Storing Secrets on Continually Leaky Devices

Yevgeniy Dodis
New York University
dodis@cs.nyu.edu

Allison Lewko
University of Texas, Austin
alewko@cs.utexas.edu

Brent Waters
University of Texas, Austin
bwaters@cs.utexas.edu

Daniel Wichs
New York University
wichs@cs.nyu.edu

Abstract— We consider the question of how to store a value secretly on devices that continually leak information about their internal state to an external attacker. If the secret value is stored on a single device from which it is efficiently retrievable, and the attacker can leak even a single predicate of the internal state of that device, then she may learn some information about the secret value itself. Therefore, we consider a setting where the secret value is *shared* between multiple devices (or multiple components of a single device), *each* of which continually leaks arbitrary adaptively chosen predicates its individual state. Since leakage is continual, each device must also continually update its state so that an attacker cannot just leak it entirely one bit at a time. In our model, the devices update their state individually and asynchronously, without any communication between them. The update process is necessarily randomized, and its randomness can leak as well.

As our main result, we construct a sharing scheme for two devices, where a constant fraction of the internal state of each device can leak in between and during updates. Our scheme has the structure of a public-key encryption, where one share is a secret key and the other is a ciphertext. As a contribution of independent interest, we also get public-key encryption in the *continual leakage model*, introduced by Brakerski et al. and Dodis et al. (FOCS '10). This scheme tolerates continual leakage on the secret key and the updates, and simplifies the recent construction of Lewko, Lewko and Waters (STOC '11). For our main result, we show how to update the ciphertexts of the encryption scheme so that the message remains hidden even if an attacker interleaves leakage on secret key and ciphertext shares. The security of our scheme is based on the *linear* assumption in prime-order bilinear groups.

We also provide an extension to general access structures realizable by linear secret sharing schemes across many devices. The main advantage of this extension is that the state of some devices can be compromised *entirely*, while that of the all remaining devices is susceptible to continual leakage. Lastly, we show impossibility of *information theoretic* sharing schemes in our model, where continually leaky devices update their state individually.

1. Introduction

One of the central tenets of theoretical computer science is that computation can be analyzed abstractly and independently of the physical processes that ultimately implement it. This is the paradigm usually followed in cryptography, where we analyze cryptographic algorithms as abstract computations that get inputs and generate outputs with the help of some internal secret state. Unfortunately, this abstraction may fail to properly model the real world where various physical attributes of a computational device (e.g. timing, power-consumption, temperature, radiation, acoustics, etc.) can be measured and may leak useful information about the internal state of the device. Attacks that use such information to break security are called *side-channel leakage attacks*, and they have been analyzed and exploited in many recent works (see e.g. [26], [12] and the references therein). These attacks pose a major challenge to the applicability of cryptographic theory to practice.

MODELING LEAKAGE. In recent years, cryptographic theory has taken on the challenge of modeling leakage attacks formally and constructing cryptographic primitives that remain provably secure even in the presence of such attacks. Several different proposed models of leakage have emerged, with an emphasis on capturing large general classes of leakage. In this work, we will focus on the *continual-leakage model*, which came as an extension of the earlier *bounded-leakage model*, both of which are discussed below.[1]

The bounded-leakage model was introduced by Akavia, Goldwasser and Vaikuntanathan [2] and it allows the attacker to learn arbitrary information about the internal secret state of a device, as long as the *total amount of leaked information* (measured in bits) is bounded by some parameter ℓ, called the *leakage bound*. In other words, the attacker can learn up to ℓ arbitrary (efficiently computable) predicates of the internal state of a device, throughout its lifetime. Unfortunately, by bounding the overall amount of observable leakage, this model does not seem to adequately capture an attacker with prolonged access to a device and the ability to make many side-channel measurements over time.

The continual-leakage model, introduced concurrently by Brakerski et al. [6] and Dodis et al. [8], addresses exactly this issue and allows the attacker to continually leak information about the internal state of a device over time, as long as only the *rate of leakage is bounded*. More specifically, the device has some internal notion of time periods and, at the end of each period, it updates its internal state, using some fresh local randomness. The attacker is allowed to learn up to ℓ predicates of the internal state (including the random coins of the update process) in each time period, but can do so for as many time periods as desired, and there is no bound on the total amount of information learned. Prior work in the bounded leakage model [2], [24], [3], [20] and

[1]For brevity, we do not discuss many prior models of leakage and other related results, but wish to emphasize that this area has a long and rich history beyond just the results in the last few years. See e.g. the survey of [4] for an overview.

the continual-leakage model [6], [8], [22], [21] construct encryption, signature and related primitives.

We also briefly mention an alternative model called the *only computation leaks information model* [23], [11]. This model also considers continual leakage but, in each time period, the attacker is limited to only leaking information about the portion of the state accessed by the "computation" during that period (and never on the full state of a device). There are several variants of this model depending on how one breaks up a computation into distinct time periods.

STORING SECRETS ON LEAKY DEVICES. In this work, we ask a basic question of how to store a secret value (message) on continually leaky devices while preserving its secrecy. Unfortunately, in the bounded and continual leakage models, it is impossible to store a message secretly on a single leaky device from which it is efficiently retrievable, because a single leaked predicate of the internal state of such device can reveal (say) the first bit of the message. There are two natural alternatives to overcoming this difficulty:

1. We can weaken the leakage model and restrict the attacker to only learning some *limited class of predicates* of the internal state of the device. This class should capture realistic attacks but cannot be powerful enough to recover the stored secret, even though there is an efficient method for doing so.
2. We can consider a model where the secret is shared between two or more devices, each of which leaks *individually* in the continual leakage model. The attacker can continually learn *arbitrary* predicates of the internal state of each individual device (but *not* of the combined joint state of all the devices).

In the rest of the paper, we will frame our discussion in terms of the *second* approach. However, this can also be naturally viewed as a concrete instantiation of the first approach, where we think of the state of a single device as divided into multiple *components*, and leakage is restricted to the limited class of predicates that each depend on only a single component. This may be a natural and realistic class of leakage attacks if the components of the state are e.g. stored in different areas of memory and accessed separately by the device. In particular, this can be seen as a *strengthening* of the "only computation leaks information" (OCLI) model. In the OCLI model, the various components leak individually but only when accessed by a computation, while here they leak individually but all the time. We note that this strengthening was explicitly considered by prior works in the OCLI model, starting with Dziembowski and Pietrzak [11] in the case of stream ciphers. Although prior results in *various* models of continual leakage construct many basic and advanced cryptographic primitives, they do not address the simple question of storing a consistent value secretly on leaky devices. Indeed, they rely on the fact that one does not need to store a consistent secret key over time to e.g. decrypt, sign, or generate a random stream.

Let us now describe our concrete model in more detail. We assume that each device has its own individual notion of *time periods*, but these notions can differ across devices and they need not be synchronized. At the end of each time period, a device updates its share using some local fresh randomness. This update is conducted individually, and the devices do *not* communicate during the update process. At any point in time, no matter how many updates occurred on each device, the shares of the devices can be efficiently combined to *reconstruct* the shared secret message.

For security, we allow the attacker to continually learn arbitrary (efficiently computable) predicates of the internal state of *each* device. The attacker can choose the predicates *adaptively* and can alternate leakage between the devices. The internal state of each device in each time period consists of the current version of its share and the randomness of the update process used to derive the next share.[2] This models the state of the device even *during* the update process, which we assume can leak as well. The attacker can arbitrarily schedule the updates on different devices. We only restrict it to leaking at most ℓ predicates from each device during each time period. The shared message should remain semantically secure throughout the game.[3] We call a scheme satisfying these criteria an *ℓ-continual-leakage-resilient sharing (CLRS)*.

A solution with just *two* continually leaky devices is optimal, and that adding more devices only makes the problem easier. However, we will also consider an extension where the state of some devices can be *fully* compromised, in which case having more devices will be useful.

OUR RESULTS. Our main result is to construct an ℓ-CLRS scheme between two devices, for any polynomial leakage-bound ℓ. The size of the shares necessarily depends on and exceeds the leakage bound ℓ. However, we guarantee that ℓ is a *constant* fraction of the share size (albeit a small constant), and hence we can interpret our results as saying that a constant fraction of each share can leak in each time period. The security of our scheme is based on the well-studied *linear* assumption in prime-order bilinear groups.

We also show that computational assumptions are necessary and that this primitive cannot be realized *information theoretically*, even for $\ell = 1$ bit of leakage. Intuitively, this is because an unbounded-time leakage function can enumerate the entire set of possible values reachable via updates. Although this set might shrink in each time period, we show how to consistently leak one *fixed* value in this set incrementally bit by bit, eventually learning it all.

[2]We implicitly assume that the devices can perfectly delete/overwrite past values during an update.

[3]The definition also implies security if one share is fully revealed at the end of the game (but no more leakage afterward). A distinguishing strategy that uses the fully revealed share to break security could also be encoded into a predicate, which we can just leak from said share to break security.

We also extend our positive result to general access structures realizable by *linear secret sharing schemes* over many devices. The main advantage is that the attacker can even *fully* corrupt some subset of the devices and continually leak from all others. We only require that the corrupted subset is unauthorized and remains unauthorized with the addition of any other single device.[4] Our main scheme for two devices becomes a special case, where we apply the above to a *two-out-of-two* linear sharing scheme.

Lastly, our ℓ-CLRS scheme has special structure where one share is a *secret key* and the other share is a *ciphertext* of a public key encryption scheme. This immediately gives *continual-leakage-resilient public-key encryption (CLR-PKE)* [6], [22], [21], where continual leakage on the secret key does not allow an attacker to decrypt future ciphertexts. Moreover, our scheme also allows for significant *leakage during the updates*. This property was recently achieved by a scheme of Lewko, Lewko and Waters [21] under a strong assumption called the generalized subgroup decisional assumption in composite-order bilinear groups. As a result of independent interest, we substantially simplify the scheme and the proof of [21], converting it into a scheme over the more common prime-order bilinear groups, with a proof of security under the more common linear assumption. We get other benefits along the way, including shorter public keys and improved efficiency by directly encrypting group elements rather than bits.

OUR TECHNIQUES. Our construction begins with the idea of using an encryption scheme, and making one share a secret key and the other share a ciphertext. Taking any of the recent results on CLR-PKE, we get a method for updating (just) the key share. We also get the guarantee that the message remains hidden even if the attacker continually leaks on the key share and later gets the ciphertext share in full. Unfortunately, this does not suffice for three reasons. *Firstly,* we need a method for updating the ciphertext, which is not a property of CLR-PKE. *Secondly,* we need a new security property to guarantee that, even if the ciphertext and secret-key shares are both continually leaking at the same time, the shared message stays hidden. This property is strictly stronger, and significantly harder to analyze, than the security of CLR-PKE. *Thirdly,* the proof strategy of [6], [22] does not deal with leakage on key updates directly, but instead uses a generic "guessing" argument to allow for some small (logarithmic) leakage. Unfortunately, this argument does not apply to the case of sharing. In particular, security without leakage on updates does not seem to imply any guarantees if even 1 bit can leak during the update.

Therefore, our starting point will be the recent CLR-PKE scheme of [21], which provides a new proof strategy to argue about leakage of key updates directly. Although we borrow many high level ideas from this proof strategy, our first result is to significantly simplify the construction and the proof of [21], getting several other benefits along the way (mentioned above). Next, we show a natural method for updating the ciphertexts of the new scheme, analogously to the way that the secret keys are updated. Lastly, we carefully lay out a new proof strategy to argue that continual leakage on secret keys and ciphertexts keeps the message hidden. This proof strategy is significantly more involved then arguing CLR-PKE security alone, and involves moving from a game where every secret key correctly decrypts every ciphertext in every time period, to a game where this is never the case.

RELATION TO OTHER PRIMITIVES. It is useful to compare CLRS schemes to other primitives from the literature. Most obviously, standard secret sharing schemes [29] provide security when some subset of the shares are fully compromised while others are fully secure. In CLRS schemes, *all* shares leak and hence none are fully secure. The idea of updating shares to protect them against continual compromise was also considered in the context of *proactive secret sharing* [16]. However, the motivation there was to protect against a mobile adversary that corrupts different subsets of the shares in different time periods, while in our case *all* shares leak in all time periods. Another important connection is to the *leakage-resilient storage* scheme of [7]. This gives an information-theoretic solution for sharing a secret securely on two leaky devices/components in the *bounded* leakage model, where the overall amount of leakage on each share is bounded. The work of [10] extends this information theoretic solution to the continual leakage model, but requires that devices have access to some correlated randomness generated in a leak-free way (e.g. using leak-free hardware) and update their shares interactively. In contrast, we do not assume any leak-free hardware. Also, our updates are performed individually and we show that this comes at the necessary expense of having computational assumptions.

Related to the above model, prior (unpublished) work by [1] was the first to propose the two processor distributed setting for public key decryption, where the systems secret state is shared by both processors, and is subject to continual memory leakage attacks, where the attacker is restricted to leak from each of the processors share of the secret state separately. Their ultimate goal was the security of the public key encryption scheme rather than the maintenance of a particular secret, which is addressed by an interactive secret state refresh protocol in their work.

The prior works [19], [14] consider general compilers for executing arbitrary computations privately on leaky devices. Both works provide solutions in variants of the "only computation leaks information model", but require some additional leak-free hardware. Implicitly, these works also address the question of storing a value secretly on leaky

[4]This is optimal as otherwise the leaked predicate of an uncorrupted device could run the reconstruction procedure using the shares of all the corrupted devices and leak (say) the first bit of the shared message.

devices, since the state of the computation must be somehow stored consistently. However, the use of leak-free hardware in these solutions greatly simplifies the problem of storage and avoids virtually all of the challenges that we address in the current work. We believe that our work provides an important first step in the direction of building general compilers without any leak-free hardware, since the question of (just) securing storage must be addressed as a part of any solution to the larger question of securing computation.

ORGANIZATION. In this proceedings version, we only describe a simplified version of our scheme whose security can be proven under the DDH assumption in bilinear groups (this is also called the SXDH assumption; see below below). We then sketch the main ideas behind the proof. The rest of our results and complete proofs are deferred to the full version [9] of this work. In particular, there we describe a generalized variant of the scheme whose security holds under a progressively weaker class of assumptions called k-linear. We also generalize the scheme to arbitrary linear secret sharing between many parties, and show information theoretic impossibility.

2. NOTATION AND PRELIMINARIES

LINEAR ALGEBRA. Let $\mathbb{F}$ be a field. We denote *row* vectors with $\vec{v} \in \mathbb{F}^n$. If $\vec{v}_1, \ldots, \vec{v}_m \in \mathbb{F}^n$ are m vectors we let $\mathsf{span}(\vec{v}_1, \ldots, \vec{v}_m) \subseteq \mathbb{F}^n$ denote the linear space spanned by these vectors. We let $\langle \vec{v}, \vec{w} \rangle \stackrel{\text{def}}{=} \vec{v} \cdot \vec{w}^\top$ be the *dot product* of $\vec{v}, \vec{w} \in \mathbb{F}_q^n$. If $A \in \mathbb{F}^{n\times m}$ is a $n \times m$ matrix of scalars, we let $\mathsf{colspan}(A), \mathsf{rowspan}(A)$ denote the subspaces spanned by the columns and rows of A respectively. If $\mathcal{V} \subseteq \mathbb{F}^n$ is a subspace, we let $\mathcal{V}^\perp$ denote the *orthogonal space* of $\mathcal{V}$, defined by $\mathcal{V}^\perp \stackrel{\text{def}}{=} \{\ \vec{w} \in \mathbb{F}_q^n \ \mid\ \langle \vec{w}, \vec{v} \rangle = 0 \quad \forall \vec{v} \in \mathcal{V}\ \}$. We write $(\vec{v}_1, \ldots, \vec{v}_m)^\perp$ as shorthand for $\mathsf{span}(\vec{v}_1, \ldots, \vec{v}_m)^\perp$. We write $\mathcal{V} \perp \mathcal{W}$ if $\mathcal{V} \subseteq \mathcal{W}^\perp$ and therefore also $\mathcal{W} \subseteq \mathcal{V}^\perp$. We define the *kernel* of a matrix A to be $\mathsf{ker}(A) \stackrel{\text{def}}{=} \mathsf{rowspan}(A)^\perp$.

For integers d, n, m with $1 \le d \le \min(n, m)$, we use the notation $\mathsf{Rk}_d(\mathbb{F}_q^{n\times m})$ to denote the set of all rank d matrices in $\mathbb{F}_q^{n\times m}$. When $\mathcal{W} \subseteq \mathbb{F}_q^m$ is a subspace, we also use the notation $\mathsf{Rk}_d(\mathbb{F}_q^{n\times m} \mid \mathsf{row} \in \mathcal{W})$ to denote the set of rank d matrices in $\mathbb{F}_q^{n\times m}$ whose rows come from the subspace $\mathcal{W}$.

MATRIX-IN-THE-EXPONENT NOTATION. Let $\mathbb{G}$ be a group of prime order q generated by an element $\mathbf{g} \in \mathbb{G}$ and let $A \in \mathbb{F}_q^{n\times m}$ be a matrix. Then we use the notation $\mathbf{g}^A \in \mathbb{G}^{n\times m}$ to denote the matrix $\left(\mathbf{g}^A\right)_{i,j} \stackrel{\text{def}}{=} \mathbf{g}^{(A)_{i,j}}$ of group elements. Note that, given a matrix of group elements $\mathbf{g}^A \in \mathbb{G}^{n\times m}$ and a matrix $B \in \mathbb{F}_q^{m\times k}$ of "exponents", one can efficiently compute $\mathbf{g}^{AB}$. However, given $\mathbf{g}^A$ and $\mathbf{g}^B$ it is (generally) not feasible to efficiently compute $\mathbf{g}^{AB}$. On the other hand, if $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ are three groups of prime order q and $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is an efficient *bilinear map*, then, given $\mathbf{g}^A$ and $\mathbf{h}^B$ for generators $\mathbf{g} \in \mathbb{G}_1, \mathbf{h} \in \mathbb{G}_2$, one can efficiently compute $e(\mathbf{g}, \mathbf{h})^{AB}$ via $(e(\mathbf{g}, \mathbf{h})^{AB})_{i,j} = \prod_{k=1}^m e\left(\mathbf{g}^{A_{i,k}}, \mathbf{h}^{B_{k,j}}\right)$. We abuse notation and let $e(\mathbf{g}^A, \mathbf{h}^B) = e(\mathbf{g}, \mathbf{h})^{AB}$ denote this operation.

HARDNESS ASSUMPTIONS. Let $\mathcal{G}$ be a pairing generation algorithm $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, q, e, \mathbf{g}, \mathbf{h}) \leftarrow \mathcal{G}(1^\lambda)$, where $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ are descriptions of cyclic group of prime order q with generators $\mathbf{g} \in \mathbb{G}_1, \mathbf{h} \in \mathbb{G}_2$ and e is a description of an efficient bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. We say that the pairing is *symmetric* if $\mathbb{G}_1 = \mathbb{G}_2$ and *asymmetric* otherwise. We will rely on an assumption that we call the *k-rank hiding assumption*. This assumption was introduced by [24] and shown to be implied by the more common *k-linear assumption* [5], [17], [28]. The *k-rank hiding assumption* on the *left* group $\mathbb{G}_1$ states that for any $k \le i < j \le \min\{m, n\}$, it is computationally infeasible to distinguish rank i and rank j matrices in the exponent of $\mathbf{g}$:

$$\left(\mathsf{prms}, \mathbf{g}^X \;\middle|\; \begin{array}{c} \mathsf{prms} \leftarrow \mathcal{G}(1^\lambda) \\ X \stackrel{\$}{\leftarrow} \mathsf{Rk}_i(\mathbb{F}_q^{n\times m}) \end{array} \right) \stackrel{\text{comp}}{\approx} \left(\mathsf{prms}, \mathbf{g}^X \;\middle|\; \begin{array}{c} \mathsf{prms} \leftarrow \mathcal{G}(1^\lambda) \\ X \stackrel{\$}{\leftarrow} \mathsf{Rk}_j(\mathbb{F}_q^{n\times m}) \end{array} \right)$$

Similarly, we can make the k-rank hiding assumption on the *right* group $\mathbb{G}_2$, by replacing $\mathbf{g}$ with $\mathbf{h}$ in the above. We say that the k-rank hiding assumption holds for $\mathcal{G}$ if it holds for *both* the left and right groups. It is easy to see that the k-rank hiding assumption gets *weaker* as k increases. Therefore, the $k = 1$ version of the assumption is the strongest. In fact, when $k = 1$, this assumption is equivalent to 1-linear which is just DDH. Unfortunately, it is known that DDH *cannot* hold in symmetric pairings where $\mathbb{G}_1 = \mathbb{G}_2$. However, it is often reasonable to assume that DDH holds in *asymmetric pairings*, and this is also called the *external Diffie-Hellman assumption SXDH* [27], [5], [13], [30]. Since the SXDH assumption is fairly strong, it is sometimes preferable to use $k \ge 2$. The $(k = 2)$*-linear assumption*, also called *decisional linear*, is commonly believed to hold in many symmetric and asymmetric pairings.

3. DEFINITIONS

3.1. Continual-Leakage-Resilient Sharing (CLRS)

We now formally define the notion of a *continual-leakage-resilient sharing (CLRS)* scheme between two devices. The scheme has the following syntax:

ShareGen $(1^\lambda, \mathbf{msg}) \to (\mathsf{sh}_1, \mathsf{sh}_2)$ The share generation algorithm takes as input the security parameter λ and a secret message $\mathbf{msg}$. It outputs two *shares*, sh_1 and sh_2 respectively.

$\mathbf{Update}_b(\mathsf{sh}_b) \to \mathsf{sh}'_b$ **:** The *randomized* update algorithm takes the index b and the current version of the share sh_b and outputs an updated version sh'_b. We use the notation $\mathsf{Update}_b^i(\mathsf{sh}_b)$ to denote the operation of updating the share

sh_b successively i times in a row so that $\mathsf{Update}_b^0(\mathsf{sh}_b) := \mathsf{sh}_b, \mathsf{Update}_b^{(i+1)}(\mathsf{sh}_b) := \mathsf{Update}_b(\mathsf{Update}_b^i(\mathsf{sh}_b))$.

Reconstruct$(\mathsf{sh}_1, \mathsf{sh}_2) \to \mathbf{msg}$ **:** The reconstruction algorithm takes in some version of secret shares $\mathsf{sh}_1, \mathsf{sh}_2$ and it outputs the secret message $\mathbf{msg}$.

Correctness. We say that the scheme is *correct* if for any shares $(\mathsf{sh}_1, \mathsf{sh}_2) \leftarrow \mathsf{ShareGen}(1^\lambda, \mathbf{msg})$ and any sequence of $i \geq 0, j \geq 0$ updates resulting in $\mathsf{sh}_1' \leftarrow \mathsf{Update}_1^i(\mathsf{sh}_1)$, $\mathsf{sh}_2' \leftarrow \mathsf{Update}_2^j(\mathsf{sh}_2)$, we get $\mathsf{Reconstruct}(\mathsf{sh}_1', \mathsf{sh}_2') = \mathbf{msg}$. Note that i and j are arbitrary, and may not be equal.

Security. We define ℓ-CLR security as an interactive game between an attacker $\mathcal{A}$ and a challenger.

- The attacker chooses two messages: $\mathbf{msg}_0, \mathbf{msg}_1 \in \{0,1\}^*$ with $|\mathbf{msg}_0| = |\mathbf{msg}_1|$.
- The challenger chooses a bit $b \leftarrow \{0,1\}$ at random, runs $(\mathsf{sh}_1, \mathsf{sh}_2) \leftarrow \mathsf{ShareGen}(1^\lambda, \mathbf{msg}_b)$. The challenger also chooses randomness $rand_1, rand_2$ for the next update of the shares 1,2 respectively and sets $\mathsf{state}_1 := (\mathsf{sh}_1, rand_1), \mathsf{state}_2 := (\mathsf{sh}_2, rand_2)$. It initializes the counters $L_1 := 0, L_2 := 0$.
- The attacker $\mathcal{A}$ can adaptively make any number of the following types of queries to the challenger in any order of its choosing:

  Leakage Queries: The attacker specifies an efficient predicate $\mathsf{Leak} : \{0,1\}^* \to \{0,1\}$ and an index $i \in \{1,2\}$. If $L_i < \ell$ then the challenger responds with the value $\mathsf{Leak}(\mathsf{state}_i)$ and increases the counter $L_i := L_i + 1$. Else it responds with $\perp$.

  Update Queries: The attacker specifies an index $i \in \{1,2\}$. The challenger parses $\mathsf{state}_i = (\mathsf{sh}_i, rand_i)$ and computes the updated share $\mathsf{sh}_i' := \mathsf{Update}_i(\mathsf{sh}_i; rand_i)$ using randomness $rand_i$. It samples fresh randomness $rand_i'$ and sets $\mathsf{state}_i := (\mathsf{sh}_i', rand_i')$, $L_i := 0$.
- At any point in the game, the attacker $\mathcal{A}$ can output a guess $\tilde{b} \in \{0,1\}$. We say that $\mathcal{A}$ *wins* if its guess matches the choice of the challenger $\tilde{b} = b$.

We say that an ℓ-CLRS scheme is *secure* if for any PPT attacker $\mathcal{A}$ running in the above game, we have $|\Pr[\mathcal{A} \text{ wins }] - \frac{1}{2}| \leq negl(\lambda)$.

REMARKS ON THE DEFINITION. The inclusion of the update randomness $rand_i$ in the state of the device models leakage during the update process itself when this randomness is used. Note that we do not need to explicitly include the next share $\mathsf{sh}_i' = \mathsf{Update}_i(\mathsf{sh}_i; rand_i)$ in the state since it is already efficiently computable from sh_i and $rand_i$.

The given definition also already implies that the message remain hidden even if one of the shares is revealed *fully* at the end of the game (but no leakage on the other share is allowed afterwards). To see this, assume that at some point in the game, there is a distinguishing strategy D that uses a fully revealed share sh_i to break security. Then we could also just leak the single predicate $D(\mathsf{sh}_i)$ to break security.

3.2. CLRS-Friendly Encryption

We consider an approach of instantiating CLRS via a *public key encryption* scheme $(\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ having the usual syntax. Given any such encryption scheme, we can define a sharing scheme where the two shares are the secret key $\mathsf{sh}_1 = \mathsf{sk}$ and the ciphertext $\mathsf{sh}_2 = \mathsf{ct}$ respectively. Formally, we define:

ShareGen$(1^\lambda; \mathbf{msg})$: Sample $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$, $\mathsf{ct} \leftarrow \mathsf{Encrypt}_{\mathsf{pk}}(\mathbf{msg})$. Output $\mathsf{sh}_1 := \mathsf{sk}, \mathsf{sh}_2 := \mathsf{ct}$.

Reconstruct$(\mathsf{sh}_1, \mathsf{sh}_2)$: Parse $\mathsf{sh}_1 = \mathsf{sk}, \mathsf{sh}_2 = \mathsf{ct}$. Output $\mathbf{msg} = \mathsf{Decrypt}_{\mathsf{sk}}(\mathsf{ct})$.

We say that an encryption scheme is *updatable* if it comes with two additional (non-standard) procedures $\mathsf{sk}' \leftarrow \mathsf{SKUpdate}(\mathsf{sk})$, $\mathsf{ct}' \leftarrow \mathsf{CTUpdate}(\mathsf{ct})$ for updating the secret keys and ciphertexts respectively. These procedures can naturally be used to define updates for the corresponding sharing via $\mathsf{Update}_1(\mathsf{sh}_1) := \mathsf{SKUpdate}(\mathsf{sk})$, $\mathsf{Update}_2(\mathsf{sh}_2) := \mathsf{CTUpdate}(\mathsf{ct})$, where $\mathsf{sh}_1 = \mathsf{sk}, \mathsf{sh}_2 = \mathsf{ct}$. The above gives us a natural syntactical transformation from an updatable encryption scheme to a corresponding CLRS scheme. We say that an updatable encryption scheme is an *ℓ-CLRS-Friendly Encryption* if:

- The corresponding CLRS scheme satisfies *correctness*.
- The corresponding CLRS scheme satisfies a *strengthening* of *ℓ-CLRS security* where the attacker is first given the public key pk and then adaptively chooses the messages $\mathbf{msg}_1, \mathbf{msg}_2$.

REMARKS. We note that the additional functionality provided by CLRS-friendly encryption on top of a plain CLRS may be useful even in the context of sharing a secret between leaky devices. For example, we can imagine a system where one (continually leaky) *master device* stores a secret key share and we publish the corresponding public key. Then other devices can enter the system in an ad-hoc manner by just encrypting their data individually under the public key to establish a shared value with the master device (i.e. no communication is necessary to establish the sharing). The same secret-key share on the master device can be *reused* to share many different messages with many different devices.

As another advantage, CLRS-friendly encryption right away implies CLR-PKE schemes with continual leakage on the secret key and the updates, in the sense of [6], [21].

4. SCHEME DESCRIPTION

We now describe our construction of CLRS going through CLRS-Friendly Encryption. We give two encryption algorithms – a *simple* one which is sufficient for CLR-PKE where we update keys but not ciphertexts, and an *updatable* one which is needed for CLRS and CLRS-Friendly Encryption.

Let m, n, d be integer parameters with $n \geq d$. The scheme is defined as follows.

KeyGen$(1^\lambda) \to (\mathsf{pk}, \mathsf{sk})$ **:** Sample the description of a bilinear group $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, \mathbf{g}, \mathbf{h}, q) \leftarrow \mathcal{G}(1^\lambda)$. Choose $\vec{p}, \vec{w} \in \mathbb{F}_q^m$ at random subject to $\langle \vec{p}, \vec{w} \rangle = 0$ and set $\mathsf{prms} = ((\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, \mathbf{g}, \mathbf{h}, q), \mathbf{g}^{\vec{p}}, \mathbf{h}^{\vec{w}})$ to be the *public parameters* of the system. These parameters can then be re-used to create the public/secret keys of all future users. For convenience, we *implicitly* think of prms as a part of each public key pk and as an input to all of the other algorithms.

Choose $\vec{t} \xleftarrow{\$} \mathbb{F}_q^m$ and set $\mathsf{pk} := e(\ \mathbf{g}^{\vec{p}}\ ,\ \mathbf{h}^{\vec{t}^\top}\ ) = e(\mathbf{g}, \mathbf{h})^\alpha$ where $\alpha = \langle \vec{p}, \vec{t} \rangle$. Choose $\vec{r} = (r_1, \ldots, r_n) \xleftarrow{\$} \mathbb{F}_q^n$ and set $\mathsf{sk} := \mathbf{h}^S$, where S is the $n \times m$ matrix given by

$$S \ := \begin{bmatrix} r_1\vec{w} + \vec{t} \\ \cdots \\ r_n\vec{w} + \vec{t} \end{bmatrix} = \begin{bmatrix} \\ \vec{r}^\top \\ \\ \end{bmatrix} \begin{bmatrix} \quad \vec{w} \quad \end{bmatrix} + \begin{bmatrix} \\ \vec{1}^\top \\ \\ \end{bmatrix} \begin{bmatrix} \quad \vec{t} \quad \end{bmatrix}.$$

In other words, each row of S is chosen at random from the 1-dimensional affine subspace $\vec{t} + \mathsf{span}(\vec{w})$. (Note that $\mathbf{h}^S$ can be computed from the components $\mathbf{h}^{\vec{w}}$, $\vec{t}$, $\vec{r}$ without knowing $\vec{w}$.)

(Simple) SimplEncrypt$_{\mathsf{pk}}(\mathbf{msg}) \to \mathsf{ct}$ **:** To encrypt $\mathbf{msg} \in \mathbb{G}_T$ under $\mathsf{pk} = \mathbf{f} = e(\mathbf{g}, \mathbf{h})^\alpha$, choose $u \in \mathbb{F}_q$ and output: $\mathsf{ct} = (\mathbf{g}^{u\vec{p}}, \mathbf{f}^u \cdot \mathbf{msg})$.

(Updatable) Encrypt$_{\mathsf{pk}}(\mathbf{msg}) \to \mathsf{ct}$ **:** To encrypt $\mathbf{msg} \in \mathbb{G}_T$ under $\mathsf{pk} = \mathbf{f} = e(\mathbf{g}, \mathbf{h})^\alpha$, choose $\vec{u} = (u_1, \ldots, u_n) \xleftarrow{\$} \mathbb{F}_q^n$ and output $\mathsf{ct} = (\mathsf{ct}^{(1)}, \mathsf{ct}^{(2)})$ where:

$$\mathsf{ct}^{(1)} = \begin{bmatrix} \mathbf{g}^{u_1\vec{p}} \\ \cdots \\ \mathbf{g}^{u_n\vec{p}} \end{bmatrix} \quad , \quad \mathsf{ct}^{(2)} = \begin{bmatrix} \mathbf{f}^{u_1} \cdot \mathbf{msg} \\ \cdots \\ \mathbf{f}^{u_n} \cdot \mathbf{msg} \end{bmatrix}$$

Each row is an independent encryption of the (same) message $\mathbf{msg}$ using the *simple encryption* process. Equivalently, we can write the ciphertext as $\mathsf{ct}^{(1)} = \mathbf{g}^C, \quad \mathsf{ct}^{(2)} = e(\mathbf{g}, \mathbf{h})^{\vec{z}^\top}$ for:

$$\begin{aligned} C &= \begin{bmatrix} \\ \vec{u}^\top \\ \\ \end{bmatrix} \begin{bmatrix} \quad \vec{p} \quad \end{bmatrix} \\ \vec{z}^\top &= \begin{bmatrix} \\ \vec{u}^\top \\ \\ \end{bmatrix} \alpha + \begin{bmatrix} \\ \vec{1}^\top \\ \\ \end{bmatrix} \mu \\ &= \begin{bmatrix} \\ \quad C \quad \\ \\ \end{bmatrix} \begin{bmatrix} \\ t^\top \\ \\ \end{bmatrix} + \begin{bmatrix} \\ \vec{1}^\top \\ \\ \end{bmatrix} \mu \end{aligned}$$

where μ is given by $\mathbf{msg} = e(\mathbf{g}, \mathbf{h})^\mu$ and $\alpha = \langle \vec{p}, \vec{t} \rangle$.

Decrypt$_{\mathsf{sk}}(\mathsf{ct}) \to \mathbf{msg}$**:** To decrypt, we only need to look at the first rows of the secret key and the ciphertext matrices. Given the first row $\mathbf{h}^{\vec{s}}$ of the secret key $\mathsf{sk} = \mathbf{h}^S$, the first row $\mathbf{g}^{\vec{c}}$ of the ciphertext component $\mathsf{ct}^{(1)} = \mathbf{g}^C$, and the first scalar component $e(\mathbf{g}, \mathbf{h})^z$ of $\mathsf{ct}^{(2)} = e(\mathbf{g}, \mathbf{h})^{\vec{z}^\top}$, the decryption algorithm outputs: $\mathbf{msg} = e(\mathbf{g}, \mathbf{h})^z / e(\ \mathbf{g}^{\vec{c}}\ ,\ \mathbf{h}^{\vec{s}^\top}\ )$.

SKUpdate$(\mathsf{sk}) \to \mathsf{sk}'$ **:** Choose a random matrix $A' \xleftarrow{\$} \mathsf{Rk}_d(\mathbb{F}_q^{n \times n})$. Derive A by "rescaling" each row of A' so that its components sum up to 1. That is, set $(A)_{i,j} := (A')_{i,j} / (\sum_{k=1}^n (A')_{i,k})$, so that $A\vec{1}^\top = \vec{1}^\top$. If the current secret key is $\mathsf{sk} = \mathbf{h}^S$, output the updated key $\mathsf{sk}' := \mathbf{h}^{AS}$.

CTUpdate$(\mathsf{ct}) \to \mathsf{ct}'$ **:** Choose a random matrix $B' \xleftarrow{\$} \mathsf{Rk}_d(\mathbb{F}_q^{n \times n})$. Derive B by "rescaling" each row of B' so that its components sum up to 1. That is, set $(B)_{i,j} := (B')_{i,j} / (\sum_{k=1}^n (B')_{i,k})$, so $B\vec{1}^\top = \vec{1}^\top$. If the current ciphertext is $\mathsf{ct} = (\mathbf{g}^C, e(\mathbf{g}, \mathbf{h})^{\vec{z}^\top})$, output the updated ciphertext $\mathsf{ct}' := (\mathbf{g}^{BC}, e(\mathbf{g}, \mathbf{h})^{B\vec{z}^\top})$.

Theorem 4.1. *For any integers $m \geq 6, n \geq 3m - 6, d := n - m + 3$ the above scheme is an ℓ-CLRS-friendly encryption scheme under the SXDH assumption for any*

$$\ell = \min(m/6 - 1, n - 3m + 6)\log(q) - \omega(\log(\lambda)).$$

In the above theorem, the absolute leakage (ℓ) scales linearly as $\min(m, n - 3m)$ or $\log(q)$ grow. The *ratio* of leakage to share size is $\ell/(nm\log(q))$, and is maximized at $m = 7, n = 16$ to roughly $1/672$.

Corollary 4.2. *For any polynomial $\ell = \ell(\lambda)$, there exist ℓ-CLRS schemes under the SXDH assumption. Furthermore, ℓ is a* constant fraction *of the share size.*

Lastly, if we only care about encryption with *continual leakage on the secret key and update randomness*, then there is no need to update ciphertexts and we can use the "simple" encryption strategy.

Corollary 4.3. *For any m, n, d as above, the scheme* (KeyGen, SimplEncrypt, Decrypt, SKUpdate) *is an ℓ-CLR-Encryption with leakage-of-updates, under the SXDH assumption and for the same ℓ as above.*

CORRECTNESS. Let $(\mathsf{prms}, \mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$ and let $\mathsf{ct} = (\mathsf{ct}^{(1)}, \mathsf{ct}^{(2)}) \leftarrow \mathsf{Encrypt}_{\mathsf{pk}}(\mathbf{msg})$. Then we can write $\mathsf{sk} = \mathbf{g}^S$, $\mathsf{ct}^{(1)} = \mathbf{h}^C, \mathsf{ct}^{(2)} = e(\mathbf{g}, \mathbf{h})^{\vec{z}}$ for some values $S, C, \vec{z}$ satisfying:

$$S = W + \vec{1}^\top\vec{t}\ \ ,\ \ \vec{z}^\top = C\vec{t}^\top + \vec{1}^\top\mu \qquad (1)$$

with $\mathsf{rowspan}(W) \perp \mathsf{rowspan}(C)$ and μ given by $\mathbf{msg} = e(\mathbf{g}, \mathbf{h})^\mu$. First, we show that for any sk and ct having the above form, we get $\mathsf{Decrypt}_{\mathsf{sk}}(\mathsf{ct}) = \mathbf{msg}$. This is because decryption looks at the first row of $\mathsf{sk}, \mathsf{ct}^{(1)}, \mathsf{ct}^{(2)}$ respectively, which are of the form $\mathbf{h}^{\vec{s}}, \mathbf{g}^{\vec{c}}, e(\mathbf{g}, \mathbf{h})^z$ where $\vec{s} = \vec{w} + \vec{t}, z = \langle \vec{c}, \vec{t} \rangle + \mu$ for some vectors $\vec{w}, \vec{c}, \vec{t}$ with $\langle \vec{c}, \vec{w} \rangle = 0$. Therefore decryption correctly recovers:

$$\begin{aligned} e(\mathbf{g}, \mathbf{h})^z / e(\ \mathbf{g}^{\vec{c}}\ ,\ \mathbf{h}^{s^\top}\ ) &= e(\mathbf{g}, \mathbf{h})^{\langle \vec{c}, \vec{t} \rangle + \mu} / e(\mathbf{g}, \mathbf{h})^{\langle \vec{c}, \vec{w} + \vec{t} \rangle} \\ &= e(\mathbf{g}, \mathbf{h})^\mu = \mathbf{msg} \end{aligned}$$

Next we show, that updates preserve the key/ciphertext structure of equation (1). Assume that we update the secret key with the matrices $A_1, A_2, \ldots, A_i$ and the ciphertext with the matrices $B_1, B_2, \ldots, B_j$. Define $\bar{A} = A_i A_{i-1} \cdots A_1$, $\bar{B} = B_j B_{j-1} \cdots B_1$. Since the update matrices are "rescaled" we know that $\bar{A}\vec{1}^\top = \bar{B}\vec{1}^\top = \vec{1}^\top$. Therefore we can write the updated values as $\mathsf{sk}_i = \mathbf{g}^{\bar{A}S}$, $\mathsf{ct}_j^{(1)} = \mathbf{h}^{\bar{B}C}, \mathsf{ct}_j^{(2)} = e(\mathbf{g}, \mathbf{h})^{\bar{B}\vec{z}^\top}$ satisfying:

$$(\bar{A}S) = (\bar{A}W) + \vec{1}^\top \vec{t} \ \ , \ \ (\bar{B}\vec{z}^\top) = (\bar{B}C)\vec{t}^\top + \vec{1}^\top \mu$$

with $\mathsf{rowspan}(\bar{A}W) \perp \mathsf{rowspan}(\bar{B}C)$. So the structure of equation (1) is satisfied by the updated keys and ciphertexts and we get $\mathsf{Decrypt}_{\mathsf{sk}_i}(\mathsf{ct}_j) = \mathbf{msg}$.

5. Security Proof Overview

Our proof of security will follow by a hybrid argument. In the real security game, the original secret key and every updated version of it correctly decrypts the original ciphertext and every updated version of it. Our goal is to move to a modified game where none of the secret keys can correctly decrypt any of the ciphertexts, and in fact the message remains hidden even given these modified keys/ciphertexts in full. We do so by slowly modifying how the challenger chooses the initial key, ciphertext and the update matrices. In Section 5.1, we first introduce several alternate distributions for selecting keys and ciphertexts, some of which decrypt correctly and some don't. We also show how to select update matrices to modify the key/ciphertext type. In Section 5.2, we then lay out a careful hybrid argument proof strategy for moving between the various distributions.

5.1. Alternate Distributions for Keys, Ciphertexts, Updates

Assume that the vectors $\vec{p}, \vec{w}, \vec{t}$ are fixed, defining the public values $\mathbf{g}^{\vec{p}}, \mathbf{h}^{\vec{w}}, \mathsf{pk} = e(\mathbf{g}, \mathbf{h})^{\langle \vec{p}, \vec{t} \rangle}$. Fix $\vec{w}_1 := \vec{w}$, and let $(\vec{w}_1, \ldots, \vec{w}_{m-1})$ be some *basis* of $(\vec{p})^\perp$ and $(\vec{c}_1, \ldots, \vec{c}_{m-1})$ be some *basis* of $(\vec{w})^\perp$. We define various distributions of keys and ciphertexts relative to these bases.

Key Distributions. The secret key is always set to $\mathbf{h}^S$ for some $n \times m$ matrix S of the form $S =$

$$\begin{bmatrix} | & & | \\ \vec{r}_1^\top & \cdots & \vec{r}_i^\top \\ | & & | \end{bmatrix} \begin{bmatrix} - & \vec{w}_1 & - \\ & \cdots & \\ - & \vec{w}_i & - \end{bmatrix} + \begin{bmatrix} \\ \vec{1}^\top \\ \\ \end{bmatrix} \begin{bmatrix} & \vec{t} & \end{bmatrix} \qquad (2)$$

where $\vec{r}_1, \ldots, \vec{r}_i \in \mathbb{F}_q^n$ are chosen randomly. Equivalently, each of the n rows of S is chosen randomly from the affine space: $\mathsf{span}(\vec{w}_1, \ldots, \vec{w}_i) + \vec{t}$. The honest key generation algorithm uses $i = 1$ and we call these ***honest keys***. In addition, we define ***mid keys*** which are chosen with $i = 2$ and ***high keys*** which are chosen with $i = m - 1$. Notice that honest/mid/high keys all correctly decrypt honestly generated ciphertexts since $\mathsf{span}(\vec{w}_1, \ldots, \vec{w}_{m-1}) \perp \mathsf{span}(\vec{p})$.

Ciphertext Distributions. The encryption of the message $\mathbf{msg} = e(\mathbf{g}, \mathbf{h})^\mu \in \mathbb{G}_T$ is always set to $\mathsf{ct} =$

keys → ↓ ciphertexts	honest	mid	high
honest	Yes	Yes	Yes
low	Yes	Cor or Super-Corr	No
mid	Yes	Super-Corr	No
high	Yes	No	No

Figure 1. Do alternate keys correctly decrypt alternate ciphertexts?

$(\mathsf{ct}^{(1)}, \mathsf{ct}^{(2)})$ where $\mathsf{ct}^{(1)} = \mathbf{g}^C$ and $\mathsf{ct}^{(2)} = e(\mathbf{g}, \mathbf{h})^{\vec{z}^\top}$ with $\vec{z}^\top = C\vec{t}^\top + \vec{1}^\top \mu$. The second component $\mathsf{ct}^{(2)}$ can always be efficiently and deterministically computed from $\mathbf{g}^C$, given $\vec{t}$ and $\mathbf{msg}$, without knowing the exponents C, μ. The different ciphertext distributions only differ in how $\mathsf{ct}^{(1)} = \mathbf{g}^C$ is chosen.

For the ***honest ciphertexts***, we set $C = \vec{u}^\top \vec{p}$ for a uniformly random $\vec{u} \in \mathbb{F}_q^n$. That is, every row of C is chosen at random from the space $\mathsf{span}(\vec{p})$. In addition to the honest way of choosing C, we define three *additional* distributions on C given by:

$$C = \begin{bmatrix} | & & | \\ \vec{u}_1^\top & \cdots & \vec{u}_j^\top \\ | & & | \end{bmatrix} \begin{bmatrix} - & \vec{c}_1 & - \\ & \cdots & \\ - & \vec{c}_j & - \end{bmatrix} \qquad (3)$$

where $\vec{u}_1, \ldots, \vec{u}_j \in \mathbb{F}_q^n$ are chosen randomly. Equivalently the rows of the C are chosen randomly from the subspace: $\mathsf{span}(\vec{c}_1, \ldots, \vec{c}_j)$. When $j = 1$, we call these ***low ciphertexts***, when $j = 2$ we call these ***mid ciphertexts*** and when $j = (m - 1)$, we call these ***high ciphertexts***. Notice that honest/low/mid/high ciphertexts are all correctly decrypted by *honest secret keys* since $\mathsf{span}(\vec{w}) \perp \mathsf{span}(\vec{c}_1, \ldots, \vec{c}_{m-1})$.

Bases Correlations. By default, we choose the basis $(\vec{w}_1, \ldots, \vec{w}_{m-1})$ of the space $(\vec{p})^\perp$ and the basis $(\vec{c}_1, \ldots, \vec{c}_{m-1})$ of the space $(\vec{w})^\perp$ uniformly at random and independently subject to fixing $\vec{w}_1 := \vec{w}$. This is statistically close to choosing $\vec{w}_2, \ldots, \vec{w}_{m-1} \xleftarrow{\$} (\vec{p})^\perp$ and $\vec{c}_1, \ldots, \vec{c}_{m-1} \xleftarrow{\$} (\vec{w})^\perp$. We call this choice of bases ***uncorrelated***. We will also consider two alternate distributions. We say that the bases are ***correlated*** if we instead choose $\vec{c}_1 \xleftarrow{\$} (\vec{w}_1, \vec{w}_2)^\perp$ and all other vectors as before. We say that the bases are ***super-correlated*** if we instead choose $\vec{c}_1, \vec{c}_2 \xleftarrow{\$} (\vec{w}_1, \vec{w}_2)^\perp$ and all other vectors as before. If the key and ciphertext bases are correlated then mid keys correctly decrypt low ciphertexts and if they are super-correlated then mid keys correctly decrypt low and mid ciphertexts. The table in Figure 1 summarizes which types of secret keys can correctly decrypt which types of ciphertexts.

Programmed Updates. Honest *key updates* in period i are performed by choosing $A_i' \xleftarrow{\$} \mathsf{Rk}_d(\mathbb{F}_q^{n \times n})$ and rescaling its rows to get A_i. Let $D_i = A_i A_{i-1} \cdots A_1$ be the product of all key-update matrices up to and including period i (as a corner case, define D_0 to be the identity matrix). We say that the update A_i is *programmed to annihilate the vectors* $\vec{v}_1, \ldots, \vec{v}_j \in \mathbb{F}_q^n$ if we instead choose $A_i' \xleftarrow{\$} \mathsf{Rk}_d(\mathbb{F}_q^{n \times n} \mid \mathsf{row} \in \mathcal{V})$ where $\mathcal{V} =$

$(D_{i-1}\vec{v}_1^\top, \ldots, D_{i-1}\vec{v}_j^\top)^\perp$. In other words, a programmed update A_i has the vectors $\{D_{i-1}\vec{v}_\rho\}_{\rho=1}^j$ in its *kernel*. We define *programmed* ciphertext updates analogously.

By programming the update matrices, we can have updates which reduce the rank of the key/ciphertext matrices (e.g. reduce high keys to mid keys, or mid ciphetexts to low ciphertexts). Let us go through an example. Assume the initial key is high with $\mathsf{sk}_1 = \mathbf{g}^S$ for S given by equation (2), and the updates $A_1, \ldots, A_{i-1}$ are chosen honestly, A_i is programmed to annihilate the vectors $\vec{r}_3, \ldots, \vec{r}_{m-1}$ and A_{i+1} is programmed to annihilate $\vec{r}_2$. Then the corresponding secret keys $\mathsf{sk}_1, \ldots, \mathsf{sk}_i$ in periods 1 to i will be high keys, sk_{i+1} will be a mid key and sk_{i+2} will be an honest key. To see this, notice that the exponent of (e.g.) the key sk_{i+1} will follow equation (2) with $\vec{r}_j^\top$ replaced by $A_i A_{i-1} \cdots A_1 \vec{r}_j^\top$. Since A_i is programmed to annihilate the vectors $\vec{r}_j$ for $j \geq 3$, these values will be replaced by $\vec{\mathbf{0}}$ in period $i+1$.

5.2. The Hybrid Proof Strategy

We use a series of hybrids, where each step either takes advantage of the fact that the adversary is *computationally bounded* and cannot distinguish the rank of various matrices in the exponent, or of the fact that the adversary is *leakage bounded* and is only seing partial leakage on any key and ciphertext. When we use computational steps, we can even assume that the attacker gets *full* leakage and so we *cannot* modify any property of the game that could be efficiently tested given the keys and ciphertexts in full – in particular, we cannot modify whether any secret key correctly decrypts any ciphertext in any time period. When we use leakage steps, we can even assume that the attacker is computationally unbounded and hence we cannot modify the distribution of any individual key/ciphertext/update – but we can modify various *correlations* between them.[5]

The main strategy is to move from a game where all keys/ciphertexts/updates are *honest* to a game where the keys and ciphertexts no longer decrypt correctly. We can use *computational steps* to change the distribution of the initial key (honest/mid/high) or ciphertext (honest/low/mid/high) but *only* if they still decrypt correctly. For example, we can make the initial key and ciphertext both be *mid*, but only if the bases are *super-correlated*. We then want to use a *leakage step* to argue that the attacker cannot notice the correlation between the bases vectors $\vec{w}_2 \perp \vec{c}_2$. We rely on the fact that given partial independent leakage on two vectors, one cannot distinguish whether the vectors are orthogonal or not, which follows from inner-product being a good *two-source extractor*. Unfortunately, since leakage on keys and ciphertexts is continual, we cannot argue that leakage on the bases vectors $\vec{w}_2, \vec{c}_2$ is partial. To get around this, we carefully program our updates to reduce the rank of future keys/ciphertexts, so that the leakage on the vectors $\vec{w}_2, \vec{c}_2$ only occurs in a single key and ciphertext respectively. We carefully arrange the hybrids so as to make this type of argument on each key/ciphertext pair, one at a time.

HYBRID GAMES. A helpful pictorial representation of the main hybrid games appears in Figure 2. Our hybrid games, called *Game* (i, j) are all of the following type. For $i \geq 2$, the challenger chooses the initial key sk_1 as a high key, the first $i-2$ updates are honest (and hence the keys $\mathsf{sk}_1, \ldots, \mathsf{sk}_{i-1}$ are high keys), the update A_{i-1} is programmed to reduce the key to a mid key sk_i, and the update A_i is programmed to reduce the key to a honest key sk_{i+1}. The rest of the updates are honest and hence the keys $\mathsf{sk}_{i+1}, \mathsf{sk}_{i+2}, \ldots$ are honest. For the special case $i = 1$, the initial key sk_1 already starts out as a mid key and the first update A_1 reduces it to an honest key. For the special case $i = 0$, the initial key sk_1 is already an honest key. This description is mirrored by the ciphertexts. When $j \geq 2$, the initial ciphertext ct_1 is a high ciphertext, the first $j-2$ ciphertext updates are honest (and hence the ciphertexts $\mathsf{ct}_1, \ldots, \mathsf{ct}_{j-1}$ are high), the update B_{j-1} is programmed to reduce the ciphertext to a mid ct_j, and the update B_j is programmed to reduce the ciphertext to a *low* ciphertext ct_{j+1}. The rest of the updates are honest and hence the other ciphertexts stay *low*. For the special case $j = 1$, the initial ciphertext ct_1 is already mid and the first update B_1 reduces it to low. For the special case $j = 0$, the initial ciphertext ct_1 is already low.

We write *Game* i as short for *Game* $(i, j = 0)$. In *Game* (i, j) the ciphertext and key bases are *un*correlated. We also define analogous games: *GameCor* (i, j) where the bases are *correlated* and *GameSuperCor* (i, j) where the bases are *super-correlated*.

SEQUENCE OF HYBRIDS. See Figure 3 for a helpful overview of the sequence of hybrid games.

Our first step is to move from *Real Game* to *Game* 0 (i.e. $i = 0,\ j = 0$). The only difference in this step is that we change the initial ciphertext ct_1 from an *honest ciphertext* to a *low ciphertext*. This is a computational step and all secret keys still correctly decrypt all ciphertexts in the game.

Next, our goal is to keep moving from *Game* i to *Game* $i+1$. We call this the *outer loop* where we increment i. Unfortunately, we *cannot* just increment i in a single step since each such move changes sk_{i+1} from an honest key to a mid key and hence changes it from decrypting all of the low ciphertexts in the game to decrypting none of them. A single computational or leakage step cannot suffice.

Instead, we *can* move from *Game* i to *GameCor* $i+1$ in a single computational step. Even though the key sk_{i+1} changes from an honest key in *Game* i to a mid key in *GameCor* $i+1$, by making the bases correlated we ensure that it still correctly decrypts all of the low ciphertexts in

[5]This is a good way of viewing essentially all prior results in leakage resilient cryptography. Since we do not have computational assumptions that address leakage directly, we alternate between using computational assumptions that work even given full leakage and information theoretic steps that take advantage of the leakage being partial.

the game. Therefore, these games cannot be distinguished even given full leakage.

To move from *GameCor* $i+1$ to *Game* $i+1$, we first introduce an *inner loop* in which we slowly increment j. Starting with $j=0$, we move from *GameCor* $(i+1, j)$ to *GameSuperCor* $(i+1, j+1)$. This is a single computational step. Even though we change ct_{j+1} from a low ciphertext to a mid ciphertext, it is still correctly decrypted (only) by the mid and low keys in periods $i+1$ and later, since the bases are super-correlated. Therefore, these games cannot be distinguished even given full leakage. Finally, we use an information theoretic step to move from *GameSuperCor* $(i+1, j+1)$ to *GameCor* $(i+1, j+1)$. Here we are actually changing whether a single key sk_{i+1} correctly decrypts a single ciphertext ct_{j+1} (it does in *GameSuperCor* but not in *GameCor*). We use the fact that the adversary is leakage-bounded to argue that it cannot notice whether the bases are *correlated* or *super-correlated*. In particular, because the bases vectors $\vec{w}_2$ and $\vec{c}_2$ only occur in the single mid key sk_{i+1} and the single mid ciphertext ct_{j+1} respectively, the leakage on these vectors is bounded overall. We argue that such partial leakage *hides* whether $\vec{w}_2 \perp \vec{c}_2$, which determines if the bases are correlated or super-correlated.

Assume that the attacker makes at most q_{ct} update queries on the ciphertext and at most q_{sk} update queries on the secret key. By repeatedly increasing j in the inner loop, we move from *GameCor* $(i+1, 0)$ to *GameCor* $(i+1, q_{\mathsf{ct}}+1)$ where *all* of the ciphertexts that the attacker can leak on are *high* ciphertexts. Therefore the mid key sk_{i+1} does not decrypt any of them correctly (but all future honest keys still do). We now apply another computational step to move from *GameCor* $(i+1, q_{\mathsf{ct}}+1)$ to *Game* $i+1$ and therefore (finally) incrementing i in the outer loop. This step preserves all interactions between keys and ciphertexts and therefore these games cannot be distinguished even given full leakage. Lastly, by repeatedly increasing i in the outer loop, we can move from *Game* 0 to *Game* $q_{\mathsf{sk}}+1$ where all of the secret keys that the attacker can leak on are *high keys* and all of the ciphertexts are *low ciphertexts*. Therefore, in *Game* $q_{\mathsf{sk}}+1$ *none* of the keys correctly decrypts *any* of the ciphertexts. Hence we can argue that even an attacker that has full leakage in *Game* $q_{\mathsf{sk}}+1$ cannot learn any information about the shared/encrypted message $\mathbf{msg}$.

UNDER THE RUG. The above discussion is slightly oversimplified. The main issue is that the computational transitions, e.g. from *Game* i to *GameCor* $i+1$, are not computationally indistinguishable the way we defined the games. This is because in *Game* i the update matrix A_{i+1} is unlikely to annihilate any vectors (i.e. its kernel is unlikely to contain non-zero vectors from the span of the previous updates) while in *GameCor* $i+1$ it is programmed to annihilate vectors so as to reduce the dimension of the key. This can be efficiently tested given full leakage of the update matrices. Therefore, in the full proof, we define the games *Game*, *GameCor* and *GameSuperCor* slightly differently with some updates programmed to annihilate additional uniformly random vectors. With this modification, we can prove computational indistinguishability. We also need extra information theoretic steps to argue that the attacker cannot tell if updates are programmed to annihilate some random vectors, given limited leakage.

REFERENCES

[1] A. Akavia, S. Goldwasser, and C. Hazay, "Distributed public key schemes secure against continual leakage," 2010, unpublished Manuscript.

[2] A. Akavia, S. Goldwasser, and V. Vaikuntanathan, "Simultaneous hardcore bits and cryptography against memory attacks," in *Sixth Theory of Cryptography Conference — TCC 2007*, ser. Lecture Notes in Computer Science, O. Reingold, Ed., vol. 5444. Springer-Verlag, 2009.

[3] J. Alwen, Y. Dodis, and D. Wichs, "Leakage-resilient public-key cryptography in the bounded-retrieval model," in *Advances in Cryptology - CRYPTO 2009*, ser. LNCS, S. Halevi, Ed., vol. 5677. Springer-Verlag, 2009, pp. 36–54.

[4] ——, "Survey: Leakage resilience and the bounded retrieval model," in *ICITS*, 2009, pp. 1–18.

[5] D. Boneh, X. Boyen, and H. Shacham, "Short group signatures," in *CRYPTO*, 2004, pp. 41–55.

[6] Z. Brakerski, J. Katz, Y. Kalai, and V. Vaikuntanathan, "Overcomeing the hole in the bucket: Public-key cryptography against resilient to continual memory leakage," in *FOCS*. Las Vegas, NV, USA: IEEE, Oct. 23–26 2010, pp. 501–510.

[7] F. Davì, S. Dziembowski, and D. Venturi, "Leakage-resilient storage," in *SCN*, ser. Lecture Notes in Computer Science, J. A. Garay and R. D. Prisco, Eds., vol. 6280. Springer, 2010, pp. 121–137.

[8] Y. Dodis, K. Haralambiev, A. López-Alt, and D. Wichs, "Cryptography against continuous memory attacks," in *FOCS*. Las Vegas, NV, USA: IEEE, Oct. 23–26 2010, pp. 511–520.

[9] Y. Dodis, A. Lewko, B. Waters, and D. Wichs, "Storing secrets on continually leaky devices," Cryptology ePrint Archive, Report 2011/369, 2011, http://eprint.iacr.org/.

[10] S. Dziembowski and S. Faust, "Leakage-resilient cryptography from the inner-product extractor," 2011, manuscript in submission.

[11] S. Dziembowski and K. Pietrzak, "Leakage-resilient cryptography," in *49th Symposium on Foundations of Computer Science*. Philadelphia, PA, USA: IEEE Computer Society, Oct. 25–28 2008, pp. 293–302.

[12] ECRYPT, "Side channel cryptanalysis lounge," last accessed: August 26, 2009. http://www.emsec.rub.de/research/projects/sclounge/.

[13] S. D. Galbraith and V. Rotger, "Easy decision-diffie-hellman groups," *LMS Journal of Computation and Mathematics*, vol. 7, p. 2004, 2004.

[14] S. Goldwasser and G. N. Rothblum, "Securing computation against continuous leakage," in *CRYPTO*, ser. Lecture Notes in Computer Science, T. Rabin, Ed., vol. 6223. Springer, 2010, pp. 59–79.

[15] S. Halevi, Ed., *Advances in Cryptology - CRYPTO 2009*, ser. LNCS, vol. 5677. Springer-Verlag, 2009.

[16] A. Herzberg, S. Jarecki, H. Krawczyk, and M. Yung, “Proactive secret sharing or: How to cope with perpetual leakage,” in *CRYPTO*, ser. LNCS, D. Coppersmith, Ed., vol. 963. Springer-Verlag, 27–31 Aug. 1995, pp. 339–352.

[17] D. Hofheinz and E. Kiltz, “Secure hybrid encryption from weakened key encapsulation,” in *CRYPTO*, 2007, pp. 553–571.

[18] *51th Symposium on Foundations of Computer Science*. Las Vegas, NV, USA: IEEE, Oct. 23–26 2010.

[19] A. Juma and Y. Vahlis, “Protecting cryptographic keys against continual leakage,” in *CRYPTO*, ser. Lecture Notes in Computer Science, T. Rabin, Ed., vol. 6223. Springer, 2010, pp. 41–58.

[20] J. Katz and V. Vaikuntanathan, “Signature schemes with bounded leakage resilience,” in *Advances in Cryptology—ASIACRYPT 2009*, ser. LNCS, M. Matsui, Ed. Springer-Verlag, 2009, to Appear.

[21] A. B. Lewko, M. Lewko, and B. Waters, “How to leak on key updates,” in *STOC*, 2011, to Appear.

[22] A. B. Lewko, Y. Rouselakis, and B. Waters, “Achieving leakage resilience through dual system encryption,” in *TCC*, 2011, pp. 70–88.

[23] S. Micali and L. Reyzin, “Physically observable cryptography (extended abstract),” in *First Theory of Cryptography Conference — TCC 2004*, ser. LNCS, M. Naor, Ed., vol. 2951. Springer-Verlag, Feb. 19–21 2004, pp. 278–296.

[24] M. Naor and G. Segev, “Public-key cryptosystems resilient to key leakage,” in *Advances in Cryptology - CRYPTO 2009*, ser. LNCS, S. Halevi, Ed., vol. 5677. Springer-Verlag, 2009, pp. 18–35.

[25] T. Rabin, Ed., *Advances in Cryptology - CRYPTO 2010, 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings*, ser. Lecture Notes in Computer Science, vol. 6223. Springer, 2010.

[26] Reliable Computing Laboratory, Boston University, “Side channel attacks database,” http://www.sidechannelattacks.com, last accessed: August 26, 2009.

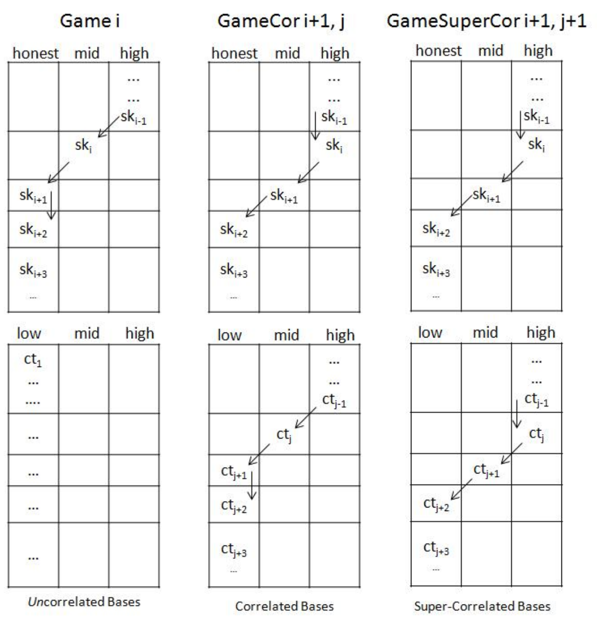
[27] M. Scott, “Authenticated id-based key exchange and remote log-in with simple token and pin number,” Cryptology ePrint Archive, Report 2002/164, 2002, http://eprint.iacr.org/.

[28] H. Shacham, “A cramer-shoup encryption scheme from the linear assumption and from progressively weaker linear variants,” 2007, cryptology ePrint Archive, Report 2007/074. [Online]. Available: http://eprint.iacr.org/2007/074

[29] A. Shamir, “How to share a secret,” *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[30] E. R. Verheul, “Evidence that xtr is more secure than supersingular elliptic curve cryptosystems,” *J. Cryptology*, vol. 17, no. 4, pp. 277–296, 2004.

APPENDIX



The arrows $\downarrow$ indicate *programmed updates.*

Figure 2. An Overview of the Main Hybrid Games

```
Real $\overset{\text{comp}}{\approx}$ Game 0.
For $i \in \{0, \dots, q_{\mathsf{sk}}\}$ :
.  Game $i$ $\overset{\text{comp}}{\approx}$ GameCor $(i+1, 0)$.
.  For $j \in \{0, \dots, q_{\mathsf{ct}}\}$
.    GameCor $(i+1, j)$ $\overset{\text{comp}}{\approx}$ GameSuperCor $(i+1, j+1)$
.                     $\overset{\text{stat}}{\approx}$ GameCor $(i+1, j+1)$.
.  GameCor $(i+1, q_{\mathsf{ct}}+1)$ $\overset{\text{comp}}{\approx}$ Game $i+1$.
Game $q_{\mathsf{sk}}+1$ $\overset{\text{comp}}{\approx}$ GameFinal.
```

Figure 3. Sequence of Hybrids: $Real \overset{\text{comp}}{\approx} GameFinal$.