Storing Secrets on Continually Leaky Devices

Yevgeniy Dodis* Allison Lewko † Brent Waters ‡ Daniel Wichs §

August 16, 2011

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.

*New York University. `dodis@cs.nyu.edu`
†University of Texas, Austin. `alewko@cs.utexas.edu`
‡University of Texas, Austin. `bwaters@cs.utexas.edu`
§New York University. `wichs@cs.nyu.edu`

Contents

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. [Koc96, KJJ99, QS01, AARR02, QK02, BE03, Rel, ECR] 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 formally modeling leakage and constructing cryptographic primitives that remain provably secure even in the presence of well defined classes of leakage attacks. Several different proposed models of leakage have emerged, with an emphasis on capturing large general classes of attacks. 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 [AGV09] 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. [BKKV10] and Dodis et al. [DHLW10], 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 [AGV09, NS09, ADW09a, KV09] and the continual-leakage model [BKKV10, DHLW10, LRW11, LLW11] construct encryption, signature and related primitives.

We also briefly mention an alternative model called the *only computation leaks information model*, which was introduced by Micali and Reyzin [MR04] and studied in [DP08, Pie09, FKPR10, JV10, GR10]. 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] 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 [ADW09b] for an overview.

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 [DP08] 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 (ℓ-CLRS).*

We mention that 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 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

[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.

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.

We also extend our 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)* [BKKV10, LRW11, LLW11], 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 [LLW11] 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 [LLW11], 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 [BKKV10, LRW11] 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 [LLW11], 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 [LLW11], 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 [Sha79] 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* [HJKY95]. 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

[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.

leakage-resilient storage scheme of [DDV10]. 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 [DF11] 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 [AGH10] 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.

Lastly, we mention the prior works [JV10, GR10], which 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 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. We define our notation and hardness assumptions in Section 2. In Section 3 we define CLRS schemes and in Section 4 we describe our (simplest) construction under a fairly strong assumption known as 1-linear or SXDH. In Section 5 we give an overview of the proof. The full proof is deferred to Appendix C. In Appendix D we present a generalized (but more complicated) construction under progressively weaker assumptions, known as the *k-linear assumptions*, for arbitrary $k \geq 1$. In Appendix E, we define and construct an extension to general access structures realizable by linear secret sharing schemes over many devices. Lastly, in Appendix F, we show *impossibility* for information-theoretic CLRS schemes.

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 $\mathcal{V}^\perp \stackrel{\text{def}}{=} \{ \vec{w} \in \mathbb{F}_q^n \mid \langle \vec{w}, \vec{v} \rangle = 0 \ \ \forall \vec{v} \in \mathcal{V} \}$. 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 $1 \leq d \leq \min(n, m)$, 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 [NS09] and shown to be implied by the more common *k-linear assumption* [BBS04, HK07, Sha07]. The *k-rank hiding assumption* on the *left* group $\mathbb{G}_1$ states that for any $k \le i < j \le \min\{m, n\}$ we cannot distinguish rank i and j matrices in the exponent of $\mathbf{g}$:

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

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* [Sco02, BBS04, GR04, Ver04]. 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.

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 $i \ge 0, j \ge 0$ 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 are not required to 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 also 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:

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

$\mathsf{Reconstruct}(\mathsf{sh}_1, \mathsf{sh}_2)$: 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 $\mathbf{msg}_1, \mathbf{msg}_2$.
  (*i.e. The challenger first chooses* $(\mathsf{pk}, \mathsf{sh}_1 = \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$ *and gives* pk *to the attacker, who chooses* $\mathbf{msg}_1, \mathbf{msg}_2$. *The challenger then generates* $\mathsf{sh}_2 \leftarrow \mathsf{Encrypt}_{\mathsf{pk}}(\mathbf{msg}_b)$ *and the game continues as before.*)

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 [BKKV10, LLW11].

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$(1^\lambda) \rightarrow (\mathsf{pk}, \mathsf{sk})$ **:** Sample $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, \mathbf{g}, \mathbf{h}, q) \leftarrow \mathcal{G}(1^\lambda)$ to be the description of a bilinear group of prime order q, with an efficient pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$, and generators $\mathbf{g} \in \mathbb{G}_1, \mathbf{h} \in \mathbb{G}_2$.

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$_{\mathsf{pk}}$**(msg)** $\rightarrow \mathsf{ct}$ **:** 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$_{\mathsf{pk}}$**(msg)** $\rightarrow \mathsf{ct}$ **:** 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 $\mathsf{ct}^{(1)} = \mathbf{g}^C$, $\mathsf{ct}^{(2)} = e(\mathbf{g}, \mathbf{h})^{\vec{z}^\top}$ for:

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

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

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 $S, C, \vec{z}$ and $W, \vec{t}$ satisfying:

$$S = W + \vec{1}^\top\vec{t}\ \ ,\ \ \vec{z}^\top = C\vec{t}^\top + \vec{1}^\top\mu \quad : \quad \mathsf{rowspan}(W) \perp \mathsf{rowspan}(C) \tag{1}$$

with μ given by $\mathbf{msg} = e(\mathbf{g},\mathbf{h})^\mu$.

First, we show that for any sk and ct satisfying equation (1), 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:

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

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 $\bar{A} = A_iA_{i-1}\cdots A_1$, $\bar{B} = B_jB_{j-1}\cdots B_1$. 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:

$$(\bar{A}S) = (\bar{A}W) + \vec{1}^\top\vec{t}\ \ ,\ \ (\bar{B}\vec{z}^\top) = (\bar{B}C)\vec{t}^\top + \vec{1}^\top\mu \quad : \quad \mathsf{rowspan}(\bar{A}W) \perp \mathsf{rowspan}(\bar{B}C).$$

So 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 and 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

$$S = \begin{bmatrix} | & & | \\ \vec{r}_1^\top & \cdots & \vec{r}_i^\top \\ | & & | \end{bmatrix} \begin{bmatrix} - & \vec{w}_1 & - \\ & \cdots & \\ - & \vec{w}_i & - \end{bmatrix} + \begin{bmatrix} \\ \vec{1}^\top \\ \\ \end{bmatrix} \begin{bmatrix} & \vec{t} & \end{bmatrix} \tag{2}$$

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 $\mathsf{ct} = (\mathsf{ct}^{(1)}, \mathsf{ct}^{(2)})$ 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:

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

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 $\vec{c}_1, \ldots, \vec{c}_{m-1} \xleftarrow{\$} (\vec{w})^\perp$ (Lemma B.1, part II). We call this choice of bases ***uncorrelated***. We will also consider two alternate distributions. We say

keys $\rightarrow$ $\downarrow$ ciphertexts	honest	mid	high
honest	Yes	Yes	Yes
low	Yes	If Correlated or Super-Correlated	No
mid	Yes	If Super-Correlated	No
high	Yes	No	No

Figure 1: Do alternate keys correctly decrypt alternate ciphertexts?

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, which may not be testable given just partial leakage on the individual values.[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 (Lemma B.9) that given partial

[5]This is a good way of viewing essentially all prior results in leakage resilient cryptography. Since we do not have computational assumptions that allow us to reason about leakage directly, we alternate between using computational steps that work even in the presence of unrestricted leakage and information theoretic steps that take advantage of the leakage being 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 *Game-SuperCor* (i, j) where the bases are *super-correlated*.

Sequence of Steps. A helpful table describing the sequence of hybrid arguments appears in Figure 3. 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 the game. Therefore, these games cannot be distinguished even given full leakage.

(*Now, we might be tempted to make an information theoretic step that moves us from* GameCor $i+1$ *to* Game $i+1$, *by arguing that a leakage-bounded attacker cannot tell if the key/ciphertext bases are correlated. Indeed, the leakage on the secret-key basis vector $\vec{w}_2$ is bounded overall, as this vector only occurs in the single mid key sk_{i+1}. Unfortunately, the leakage on the ciphertext basis vector $\vec{c}_1$ is* not *bounded overall as it occurs in every single ciphertext, and so a computationally unbounded attacker can learn* $\mathsf{span}(\vec{c}_1)$ *in full and test if $\vec{w}_2 \perp \vec{c}_1$. On the other hand, if we only cared about CLR encryption and* not *secret sharing, then the leakage on the secret key occurs before any information about the ciphertext is seen, and therefore the above argument would be legitimate. Indeed, this matches the high-level structure of the proof of security for the CLR Encryption scheme of [LLW11].*)

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 **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.

6 Conclusions

In this work, we show how to store secret values securely on device(s) that continually leak information about their internal state. We can view of our result as applying to either many devices, each of which is leaking individually in the continual leakage model, or to a single device which is continually leaking a restricted class of predicates that can only access a single memory component at a time.

One interesting question, under the latter view, is to consider different restricted classes of predicates. For example, we can look at predicates of the entire state that are computable in some fixed polynomial time, smaller than the complexity of reconstructing the secret. The work of [DDV10] considers several such classes in the bounded leakage model, where the overall number of leaked predicates is bounded.

Another interesting question is whether sharing schemes between two or more devices in the continual leakage model can be achieved *information theoretically*, if the devices are allowed to run an interactive protocol to update their shares. Our information-theoretic impossibility result only rules out the case where updates are performed by each device individually without any communication.

Yet another question would be to improve the fraction of the share that can leak from a small constant achieved in this work to larger constants, ideally approaching 1. At the very least, we would need to rely on better two-source extractors than the inner product to beat $\frac{1}{2}$. This would seem to require fundamentally new techniques.

Lastly, an important open question is to design methods for not only *storing* secrets securely but also *computing* on secrets securely in the presence of continual leakage. This question was studied in [ISW03, FRR+10, GR10, JV10], but solutions only exist in restrictive models and/or using some secure leak-free hardware. One elegant solution would be to design *secure computation protocols* where two (or more) continually leaky devices can securely compute functions of a secret value that is shared between them under our CLRS scheme.

References

[AARR02] Dakshi Agrawal, Bruce Archambeault, Josyula R. Rao, and Pankaj Rohatgi. The EM side-channel(s). In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *CHES*, volume 2523 of *Lecture Notes in Computer Science*, pages 29–45. Springer, August 13-15 2002.

[ADW09a] Joël Alwen, Yevgeniy Dodis, and Daniel Wichs. Leakage-resilient public-key cryptography in the bounded-retrieval model. In Halevi [Hal09], pages 36–54.

[ADW09b] Joël Alwen, Yevgeniy Dodis, and Daniel Wichs. Survey: Leakage resilience and the bounded retrieval model. In *ICITS*, pages 1–18, 2009.

[AGH10] Adi Akavia, Shafi Goldwasser, and Carmit Hazay. Distributed public key schemes secure against continual leakage. Unpublished Manuscript, 2010.

[AGV09] Adi Akavia, Shafi Goldwasser, and Vinod Vaikuntanathan. Simultaneous hardcore bits and cryptography against memory attacks. In Omer Reingold, editor, *Sixth Theory of Cryptography Conference — TCC 2007*, volume 5444 of *Lecture Notes in Computer Science*. Springer-Verlag, 2009.

[BBS04] Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures. In *CRYPTO*, pages 41–55, 2004.

[BE03] Hagai Bar-El. Known attacks against smartcards, 2003. last accessed: August 26, 2009. `http://www.hbarel.com/publications/Known_Attacks_Against_Smartcards.pdf`.

[Bei96] A. Beimel. PhD thesis, Israel Institute of Technology, Technion, Haifa, Israel, 1996.

[BKKV10] Zvika Brakerski, Jonathan Katz, Yael Kalai, and Vinod Vaikuntanathan. Overcomeing the hole in the bucket: Public-key cryptography against resilient to continual memory leakage. In *FOCS* [IEE10], pages 501–510.

[CG88] Benny Chor and Oded Goldreich. Unbiased bits from sources of weak randomness and probabilistic communication complexity. *SIAM Journal on Computing*, 17(2):230–261, 1988.

[DDV10] Francesco Davì, Stefan Dziembowski, and Daniele Venturi. Leakage-resilient storage. In Juan A. Garay and Roberto De Prisco, editors, *SCN*, volume 6280 of *Lecture Notes in Computer Science*, pages 121–137. Springer, 2010.

[DF11] Stefan Dziembowski and Sebastian Faust. Leakage-resilient cryptography from the inner-product extractor, 2011. Manuscript in submission.

[DHLW10] Yevgeniy Dodis, Kristiyan Haralambiev, Adriana López-Alt, and Daniel Wichs. Cryptography against continuous memory attacks. In *FOCS* [IEE10], pages 511–520.

[DORS08] Yevgeniy Dodis, Rafail Ostrovsky, Leonid Reyzin, and Adam Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. *SIAM Journal on Computing*, 38(1):97–139, 2008.

[DP08] Stefan Dziembowski and Krzysztof Pietrzak. Leakage-resilient cryptography. In *49th Symposium on Foundations of Computer Science*, pages 293–302, Philadelphia, PA, USA, October 25–28 2008. IEEE Computer Society.

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

[FKPR10] Sebastian Faust, Eike Kiltz, Krzysztof Pietrzak, and Guy N. Rothblum. Leakage-resilient signatures. In *TCC*, pages 343–360, 2010.

[FRR+10] Sebastian Faust, Tal Rabin, Leonid Reyzin, Eran Tromer, and Vinod Vaikuntanathan. Protecting circuits from leakage: the computationally-bounded and noisy cases. In Henri Gilbert, editor, *EUROCRYPT*, volume 6110 of *Lecture Notes in Computer Science*, pages 135–156. Springer, 2010.

[GR04] Steven D. Galbraith and Victor Rotger. Easy decision-diffie-hellman groups. *LMS Journal of Computation and Mathematics*, 7:2004, 2004.

[GR10] Shafi Goldwasser and Guy N. Rothblum. Securing computation against continuous leakage. In Rabin [Rab10], pages 59–79.

[Hal09] Shai Halevi, editor. *Advances in Cryptology - CRYPTO 2009*, volume 5677 of *LNCS*. Springer-Verlag, 2009.

[HJKY95] Amir Herzberg, Stanislaw Jarecki, Hugo Krawczyk, and Moti Yung. Proactive secret sharing or: How to cope with perpetual leakage. In Don Coppersmith, editor, *CRYPTO*, volume 963 of *LNCS*, pages 339–352. Springer-Verlag, 27–31 August 1995.

[HK07] Dennis Hofheinz and Eike Kiltz. Secure hybrid encryption from weakened key encapsulation. In *CRYPTO*, pages 553–571, 2007.

[IEE10] IEEE. *51th Symposium on Foundations of Computer Science*, Las Vegas, NV, USA, October 23–26 2010.

[ISW03] Yuval Ishai, Amit Sahai, and David Wagner. Private circuits: Securing hardware against probing attacks. In Dan Boneh, editor, *Advances in Cryptology—CRYPTO 2003*, volume 2729 of *LNCS*. Springer-Verlag, 2003.

[JV10] Ali Juma and Yevgeniy Vahlis. Protecting cryptographic keys against continual leakage. In Rabin [Rab10], pages 41–58.

[KJJ99] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael Wiener, editor, *Advances in Cryptology—CRYPTO '99*, volume 1666 of *LNCS*, pages 388–397. Springer-Verlag, 15–19 August 1999.

[Koc96] Paul Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In Neal Koblitz, editor, *Advances in Cryptology—CRYPTO '96*, volume 1109 of *LNCS*, pages 104–113. Springer-Verlag, 18–22 August 1996.

[KV09] Jonathan Katz and Vinod Vaikuntanathan. Signature schemes with bounded leakage resilience. In Mitsuru Matsui, editor, *Advances in Cryptology—ASIACRYPT 2009*, LNCS. Springer-Verlag, 2009. To Appear.

[LLTT05] Chia-Jung Lee, Chi-Jen Lu, Shi-Chun Tsai, and Wen-Guey Tzeng. Extracting randomness from multiple independent sources. *IEEE Transactions on Information Theory*, 51(6):2224–2227, 2005.

[LLW11] Allison B. Lewko, Mark Lewko, and Brent Waters. How to leak on key updates. In *STOC*, 2011. To Appear.

[LRW11] Allison B. Lewko, Yannis Rouselakis, and Brent Waters. Achieving leakage resilience through dual system encryption. In *TCC*, pages 70–88, 2011.

[MR04] Silvio Micali and Leonid Reyzin. Physically observable cryptography (extended abstract). In Moni Naor, editor, *First Theory of Cryptography Conference — TCC 2004*, volume 2951 of *LNCS*, pages 278–296. Springer-Verlag, February 19–21 2004.

[NS09] Moni Naor and Gil Segev. Public-key cryptosystems resilient to key leakage. In Halevi [Hal09], pages 18–35.

[NZ96] Noam Nisan and David Zuckerman. Randomness is linear in space. *Journal of Computer and System Sciences*, 52(1):43–53, 1996.

[Pie09] Krzysztof Pietrzak. A leakage-resilient mode of operation. In Antoine Joux, editor, *Advances in Cryptology - EUROCRYPT 2009*, volume 5479 of *LNCS*, pages 462–482. Springer-Verlag, 2009.

[QK02] Jean-Jaques Quisquater and François Koene. Side channel attacks: State of the art, October 2002. `http://www.ipa.go.jp/security/enc/CRYPTREC/fy15/doc/1047_Side_Channel_report.pdf`, last accessed: August 26, 2009.

[QS01] Jean-Jacques Quisquater and David Samyde. Electromagnetic analysis (ema): Measures and counter-measures for smart cards. In Isabelle Attali and Thomas P. Jensen, editors, *E-smart*, volume 2140 of *LNCS*, pages 200–210. Springer-Verlag, September 19-21 2001.

[Rab10] Tal Rabin, editor. *Advances in Cryptology - CRYPTO 2010, 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings*, volume 6223 of *Lecture Notes in Computer Science*. Springer, 2010.

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

[Sco02] Mike 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/`.

[Sha79] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[Sha07] Hovav Shacham. A cramer-shoup encryption scheme from the linear assumption and from progressively weaker linear variants, 2007. Cryptology ePrint Archive, Report 2007/074.

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

A The Proof in Pictures

Figure 2: An Overview of the Main Hybrid Games

Real $\stackrel{\text{comp}}{\approx}$ Game 0.

For $i \in \{0, \ldots, q_{\mathsf{sk}}\}$:

Game i $\stackrel{\text{comp}}{\approx}$ GameCor $(i+1, 0)$.

For $j \in \{0, \ldots, q_{\mathsf{ct}}\}$: GameCor $(i+1, j)$ $\stackrel{\text{comp}}{\approx}$ GameSuperCor $(i+1, j+1)$ $\stackrel{\text{stat}}{\approx}$ GameCor $(i+1, j+1)$.

GameCor $(i+1, q_{\mathsf{ct}}+1)$ $\stackrel{\text{comp}}{\approx}$ Game $i+1$.

Game $q_{\mathsf{sk}}+1$ $\stackrel{\text{comp}}{\approx}$ GameFinal.

Figure 3: Sequence of Hybrid Arguments Showing $Real \stackrel{\text{comp}}{\approx} GameFinal$.

B Background Lemmas

Random Matrices. We first prove two simple properties of random matrices.

Lemma B.1. *Assume q is super-polynomial in the security parameter.*
(I) *For $n \geq m$ the uniform distributions over $\mathsf{Rk}_m(\mathbb{F}_q^{n\times m})$ and $\mathbb{F}_q^{n\times m}$ are statistically indistinguishable.*
(II) *For $n \geq d$ and $\mathcal{W} \subseteq \mathbb{F}_q^m$ a subspace of dimension d, the uniform distributions over $\mathsf{Rk}_d(\mathbb{F}_q^{n\times m} \mid \mathsf{row} \in \mathcal{W})$ and $\mathcal{W}^n$ (seen as n rows) are statistically indistinguishable.*

Proof. Notice that (II) implies (I) with $d = m$ and $\mathcal{W} = \mathbb{F}_q^m$. The statistical distance between the uniform distributions over $\mathsf{Rk}_d(\mathbb{F}_q^{n\times m} \mid \mathsf{row} \in \mathcal{W})$ and $\mathcal{W}^n$ is just the probability that n samples from $\mathcal{W}$ span a sub-space of dimension $< d$. Think of choosing the rows from $\mathcal{W}$ one-by-one. Then the probability that the ith sample falls into the subspace of the previous ones is at most q^{i-1}/q^d. By union-bound, the probability of this happening in the first d samples is upper-bounded by $\sum_{i=1}^{d} q^{i-1}/q^d \leq 2/q$. □

Lemma B.2. *Let d, n, m be integers with $\min(n, m) \geq d \geq 1$.*
(I) *The uniform distribution over $\mathsf{Rk}_d(\mathbb{F}_q^{n\times m})$ is equivalent to sampling $C \xleftarrow{\$} \mathsf{Rk}_d(\mathbb{F}_q^{n\times d})$, $R \xleftarrow{\$} \mathsf{Rk}_d(\mathbb{F}_q^{d\times m})$ and outputting $A = CR$. Notice $\mathsf{colspan}(A) = \mathsf{colspan}(C), \mathsf{rowspan}(A) = \mathsf{rowspan}(R)$.*
(II) *If $\mathcal{W} \subseteq \mathbb{F}_q^m$ is a* fixed subspace *of dimension $\geq d$, the uniform distribution over $\mathsf{Rk}_d(\mathbb{F}_q^{n\times m} \mid \mathsf{row} \in \mathcal{W})$ is equivalent to sampling $C \xleftarrow{\$} \mathsf{Rk}_d(\mathbb{F}_q^{n\times d})$, $R \xleftarrow{\$} \mathsf{Rk}_d(\mathbb{F}_q^{d\times m} \mid \mathsf{row} \in \mathcal{W})$ and outputting $A = CR$.*
(III) *If $\mathcal{W} \subseteq \mathbb{F}_q^m$* a uniformly random *subspace of a fixed dimension $\geq d$, the uniform distribution over $\mathsf{Rk}_d(\mathbb{F}_q^{n\times m} \mid \mathsf{row} \in \mathcal{W})$ is equivalent to the uniform distribution over $\mathsf{Rk}_d(\mathbb{F}_q^{n\times m})$.*

Proof. Notice that (II) implies (I) with $\mathcal{W} = \mathbb{F}_q^m$. For (II), it suffices to show that number of ways of writing $A = CR$ as a product of some C and R is the same for every A (for appropriate domains of A, C, R). In particular, it is equal the number of ways of choosing such R so that its rows form a basis of $\mathsf{rowspan}(A)$, which is $\prod_{i=0}^{d-1}(q^d - q^i)$. For (III), we notice that for every $A \in \mathsf{Rk}_d(\mathbb{F}_q^{n\times m})$ the number of spaces $\mathcal{W}$ (of any fixed dimension) such that $A \in \mathsf{Rk}_d(\mathbb{F}_q^{n\times m} \mid \mathsf{row} \in \mathcal{W})$ is just the number of spaces $\mathcal{W}$ that $\mathsf{rowspan}(A) \subseteq \mathcal{W}$ which is the same for every A. □

Entropy, Statistical Distance, Leftover Hash. The *statistical distance* between two random variables X, Y is defined by $\mathbf{SD}(X, Y) = \frac{1}{2}\sum_x |\Pr[X = x] - \Pr[Y = x]|$. We write $X \approx_\epsilon Y$ to denote $\mathbf{SD}(X, Y) \leq \epsilon$, and $X \stackrel{\text{stat}}{\approx} Y$ to denote that the statistical distance is negligible. The *min-entropy* of a random variable X is $\mathbf{H}_\infty(X) \stackrel{\text{def}}{=} -\log(\max_x \Pr[X = x])$. This is a standard notion of entropy used in cryptography, since it measures the worst-case predictability of X. We also review a generalization from [DORS08], called *average conditional min-entropy* defined by

$$\widetilde{\mathbf{H}}_\infty(X|Z) \stackrel{\text{def}}{=} -\log\left(\mathop{\mathbb{E}}_{z\leftarrow Z}\left[\ \max_x \Pr[X = x|Z = z]\ \right]\right) = -\log\left(\mathop{\mathbb{E}}_{z\leftarrow Z}\left[2^{-\mathbf{H}_\infty(X|Z=z)}\right]\right).$$

This measures the *worst-case* predictability of X by an adversary that may observe an *average-case* correlated variable Z.

Lemma B.3 ([DORS08])**.** *Let X, Y, Z be random variables where Y takes on values in a set of size at most 2^ℓ. Then $\widetilde{\mathbf{H}}_\infty(X|(Y, Z)) \geq \widetilde{\mathbf{H}}_\infty((X, Y)|Z) - \ell \geq \widetilde{\mathbf{H}}_\infty(X|Z) - \ell$ and, in particular, $\widetilde{\mathbf{H}}_\infty(X|Y) \geq \mathbf{H}_\infty(X) - \ell$.*

We now define the notion of an (average case) randomness extractor.

Definition B.4 (Extractor)**.** *A* randomized *function $\mathsf{Ext} : \mathcal{X} \to \mathcal{Y}$ is an (k, ϵ)*-extractor *if for all r.v. X, Z such that X is distributed over $\mathcal{X}$ and $\widetilde{\mathbf{H}}_\infty(X|Z) \geq k$, we get $(Z, R, \mathsf{Ext}(X; R)) \stackrel{\text{stat}}{\approx}_\epsilon (Z, R, Y)$ where R is a random variable for the coins of Ext and Y is the uniform over $\mathcal{Y}$.*

Lemma B.5 (Leftover-Hash Lemma [NZ96, DORS08])**.** *Assume that the family $\mathcal{H}$ of functions $h : \mathcal{X} \to \mathcal{Y}$, is a* universal hash family *so that for any $x \neq x' \in \mathcal{X}$ we have $\Pr_{h \xleftarrow{\$} \mathcal{H}}[h(x) = h(x')] \leq 1/|\mathcal{Y}|$. Then the randomized extractor $\mathsf{Ext}(x; h) = h(x)$ is a (k, ϵ)-extractor for any k, ϵ satisfying $k \geq \log(|\mathcal{Y}|) + 2\log(1/\epsilon)$.*

Two-Source Extractors. We crucially rely on the *inner product* being a good two-source extractor [CG88]. Here we use an *average case* version of two-source extractors (analogous to the average case version of seeded extractors defined by [DORS08])

Lemma B.6 (Inner Product Two-Source Extractor)**.** *Let X, Y, Z be correlated r.v. where X, Y have their support in $\mathbb{F}_q^m$ and are independent conditioned on Z. Then $(Z, \langle X, Y\rangle) \overset{\text{stat}}{\approx}_\epsilon (Z, U)$ where U is uniform over $\mathbb{F}_q$ and $\epsilon \leq 2^{-s}$ for $s = 1 + \frac{1}{2}(k_X + k_Y - (m+1)\log(q))$ where $k_X := \widetilde{\mathbf{H}}_\infty(X \mid Z)$, $k_Y := \widetilde{\mathbf{H}}_\infty(Y \mid Z)$.*

The *worst-case* version of the lemma, where Z is empty or fixed, is proved in [CG88] (see [LLTT05] for a very simple proof giving the above parameters). We now prove the *average-case* version, where Z is random. The proof follows that of [DORS08] showing that leftover-hash is a good average-case extractor.

Proof. Let $(X_z, Y_z) = (X, Y|Z = z)$. Then

$$\begin{aligned}
\mathbf{SD}(\ (Z, \langle X, Y\rangle)\ ,\ (Z, U)\ ) = \mathop{\mathbb{E}}_z[\mathbf{SD}(\ \langle X_z, Y_z\rangle\ ,\ U\ )] &\leq \frac{1}{2}\mathop{\mathbb{E}}_z\left[\sqrt{2^{-(\mathbf{H}_\infty(X_z)+\mathbf{H}_\infty(Y_z))}q^{m+1}}\right] \\
&\leq \frac{1}{2}\sqrt{\mathop{\mathbb{E}}_z\left[2^{-(\mathbf{H}_\infty(X_z)+\mathbf{H}_\infty(Y_z))}q^{m+1}\right]} \\
&\leq \frac{1}{2}\sqrt{2^{-(\widetilde{\mathbf{H}}_\infty(X|Z)+\widetilde{\mathbf{H}}_\infty(Y|Z))}q^{m+1}}
\end{aligned}$$

where the first inequality follows from the *worst-case* version of the lemma and the second inequality is Jensen's inequality. This gives us the average case version of the lemma. □

B.1 Statistical Indistinguishability: Hiding Subspaces and Orthogonality

Hiding Subspaces. The following lemma says that, given some leakage on a random matrix A, it is hard to distinguish random vectors from $\mathsf{colspan}(A)$ from uniformly random vectors. A similar lemma was shown in [BKKV10]. Here we give a significantly simpler proof using *leftover-hash* (Lemma B.5).

Lemma B.7 (Subspace Hiding with Leakage)**.** *Let the integers d, n, s, u be polynomial in the security parameter λ. Let $S \in \mathbb{F}_q^{d\times s}$ be an arbitrary (fixed and public) matrix and $\mathsf{Leak} : \{0,1\}^* \to \{0,1\}^\ell$ be an arbitrary function with ℓ-bit output (possibly depending on S). For randomly sampled $A \overset{\$}{\leftarrow} \mathbb{F}_q^{n\times d}$, $V \overset{\$}{\leftarrow} \mathbb{F}_q^{d\times u}, U \overset{\$}{\leftarrow} \mathbb{F}_q^{n\times u}$, we have:*

$$(\mathsf{Leak}(A), AS, V, AV) \overset{\text{stat}}{\approx} (\ \mathsf{Leak}(A), AS, V, U)$$

as long as $(d - s - u)\log(q) - \ell = \omega(\log(\lambda))$.

Proof. The lemma follows by applying leftover-hash (see Lemma B.5) to each row of A independently. In particular, take any row $\vec{a}_i$ of A and think of it as a random source (while all the other rows of A are arbitrarily fixed) whose conditional min-entropy is

$$\widetilde{\mathbf{H}}_\infty(\vec{a}_i \mid AS, \mathsf{Leak}(A)) \geq d\log(q) - (s\log(q) + \ell).$$

Think of V as the seed of the universal hash function $h_V(\vec{a}_i) = \vec{a}_i \cdot V$ whose output size is $u\log(q)$ bits. The leftover-hash lemma tells us that the ith row of AV looks uniform. By using the hybrid argument over all n rows, the first part of the lemma follows.

□

The following corollary can be interpreted as saying that, given leakage on a matrix A, one cannot distinguish random vectors from $\mathsf{colspan}(A)$ from uniformly random vectors, even if A is a random matrix of some (non-full) rank d and the row-space of A is fixed.

Corollary B.8. *Let the integers n, d, s, u be polynomial in the security parameter λ, with $n \geq d \geq 1$. Let $S \in \mathbb{F}_q^{n\times s}$ be a fixed matrix, $\mathcal{W} \subseteq \mathbb{F}_q^n$ be some fixed subspace of dimension at least d, and $\mathsf{Leak} : \{0,1\}^* \to \{0,1\}^\ell$ be an arbitrary function (possibly depending on $S, \mathcal{W}$). Then, for a random $V \xleftarrow{\$} \mathbb{F}_q^{n\times u}$, $U \xleftarrow{\$} \mathbb{F}_q^{n\times u}$ and $A \xleftarrow{\$} \mathsf{Rk}_d(\mathbb{F}_q^{n\times n} \mid \mathsf{row} \in \mathcal{W})$ we have*

$$(\mathsf{Leak}(A), AS, V, AV) \overset{\text{stat}}{\approx} (\ \mathsf{Leak}(A), AS, V, U)$$

as long as $(d - s - u)\log(q) - \ell = \omega(\log(\lambda))$ and $q = \omega(\log(\lambda))$.

Proof. Sampling $A \xleftarrow{\$} \mathsf{Rk}_d(\mathbb{F}_q^{n\times n} \mid \mathsf{row} \in \mathcal{W})$ is equivalent to sampling $C \xleftarrow{\$} \mathsf{Rk}_d(\mathbb{F}_q^{n\times d})$, $R \xleftarrow{\$} \mathsf{Rk}_d(\mathbb{F}_q^{d\times n} \mid \mathsf{row} \in \mathcal{W})$ and setting $A = CR$. The corollary then follows by applying Lemma B.7 to the matrix C while thinking of R as public. In particular, ℓ-bit leakage $\mathsf{Leak}(A)$ on the matrix $A = CR$ is equivalent to ℓ-bit leakage $\mathsf{Leak}'(C)$ on the matrix C. Furthermore $AS = CRS = CS'$ for some $S' \in \mathbb{F}_q^{n\times s}$ and $AV = CRV = CV'$ where V' is uniformly random over the choice of V since R is full rank. We only use the fact that q is super-polynomial to switch from $C \xleftarrow{\$} \mathsf{Rk}_d(\mathbb{F}_q^{n\times d})$ to $C \xleftarrow{\$} \mathbb{F}_q^{n\times d}$. So, by Lemma B.7, we have $(\mathsf{Leak}'(C), AS', V', AV') \overset{\text{stat}}{\approx} (\mathsf{Leak}'(C), AS', V', U)$ which gives us our corollary. □

Hiding Orthogonality. The following lemma can be interpreted as saying that if two random vectors X, Y leak individually, and the total amount of leakage Z is sufficiently small, an attacker cannot distinguish whether X, Y are random orthogonal vectors or uniformly random vectors. A slightly different but related lemma with a related proof strategy appears in [DDV10].

Lemma B.9 (Orthogonality Hiding)**.** *Let $X, Y, Z = \mathsf{Leak}(X, Y)$ be correlated r.v. where X, Y have their support in $\mathbb{F}_q^m$ and are independent conditioned on Z. Let E be the event that $\langle X, Y\rangle = 0$ and let $(X', Y', Z' = \mathsf{Leak}(X', Y')) := (X, Y, Z \mid E)$ be the joint distribution conditioned on the event E. Then $Z \overset{\text{stat}}{\approx} Z'$ as long as $\widetilde{\mathbf{H}}_\infty(X \mid Z) + \widetilde{\mathbf{H}}_\infty(Y \mid Z) - (m + 3)\log(q) = \omega(\log(\lambda))$.*

Proof. We rely on the fact that the inner product is a good *two source extractor* (Lemma B.6). In particular, let ϵ' be the bound from Lemma B.6 such that $(Z, \langle X, Y\rangle) \overset{\text{stat}}{\approx}_{\epsilon'} (Z, U)$ where U is uniform over $\mathbb{F}_q$. Assume there is a statistical test D such that $\Pr[D(Z') = 1] - \Pr[D(Z) = 1] = \epsilon$. Then we claim that there is a statistical test D' that distinguishes $(Z, \langle X, Y\rangle)$ from (Z, U) with advantage $\epsilon/q - \epsilon'$. This implies that $\epsilon/q - \epsilon' \leq \epsilon' \Rightarrow \epsilon \leq 2q\epsilon'$ and, using the bound on ϵ' from Lemma B.6, our lemma follows. Therefore we are left to describe and analyze the statistical test D'.

The test $D'(\cdot, \cdot)$ just outputs 0 if the second component is non-zero and otherwise outputs the evaluation of D on the first component. This gives

$$\begin{aligned}
\Pr[D'(Z, \langle X, Y\rangle) = 1] &= \Pr[D'(Z, \langle X, Y\rangle) = 1 \mid \langle X, Y\rangle = 0] \Pr[\langle X, Y\rangle = 0] \geq \Pr[D(Z') = 1](1/q - \epsilon') \\
\Pr[D'(Z, U) = 1] &= \Pr[D'(Z, U) = 1 \mid U = 0] \Pr[U = 0] = \Pr[D(Z) = 1]/q
\end{aligned}$$

and so D' has the claimed advantage $\epsilon/q - \epsilon'$. □

B.2 Computational Indistinguishability: Extended Rank-Hiding Assumption

We define an *extended rank hiding assumption* which says that one cannot distinguish between matrices of different ranks *in the exponent*, even given some additional vectors in the kernel of the matrix.

Definition B.10 (Extended Rank Hiding Assumption)**.** *The k* extended rank hiding assumption *for a pairing generation algorithm $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, q, e, \mathbf{g}, \mathbf{h}) \leftarrow \mathcal{G}(1^\lambda)$ states that for any integer constants i, j, n, m*

satisfying $k \leq i < j \leq \min\{n, m\}$ and for $t := m - j$, we get the indistinguishability property:

$$\left(\mathsf{prms}, \mathbf{g}^X, \vec{v}_1, \ldots, \vec{v}_t \;\middle|\; \mathsf{prms} \leftarrow \mathcal{G}(1^\lambda), X \xleftarrow{\$} \mathsf{Rk}_i(\mathbb{F}_q^{n\times m}), \{\vec{v}_\rho\}_{\rho=1}^t \xleftarrow{\$} \ker(X)\right)$$
$$\overset{\text{comp}}{\approx}$$
$$\left(\mathsf{prms}, \mathbf{g}^X, \vec{v}_1, \ldots, \vec{v}_t \;\middle|\; \mathsf{prms} \leftarrow \mathcal{G}(1^\lambda), X \xleftarrow{\$} \mathsf{Rk}_j(\mathbb{F}_q^{n\times m}), \{\vec{v}_\rho\}_{\rho=1}^t \xleftarrow{\$} \ker(X)\right)$$

for the left group $\mathbb{G}_1$ and also for the right group $\mathbb{G}_2$ by substituting the generator $\mathbf{h}$ instead of $\mathbf{g}$ above.

Lemma B.11. *The k* extended rank hiding *assumption is implied by the (regular) k* rank hiding *assumption which is in-turn implied by the k-*linear *assumption.*

Proof. The second part of the lemma (k-linear $\Rightarrow$ k rank hiding) is shown in [NS09] and hence we skip it. For the first part of the lemma we show a reduction that converts a rank hiding challenge into an extended rank hiding challenge. In particular, the reduction is given the challenge $(\mathsf{prms}, \mathbf{g}^{X'})$ where either (I) $X' \xleftarrow{\$} \mathsf{Rk}_i(\mathbb{F}_q^{n\times j})$ or (II) $X' \xleftarrow{\$} \mathsf{Rk}_j(\mathbb{F}_q^{n\times j})$ for some i, j, n satisfying $i < j \leq n$. The reduction chooses a random matrix $R \xleftarrow{\$} \mathsf{Rk}_j(\mathbb{F}_q^{j\times m})$ and sets $\mathbf{g}^X = \mathbf{g}^{X'R}$ (which can be computed efficiently without knowing X'). It also chooses $\vec{v}_1, \ldots, \vec{v}_t \xleftarrow{\$} \ker(R)$ and outputs the challenge $(\mathsf{prms}, \mathbf{g}^X, \vec{v}_1, \ldots, \vec{v}_t)$.

Assume that the received challenge if of type (II). Then, by Lemma B.2, we see that the distribution of $X = X'R$ is the same as a random sample from $\mathsf{Rk}_j(\mathbb{F}_q^{n\times m})$. Furthermore since $\mathsf{rowspan}(X) = \mathsf{rowspan}(R)$, the vectors $\vec{v}_1, \ldots, \vec{v}_t$ are random over $\ker(R) = \mathsf{rowspan}(R)^\perp = \mathsf{rowspan}(X)^\perp$. Therefore the outputs of the reduction is distributed the same as an extended rank-hiding assumption challenge with rank j.

Assume that the received challenge is of type (I). Then, by Lemma B.2, we can sample X' via $X' = X_1X_2$ where $X_1 \xleftarrow{\$} \mathsf{Rk}_i(\mathbb{F}_q^{n\times i})$ and $X_2 \xleftarrow{\$} \mathsf{Rk}_i(\mathbb{F}_q^{i\times j})$. Applying Lemma B.2 once more, we get X_2R is uniformly random over $\mathsf{Rk}_i(\mathbb{F}_q^{i\times m})$ and applying it once again we see that $X = X'R = X_1(X_2R)$ is uniformly random over $\mathsf{Rk}_i(\mathbb{F}_q^{n\times m})$. Furthermore $\mathsf{rowspan}(X) = \mathsf{rowspan}(X_2R) \subseteq \mathsf{rowspan}(R)$ and so $\ker(R) \subseteq \ker(X)$. Moreover, $\ker(R)$ is a random $t = (m - j)$ dimensional subspace of the $(m - i)$ dimensional space $\ker(X)$. Since sampling t random vectors from $\ker(X)$ is statistically close to first choosing a random t-dimensional subspace $\ker(R) \subseteq \ker(X)$ and then sampling t random vectors from that, we see the joint distribution on $\mathbf{g}^X, \vec{v}_1, \ldots, \vec{v}_t$ produced by the reduction is (statistically close to) the extended rank hiding assumption challenge with rank i. □

B.3 (Re)Programming Updates

Here we show two useful results about the scenario where we continually update a key/ciphertext and the attacker is continually leaking on the process. The first lemma (Lemma B.12) shows that after continually updating a key/ciphertext of some rank (e.g. low, mid, high) honestly, we get a random key/ciphertext of the same rank at end. If we program updates to reduce the rank, we get a random key/ciphertext of the appropriate reduced rank at the end. This even holds if some of the updates are programmed to annihilate random vectors unrelated to the key/ciphertext. The second lemma (Lemma B.13) says that an attacker who is leaking sufficiently few bits of information on the update process cannot tell whether some updates are programmed to annihilate random vectors unrelated to the key/ciphertext.

We consider a randomized process called the *Programming Experiment*, defined as follows. First, we choose a random matrix $R \xleftarrow{\$} \mathbb{F}_q^{n\times l}$ and label its columns $R = [\vec{r}_1^\top \mid \cdots \mid \vec{r}_l^\top]$. Then we choose a series of (some polynomial number) t update matrices $A_1, \ldots, A_t \in \mathsf{Rk}_d(\mathbb{F}_q^{n\times n})$ and iteratively set $R_{i+1} = A_iR_i$ where $R_1 = R$. The experiment is defined by an *update regime* $\mathfrak{R}$ which specifies if/how the various updates are programmed. Each update can be programmed to annihilate some of the columns of R and some additional random vectors. That is, for each update i, the regime $\mathfrak{R}$ specifies a (possibly empty) subset $J_i \subseteq \{1, \ldots, l\}$ of column-indices to be annihilated and a number $u_i \geq 0$ of random vectors to be annihilated, subject to $|J_i| + u_i \leq n - d$. The programming experiment then chooses update i so that A_i is programmed to annihilate the vectors $\{\vec{r}_j | j \in J_i\} \cup \{\vec{v}_{i,1}, \ldots, \vec{v}_{i,u_i}\}$ for fresh and random $\vec{v}_{i,j} \stackrel{\$}{\leftarrow} \mathbb{F}_q^n$. The update regime is fully specified by $\mathfrak{R} = (J_1, u_1, \ldots, J_t, u_t)$.

Lemma B.12 (Program Correctness)**.** *Let $\mathfrak{R} = (J_1, u_1, \ldots, J_t, u_t)$ be an update regime that annihilates the columns with indices $\bigcup_{i=1}^t J_i = \{\rho + 1, \ldots, l\}$ along with $u = \sum_{i=1}^t u_i$ additional random vectors. Assume $(d - l - u - 1)\log(q) = \omega(\log(\lambda))$. Then the distribution of the final matrix R_t is statistically close to*

$$R_t = \begin{bmatrix} | & & | & \ddots & & \\ \vec{v}_1^\top & \cdots & \vec{v}_\rho^\top & & \mathbf{0} & \\ | & & | & & & \ddots \end{bmatrix}$$

for uniformly random and independent columns $\vec{v}_1, \ldots, \vec{v}_\rho \in \mathbb{F}_q^n$. [6]

Let *GameProgram*$(\mathfrak{R})$ be a game between a challenger and an attacker, where the challenger runs the programming experiment with the regime $\mathfrak{R}$. In each round $i = 1, 2, \ldots, t$, the challenger gives the attacker the matrix R_i *in full* and answers ℓ leakage queries on the update matrix A_i (the attacker can choose leakage queries adaptively depending on its view thus far). The output of the game is the view of the attacker at the end of the game.

Lemma B.13 (Reprogramming)**.** *Let $\mathfrak{R} = (J_1, u_1, \ldots, J_t, u_t)$ and $\mathfrak{R}' = (J_1, u'_1, \ldots, J_t, u'_t)$ be two update regimes that agree on which columns of R are annihilated and when (i.e. the sets J_i) but not on how many additional random vectors are annihilated and when (i.e. the values u_i, u'_i). Let $u^* = \max\left(\sum_{i=1}^t u_i, \sum_{i=1}^t u'_i\right)$ be the maximum number of additional random vectors annihilated by either of the regimes. If $(d - l - u^* - 1)\log(q) - \ell = \omega(\log(\lambda))$ then* $\text{GameProgram}(\mathfrak{R}) \stackrel{\text{stat}}{\approx} \text{GameProgram}(\mathfrak{R}')$.

Both of the above lemmas (Lemma B.12, Lemma B.13) follow as special cases from the following lemma.

Lemma B.14. *Let $\mathfrak{R} = (J_1, u_1, \ldots, J_t, u_t)$ be an update regime and let $u = \sum_{i=1}^t u_i$ be the total number of random vectors it annihilates. Let $\mathfrak{R}^* = (J_1, 0, \ldots, J_t, 0)$ be an update regime that agrees with $\mathfrak{R}$ on which columns of R are annihilated and when (i.e. the sets J_i) but does not annihilate any additional random vectors (i.e. $u_i^* = 0$). Let $D = A_t \cdots A_1$ be the product of all the update matrices at the end of the game and let $\vec{\mu}_1, \ldots, \vec{\mu}_\rho \stackrel{\$}{\leftarrow} \mathbb{F}_q^n$ be fresh random vectors. Then*

$$\left(\mathit{GameProgram}(\mathfrak{R}), D\vec{\mu}_1^\top, \ldots, D\vec{\mu}_\rho^\top\right) \stackrel{\text{stat}}{\approx} \left(\mathit{GameProgram}(\mathfrak{R}^*), \vec{\mu}_1^\top, \ldots, \vec{\mu}_\rho^\top\right)$$

as long as $(d - l - u - \rho - 1)\log(q) - \ell = \omega(\log(\lambda))$.

Proof. For $i \geq j \geq 0$, we define $D[i,j] := A_i A_{i-1} \cdots A_j$ and $D_i := D[i,1]$ to be products of update matrices A_i (as a a corner case, D_0, A_0 are defined to be the identity matrix as is $D[i,j]$ for $i < j$).

To prove the lemma, we slowly move from the distribution on the left to the one on the right. That is, we define a series of hybrid games *GameHybrid* j where *GameHybrid* 0 is the left-hand distribution and *GameHybrid* $t+1$ is the right-hand one. In *GameHybrid* j, the update matrices A_i for $i \leq j$ are chosen as specified by the regime $\mathfrak{R}^*$. Moreover, the update matrices A_i for $i > j$ are chosen by placing u_i random vectors from the span of $D[i-1,j]$ into their kernel (instead of the span of D_{i-1}). More formally, for $i > j$ the update A_i in *GameHybrid* j is chosen by rescaling a randomly sampled $A'_i \stackrel{\$}{\leftarrow} \mathsf{Rk}_d(\mathbb{F}_q^{n\times n} \mid \mathcal{W})$ where

$$\mathcal{W} = (\ \ \{\ D_{i-1}\vec{r}_k \mid k \in J_i\ \}\ ,\ \{\ D[i-1,j]\vec{v}_{i,k} \mid 1 \leq k \leq u_i\}\ \ )^\perp$$

[6] Note: in this lemma, we do *not* condition on seing any information about the initial matrix R or the update matrices A_i.

and $\vec{v}_{i,k} \in \mathbb{F}_q^n$ are uniformly random. Lastly, at the end of *GameHybrid j*, we append the additional vectors chosen as $D[t,j]\vec{\mu}_1, \ldots, D[t,j]\vec{\mu}_\rho$ for uniformly random $\{\vec{\mu}_k \xleftarrow{\$} \mathbb{F}_q^n\}_{k=1}^{\rho}$.

We show that for each $j \in \{0, \ldots, t\}$ we have *GameHybrid j* $\overset{\text{stat}}{\approx}$ *GameHybrid* $j+1$. We do this by relying on the *subspace hiding lemma* (Corollary B.8) and showing a "reduction" which uses a distinguishing strategy for the above two hybrids to get a distinguishing strategy for the two distributions in the subspace hiding lemma. The reduction chooses the initial matrix $R_1 \xleftarrow{\$} \mathbb{F}_q^{n \times l}$.

For updates $i < j$, the reduction chooses the update matrices A_i according to the regime $\mathfrak{R}^*$ and gives the attacker the corresponding leakage. It also gives the attacker the values $R_{i+1} = A_i R_i$ in full.

For the update A_j, the reduction chooses its *row space* $\mathcal{W}_j$ honestly as specified by the regime $\mathfrak{R}^*$ so that A_j should be sampled by rescaling a random sample $A'_j \xleftarrow{\$} \mathsf{Rk}_d(\mathbb{F}_q^{n \times n} \mid \mathsf{row} \in \mathcal{W}_j)$. However, the reduction does not choose A_j itself, but instead uses a *subspace hiding challenge* (Corollary B.8). That is, let Leak_j be the leakage function chosen by the attacker in round j (on matrix A_j) and let Leak'_j be a modified function which, given matrix A'_j first rescales its rows to get A_j and then outputs $\mathsf{Leak}_j(A_j)$. Define the matrix $S = [R_j | \vec{1}^\top]$. The reduction specifies $S, \mathcal{W}_j, \mathsf{Leak}'_j$ and gets a *subspace hiding* challenge of the form:

$$(\mathsf{Leak}'_j(A'_j), A'_j S, V')$$

where $A'_j \in \mathsf{Rk}_d(\mathbb{F}_q^{n \times n} \mid \mathsf{row} \in \mathcal{W}_j)$ and V' is either of the form $A'_i U$ or just U for a random $n \times (u + \rho)$ matrix U. We call the former distribution *type I* and the latter *type II*. The reduction gives the first component of the challenge to the attacker as its leakage on A_j. It uses the value $\vec{\eta}^\top = A'_j \vec{1}^\top$ (from the second component of the challenge) to compute an $n \times n$ rescaling-matrix N whose diagonals are the entries of $\vec{\eta}$ so that $A_j = N A'_j$ is the rescaled version of A'_j. It sets $R_{j+1} := N(A'_j R_j)$ (where $A'_j R_j$ is in the second component of the challenge) and gives R_{j+1} to the attacker. It also sets $V := NV' \in \mathbb{F}_q^{n \times (u+\rho)}$ and interprets the columns of V as the $u + \rho$ vectors

$$\{\vec{v}_{i,k} \mid 1 \le i \le t, 1 \le k \le u_i\} \quad , \quad \{\vec{\mu}_k\}_{k=1}^{\rho}$$

For each update $i > j$, the reduction chooses the updates A_i iteratively by rescaling a random $A'_i \xleftarrow{\$} \mathsf{Rk}_d(\mathbb{F}_q^{n \times n} \mid \mathsf{row} \in \mathcal{W}_i)$ where

$$\mathcal{W}_i = (\ \{ \ D_{i-1}\vec{r}_k \mid k \in J_i \ \} \quad , \quad \{ \ D[i-1, j+1]\vec{v}_{i,k} \mid 1 \le k \le u_i \} \ )^\perp$$

and the values $D_{i-1}\vec{r}_k$ are derived from R_{j+1} and $A_{j+1}, A_{j+2} \ldots$ (without knowing A_j). Similarly, The reduction also samples the final ρ vectors as $\{ \ D[t, j+1]\vec{\mu}_k\}_{k=1}^{\rho}$. It gives the attacker leakage on the updates A_i, the full values R_i, and finally the ρ vectors $\vec{\mu}_k$.

If the challenge is of *type I* and $V' = A'_i U$ for a uniform U then the reduction produces the distribution *GameHybrid j*. This is because, for any columns $\vec{v}$ of V we can write $D[i-1, j+1]\vec{v} = D[i-1, j+1]A_j\vec{u} = D[i-1, j]\vec{u}$ where $\vec{u}$ is uniformly random. So all of the spaces $\mathcal{W}_i$ and the final vectors $\vec{\mu}_k$ are distributed as in *GameHybrid j*. On the other hand, if the challenge is *type II*, then the reduction produces the distribution *GameHybrid* $j+1$. This is because $V' = U$ is uniformly random and hence $V = NU$ is also uniformly random (since N is full-rank and U is independent of N).

Therefore the hybrids are indistinguishable from each other and hence the left and right hand distributions of the lemma are indistinguishable.

□

Proof of Lemma B.13. We just apply Lemma B.14 twice with $\rho = 0$ to get

$$\mathrm{GameProgram}(\mathfrak{R}) \overset{\text{stat}}{\approx} \mathrm{GameProgram}(\mathfrak{R}^*) \overset{\text{stat}}{\approx} \mathrm{GameProgram}(\mathfrak{R}').$$

Proof of Lemma B.12. Let us write the initial matrix as $R = [R^{\text{left}} | R^{\text{right}}]$ where R^{left} consists of the first ρ columns. Then, the choice of the updates $A_1, \ldots, A_t$ in the programming experiment is completely independent of R^{left}. Therefore, we can think of playing the programming experiment with *only* R^{right} and choosing the ρ columns of R^{left} randomly afterwards. Therefore, applying Lemma B.14 to only the $l' = l - \rho$ sub-matrix R^{right} (and setting $\ell = 0$) we get statistical indistinguishability between the distributions on the ρ columns given by $R_t^{\text{left}} = D_t R^{\text{left}} \overset{\text{stat}}{\approx} V$ where V is uniformly random over $\mathbb{F}_q^{n \times \rho}$. It's also clear that $R_t^{\text{right}} = D_t R^{\text{right}} = \mathbf{0}$, and so $R_t = [R_t^{\text{left}} | R_t^{\text{right}}] \overset{\text{stat}}{\approx} [V|\mathbf{0}]$, as we wanted to show.

C Proof of Security (Theorem 4.1)

In Section C.1, we define the main hybrid games that are used in the proof. Then, in Section C.2, we proceed to prove the indistinguishability of the various hybrid games.

C.1 Hybrid Game Definitions

We first define several hybrid games. The main part of the proof is to show computational/statistical indistinguishability between these games. The output of each game consists of the view of the attacker $\mathcal{A}$ as well as a bit b chosen by the challenger representing its choice of which message $\mathbf{msg}_0, \mathbf{msg}_1$ to encrypt. We assume that the attacker $\mathcal{A}$ makes a maximum of q_{sk} update queries on the secret-key share, and at most q_{ct} update queries on the ciphertext share during the course of the game. We describe the games in detail below, and also give a helpful pictorial representation in Figure 2.

Real Game : This is the original "ℓ-CLRS-Friendly Encryption" security game between the adversary and the challenger (see Section 3.2). The output of the game consists of the view of the attacker $\mathcal{A}$ and the bit b chosen by the challenger.

Game' 0 : In this game, the challenger chooses the initial ciphertext incorrectly as a *low* ciphertext of the message $\mathbf{msg}_b$ instead of encrypting it honestly (the key/ciphertext bases are chosen as random *un*correlated bases). The initial secret key is chosen honestly and all ciphertext/key updates are chosen honestly as before.

Game i : We define Game i for $i = 0, \ldots, q_{\mathsf{sk}} + 1$. In all future game definitions, we will include two additional "dummy keys" $\mathsf{sk}_{-1}, \mathsf{sk}_0$ and key-update matrices A_{-1}, A_0 chosen by the challenger (but not observed or leaked on by the attacker). That is, the challenger (in its head) always initially chooses sk_{-1}, then updates it to sk_0 using an update matrix A_{-1}, then updates sk_0 to sk_1 using an update matrix A_0 and so on. However, the values $\mathsf{sk}_{-1}, A_{-1}, \mathsf{sk}_0, A_0$ are then ignored, and the first key and update matrix that the attacker can ask leakage queries on are sk_1 and A_1 respectively. In every Game i, the initial key sk_{-1} is chosen as a *high key* with the exponent matrix:

$$S = \begin{bmatrix} | & \cdots & | \\ \vec{r}_1^{\top} & \cdots & \vec{r}_{m-1}^{\top} \\ | & \cdots & | \end{bmatrix} \begin{bmatrix} - & \vec{w}_1 & - \\ & \cdots & \\ - & \vec{w}_{m-1} & - \end{bmatrix} + \begin{bmatrix} \\ \vec{1}^{\top} \\ \\ \end{bmatrix} \begin{bmatrix} & \vec{t} & \end{bmatrix}$$

The first key update matrices $A_{-1}, \ldots, A_{i-2}$ are chosen honestly. The update A_{i-1} is *programmed to annihilate* the $m-3$ vectors $(\vec{r}_3, \ldots, \vec{r}_{m-1})$ reducing the key to a *mid*. The update A_i is programmed to annihilate $\vec{r}_2$ along with $m-4$ random vectors, reducing the key to an *honest* key. The update A_{i+1} is programmed to annihilate a single random vector. All other key updates are chosen honestly. Note that, in Game i, the keys $\mathsf{sk}_{-1}, \ldots, \mathsf{sk}_{i-1}$ are high keys, sk_i is a mid key, and $\mathsf{sk}_{i+1}, \ldots$ are honest keys.[7] The initial

[7] In particular, in *Games 0* the initial observed (non-dummy) key sk_1 is an honest key, in *Game* 1 it is a mid key, and in all future games it is a high key. For the proof, it becomes easier to just pretend that there are some "dummy" mid and high keys $\mathsf{sk}_{-1}, \mathsf{sk}_0$ even in Games 0,1 so as not to have to define special corner cases for these games.

ciphertext ct_1 is chosen as in Game' 0 (a low ciphertext) and all ciphertext updates are performed honestly. The ciphertext and key bases are random and *un*correlated. Notice that the secret keys $\mathsf{sk}_{-1}, \ldots, \mathsf{sk}_i$ do *not* decrypt the low ciphertexts correctly, but all future keys $\mathsf{sk}_{i+1}, \ldots, \mathsf{sk}_{q_{\mathsf{sk}}+1}$ do.

GameCor' $(i+1, 0)$ **:** We define GameCor' $(i+1, 0)$ for $i = 0, \ldots, q_{\mathsf{sk}} + 1$. In this game, the initial secret key sk_{-1} is a *high* key and the initial ciphertext ct_1 is a *low* ciphertext as in *Game i*. However, the ciphertext and key bases are now *correlated*. Also, the regime of key updates is modified from *Game i*. The first updates $A_{-1}, \ldots, A_{i-2}$ are chosen honestly. The update A_{i-1} is programmed to annihilate $m-3$ random vectors, the update A_i is programmed to annihilate the vectors $(\vec{r}_3, \ldots, \vec{r}_{m-1})$ reducing the key to a *mid*, the update A_{i+1} is programmed to annihilate $\vec{r}_2$ along with $m-4$ random vectors reducing the key to *honest*. All future key updates are then chosen honestly. All ciphertext updates are also chosen honestly (as in Game i). Note that, in GameCor' $(i+1, 0)$, the keys $\mathsf{sk}_{-1}, \ldots, \mathsf{sk}_i$ are high keys and do not decrypt the low ciphertexts correctly, sk_{i+1} is a mid key but does decrypt the low ciphertexts since the bases are correlated, and $\mathsf{sk}_{i+2}, \ldots$ are honest keys which always decrypt correctly.

GameCor $(i+1, j)$ **:** We define GameCor' $(i+1, j)$ for $i = 0, \ldots, q_{\mathsf{sk}} + 1$ and $j = 0, \ldots, q_{\mathsf{ct}} + 1$. As in GameCor' $(i+1, 0)$, the key and ciphertext bases are *correlated*. Also, the initial secret key sk_{-1} and the regime of key updates is chosen the same way as in GameCor' $(i+1, 0)$. In all future game definitions, we will include two additional "dummy ciphertexts" $\mathsf{ct}_{-1}, \mathsf{ct}_0$ and ciphertext-update matrices B_{-1}, B_0 chosen by the challenger (but not observed or leaked on by the attacker). That is, the challenger initially chooses ct_{-1}, then updates it to ct_0 using an update matrix B_{-1}, then updates that to ct_1 using an update matrix B_0 and so on. However, the values $\mathsf{ct}_{-1}, B_{-1}, \mathsf{ct}_0, B_0$ are then ignored, and the first ciphertext and ciphertext-update matrix that the attacker can leak on are ct_1 and B_1 respectively. The initial ciphertext ct_{-1} is chosen as a *high ciphertext* using the exponent matrix

$$C = \begin{bmatrix} | & & | \\ \vec{u}_1^\top & \cdots & \vec{u}_{m-1}^\top \\ | & & | \end{bmatrix} \begin{bmatrix} - & \vec{c}_1 & - \\ & \cdots & \\ - & \vec{c}_{m-1} & - \end{bmatrix}$$

The first ciphertext updates $B_{-1}, \ldots, B_{j-2}$ are chosen honestly, the update B_{j-1} is programmed to annihilate the $m-3$ vectors $(\vec{u}_3, \ldots, \vec{u}_{m-1})$ reducing the ciphertext to a*mid*, the update B_j is programmed to annihilate $\vec{u}_2$ along with $m-4$ other random vectors reducing the ciphertext to a *low*, the update matrix B_{j+1} is programmed to annihilate a single random vector. All future ciphertext updates are chosen honestly. Note that, in GameCor $(i+1, j)$, the initial ciphertexts $\mathsf{ct}_{-1}, \ldots, \mathsf{ct}_{j-1}$ are high, ct_j is mid, and $\mathsf{ct}_{j+1}, \ldots$ are low.[8]

GameSuperCor $(i+1, j+1)$ **:** We define GameSuperCor $(i+1, j+1)$ for $i = 0, \ldots, q_{\mathsf{sk}} + 1$ and $j = 0, \ldots, q_{\mathsf{ct}} + 1$. In this game, the initial secret-key sk_{-1} is a *high key* and the initial ciphertext ct_{-1} is a high ciphertext as in GameCor $(i+1, j)$. However, the key and ciphertext bases are now *super-correlated*. Also, the regime of ciphertext updates is modified from GameCor $(i+1, j)$. The first updates $B_{-1}, \ldots, B_{j-2}$ are chosen honestly, the update B_{j-1} is programmed to annihilate $m-3$ uniformly random vectors, the update matrix B_j is programmed to annihilate the vectors $(\vec{u}_3, \ldots, \vec{u}_{m-1})$ reducing the ciphertext to a *mid*, the update B_{j+1} is programmed to annihilate $\vec{u}_2$ reducing the ciphertext to a *low*. All future ciphertext updates are then chosen honestly. The regime of key updates is the same way as in GameCor $(i+1, j)$. Note that the ciphertexts $\mathsf{ct}_{-1}, \ldots, \mathsf{ct}_j$ are high, ct_{j+1} is mid, and $\mathsf{ct}_{j+2}, \ldots$ are low.

[8]In particular, in *GamesCor* $i+1, 0$ the initial observed (non-dummy) ciphertext ct_1 is low, in *GameCor* $i+1, 1$ it is mid, and in all future games it is high. For the proof, it becomes easier to just pretend that there are some "dummy" mid and high ciphertexts ct_{-1}, ct_0 even when $j = 0, 1$ so as not to have to define special corner cases for these games.

GameFinal : This game is defined the same way as *Game* $q_{\mathsf{sk}} + 1$, except that instead of using the message $\mathbf{msg}_b$ in the (low) ciphertext, we just use message $1_{\mathbb{G}_T}$. More specifically, in *GameFinal*, the secret key is a random *high key* and all of the key updates are honest. The ciphertext is a random *low ciphertext* of the message $1_{\mathbb{G}_T}$ and all of the ciphertext updates are honest as well. In particular, the view of the attacker in GameFinal is independent of the challenger's bit b.

C.2 Hybrid Indistinguishability Arguments

In Figure 3, we show the sequence of hybrid arguments that is used to derive the indistinguishability: *Real* $\overset{\text{comp}}{\approx}$ *GameFinal*. In this section, we prove each of the necessary sub-steps in separate lemmas.

Lemma C.1. *Real* $\overset{\text{comp}}{\approx}$ *Game' 0.*

Proof. We show a reduction from the $(k = 1)$-*extended rank hiding assumption* (Definition B.10, Lemma B.11). The reduction is given a challenge $\mathbf{g}^P$, $\vec{w}$ where $P \xleftarrow{\$} \mathsf{Rk}_x(\mathbb{F}_q^{2\times m})$ is either of rank $x = 1$ or $x = 2$ and $\vec{w} \xleftarrow{\$} \mathsf{ker}(P)$. Let us denote the two rows of P by $\vec{p}, \vec{c}_1$ respectively. The reduction puts the values $\mathbf{g}^{\vec{p}}, \mathbf{h}^{\vec{w}}$ in prms and chooses its own random $\vec{t}$ to set up the initial public/secret key pk, sk. To create the encryption of $\mathbf{msg}_b$, the reduction sets $\mathsf{ct} = (\mathbf{g}^C, e(\mathbf{g}, \mathbf{h})^{\vec{z}})$ where it uses the challenge $\mathbf{g}^{\vec{c}_1}$ to compute $\mathbf{g}^C$ for $C = \vec{u}^\top \vec{c}_1$ where $\vec{u} \xleftarrow{\$} \mathbb{F}_q^n$ is random. The matching second component $e(\mathbf{g}, \mathbf{h})^{\vec{z}}$ can then be computed efficiently as a deterministic function of $\mathbf{g}^C, \vec{t}$ and the message $\mathbf{msg}_b$. The reduction chooses all of the key/ciphertext update matrices honestly and answers all update/leakage queries honestly.

If the challenge has $x = 1$ then $\vec{c}_1 = u'\vec{p}$ for some scalar u' and hence the ciphertext is a correctly distributed *honest* encryption of $\mathbf{msg}_b$, so the reduction produces the distribution of the *Real Game*. If $x = 2$ then $\vec{c}_1$ is a random and independent vector in the space $(\vec{w})^\perp$ and hence the ciphertext is a correctly distributed *low* encryption of $\mathbf{msg}_b$, so the reduction produces the distribution of *Game' 0*. Therefore the two are computationally indistinguishable. □

Lemma C.2. *If* $\ell \le (n - 3m + 6)\log(q) - \omega(\log(\lambda))$ *then* Game' *0* $\overset{\text{stat}}{\approx}$ Game *0.*

Proof. The only difference between *Game' 0* and *Game 0* is the joint distribution on the initial secret key sk_1 and the update matrix A_1. Conditioned on sk_1, A_1, all other values are sampled the same way in the two games. In *Game' 0*, the key sk_1 is a randomly chosen *honest key*, and A_1 is an *honest update*. In *Game 0* the distribution on sk_1, A_1 is slightly more complicated. First we choose a random *high key* $\mathsf{sk}_{-1} = \mathbf{h}^{S-1}$ with exponent

$$S_{-1} = \begin{bmatrix} | & \cdots & | \\ \vec{r}_1^\top & \cdots & \vec{r}_{m-1}^\top \\ | & \cdots & | \end{bmatrix} \begin{bmatrix} - & \vec{w}_1 & - \\ & \cdots & \\ - & \vec{w}_{m-1} & - \end{bmatrix} + \begin{bmatrix} \vec{1}^\top \end{bmatrix} \begin{bmatrix} & \vec{t} & \end{bmatrix}$$

Then we choose the updates: A_{-1} programmed to annihilate $\vec{r}_3, \ldots, \vec{r}_{m-1}$ yielding a mid-key sk_0, A_0 is programmed to annihilate $\vec{r}_2$ along with $m - 4$ random vectors yielding an honest key sk_1, and A_1 is programmed to annihilate a single random vector.

We claim this joint distribution on A_1 and sk_1 in *Game 0* is statistically indistinguishable from that of *Game' 0*. To see this, we first apply the *reprogramming lemma* (Lemma B.13 with $u^* = m - 3, d = n - m + 3, l = m - 1$) to switch from A_1 being programmed to annihilate a single vector to just being honest. Next we just use *program correctness lemma* (Lemma B.12; with $l = m - 1$, $\rho = 1$, $u = m - 4$, $d = n - m + 3$) to argue that the distribution of the key sk_1 in *Game 0* is statistically close to choosing a fresh honest key. In particular, the lemma tells us that $A_0 A_{-1}[\vec{r}_1^\top | \cdots | \vec{r}_{m-1}^\top] \overset{\text{stat}}{\approx} [\vec{r}^\top | \mathbf{0} \cdots]$ for a uniformly random $\vec{r} \in \mathbb{F}_q^n$. So the exponent of the key sk_1 in *Game 0* is

$$A_0 A_{-1} S_{-1} = A_0 A_{-1} \vec{r}_1^\top \vec{w}_1 + \vec{1}^\top \vec{t} \quad \overset{\text{stat}}{\approx} \quad \vec{r}^\top \vec{w} + \vec{1}^\top \vec{t}$$

and hence statistically indistinguishable from that of a random honest key as in *Game' 0*.

□

□

Figure 4: Intermediate Hybrids for Lemma C.3: From Game i to GameCor' $i+1$

Lemma C.3. *For $i \in \{0, \dots, q_{\mathsf{sk}}\}$: Game $i \overset{\text{comp}}{\approx}$ GameCor' $i+1$.*

Proof. For the proof of the lemma, we introduce two additional intermediate games (see Figure 4). Firstly, we define *GameAlt i*. In this game, the initial secret key sk_{-1} is a random *mid key* with exponent matrix

$$S \stackrel{\text{def}}{=} \begin{bmatrix} | & | \\ \vec{r}_1^\top & \vec{r}_2^\top \\ | & | \end{bmatrix} \begin{bmatrix} - & \vec{w}_1 & - \\ - & \vec{w}_2 & - \end{bmatrix} + \begin{bmatrix} \vec{1}^\top \end{bmatrix} \begin{bmatrix} & \vec{t} & \end{bmatrix}$$

The key update matrices $A_1, \dots, A_{i-2}$ are chosen honestly, the update A_{i-1} is programmed to annihilate $m-3$ random vectors, the update A_i is programmed to annihilate $\vec{r}_2$ along with $m-4$ random vectors, and the update matrix A_{i+1} is programmed to annihilate a single random vector. The initial ciphertext and ciphertext update matrices are chosen as in *Game i*.

Claim C.4. *Game i $\overset{\text{comp}}{\approx}$ GameAlt i*

Proof. We show a reduction from the $(k=1)$-*extended rank hiding assumption* (Definition B.10, Lemma B.11). The challenger gets a challenge $\mathbf{h}^{W'}$ and $\vec{p}$, where $W' \xleftarrow{\$} \mathsf{Rk}_x(\mathbb{F}_q^{(m-2)\times m})$ is either of rank $x=1$ or of rank $x=m-2$ and $\vec{p} \in \ker(W')$. Let us label the rows of W' by $\vec{w}_2, \dots, \vec{w}_{m-1}$. Choose $\vec{w}_1 \xleftarrow{\$} (\vec{p})^\perp$ and $\vec{t} \xleftarrow{\$} \mathbb{F}_q^m$. Set $\mathsf{prms} = (\mathbf{g}^{\vec{p}}, \mathbf{h}^{\vec{w}_1})$, $pk = e(\mathbf{g}, \mathbf{h})^{\vec{t}\cdot\vec{p}}$ and $\mathsf{sk}_{-1} = \mathbf{h}^S$ where

$$S = \begin{bmatrix} | & & | \\ \vec{r}_1^\top & \cdots & \vec{r}_{m-1}^\top \\ | & & | \end{bmatrix} \begin{bmatrix} - & \vec{w}_1 & - \\ & \cdots & \\ - & \vec{w}_{m-1} & - \end{bmatrix} + \begin{bmatrix} \vec{1}^\top \end{bmatrix} \begin{bmatrix} & \vec{t} & \end{bmatrix}$$

for uniformly random $\vec{r}_1, \dots, \vec{r}_{m-1}$ (note: the challenger can do this efficiently given $\mathbf{h}^{W'}$ without knowing W'). Create a "low" ciphertext by choosing a random vector $\vec{c} \xleftarrow{\$} (\vec{w}_1)^\perp$. Run the rest of *Game i* correctly as a challenger, where the key-update A_{i-1} is programmed to annihilate $\vec{r}_3, \dots, \vec{r}_{m-1}$, the key-update A_i is programmed to annihilate $\vec{r}_2$ and $m-4$ random vectors and the key update A_{i+1} update is programmed to annihilate a single random vector.

If W' is of rank $x = m-2$, then it is easy to see that the above distribution is that of *Game i*.

If W' is of rank $x=1$ then we claim that the above is distribution is that of *GameAlt i*. Firstly, we can write $\vec{w}_3 = \mu_3\vec{w}_2, \dots, \vec{w}_{m-1} = \mu_{m-1}\vec{w}_2$ for some scalars $\mu_3, \dots, \mu_{m-1}$ in $\mathbb{F}_q$. Letting $\vec{\mu} = (1, \mu_3, \dots, \mu_{m-1}) \in \mathbb{F}_q^{m-2}$, we can write

$$S = \begin{bmatrix} | & | \\ \vec{r}_1^\top & \vec{r}_2{}'^\top \\ | & | \end{bmatrix} \begin{bmatrix} - & \vec{w}_1 & - \\ - & \vec{w}_2 & - \end{bmatrix} + \begin{bmatrix} \vec{1}^\top \end{bmatrix} \begin{bmatrix} & \vec{t} & \end{bmatrix} \text{ for } \begin{bmatrix} \vec{r}_2{}'^\top \end{bmatrix} = \begin{bmatrix} | & & | \\ \vec{r}_2^\top & \cdots & \vec{r}_{m-1}^\top \\ | & & | \end{bmatrix} \begin{bmatrix} \vec{\mu}^\top \end{bmatrix}$$

So sk_{-1} is a random *mid* key as in *GameAlt i*. Moreover, the vectors $\vec{r}_3, \dots, \vec{r}_{m-1}$ are random and independent of $S, \vec{w}_1, \vec{w}_2, \vec{r}_1, \vec{r}_2{}'$. Therefore, the update matrix A_{i-1} is programmed to annihilate $m-3$ random and independent vectors $\vec{r}_3, \dots, \vec{r}_{m-1}$ as in *GameAlt i*. Finally, the update A_i is programmed to annihilates $\vec{r}_2$ (along with $m-4$ random vectors), which is equivalent to being programmed to annihilate $\vec{r}_2'$ since $\mathsf{span}(D_{i-1}\vec{r}_2{}') = \mathsf{span}(D_{i-1}\vec{r}_2)$ where D_{i-1} is the product of all update matrices prior to A_i. So we see that the, when W' is of rank 1 then the choice of sk_{i-1} and all the update matrices is distributed correctly as in *GameAlt i*. Hence an attacker that distinguishes *Game i* and *GameAlt i* breaks rank-hiding. □

We now introduce a second intermediate game called *Game2Alt i* where the initial secret key sk_{-1} has an exponent of the form

$$S_{-1} \stackrel{\text{def}}{=} \begin{bmatrix} | & \cdots & | \\ \vec{r}_1^\top & \cdots & \vec{r}_{m-2}^\top \\ | & \cdots & | \end{bmatrix} \begin{bmatrix} - & \vec{w}_1 & - \\ & \cdots & \\ - & \vec{w}_{m-2} & - \end{bmatrix} + \begin{bmatrix} \vec{1}^\top \end{bmatrix} \begin{bmatrix} & \vec{t} & \end{bmatrix}$$

so that the rows are chosen randomly from the $m-2$ dimensional affine space $\vec{t} + \mathsf{span}(\vec{w}_1, \ldots, \vec{w}_{m-2})$. The key update matrices $A_1, \ldots, A_{i-2}$ are chosen honestly, the update A_{i-1} is programmed to annihilate $m-3$ random vectors, the update A_i is programmed to annihilate the vectors $\vec{r}_2, \ldots, \vec{r}_{m-2}$, and the update matrix A_{i+1} is programmed to annihilate a single random vector. The initial ciphertext and ciphertext update matrices are chosen as in *Game i*.

We now show (in 2 steps) that *Game2Alt i* is computationally indistinguishable from *GameAlt i* and from *GameCor* $(i+1, 0)$, which completes the proof of Lemma C.3

Claim C.5. *GameAlt i* $\stackrel{\text{comp}}{\approx}$ *Game2Alt i.*

Proof. We show a reduction from the $(k = 1)$-*extended rank hiding assumption* (Definition B.10, Lemma B.11). The challenger gets a challenge $\mathbf{h}^{W'}$ and $\vec{p}$, where $W' \stackrel{\$}{\leftarrow} \mathsf{Rk}_x(\mathbb{F}_q^{(m-3)\times m})$ is either rank $x = 1$ or rank $x = (m-3)$ and $\vec{p} \in \mathsf{ker}(W')$. Let us label the rows of W' by $\vec{w}_2, \ldots, \vec{w}_{m-2}$. Choose $\vec{w}_1 \stackrel{\$}{\leftarrow} (\vec{p})^\perp$ and $\vec{t} \stackrel{\$}{\leftarrow} \mathbb{F}_q^m$. Set $\mathsf{prms} = (\mathbf{g}^{\vec{p}}, \mathbf{h}^{\vec{w}_1})$, $pk = e(\mathbf{g}, \mathbf{h})^{\vec{t}\cdot\vec{p}}$ and $\mathsf{sk}_{-1} = \mathbf{h}^S$ where

$$S = \begin{bmatrix} | & & | \\ \vec{r}_1^\top & \cdots & \vec{r}_{m-2}^\top \\ | & & | \end{bmatrix} \begin{bmatrix} - & \vec{w}_1 & - \\ & \cdots & \\ - & \vec{w}_{m-2} & - \end{bmatrix} + \begin{bmatrix} \vec{1}^\top \end{bmatrix} \begin{bmatrix} & \vec{t} & \end{bmatrix}$$

for uniformly random $\vec{r}_1, \ldots, \vec{r}_{m-2}$ (note: the challenger can do this efficiently given $\mathbf{h}^{W'}$ without knowing W'). Create a "low" ciphertext by choosing a random vector $\vec{c} \stackrel{\$}{\leftarrow} (\vec{w}_1)^\perp$. The key-update A_{i-1} is programmed to annihilate $m-3$ random vectors, the key-update A_i is programmed to annihilate $\vec{r}_2, \ldots, \vec{r}_{m-2}$ and the key update A_{i+1} update is programmed to annihilate a single random vector.

If W' is of rank $x = m-3$, then it is easy to see that the above distribution is that of *Game2Alt i*.

If W' is of rank $x = 1$ then we claim that the above distribution is that of *GameAlt i*. This follows since we can write

$$S = \begin{bmatrix} | & | \\ \vec{r}_1^\top & \vec{r}_2{}'^\top \\ | & | \end{bmatrix} \begin{bmatrix} - & \vec{w}_1 & - \\ - & \vec{w}_2 & - \end{bmatrix} + \begin{bmatrix} \vec{1}^\top \end{bmatrix} \begin{bmatrix} & \vec{t} & \end{bmatrix} \quad \text{for} \quad \begin{bmatrix} \vec{r}_2{}'^\top \end{bmatrix} = \begin{bmatrix} | & & | \\ \vec{r}_2^\top & \cdots & \vec{r}_{m-2}^\top \\ | & & | \end{bmatrix} \begin{bmatrix} \vec{\mu}^\top \end{bmatrix}$$

and some vector $\vec{\mu}^\top \in \mathbb{F}_q^{m-3}$. So sk_{-1} is a random *mid key* as in *GameAlt i*. Moreover, an update A_i which is programmed to annihilate $\vec{r}_2, \ldots, \vec{r}_{m-2}$ is equivalent to being programmed to annihilate $\vec{r}_2{}'$ along with $m-4$ random vectors since $\mathsf{span}(\vec{r}_2, \ldots, \vec{r}_{m-2}) = \mathsf{span}(\vec{r}_2{}', \vec{r}_3, \ldots, \vec{r}_{m-2})$ where $\vec{r}_3, \ldots, \vec{r}_{m-2}$ are random an independent of sk_{-1} or any of the previous updates. Therefore sk_{-1} and all of the update matrices are distributed as in *GameAlt i*. $\square$

Claim C.6. *Game2Alt i* $\stackrel{\text{comp}}{\approx}$ *GameCor' i* $+ 1$.

Proof. We show a reduction from the $(k=1)$-*extended rank hiding assumption* (Definition B.10, Lemma B.11). The reduction gets a challenge $\mathbf{h}^{W'}$ and $\vec{p}, \vec{c}$, where $W' \xleftarrow{\$} \mathsf{Rk}_x(\mathbb{F}_q^{2\times m})$ is either rank $x=1$ or rank $x=2$ and $\vec{p}, \vec{c} \in \mathsf{ker}(W')$. Let us label the rows of W' by $\vec{w}_1, \vec{w}_2$. The reduction chooses $\vec{w}_3, \ldots, \vec{w}_{m-1} \xleftarrow{\$} (\vec{p})^\perp$, $\vec{t} \xleftarrow{\$} \mathbb{F}_q^m$ and sets $\mathsf{prms} = (\mathbf{g}^{\vec{p}}, \mathbf{h}^{\vec{w}_1})$, $pk = e(\mathbf{g}, \mathbf{h})^{\vec{t}\cdot\vec{p}}$ and $\mathsf{sk}_{-1} = \mathbf{h}^S$ where

$$S = \begin{bmatrix} | & & | \\ \vec{r}_1^\top & \cdots & \vec{r}_{m-1}^\top \\ | & & | \end{bmatrix} \begin{bmatrix} - & \vec{w}_1 & - \\ & \cdots & \\ - & \vec{w}_{m-1} & - \end{bmatrix} + \begin{bmatrix} \vec{1}^\top \end{bmatrix} \begin{bmatrix} & \vec{t} & \end{bmatrix}$$

for uniformly random $\vec{r}_1, \ldots, \vec{r}_{m-1}$. (Note: the reduction can do this efficiently given $\mathbf{h}^{W'}$ without knowing W'). Lastly, the reduction creates a "low" ciphertext using the vector $\vec{c}$ that comes from the challenge. The reduction does everything else as in *GameCor' $i+1$* where the key-update A_{i-1} is programmed to annihilate $m-3$ random vectors, the key-update A_i is programmed to annihilate $\vec{r}_3, \ldots, \vec{r}_{m-1}$ and the key update A_{i+1} update is programmed to annihilate $\vec{r}_2$.

If W' is of rank $x=2$, then it is easy to see that the above distribution is that of *GameCor' $i+1$*. Notice that the key and ciphertext basis are random *correlated* basis with $\vec{c} \in (\vec{w}_1, \vec{w}_2)^\perp$.

If W' is of rank $x=1$ then we claim that the above distribution is that of *Gam2Alt i*. This follows since the exponent of sk_{-1} can be written as

$$S = \begin{bmatrix} | & | & & | \\ \vec{r}_1{}'^\top & \vec{r}_3^\top & \cdots & \vec{r}_{m-1}^\top \\ | & | & & | \end{bmatrix} \begin{bmatrix} - & \vec{w}_1 & - \\ - & \vec{w}_3 & - \\ & \cdots & \\ - & \vec{w}_{m-1} & - \end{bmatrix} + \begin{bmatrix} \vec{1}^\top \end{bmatrix} \begin{bmatrix} & \vec{t} & \end{bmatrix}$$

where $\vec{r}_1{}'^\top = \mu_1 \vec{r}_1^\top + \mu_2 \vec{r}_2^\top$ for some scalars μ_1, μ_2. Note that $\vec{r}_1{}'$ is therefore uniformly random and independent of $\vec{r}_2$. By relabeling, this has the same distribution as the initial key in *Game2Alt i*. Moreover, the update A_i is programmed to annihilate the $m-3$ vectors $\vec{r}_3, \ldots, \vec{r}_{m-1}$ in the initial secret key, while the update A_{i+1} annihilates the vector $\vec{r}_2$ which is random and independent of the initial secret key. Therefore the initial secret key and all of the secret key updates are distributed as in *Game2Alt i*. □

Putting Claim C.4, Claim C.5 and Claim C.6 together, we complete the proof of the lemma. □

Lemma C.7. *If* $\ell \leq (n-3m+6)\log(q) - \omega(\log(\lambda))$ *then* GameCor' $i+1 \overset{\text{stat}}{\approx}$ GameCor $(i+1, 0)$.

Proof. The only difference between the two games is the joint distribution on the initial ciphertext ct_1 and ciphertext-update matrix B_1. In *GameCor' $i+1$* the ciphertext ct_1 is chosen as a low ciphertext and the update B_1 is honest. In *GameCor* $(i+1, 0)$, we first choose a high ciphertext ct_{-1} with the exponent

$$C_{-1} \overset{\text{def}}{=} \begin{bmatrix} | & \cdots & | \\ \vec{u}_1^\top & \cdots & \vec{u}_{m-1}^\top \\ | & \cdots & | \end{bmatrix} \begin{bmatrix} - & \vec{c}_1 & - \\ & \cdots & \\ - & \vec{c}_{m-1} & - \end{bmatrix}$$

We then choose updates: B_{-1} programmed to annihilate $(\vec{u}_3, \ldots, \vec{u}_{m-1})$ yielding a mid ciphertext ct_0, B_0 programmed to annihilate $\vec{u}_2$ and $m-4$ random vectors yielding a low ciphertext ct_1, and the update B_1 programmed to annihilate a single random vector. We claim that ct_1, B_1 are distributed the same way in the two games. The proof exactly follows that of Lemma C.2. □

□

Figure 5: Intermediate Hybrids for Lemma C.8: From GameCor $(i+1, j)$ to GameSuperCor $(i+1, j+1)$

Lemma C.8. *GameCor* $(i+1, j) \overset{\text{comp}}{\approx}$ *GameSuperCor* $(i+1, j+1)$.

Proof. The proof of this lemma is analogous to the proof of Lemma C.3. We introduce two additional intermediate games (see Figure 5) called *GameCorAlt* $(i+1, j)$ and *GameCor2Alt* $(i+1, j)$. In all these games the key basis $(\vec{w}_1, \ldots, \vec{w}_{m-1}) \in (\vec{p})^{\perp}$ and the ciphertext basis $(\vec{c}_1, \ldots, \vec{c}_{m-1}) \in (\vec{w}_1)^{\perp}$ are *correlated* with $\vec{w}_2 \in (\vec{c}_1)^{\perp}$.

Firstly, we define *GameCorAlt* $(i+1, j)$ where the initial ciphertext ct_{-1} is a random *mid ciphertext* using exponent matrix

$$C \overset{\text{def}}{=} \begin{bmatrix} | & | \\ \vec{u}_1^{\top} & \vec{u}_2^{\top} \\ | & | \end{bmatrix} \begin{bmatrix} - & \vec{c}_1 & - \\ - & \vec{c}_2 & - \end{bmatrix}$$

for uniformly random $\vec{u}_1, \vec{u}_2$. The initial ciphertext-update matrices $\ldots, B_{j-2}$ are chosen honestly, the update B_{j-1} is programmed to annihilate $m-3$ random vectors, the update B_j is programmed to annihilate $\vec{u}_2$ along with $m-4$ random vectors, and the update matrix B_{j+1} is programmed to annihilate a single random vector. The initial secret key and all key update matrices are chosen as in *GameCor* $(i+1, j)$.

Claim C.9. *GameCor* $(i+1, j) \overset{\text{comp}}{\approx}$ *GameCorAlt* $(i+1, j)$.

Proof. We show a reduction from the $(k=1)$-*extended rank hiding assumption* (Definition B.10, Lemma B.11). The challenge gets a challenge $\mathbf{g}^{C'}$ and $\vec{w}_1$, where $C' \xleftarrow{\$} \mathsf{Rk}_x(\mathbb{F}_q^{(m-2)\times m})$ is either of rank $x = 1$ or $x = m-2$ and $\vec{w}_1 \in \mathsf{ker}(C')$. Let us label the rows of C' by $\vec{c}_2, \ldots, \vec{c}_{m-1}$. Choose $\vec{c}_1, \vec{p} \xleftarrow{\$} (\vec{w}_1)^{\perp}$ and $\vec{t} \xleftarrow{\$} \mathbb{F}_q^m$. Set $\mathsf{prms} = (\mathbf{g}^{\vec{p}}, \mathbf{h}^{\vec{w}_1})$, $pk = e(\mathbf{g}, \mathbf{h})^{\vec{t}\cdot\vec{p}}$. The initial ciphertext $\mathsf{ct}_{-1}$is chosen so that the exponent matrix in the first component $\mathbf{g}^C$ is

$$C = \begin{bmatrix} | & & | \\ \vec{u}_1^{\top} & \cdots & \vec{u}_{m-1}^{\top} \\ | & & | \end{bmatrix} \begin{bmatrix} - & \vec{c}_1 & - \\ & \cdots & \\ - & \vec{c}_{m-1} & - \end{bmatrix}$$

for uniformly random $\vec{u}_1, \ldots, \vec{u}_{m-1}$ (note: the challenger can do this efficiently given $\mathbf{g}^{C'}$ without knowing C'). The ciphertext-update matrices are chosen as in *GameCor* $(i+1, j)$ where B_{j-1} is programmed to annihilate $\vec{u}_3, \ldots, \vec{u}_{m-1}$, B_j is programmed to annihilate $\vec{u}_2$ and $m-4$ random vectors and B_{j+1} is programmed to annihilate a single random vector.

The initial secret key is chosen by first sampling $\vec{w}_2, \ldots, \vec{w}_{m-1}$ so that, along with $\vec{w}_1$, they form a random *correlated* basis of $(\vec{p})^{\perp}$, having $\vec{w}_2 \in (\vec{c}_1)^{\perp}$. This basis is then used to sample the initial *high* secret key sk_{-1}. The secret key updates are chosen as in *GameCor* $(i+1, j)$.

If C' is of rank $x = m-2$, then the above distribution is just that of *GameCor* $(i+1, j)$. On the other hand, if C' is of rank $x = 1$, we argue that the above distribution is that of *GameCorAlt* $(i+1, j)$. The argument mirrors that of Claim C.4.

□

We now introduce a second intermediate game called *GameCor2Alt* $(i+1, j)$ where the initial ciphertext ct_{-1} is chosen so that the exponent in the first component $\mathbf{g}^C$ is

$$C = \begin{bmatrix} | & \cdots & | \\ \vec{u}_1^{\top} & \cdots & \vec{u}_{m-2}^{\top} \\ | & \cdots & | \end{bmatrix} \begin{bmatrix} - & \vec{c}_1 & - \\ & \cdots & \\ - & \vec{c}_{m-2} & - \end{bmatrix}$$

for uniformly random $\vec{u}_1, \ldots, \vec{u}_{m-2}$. The initial ciphertext-update matrices $\ldots, B_{j-2}$ are chosen honestly, the update B_{j-1} is programmed to annihilate $m-3$ random vectors, the update B_j is programmed to annihilate the $m-3$ vectors $\vec{u}_2, \ldots, \vec{u}_{m-2}$, and the update matrix B_{j+1} is programmed to annihilate a single random vector. The initial secret key and all key update matrices are chosen as in *GameCor* $(i+1, j)$.

We now show (in 2 steps) that *GameCor2Alt* $(i+1, j)$ is computationally indistinguishable from *GameAlt* $(i+1, j)$ and from *GameSuperCor* $(i+1, j+1)$, which completes the proof of Lemma C.8.

Claim C.10. *GameCorAlt* $(i+1, j) \overset{\text{comp}}{\approx}$ *GameCor2Alt* $(i+1, j)$.

Proof. We show a reduction from the $(k = 1)$-*extended rank hiding assumption* (Definition B.10, Lemma B.11). The challenger gets a challenge $\mathbf{g}^{C'}$ and $\vec{w}_1$, where $C' \xleftarrow{\$} \mathsf{Rk}_x(\mathbb{F}_q^{(m-3\times m)})$ is either rank $x = 1$ or rank $x = m-3$ and $\vec{w}_1 \in \mathsf{ker}(C')$. Let us label the rows of C' by $\vec{c}_2, \ldots, \vec{c}_{m-2}$. Choose $\vec{c}_1, \vec{p} \xleftarrow{\$} (\vec{w}_1)^\perp$ and $\vec{t} \xleftarrow{\$} \mathbb{F}_q^m$. Set $\mathsf{prms} = (\mathbf{g}^{\vec{p}}, \mathbf{h}^{\vec{w}_1})$, $pk = e(\mathbf{g}, \mathbf{h})^{\vec{t}\cdot\vec{p}}$. The initial ciphertext ct_{-1} is chosen so that the exponent in the first component $\mathbf{g}^C$ is

$$C = \begin{bmatrix} | & & | \\ \vec{u}_1^\top & \cdots & \vec{u}_{m-2}^\top \\ | & & | \end{bmatrix} \begin{bmatrix} - & \vec{c}_1 & - \\ & \cdots & \\ - & \vec{c}_{m-2} & - \end{bmatrix}$$

for random $\vec{u}_1, \ldots, \vec{u}_{m-2} \in \mathbb{F}_q^m$ (note: the challenger can do this efficiently given $\mathbf{g}^{C'}$ without knowing C'). The ciphertext-update B_{j-1} is programmed to annihilate $m-3$ random vectors, B_j is programmed to annihilate $\vec{u}_2, \ldots, \vec{u}_{m-2}$, and B_{j+1} is programmed to annihilate a single random vector.

The initial secret key is chosen by first sampling $\vec{w}_2, \ldots, \vec{w}_{m-1}$ so that, along with $\vec{w}_1$, they form a random *correlated* basis of $(\vec{p})^\perp$, having $\vec{w}_2 \in (\vec{c}_1)^\perp$. This basis is then used to sample the initial *high* secret key sk_{-1}. The secret key updates are chosen as in *GameCor* $(i+1, j)$.

If C' is of rank $x = m-3$, then the above distribution is just that of *GameCor2Alt* $(i+1, j)$. On the other hand, if C' is of rank $x = 1$, we argue that the above distribution is that of *GameCorAlt* $(i+1, j)$. The argument mirrors that of Claim C.5. ☐

Claim C.11. *GameCor2Alt* $(i+1, j) \overset{\text{comp}}{\approx}$ *GameSuperCor* $(i+1, j+1)$.

Proof. We show a reduction from the $(k = 1)$-*extended rank hiding assumption* (Definition B.10, Lemma B.11). The challenger gets a challenge $\mathbf{g}^{C'}$ and $\vec{w}_1, \vec{w}_2$, where $C' \xleftarrow{\$} \mathsf{Rk}_x(\mathbb{F}_q^{2\times m})$ is either rank $x = 1$ or rank $x = 2$ and $\vec{w}_1, \vec{w}_2 \in \mathsf{ker}(C')$. Let us label the rows of C' by $\vec{c}_1, \vec{c}_2$. Choose $\vec{p}, \vec{c}_3, \ldots, \vec{c}_{m-1} \xleftarrow{\$} (\vec{w}_1)^\perp$ and $\vec{t} \xleftarrow{\$} \mathbb{F}_q^m$. Set $\mathsf{prms} = (\mathbf{g}^{\vec{p}}, \mathbf{h}^{\vec{w}_1})$, $pk = e(\mathbf{g}, \mathbf{h})^{\vec{t}\cdot\vec{p}}$. The initial ciphertext ct_{-1} is chosen so that the exponent in the first component $\mathbf{g}^C$ is

$$C = \begin{bmatrix} | & & | \\ \vec{u}_1^\top & \cdots & \vec{u}_{m-1}^\top \\ | & & | \end{bmatrix} \begin{bmatrix} - & \vec{c}_1 & - \\ & \cdots & \\ - & \vec{w}_{c-1} & - \end{bmatrix}$$

for random $\vec{u}_1, \ldots, \vec{u}_{m-2} \in \mathbb{F}_q^m$ (note: the challenger can do this efficiently given $\mathbf{g}^{C'}$ without knowing C'). The ciphertext-update B_{j-1} is programmed to annihilate $m-3$ random vectors, B_j is programmed to annihilate $\vec{u}_3, \ldots, \vec{u}_{m-1}$, and B_{j+1} is programmed to annihilate $\vec{u}_2$.

The initial secret key is chosen by sampling $\vec{w}_3, \ldots, \vec{w}_{m-1}$ so that, along with $\vec{w}_1, \vec{w}_2$, they form a basis of $(\vec{p})^\perp$. This basis is then used to sample the initial *high* secret key sk_{-1}. The secret key updates are chosen as in *GameCor* $(i+1, j)$.

If C' is of rank $x = 2$, then the above distribution is just that of *GameSuperCor* $(i+1, j+1)$. Notice that, in this case, the key and ciphertext bases are super-correlated with $\vec{c}_1, \vec{c}_2 \in (\vec{w}_1, \vec{w}_2)^{\perp}$. On the other hand, if C' is of rank $x = 1$, we argue that the above distribution is that of *GameCor2Alt* $(i+1, j)$. The argument mirrors that of Claim C.6. □

Putting Claim C.9, Claim C.10 and Claim C.11 together, we complete the proof of the lemma.

□

Lemma C.12. *If* $\ell \leq \min(m/6 - 1, n - 3m + 6)\log(q) - \omega(\log(\lambda))$ *then*

$$\text{GameSuperCor}(i+1, j+1) \overset{\text{stat}}{\approx} \text{GameCor}(i+1, j+1).$$

Proof. There are two differences between *GameSuperCor* and *GameCor*. The first difference lies in wether the ciphertext and key bases are correlated or super-correlated. The second difference lies in the regime of ciphertext updates. We define an intermediate game, *GameCorInter* $(i+1, j+1)$, in which the ciphertext bases are (only) *correlated*, but the regime of ciphertext updates follows that of *GameSuperCor* $(i+1, j+1)$.

Claim C.13. *If* $\ell \leq (m/6 - 1)\log(q) - \omega(\log(\lambda))$, *then*

$$\textit{GameSuperCor}(i+1, j+1) \overset{\text{stat}}{\approx} \textit{GameCorInter}(i+1, j+1).$$

Proof. This follows by the *orthogonality hiding lemma* (Lemma B.9). Notice that, once we fix $\vec{p}, \vec{w}_1$ and $\vec{c}_1$ in *GamrCor'*, we can think of $\vec{w}_2 \xleftarrow{\$} (\vec{c}_1, \vec{p})^{\perp}$ and $\vec{c}_2 \xleftarrow{\$} (\vec{w}_1)^{\perp}$ as two independent sources. Moreover, we claim that the only leaked-upon values in the two games above that are related to $\vec{w}_2$ are the key updates A_i, A_{i+1} and the secret key sk_{i+1}. Therefore only *three* time periods contain relevant information about $\vec{w}_2$ This is because the initial *high* key sk_{-1} has rows from the full space $(\vec{p})^{\perp}$ (and hence does not depend on $\vec{w}_2$) and the updates $A_{-1}, \ldots, A_{i-1}$ do not depend on the key at all. (On the other hand the secret key sk_{i+1} has the span of $\vec{w}_1, \vec{w}_2$ in the exponent, and the updates A_i, A_{i+1} annihilate vectors that are correlated to sk_{i+1} and hence also to $\vec{w}_2$). Lastly the keys $\mathsf{sk}_{i+2}, \ldots$ are random *low* keys and the update $A_{i+2}, \ldots$ are chosen honestly and hence have no information about $\vec{w}_2$. Therefore, the leakage on $\vec{w}_2$ is bounded by 3ℓ. Similarly, the only leaked-upon values in the two games above that are related to $\vec{c}_2$ are the ciphertext updates B_j, B_{j+1} and the ciphertext ct_{j+1}. Hence the leakage on $\vec{c}_2$ is bounded by 3ℓ as well. Lastly, since the secret key and ciphertext leak independently, if we condition on the leakage Z observed by the attacker, the distributions of $\vec{c}_2$ and $\vec{w}_2$ are independent. We get $\widetilde{\mathbf{H}}_{\infty}(\vec{w}_2 \mid Z) \geq (m-2)\log(q) - 3\ell, \widetilde{\mathbf{H}}_{\infty}(\vec{c}_2 \mid Z) \geq (m-1)\log(q) - 3\ell$. Since *GameSuperCor* is equivalent to *GameCorInter* is we condition on the event $\langle \vec{w}_2, \vec{c}_2 \rangle = 0$, we can apply the orthogonality hiding lemma (Lemma B.9) to bound the statistical distance between these games. In particular, the games *GameSuperCor* and *GameCorInter* are statistically indistinguishable as long as $\ell \leq (m/6 - 1)\log(q) - \omega(\log(\lambda))$. □

We now turn to changing the regime of ciphertext updates from *GameCorInter* to *GameCor*. But for this we just use the *reprogramming lemma* (Lemma B.13) on the ciphertext updates. In particular, we rely on the fact that the regime of ciphertext updates between *GameCorInter* and *GameCor* only differs in which updates annihilate additional random vectors (but *not* in how they annihilate the ciphertext exponent vectors $\vec{u}_i$). Therefore, using the reprogramming lemma, we see that the two games are indistinguishable as long as $\ell \leq (n - 3m + 6)\log(q) - \omega(\log(\lambda))$.

□

Lemma C.14. *If* $\ell \leq (n - 3m + 6)\log(q) - \omega(\log(\lambda))$ *then* GameCor $(i+1, q_{\mathsf{ct}} + 1) \overset{\text{comp}}{\approx}$ Game $i+1$.

Proof. There are three differences between the above games: (I) in *GameCor* $(i+1, q_{\mathsf{ct}}+1)$ the ciphertext and key bases are *correlated* ($\langle \vec{c}_2, \vec{w}_1 \rangle = 0$) whereas in *Game* $i+1$ they are *un*correlated, (II) the regime of programmed *key updates* differs between the two games in when and how many random vectors the updates are programmed to annihilate, (III) in *GameCor* $(i+1, q_{\mathsf{ct}}+1)$ the initial ciphertext is *high* while in *Game* $i+1$ it is low (all ciphertext updates are honest in both games).

Firstly, we can ignore (I) since both games are completely independent of the choice of the basis vector $\vec{c}_2$ as there are no mid ciphertexts in either game. In particular, the distribution of the initial low/high ciphertext in the two games does not depend at all on whether the bases are correlated or not.

Secondly, we use the reprogramming lemma (Lemma B.13) to change the regime of key updates from that of *GameCor* $(i+1, q_{\mathsf{ct}}+1)$ to that of *Game* $i+1$. The total number of random vectors that are programmed to be annihilated in either game is $u = m-3$. Therefore, the change is statistically indistinguishable as long as $\ell \leq (n-3m+6)\log(q) - \omega(\log(\lambda))$.

Lastly, we provide a simple reduction from the $(k=1)$-*extended rank hiding assumption* (Definition B.10, Lemma B.11) to distinguishing whether the initial ciphertext is low or high. The reduction gets a challenge $\mathbf{g}^C$ and $\vec{w}$ where $C \xleftarrow{\$} \mathsf{Rk}_x(\mathbb{F}_q^{n\times m})$ is a random matrix of either rank $x = m-1$ or rank $x = 1$, and $\vec{w} \in \mathsf{ker}(C)$. The reduction samples a random $\vec{t} \xleftarrow{\$} \mathbb{F}_q^n$, $\vec{p} \xleftarrow{\$} (\vec{w})^\perp$ and $\vec{w}_2, \ldots, \vec{w}_{m-1} \xleftarrow{\$} (\vec{p})^\perp$. It uses these to create the public parameters $\mathbf{g}^{\vec{p}}, \mathbf{h}^{\vec{w}}$, public key $\mathsf{pk} = e(\mathbf{g}, \mathbf{h})^{\langle \vec{p}, \vec{t}\rangle}$ and the initial high secret key $\mathbf{h}^S$. For the initial ciphertext, it just uses $\mathsf{ct}^{(1)} = \mathbf{g}^C$ and creates the matching second component $\mathsf{ct}^{(2)}$ efficiently using $\mathsf{ct}^{(1)}, \mathbf{msg}_b, \vec{t}$. It runs the rest of the game by choosing the key updates as specified in *Game* $i+1$ and all the ciphertext updates honestly. It's easy to see that if the challenge $\mathbf{g}^C$ is of rank $x = m-1$ then this the same as *GameCor* $(i+1, q_{\mathsf{ct}}+1)$ with the modified key updates as above, while if the challenge is of rank $x = 1$ then this is just *Game* $i+1$.

□

Lemma C.15. Game $q_{\mathsf{sk}}+1 \overset{\text{comp}}{\approx}$ GameFinal.

Proof. We define several hybrid distributions for choosing the initial ciphertext ct. Recall that we can think of ct as consisting of n rows where each row is a ciphertext under the "simple" (un-updatable) encryption scheme. We will consider the following distributions on the ciphertext-rows:

1. Low Encryptions of $\mathbf{msg}$: The ciphertext-row is of the form $(\mathbf{g}^{\vec{c}}, e(\mathbf{g}, \mathbf{h})^{\langle \vec{c}, \vec{t}\rangle}\mathbf{msg})$ where $\vec{c} = u\vec{c}_1$ for a random $u \xleftarrow{\$} \mathbb{F}_q$ and the ciphertext-basis vector $\vec{c}_1$.
2. Mid Encryptions of $\mathbf{msg}$: The ciphertext-row is of the form $(\mathbf{g}^{\vec{c}}, e(\mathbf{g}, \mathbf{h})^{\langle \vec{c}, \vec{t}\rangle}\mathbf{msg})$ where $\vec{c} = u_1\vec{c}_1 + u_2\vec{c}_2$ for random $u_1, u_2 \xleftarrow{\$} \mathbb{F}_q$ and the ciphertext-basis vectors $\vec{c}_1, \vec{c}_2$.

In *Game* $q_{\mathsf{sk}}+1$, the initial ciphertext ct has all of its n rows chosen as random *low encryptions of* $\mathbf{msg}_b$. In *GameFinal*, the initial ciphertext ct has all of its n rows chosen as random *low encryptions of* $1_{\mathbb{G}_T}$.

We define hybrid games *GameHyb* $i = 0, \ldots, n$, where the first i rows of the initial ciphertext ct are chosen as random *low encryptions of* $1_{\mathbb{G}_T}$ and the rest are random *low encryptions of* $\mathbf{msg}_b$. Note that *GameHyb* 0 is really just *Game* $q_{\mathsf{sk}}+1$ and *GameHyb* n is really just *GameFinal*.

We also define the hybrid game *GameHybMid* $i = 0, \ldots, n-1$ where the first i rows of the initial ciphertext ct are random low encryptions of $1_{\mathbb{G}_T}$, the row $i+1$ is a random *mid* encryption of $\mathbf{msg}_b$, and the rest of the rows are random low encryptions of $\mathbf{msg}_b$.

In all these games, the ciphertext updates are all honest, the secret key is a high key and the key updates are all honest (as in *Game* $q_{\mathsf{sk}}+1$).

Claim C.16. *For* $i = 0, \ldots, n-1$*:* *GameHyb* $i \overset{\text{comp}}{\approx}$ *GameHybMid* i.

Proof. We show a reduction from the $(k=1)$-*extended rank hiding assumption* (Definition B.10, Lemma B.11) to distinguishing whether the $(i+1)$ row is a low or mid encryption of $\mathbf{msg}_b$. The reduction gets a challenge $\mathbf{g}^{C'}$ and $\vec{w}$ where $C' \xleftarrow{\$} \mathsf{Rk}_x(\mathbb{F}_q^{2\times m})$ is a random matrix of either rank $x = 2$ or rank $x = 1$, and $\vec{w} \in \ker(C)$. Let us label the rows of C' by $\vec{c}_1, \vec{c}_2$ The reduction samples a random $\vec{t} \xleftarrow{\$} \mathbb{F}_q^n$, $\vec{p} \xleftarrow{\$} (\vec{w})^\perp$ and $\vec{w}_2, \ldots, \vec{w}_{m-1} \xleftarrow{\$} (\vec{p})^\perp$. It uses these to create the public parameters $\mathbf{g}^{\vec{p}}, \mathbf{h}^{\vec{w}}$, public key $e(\mathbf{g}, \mathbf{h})^{\langle \vec{p}, \vec{t} \rangle}$ and the initial high secret key $\mathbf{h}^S$.

For the initial ciphertext, it sets $\mathsf{ct}^{(1)} = \mathbf{g}^C$ where $C = \vec{u}^T \vec{c}_1 + \vec{e}^\top \vec{c}_2$ for uniformly random $\vec{u} \in \mathbb{F}_q^n$ and the vector $\vec{e} \in \mathbb{F}_q^n$ being it $i+1$ standard basis vector whose $i+1$ coordinate is 1 and all others are 0. In other words each row j of the matrix C can be written as $u_j \vec{c}_1 + e_j \vec{c}_2$ where u_j are random, $e_{i+1} = 1$ and $e_j = 0$ for $j \neq i+1$. The reduction creates the matching second component $\mathsf{ct}^{(2)}$ efficiently using $\mathsf{ct}^{(1)}, \mathbf{msg}_b, \vec{t}$ as either an encryption of $1_{\mathbb{G}_T}$ (for rows $j \leq i$) or the message $\mathbf{msg}_b$ (for rows $j > i$). It runs the rest of the game by choosing the key updates the ciphertext updates honestly. It's easy to see that if the challenge is of rank $x = 1$ then this the same distribution as *GameHyb i* while if the challenge is of rank $x = 2$ then this is the same distribution as *GameHybMid i*. ☐

Claim C.17. *For $i = 0, \ldots, n-1$: GameHybMid $i \overset{\text{stat}}{\approx}$ GameHyb $i+1$.*

Proof. In *GameHybMid i*, the $(i+1)$ row of the initial ciphertext ct is of the form

$$(\mathbf{g}^{u_1 \vec{c}_1 + u_2 \vec{c}_2}, e(\mathbf{g}, \mathbf{h})^{u_1 \beta + u_2 \gamma} \mathbf{msg}_b)$$

where $\beta = \langle \vec{c}_1, \vec{t} \rangle, \gamma = \langle \vec{c}_2, \vec{t} \rangle$. We claim that γ information theoretically "blinds" the message m. That is, we claim that given everything else in the game other than $e(\mathbf{g}, \mathbf{h})^{u_1 \beta + u_2 \gamma} \mathbf{msg}_b$, the value γ looks uniformly random. This is because the only information that's available in the entire game about $\vec{t}$ is the value $\alpha = \langle \vec{t}, \vec{p} \rangle$ given in the public key and the value β which is revealed by the other rows of the ciphertext ct. Since the secret key is a *high key*, the rows of the secret key are chosen uniformly at random from the space

$$\mathsf{span}(\vec{w}_1, \ldots, \vec{w}_{m-1}) + \vec{t} \quad = (\vec{p})^\perp + \vec{t} \quad = \{\vec{s} \in \mathbb{F}_q^m \mid \langle \vec{s}, \vec{p} \rangle = \alpha\}$$

which does not depend on $\vec{t}$ beyond its dependence on α. If $m \geq 3$ then the value $\vec{c}_2$ is linearly independent of $\vec{p}, \vec{c}_1$ (w.o.p.) and hence the value γ is a random and independent of α, β and everything else observed in the game. Therefore, the message $\mathbf{msg}_b$ contained in the $(i+1)$st row of the ciphertext is statistically hidden in *GameHybMid i*. So we may as well replace the $(i+1)$ row from containing $\mathbf{msg}_b$ to just containing $1_{\mathbb{G}_T}$.

We now repeat the same argument as in the proof of Claim C.16 to change the $(i+1)$ row from being a *mid encryption of* $1_{\mathbb{G}_T}$ to being a *low encryption of* $1_{\mathbb{G}_T}$, which gets us to *GameHyb $i+1$*. ☐

Combining these hybrids, we get the statement of the lemma.

C.3 Putting It All Together

Using Lemmata C.1 - C.15 in the sequence of hybrids given by Figure 3, we get the indistinguishability: *Real* $\overset{\text{comp}}{\approx}$ *GameFinal*. Recall that the output of each game includes the view of the attacker $\mathcal{A}$ at the end of the experiment along with the challenger's selection bit b. Since the attacker's guess $\tilde{b}$ at the end of the game can be efficiently computed from the view of the attacker, the predicate ($\mathcal{A}$ wins) $\Leftrightarrow$ ($\tilde{b} \overset{?}{=} b$) can be efficiently computed from the output of each game. In *GameFinal* the view of the attacker is independent of the random bit b and hence we have $\Pr[\mathcal{A} \text{ wins }] = \frac{1}{2}$. Therefore, in the *Real* game, we must have $|\Pr[\mathcal{A} \text{ wins }] - \frac{1}{2}| \leq negl(\lambda)$ since the two games are indistinguishable. This concludes the proof of Theorem 4.1.

☐

D Generalization to k-Linear

In this section, we provide a generalized scheme which we can prove secure under the k-linear assumption for arbitrary choices of k. When $k = 1$, the scheme description and proof coincide exactly with the original.

D.1 Scheme Description

Let k, n, m, d be integer parameters of the system with $n \geq d$.

KeyGen(1^λ) $\to (\mathsf{pk}, \mathsf{sk})$ **:** Sample $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, \mathbf{g}, \mathbf{h}, q) \leftarrow \mathcal{G}(1^\lambda)$ to be the description of a bilinear group of prime order q, with an efficient pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, and generators $\mathbf{g} \in \mathbb{G}_1, \mathbf{h} \in \mathbb{G}_2$.

Choose matrices $P, W \in \mathbb{F}_q^{k \times m}$ at random subject to $\mathsf{rowspan}(P) \perp \mathsf{rowspan}(W)$ and set

$$\mathsf{prms} = ((\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, \mathbf{g}, \mathbf{h}, q), \mathbf{g}^P, \mathbf{h}^W)$$

to be the *public parameters* of the system.[9]

Choose $\vec{t} \xleftarrow{\$} \mathbb{F}_q^m$ and set $\mathsf{pk} := e(\ \mathbf{g}^P\ ,\ \mathbf{h}^{\vec{t}^\top}\ ) = e(\mathbf{g}, \mathbf{h})^{\vec{\alpha}}$ where $\vec{\alpha}^\top = P\vec{t}^\top$. Choose $R \xleftarrow{\$} \mathbb{F}_q^{n \times k}$ and set $\mathsf{sk} := \mathbf{h}^S$, where S is the $n \times m$ matrix given by

$$S \ := \ \begin{bmatrix} \\ R \\ \\ \end{bmatrix} \begin{bmatrix} & W & \end{bmatrix} + \begin{bmatrix} \\ \vec{1}^\top \\ \\ \end{bmatrix} \begin{bmatrix} & \vec{t} & \end{bmatrix}.$$

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

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

(Updatable) Encrypt$_{\mathsf{pk}}$(msg) $\to \mathsf{ct}$ **:** To encrypt $\mathbf{msg} \in \mathbb{G}_T$ under $\mathsf{pk} = e(\mathbf{g}, \mathbf{h})^{\vec{\alpha}}$, choose $U \xleftarrow{\$} \mathbb{F}_q^{n \times k}$ and label its rows $\vec{u}_1, \ldots, \vec{u}_n$. Output $\mathsf{ct} = (\mathsf{ct}^{(1)}, \mathsf{ct}^{(2)})$ where:

$$\mathsf{ct}^{(1)} = \begin{bmatrix} \mathbf{g}^{\vec{u}_1 P} \\ \ldots \\ \mathbf{g}^{\vec{u}_n P} \end{bmatrix} \quad , \quad \mathsf{ct}^{(2)} = \begin{bmatrix} e(\mathbf{g}, \mathbf{h})^{\vec{u}_1 \cdot \vec{\alpha}^\top} \cdot \mathbf{msg} \\ \ldots \\ e(\mathbf{g}, \mathbf{h})^{\vec{u}_n \cdot \vec{\alpha}^\top} \cdot \mathbf{msg} \end{bmatrix}$$

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

$$C = \begin{bmatrix} \\ U \\ \\ \end{bmatrix} \begin{bmatrix} & P & \end{bmatrix} \quad , \quad \vec{z}^\top \ = \ \begin{bmatrix} \\ U \\ \\ \end{bmatrix} \vec{\alpha}^\top + \begin{bmatrix} \\ \vec{1}^\top \\ \\ \end{bmatrix} \mu \ = \ \begin{bmatrix} \\ & C & \\ \\ \end{bmatrix} \begin{bmatrix} \\ t^\top \\ \\ \end{bmatrix} + \begin{bmatrix} \\ \vec{1}^\top \\ \\ \end{bmatrix} \mu$$

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

Decrypt$_{\mathsf{sk}}$(ct) $\to$ **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: $\mathbf{msg} = e(\mathbf{g}, \mathbf{h})^z / e(\ \mathbf{g}^{\vec{c}}\ ,\ \mathbf{h}^{\vec{s}^\top}\ )$.

[9] We can interpret the above as choosing W at random and choosing each row of P at random from $\mathsf{ker}(W)$.

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 $(A)_{i,j} := (A')_{i,j}/(\sum_{l=1}^{n}(A')_{i,l})$.

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 $(B)_{i,j} := (B')_{i,j}/(\sum_{l=1}^{n}(B')_{i,l})$.

If the current ciphertext is $\mathsf{ct} = (\mathbf{g}^C, e(\mathbf{g},\mathbf{h})^{\vec{z}})$, output the updated ciphertext $\mathsf{ct}' := (\mathbf{g}^{BC}, e(\mathbf{g},\mathbf{h})^{B\vec{z}})$.

Theorem D.1. *For any integers $m \geq 7k, n \geq 3m - 7k + 1$ and $d := n - m + 3k$ the scheme* (KeyGen, Encrypt, Decrypt, SKUpdate, CTUpdate) *is an ℓ-CLRS-Friendly Encryption scheme under the k-linear assumption for any $\ell \leq \min\left((m-5k-1)/6, n-3m+7k-1\right)\log(q) - \omega(\log(\lambda))$.*

The argument for correctness is the same as for the original scheme in Section 4 and we sketch the modifications needed to make the proof of security go through below.

D.2 The Generalized Proof (Sketch)

We now give an overview of the modifications to the previous proof necessary to generalize it to the k-linear assumption. The overall structure of the proof and the hybrid games is exactly the same, except that we modify how low/mid/high keys and ciphertexts are defined.

Alternate Key and Ciphertext Distributions. Assume the matrices P, W and the vector $\vec{t}$ are fixed defining $\mathbf{g}^P, \mathbf{h}^W$ and $\mathsf{pk} = e(\mathbf{g},\mathbf{h})^{P\vec{t}^\top}$. Let us label the rows of W by $\vec{w}_1, \ldots, \vec{w}_k$ and let $(\vec{w}_1, \ldots, \vec{w}_{(m-k)})$ be a basis of $\mathsf{ker}(P) = \mathsf{rowspan}(P)^\perp$ and let $(\vec{c}_1, \ldots, \vec{c}_{(m-k)})$ be a basis of $\mathsf{ker}(W) = \mathsf{rowspan}(W)^\perp$. We define the various key distributions on $\mathsf{sk} = \mathbf{h}^S$ the same way as before with

$$S = \begin{bmatrix} | & & | \\ \vec{r}_1^\top & \cdots & \vec{r}_i^\top \\ | & & | \end{bmatrix} \begin{bmatrix} - & \vec{w}_1 & - \\ & \cdots & \\ - & \vec{w}_i & - \end{bmatrix} + \begin{bmatrix} \vec{1}^\top \end{bmatrix} \begin{bmatrix} & \vec{t} & \end{bmatrix} \tag{4}$$

but now we have $i = k$ for ***honest***, $i = 2k$ for ***mid*** and $i = (m-k)$ for ***high*** keys. Similarly, we define the low/mid/high ciphertext distributions the same as before with $\mathsf{ct}^{(1)} = \mathbf{g}^C$ where

$$C = \begin{bmatrix} | & & | \\ \vec{u}_1^\top & \cdots & \vec{u}_j^\top \\ | & & | \end{bmatrix} \begin{bmatrix} - & \vec{c}_1 & - \\ & \cdots & \\ - & \vec{c}_j & - \end{bmatrix} \tag{5}$$

but now we have $j = k$ for ***low***, $j = 2k$ for ***mid*** and $j = (m-k)$ for ***high*** ciphertexts.

By default, we choose the ciphertext and keys basses ***uncorrelated*** with $\vec{w}_{k+1}, \ldots, \vec{w}_{m-k} \xleftarrow{\$} \mathsf{ker}(P)$ and $\vec{c}_1, \ldots, \vec{c}_{m-k} \xleftarrow{\$} \mathsf{ker}(W)$. We say that the bases are ***correlated*** if we instead choose $\vec{c}_1, \ldots, \vec{c}_k \xleftarrow{\$} (\vec{w}_1, \ldots \vec{w}_{2k})^\perp$ and all other vectors as before. We say that the bases are ***super-correlated*** if we instead choose $\vec{c}_1, \ldots, \vec{c}_{2k} \xleftarrow{\$} (\vec{w}_1, \ldots \vec{w}_{2k})^\perp$ and all other vectors as before. The table in Section 5, Figure 1 still accurately summarizes interactions between keys and ciphertexts.

The Hybrids. We also define the various games *Game i*, *GameCor* (i,j), *GameSuperCor* (i,j) analogously as before with appropriate modifications. For example, in *Game i*, the key updates $\ldots, A_{i-2}$ are honest, the update A_i (high to mid) is now programmed to annihilate the $m-3k$ vectors $\vec{r}_{2k+1}, \ldots, \vec{r}_{m-k}$, the update A_{i+1} (mid to low) is programmed to annihilate the k vectors $\vec{r}_{k+1}, \ldots, \vec{r}_{2k}$ along with $m-4k$ random vectors, and the update A_{i+1} is now programmed to annihilate k random vectors. The other game definitions are all analogous.

All of the computational steps are performed analogously, but now under the *k-extended rank hiding assumption* which follows from k-linear. The information theoretic steps in Lemma C.2, Lemma C.7, are also analogous and the new bound becomes: $\ell \leq n - 3m + 7k - 1$.

The only tricky part becomes the information theoretic argument where we change from *super-correlated* bases to just *correlated* bases (the analogue of Claim C.13). We want to switch the condition

$$(\vec{w}_{k+1}, \ldots, \vec{w}_{2k}) \perp (\vec{c}_{k+1}, \ldots, \vec{c}_{2k})$$

from being true to being false. We do this by using an information theoretic argument on each pair $\vec{w}_i, \vec{c}_j$ separately (in any order). Whenever we do so, we think of all other basis components as fixed. That is $\vec{w}_i$ *always* comes from the space orthogonal to $\mathsf{rowspan}(P) \cup \mathsf{span}(\vec{c}_1, \ldots, \vec{c}_{j-1}, \vec{c}_{j+1}, \ldots, \vec{c}_{2k})$ and $\vec{c}_j$ *always* comes from the space orthogonal to $\mathsf{rowspan}(W) \cup \mathsf{span}(\vec{w}_{k+1}, \ldots, \vec{w}_{i-1}, \vec{w}_{i+1}, \ldots, \vec{w}_{2k})$. We are only changing the condition $\vec{w}_i \perp \vec{c}_j$ by applying arguing that leakage on these vectors is bounded by 3ℓ since they only "occur" in *three* time periods, and then applying Lemma B.9. We then get indistinguishability assuming the parameters $\ell \leq ((m - 5k - 1)/6)\log(q) - \omega(\log(\lambda))$.

E CLRS for General Access Structures

We now present a construction of CLRS schemes for general *access structures* over N devices. We only require that the access structure is realizable by some underlying *linear secret sharing scheme* (LSSS). The main advantage is that the attacker can *fully* corrupt some subset of devices, getting their shares in full, and continually leak on all other shares.

Linear Secret Sharing Scheme (LSSS). We follow the terminology and definition of [Bei96]. A linear secret sharing scheme can be expressed in terms of a share-generating matrix $M \in \mathbb{F}_q^{L \times K}$ and a map ρ which associates each row of M with a party (i.e. $\rho(i)$ denotes the party associated with the i^{th} row of M). To share a secret $s \in \mathbb{F}_q$, we choose random values $s_2, \ldots, s_K \in \mathbb{F}_q$, and we define the vector $\vec{s} := (s, s_2, \ldots, s_K)$. Letting $\vec{M}_i$ denote the i^{th} row of M, we produce the share $\langle \vec{M}_i, \vec{s} \rangle \in \mathbb{F}_q$ for each i, which is given to the party $\rho(i)$. A subset of the parties can reconstruct the secret if and only if the vector $(1, 0, \ldots, 0) \in \mathbb{F}_q^K$ is in the span of the rows $\vec{M}_i$ such that party $\rho(i)$ is in the subset (when this occurs, the secret s will be a linear combination of the shares belonging to these parties, and the coefficients of this combination are efficiently computable). We say a set of parties/shares is *authorized* if they can reconstruct the secret, and otherwise we say a set of parties/shares is *unauthorized*. See [Bei96] for various alternate characterizations of the access structures that are achievable using LSSS.

Threshold Encryption. As a stepping stone to achieving CLRS for general access structures, we first observe that we can build an *updatable threshold encryption scheme* for general access structures realizable by LSSS over N parties. This scheme allows us to convert a single secret key into N secret-key shares and achieves the following security guarantee. Suppose an attacker initially specifies a set of secret-key shares that it wishes to corrupt (this is a static corruption model). The attacker is given the key shares belonging to these parties, and gets continual leakage on all the remaining uncorrupted key shares and the ciphertext (individually). As long as the set of corrupted shares is not authorized by the underlying LSSS, the message remains hidden from the attacker. Notice that the attacker *cannot* corrupt the ciphertext and then leak on the key shares afterward (but w.l.o.g. can get the ciphertext in full *at the end*).

To achieve this, we essentially use our basic scheme but share the vector $\vec{t}$ in the exponent of the secret keys according to the share-generating matrix (M, ρ). More precisely, we share each coordinate of $\vec{t}$ (each coordinate is an element of $\mathbb{F}_q$, and can be shared as described above). This means that each party will be given secret key shares which are of the same form as secret keys in our basic updatable encryption scheme, except that in the i^{th} share, the vector $\vec{t}$ has been replaced by the vector $\vec{t}_i$, where each coordinate of $\vec{t}_i$ is a share of the corresponding coordinate of $\vec{t}$ generated by the row $\vec{M}_i$ of the share-generating matrix. Since each secret key share and the ciphertext retain the structure of our core updatable encryption scheme, updates can be done the same as before. To decrypt, parties will reconstruct the vector $\vec{t}$ in the exponent from their secret key shares, and then apply the decryption algorithm for the base scheme.

Security follows from a hybrid argument in which we essentially apply our proof for the basic scheme to change each honestly distributed secret key to a high key. Observe that in the steps of our proof which are used to change a key from honest to high (once the ciphertext is low), the vector $\vec{p}$ is always known to the simulator. This allows the simulator to sample uniformly from $(\vec{p})^{\perp}$ in order to produce high keys.

General CLRS. To get CLRS for general access structures, we additionally observe that we can also split the message to be encrypted into shares by again using the share-generating matrix in the exponent. Each resulting share will be an element of $\mathbb{G}_T$, and can be separately encrypted in the form of a ciphertext for our base scheme. Combining this with the threshold encryption idea, we get a scheme where each party is given a total share containing *both* a secret-key share and a ciphertext share that correspond to its shares of $\vec{t}$ and **msg** under the LSSS. All of these keys and ciphertexts will be of the same form as keys and ciphertexts in the base scheme, and so can be updated in the same way.

We prove the following security property. We again consider an attacker who can corrupt some parties at the outset of the game, and is given all their shares. However, we now require something a bit stronger than simply insisting that the set of corrupted parties be unauthorized. We now also require that adding any one additional party to the set would *still* result in an unauthorized set. After fully corrupting this set of parties, the attacker can get continual leakage from all the remaining uncorrupted parties. We note that this requirement on corruptions is optimal – otherwise the attacker could pool the shares of all the corrupted parties and use them to define a predicate (of some remaining uncorrupted share) which runs the reconstruction procedure and outputs (say) the first bit of the message.

Security is proven via a hybrid argument that changes each uncorrupted ciphertext from an honest encryption of the proper message share to an honest encryption of the identity element. To accomplish this transformation for each ciphertext, we use an inner hybrid over the uncorrupted keys that belong parties *other than* the party who holds the ciphertext that we are currently changing. Each step of this inner hybrid closely follows the proof for the base scheme. We first change the ciphertext to being a low encryption (still encrypting the proper message share). We then proceed to change the key to be a high key. Once this is accomplished, we move on to change the next key to high. Once all of the uncorrupted keys belonging to other parties are distributed as high keys, we argue (similarly to the final lemma for the base scheme) that the message encrypted by the low ciphertext can be changed to the identity. We then rewind the process, returning the keys to honest encryptions and the ciphertext in question to an honest encryption of the identity element. Once we have treated all of the ciphertexts in this way, they are all encryptions of the identity element, and the original message is information-theoretically hidden. One additional subtlety encountered in this proof is that the information-theoretic argument used to change the encrypted message must now be executed in a setting where the corrupted keys as well as the shares belonging to the same party as the ciphertext are *not* high keys, and are given to the attacker. However, since these keys are insufficient to reconstruct the secret, they do not reveal *any* information about the underlying secret vector $\vec{t}$.

E.1 Definition

We now give a formal definition of CLRS for general access structures realizable by some LSSS. We consider sharing a secret among N parties, and we let (M, ρ) denote a share-generating matrix of the LSSS, where $M \in \mathbb{F}_q^{L \times K}$ and $\rho : [L] \to [N]$ is the map associating rows of M with parties. The syntax of the CLRS is as follows:

ShareGen$(1^\lambda, \mathbf{msg}, M, \rho) \to (\mathsf{sh}_1, \mathsf{sh}_2, \ldots, \mathsf{sh}_L)$ **:** The share generation algorithm takes as input the security parameter λ, a secret message **msg**, and the share generating matrix (M, ρ). It outputs L *secret shares*, $\mathsf{sh}_1, \ldots, \mathsf{sh}_L$, where each share sh_l is given to party $\rho(l)$.

$\mathbf{Update}_l(\mathsf{sh}_l) \rightarrow \mathsf{sh}'_l$: The *randomized* update algorithm takes in the current version of the share sh_l and outputs an updated version sh'_l. We use the notation $\mathsf{Update}^i_l(\mathsf{sh}_l)$ to denote the operation of updating the share sh_l successively i times in a row. That is, sample:

$$\mathsf{sh}_l^{(1)} := \mathsf{sh}_l, \quad \mathsf{sh}_l^{(2)} \leftarrow \mathsf{Update}_l\left(\mathsf{sh}_l^{(1)}\right), \; \ldots, \; \mathsf{sh}_l^{(i+1)} \leftarrow \mathsf{Update}_l\left(\mathsf{sh}_l^{(i)}\right)$$

and output $\mathsf{sh}_l^{(i+1)}$. (Note: the above definition implies $\mathsf{Update}^0_l(\mathsf{sh}_l) = \mathsf{sh}_l$).

$\mathbf{Reconstruct}(\mathsf{sh}_{l_1}, \ldots, \mathsf{sh}_{l_P}) \rightarrow \mathbf{msg}$: The reconstruction algorithm takes in some version of some authorized subset of the secret shares $\mathsf{sh}_{l_1}, \ldots, \mathsf{sh}_{l_P}$ and it outputs the secret message $\mathbf{msg}$.

Correctness. We say that the scheme is *correct* if for any *authorized* set of shares $(\mathsf{sh}_{l_1}, \ldots, \mathsf{sh}_{l_p}) \leftarrow \mathsf{ShareGen}(1^\lambda, \mathbf{msg}, M, \rho)$ and any sequence of $i_1 \geq 0, \ldots, i_P \geq 0$ updates on these shares respectively resulting in updated shares $\mathsf{sh}'_{l_1} \leftarrow \mathsf{Update}^{i_1}_{l_1}(\mathsf{sh}_{l_1})$, ..., $\mathsf{sh}'_{l_P} \leftarrow \mathsf{Update}^{i_P}_{l_P}(\mathsf{sh}_{l_P})$, we get $\mathsf{Reconstruct}(\mathsf{sh}'_{l_1}, \ldots, \mathsf{sh}'_{l_P}) = \mathbf{msg}$. Note that $i_1, \ldots, i_P$ are arbitrary, and are not required to be equal.

Security. We define ℓ-CLR security for linear secret sharing schemes 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 $(\mathsf{sh}_1, \ldots, \mathsf{sh}_L) \leftarrow \mathsf{ShareGen}(1^\lambda, \mathbf{msg}_b, M, \rho)$. The attacker then chooses a subset $\mathcal{C} \subset [N]$ of corrupted parties. This subset must satisfy the restriction that for any $z \in [N]$, the set of shares belonging to parties $\mathcal{C} \cup \{z\}$ is an unauthorized set. In other words, adding any single other party to $\mathcal{C}$ would be insufficient to form an authorized set of shares. The challenger gives the attacker the shares belonging to the parties in $\mathcal{C}$. We let N' denote the number of remaining, uncorrupted parties, and we let L' denote the number of uncorrupted shares (i.e. the shares held by these parties). For convenience of notation, we will assume these parties are indexed by 1 through N' and their shares are $\mathsf{sh}_1, \ldots, \mathsf{sh}_{L'}$.

The challenger then chooses randomness $rand_1, rand_2, \ldots, rand_{L'}$ for the next updates of the shares 1, 2, ..., L' respectively and sets

$$\mathsf{state}_1 := \cup_{l \text{ s.t. } \rho(l)=1}(\mathsf{sh}_l, rand_l), \ldots \mathsf{state}_{N'} := \cup_{l \text{ s.t. } \rho(l)=N'}(\mathsf{sh}_l, rand_l)$$

(here we have abused the union symbol to denote concatenation). It also initializes counters $\mathcal{L}_1 := 0, \ldots, \mathcal{L}_{N'} := 0$. 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 $\mathsf{Leak} : \{0,1\}^* \rightarrow \{0,1\}$ and an index $\sigma \in \{1, \ldots, N'\}$. If $\mathcal{L}_\sigma < \ell$, then the challenger responds with the value $\mathsf{Leak}(\mathsf{state}_\sigma)$ and increases the counter $\mathcal{L}_\sigma := \mathcal{L}_\sigma + 1$. Else it responds with $\perp$.

Update Queries: The attacker specifies an index $\sigma \in \{1, \ldots, N'\}$. The challenger parses

$$\mathsf{state}_\sigma = \cup_{l \text{ s.t. } \rho(l)=\sigma}(\mathsf{sh}_l, rand_l)$$

and computes the updated shares $\mathsf{sh}'_l := \mathsf{Update}_l(\mathsf{sh}_l; rand_l)$ using randomness $rand_l$. It samples fresh randomness $rand'_l$ and sets $\mathsf{state}_l := (\mathsf{sh}'_l, rand'_l)$, $\mathcal{L}_\sigma := 0$.

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 $|\Pr[\mathcal{A} \text{ wins }] - \frac{1}{2}| \leq negl(\lambda)$.

Remark 1: Final Corruption. We note that the above definition also generically implies security if one *additional* party is (adaptively) fully corrupted at the end of the game, but then there is *no more leakage* on any other shares afterwards. The reasoning is the same as in the two party case. Assume that at some point in the game, there is a distinguishing strategy D that would use the additional fully corrupted share sh_i to break security. Then we could also just leak the predicate $D(\mathsf{sh}_i)$ to break security.

Remark 2: Comparison to Basic CLRS Definition. If we instantiate the above definition with $N = 2$ and a simple *two-out-of-two* LSSS where the only authorized set includes both parties, then we get the original basic definition of CLRS from Section 3.1. That is, the attacker cannot fully corrupt either share, but can continually leak on both shares.

Remark 3: Adaptivity. The above definition only considers *static* corruptions, where the attacker decides which parties to corrupt ahead of time before seing any leakage. When the number of parties N is small (constant or logarithmic) then this also automatically implies security against *adaptive* corruptions via a "guessing argument" where we just guess the set of parties the attacker will corrupt ahead of time.

Remark 4: Updates. We note that there is no compelling reason to require that a party update all of the shares belonging to it at the same time. We could alternatively define update queries to act on a single index $l \in L'$ at a time instead if we preferred (though in this case, we would wait for every share belonging to the party to be updated at least once before resetting the leakage counter $\mathcal{L}_\sigma$). This distinction makes no difference in the proof of security for our system.

E.2 Scheme Description

Let $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ be groups of prime order q, with pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ and let $\mathbf{g} \in \mathbb{G}_1, \mathbf{h} \in \mathbb{G}_2$ be generators. Let n, m, d be integers with $n \geq d$.

Key Generation($\mathbf{msg}, (M, \rho)$): To share a secret $\mathbf{msg} \in \mathbb{G}_T$ according to the share-generating matrix (M, ρ), we first choose $\vec{p}, \vec{w} \in \mathbb{F}_q^m$ at random subject to $\langle \vec{p}, \vec{w} \rangle = 0$. We also choose $\vec{t} \xleftarrow{\$} \mathbb{F}_q^m$ and define $\alpha = \langle \vec{p}, \vec{t} \rangle$. We also define $\mathbf{f} := e(\mathbf{g}, \mathbf{h})^\alpha$. We let $L \times K$ denote the dimensions of the matrix M, and we choose a matrix $T \in F_q^{K \times m}$ whose first row is $\vec{t}$ and whose remaining rows are chosen randomly. For each i from 1 to L, we let $\vec{M}_i$ denote the i^{th} row of the matrix M. Then $\vec{M}_i T$ is a $1 \times m$ vector of shares of the coordinates of $\vec{t}$. We denote this vector by $\vec{t}_i$ (we emphasize that this is a vector, and should not be confused with the i^{th} coordinate of the vector $\vec{t}$). We choose $\vec{r}_i = (r_1^i, \dots, r_n^i) \xleftarrow{\$} \mathbb{F}_q^n$ and define $\mathsf{sk}_i := \mathbf{h}^{S_i}$, where S_i is the $n \times m$ matrix given by:

$$S_i \;:=\; \begin{bmatrix} r_1^i \vec{w} + \vec{t}_i \\ \cdots \\ r_n^i \vec{w} + \vec{t}_i \end{bmatrix} \;=\; \begin{bmatrix} \\ (\vec{r}_i)^\top \\ \\ \end{bmatrix} \begin{bmatrix} \quad \vec{w} \quad \end{bmatrix} + \begin{bmatrix} \\ \vec{1}^\top \\ \\ \end{bmatrix} \begin{bmatrix} \quad \vec{t}_i \quad \end{bmatrix}.$$

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

We additionally choose random values $v_2, \dots, v_K \in F_q$ and implicitly define $\vec{v} \in \mathbb{F}_q^K$ by setting $e(\mathbf{g}, \mathbf{h})^{\vec{v}} := (\mathbf{msg}, e(\mathbf{g}, \mathbf{h})^{v_2}, \dots, e(\mathbf{g}, \mathbf{h})^{v_K}) \in \mathbb{G}_T^K$. For each i from 1 to L, we define $\mathbf{msg}_i := e(\mathbf{g}, \mathbf{h})^{\vec{M}_i \cdot \vec{v}}$, which can be computed from $e(\mathbf{g}, \mathbf{h})^{\vec{v}}$ and $\vec{M}_i$. We choose $\vec{u}_i = (u_1^i, \dots, u_n^i) \xleftarrow{\$} \mathbb{F}_q^n$ and define:

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

Equivalently, we can write the ciphertext as $\mathsf{ct}_i^{(1)} = \mathbf{g}^{C_i}$, $\mathsf{ct}_i^{(2)} = e(\mathbf{g}, \mathbf{h})^{\vec{z}_i^\top}$ for:

$$C_i = \begin{bmatrix} \\ \vec{u}_i^\top \\ \\ \end{bmatrix} \begin{bmatrix} & \vec{p} & \end{bmatrix} \quad , \quad \vec{z}_i^\top = \begin{bmatrix} \\ \vec{u}_i^\top \\ \\ \end{bmatrix} \alpha + \begin{bmatrix} \\ \vec{1}^\top \\ \\ \end{bmatrix} \mu_i$$

where μ_i is given by $\mathbf{msg}_i = e(\mathbf{g}, \mathbf{h})_i^\mu$ and $\alpha = \langle \vec{p}, \vec{t} \rangle$. In the language of our updatable encryption scheme, $\mathsf{ct}^{(1)}, \mathsf{ct}_i^{(2)}$ are encryptions of $\mathbf{msg}_i$.

Each party is given the values $\mathsf{sk}_i, \mathsf{ct}_i^{(1)}, \mathsf{ct}_i^{(2)}$ for the i's which ρ maps to that party, and these values comprise the share belonging to the party.

Share Updates: To update an sk_i in its share, a party chooses $A' \xleftarrow{\$} \mathsf{Rk}_d(\mathbb{F}_q^{n \times n})$ and derives a matrix A by rescaling each row of A' so that its entries sum to 1. Denoting the current value of sk_i by $\mathbf{h}^{S_i}$, the updated value is $\mathbf{h}^{AS_i}$.

To update a pair $\mathsf{ct}_i^{(1)}, \mathsf{ct}_i^{(2)}$ in its share, a party chooses $B' \xleftarrow{\$} \mathsf{Rk}_d(\mathbb{F}_q^{n \times n})$ and derives B by rescaling each row of B' so that its entries sum to 1. Denoting the current values by $\mathbf{g}_i^C, e(\mathbf{g}, \mathbf{h})^{\vec{z}_i}$, the updated values are $\mathbf{g}^{BC_i}, e(\mathbf{g}, \mathbf{h})^{B\vec{z}_i}$.

Reconstruction: We let P denote a set of parties who are authorized to reconstruct the shared value $\mathbf{msg}$. This means that the rows $\vec{M}_i$ of M such that $\rho(i) \in P$ include the vector $(1, 0, \dots, 0) \in \mathbb{F}_q^K$ in their span. We let P' denote the set of these indices i such that $\rho(i) \in P$. The parties first compute coefficients $\{\beta_i\}_{i \in P'}$ such that $\sum_{i \in P'} \beta_i \vec{M}_i = (1, 0, \dots, 0)$. For each sk_i where $i \in P'$, the party takes the first row of this and raises it to the power β_i. Taking the product of these over $i \in P'$, the parties obtain $\mathbf{h}^{\vec{t} + \vec{w}'}$ for some $\vec{w}'$ which is orthogonal to $\vec{p}$. (To see this, note that $\sum_{i \in P'} \beta_i \vec{t}_i = \vec{t}$.)

For each $\mathsf{ct}_i^{(1)}, \mathsf{ct}^{(2)_i}$ where $i \in P'$, the party takes the first entry of $\mathsf{ct}_i^{(2)}$ and raises it to the power β_i. Taking the product of these values over $i \in P'$, the parties obtain

$$\mathbf{f}^{\sum_{i \in P'} u_1^i \beta_i} \mathbf{msg} = \mathbf{f}^{u'} \mathbf{msg},$$

where we define $u' = \sum_{i \in P'} u_1^i \beta_i$. Similarly, the parties take the first row of each $\mathsf{ct}_i^{(1)}$, raise it to the power β_i, and multiply the results together to obtain: $\mathbf{g}^{u'\vec{p}}$.

The process is now completed by decryption. The parties compute:

$$e(\mathbf{g}^{u'\vec{p}}, \mathbf{h}^{\vec{t} + \vec{w}'}) = e(\mathbf{g}, \mathbf{h})^{u'\alpha} = \mathbf{f}^{u'},$$

since $\vec{p} \cdot \vec{w}' = 0$. The value $\mathbf{f}^{u'}$ can now be divided from $\mathbf{f}^{u'} \mathbf{msg}$ to obtain $\mathbf{msg}$.

E.3 Security

We prove security in a non-adaptive corruption model where we suppose the attacker initially corrupts a subset P of the parties (meaning the attacker is given the shares of these parties). We further assume that adding any one additional party to the set P would still not produce an authorized set. The attacker is then allowed to make continual leakage queries on the uncorrupted shares. We let Game Real denote the real security game, formally defined in the subsection E.1.

We will prove security via a hybrid argument over a sequence of games. To index these games, we let U denote the number of rows of the share-generating matrix (M, ρ) which correspond to uncorrupted parties. For convenience of notation, we will assume (without loss of generality) that these rows are the first U rows of M. We let $U_j \leq U$ for each j from 1 to U denote the number of rows of M which belong to uncorrupted parties *other than* party $\rho(j)$. We define the following games (for $0 \leq j \leq U$, $0 \leq k \leq U_j$). In all of these games, all updates are chosen honestly. For convenience of notation, we will drop the subscript b from the shares of $\mathbf{msg}_b$ and interpret $\mathbf{msg}_j$ as the j^{th} share of the message $\mathbf{msg}_b$.

GameCT j **:** In GameCT j, the values $\mathsf{ct}_i^{(1)}, \mathsf{ct}_i^{(2)}$ for $i \leq j$ are *honest ciphertexts* encrypting the identity in $\mathbb{G}_T$ (i.e. each $\mathbf{msg}_i$ for $i \leq j$ has been replaced by the identity element of $\mathbb{G}_T$). The values $\mathsf{ct}_i^{(1)}, \mathsf{ct}_i^{(2)}$ for $j < i \leq U$ are distributed as in Game$_{Real}$. The keys sk_i for all $i \leq U$ are also distributed as in Game Real.

GameCTSK (j, k) **:** (for $j \geq 1$) In GameCTSK (j, k), the values $\mathsf{ct}_i^{(1)}, \mathsf{ct}_i^{(2)}$ are distributed as in Game CT j for all $i \neq j$, and $\mathsf{ct}_j^{(1)}, \mathsf{ct}_j^{(2)}$ are distributed as a *low ciphertext* encrypting $\mathbf{msg}_j$. The keys sk_i for the first the first k values of i such that $\rho(i) \neq \rho(j)$ are distributed as *high keys*, while the remaining keys are distributed as *honest keys*.

GameCT′ j **:** In GameCT′ j, all sk_i for $i \leq U$ such that $\rho(i) \neq j$ are *high keys* (while sk_i for $i \leq U$ where $\rho(i) = \rho(j)$ are honest keys), and the ciphertexts are distributed as follows. The ciphertexts $\mathsf{ct}_i^{(1)}, \mathsf{ct}_i^{(2)}$ for $i < j$ are *honest ciphertexts* encrypting the identity, $\mathsf{ct}_j^{(1)}, \mathsf{ct}_j^{(2)}$ is is a *low ciphertext* encrypting the identity element, and $\mathsf{ct}_i^{(1)}, \mathsf{ct}_i^{(2)}$ for $i > j$ are *honest ciphertexts* encrypting the proper values $\mathbf{msg}_i$.

GameCTSK′ (j, k) **:** In GameCTSK′ (j, k) (for $j \geq 1$), the ciphertexts are distributed as in GameCT′ j, and the secret keys are distributed as in GameCTSK (j, k). We note that GameCTSK′ (j, U_j) is equal to GameCT′ j, and in GameCTSK′ $(j, 0)$, all the keys are distributed as *honest keys*.

We transition from Game Real (which is equal to GameCT 0) to GameCTSK $(1, 0)$, then to GameCTSK $(1, 1)$, and so on, until we reach GameCTSK $(1, U)$. (At this point, the first ciphertext is low and all of the keys are high). We next move to GameCT′ 1, then to GameCT″ 1, and next to GameCT 1, and so on. (In general, we move from GameCT $j - 1$ to GameCTSK $(j, 0)$, from GameCTSK (j, k) to GameCTSK $(j, k + 1)$, from GameCTSK (j, U_j) to Game CT′ j = GameCTSK′ (j, U_j), from GameCTSK′ (j, k) to GameCTSK′ $(j, k - 1)$, and from GameCTSK′ $(j, 0)$ to GameCT j, until we arrive at Game U, where *all* of the ciphertexts are encryptions of the identity element. At this point, $\mathbf{msg}$ is information-theoretically hidden, and it is clear that security holds.

Lemma E.1. GameCT $j - 1 \overset{\text{comp}}{\approx}$ GameCTSK $(j, 0)$ *for all j from 1 to U.*

Proof. This is essentially the same proof as Lemma C.1. We show a reduction to the $(k = 1)$-extended rank hiding problem, where the reduction is given a challenge $g^{\vec{p}}, g^{\vec{c}}$ and $\vec{w} \xleftarrow{\$} (\vec{p}, \vec{c})^{\perp}$ such that either *(I)* $\vec{p}, \vec{c}$ span a random 1-dimensional space or *(II)* $\vec{p}, \vec{c}$ span a random 2-dimensional space. The honest secret keys will be created using $\vec{w}$, and the initial ciphertext $\mathsf{ct}_j^{(1)}, \mathsf{ct}_j^{(2)}$ will be created using $g^{\vec{c}}$. To make the honest ciphertexts $\mathsf{ct}_i^{(1)}, \mathsf{ct}_i^{(2)}$ for $i \neq j$, $g^{\vec{p}}$ is used. If the challenge is of type *(I)*, then the game is distributed just as GameCT $j - 1$, while if it is of type *(II)* then the game is distributed just as GameCTSK $(j, 0)$. Therefore, the two are computationally indistinguishable under the 1-linear (SXDH) assumption. □

Lemma E.2. GameCTSK $(j, k) \overset{\text{comp}}{\approx}$ GameCTSK $(j, k + 1)$ *for all j from 1 to U and all k from 0 to $U_j - 1$.*

Proof. This is essentially the same proof as the transitions from Game' 0 to Game $q_{sk}+1$ for the updatable encryption scheme (a.k.a 2-out-of-2 sharing). We note that as these transitions are accomplished in Lemma C.2, Lemma C.3, Lemma C.7, Lemma C.8, Lemma C.12, Lemma C.14, in all computational steps, the simulator knows the vector $\vec{p}$ and chooses $\vec{t}$ for itself. Thus, we can apply these same arguments to move from GameCTSK (j,k) to GameCTSK $(j,k+1)$. Now our simulator will pick the vector $\vec{t}$ to be shared for itself, so it will know all of the vectors $\vec{t_i}$ for the secret key pieces belonging to all parties, and it can easily create the high keys because it knows the vector $\vec{p}$ (and hence can sample from the $m-1$ dimensional space $(\vec{p})^{\perp}$). It can easily make the honest ciphertexts $\mathsf{ct}_i^{(1)}, \mathsf{ct}_i^{(2)}$ for $i \neq j$, also because $\vec{p}$ is known. It can create the low ciphertext $\mathsf{ct}_j^{(1)}, \mathsf{ct}_j^{(2)}$ in the same way as in the previous proofs. The information-theoretic arguments employed before also apply here without modification, since the secret key and ciphertext pieces which are changing form belong to *different* uncorrupted parties. (Recall that secret keys sk_i which belong to the same party who holds $\mathsf{ct}_j^{(1)}, \mathsf{ct}_j^{(2)}$ remain honest and unchanged throughout.) □

Lemma E.3. GameCTSK $(j, U_j) \overset{\text{comp}}{\approx}$ GameCT′ j *for all j from 1 to U.*

Proof. This proof is very similar to the proof of Lemma C.15. We define subgames GameCTHyb (j,i) for i from 0 to n (recall that n is the number of rows in the ciphertext piece $\mathsf{ct}_j^{(1)}, \mathsf{ct}_j^{(2)}$). GameCTHyb (j,i) only differs from GameCTSK (j,U) and GameCT′ j in the distribution of $\mathsf{ct}_j^{(1)}, \mathsf{ct}_j^{(2)}$ - all other aspects of the games are identical. In GameCTHyb (j,i), the first i rows of $\mathsf{ct}_j^{(1)}, \mathsf{ct}_j^{(2)}$ are *low encryptions* of the identity element, while the rest are *low encryptions* of the proper shares of $\mathbf{msg}_b$. We note that GameCTHyb $(j,0)$ is identical to GameCTSK (j,U), and GameCTHyb (j,n) is identical to GameCT′ j. We additionally define GameCTHybMid (j,i) for i from 0 to $n-1$ as being the same as GameCTHyb $(j,i+1)$, except that row $i+1$ of $\mathsf{ct}_j^{(1)}, \mathsf{ct}_j^{(2)}$ is a *mid encryption* of $\mathbf{msg}_j$.

We transition from GameCTHyb (j,i) to GameCTHyb (j,i) as in the proof of Claim C.16. We note that the simulator here knows $\vec{t}$ and $\vec{p}$, so it can easily generate all of the appropriately distributed sk_i values and the honest ciphertexts for $\mathsf{ct}_i^{(1)}, \mathsf{ct}_i^{(2)}$ where $i \neq j$. It makes $\mathsf{ct}_j^{(1)}$ and $\mathsf{ct}_j^{(2)}$ as in the proof of Claim C.16.

We transition from GameCTHybMid (j,i) to GameCTHyb $(j,i+1)$ in a similar way to the proof of Claim C.17, but we must now consider also the corrupted shares in our information-theoretic argument. We consider the information about $\vec{t}$ that is revealed in the attacker's view. As in the proof of Claim C.17, the $(i+1)$ row of the initial ciphertext $\mathsf{ct}_j^{(1)}, \mathsf{ct}_j^{(2)}$ is of the form

$$(\mathbf{g}^{u_1\vec{c}_1+u_2\vec{c}_2}, e(\mathbf{g},\mathbf{h})^{u_1\beta+u_2\gamma}\mathbf{msg}_j)$$

where $\beta = \langle \vec{c}_1, \vec{t}\rangle, \gamma = \langle \vec{c}_2, \vec{t}\rangle$.

We claim that the only information about $\vec{t}$ that is available in the attacker's view elsewhere are the values of β and $\alpha = \langle \vec{t}, \vec{p}\rangle$. We note that the value β is the only information about $\vec{t}$ that is revealed by the other rows of this ciphertext, and no information is revealed by the other honest ciphertexts. We then claim that α is the only information about $\vec{t}$ that is revealed by the secret keys. To prove this claim, we first observe that the corrupted keys in addition to the honest keys sk_i for $i \leq U$ such that $\rho(i) = \rho(j)$ are insufficient to reconstruct the secret vector $\vec{t}$. This is because we have limited the attacker to corrupting a set of parties such that adding any one additional party (e.g. the party $\rho(j)$) does not make the set become authorized.

We next claim that the shares $\vec{t_i}$ for this (unauthorized) set of keys *reveals no information about $\vec{t}$.* To see this, consider the set R of rows of M that ρ maps to corrupted parties or to $\rho(j)$. The vector $(1,0,\ldots,0)$ is *not* in the span of these rows. Therefore, there exists some vector $\vec{y} \in \mathbb{F}_q^K$ such that $\vec{y} \cdot \vec{M}_i = 0$ for all rows $\vec{M}_i \in R$, and $\langle \vec{y}, (1,0,\ldots,0)\rangle \neq 0$. Without loss of generality, we can assume that the first coordinate of $\vec{y}$ is 1. We consider the first coordinate of $\vec{t}$, denoted by t_1. The shares of this coordinate are computed as $\langle \vec{M}_i, \vec{v}\rangle$, where $\vec{v}$ is a vector $\in \mathbb{F}_q^K$ whose first coordinate is t_1 and the other coordinates are chosen uniformly at random. We can achieve the exact same distribution for $\vec{v}$ by setting $\vec{v} = t_1\vec{y} + \vec{z}$, where $\vec{z} \in \mathbb{F}_q^K$ has its first coordinate equal to 0 and its other coordinates chosen randomly. Now, for rows $\vec{M}_i$ in R, $\langle \vec{M}_i, \vec{v}\rangle = \langle \vec{z}, \vec{v}\rangle$, since $\vec{M}_i$ is orthogonal to $\vec{y}$. Since these values have no dependence on $\vec{t}_1$, it is clear that they reveal nothing about the value t_1. The same holds for all other coordinates of $\vec{t}$. Therefore, the attacker can learn nothing about $\vec{t}$ from the keys of the corrupted parties and the keys held by party $\rho(j)$. All other keys are *high keys*, whose distribution only depends on $\alpha = \langle \vec{t}, \vec{p}\rangle$. Hence, the only information about $\vec{t}$ that is revealed by the secret keys is $\alpha = \langle \vec{t}, \vec{p}\rangle$.

Putting this all together, we see that the value of γ appearing the $i+1$ row of the ciphertext is distributed as a uniformly random element, conditioned on the view of the attacker. This information-theoretically hides the message $\mathbf{msg}_j$. □

Lemma E.4. GameCTSK′ $(j, k+1) \overset{\text{comp}}{\approx}$ GameCTSK′ (j, k) *for all j from 1 to U and all k from 0 to $U_j - 1$.*

Proof. This follows from the same proof as Lemma E.2, with the transitions applied in reverse. In other words, we can simply rewind our arguments to revert the relevant secret key to its original state as an honest key while the ciphertext $\mathsf{ct}_j^{(1)}, \mathsf{ct}_j^{(2)}$ remains a low encryption of the identity element. □

Lemma E.5. GameCTSK′ $(j, 0) \overset{\text{comp}}{\approx}$ GameCT j *for all j from 1 to U.*

Proof. This follows from the same proof as Lemma E.1. (Here we are rewinding the ciphertext back to an honest encryption, with the underlying message now being the identity element.) □

F Impossibility of Information-Theoretic Security

We notice that, unlike public-key encryption and digital signatures, traditional secret sharing schemes (including 2-out-2 secret sharing) can achieve *information-theoretic* security. Namely, the secret message remains perfectly hidden even against a *computationally unbounded* attacker. Unfortunately, we show that the same cannot be true for CLRS schemes, which must withstand *continuous* leakage of shares. Namely, by specifying *arbitrary* (as opposed to efficient) leakage predicates, the attacker can reconstruct the hidden secret with probability arbitrarily close to 1. In fact, the result holds even when the following additional restrictions are placed on the attacker:

- The leakage bound $\ell = 1$. Namely, at most one bit can leak in between successive share updates.
- Each leakage predicate can only depend on the current share value, but not on the randomness for the next update. Namely, we can assume *leak-free updates*.
- If s_1 is the bit size of the first share and s_2 is the bit size of the second share, the attacker will use only s_1 **Leakage** queries on the first share and only s_2 **Leakage** queries on the second share.[10]
- The sequence of **Leakage** and **Update** queries is specified *non-adaptively*.[11]
- The attacker can even break *one-wayness* of the CLRS scheme, and not just semantic security. Namely, the shared message **msg** can be chosen at random (as opposed to being either $\mathbf{msg}_0$ or $\mathbf{msg}_1$ chosen by the attacker) and will be recovered in full with probability $1 - \varepsilon$ (for any $\varepsilon > 0$).

[10] This is essentially optimal, since otherwise both shares still have some entropy left after leakage, and the results of [DDV10] give an information-theoretic CLRS scheme in this setting.

[11] Alternatively, one can use s_1 non-adaptive queries on the first share, and a single *adaptive* query on the second share.

As it turns out, all these properties will easily follow from the following more general attack, which we call *Continuous Leakage of Consistent Value* (CLCV) attack. The attack, parameterized by an arbitrary key refreshing procedure Update, shows how to leak a value "Update-consistent" with an s-bit initial secret, using at most s non-adaptive (but computationally unbounded) **Leakage** queries. After specifying this attack below, the attack on the CLRS scheme will simply perform a separate CLCV attack on each share, and then run the honest reconstruction algorithm to recover the secret.

CLCV Attack. Assume $\mathsf{Update} : \{0,1\}^s \to \{0,1\}^s$ is an arbitrary randomized procedure. Given such a procedure, we say that a value x' is *consistent* with x if either $x = x'$ or there exists some $i \geq 1$ and a sequence of randomness strings $r_1, \ldots, r_i$ such that $x' = \mathsf{Update}(\ldots \mathsf{Update}(x; r_1) \ldots; r_i)$. We let $C(x)$ denote the set of all strings x' consistent with x.

We define the following *CLRV game* between a (computationally unbounded) attacker $\mathcal{A}$ and a challenger $\mathcal{C}$. The challenger $\mathcal{C}$ gets an input $x \in \{0,1\}^s$ and sets $x_0 = x$, $i = 0$. The attacker $\mathcal{A}$ can adaptively make any number of the following two queries:

Update Queries: $\mathcal{C}$ picks a random string r_{i+1}, sets $x_{i+1} = \mathsf{Update}(x_i; r_{i+1})$, and increments i.

Leakage Queries: $\mathcal{A}$ specifies a predicate $\mathsf{Leak} : \{0,1\}^s \to \{0,1\}$ and gets back the value $\mathsf{Leak}(x_i)$ from $\mathcal{C}$. (Notice, the leakage predicate does not take the update randomness as its input.) $\mathcal{C}$ then automatically makes an **Update** query described above, updating x_i to x_{i+1} and incrementing i.[12]

At the end of the game the attacker outputs a value x' and wins if x' is consistent with x: $x' \in C(x)$.

Lemma F.1. *For any randomized procedure* $\mathsf{Update} : \{0,1\}^s \to \{0,1\}^s$ *and any* $\varepsilon > 0$ *there exist an (inefficient) attacker* $\mathcal{A}^*$ *such that, for all* $x \in \{0,1\}^s$, $\mathcal{A}^*$ *wins the CLCV game against* $\mathcal{C}(x)$ *with probability* $1 - \varepsilon$. *Moreover,* $\mathcal{A}^*$ *is non-adaptive and makes only* s **Leakage** *queries.*

We prove the lemma below, but, as an immediate corollary, we get the attack on the CLRS mentioned at the beginning of the section. Namely, we set the failure parameter to $\varepsilon/2$ and run the CLCV attacker from the Lemma above on both shares sh_1 and sh_2. Then, with probability $1 - \varepsilon$ we get correct share values sh'_1 and sh'_2 consistent with sh_1 and sh_2. By perfect correctness of CLRS, running the reconstruction procedure on sh'_1 and sh'_2 will return the correct message $\mathbf{msg}$.

In fact, we notice that the CLCV attack essentially rules out information-theoretic security for any cryptographic primitive in the continuous leakage model enjoying perfect correctness. As we mentioned, however, this is primarily interesting for cryptographic primitives which permit information-theoretic solutions (without leakage) in the first place, such as secret sharing, one-time pad, one-time MACs, etc.

Proof of Lemma F.1. We start with some notation before we describe our CLCV attacker $\mathcal{A}^*$. Given a permutation $\pi : \{0,1\}^s \to \{0,1\}^s$ and a non-empty set $X \subseteq \{0,1\}^s$, we let $\mathsf{smallest}_\pi(X)$ denote the (unique) string $x \in X$ having the lexicographically smallest value $\pi(x)$ among $\{\pi(x') \mid x' \in X\}$. Notice, if π is a random permutation, then $\mathsf{smallest}_\pi(X)$ is simply a random element of X. More generally, for any sequence $X_i \supseteq X_{i+1} \supseteq \ldots \supseteq X_{i+s-1}$ of s non-empty "shrinking" sets, if π is a random permutation, the probability that *all* s values $\mathsf{smallest}_\pi(X_{i+j})$ are the same only depends on the ratio between the sizes of the smallest and the largest sets:

$$\Pr_\pi\left[\mathsf{smallest}_\pi(X_i) = \mathsf{smallest}_\pi(X_{i+1}) = \ldots = \mathsf{smallest}_\pi(X_{i+s-1})\right] = \frac{|X_{i+s-1}|}{|X_i|} \tag{6}$$

Indeed, since the sets are contained in each other, all the values $\mathsf{smallest}_\pi(X_{i+j})$ are equal if and only if $\mathsf{smallest}_\pi(X_i) \in X_{i+s-1}$. However, since π is a random permutation, the latter happens with probability precisely $|X_{i+s-1}|/|X_i|$.

For $i \geq 0$, let $X_i = C(x_i)$ be the (non-empty) set of values consistent with the i-th secret x_i. The non-adaptive strategy of $\mathcal{A}^*$ is the following (recall, ε is the maximum allowed failure probability):

[12] This corresponds to the leakage bound $\ell = 1$, meaning that at most 1 bit can leak in between the updates.

- Pick a random permutation $\pi : \{0,1\}^s \to \{0,1\}^s$, and a random integer $t \in \{1, \ldots, N \stackrel{\text{def}}{=} \frac{4s}{\varepsilon^2 \log e}\}$.
- Perform the sequence of $i \stackrel{\text{def}}{=} (t-1)s$ **Update** queries, so that the current secret is x_i.
- Perform s **Leakage** queries, where query $j \in [s]$ will leak the j-th bit of $\mathsf{smallest}_\pi(X_{i+j-1})$. (Formally, the j-th leakage predicate $\mathsf{Leak}_j(y)$ returns the j-th bit of $\mathsf{smallest}_\pi(C(y))$.)
- Let $x' = b_1, \ldots, b_s$ be the concatenation of s answers to the **Leakage** queries above. Output x' as a candidate secret consistent with $x = x_0$.

It remains to argue that the probability that x' is consistent with x is at least $1 - \varepsilon$. For that, let $y_j \stackrel{\text{def}}{=} \mathsf{smallest}_\pi(X_{i+j-1})$ be the value whose j-th bit we leak in the j-th **Leakage** query. Let us call these s values $y_1, \ldots, y_s$ *critical*. Notice, each critical value is consistent with x, by definition. Thus, it suffices to argue that the probability all the critical values are *the same* is at least $1 - \varepsilon$. Indeed, in this case the leaked value x' is actually equal to all the critical values, and, hence, consistent with x. This also gives the intuition behind our construction of $\mathcal{A}^*$. Essentially, $\mathcal{A}^*$ choose a "random" index i hoping that the set of consistent values $X_i = C(x_i)$ will not shrink too much during the next s updates. If this is so (which we prove below happens with high probability), $\mathcal{A}^*$ wants to choose a "consistent representative" from the slowly shrinking sets $X_i, X_{i+1}, \ldots, X_{i+s-1}$. Since this "shrinkage" might be adversarial, $\mathcal{A}^*$ randomly permutes the elements of the "slowly shrinking" sets, and hopes that the smallest element of these permuted sets does not "disappear" as the sets slowly shrink. Luckily, Equation (6) tells us precisely this, as long as the smallest set X_{i+s-1} is almost as big as the original set X_i. The formal details are given below.

Let us call a sequence of s consecutive updates an *epoch*, and let us examine the consistent sets $X_0, X_s, X_{2s}, \ldots$ at the end of each epoch. We say that epoch $t \geq 1$ is "bad" if $|X_{ts}| < (1 - \frac{\varepsilon}{2}) \cdot |X_{(t-1)s}|$; namely, the set of consistent values shrunk by more than a factor $(1 - \frac{\varepsilon}{2})$. Notice, since the set of consistent values has size at most 2^s and at least 1, we know that the maximal number n of bad epochs must satisfy the relation $2^s(1 - \frac{\varepsilon}{2})^n \geq 1$. Solving for n, we get the number of bad epochs $n \leq 2s/(\varepsilon \log e)$.

Recall now that $\mathcal{A}^*$ choose a random epoch t in the range $\{1, \ldots, N \stackrel{\text{def}}{=} \frac{4s}{\varepsilon^2 \log e}\}$. Since there are at most $n \leq 2s/(\varepsilon \log e)$ bad epochs, we get that the probability $\mathcal{A}^*$ chose a bad epoch is at most $n/N \leq \varepsilon/2$. Otherwise, if $\mathcal{A}^*$ chose a "good" epoch, we know that $|X_{ts}| \geq (1 - \frac{\varepsilon}{2}) \cdot |X_{(t-1)s}|$. Since π was random and epoch t is good, Equation (6) then tells us that s critical values $y_j = \mathsf{smallest}_\pi(X_{i+j-1})$ are *all the same* with probability $1 - \varepsilon/2$, meaning they are not the same with probability at most $\varepsilon/2$. Hence, we see that $\mathcal{A}^*$ fails either it chose a bad epoch t (probability at most $\varepsilon/2$), or the epoch was good but the critical values were inconsistent (again, probability at most $\varepsilon/2$). Summing these, we get that $\mathcal{A}^*$ fails with probability at most ε, completing the proof. $\square$