_{\mathsf{R0},i} = \mathsf{Enc}_{\mathit{pub}_{i+1}}(\bar{0}; r^i_{LF1});\ C_{\mathsf{R1},i} = \mathsf{Enc}_{\mathit{pub}_{i+1}}(\bar{0}; r^i_{LF2})$$
$$\mathsf{OutLF}_B \leftarrow (\mathit{pub}_{i+1}, C_{\mathsf{R0},i}, C_{\mathsf{R1},i});\ \mathsf{OutLF}_A \leftarrow (\mathit{pri}_{i+1}, \mathit{pub}_{i+1})$$

and output $(\mathsf{OutLF}_A, \mathsf{OutLF}_B)$.

The subroutines EvalA_1, EvalB, and EvalA_2 are described in Figure 2 as a two round two party protocol where EvalA_1 and EvalA_2 specify the actions of party A and EvalB specifies the actions of party B. In the definition of EvalB we use subroutines Evaluate and Refresh that are defined as follows:

$\mathsf{Evaluate}(F, C, pri)$: Compute and output $F(\mathsf{Dec}_{pri}(C))$
$\mathsf{Refresh}(C, pri)$: Compute and output $\mathsf{Dec}_{pri}(C)$

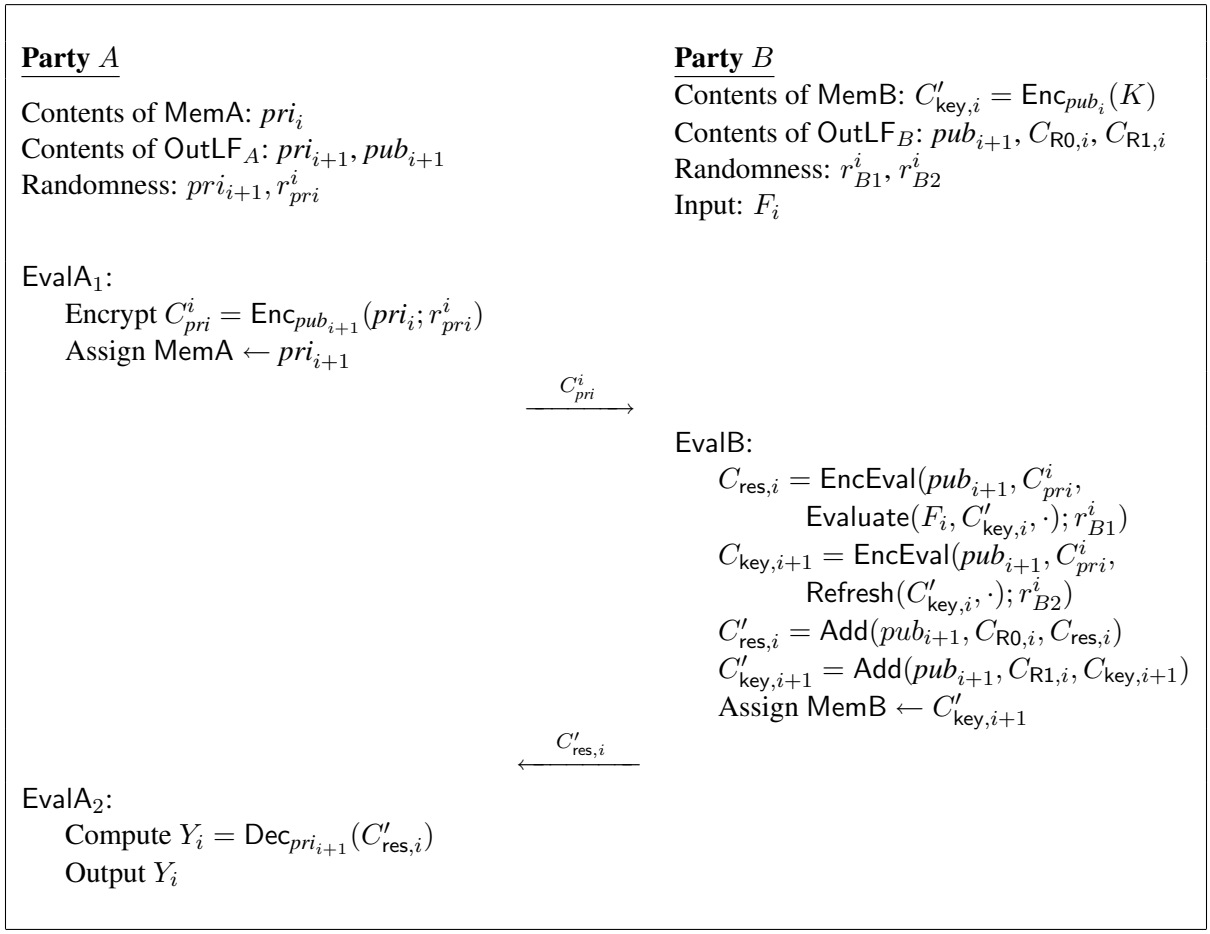

Figure 2: The algorithm KPEval in its ith invocation.

The correctness of this construction follows in a straightforward manner from the correctness of the underlying FHE. We also note that our construction is *history free* according to Definition 3.5. This is due to the fact that the values assigned to MemA and MemB at the end of KPEval are independent from the function F. In particular, MemA is simply a random private key, and MemB contains an encryption of K which was obtained by a homomorphic evaluation of Refresh on the previous contents of MemB and an encryption of the previous private key, neither of which depends on F.

The bulk of the analysis is in showing that our construction is in fact leakage resilient according to Definition 3.2, where during each invocation the leakage structure on the computation of KPEval is given in

Definition 3.4. We now state our main theorem. Due to space limitations, we defer the proof to Appendix B.

Theorem 1. *Let* LRKP *be the 2-round split state key proxy described in the above construction, and let* $l : \mathbb{N} \to \mathbb{N}$. *If* FHE *is a* $2^{O(l(n))}$*-secure fully homomorphic encryption then* LRKP *is leakage resilient against all* $O(l(n))$*-bounded adversaries in the OCL model.*

5 Extensions and Applications

Below we describe several variants of our scheme that provide various tradeoffs in security and functionality. In Appendix E, we describe some of the definitional issues that arise when dealing with semantic security in a setting with leakage, and then define and construct a leakage resilient private key encryption scheme, providing a complete proof of security. In Appendix F, we describe informally how one may construct a leakage resilient CCA-PKE. The construction of the CCA-PKE is quite simple, and the proof of security follows the same principles as the proof for the private key scheme.

An alternative leak-free component. In the construction depicted in Figure 2, the leak-free component is randomized but input-less, and produces tuples of the form (pub, pri, C, C'), where C and C' are encryptions of 0 under *pub*. As an alternative, we can use a leak-free component that is randomized, takes *pub* as input, and generates two random encryptions of 0. We note that the component's input is independent of F_i and K. To modify the construction to use such a component, we let party A generate *pub* and *pri* itself; then, party A includes *pub* in its message to B, who gives *pub* to the leak-free component. The leak-free component then produces two encryptions of 0 as before, giving these to B. It is straightforward to modify the proof of Theorem 1 to handle this alternative leak-free component – it suffices to to make some small modifications to the proof of Claim B.3, and the parameters in the theorem remain unchanged.

Resilience against complete compromise. Using the above alternative leak-free component allows us to view the scheme as a protocol between two devices that communicate over a public channel. In this case, the key remains hidden even if the memory contents of one the devices are leaked completely (for example, in a cold boot attack), provided that the compromise is detected and no further computation is performed using the counterpart device. The argument is a straightforward adaptation of the ideas in Claim B.5 and Claim B.6.

One-time programs. Our construction can be modified to work without any leak-free components by pre-computing a large number of tuples of the form (pri, pub, C, C') where C and C' are encryptions of 0 under *pub*, and storing the tuples in memory. Then, at each invocation, one such tuple is used (first *pri* and *pub* are used by EvalA_1, and then C, C' are used by EvalB). Assuming that only computation leaks information, the remaining tuples remain hidden until they are accessed. Therefore, security is obtained following essentially the same argument as the proof of Theorem 1. The number of invocations in this case is bounded by the number of pre-computed tuples. This approach provides a weaker security guarantee than the one time programs of [17] (i.e. only security against leakage), but has the advantage that the pre-computing phase is independent from the functionality that is being protected.

Acknowledgements. We thank Charles Rackoff for many hours of discussion.

References

[1] Adi Akavia, Shafi Goldwasser, and Vinod Vaikuntanathan. Simultaneous hardcore bits and cryptography against memory attacks. In *TCC '09: Proceedings of the 6th Theory of Cryptography Conference*, pages 474–495, Berlin, Heidelberg, 2009. Springer.

[2] Joel Alwen, Yevgeniy Dodis, and Daniel Wichs. Leakage resilient public-key cryptography in the bounded retrieval model. In *Advances in Cryptology — CRYPTO 2009*, pages 36–54, Berlin, Heidelberg, 2009. Springer.

[3] Zvika Brakerski, Yael Kalai, Jonathan Katz, and Vinod Vaikuntanathan. Cryptography resilient to continual memory leakage. Manuscript, 2010.

[4] Ronald Cramer and Victor Shoup. Universal hash proofs and a paradigm for adaptive chosen ciphertext secure public-key encryption. In *Advances in Cryptology – EUROCRYPT 2002*, pages 45–64. Springer, 2002.

[5] Yevgeniy Dodis, Shafi Goldwasser, Yael Kalai, Chris Peikert, and Vinod Vaikuntanathan. Public-key encryption schemes with auxiliary inputs. 2009.

[6] Yevgeniy Dodis, Shafi Goldwasser, Yael Tauman Kalai, Chris Peikert, and Vinod Vaikuntanathan. Public-key encryption schemes with auxiliary inputs. In *TCC*, pages 361–381, 2010.

[7] Yevgeniy Dodis, Kristiyan Haralambiev, Adriana Lopez-Alt, and Daniel Wichs. Cryptography against continuous memory attacks. Cryptology ePrint Archive, Report 2010/196, 2010. http://eprint.iacr.org/.

[8] Yevgeniy Dodis, Yael Tauman Kalai, and Shachar Lovett. On cryptography with auxiliary input. In *STOC '09: Proceedings of the 41st annual ACM symposium on Theory of computing*, pages 621–630, New York, NY, USA, 2009. ACM.

[9] Stefan Dziembowski and Krzysztof Pietrzak. Leakage-resilient cryptography. In *FOCS '08: Proceedings of the Annual IEEE Symposium on Foundations of Computer Science*, pages 293–302, Washington, DC, USA, 2008. IEEE Computer Society.

[10] Sebastian Faust, Eike Kiltz, Krzysztof Pietrzak, and Guy N. Rothblum. Leakage-resilient signatures. In *TCC*, pages 343–360, 2010.

[11] Sebastian Faust, Tal Rabin, Leonid Reyzin, Eran Tromer, and Vinod Vaikuntanathan. Protecting against computationally bounded and noisy leakage. In *EUROCRYPT*, 2010 (to appear).

[12] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *STOC '09: Proceedings of the 41st annual ACM symposium on Theory of computing*, pages 169–178, New York, NY, USA, 2009. ACM.

[13] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *STOC '08: Proceedings of the 40th annual ACM symposium on Theory of computing*, pages 197–206, New York, NY, USA, 2008. ACM.

[14] Oded Goldreich. *Foundations of Cryptography: Volume 2: Basic Applications*. Cambridge University Press, 2004.

[15] Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious rams. *J. ACM*, 43(3):431–473, 1996.

[16] Shafi Goldwasser, Yael Kalai, Chris Peikert, and Vinod Vaikuntanathan. Robustness of the learning with errors assumption. In *Proceedings of the 1st Innovations in Computer Science conference (ICS 2010)*, 2010.

[17] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. One-time programs. In *Advances in Cryptology — CRYPTO 2008*, pages 39–56, Berlin, Heidelberg, 2008. Springer.

[18] Yuval Ishai, Amit Sahai, and David Wagner. Private circuits: Securing hardware against probing attacks. In *Advances in Cryptology — CRYPTO 2003*, pages 463–481, Berlin, Heidelberg, 2003. Springer.

[19] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography (Chapman & Hall/Crc Cryptography and Network Security Series)*. Chapman & Hall/CRC, 2007.

[20] Jonathan Katz and Vinod Vaikuntanathan. Signature schemes with bounded leakage resilience. In Mitsuru Matsui, editor, *Advances in Cryptology - ASIACRYPT 2009. Proceedings*, volume 5912 of *Lecture Notes in Computer Science*, pages 703–720. Springer, 2009.

[21] Silvio Micali and Leonid Reyzin. Physically observable cryptography. In *TCC '04: Proceedings of the 1st Theory of Cryptography Conference*, pages 278–296, Berlin, Heidelberg, 2004. Springer.

[22] Krzysztof Pietrzak. A leakage-resilient mode of operation. In *Advances in Cryptology – EUROCRYPT 2009*, pages 462–482, Berlin, Heidelberg, 2009. Springer-Verlag.

[23] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In *STOC '05: Proceedings of the 37th annual ACM symposium on Theory of computing*, pages 84–93, New York, NY, USA, 2005. ACM.

[24] Gil Segev and Moni Naor. Public-key cryptosystems resilient to key leakage. In *Advances in Cryptology — CRYPTO 2009*, pages 18–35, Berlin, Heidelberg, 2009. Springer.

[25] François-Xavier Standaert, Tal Malkin, and Moti Yung. A unified framework for the analysis of side-channel key recovery attacks. In *Advances in Cryptology – EUROCRYPT 2009*, pages 443–461, Berlin, Heidelberg, 2009. Springer.

[26] Francois-Xavier Standaert, Olivier Pereira, Yu Yu, Jean-Jacques Quisquater, Moti Yung, and Elisabeth Oswald. Leakage resilient cryptography in practice. Cryptology ePrint Archive, Report 2009/341, 2009. http://eprint.iacr.org/.

[27] Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. In *EUROCRYPT*, 2010 (to appear).

A Probabilistic Lemmas

Lemma A.1 (Chernoff Bound)**.** *Let $0 \leq p \leq 1$, and let $X_1, \ldots, X_n$ be independent 0-1 random variables, so that $\Pr[X_i = 1] = p$ for each i. Then, for all $\varepsilon > 0$, we have*

$$\Pr\left[\left|\frac{\sum_{i=1}^{n} X_i}{n} - p\right| > \varepsilon\right] < 2 \cdot e^{-2n\varepsilon^2}$$

B Proof of Theorem 1

The theorem follows as a corollary from the following lemma:

Lemma 1. *Consider the experiment $ExpReal$ instantiated using scheme $LRKP$. Then, for every function $\varepsilon(n) > 0$, every $d > 0$, every $l : \mathbb{N} \to \mathbb{N}$, and every l-bounded PPT adversary A that makes n^d queries and gets leakage according to the only-computation-leaks model, there exists a PPT simulator S such that if*

$$|\Pr[(A \leftrightarrows ExpReal) = 1] - \Pr[(A \leftrightarrows S) = 1]| \geq \varepsilon(n)$$

for infinitely many n, then for every function $\varepsilon'(n) > 0$ there exists an adversary A' that runs in time

$$\frac{2^{3l(n)+5}}{\varepsilon'(n)^2}\left(3l(n) + 4 + \log\frac{1}{\varepsilon'(n)}\right)\left(4 \cdot time_n\left(\mathsf{Enc}\right) + time_n\left(LRKP \leftrightarrow A\right)\right) + time_n(\mathsf{KeyGen})$$

and breaks the semantic security of $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *with advantage $\frac{\varepsilon(n)}{3 \cdot 2^{2l(n)}(n^d+1)} - 2\varepsilon'(n)$ for infinitely many n. Specifically, S runs in time $time_n(LRKP \leftrightarrow A)$.*

Let A be a PPT adversary according to Definition 3.2 that makes n^d function evaluation queries and gets leakage according to the only-computation-leaks model described in Definition 3.4. We describe a sequence of experiments where the initial experiment Hyb_0 is the real security experiment $\mathsf{ExpReal}$, and the final experiment Hyb_3 is such that the leakage obtained by the adversary for each KPEval query F can be simulated given only $(F, F(K))$. We then show that A cannot distinguish between interacting with Hyb_0 and Hyb_3.

We first introduce some notation. We denote by MemA_i and MemB_i the saved state of party A and party B before round i. We denote by F_i the ith function that the adversary submits to be evaluated on K, we denote by f_i^j the jth leakage query during the computation of KPEval on F_i, and we denote by λ_i^j the response to the leakage query f_i^j. For our construction, $j \in \{1, 2, 3\}$. Specifically, λ_i^1 is the initial leakage on EvalA_1 from $(pri_i, pub_{i+1}, pri_{i+1}, r_{pri}^i)$ in round i, λ_i^2 is the leakage on EvalB from $(C'_{\mathsf{key},i}, pub_{i+1}, C_{\mathsf{R0},i}, C_{\mathsf{R1},i}, r_{B1}^i, r_{B2}^i)$, and λ_i^3 is the final leakage on EvalA_2 from (pri_{i+1}). In addition to seeing leakage, the adversary also gets all communication between party A and party B. Specifically, after seeing λ_i^1 but before submitting f_i^2, the adversary is given C_{pri}^i, and after seeing λ_i^2 but before submitting f_i^3, the adversary is given $C'_{\mathsf{res},i}$.

B.1 Hybrid Experiment Structure

We now describe the sequence of hybrid experiments:

Experiment Hyb_0**.** Hyb_0 is the real security experiment $\mathsf{ExpReal}$.

Experiment Hyb_1**.** Experiment Hyb_1 is the same as Hyb_0, except that a dummy round is added at the beginning and at the end of the experiment. More precisely, before the first evaluation query of the adversary, the initialization algorithm KPInit is run, and then a single round of KPEval is performed with a dummy function (e.g. one that always outputs $\bar{0}$). Similarly, after the adversary makes the last evaluation query, another dummy round of KPEval is performed.

Note that $(\mathsf{MemA}_1, \mathsf{MemB}_1)$ are distributed identically in Hyb_0 and Hyb_1, and the additional dummy round n^d+1 has no effect on the view of the adversary. Thus, the above changes are purely conceptual.

We now describe a second hybrid experiment, where the changes are more substantial.

Experiment Hyb_2**.** In experiment Hyb_2 we remove the key K from all variables that are exposed to the adversary. In particular, MemB will now contain an encryption of $\bar{0}$ instead of K. This change, by itself, would corrupt the output of KPEval, which depends on the contents of MemB. We correct this error by changing the way we compute the ciphertext $C_{\mathsf{R0},i}$ so that when this ciphertext is added to $C_{\mathsf{res},i}$, the resulting ciphertext $C'_{\mathsf{res},i}$ contains the intended output $F_i(K)$. More formally: experiment Hyb_2 proceeds in the same way as Hyb_1 with the following changes.

1. During the initialization process, $C'_{\mathsf{key},1}$ is computed as $\mathsf{Enc}_{pub_1}(\bar{0})$.
2. In each round $0 < i \leq n^d$, $C_{\mathsf{R0},i}$ is computed as $\mathsf{Enc}_{pub_{i+1}}(F_i(K) - F_i(\bar{0}))$.

Observe that aside from in the initialization round, the only information about K that is needed to carry out Hyb_2 are the values $F_i(K)$ for each query F_i produced by the adversary. It is easy to see that, in fact, the initialization round can be modified so that K is not needed, without changing the distribution of the leakage values and communication seen by the adversary during the experiment. This modification is described by the following hybrid experiment:

Experiment Hyb_3**.** Experiment Hyb_3 proceeds in the same way as Hyb_2, except that dummy round $n^d + 1$ is omitted and the initialization process is done differently: dummy round 0 is omitted, and $C'_{\mathsf{key},1}$ is set directly to $\mathsf{Enc}_{pub_1}(\bar{0})$. The entire modified initialization is as follows:

1. Run KeyGen to obtain (pub_1, pri_1).
2. Compute $C'_{\mathsf{key},1} = \mathsf{Enc}_{pub_1}(\bar{0})$.
3. Set $\mathsf{MemA}_1 = pri_1$ and $\mathsf{MemB}_1 = C'_{\mathsf{key},1}$.

Note that $(\mathsf{MemA}_1, \mathsf{MemB}_1)$ are distributed identically in Hyb_2 and Hyb_3. Furthermore, omitting dummy round $n + 1$ has no effect on the view of the adversary. Thus, the above change is purely conceptual.

Our simulator S interacts with the adversary as in Hyb_3. Note that S runs in time at most $time_n(LRKP \leftrightarrow A)$. To show that the adversary is unable to distinguish between leakage produced according to Hyb_3, and therefore between simulated leakage and real leakage, we show that each pair of consecutive hybrid experiments is indistinguishable.

To facilitate the analysis, we denote by X_i the random variable corresponding to the output of the adversary in experiment Hyb_i. We have already mentioned the following two facts:

Fact B.1. $\Pr[X_0 = 1] = \Pr[X_1 = 1]$

Fact B.2. $\Pr[X_2 = 1] = \Pr[X_3 = 1]$

The crux of the proof is comparing experiments Hyb_1 and Hyb_2. For this purpose, we first define a sequence of intermediate hybrids that are between Hyb_1 and Hyb_2. For $0 \leq i \leq n^d + 1$, Hyb_{12}^i behaves the same as Hyb_1 up to round $i-1$, behaves the same as Hyb_2 from round $i+1$ onward, and behaves specially in round i. More specifically, for $0 \leq i \leq n^d$, Hyb_{12}^i is defined as follows.

Experiment Hyb_{12}^i.

1. For $0 \leq j \leq i-1$, round j proceeds the same as in Hyb_1.
2. Round i proceeds the same as Hyb_1, except that $C_{\mathsf{R1},i}$ is set to $\mathsf{Enc}_{pub_{i+1}}(K)$.
3. For $i+1 \leq j \leq n^d+1$, round j proceeds the same as in Hyb_2.

Note that the dummy round 0 that takes place during the initialization process proceeds identically to Hyb_2 if $i = 0$, and to Hyb_1 otherwise. Also, note that dummy round $n^d + 1$ always proceeds identically in both Hyb_1 and Hyb_2. Consequently, Hyb_1 is identical to $\mathsf{Hyb}_{12}^{n^d+1}$, and Hyb_2 is identical to Hyb_{12}^0.

We now show that if there exists an adversary A that distinguishes Hyb_{12}^0 and $\mathsf{Hyb}_{12}^{n^d+1}$, then there exists an adversary A' that succeeds in the following experiment Exp_1.

Experiment Exp_1. Say that on inputs of length n, the output of Enc has length n'.

1. The adversary submits two messages m_0, m_1 such that $m_0 \neq m_1$.
2. A pair of public and private keys are generated $(pub, pri) = \mathsf{KeyGen}(1^n)$, and *pub* is given to the adversary.
3. The adversary submits a leakage function $\mathsf{leak}_1 : \{0,1\}^n \to \{0,1\}^{l(n)}$, and sees $\mathsf{leak}_1(pri)$.
4. A random bit b is chosen, and an encryption $C = \mathsf{Enc}_{pub}(m_b)$ is computed.
5. The adversary submits a leakage function $\mathsf{leak}_2 : \{0,1\}^{n'} \to \{0,1\}^{l(n)}$, and sees $\mathsf{leak}_2(C)$.
6. The adversary submits a leakage function $\mathsf{leak}_3 : \{0,1\}^n \to \{0,1\}^{l(n)}$, and sees $\mathsf{leak}_3(pri)$.
7. A new pair of public and private keys are generated $(pub', pri') = \mathsf{KeyGen}(1^n)$, and a random string $r_{pri'}$ is chosen. The public key pub' is given to the adversary.
8. The adversary submits a leakage function $\mathsf{leak}_4 : \{0,1\}^{3n} \to \{0,1\}^{l(n)}$, and sees $\mathsf{leak}_4(pri, pri', r_{pri'})$.
9. The adversary sees $C' = \mathsf{Enc}_{pub'}(pri; r_{pri'})$.
10. The adversary submits a leakage function $\mathsf{leak}_5 : \{0,1\}^{n'} \to \{0,1\}^{l(n)}$, and sees $\mathsf{leak}_5(C)$.
11. The adversary sees pri, pri', and outputs a bit $\hat{b}$.

We say that an adversary A' succeeds with advantage $\varepsilon(n)$ in Exp_1 if $|\Pr[(A' \leftrightarrows Exp_1) = 1 | b = 1] - \Pr[(A' \leftrightarrows Exp_1) = 1 | b = 0]| \geq \varepsilon(n)$.

Claim B.3. *Let A be an adversary and define, for all n, $\varepsilon(n) = |\Pr[(A \leftrightarrows \mathsf{Hyb}_{12}^0) = 1] - \Pr[(A \leftrightarrows \mathsf{Hyb}_{12}^{n^d+1}) = 1]|$. Then there exists an adversary A' that, for all n, runs in time $4 \cdot time_n(\mathsf{Enc}) + time_n(LRKP \leftrightarrow A)$ and succeeds with advantage $\varepsilon(n)/(n^d+1)$ in Exp_1.*

Proof. We first summarize the construction of A'. A' randomly selects an i, $0 \leq i \leq n^d$, and then simulates A according to Hyb_1 up to round $i-1$. Then, A' submits the two messages $m_0 = K$ and $m_1 = \bar{0}$, and uses the leakage queries permitted by Exp_1 to answer the queries of A during the ith and $i+1$st rounds. During the simulation, *pub* plays the role of pub_{i+1}, *pub'* the role of pub_{i+2}, C the role of $C'_{\mathsf{key},i+1}$ and C' the role of $C^{i+1}_{pri} = \mathsf{Enc}_{pub_{i+2}}(pri_{i+1})$. A' uses C and the properties of Add to "work backwards" and obtain correctly distributed values for $C_{\mathsf{R1},i}$, $C_{\mathsf{R0},i+1}$, and $C_{\mathsf{R1},i+1}$. Then, from round $i+2$ onward, A' simulates A according to Hyb_2, and outputs whatever A outputs. By construction, we have that if C is an encryption of $\bar{0}$ then A' simulates A perfectly in Hyb^i_{12}, and if C is an encryption of K, A is simulated perfectly in Hyb^{i+1}_{12}. The details follow.

A' begins by randomly selecting i such that $0 \leq i \leq n^d$. We first handle the case $1 \leq i \leq n^d - 1$. Our adversary A' simulates A according to Hyb_1 up to round $i-1$ (note that Hyb^i_{12} and Hyb^{i+1}_{12} proceed identically up to that round). Then, A' submits the two messages $m_0 = K$ and $m_1 = \bar{0}$, and obtains a public key *pub*. A' starts simulating A in round i by obtaining the first leakage function f^1_i. A' then generates uniformly r^i_{pri}, and creates the following leakage function:

- $\mathsf{leak}_1(pri)$: Compute and return $f^1_i(pri_i, pub, pri, r^i_{pri})$.

A' submits the above leakage function in step 3, and obtains a string λ^1_i. A' also computes $C^i_{pri} = \mathsf{Enc}_{pub}(pri_i; r^i_{pri})$, and gives the pair (λ^1_i, C^i_{pri}) to A. A then outputs leakage functions f^2_i. A' generates an encryption $C'_{\mathsf{res},i} = \mathsf{Enc}_{pub}(F_i(K))$, and randomly selects r^i_{B1} and r^i_{B2}. Then A' constructs the following leakage function:

- $\mathsf{leak}_2(C)$:
  - Compute $C_{\mathsf{res},i} = \mathsf{EncEval}(pub, C^i_{pri}, \mathsf{Evaluate}(F_i, \mathsf{MemB}_i, \cdot); r^i_{B1})$.
  - Compute $C_{\mathsf{key},i+1} = EncEval(pub, C^i_{pri}, \mathsf{Refresh}(\mathsf{MemB}_i, \cdot); r^i_{B2})$.
  - Compute $C_{\mathsf{R0},i} = \mathsf{Subtract}(pub, C'_{\mathsf{res},i}, C_{\mathsf{res},i})$.
  - Compute $C_{\mathsf{R1},i} = \mathsf{Subtract}(pub, C, C_{\mathsf{key},i+1})$.
  - Compute and return $f^2_i(\mathsf{MemB}_i, pub, C_{\mathsf{R0}}, C_{\mathsf{R1}}, r^i_{B1}, r^i_{B2})$.

A' submits the leakage function in step 5, and obtains λ^2_i. A' gives $(\lambda^2_i, C'_{\mathsf{res},i})$ to A. A then outputs f^3_i. A' sets:

- $\mathsf{leak}_3(pri)$: Compute and return $f^3_i(pri)$.

A' is now given λ^3_i and pub'. Using λ^3_i, A' obtains the first leakage function f^1_{i+1} for round $i+1$, and sets:

- $\mathsf{leak}_4(pri, pri', r^{i+1}_{pri})$: Compute and return $f^1_{i+1}(pri, pub', pri', r^{i+1}_{pri})$.

A' is now given λ^1_{i+1} and a ciphertext C'. Using the pair (λ^1_{i+1}, C') A' obtains from A a leakage function f^2_{i+1}. A' also computes encryptions $C'_{\mathsf{key},i+2} = \mathsf{Enc}_{pub'}(\bar{0})$ and $C'_{\mathsf{res},i+1} = \mathsf{Enc}_{pub'}(F_{i+1}(K))$, and randomly selects r^{i+1}_{B1} and r^{i+1}_{B2}. A' sets

- $\mathsf{leak}_5(C)$:
  - Compute $C_{\mathsf{res},i+1} = \mathsf{EncEval}(pub', C', \mathsf{Evaluate}(F_{i+1}, C, \cdot); r^{i+1}_{B1})$.
  - Compute $C_{\mathsf{key},i+2} = \mathsf{EncEval}(pub', C', \mathsf{Refresh}(C, \cdot); r^{i+1}_{B2})$.

– Compute $C_{\mathsf{R0},i+1} = \mathsf{Subtract}(\mathit{pub}', C'_{\mathsf{res},i+1}, C_{\mathsf{res},i+1})$.
– Compute $C_{\mathsf{R1},i+1} = \mathsf{Subtract}(\mathit{pub}', C'_{\mathsf{key},i+2}, C_{\mathsf{key},i+2})$.
– Compute and return $f^2_{i+1}(C, \mathit{pub}', C_{\mathsf{R0}}, C_{\mathsf{R1}}, r^{i+1}_{B1}, r^{i+1}_{B2})$.

and obtains a value λ^2_{i+1}. A' uses $(\lambda_{i+1}, C'_{\mathsf{res},i+1})$ to obtain f^3_{i+1} from A. A' is then given pri'. From this point onward, A' simulates A according to Hyb_2. Note that the only value which is not generated by A' that is needed to perform this simulation is pri'. At the end of the simulation A outputs a bit $\hat{b}$, which A' also outputs. By construction, we have that if C is an encryption of $\bar{0}$ then A' simulates A perfectly in Hyb^i_{12}, and if C is an encryption of K, A is simulated perfectly in Hyb^{i+1}_{12}.

Notice that since A' simulates A along with the experiment with which A is interacting, and does some additional work in rounds i and $i+1$, A' runs it time at most $4 \cdot time_n(\mathsf{Enc}) + time_n(LRKP \leftrightarrow A)$.

It remains to handle the cases $i = 0$ and $i = n^d$. These are handled similarly to the first case, except we have to take into account the fact that A sees no leakage or communication during "rounds" 0 and $n^d + 1$. More specifically, for the case $i = 0$, A' proceeds as in the first case except that it does not submit leak_1, leak_2, or leak_3 in round i (or, alternatively, it submits constant functions and ignores their output), nor does it produce $C'_{\mathsf{res},0}$ or give anything to A during round i; for round $i+1$, A' proceeds as in the first case, starting by obtaining leakage function f^1_{i+1} from A. For the case $i = n^d$, A' proceeds as in the first case except that it does not submit leak_4 or leak_5 (or, alternatively, it submits constant functions and ignores their output), nor does it produce C'_{res,n^d+1} or give anything to A during round $i+1$.

Once again, for both these cases, we have by construction that if C is an encryption of $\bar{0}$ then A' simulates A perfectly in Hyb^i_{12}, and if C is an encryption of K, A is simulated perfectly in Hyb^{i+1}_{12}. It then follows by standard arguments that A' succeeds with advantage $\varepsilon(n)/(n^d+1)$ in Exp_1.

□

B.2 Analysis of Exp_1

We shall now prove an upper bound on the advantage of adversaries in Exp_1. Let A_1 be an adversary in Exp_1 and let ε be its advantage. Then, we show that there exists an adversary A_2 that succeeds with advantage $\varepsilon(n)/2^{l(n)}$ in the following experiment Exp_2.

Experiment Exp_2. Exp_2 proceeds identically to Exp_1, except steps 8-11 are modified as follows

8. The adversary submits a leakage function $\mathsf{leak}_4 : \{0,1\}^{3n} \to \{0,1\}^{l(n)+1}$, and sees $\mathsf{leak}_4(\mathit{pri}, \mathit{pri}', r_{\mathit{pri}'})$.
9. The adversary sees $C' = \mathsf{Enc}_{\mathit{pub}'}(\mathit{pri}; r_{\mathit{pri}'})$ and C.
10. The adversary outputs a bit $\hat{b}$.

Claim B.4. *Let A_1 be an adversary and define, for all n, $\varepsilon(n)$ to be the advantage of A_1 in Exp_1. Then there exists an adversary A_2 that, for all n, runs in time at most $3 \cdot time_n(A_1) + time_n(\mathsf{Enc})$ and succeeds in Exp_2 with advantage $\varepsilon(n)/2^{l(n)}$.*

Proof. A_2 randomly selects a string r_{sim} to use as the randomness of A_1. Then, using r_{sim}, A_2 simulates A_1 up to and including step 7 without any modifications. In step 8, A_2 obtains a leakage function leak_4 from A_1, and randomly selects guessed leakage value $\hat{\lambda}_5 \in \{0,1\}^{l(n)}$. A_2 then constructs a new leakage function:

- $\mathsf{leak}'_4(\mathit{pri}, \mathit{pri}', r_{\mathit{pri}'})$: First, compute $\lambda_4 = \mathsf{leak}_4(\mathit{pri}, \mathit{pri}', r_{\mathit{pri}'})$. Then, compute $C' = \mathsf{Enc}_{\mathit{pub}'}(\mathit{pri}; r_{\mathit{pri}'})$, and use $(\lambda_4, C', \hat{\lambda}_5)$ to complete the simulation of A_1 (using randomness r_{sim}). Let $\hat{b}$ be the bit output by A_1. The output of leak'_4 is then $(\lambda_4, \hat{b})$.

A_2 submits leak'_4 in step 8, and is given (λ'_4, C', C) where $\lambda'_4 = (\lambda_4, \hat{b})$. Using (λ_4, C'), A_2 obtains from A_1 the leakage function leak_5. Now, A_2 checks whether $\mathsf{leak}_5(C) = \hat{\lambda}_5$, and if so it outputs $\hat{b}$. Otherwise, A_2 flips an unbiased coin and outputs the outcome. Observe that A_2 runs in time at most $3 \cdot time_n(A_1) + time_n(\mathsf{Enc})$.

Notice that if A_2 guesses the leakage $\hat{\lambda}_5$ correctly then it simulates A_1 perfectly, and that the leakage is guessed correctly with probability $1/2^{l(n)}$. We therefore conclude:

$$|\Pr[(A_2 \leftrightarrows Exp_2) = 1 | b = 0] - \Pr[(A_2 \leftrightarrows Exp_2) = 1 | b = 1]| \geq$$
$$\frac{1}{2^{l(n)}} |\Pr[(A_1 \leftrightarrows Exp_1) = 1 | b = 0] - \Pr[(A_1 \leftrightarrows Exp_1) = 1 | b = 1]|$$

□

We now simplify the experiment further. For clarity, we describe the modified experiment completely:

Experiment Exp_3**.** The new experiment proceeds as follows:

1. The adversary submits two messages m_0, m_1 such that $m_0 \neq m_1$.
2. Private keys pri, pri' are randomly chosen, and public keys $pub = \mathsf{KeyGen}(pri)$, $pub' = \mathsf{KeyGen}(pri')$ are computed, and given to the adversary.
3. The adversary submits a leakage function $\mathsf{leak}_1 : \{0,1\}^{3n} \to \{0,1\}^{3l(n)+1}$, and sees $\mathsf{leak}_1(pri, pri', r_{pri'})$.
4. A random bit b is chosen, and an encryption $C = \mathsf{Enc}_{pub}(m_b)$ is computed.
5. The adversary sees $C' = \mathsf{Enc}_{pub'}(pri; r_{pri'})$ and C.
6. The adversary outputs a bit $\hat{b}$.

Claim B.5. *Let A_2 be an adversary and define, for all n, $\varepsilon(n)$ to be the advantage of A_2 in Exp_2. Then there exists an adversary A_3 that, for all n, runs in time at most $4 \cdot time_n(A_2)$ that succeeds in Exp_3 with advantage $\varepsilon(n)/2^{l(n)}$.*

Proof. The basic idea is the same as in the proof of Claim B.4. A_3 guesses a response $\hat{\lambda}_2 \in \{0,1\}^{l(n)}$ to A_2's leak_2 query, uses this guess to simulate A_2 within the leakage function that A_3 submits, and then verifies its guess. The details follow.

A_3 randomly selects a string r_{sim} to use as the randomness of A_2. Then, using r_{sim}, A_3 starts simulating A_2, obtaining a leakage function leak_1. A_3 randomly selects a guessed leakage value $\hat{\lambda}_2 \in \{0,1\}^{l(n)}$, and then constructs a new leakage function:

- $\mathsf{leak}'_1(pri, pri', r_{pri'})$: First, compute $\lambda_1 = \mathsf{leak}_1(pri)$. Simulate A_2, using randomness r_{sim} and using $(\lambda_1, \hat{\lambda}_2)$ as the responses to the first two leakage queries. A_2 then produces a leakage function leak_3. Compute $\lambda_3 = \mathsf{leak}_3(pri)$, and continue simulating A_2 using λ_3. A_2 produces a leakage function leak_4. Compute $\lambda_4 = \mathsf{leak}_4(pri, pri', r_{pri'})$. The output of leak'_1 is then $(\lambda_1, \lambda_3, \lambda_4)$.

A_3 submits leak'_1, and is given $(\lambda'_1, C'C)$, where $\lambda'_1 = (\lambda_1, \lambda_3, \lambda_4)$. A_3 continues its simulation of A_2, using λ_1 as the response to the first leakage query. A_2 then produces a leakage function leak_2. Now, A_3 checks whether $\mathsf{leak}_2(C) = \hat{\lambda}_2$; if not, A_3 outputs a randomly selected bit. Otherwise, A_3 continues simulating A_2, using λ_3 and λ_4 as the responses to the next two leakage queries. Then, A_3 gives C' and C to A_2, and outputs the bit output by A_2. Observe that A_3 runs in time at most $4 \cdot time_n(A_2)$.

Notice that if A_3 guesses the leakage $\hat{\lambda}_2$ correctly then it simulates A_2 perfectly, and that the leakage is guessed correctly with probability $1/2^{l(n)}$. We therefore conclude:

$$|\Pr[(A_3 \leftrightarrows Exp_3) = 1|b = 0] - \Pr[(A_3 \leftrightarrows Exp_3) = 1|b = 1]| \geq$$
$$\frac{1}{2^{l(n)}} |\Pr[(A_2 \leftrightarrows Exp_2) = 1|b = 0] - \Pr[(A_2 \leftrightarrows Exp_2) = 1|b = 1]|$$

□

We again simplify the experiment, this time moving to a leakage-free setting.

Experiment Exp_4**.** Exp_4 proceeds identically to Exp_3, except step 3 is omitted.

Claim B.6. *For all functions $\varepsilon'(n) > 0$ and $\varepsilon(n) > 0$, and for every adversary A_3 that succeeds in Exp_3 with advantage $\varepsilon'(n)$ for infinitely many n, there exists an adversary A_4 that runs in time at most $\frac{2^{3l(n)+1}}{\varepsilon(n)^2}(3l(n)+4+\log\frac{1}{\varepsilon(n)})(time_n(A_3)+time_n(\mathsf{Enc}))$ and succeeds in Exp_4 with advantage $\varepsilon'(n)-6\varepsilon(n)$ for infinitely many n.*

Proof. The key observation is that the response to A_3's leakage query is independent of bit b and the randomness used when producing encryption $C = \mathsf{Enc}_{pub}(m_b)$. This allows us to use the observation of Akavia *et al* [1] that for every public key encryption system, every adversary that breaks semantic security given leakage on KeyGen can be simulated by an adversary that is not given leakage but instead guesses the leakage and then tests whether the guessed leakage is good. Specifically, given pub, pub', and $C' = \mathsf{Enc}_{pub'}(pri)$, we can find a good response $\hat{\lambda}_1 \in \{0,1\}^{3l(n)+1}$ to A_3's leakage query that (almost) maximizes the distinguishing advantage of A_3 conditioned on pri, pri', and C'. To do so, we define an adversary A_4 that tests all strings $\hat{\lambda}_1 \in \{0,1\}^{l(n)}$ until it finds a leakage value that maximizes the gap between A_3's probability of outputting 1 on an encryption of m_0, and on an encryption of m_1. This is done by sampling, for each value $\hat{\lambda}_1$, many encryptions of m_0 and of m_1, and recording A_3's output. The details follow.

Without loss of generality, suppose

$$\Pr[(A_3 \leftrightarrows Exp_3) = 1|b = 1] - \Pr[(A_3 \leftrightarrows Exp_3) = 1|b = 0] \geq \varepsilon'(n)$$

for infinitely many n. A_4 behaves as follows. A_4 randomly selects a string r_{sim} to use as the randomness of A_3. Then, using r_{sim}, A_4 starts simulating A_3. A_3 submits messages m_0 and m_1, which are in turn submitted by A_4. Then, A_4 is given (pub, pub', C', C), where $C' = \mathsf{Enc}_{pub'}(pri, r_{pri'})$ and $C = \mathsf{Enc}_{pub}(m_b)$. A_4 continues simulating A_3, giving it pub and pub', and obtaining a leakage function leak_1. Recall from Exp_3 that leak_1 takes input $(pri, pri', r_{pri'})$. This means that the correct response to this leakage query is independent of bit b and the randomness used when producing C. Since A_4 cannot make any leakage queries, it runs experiments in order to determine the response that (almost) maximizes the distinguishing advantage of A_3 (conditioned on r_{sim}, pri, pri', and $r_{pri'}$).

Specifically, for each $\hat{\lambda}_1 \in \{0,1\}^{3l(n)+1}$ and $\hat{b} \in \{0,1\}$, A_4 does the following $m = \frac{1}{2\varepsilon(n)^2}(3l(n) + 3 + \log\frac{1}{\varepsilon(n)})$ times: A_4 produces a random encryption $C'' = \mathsf{Enc}_{pub}(m_b)$, runs A_3 with randomness r_{sim} on $(pub, pub', \hat{\lambda}_1, C', C'')$, and notes the output of A_3. This allows A_4 to obtain an estimate $p_{\hat{\lambda}_1,\hat{b}}$ of the probability that A_3 outputs 1 conditioned on r_{sim}, pri, pri', $r_{pri'}$, $\hat{\lambda}_1$, and $\hat{b}$. Then, for each $\hat{\lambda}_1 \in \{0,1\}^{3l(n)+1}$, A_4 computes $\varepsilon_{\hat{\lambda}_1} = p_{\hat{\lambda}_1,1} - p_{\hat{\lambda}_1,0}$ to obtain an estimate of the distinguishing advantage of A_3 (conditioned on r_{sim}, pri, pri', and $r_{pri'}$) when $\hat{\lambda}_1$ is used as the response to the leakage query.

A_4 then continues its original simulation of A_3, letting the response to the leakage query be the $\hat{\lambda}_1$ which maximizes $\varepsilon_{\hat{\lambda}_1}$. A_4 then gives C' and C to A_3, and outputs the bit output by A_3. Observe that A_4 runs in time $time_n(A_3) + m(time_n(A_3) + time_n(\mathsf{Enc})) \leq \frac{2^{3l(n)+1}}{\varepsilon(n)^2}(3l(n) + 4 + \log\frac{1}{\varepsilon(n)})(time_n(A_3) + time_n(\mathsf{Enc}))$.

Now, fix an n such that $\Pr[(A_3 \leftrightarrows Exp_3) = 1|b = 1] - \Pr[(A_3 \leftrightarrows Exp_3) = 1|b = 0] \geq \varepsilon'(n)$ and consider the advantage A_4 in Exp_4 for such n. Note that A_4 produces $2(2^{3l(n)+1})$ estimates. Using Chernoff bounds, each estimate $p_{\hat{\lambda}_1,\hat{b}}$ is within additive error $\varepsilon(n)$ of its true value with probability at least $1 - 2e^{-2m\varepsilon(n)^2} = 1 - 2e^{-3l(n)-3-\log\frac{1}{\varepsilon}} \geq 1 - 2^{-3l(n)-2-\log\frac{1}{\varepsilon}} = 1 - \frac{\varepsilon(n)}{2^{3l(n)+2}}$. Then, by the union bound, *all* estimates $p_{\hat{\lambda}_1,\hat{b}}$ are within $\varepsilon(n)$ of their true values with probability at least $1 - \varepsilon(n)$. Observe that when this happens, all estimates $\varepsilon_{\hat{\lambda}_1}$ are within $2\varepsilon(n)$ of their true values. In this case, the $\hat{\lambda}_1$ chosen by A_4 results in A_3 having true distinguishing advantage within $4\varepsilon(n)$ of whichever response yields the best true distinguishing advantage conditioned on r_{sim}, pri, pri', and $r_{pri'}$

Putting this all together, conditioned on each r_{sim}, pri, pri', and $r_{pri'}$, we have that with probability at least $1 - \varepsilon(n)$, A_4 has distinguishing advantage within $4\varepsilon(n)$ of the distinguishing advantage of A_3 (subject to the same conditioning). It follows that overall (without conditioning), with probability at least $1 - \varepsilon(n)$, A_4 has distinguishing advantage within $4\varepsilon(n)$ of the distinguishing advantage of A_3. That is, A_4 has distinguishing advantage at least $(1 - \varepsilon(n))(\varepsilon'(n) - 4\varepsilon(n)) - \varepsilon(n) \geq \varepsilon'(n) - 6\varepsilon(n)$. □

It is easy to see that an adversary that succeeds in experiment Exp_4 can be used to break the semantic security of $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$. The idea is that such an adversary must either distinguish $\mathsf{Enc}_{pub'}(pri)$ from $\mathsf{Enc}_{pub'}(\bar{0})$, or must succeed at guessing b even when given $\mathsf{Enc}_{pub'}(\bar{0})$ instead of $\mathsf{Enc}_{pub'}(pri)$. The proof of Claim B.7 is deferred to Appendix C.

Claim B.7. *For every function $\varepsilon(n) > 0$ and for every adversary A_4 that succeeds in Exp_4 with advantage $\varepsilon(n)$ for infinitely many n, there exists an adversary A_5 that runs in time at most $time_n(A_4) + time_n(\mathsf{KeyGen}) + time_n(\mathsf{Enc})$ and breaks the semantic security of $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ with advantage $\varepsilon(n)/3$ for infinitely many n.*

Combining all the claims, we see that for all functions $\varepsilon(n) > 0$ if there exists an adversary A such that $|\Pr[(A \leftrightarrows \mathsf{Hyb}_{12}^0) = 1] - \Pr[(A \leftrightarrows \mathsf{Hyb}_{12}^{n^d+1}) = 1]| > \varepsilon(n)$ for infinitely many n, then for every function $\varepsilon'(n) > 0$ there exists an adversary A' that runs in time

$$\frac{2^{3l(n)+5}}{\varepsilon'(n)^2}\left(3l(n) + 4 + \log\frac{1}{\varepsilon'(n)}\right)(4 \cdot time_n(\mathsf{Enc}) + time_n(LRKP \leftrightarrow A)) + time_n(\mathsf{KeyGen})$$

and breaks the semantic security of $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ with advantage $\frac{\varepsilon(n)}{3 \cdot 2^{2l(n)}(n^d+1)} - 2\varepsilon'(n)$ for infinitely many n.

C Proof of Claim B.7

Proof. Let experiment Exp_5 be identical to Exp_4 except C' is set to $\mathsf{Enc}_{pub'}(\bar{0})$ instead of $\mathsf{Enc}_{pub'}(pri)$. There are two cases to consider:

Case 1: For infinitely many n, A_4 has advantage at least $\varepsilon(n)$ in Exp_4 and has advantage at least $\varepsilon(n)/3$ in Exp_5.

Let A_5 behave as follows. A_5 starts simulating A_4. A_4 submits messages m_0 and m_1, which are in turn submitted by A_5. Then, A_5 is given (pub, C), where $C = \mathsf{Enc}_{pub}(m_b)$. A_5 randomly selects pri', and lets $pub' = \mathsf{KeyGen}(pri')$. Then A_5 produces $C' = \mathsf{Enc}_{pub'}(\bar{0})$, gives (pub, pub', C', C) to A_4, and outputs the bit output by A_4.

Notice that A_5 simulates $A_4 \leftrightarrows Exp_5$ perfectly, and hence has the same distinguishing advantage as A_4. That is, A_5 breaks the semantic security of $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ with advantage $\varepsilon(n)/3$ for infinitely many n. Observe that A_5 runs in time $time_n(A_4) + time_n(\mathsf{KeyGen}) + time_n(\mathsf{Enc})$.

Case 2: For infinitely many n, A_4 has advantage at least $\varepsilon(n)$ is Exp_4 and has advantage less than $\varepsilon(n)/3$ in Exp_5. Without loss of generality, suppose

$$\Pr[(A_4 \leftrightarrows Exp_4) = 1 | b = 1] - \Pr[(A_4 \leftrightarrows Exp_4) = 1 | b = 0] \geq \varepsilon(n) \tag{1}$$

and

$$\Pr[(A_4 \leftrightarrows Exp_5) = 1 | b = 1] - \Pr[(A_4 \leftrightarrows Exp_5) = 1 | b = 0] < \frac{\varepsilon(n)}{3} \tag{2}$$

for infinitely many n. Let A_5 behave as follows. A_5 randomly selects pri, and lets $pub = \mathsf{KeyGen}(pri)$. A_5 submits $m'_0 = \bar{0}$ and $m'_1 = pri$. Then, A_5 is given (pub', C'), where $C' = \mathsf{Enc}_{pub'}(m'_{b'})$ for some $b' \in \{0, 1\}$. Now, A_5 starts A_5 starts simulating A_4. A_4 submits messages m_0 and m_1. A_5 randomly selects $b \in \{0, 1\}$, produces $C = \mathsf{Enc}_{pub}(m_b)$, and gives (pub, pub', C', C) to A_4. Then, A_4 outputs a bit. If this bit is b, A_5 outputs 1, and otherwise A_5 outputs 0.

Now, fix an n such that (1) and (2) both hold, and consider the advantage of A_5 in breaking the semantic security of $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$. The key observation is that when $b' = 0$, A_5 simulates $A_4 \leftrightarrows Exp_5$ perfectly, and when $b' = 1$, A_5 simulates $A_4 \leftrightarrows Exp_4$ perfectly. Then we have

$$\begin{aligned}
\Pr[A_5 \text{ outputs } 1 | b' = 0] &= \frac{1}{2}\Pr[(A_4 \leftrightarrows Exp_5) = 1 | b = 1] + \frac{1}{2}\Pr[(A_4 \leftrightarrows Exp_5) = 0 | b = 0] \\
&= \frac{1}{2}\left(\Pr[(A_4 \leftrightarrows Exp_5) = 1 | b = 1] + 1 - \Pr[(A_4 \leftrightarrows Exp_5) = 1 | b = 0]\right) \\
&< \frac{1}{2}\left(1 + \frac{\varepsilon(n)}{3}\right)
\end{aligned}$$

where the inequality is by (2). We also have

$$\begin{aligned}
\Pr[A_5 \text{ outputs } 1 | b' = 1] &= \frac{1}{2}\Pr[(A_4 \leftrightarrows Exp_4) = 1 | b = 1] + \frac{1}{2}\Pr[(A_4 \leftrightarrows Exp_4) = 0 | b = 0] \\
&= \frac{1}{2}\left(\Pr[(A_4 \leftrightarrows Exp_4) = 1 | b = 1] + 1 - \Pr[(A_4 \leftrightarrows Exp_4) = 1 | b = 0]\right) \\
&\geq \frac{1}{2}\left(1 + \varepsilon(n)\right)
\end{aligned}$$

where the inequality is by (1). But this means that

$$\Pr[A_5 \text{ outputs } 1 | b' = 1] - \Pr[A_5 \text{ outputs } 1 | b' = 0] > \frac{1}{2}\left(1 + \varepsilon(n)\right) - \frac{1}{2}\left(1 + \frac{\varepsilon(n)}{3}\right) = \frac{\varepsilon(n)}{3}$$

That is, A_5 breaks the semantic security of $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ with advantage $\varepsilon(n)/3$ for infinitely many n. Observe that A_5 runs in time $time_n(A_4) + time_n(\mathsf{KeyGen}) + time_n(\mathsf{Enc})$. □

D Concurrent Composition

We have shown that an adversary interacting with a single LRKP gains no information about the underlying key. However, for some applications, such as private key encryption where several parties compute on the same agreed upon key, this may not suffice. It is quite possible that the adversary is performing side-channel attacks on several parties simultaneously, and is coordinating his leakage functions adaptively. In this section we show that an adversary interacting with several instances of LRKP concurrently still gains no information through leakage. This allows us to obtain some of the applications described in Section 5. We start with a definition.

Definition D.1. Let A and S be PPT algorithms, $n \in \mathbb{N}$, and consider the following two experiments:

$\mathsf{ExpConcurrentReal}$**.** The adversary chooses n keys $K_1, \ldots, K_n$ and interacts with n instances of $\mathsf{ExpReal}$ where in instance i, K_i is protected by LRKP_i. At the end, the adversary outputs a bit b. During the interaction the adversary controls the schedule of the queries completely, and in particular leakage queries on LRKP_i may depend on leakage obtained from LRKP_j for $i \neq j$.

$\mathsf{ExpConcurrentIdeal}$**.** The adversary chooses n keys $K_1, \ldots, K_n$ interacts with a single simulator S, and eventually outputs a bit b.

Then, we say that LRKP is a *Concurrent-Leakage Resilient Key Proxy (C-LRKP)* if for every PPT A there exists a PPT S and a negligible function $neg(\cdot)$ such that

$$|\Pr[(A \leftrightarrows \mathsf{ExpConcurrentReal}) = 1] - \Pr[(A \leftrightarrows \mathsf{ExpConcurrentIdeal}) = 1]| \leq neg(n)$$

We now show that any LRKP is also concurrent-LRKP. This follows due to the strong security guarantee of LRKP: even when the adversary herself selects the key K, she still cannot distinguish between simulated and actual leakage.

Theorem D.2. *Let* LRKP *be a leakage resilient key proxy. Then,* LRKP *is also concurrent leakage resilient.*

Proof. Suppose that LRKP is insecure according to Definition D.1. Then, there exists an adversary A such that for every simulator, A distinguishes between $\mathsf{ExpConcurrentReal}$ and $\mathsf{ExpConcurrentIdeal}$. Let S be the simulator of LRKP in the non-concurrent setting, and consider a simulator S' that runs n copies of S in parallel. Copy i is used to simulate leakage from LRKP_i.

Consider a sequence of hybrid experiments Hyb_i, $0 \leq i \leq n$ such that in Hyb_i the adversary obtains leakage from the actual state of LRKP_j for $1 \leq j \leq i$, and obtains simulated leakage for $i+1 \leq j \leq n$. Note that Hyb_0 is $\mathsf{ExpConcurrentIdeal}$ and Hyb_n is $\mathsf{ExpConcurrentReal}$. Thus, there exists an i such that if A distinguishes between $\mathsf{ExpConcurrentReal}$ and $\mathsf{ExpConcurrentIdeal}$ with probability ε then A distinguishes Hyb_i from Hyb_{i+1} with probability at least ε/n.

We now construct an adversary A' that simulates A and breaks the non-concurrent security of LRKP. The simulation proceeds as follows: A' starts simulating A, which chooses n keys $K_1, \ldots, K_n$. A' then initializes LRKP_j for $1 \leq j \leq i-1$ with K_j, submits K_i as its own key in the LRKP security experiment, and chooses the initial randomness independently for $n - i$ copies of S.

A' then continues simulating A, answering queries as follows: leakage queries about LRKP_j for $1 \leq j \leq i-1$ are answered by applying the leakage function to the actual state of LRKP_j. For $i+1 \leq j \leq n$ the leakage queries are forwarded to the $j - i$th copy of S. Leakage queries for LRKP_i are forwarded by A' as her own queries. At the end of the simulation A' outputs what A outputs. It is not hard to see that when A' is interacting with $\mathsf{ExpReal}$ and $\mathsf{ExpIdeal}$ it is simulating A perfectly in Hyb_i and Hyb_{i+1} respectively. □

E Application: Private Key Encryption

E.1 Semantic Security Under Leakage

Encryption is one of the most important products of cryptography. In the classical setting, where side-channel attacks are not taken into account, there are widely accepted definitions of security for both the private and the public key setting. For a rigorous exposition, we direct the reader to [14, 19]. Informally, the accepted notion of privacy, which is commonly referred to as "semantic security", is to require that no efficient adversary can distinguish between the encryptions of two messages of her choice.

Extending the traditional notions of semantic security to the leakage setting is non-trivial. In particular, suppose that we assume that every invocation of the encryption algorithm leaks information. Then, since the message plaintext is an input to that algorithm, the adversary can trivially break semantic security by simply leaking a bit that differentiates the two messages in question. Consequently, in the setting where "everything leaks", traditional semantic security cannot be achieved. This leads us to consider several alternatives to the naive definition, which permit non-trivial results. Below, we outline some of the possible approaches for dealing with privacy under leakage, and describe the choice that we made in this paper.

Leak-free challenge. One approach to dealing with the trivial impossibility described above is to weaken the requirement that "everything leaks" to allowing everything to leak *except* the computation of the actual ciphertext that the adversary is trying to distinguish. This solution has been adopted by several works on leakage-resilient encryption (see e.g. [9, 24, 8, 5]). These works all deal with what we call "bounded leakage", that is, the amount of information that the adversary obtains on the key during its entire lifetime is bounded. Still, the issue that we mentioned about semantic security applies, but for a different reason. In most constructions in the bounded leakage model, the key remains fixed after it is generated; such constructions are clearly insecure in the "everything leaks" model since the entire key eventually leaks. In the bounded leakage model, such constructions turn out to be insecure when there is leakage on the challenge ciphertext.

The problem is that if the adversary is allowed to obtain leakage on the key after she has seen the challenge ciphertext, she can simply use the key to decrypt the ciphertext within the leakage function, and leak the information that distinguishes the two messages in question, thereby breaking semantic security. Consequently, as in our setting, some restrictions on the leakage, or a weakening of the definition of security are necessary.

We adopt the leak-free challenge approach for our applications. We prefer this solution to the ones listed below because it permits fairly clean definitions while allowing the other notions to be achieved through simple transformations and reductions.

Leakage on random messages. Instead of weakening the requirement that everything leaks, we can relax the definition of semantic security so that it is still meaningful in the leakage setting. Instead of requiring that the adversary fail to distinguish between the encryption of two messages, we can require that she does not learn too much about a message, if it is sampled from a distribution with a sufficiently high min-entropy. That is, we can ask for essentially the best that can be hoped for: that the adversary obtains no more information through leakage on the encryption process than what she would be able to obtain through leakage only on the message that is being encrypted. This notion of security seems to capture accurately what is achievable in terms of privacy in a setting with leakage. However, we choose not to adopt this notion both because it is more cumbersome than assuming a leak free

challenge, and, more importantly, because it does not seem to be easily usable in applications which require semantically secure private key encryption as an underlying tool.

E.2 Leakage-Resilient Private Key Encryption Using Key Proxies

We extend the standard definition of semantic security to the leakage setting with a leak-free challenge. One issue that arises in the private key setting is that in a typical application, several parties will hold the same key K which is used both for encryption and decryption. In order to maintain generality, it is therefore important to allow a leakage adversary to obtain leakage on each of the parties according to her own schedule. With this in mind, we now define a leakage resilient private key encryption scheme.

A stateful private key encryption scheme consists of three PPT algorithms $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$. The key generation algorithm $\mathsf{KeyGen}(1^n)$ outputs n initial states $S_1^0, \ldots, S_n^0$ that are held by n individual parties. These states correspond to the initial encodings of some key K. For $j \in \mathbb{N}$, the encryption algorithm $\mathsf{Enc}(M, S_i^j)$ outputs a ciphertext C, and an updated state S_i^{j+1}. The decryption algorithm $\mathsf{Dec}(C, S_i^j)$ outputs a message M, and an updated state S_i^{j+1}.

Definition E.1. A triple of PPT algorithms $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ is a correct *stateful private key encryption scheme* if for all random tapes R, for all $1 \leq i, i' \leq n$, all $j, j' \in \mathbb{N}$, and all $M \in \{0,1\}^n$, $\mathsf{Dec}(\mathsf{Enc}(M, S_i^j; R), S_{i'}^{j'}) = M$.

We can now describe the experiment *ExpSemSec*(b), for $b \in \{0,1\}$, of semantic security under leakage.

1. *Initialization.* The key generation algorithm $\mathsf{KeyGen}(1^n)$ is run to obtain $S_1^0, \ldots, S_n^0$.
2. *Encryption Queries.* The adversary may initiate an arbitrary number of encryption processes by submitting a message M, and an index $1 \leq i \leq n$. An encryption $C = \mathsf{Enc}(M, S_i^j)$ is then computed, where j is the number of times party i encrypted until now, and the adversary concurrently obtains single invocation leakage on all the active encryption processes (the single invocation leakage model for private key encryption is described in Section E.2.2). The adversary is then given C.
3. *Challenge.* At some point the adversary submits two messages M_0, M_1, and an index $1 \leq i \leq n$ for which there is no current active encryption process, and obtains $C^* = \mathsf{Enc}(M_b, S_i^j)$.
4. *Encryption Queries.* The adversary continues to initiate encryption processes, and concurrently obtain leakage on these processes.
5. *Guess.* The adversary outputs a bit b'.

Definition E.2. A stateful private key encryption scheme $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ is *semantically secure under leakage*, if for all PPT adversaries A,

$$|\Pr[(A \leftrightarrows \textit{ExpSemSec}(0)) = 1] - \Pr[(A \leftrightarrows \textit{ExpSemSec}(1)) = 1]| \leq neg(n)$$

E.2.1 Construction

Let $F = \{F_n\}_{n\in\mathbb{N}}$ be a family of pseudorandom functions (when n is clear from context we write F instead of F_n) such that $F_n : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ for all $n \in \mathbb{N}$. Let LRKP be a leakage resilient key proxy. Our stateful private key encryption scheme $\mathsf{PRI-ENC} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ works as follows:

Key Generation. The key generation algorithm, $\mathsf{KeyGen}(1^n)$, first chooses a key $K \in_R \{0,1\}^n$ at random, and runs $\mathsf{KPInit}(1^n, K)$ n times with independently chosen randomness to obtain n initial states $S_1, \ldots, S_n$. The states $S_1, \ldots, S_n$ are the output of KeyGen.

Encryption. For a message $M \in \{0,1\}^n$, the encryption algorithm $\mathsf{Enc}(M, S)$ chooses a random string $R \in_R \{0,1\}^n$, and generates a circuit $H(x)$ that computes the function $F_x(R) \oplus M$. It then runs $\mathsf{KPEval}(H, S)$ to obtain an output Y, and an updated state S'. The ciphertext is then $C = (Y, R)$, and output of Enc is (C, S').

Decryption. The decryption algorithm $\mathsf{Dec}(C, S)$ parses C as (Y, R), and generates a circuit $G(x)$ that computes the function $F_x(R) \oplus Y$. It then runs $\mathsf{KPEval}(G, S)$ to obtain an output M, and an updated state S'. The output of Dec is then (M, S').

E.2.2 Single Invocation Leakage Model

The single invocation leakage for our construction is quite simple: during encryption, the adversary is given R and M, and then proceeds to interact in a single invocation leakage experiment with the computation of KPEval. During decryption, the adversary simply interacts in a single invocation leakage experiment with KPEval. In other words, for both encryption and decryption the adversary obtains leakage on the computation of KPEval, and obtains all the other inputs completely.

We note that although it may seem more reasonable to allow the adversary to learn only part of the randomness and message of the encryption, it would give us a weaker theorem. The fact that our construction is secure in the above leakage model implies security in other more realistic but weaker models.

E.2.3 Security Analysis

We show that any adversary that breaks the semantic security of $\mathsf{PRI-ENC}$ can be used to break either the concurrent leakage resilience of LRKP or the pseudorandomness of F. We start by stating the theorem:

Theorem E.3. *Let A be a PPT adversary for the semantic security under leakage of $\mathsf{PRI-ENC}$, let LRKP be a concurrently leakage resilient key proxy such that all adversaries running in time at most $time_n(A)$ can distinguish real and simulated leakage with advantage at most $\varepsilon_{\mathsf{c-lrkp}}$, and let $F = \{F_n\}_{n\in\mathbb{N}}$ be a family of PRFs such that all adversaries running in time at most $time_n(A) \cdot (time_n(\mathsf{KPEval}) + time_n(\mathsf{Enc})) + time_n(\mathsf{KeyGen})$ can distinguish F_n from random with advantage at most $\varepsilon_{\mathsf{prf}}$. Then, A breaks the semantic security of $\mathsf{PRI-ENC}$ with advantage at most $\varepsilon_{\mathsf{c-lrkp}} + \varepsilon_{\mathsf{prf}} + \frac{time_n(A)}{2^n}$.*

To prove security we define several hybrid experiments where the first hybrid Hyb_0 is the original experiment of semantic security with leakage, and in the final hybrid the adversary obtains no information about the bit b.

Experiment Hyb_1**.** Experiment Hyb_1 proceeds as Hyb_0 except that the computation of the challenge is performed differently. Instead of using KPEval to compute it, the challenge ciphertext is computed directly by choosing a random string $R^* \in_R \{0,1\}^n$ and outputting $(F_K(R^*) \oplus M_b, R^*)$. The LRKP is evaluated on some constant function (e.g. one that always outputs $\bar{0}$) in order to refresh its state.

Experiment Hyb_2**.** In this experiment, the leakage obtained by the adversary is replaced by simulated leakage. More precisely, let S be the simulator for concurrent leakage that is guaranteed by Theorem D.2 to exist for LRKP. In experiment Hyb_2, whenever the adversary initiates an encryption process, the simulator S is given the corresponding circuit H, and the ciphertext C. Then, the adversary interacts with the simulator to obtain leakage on the underlying invocation of LRKP.

Experiment Hyb_3**.** In this experiment we replace the pseudorandom function F with a random one. Namely, for each new encryption process started by the adversary, the simulator S is given the circuit H, and a ciphertext of the form $C = (\hat{F}(R) \oplus M, R)$ where $\hat{F} : \{0,1\}^n \to \{0,1\}^n$ is a random function. The challenge ciphertext is also computed using the random function $\hat{F}$.

Let A be PPT adversary for the semantic security under leakage of $\mathsf{PRI-ENC}$. We define X_i to be the random variable that is 1 if A guesses the bit b correctly in experiment Hyb_i, for $0 \leq i \leq 3$.

Claim E.4. $\Pr[X_0 = 1] = \Pr[X_1 = 1]$

Proof Sketch. This follows directly from the fact that LRKP is history free according to Definition 3.5. In particular, it makes no difference whether we refresh the state of LRKP during the challenge by evaluating the actual circuit that is needed to compute the challenge, or a circuit that always outputs $\bar{0}$. □

Claim E.5. $|\Pr[X_1 = 1] - \Pr[X_2 = 1]| \leq \varepsilon_{\mathsf{c-lrkp}}$

Proof Sketch. Suppose that the claim is false, then we can use the adversary A to distinguish between simulated and real leakage in the concurrent LRKP experiment. Our adversary A' initializes n copies of LRKPs with some random PRF key K, and then simply acts as a middleman between A and the security experiment of the concurrent LRKP. □

Claim E.6. $|\Pr[X_2 = 1] - \Pr[X_3 = 1]| \leq \varepsilon_{\mathsf{prf}}$

Proof Sketch. Suppose that the claim is false, then we can use the adversary A to distinguish between an oracle for F_K and an oracle for a random function $\hat{F}$. To see this, note that in both Hyb_2 and Hyb_3 the leakage is simulated. Therefore, we can construct an adversary A' that simulates A given an oracle O. If $O \equiv F_K$ then A is simulated perfectly in Hyb_2. If O is a random function, then it is a correct simulation in Hyb_3. □

Claim E.7. $\Pr[X_3 = 1] \leq \frac{time_n(A)}{2^n}$

Proof Sketch. This follows from the fact that $\hat{F}$ is a random function, and that the simulated leakage is independent from the randomness of the challenge. □

F Application: Chosen Ciphertext Secure Public Key Encryption

Constructions of public key encryption schemes that are resilient to an a-priori bounded amount of leakage were recently given by [24, 2, 5]. However, no constructions are known of PKEs that remain secure under the Chosen Ciphertext Attack, if the adversary can obtain leakage during each decryption query. Chosen Ciphertext security is considered the right notion of security for PKE in practice, and providing schemes that satisfy this notion of security even under side-channel attacks is an important open problem.

LRKPs provide a convenient way of immunizing any CCA-PKE against leakage. More precisely, given a CCA-PKE scheme $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$, we construct a new PKE $(\mathsf{KeyGen}', \mathsf{Enc}, \mathsf{Dec}')$ where the encryption algorithm stays the same; the key generation KeyGen' runs KeyGen to obtain $(\mathit{pub}, \mathit{pri})$ and then initializes an LRKP with *pri*. The public key is *pub*, and the private key is the initial state state_1 of the LRKP. The decryption algorithm is stateful, and to decrypt a ciphertext C, Dec' generates a circuit $H(x)$ that computes that function $\mathsf{Dec}_x(C)$, and then uses KPEval to evaluate it on the private key *pri*.