Distributed Public Key Schemes Secure against Continual Leakage

Distributed Public Key Schemes Secure against Continual Leakage

[Extended Abstract]

Adi Akavia
Tel-Aviv Academic College
akavia@mta.ac.il

Shafi Goldwasser
Weizmann Institute and MIT
shafi@theory.csail.mit.edu*

Carmit Hazay
Aarhus University
carmit@cs.au.dk†

ABSTRACT

In this work we study *distributed* public key schemes secure against continual memory leakage. The secret key will be shared among two computing devices communicating over a public channel, and the decryption operation will be computed by a simple 2-party protocol between the devices. Similarly, the secret key shares will be periodically refreshed by a simple 2-party protocol executed in discrete time periods throughout the lifetime of the system. The leakage adversary can choose pairs, one per device, of polynomial time computable length shrinking (or entropy shrinking) functions, and receive the value of the respective function on the internal state of the respective device (namely, on its secret share, internal randomness, and results of intermediate computations).

We present distributed public key encryption (DPKE) and distributed identity based encryption (DIBE) schemes that are secure against continual memory leakage, under the Bilinear Decisional Diffie-Hellman and 2-linear assumptions. Our schemes have the following properties:

1. Our DPKE and DIBE schemes tolerate leakage at all times, including during refresh. During refresh the tolerated leakage is a $(1/2 - o(1), 1)$-fraction of the secret memory of P_1, P_2 respectively; and at all other times (post key generation) the tolerated leakage is a $((1 - o(1)), 1)$-fraction of the secret memory of P_1, P_2 respectively.

2. Our DIBE scheme tolerates leakage from both the *master secret key* and the identity based secret keys.

3. Our DPKE scheme is CCA2-secure against continual memory leakage.

4. Our DPKE scheme also implies a secure storage system on leaky devices, where a value s can be secretely stored on devices that continually leak information about their internal state to an external attacker. The devices go through a periodic refresh protocol.

These properties improve on bounds and properties of known constructions designed to be secure against continual memory leakage in the single processor model.

Categories and Subject Descriptors

F.m [**Theory of Computation**]: Miscellaneous

1. INTRODUCTION

The absolute privacy of secret keys is the basic underlying assumption made in the security proof methodology of modern cryptography. Yet, as demonstrated by a large volume of works on side channel attacks [27, 7, 3, 28, 35, 21, 24, 37], when implementing cryptographic protocols in real world hardware, some information on the secret key may leak to the adversary. This turns out to completely compromise the security of well known cryptographic protocols, including for example the RSA cryptosystem and the AES [24]. To combat such threats, much recent work [36, 9, 12, 19, 26, 31, 20, 23, 1, 2, 32, 16, 34, 33, 13, 11, 15] has been done on *leakage resilient cryptography*, that is, designing cryptographic protocols that remain secure even when some information on the secret key is leaked to the adversary.

The model of leakage resilience most relevant to our work is the *Continual memory leakage* [11, 15]. In this model the memory of the computing device is viewed as consisting of two types of memory: (i) *Public memory* that stores the public key, the public randomness used by the system, and the inputs and outputs of the computation; and (ii) *Secret memory* that stores the secret key, the secret randomness used by the system, and the intermediate steps in the computations. An adversary attacking the system can see the contents of the public memory in its entirety, and on top of that the adversary is allowed to obtain a limited amount of information about the secret memory as defined next. Specifically, time is viewed as partitioned into discrete time periods, where during each time period the adversary can choose an arbitrary polynomial-time computable leakage function, and obtain as a result the leakage function applied to the secret memory of the device. The only restriction on the leakage

*ISF 700/08, BSF 2008362, DARPA Contract Number: FA8750-11-2-0225 and NSF Contract CCF-1018064

†Supported in part by The Danish National Research Foundation and The National Science Foundation of China (grant 61061130540) for the Sino-Danish Center for the Theory of Interactive Computation; The CFEM research center (supported by the Danish Strategic Research Council).

PODC'12, July 16–18, 2012, Madeira, Portugal.

function is that it is *length shrinking* [1], namely, its output length is at most a pre-specified fraction of the number of bits of the secret memory.[1] At the end of each time-period, the secret key is *refreshed*; namely, a randomized procedure is executed that takes as input a secret key sk corresponding to a public key pk, and outputs a uniformly random secret key sk' for the same public key. We point out that leakage could happen during key refresh, can depend on the randomness used for the refreshing procedure (among other things). In summary, in this model the total leakage is *unbounded*, but leakage is bounded *within* each time period.

1.1 Our Work

In this work we study *distributed* encryption schemes resilient against continual memory leakage:[2] We propose that the secret key will be shared among two computing devices communicating over public channels, and the decryption and key refresh operations will be computed by simple 2-party protocols between the devices. The leakage adversary can now choose pairs, one per device, of polynomial time length shrinking functions, and receive the value of the respective function on the secret memory of the respective device. The secret memory contains: the secret share of the secret key as well as the device's internal randomness and results of intermediate computations. (See a detailed description of our model in Section 3.)

The advantage that this model offers is that the leakage functions chosen by the adversary are limited now to only apply to the shares of the secret memory rather than the entire secret memory at once. We remark that the choice of the leakage function can be *adaptive* based on all public information up to and including the current time period and all leakage bits from both devices obtained in *earlier* time periods, but is chosen independently of the bits leaked from the "other" device during the current time period. We believe that the limitation this model sets on leakage from the entire secret memory at once is both useful and realistic. Examples in which the distributed leakage setting is especially suitable are:

- **Symmetric Encryption:** Two processors would like to set up a symmetric encryption scheme in presence of leakage attacks. The classical solution calls for agreeing in person on a secret key based on which future decryption (and encryption) can be computed. Both processors store the common secret key in their local memory, and as such an adversary can receive leakage computed on the entire stored secret key. If instead the processors agree in person on a common secret key but each stores only a share of it, they could still decrypt and refresh the secret key via an interactive protocol, but the leakage will be restricted to be computed on each share separately. We remark that splitting decryption keys and doing distributed decryption is *not* a new idea but was extensively pursued in the proactive world. But the motivation as well as the adversary model here are different. In the proactive setting the fear was that one of the processors could be fully compromised whereas here both processors are partially compromised in the sense that their secrets leak.
- **Auxiliary Device:** To battle leakage attacks in a public key encryption scheme, do not store the secret memory on the device in its entirety but instead add an auxiliary simpler computing gadget (say, a smart card) and store shares of the secret on the main processor and the auxiliary device respectively. To decrypt, a protocol between the main processor and the auxiliary device ensues. This will be particularly attractive, if one can make the computation on the auxiliary device much simpler than the computation on the main processor, as shall indeed be the case in the schemes proposed in this paper.
- **Secure Storage on Leaky Hardware:** Say one is merely interested in long term secret storage of data s on hardware that leaks. This problem can be solved by a distributed encryption scheme resilient to leakage as follows. Store $Enc_{pk}(s)$ on one leaky hardware device and sk on another leaky hardware device. To battle leakage the devices will periodically refresh the ciphertext (stored on the first device) and the secret key (stored on the second device) using a refresh protocol. We note that this case was addressed by an independent work of Dodis et al [17] who designed a private key encryption scheme E_{sk} and proved it is leakage resilient under the linear assumption in prime order groups. They then store $E_{sk}(s)$ on one leaky device and sk on another, and show how to refresh $E_{sk}(s)$ and sk periodically without requiring interaction between the devices.

In this work we construct a distributed public key encryption (DPKE) scheme secure against continual memory leakage, and a distributed identity based encryption (DIBE) scheme secure against continual memory leakage. The security of our schemes is under the Bilinear Decisional Diffie-Hellman (BDDH) and 2-linear (2Lin) assumptions. The schemes achieve the following properties:

Leakage Parameters: Our DPKE and DIBE schemes tolerate during key refresh periods a leakage of $((1/2-o(1)),1)$-fraction of the secret memory of P_1 and P_2 respectively (note that this is optimal as in these periods each device holds both the current and the next secret key share, and hence the size of the secret memory doubles); and during all other periods (post key generation) $((1-o(1)),1)$-fraction of the secret memory of P_1 and P_2 respectively. Assuming leakage freeness during key generation is standard. We show nevertheless that this assumption can be relaxed: Our schemes can tolerate leakage during key generation, where the leakage is up to $O(\log n)$ bits under the standard BDDH and 2Lin assumptions, and up to n^{ϵ} bits under the sub-exponential BDDH and standard 2Lin assumptions (for n the security parameter).

Leakage from Master Secret Key in IBE: Our DIBE scheme tolerates leakage from both the master secret key and the identity based secret keys.

CCA2 Security: Our DPKE scheme is CCA2-secure against continual memory leakage. Namely, it is secure even when the adversary has access to a decryption oracle (on top of

[1] More generally, both in [11, 15] and in our work it suffices to restrict the leakage function to be *entropy shrinking* [32], namely, requiring that the secret key has non-trivial average min-entropy conditioned on the leakage.

[2] Historical remark. An earlier version of this work (unpublished manuscript) initiated the study of distributed schemes resilient against continual memory leakage predating [29, 17].

access to a leakage oracle). Leakage occurs only *prior* to seeing the challenge ciphertext.

Simplicity of One of the Two Devices: In our schemes the computation performed by one of the computing devices is indeed quite simple; let's call this device P_2. All P_2 does is: (a) sample random coins $s_1, \ldots, s_\ell \in \mathbb{Z}_p$, and (b) given a list of group elements (sent from P_1), P_2 computes (and returns to P_1) the product of these elements to the power of $s_1, \ldots, s_\ell$, respectively. Thus, we may view the other device P_1 as being our main processing device (say, our computer), while P_2 can be an auxiliary device (say, a smart card) communicating with P_1.

We remarks that our DPKE scheme can also be used for securely storing data on leaky devices.

1.2 Related Works

1.2.1 Results in the Single Processor Model

There are several known constructions, which address continual memory leakage in the single processor model, of public key encryption (PKE) and identity based encryption (IBE) schemes secure against continual memory leakage. These include the PKE and IBE schemes of Brakerski et al. [11]; the PKE scheme of Lewko et al. [29] with a recent followup work by Dodis et al. [17]; and the recent IBE scheme (as well as Hierarchical IBE (HIBE) and attribute-based encryption (ABE) schemes) of Lewko et al. [30]. We also mention the work [15] that addresses identification and authenticated key agreement schemes. All these schemes consider continual memory leakage attacks, for leakage functions that are length (similarly, min-entropy) shrinking, but achieve different levels of security, leakage parameters, and efficiency.

In terms of security our DPKE scheme achieves CCA2-security, in contrast to semantic-security in [11, 29, 17]. Our DIBE achieves the same security notion as the IBE of [30].

A central ingredient in all schemes secure against continual memory leakage is designing a mechanism for periodically *refreshing* the secret key, that is, replacing the secret key by a new secret key while maintaining the same public key. The fraction of leakage bits tolerated by [11, 15, 30] differ greatly on whether or not the leakage is taking place during refreshing of the secret key. As mentioned about, during refresh our DPKE is resilient to the optimal leakage of $((1/2 - o(1)), 1)$-fraction of the secret memory of P_1 and P_2 respectively, in contrast to only $o(1)$-fraction in [11, 30], 1/258-fraction in [29], and 1/672-fraction in [17]. No leakage during refresh is tolerated in [15].

Another key question in the IBE case is whether leakage is allowed from the master secret key as well as the secret keys of different identities. As mentioned above, our DIBE tolerates optimal leakage of $((1 - o(1)), 1)$-fraction of the bits of the master secret key shares msk_1, msk_2 at all times other than during refresh, and optimal leakage of $((1/2 - o(1)), 1)$-fraction during refresh, in contrast to allowing no leakage from the master secret key in [11], and restricting the leakage during the refresh period to $o(1)$-fraction in [30].

Finally, our DPKE scheme is more efficient than [11, 29, 30] when considering the parameters necessary to achieve leakage fraction of $(1 - o(1))$ in [11, 29, 30]. In particular, our scheme encrypts group elements rather than single bits, encryption requires a single pairing operation (which can be provided as part of the public key) and two exponentiations (over a prime order group), and the size of our ciphertext is two group elements.[3]

1.2.2 Leakage Resilience in Distributed Setting

In our setting the two processors are cooperating rather than adversarial and are both being leaked on by an adversary. The shares of the secret key are held by two parties as means to fight the leakage adversary rather than being the initial inputs of two processors.

In several recent papers, leakage resilience in a distributed setting was considered where parties may have initial private inputs, and some of which may be faulty. This is the case in [10] where n processors want to toss a common coin in the presence of both full corruption of processors and leakage on the private state of non-corrupted processors. The works of [22, 4] consider two-party protocols with leakage for the oblivious-transfer, commitment, and zero-knowledge functionalities; identification protocols with leakage [2, 14, 15] were also considered.

1.3 Paper Organization

The rest of the paper is organized as follows. Preliminary definitions and facts appear in Section 2. We formally define our model in Section 3; state our main results and give an overview of our constructions and proof techniques in Section 4; present the construction of our DPKE semantically secure against continual memory leakage in Section 5; and give an overview of its security proof in Section 6.

2. PRELIMINARIES

We briefly review some standard definitions and facts.

A function $p(n)$ is polynomial if $\exists c > 0$ s.t. $\forall n,\ p(n) \leq n^c$. A function $\mu(n)$ is negligible if for every polynomial $p(\cdot)$, $\exists N$ s.t. $\forall n > N$, $\mu(n) < \frac{1}{p(n)}$. We denote the security parameter by n, and adopt the convention whereby a machine is said to run in polynomial-time if its number of steps is polynomial in its *security parameter* alone. We use the shorthand PPT to denote probabilistic polynomial-time. Another shorthand notation we use is $[\ell] = \{1, 2, \ldots, \ell\}$ (for natural numbers ℓ).

Let $X = \{X_n(a)\}_{n\in\mathbb{N}, a\in\{0,1\}^*}$ and $Y = \{Y_n(a)\}_{n\in\mathbb{N}, a\in\{0,1\}^*}$ be distribution ensembles. We say that X and Y are computationally indistinguishable, denoted $X \approx_c Y$, if for every family $\{C_n\}_{n\in\mathbb{N}}$ of polynomial-size circuites, there exists a negligible function $\mu(\cdot)$ such that for all $a \in \{0,1\}^*$, $|\Pr[C_n(X_n(a)) = 1] - \Pr[C_n(Y_n(a)) = 1]| < \mu(n)$.

Let $X = X_n$ and $Y = Y_n$ be random variables accepting values taken from a finite domain D. The statistical distance between X and Y is $SD(X, Y) = \frac{1}{2}\sum_{v\in D} |\Pr[X = v] - \Pr[Y = v]|$. We say that X and Y are ϵ-close if their statistical distance is at most $SD(X, Y) \leq \epsilon(n)$. We say that X and Y are statistically close, denoted $X \approx_s Y$, if $\epsilon(n)$ is negligible.

[3]For comparison, the scheme of [11] encrypts bit-by-bit, encryption requires $\omega(n)$ exponentiations, and the ciphertext size is $\omega(n)$ group elements; the scheme of [29] encrypts bit-by-bit, the number of exponentiations for encryption (as well as the number of group elements in the ciphertexts) is constant, but these exponentiations are over composite order groups of the order of a product of four primes; and the scheme of [30] encrypts group elements, but requires $\omega(1)$ exponentiations for encryption, and the ciphertexts size is $\omega(1)$ group elements.

The min-entropy of X is $H_\infty(X) = \min_{v \in D}(-\log \Pr[X = v])$. The average min-entropy [18] defined by: $\tilde{H}_\infty(X|Y) = -\log\left(\mathbb{E}_{v \to Y}\left[2^{-H_\infty(X|Y=v)}\right]\right)$. captures the remaining uncertainty of the random variable X conditioned on the value of the random variable Y.

The leftover hash lemma says that if X is a random variable with min-entropy at least $H_\infty(X) \geq k$, and $H = \{h\colon D \to R\}$ is a family of pairwise independent functions (i.e., for all $x \neq y \in D$ and $a, b \in R$, $\Pr_{h \in H}[h(x) = a \ \& \ h(y) = b] = 1/|R|^2$), s.t. $\log|R| \leq k - 2\log(1/\epsilon)$, then $SD((h, h(x)), (h', r)) \leq \epsilon$ for h, x chosen independently at random from H and X respectively; and (h', r) uniformly random in $H \times R$.

2.1 Hardness Assumptions

We define bilinear mappings and the Bilinear Decisional Diffie-Hellman (BDDH) and k-Linear (kLin) assumptions.

An (admissible) bilinear map is a function $e\colon \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ between two multiplicative prime order cyclic groups $\mathbb{G}$, $\mathbb{G}_T$ satisfying the following: (1) Bi-linearity: $\forall u, v \in \mathbb{G}$, $\forall a, b \in \mathbb{Z}_p$, $e(u^a, v^b) = e(u, v)^{ab}$. (2) Non-degeneracy: For g a generator of $\mathbb{G}$, $e(g, g)$ generates $\mathbb{G}_T$. (3) e is efficiently computable.

A parameters generating algorithm for a bilinear map e is a PPT algorithm $(p, g, e) \leftarrow \mathcal{G}(1^n)$ that, given a security parameter n (in unary), outputs an n-bits prime number p, a generator g of an order p group $\mathbb{G}$, and an admissible bilinear map $e\colon \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ (for $\mathbb{G}_T$ the order p group generated by $e(g, g)$).

The Bilinear Decisional Diffie-Hellman (BDDH) assumption for $\mathcal{G}$ says that for $(p, g, e) \leftarrow \mathcal{G}(1^n)$ and a, b, c, r independent and uniformly random elements in $\mathbb{Z}_p$,

$$(p, g, e, g^a, g^b, g^c, e(g, g)^{abc}) \approx_c (p, g, e, g^a, g^b, g^c, e(g, g)^r).$$

We define the kLin and matrix kLin assumptions for $k \geq 1$ a constant. The k-Linear (kLin) assumption for $\mathcal{G}$ says that for $(p, g, e) \leftarrow \mathcal{G}(1^n)$, $g_1, \ldots, g_k \leftarrow \mathbb{G}$ and $r_0, r_1, \ldots, r_k$ independent and uniformly random elements in $\mathbb{Z}_p$,

$$\begin{aligned}(p, g, g_0, g_1, \ldots, g_k, g_1^{r_1}, \ldots, g_k^{r_k}, g_0^{\sum_{i=1}^k r_i}) \\ \approx_c \quad (p, g, g_0, g_1, \ldots, g_k, g_1^{r_1}, \ldots, g_k^{r_k}, g_0^{r_0}).\end{aligned}$$

The matrix kLin assumption says that for every integers a and b, and every $k \leq i < j \leq \min\{a, b\}$ the ensembles $\left\{(p, g, g^R)\right\}_{R \in Rk_i(\mathbb{Z}_p^{a \times b}), n \in \mathbb{N}}$ and $\left\{(p, g, g^R)\right\}_{R \in Rk_j(\mathbb{Z}_p^{a \times b}), n \in \mathbb{N}}$ are computationally indistinguishable, where $Rk_i(\mathbb{Z}_p^{a \times b})$ is the set of all rank i matrices. The kLin assumption implies the matrix kLin assumption (see [32] Appendix A).

3. OUR MODEL

Our model extends the continual memory leakage model of [11, 15] to distributed settings. We focus here on distributed public key encryption (DPKE) schemes secure against continual memory leakage; our definitions for distributed identity based encryption (DIBE) are analogous.

3.1 Distributed Public Key Encryption

We propose that the secret key will be *shared* between two computing devices communicating over a *public channel*, and decryption will be executed by a simple 2-party protocol. Likewise, refreshing of the secret key shares will be executed by a simple 2-party protocol.

To be concrete, let us denote the two computing devices by P_1 and P_2, and their shares of the secret key by sk_1 and sk_2 respectively. Time is viewed as partitioned into discrete time periods; and *refreshing* of the secret key shares is executed at the end of each time-period.[4] Namely, at the end of each time-period a 2-party protocol is executed that takes as input secret key shares (sk_1, sk_2) corresponding to a public key pk, and outputs new secret key shares (sk_1', sk_2') for the same public key, where the new secret key shares are drawn from the same distribution as the old ones. By the termination of the refresh protocol the old secret key share sk_i has been erased from the secret memory of P_i, and the new secret key share sk_i' has been put in its place. We emphasize that the public key remains unchanged throughout the life time of the system. The definition below summarizes the above and sets some notations:

DEFINITION 3.1 (**DPKE**). Distributed public key encryption (DPKE) *schemes are defined by a tuple of algorithms and protocols* $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Ref})$ *where:*

- *Key generation* $(pk, sk_1, sk_2) \leftarrow \mathsf{Gen}(1^n)$ *is a randomized algorithm that given a security parameter* n *outputs the public key* pk *and the secret key shares* sk_1, sk_2 *given to devices* P_1 *and* P_2 *respectively.*[5]

- *Encryption* $c \leftarrow \mathsf{Enc}_{pk}(m)$ *is a randomized algorithm that given the public key* pk *and a message* m *outputs a ciphertext* c. *We denote* $c \leftarrow \mathsf{Enc}_{pk}(m; r)$ *to emphasize the randomness* r *used by the encryption algorithm.*

- *Decryption* $m \leftarrow \mathsf{Dec}_{pk, sk_1, sk_2}(c)$ *is a randomized 2-party protocol executed by* P_1 *and* P_2. *The input of* P_i *(*$i = 1, 2$*) is* (pk, sk_i, c) *for* pk *the public key,* sk_i *the secret key share of* P_i*, and* c *a ciphertext. The output (outputted by either* P_1 *or* P_2 *or both) is a message* m'. *We require that for all* m, $c \leftarrow \mathsf{Enc}_{pk}(m)$ *and* $m' \leftarrow \mathsf{Dec}_{pk, sk_1, sk_2}(c)$, *it holds that* $m' = m$.

- *Refreshing* $(sk_1', sk_2') \leftarrow \mathsf{Ref}_{pk}(sk_1, sk_2)$ *is a randomized 2-party protocol executed by* P_1 *and* P_2. *The input of* P_i *(*$i = 1, 2$*) is* (pk, sk_i) *for* pk *the public key and* sk_i *the current secret key share of* P_i. *The output is new secret key shares* sk_1', sk_2' *given to* P_1, P_2 *respectively (the old shares* sk_1, sk_2 *are erased). We require that for all* $(pk, sk_1^0, sk_2^0) \leftarrow \mathsf{Gen}(1^n)$, $t^* \in \mathbb{N}$, *and* $(sk_1^t, sk_2^t) \leftarrow \mathsf{Ref}_{pk}(sk_1^{t-1}, sk_2^{t-1})$ *for* $t \in [t^*]$, *it holds that*

$$SD((sk_1^0, sk_2^0), (sk_1^t, sk_2^t)) = 0$$

(where the probability is taken over the random coins of Gen *and* Ref*).*

We point out that our modeling assumes that the devices trust each other to follow the protocols specifications for refreshing and decryption *honestly.*

[4]The partition to time-periods may in fact be defined by executions of the refresh protocol, where each execution marks the end of a time-period.

[5]Without loss of generality Gen is executed by a trusted third party; otherwise, replace the trusted party by a generic 2-party protocol revealing nothing except its output; where the latter holds even if the adversary obtains during key generation the number of leakage bits considered in this work.

3.2 Continual Memory Leakage from DPKE

The memory of each computing device P_i ($i = 1, 2$) in a DPKE scheme $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Ref})$ is viewed as composed of two types of memory: (i) *Public memory* that stores the public key, the public randomness used by the system, and the public inputs and outputs of the computations executed by device P_i; and (ii) *Secret memory* that stores the share sk_i of the secret key, as well as the secret randomness of P_i, and the intermediate steps in the computations executed by P_i.

An adversary attacking the scheme can see all the following: (1) The transcript of communication to and from the devices; (2) The contents of the public memory of both devices in its entirety; and (3) A limited amount of information about the secret memory of both devices as defined next (aka, "leakage").

Essentially, during each time-period the adversary can choose a pair of polynomial time length shrinking (or entropy shrinking) functions, one per device, and receive the value of the respective function on the secret memory of the respective device. Information leakage can happen at all times, including during key refresh. The choice of the leakage function can be *adaptive* based on all public information (including the communication transcript) up to and including the current time period, and all leakage bits obtained in earlier time periods. The precise details follow.

Leakage functions. We distinguish three phases throughout the life time of the system: key generation phase, refresh phase, all other times. The content of the secret memory – and hence the input to the leakage function – varies between those phases. To address these varying inputs the adversary actually chooses two (polynomial time and length shrinking) functions per device P_i ($i = 1, 2$) at each time period t, denoted by $h_i^t, h_i^{t,\mathsf{Ref}}$, where: h_i^t is a function to be applied on the secret memory of P_i at time period t other than during refresh, and $h_i^{t,\mathsf{Ref}}$ is a function to be applied on the secret memory of P_i during refresh of time period t. In addition, the adversary chooses a (polynomial time and length shrinking) function h^{Gen} for the key generation phase.

Leakage functions can be chosen *adaptively* based on leakage and public information as said above. We point out that dependence on leakage bits in this adaptive choice is on leakage obtained in earlier time periods, not in the current one. Without loss of generality this is modeled by requiring a simultaneous choice of the leakage functions for the same time period t, namely, $(h_1^t, h_1^{t,\mathsf{Ref}}, h_2^t, h_2^{t,\mathsf{Ref}})$.

Inputs to leakage functions. For each of the aforementioned phases we specify the content of the secret memory, or equivalently, the input to the leakage function. Without loss of generality we define the latter to be solely the essential parts of the secret memory, namely, parts from which the entire secret memory is efficiently computable (given the public memory). The input to h^{Gen} is the secret randomness r^{Gen} held in memory during the key generation algorithm. The input to $h_i^{t,\mathsf{Ref}}$ and h_i^t is $(sk_i^t, r_i^{t,\mathsf{Ref}})$ and (sk_i^t, r_i^t) respectively, where: sk_i^t is the secret key share of P_i at time t; $r_i^{t,\mathsf{Ref}}$ is the secret randomness held in memory of P_i during the execution of the refresh protocol at time t; and r_i^t is the secret randomness held in memory of P_i at time t other than during the execution of the refresh protocol.

Looking ahead, to simplify the description of the security game in our security definitions we include in the input to the leakage function also the *public information* held in memory during the current time period, denoted pub^t. This includes the communication transcript to and from both devices and the content of their public memory. This captures the adversary that first sees the public information and only then chooses its leakage functions (by encoding both the choice of leakage functions and their functionalities into the submitted functions), while simplifying the security game by allowing the leakage functions to be chosen once at the beginning of each time period.

Outputs of leakage functions. The length shrinking restriction on the leakage functions says that the sum of outputs length of the functions leaking while the share sk_i^t is in memory — that is, the functions h_i^t and $h_i^{(t-1),\mathsf{Ref}}$ — is upper bounded by a pre-specified bound b_i ($i = 1, 2$). Similarly, the output length of h^{Gen} is upper bounded by a pre-specified bound b_0. Namely, the output length is upper bounded by $\left|h^{\mathsf{Gen}}(r^{\mathsf{Gen}})\right| \leq b_0$ and $\left|h_i^t(sk_i^t, r_i^t, pub^t)\right| + \left|h_i^{(t-1),\mathsf{Ref}}(sk_i^{(t-1)}, r_i^{(t-1),\mathsf{Ref}}, pub^{(t-1)})\right| \leq b_i$ for $i = 1, 2$ (for $|x|$ denoting the binary representation length of x).

Leakage rate is the ratio between the number of bits being leaked (per device) per time period and the size of the secret memory of the device at that time. Namely, the leakage rate is specified by five parameters $(\rho^{\mathsf{Gen}}, \rho_1^{\mathsf{Ref}}, \rho_2^{\mathsf{Ref}}, \rho_1, \rho_2)$ for $\rho^{\mathsf{Gen}} = b_0/\left|r^{\mathsf{Gen}}\right|$ the leakage rate during key generation; and $\rho_i^{\mathsf{Ref}} = b_i/(|sk_i| + \left|r_i^{\mathsf{Ref}}\right|)$ and $\rho_i = b_i/(|sk_i| + |r_i|)$ the rates of leakage from P_i ($i = 1, 2$) during key refresh, and during all other (post key generation) times, respectively.

3.3 Security Definitions

Our security definitions are the natural augmentation of the standard definitions by allowing the adversary to obtain continual memory leakage for as long as it chooses:

A DPKE scheme is *semantically secure against* (b_0, b_1, b_2)-*continual memory leakage* (*CPA-secure against* (b_0, b_1, b_2)-*CML*, in short) if every PPT adversary has at most a negligible advantage in winning the variant of the semantic security game, where before seeing the challenge ciphertext the adversary can receive leakage on the secret memory of both devices (as described above) for as many time periods as the adversary chooses. Here b_0, b_1, b_2 are the upper bounds for the length shrinking property as discussed above.

To model leakage on the memory state during decryption we include in the semantic security game executions of the decryption protocol. We remark that in the single processor model [11, 15] such executions are not included, because there the leakage functions is given the entire secret key and thus can simulate such executions. In contrast, in our distributed setting, the leakage function is given only one of the two secret key shares as its input and thus it cannot simulate such executions. The input ciphertexts for these executions are drawn from a (polynomial-time sampleable) distribution $\mathcal{C}$. This distribution should be thought of as modeling executions of the decryption protocol run in the background, say, by other users of the scheme (note, the adversary has no control on the choice of decryption input in a semantic security game). To simplify the presentation we assume that a single execution of the decryption protocol occurs at each time period (as achieved, say, by frequent refreshing). Extensions allowing multiple executions of the decryption protocol at each time period are simple; details omitted from this extended abstract.

DEFINITION 3.2. *A DPKE scheme* $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Ref})$ *is* semantically secure against (b_0, b_1, b_2)-continual memory leakage (CPA-secure against (b_0, b_1, b_2)-CML) *if for every* PPT *algorithm* $\mathcal{C} = \mathcal{C}(n, pk, t)$ *for sampling ciphertexts, every* PPT *adversary has at most a negligible advantage over 1/2 in wining the following game:*

1. **Key Generation Phase.** *The challenger generates* $(pk, sk_1^0, sk_2^0) \leftarrow \mathsf{Gen}(1^n)$*, and sends the adversary* pk*.*

2. **Leakage on Key Generation.** *The adversary chooses a polynomial-time computable function* h^{Gen}*.*

   Denote the requested leakage by $\ell^{\mathsf{Gen}} = h^{\mathsf{Gen}}(r^{\mathsf{Gen}})$ *for* r^{Gen} *the secret randomness held in memory during the execution of the key generation algorithm.*

   If $\left|\ell^{\mathsf{Gen}}\right| \leq b_0$ *then the challenger returns to the adversary* ℓ^{Gen}*, and sets* $L_i^0 \leftarrow \left|\ell^{\mathsf{Gen}}\right|$, $L_i^{0,\mathsf{Ref}} \leftarrow \left|\ell^{\mathsf{Gen}}\right|$ *(for* $i = 1, 2$*) and* $t \leftarrow 0$*; otherwise, the challenger aborts.*

3. **Leakage at Every Time Period.** *The adversary chooses a tuple* $(h_1^t, h_1^{t,\mathsf{Ref}}, h_2^t, h_2^{t,\mathsf{Ref}})$ *of polynomial-time computable functions. In response the challenger draws a random ciphertext* $c \leftarrow \mathcal{C}(n, pk, t)$*, and executes the decryption and the key refresh protocols*

$$m \leftarrow \mathsf{Dec}_{pk, sk_1^t, sk_2^t}(c)$$
$$(sk_1^{t+1}, sk_2^{t+1}) \leftarrow \mathsf{Ref}_{pk}(sk_1^t, sk_2^t)$$

   Denote the requested leakage from computing device P_i *(*$i = 1, 2$*) by*

$$\ell_i^t = h_i^t(sk_i^t, r_i^t, pub^t)$$
$$\ell_i^{t,\mathsf{Ref}} = h_i^{t,\mathsf{Ref}}(sk_i^t, r_i^{t,\mathsf{Ref}}, pub^t)$$

   for sk_i^t *the secret key share of* P_i *at time* t*;* $(r_i^{t,\mathsf{Ref}}, r_i^t)$ *the secret randomness held in memory of* P_i *at time* t *during and not during the execution of the refresh protocol, respectively; and* $pub^t = (\mathsf{comm}^t, c, m)$ *the public information at time* t *consisting of the communication transcript* comm^t *to and from devices* P_1, P_2*, and of the input/output to the decryption protocol* c *and* m *held in their public memory.*

   If $L_i^t + \left|\ell_i^t\right| + \left|\ell_i^{t,\mathsf{Ref}}\right| \leq b_i$ *for* $i = 1, 2$*, then the challenger returns to the adversary* $(\ell_1^t, \ell_1^{t,\mathsf{Ref}}, \ell_2^t, \ell_2^{t,\mathsf{Ref}})$*, sets* $L_i^{(t+1)} \leftarrow \left|\ell_i^{t,\mathsf{Ref}}\right|$ *for* $i = 1, 2$*, and sets* $t \leftarrow t + 1$*; otherwise, the challenger aborts.*

4. **Challenge Phase.** *The adversary sends to the challenger two messages* m_0, m_1 *of equal length* $|m_0| = |m_1|$*. The challenger sends to the adversary the ciphertext* $c \leftarrow \mathsf{Enc}_{pk}(m_b)$ *for a uniformly random bit* $b \in \{0, 1\}$*. The adversary outputs* $b' \in \{0, 1\}$*. The adversary wins if* $b' = b$*.*

We call (b_0, b_1, b_2) *the* leakage parameter of the game.

Likewise, a DPKE scheme is CCA2-secure against (b_0, b_1, b_2)-continual memory leakage (CCA2-secure against (b_0, b_1, b_2)-CML, in short), if every PPT adversary has at most a negligible advantage in winning the extension of the game specified in Definition 3.2, where the adversary is given extra power in the form of access to a decryption oracle (namely, an oracle that given ciphertexts c' returns messages $m \leftarrow \mathsf{Dec}(c')$); where the only restriction is that the adversary does not query the decryption oracle on the challenge ciphertext. Note that leakage occurs only *prior* to seeing the challenge ciphertext (as in the semantic-security game).

4. RESULTS & TECHNIQUES OVERVIEW

We give an overview of our constructions: Our DPKE scheme semantically secure against continual leakage (Section 4.1); our DIBE scheme semantically secure against continual leakage (Section 4.2); Our DPKE scheme CCA2-secure against continual leakage (Section 4.3). We name these schemes $\mathbf{DLR}$, $\mathbf{DLR}_{\text{IBE}}$ and $\mathbf{DLR}_{\text{CCA2}}$, respectively (for distributed leakage resilient). The theorem below summarizes our main results.

THEOREM 4.1 (MAIN). *The following holds under BDDH and 2Lin assumptions:*

1. $\mathbf{DLR}$ *is a DPKE scheme that is CPA-secure against* (b_0, b_1, b_2)*-CML*
2. $\mathbf{DLR}_{\text{IBE}}$ *is a DIBE scheme that is CPA-secure against* (b_0, b_1, b_2)*-CML*
3. $\mathbf{DLR}_{\text{CCA2}}$ *is a DPKE scheme that is CCA2-secure against* (b_0, b_1, b_2)*-CML*

where $(b_0, b_1, b_2) = \left(\Omega(\log n), \left(1 - \frac{cn}{\lambda + cn}\right) m_1, m_2\right)$ *for* m_1, m_2 *the size of the secret key shares of* P_1, P_2 *respectively;*[6] n, λ *the security and leakage parameters of our schemes; and* $c > 0$ *a constant.*

Proof: The heart of our analysis is the proof of part 1 of the theorem; details appear in section 6. Proving parts 2 and 3 is simple given the former (cf. Sections 4.2-4.3 and the full version of this paper). ∎

The leakage rate tolerated by our schemes as derived from Theorem 4.1 is: $\rho^{\mathsf{Gen}} = o(1)$, $(\rho_1, \rho_2) = (1 - o(1), 1)$ and $(\rho_1^{\mathsf{Ref}}, \rho_2^{\mathsf{Ref}}) = (1/2 - o(1), 1/2)$. This holds because in our schemes the size of the secret memory of P_1 and P_2 (other than during refresh) is $m_1 + \log p$ and m_2 respectively; and it is $2m_1 + \log p$ and $2m_2$ respectively during refresh (when P_1, P_2 hold both the current and the next secret key shares). Our proof shows also a stronger bound on the leakage rate tolerated during refresh, showing it is $\rho_2^{\mathsf{Ref}} = 1$.

REMARK 4.1. *For our DIBE scheme* $\mathbf{DLR}_{\text{IBE}}$*, we note the following. First, the above leakage bounds hold both when* P_1, P_2 *are sharing the master secret key and when they are sharing an identity based secret key. Second, when generating the identity based secret key the leakage upper bound is* b_1, b_2 *bits from the secret memory of* P_1, P_2 *respectively. That is, the more restrictive leakage upper bound of* b_0 *bits applies only during generation of the* master *secret key.*

4.1 Overview of DPKE

The scheme $\mathbf{DLR} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Ref})$ builds on (modifications of) the IBE scheme of Boneh-Boyen (BB) [5] and the key-dependent/leakage-resilient encryptions of Boneh-Halevy-Hamburg-Ostrovsky (BHHO) [8] / Naor-Segev (NS) [32] as described next.

[6] Our schemes tolerate leakage of $b_0 = n^\epsilon$ bits during key generation when assuming sub-exponential hardness of BDDH.

The *public key* of **DLR** is $pk = (p, g, e, g_1 = g^\alpha, g_2)$ for $g, e(g,g)$ generating groups $\mathbb{G}, \mathbb{G}_T$ of prime order p that are connected by a bilinear map $e\colon \mathbb{G} \to \mathbb{G}_T$, and for independent and uniformly random $\alpha \in \mathbb{Z}_p$ and $g_2 \in \mathbb{G}$. Note that the public key is the public parameters in BB's IBE.

The *secret key shares* are a sharing of the master secret key in BB's IBE $msk = g_2^\alpha$ using a secret sharing scheme that we construct to be refreshable, leakage resilient, and with the special property that it allows to decrypt ciphertexts of **DLR** without reconstructing its underlying secret key g_2^α. This secret sharing scheme is constructed using a secondary symmetric key encryption scheme $\Pi_{\mathsf{ss}} = (\mathsf{Gen}_{\mathsf{ss}}, \mathsf{Enc}_{\mathsf{ss}}, \mathsf{Dec}_{\mathsf{ss}})$, where the key generation algorithm $\mathsf{Gen}_{\mathsf{ss}}$ chooses $sk_{\mathsf{ss}} = (s_1, \ldots, s_\ell)$ for independent and uniformly random $s_i \in \mathbb{Z}_p$; the encryption algorithm $(a_1, \ldots, a_\ell, m \cdot \prod_{i\in[\ell]} a_i^{s_i}) \leftarrow \mathsf{Enc}_{\mathsf{ss}}(m)$ outputs ciphertexts for independent and uniformly random $a_i \in \mathbb{G}$; and the decryption algorithm $\mathsf{Dec}_{\mathsf{ss}}(c_1, \ldots, c_\ell, c_0)$ outputs $c_0 / \left(\prod_{i\in[\ell]} {c_i}^{s_i}\right)$. We use Π_{ss} to secret share g_2^α as follows. Device P_2 is given $(s_1, \ldots, s_\ell)$ the secret key of Π_{ss}, whereas P_1 is given $(a_1, \ldots, a_\ell, g_2^\alpha \cdot \prod_{i\in[\ell]} a_i^{s_i}) \leftarrow \mathsf{Enc}_{\mathsf{ss}}(g_2^\alpha)$ a ciphertext Π_{ss} encrypting g_2^α. We note that those secret key shares are a sharing of the aforementioned master secret key $msk = g_2^\alpha$ in a leakage resilient way inspired by BHHO/NS techniques. Consequently, Π_{ss} is resilient to bounded leakage (by the leftover hash lemma); this will be crucial for our security proof for **DLR**.

Encryption is a simplified version of BB's IBE encryption. Specifically, given a message $m \in \mathbb{G}_T$, the encryption algorithm outputs a ciphertext $(g^t, m \cdot e(g_1, g_2)^t) \leftarrow \mathsf{Enc}_{pk}(m)$ for a uniformly random $t \in \mathbb{Z}_p$.

Decryption and *Refreshing* are executed via 2-party protocols. We describe the refresh protocol in details. At the outset of the refresh protocol P_1 holds a ciphertext of Π_{ss} encrypting g_2^α and P_2 holds the secret key of Π_{ss}. After each refresh stage, P_1 will hold a new random encryption of g_2^α in Π_{ss} under a new random key sk_{ss} to be held by P_2. To this end, we make use of yet another encryption scheme, denoted by $\Pi_{\mathsf{comm}} = (\mathsf{Gen}', \mathsf{Enc}', \mathsf{Dec}')$, with useful homomorphic properties that allow us to do the following. During refresh P_1 runs Gen' and generates a secret key sk_{comm}, together with fresh new randomness a_i' (aimed to replace the a_i's), and sends ciphertexts of Π_{comm} encrypting its share and a_i''s to P_2. Upon receiving these, P_2 picks fresh new s_i''s (aimed to replace the s_i's) and sends back a ciphertext of Π_{comm} encrypting $(a_1', \ldots, a_\ell', g_2^\alpha \cdot \prod_{i\in[\ell]} {a_i'}^{s_i'})$. P_2 will be able to do so, although sk_{comm} is *unknown* to it, due to the homomorphic nature of Π_{comm}. Note that s_i's and a_i's are never stored unencrypted on the same device.

For our scheme to be secure we require yet one more property of Π_{comm}. Specifically, we require that ℓ independent and uniformly random plaintexts have sufficient (pseudo average min-) entropy even when conditioned on their ciphertexts in Π_{comm} together with bounded leakage from the secret key sk_{comm} and random coins used to generate these ciphertexts, as well as leakage on the plaintexts themselves. This property allows us to prove security against the adversary that chooses leakage functions based on the communication between the devices (i.e., the ciphertexts of Π_{comm}), and where the leakage is on the secret key of Π_{comm}, the random coins used for generating the ciphertexts, and on the plaintexts themselves (as all are held in the memory of P_1). We name secret key encryption schemes Π_{comm} achieving the above two properties *homomorphic proxy secret key encryption (HPSKE)*; cf. Definition 5.1, Section 5.1.

Decryption follows in a similar manner where the parties run a protocol implementing BB's decryption algorithm.

4.2 DIBE Semantically Secure against CML

Our distribute identity based encryption semantically secure against continual leakage, named, $\mathbf{DLR}_{\mathrm{IBE}}$, is an extension of our DPKE scheme **DLR**, where both the master secret key and the identity based secret keys are shared among two computing devices. The master secret key shares and their refreshing protocol are identical to the secret key shares and their refreshing in **DLR**. The identity based secret key shares are a sharing of the identity based secret keys in BB's IBE, while following our leakage resilient techniques for key sharing discussed above. Specifically, the BB's identity based secret key is $sk_{\mathcal{ID}} = (g^{r_1}, \ldots, g^{r_n}, M = g_2^\alpha \cdot \prod_{j\in[n]} u_{j,b_j}^{r_j})$ where $H(\mathcal{ID}) = (b_1, \ldots, b_n) \in \mathbb{Z}_p^n$ is the evaluation of an appropriate hash function H on the underlying identity $\mathcal{ID}$, and $U = (u_{i,j}) \in \mathbb{G}^{n\times 2}$ is a uniformly random matrix. Our sharing of $sk_{\mathcal{ID}}$ is with the two shares $sk^1_{\mathcal{ID}} = (g^{r_1}, \ldots, g^{r_n}, a_1', \ldots, a_\ell', M \cdot \prod_{i\in[\ell]} {a_i'}^{s_i'})$ and $sk^2_{\mathcal{ID}} = (s_1', \ldots, s_\ell')$. Refreshing these shares is a straightforward extension of the protocol for refreshing the secret key shares in **DLR**. Likewise, decryption is a straightforward extension of **DLR** decryption protocol.

4.3 DPKE CCA2-Secure against CML

Our DPKE CCA2-secure against continual memory leakage, named, $\mathbf{DLR}_{\mathrm{CCA2}}$, is derived from our DIBE semantically secure against continual memory leakage discussed above by using a general purpose transformation from semantically secure IBE to CCA2-secure PKE. For the single processor setting where there is no leakage, such a transformation was given by Boneh et al. [6]. We use the same transformation, while extending their proof to show that CCA2-security holds even in the presence of continual leakage (as long the IBE is secure against continual leakage). Our proof (straightforwardly) extended to the distributed setting as well.

4.4 Secure Storage on Leaky Devices

Our DPKE scheme can also be used for securely storing data on continually leaky devices. We point out that storing a secret on continually leaky devices is a special case of our problem as we must implicitly maintain the secret "decryption key" of the decryption algorithm throughout its continual execution in a way that still allows to decrypt.

5. DPKE CPA-SECURE AGAINST CML

We present the details of our DPKE scheme semantically secure against continual memory leakage, named, **DLR**. For this purpose we first present the HPSKE that we use.

Throughout this section $\lambda > 0$ is a leakage parameter of the scheme; n is the security parameter; $(p, g, e) \leftarrow \mathcal{G}(1^n)$ is the output of a parameters generating algorithm (cf. Section 2) with $\mathbb{G}, \mathbb{G}_T$ denoting the order p groups generated by g and $e(g,g)$ respectively; $\mathbb{G}'$ is a group of order p (to be thought of as either $\mathbb{G}$ or $\mathbb{G}_T$); and $\ell = 7 + 3\kappa + \frac{2\log(1/\epsilon)}{\log p}$ for $\epsilon = 2^{-n}$ and $\kappa = 1 + \frac{\lambda + 2\log(1/\epsilon)}{\log p}$.

5.1 Building Block: HPSKE

We present the primitive we named: homomorphic proxy secret key encryption (HPSKE), and a construction for it.

DEFINITION 5.1 (HPSKE.). *A* homomorphic proxy secret key encryption (HPSKE) for $\ell, \mathbb{G}'$ *is a secret key encryption scheme* $\Pi_{\mathsf{comm}} = (\mathsf{Gen}', \mathsf{Enc}', \mathsf{Dec}')$ *with message space* $\mathbb{G}'$ *and ciphertexts that are tuples of elements from* $\mathbb{G}'$*, such that the following holds for* $sk_{\mathsf{comm}} \leftarrow \mathsf{Gen}'(1^n)$*:*

1. *For every two messages* $m_0, m_1 \in \mathbb{G}'$ *and their ciphertexts* $c_i \leftarrow \mathsf{Enc}'_{sk_{\mathsf{comm}}}(m_i)$ $(i = 0, 1)$*, it holds that*

$$m_0 m_1 \leftarrow \mathsf{Dec}'_{sk_{\mathsf{comm}}}(c_0 c_1)$$

(where the product $c_0 c_1$ *is computed coordinate-wise).*

2. *Given ciphertexts* $c_1 \leftarrow \mathsf{Enc}'_{sk_{\mathsf{comm}}}(m_1; r_1), \ldots, c_\ell \leftarrow \mathsf{Enc}'_{sk_{\mathsf{comm}}}(m_\ell; r_\ell)$ *for independent and uniformly random plaintexts* $m_1, \ldots, m_\ell \in \mathbb{G}'$*, and given leakage* $L = h(sk_{\mathsf{comm}}, m_1, \ldots, m_\ell, r_1, \ldots, r_\ell)$ *for* h *a polynomial-time computable function of output length upper bounded by* λ*, there is still sufficient pseudo average min-entropy left in the plaintexts. Namely: There exists a computationally indistinguishable distribution*

$$((m_i, c'_i)_{i\in[\ell]}, L') \approx_c ((m_i, c_i)_{i\in[\ell]}, L)$$

for ciphertexts $c'_i \leftarrow \mathsf{Enc}'_{sk_{\mathsf{comm}}}(m_i; r'_i)$ *and leakage* $L' = h(sk_{\mathsf{comm}}, m_1, \ldots, m_t, r'_1, \ldots, r'_t)$ *such that*

$$\tilde{H}_\infty(m_1, \ldots, m_t \mid \ c'_1, \ldots, c'_t, L') \geq \log p + 2\log(1/\epsilon)$$

We say that Π_{comm} *is a "HPSKE for* $\ell, \mathbb{G}, \mathbb{G}_T$*" if it is a HPSKE for both* $\ell, \mathbb{G}$ *and* $\ell, \mathbb{G}_T$*.*

REMARK 5.1. *Definition 5.1, Part 2, does not follow from existing notions of security against leakage (to the best of our knowledge). For example, the bounded leakage model [1, 32] does not allow leakage to depend on the challenge ciphertexts; and the after-the-fact leakage model [25] does not allow leakage to depend on the encryption randomness.*

LEMMA 5.2 (HPSKE EXISTS). *There exists a HPSKE scheme for* $\ell, \mathbb{G}, \mathbb{G}_T$ *(under 2Lin assumption).*

Proof: Fix $\mathbb{G}' \in \{\mathbb{G}, \mathbb{G}_T\}$. We show that the scheme Π_{comm} defined next is a HPSKE for $\ell, \mathbb{G}'$ (the proof is omitted from this extended abstract): The key generation algorithm $\mathsf{Gen}'(1^n)$ outputs a uniformly random secret key $sk_{\mathsf{comm}} = (\sigma_1, ..., \sigma_\kappa)$ in $\mathbb{Z}_p^\kappa$. The encryption algorithm $\mathsf{Enc}'_{sk_{\mathsf{comm}}}(m)$ outputs ciphertexts $(b_1, \ldots, b_\kappa, m \cdot \prod_{i\in[\kappa]} b_i^{\sigma_i})$ for independent and uniformly random $b_i \in \mathbb{G}'$. The decryption algorithm $\mathsf{Dec}'_{sk_{\mathsf{comm}}}(b_1, ..., b_\kappa, b_0)$ outputs $b_0 / \left(\prod_{i\in[\kappa]} b_i^{\sigma_i}\right)$. ∎

5.2 Construction of DLR

We present the scheme **DLR**. As a building block we use a HPSKE scheme $\Pi_{\mathsf{comm}} = (\mathsf{Gen}', \mathsf{Enc}', \mathsf{Dec}')$ for $\ell, \mathbb{G}, \mathbb{G}_T$ (say, as given in Section 5.1, Lemma 5.2).

CONSTRUCTION 5.3 (**DLR**). *The DPKE scheme* $\mathbf{DLR} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Ref})$ *is defined by the following algorithms and protocols:*

- $\mathsf{Gen}(1^n)$ *is an algorithm that, given a security parameter* n*, outputs the public key and secret key shares:*

$$pk = (p, g, e, e(g_1, g_2))$$
$$sk_1 = \left(a_1, \ldots, a_\ell, \Phi = g_2^\alpha \cdot \textstyle\prod_{i\in[\ell]} {a_i}^{s_i}\right)$$
$$sk_2 = (s_1, \ldots, s_\ell)$$

for independent and uniformly random $\alpha \in \mathbb{Z}_p$*,* $g_2 \in \mathbb{G}$*,* $s_i \in \mathbb{Z}_p$*, and* $a_i \in \mathbb{G}$*; and for* $g_1 = g^\alpha$*.*

- $\mathsf{Enc}_{pk}(m)$ *is an algorithm that, given a message* $m \in \mathbb{G}_T$*, outputs* $\left(g^t, m \cdot e(g_1, g_2)^t\right)$ *for a uniformly random* $t \in \mathbb{Z}_p$*.*

- $\mathsf{Dec}_{pk, sk_1, sk_2}(c)$ *is the following 2-party protocol executed by* P_1 *and* P_2 *on a given ciphertext* $c = (A, B)$*:*

  1. P_1 *samples a key* $sk_{\mathsf{comm}} \leftarrow \mathsf{Gen}'(1^n)$*, and sends to* P_2 *the ciphertexts of* Π_{comm}*:* $\mathsf{Enc}'_{sk_{\mathsf{comm}}}(e(A, a_1)), \ldots, \mathsf{Enc}'_{sk_{\mathsf{comm}}}(e(A, a_\ell)), \mathsf{Enc}'_{sk_{\mathsf{comm}}}(e(A, \Phi))$*, and* $\mathsf{Enc}'_{sk_{\mathsf{comm}}}(B)$*.*
  2. *Upon receiving* $(d_1, ..., d_\ell, d_\Phi, d_B)$ *from* P_1*,* P_2 *sends to* P_1 *the coordinate-wise product* $d_B \cdot \prod_{i\in[\ell]} d_i^{s_i} / d_\Phi$*.*
  3. *Upon receiving* c' *from* P_2*,* P_1 *outputs* $\mathsf{Dec}'_{sk_{\mathsf{comm}}}(c')$*.*

- $\mathsf{Ref}_{pk}(sk_1, sk_2)$ *is the following 2-party protocol executed by* P_1 *and* P_2 *on their secret key shares* $sk_1 = (a_1, \ldots, a_\ell, \Phi = g_2^\alpha \prod_{i\in[\ell]} a_i^{s_i})$ *and* $sk_2 = (s_1, \ldots, s_\ell)$*:*

  1. P_1 *chooses independent and uniformly random* $a'_1, \ldots, a'_\ell \in \mathbb{G}$*, and sends to* P_2 *the ciphertext of* Π_{comm}*:* $(\mathsf{Enc}'_{sk_{\mathsf{comm}}}(a_i), \mathsf{Enc}'_{sk_{\mathsf{comm}}}(a'_i))$ *for* $i \in [\ell]$ *and* $\mathsf{Enc}'_{sk_{\mathsf{comm}}}(\Phi)$*.*
  2. *Upon receiving* $\left((f_i, f'_i)_{i\in[\ell]}, f_\Phi\right)$*,* P_2 *chooses a uniformly random* $(s'_1, \ldots, s'_\ell) \in \mathbb{Z}_p^\ell$*, and sends to* P_1 *the coordinate-wise product:* $\left(\prod_{i\in[\ell]} f_i^{\prime s'_i} / f_i^{s_i} \cdot f_\Phi\right)$*. Next,* P_2 *replaces its old secret key share by*

$$sk_2 = \left(s'_1, \ldots, s'_\ell\right).$$

  3. *Upon receiving* f*,* P_1 *computes* $\Phi' = \mathsf{Dec}'_{sk_{\mathsf{comm}}}(f)$ *and replaces its old secret key share by*

$$sk_1 = \left(a'_1, .., a'_\ell, \Phi'\right).$$

Remarks

To ease the reading of our scheme in the above we overlooked some necessary implementation choices. We next specify those choices.

Optimal leakage rate. To achieve a better leakage rate we slightly change the above: In the above, we defined P_1 as holding *both* the secret key share sk_1 and the secret key sk_{comm} for encrypting the communication. To reach leakage rate $(1 - o(1))$ from P_1 we reduce the size of the secret memory of P_1 by defining it to hold only sk_{comm}; whereas instead of holding the secret key share sk_1, P_1 holds the public (coordinate-wise) encryption of sk_1 under Π_{comm}. (The latter is public as it is to be transmitted over the public channel.) We then adapt the decryption and refresh protocol so that P_1 never holds in its memory more than a single un-encrypted coordinate of sk_1. With these modifications, the secret memory of P_1 is of size $|sk_{\mathsf{comm}}| + \log p$, and our tolerated leakage is a $(1 - o(1))$-fraction of this size.

Reusing ciphertexts and hiding discrete logs of random coins. We observe that the ciphertexts f_i's and d_i's encrypt the same set of values, only in two different groups, and reuse the ciphertexts f_i's to compute the ciphertexts d_i's. This enables us to simplify our security proof. Specifically, for every time period t, P_1 first computes the f_i's to be ciphertexts $f_i = (b_{i1}, \ldots, b_{i\kappa}, a_i \cdot \prod_{j\in[\kappa]} b_j^{\sigma_j}) \leftarrow \mathsf{Enc}'_{sk_{\mathsf{comm}}}(a_i; b_i)$ computed using fresh randomness $b_i = (b_{i1}, \ldots, b_{i\kappa})$ and the secret key $sk_{\mathsf{comm}} = (\sigma_1, \ldots, \sigma_\kappa)$; then computes $d_i = (e(A, b_{i1}), \ldots, e(A, b_{i\kappa}), e(A, a_i) \cdot \prod_{j\in[\kappa]} e(A, b_j)^{\sigma_j})$ to be the coordinate-wise pairing of f_i with A (for $c = (A, B)$ the ciphertext given as input to the decryption protocol at time period t). P_1 then sends these d_i's and f_i's during the decryption and refresh protocol (respectively) of time t.

Looking ahead, for proving **DLR** is secure we require that the discrete logarithms of the random coins b_{ij} (similarly, the a_i's) are not exposed to leakage. To achieve this we sample these elements directly as random group elements (rather than first choosing a random exponent r_{ij} and then defining $b_{ij} = g^{r_{ij}}$). This is feasible in the groups used in our scheme.

6. OUR SECURITY PROOF FOR DLR

We give an overview of our proof of Theorem 4.1, Part 1, stating that our scheme **DLR** is semantically secure against continual memory leakage.

Our proof is by a reduction to the BDDH and 2Lin assumptions: We assume for contradiction that there exists a PPT adversary $\mathcal{A}$ winning the semantic security game for **DLR** (aka, the real game) with non-negligible advantage over half; and where the leakage parameter for this semantic security game is $(b_0 = 0, b_1 = \lambda, b_2 = |sk_2|)$ for λ the leakage parameter of **DLR**, and $|sk_2|$ the size of the secret key share of P_2.[7] We then show that there exists a distinguisher $\mathcal{D}$ that breaks either the BDDH assumption or the 2Lin assumption; details below. We conclude therefore that there exists no such adversary $\mathcal{A}$. Namely, **DLR** is CPA-secure against (b_0, b_1, b_2)-CML (under BDDH and 2Lin assumptions). To conclude the proof observe that $b_1 = \left(1 - \frac{3n}{\lambda+3n}\right) m_1$ for $m_1 = |sk_{\mathsf{comm}}|$ the size of the secret key share of P_1 (as $|sk_{\mathsf{comm}}| = \kappa \log p = \lambda + 3n$ for our parameters setting).

In the following we first define the distinguisher $\mathcal{D}$, and then outline our proof showing that $\mathcal{D}$ breaks either the BDDH or the 2Lin assumptions.

Defining the distinguisher $\mathcal{D}$. The distinguisher $\mathcal{D}$, given a BDDH tuple $(p, g, e, g^a, g^b, g^c, T)$, plays a fake semantic security game for **DLR** with $\mathcal{A}$ playing the role of the adversary, and outputs 1 iff $\mathcal{A}$ wins this fake game and 0 otherwise. When running this fake game, the distinguisher $\mathcal{D}$ simulates the role of the challenger, while deviating from the latter in how it generates the random variables used in the game:

First, the distinguisher $\mathcal{D}$ plants the BDDH tuple as part of the public key and the challenge ciphertext. Specifically, the public key is $pk = (p, g, e, e(g^a, g^b))$; and the challenge ciphertext is $C^{\mathsf{fake}} = (g^c, m_b \cdot T)$ for m_0, m_1 the messages sent to the challenger from the adversary $\mathcal{A}$, and $b \in \{0, 1\}$ uniformly random.

[7]Extending our proof to address leakage $b_0 > 0$ during key generation is simply by guessing those leakage bits; details omitted from this extended abstract.

Second, the distinguisher $\mathcal{D}$ samples the remaining random variables from a new distribution where, most notably, sk_1 is chosen uniformly at random, and yet, despite using this flawed share, the decryption protocol produces the correct output. To specify this distribution we fix a time period t and drop its indices (albeit the sampling itself actually takes place at once for all time periods of the game): (a) $sk_1 = (a_1, \ldots, a_\ell, \Phi) \in \mathbb{G}^{\ell+1}$ and $sk_{\mathsf{comm}} \in \mathbb{Z}_p^\kappa$ are chosen independently and uniformly random; (b) $c', d_\Phi, d_B, f_\Phi, f_i, f'_i$ are ciphertexts of Π_{comm} encrypting the plaintexts M, $e(A, \Phi)$, B, Φ, a_i, a'_i under secret key sk_{comm} (where $C = (A, B)$ and M are the input and output in the execution of the decryption protocol at time t, given to the distinguisher as advice; and a'_i is the i-th component in sk^{t+1}); (c) d_i is the coordinate-wise pairing of f_i and A (for $i = 1, \ldots, \ell$); (d) $sk_2 = (s_1, \ldots, s_\ell) \in \mathbb{Z}_p^\ell$ is chosen uniformly at random subject to the constraint that $c' = d_B \cdot \prod_{i\in[\ell]} d_i^{s_i} / d_\Phi$. Satisfying this constraint boils down to solving a system of $\kappa + 1$ linear equations (one equation per each component of c') in unknowns $s_1, \ldots, s_\ell$ and with coefficients the discrete logarithms of the corresponding ciphertexts. (e) $f = \prod_{i\in[\ell]} \left(f'_i{}^{s'_i} / f_i^{s_i}\right) \cdot f_\Phi$ when denoting by $sk_2^{t+1} = (s'_1, \ldots, s'_\ell)$ the next secret key share of P_2.

We elaborate on step (d). First, to ensure a solution to the said constraint exists $\mathcal{D}$ imposes a full rank requirement on the coefficients matrix (satisfied via re-sampling). Second, to ensure the solution can be found efficiently the distinguisher $\mathcal{D}$ keeps track of the discrete logarithms involved in stages (a)-(c).

$\mathcal{D}$ breaks BDDH or 2Lin. To prove that $\mathcal{D}$ breaks the BDDH or 2Lin assumptions we do the following.

First, we show that, when $T = e(g, g)^{abc}$ in the given BDDH tuple, the view of the adversary in the fake and real games is computationally indistinguishable (under 2Lin assumption). For this purpose, we define an auxiliary game that is identical to the real game except for imposing the full rank requirement as in the fake game (see above); denote by $\mathsf{real}, \mathsf{fake}$ and aux the adversary's view in the real, fake, and auxiliary games, respectively. We prove that $\mathsf{real} \approx_c \mathsf{aux}$ with overwhelming probability (under 2Lin assumption). We then prove that $\mathsf{aux} \approx_s \mathsf{fake}$ by observing that the corresponding two games differ only in how they generate the random variables $(pk, C^{\mathsf{challenge}}, sk_2^t, \Phi_t)_t$ (for Φ_t the last component of sk_1^t), and proving that the following holds (even conditioned on the rest of the view): (i) The joint distribution of $(pk, C^{\mathsf{challenge}}, sk_2^t)_t$ is identical in aux and fake; (ii) The distribution of $(\Phi_t)_t$ is statistically close in aux and fake; details are omitted from this extended abstract. We conclude that $\mathsf{real} \approx_c \mathsf{fake}$ with overwhelming probability (under 2Lin assumption).

Now, as by our contradiction assumption $\mathcal{A}$ wins the real game with a non-negligible advantage over half, we conclude that — when $T = e(g, g)^{abc}$ — the adversary $\mathcal{A}$ wins the fake game with a non-negligible advantage over half (under 2Lin assumption). Namely, when $T = e(g, g)^{abc}$, the distinguisher $\mathcal{D}$ outputs 1 with a non-negligible advantage over half. Second, we observe that, when T is uniformly random, the distinguisher $\mathcal{D}$ outputs 1 with probability at most half, because in this case the challenge ciphertext in the fake game is uniformly random and independent of the rest of the view of the adversary, namely, the adversary $\mathcal{A}$ cannot win the game with any advantage over half. Third,

we observe that when $\mathcal{A}$ is a PPT algorithm, then the distinguisher $\mathcal{D}$ is also a PPT algorithm. We conclude that (under 2Lin assumption) $\mathcal{D}$ is a PPT algorithm that distinguishes BDDH tuples with $T = e(g,g)^{abc}$ from BDDH tuples with uniform T. Namely, $\mathcal{D}$ breaks either the BDDH or the 2Lin assumptions.

7. REFERENCES

[1] A. Akavia, S. Goldwasser, and V. Vaikuntanathan. Simultaneous hardcore bits and cryptography against memory attacks. In *TCC*, pages 474–495, 2009.

[2] J. Alwen, Y. Dodis, and D. Wichs. Leakage-resilient public-key cryptography in the bounded-retrieval model. In *CRYPTO*, pages 36–54, 2009.

[3] E. Biham and A. Shamir. Differential fault analysis of secret key cryptosystems. In *CRYPTO*, pages 513–525, 1997.

[4] N. Bitansky, R. Canetti, and S. Halevi. Leakage-tolerant interactive protocols. In *TCC*, pages 266–284, 2012.

[5] D. Boneh and X. Boyen. Secure identity based encryption without random oracles. In *CRYPTO*, pages 443–459, 2004.

[6] D. Boneh, R. Canetti, S. Halevi, and J. Katz. Chosen-ciphertext security from identity-based encryption. *SIAM J. Comput.*, 36(5):1301–1328, 2007.

[7] D. Boneh, R. A. DeMillo, and R. J. Lipton. On the importance of checking cryptographic protocols for faults (extended abstract). In *EUROCRYPT*, pages 37–51, 1997.

[8] D. Boneh, S. Halevi, M. Hamburg, and R. Ostrovsky. Circular-secure encryption from decision diffie-hellman. In *CRYPTO*, pages 108–125, 2008.

[9] V. Boyko. On the security properties of oaep as an all-or-nothing transform. In *CRYPTO*, pages 503–518, 1999.

[10] E. Boyle, S. Goldwasser, and Y. T. Kalai. Leakage-resilient coin tossing. In *DISC*, pages 181–196, 2011.

[11] Z. Brakerski, Y. T. Kalai, J. Katz, and V. Vaikuntanathan. Overcoming the hole in the bucket: Public-key cryptography resilient to continual memory leakage. In *FOCS*, pages 501–510, 2010.

[12] R. Canetti, Y. Dodis, S. Halevi, E. Kushilevitz, and A. Sahai. Exposure-resilient functions and all-or-nothing transforms. In *EUROCRYPT*, pages 453–469, 2000.

[13] Y. Dodis, S. Goldwasser, Y. T. Kalai, C. Peikert, and V. Vaikuntanathan. Public-key encryption schemes with auxiliary inputs. In *TCC*, pages 361–381, 2010.

[14] Y. Dodis, K. Haralambiev, A. López-Alt, and D. Wichs. Cryptography against continuous memory attacks. In *FOCS*, pages 511–520, 2010.

[15] Y. Dodis, K. Haralambiev, A. López-Alt, and D. Wichs. Efficient public-key cryptography in the presence of key leakage. In *ASIACRYPT*, pages 613–631, 2010.

[16] Y. Dodis, Y. T. Kalai, and S. Lovett. On cryptography with auxiliary input. In *STOC*, pages 621–630, 2009.

[17] Y. Dodis, A. B. Lewko, B. Waters, and D. Wichs. Storing secrets on continually leaky devices. In *FOCS*, pages 688–697, 2011.

[18] Y. Dodis, L. Reyzin, and A. Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. In *EUROCRYPT*, pages 523–540, 2004.

[19] Y. Dodis, A. Sahai, and A. Smith. On perfect and adaptive security in exposure-resilient cryptography. In *EUROCRYPT*, pages 301–324, 2001.

[20] S. Dziembowski and K. Pietrzak. Leakage-resilient cryptography. In *FOCS*, pages 293–302, 2008.

[21] K. Gandolfi, C. Mourtel, and F. Olivier. Electromagnetic analysis: Concrete results. In *CHES*, number Generators, pages 251–261, 2001.

[22] S. Garg, A. Jain, and A. Sahai. Leakage-resilient zero knowledge. In *CRYPTO*, 2011.

[23] S. Goldwasser, Y. T. Kalai, and G. N. Rothblum. One-time programs. In *CRYPTO*, pages 39–56, 2008.

[24] J. A. Halderman, S. D. Schoen, N. Heninger, W. Clarkson, W. Paul, J. A. Calandrino, A. J. Feldman, J. Appelbaum, and E. W. Felten. Lest we remember: cold-boot attacks on encryption keys. *Commun. ACM*, 52(5):91–98, 2009.

[25] S. Halevi and H. Lin. After-the-fact leakage in public-key encryption. In *TCC*, pages 107–124, 2011.

[26] Y. Ishai, A. Sahai, and D. Wagner. Private circuits: Securing hardware against probing attacks. In *CRYPTO*, pages 463–481, 2003.

[27] P. C. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In *CRYPTO*, pages 104–113, 1996.

[28] P. C. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In *CRYPTO*, pages 388–397, 1999.

[29] A. B. Lewko, M. Lewko, and B. Waters. How to leak on key updates. In *STOC*, pages 725–734, 2011.

[30] A. B. Lewko, Y. Rouselakis, and B. Waters. Achieving leakage resilience through dual system encryption. In *TCC*, pages 70–88, 2011.

[31] S. Micali and L. Reyzin. Physically observable cryptography (extended abstract). In *TCC*, pages 278–296, 2004.

[32] M. Naor and G. Segev. Public-key cryptosystems resilient to key leakage. In *CRYPTO*, pages 18–35, 2009.

[33] C. Petit, F.-X. Standaert, O. Pereira, T. Malkin, and M. Yung. A block cipher based pseudo random number generator secure against side-channel key recovery. In *ASIACCS*, pages 56–65, 2008.

[34] K. Pietrzak. A leakage-resilient mode of operation. In *EUROCRYPT*, pages 462–482, 2009.

[35] J.-J. Quisquater and D. Samyde. Electromagnetic analysis (ema): Measures and counter-measures for smart cards. In *E-smart*, pages 200–210, 2001.

[36] R. L. Rivest. All-or-nothing encryption and the package transform. In *FSE*, pages 210–218, 1997.

[37] B. University. Reliable computing laboratory. Side channel attacks database. http://www.sidechannelattacks.com.