Leakage Resilient ElGamal Encryption

Eike Kiltz[1] and Krzysztof Pietrzak[2]

[1] Ruhr-Universität Bochum, Germany*
eike.kiltz@rub.de
[2] CWI, Amsterdam, The Netherlands
pietrzak@cwi.nl

Abstract. Blinding is a popular and well-known countermeasure to protect public-key cryptosystems against side-channel attacks. The high level idea is to randomize an exponentiation in order to prevent multiple measurements of the same operation on different data, as such measurements might allow the adversary to learn the secret exponent. Several variants of blinding have been proposed in the literature, using additive or multiplicative secret-sharing to blind either the base or the exponent. These countermeasures usually aim at preventing particular side-channel attacks (mostly power analysis) and come without any formal security guarantee.

In this work we investigate to which extend blinding can provide *provable security* against a *general class* of side-channel attacks. Surprisingly, it turns out that in the context of public-key encryption some blinding techniques are more suited than others. In particular, we consider a *multiplicatively blinded* version of ElGamal public-key encryption where

- we *prove* that the scheme, instantiated over bilinear groups of prime order p (where $p-1$ is not smooth) is leakage resilient in the generic-group model. Here we consider the model of *chosen-ciphertext security* in the presence of *continuous leakage*, i.e., the scheme remains chosen-ciphertext secure even if with *every* decryption query the adversary can learn a bounded amount (roughly $\log(p)/2$ bits) of arbitrary, adversarially chosen information about the computation.
- we *conjecture* that the scheme, instantiated over arbitrary groups of prime order p (where $p-1$ is not smooth) is leakage resilient.

Previous to this work no encryption scheme secure against continuous leakage was known. Constructing a scheme that can be *proven* secure in the *standard model* remains an interesting open problem.

1 Introduction

Side-channel attacks are cryptanalytic attacks against physical implementations of cryptosystems that exploit some kind of information leakage from the cryptodevice during execution. Traditional security notions (such as chosen-ciphertext security for encryption schemes) do not provide any security guarantee against such attacks, and many implementations of provably secure cryptosystems were

* Part of the work conducted while the author was at CWI, Amsterdam.

broken by side-channel attacks exploiting side-channels such as running-time [36], electromagnetic radiation [45, 26], power consumption [38], fault detection [8, 6] and many more (see, e.g., [46, 43]).

Countermeasures against side channel attacks can either be algorithmic, or on the hardware level. In the latter case, one generally tries to build hardware that leaks as few information as possible (e.g., by shielding electromagnetic radiation.) Algorithmic countermeasures means that one designs algorithms, such that their mere description already provides security against side channel attacks. (E.g., one can protect against timing attacks by making sure that the running time of the algorithm is independent of the secret.) Traditionally, such algorithmic countermeasures (such as masking or blinding, cf. [43] for a list of relevant papers) are mostly ad-hoc in the sense that they defend against some specific and known attacks.

Leakage Resilient Cryptography. Recently, formal models were proposed where one does not assume any particular side-channel against which to protect, but only requires that potential side-channels are in some sense "resource bounded." In the model of *leakage resilience* [23], one considers adversaries which, on each invocation of the cryptographic primitive, can learn a bounded amount of arbitrary information about the secret internal state that was accessed during invocation. Since the overall amount of leaked information is unbounded (and may be much larger than the size of the secret state), this model is also often referred to as *continuous leakage* (e.g., [15, 9]). As we will discuss below, this is in sharp contrast to the model of "memory leakage" (e.g., [2, 41, 4, 3, 16]) which has the inherent limitation that the amount of leaked information is a-priory bounded and therefore cannot exceed the size of the secret state.)

An *implementation* of a leakage resilient primitive will then be secure against every side-channel attack that fits our general model, i.e., as long as the *amount* of information that is leaked on each invocation is sufficiently bounded, and moreover the device adheres the "only computation leaks information" axiom from [40], which states that memory content that is not accessed during an invocation, does not leak. Security in this bounded leakage model hence means that the hardware implementation of the cryptosystem only has to be protected to fit the above model; once that is done, the proof provides security of the scheme. Using bounded leakage is inspired by the bounded retrieval model [13, 20, 19, 10, 22, 4] which in turn was inspired by the bounded-storage model [39, 21, 53, 30].

So far most theoretical research has focused on preventing memory leakage [13, 20, 19, 10, 22, 4] and the only known leakage resilient primitives (in our sense of security against continuous leakage) are stream-ciphers [23, 44], digital signatures [25] and — in a weaker "non-adaptive" model — pseudorandom functions and permutations [18]. Recently, general compilers have been proposed which turn *any* circuit into a leakage-resilient one [28, 33]. Currently, these general compilers are just a proof of concept and too inefficient to be used in practice, relying on fully homomorphic encryption [33] or requiring one full encryption per gate [28].

In this paper, we address the problem of leakage resilient public-key encryption (PKE). The standard security notion for PKE is indistinguishability under a chosen plaintext attack (IND-CPA) or the stronger notion of indistinguishability under a chosen ciphertext attack (IND-CCA).[3]

Modelling Leakage Resilience. Consider some cryptosystem CS, let S_0 denote its initial internal state and S_i its state after the ith invocation. On the ith invocation of CS, the adversary chooses some input X_i and gets Y_i where $(S_{i+1}, Y_i) \leftarrow \mathsf{CS}(S_i, X_i)$.

In the original definition of leakage resilience [23], the adversary gets the additional power to choose, besides the regular input X_i, some leakage function f_i whose range is bounded to some fixed $\lambda \in \mathbb{N}$ bits with every query. After the ith invocation she not only gets the regular output Y_i, but additionally the leakage $\Lambda_i \leftarrow f_i(S_i^+, R)$ where R is the randomness that CS used during its computation, and S_i^+ is the subset of the state S_i that was accessed (i.e., read and/or written) during computation. Note that to be leakage resilient, a primitive must be stateful (i.e. $S_i \neq S_{i-1}$), as otherwise one can just leak the state λ bits at a time.

In this paper we will use a more fine-grained notion of leakage resilience, where an invocation of CS (which will be a decryption query) is split in two phases, and those two phases leak individually. More precisely, the computation of a decryption can syntactically be split into two phases $\mathsf{Dec1}^*$ and $\mathsf{Dec2}^*$, which are executed in a sequential order to decrypt the message. As in a CCA attack, the adversary can make decryption queries with respect to a ciphertext C, and can furthermore specify two (efficiently computable) leakage functions, f and g, whose range is bounded by λ bits. (λ is the leakage parameter.) In addition to the decryption of C the adversary also obtains the output of f and g applied to all the inputs of $\mathsf{Dec1}^*$ and $\mathsf{Dec2}^*$, respectively, including the algorithm's internal random coin tosses.

On Bounded Range and Domain. Summing up, leakage resilience considers attackers who, with every invocation, can adaptively choose a leakage function f and then get the output of f applied to the internal secret state (if the system is probabilistic also all internal coin tosses) of the cryptosystem. The function f can be arbitrary, but is restricted in its input domain and range:

Bounded range: The range of f is $\{0,1\}^\lambda$ for some parameter $\lambda \in \mathbb{N}$.
Bounded domain: f gets as input only the secret state that is actually *accessed* during this invocation.

A mathematical model of side-channel leakage is only useful if it captures (and thus implies security against) leakage that occurs in practice. As f gets the

[3] In a CPA the adversary only gets the public-key and then has to distinguish the encryptions of two different messages. In a CCA [47] the adversary can also ask for decryptions of ciphertexts of her choice. We distinguish between CCA1 and the stronger CCA2 security, in the latter the adversary can make decryption queries also after she got the challenge ciphertext.

same input as the cryptosystem CS, it can simulate the computation of CS on any conceivable hardware (e.g., all the values carried by wires on a computing circuit), and thus also compute any kind of leakage that might occur. Though, the restriction on bounded range might not allow f to actually output the entire leakage, and the restriction on bounded domain might make it impossible to simulate leakage that depends on earlier invocations, we discuss this points below.

Bounded range. In practice, it seems hard to quantify how much information actual hardware (like a smart-card) actually leaks. In most side-channel attacks the adversary measures large amounts of data, e.g., an entire power-consumption curve. So at a glance this assumption might seem unreasonable, but this is a bit overly pessimistic.

Even though side-channel leakage may contain lots of data, only a *small fraction* can actually be exploited in each measurement. The model of leakage resilience allows only for the leakage of a small number λ of bits, but this leakage is "worst case" in the sense that the adversary may choose the leakage function which outputs the most useful information. Below we outline two ways in which this observation can be made precise. The first shows that side-channel attacks used in practice are captured by leakage resilience as they only exploit few bits of information from each actual measurement. The second is a relaxation of bounded leakage which can reasonably be assumed to be satisfied in practice.

Side-Channel Attacks Exploit Few Bits. Many side-channel attacks first measure large amounts of leakage $\Lambda_1, \Lambda_2, \ldots$ from every invocation, like a power consumption curve. Then, in a first step, each leakage Λ_i is preprocessed in order to extract some "useful" information Λ'_i (this Λ'_i could, e.g., be a list of the most likely sub-keys.) The attack then proceeds by trying to recover the secret key from $\Lambda'_1, \Lambda'_2, \ldots$. Such attacks are covered by leakage resilience whenever the amount of extracted data $|\Lambda'_i|$ is at most the amount of leakage λ allowed per invocation.

Relaxing Bounded Range. By inspecting the proofs of our constructions (as well as the ones from [23, 44, 25]), one sees that a restriction on the leakage functions is required which is considerably weaker than restricting the range to λ bits: it is only required that the leakage $f(S^+)$ does not decrease the HILL-pseudoentropy [31, 5][4] the adversary has about the active state S^+ by more than λ bits. Thus, although it may be unreasonable to assume that no more than λ bits leak per invocation of a smart-card, assuming that this leakage will only degrade the HILL-pseudoentropy by λ bits seems much more realistic in practice.

Bounded domain. The *bounded domain* restriction is a very mild restriction. Unlike for bounded range, it is non-trivial to even imagine a remotely realistic side-channel attack which would break a scheme by not adhering to it. This

[4] HILL-pseudoentropy is a computational analogue of min-entropy. As for min-entropy, λ bits of information cannot decrease it (in expectation) by more than λ bits.

restriction (on how leakage functions are mathematically modeled) is implied by the "only computation leaks information" axiom (which states something about physical properties of devices) of [40]. But it also covers other practical attacks which do not satisfy this axiom. For example note that an adversary can learn any linear function $f(S)$ of the *entire* state S (which is split in, say, two parts S_1, S_2 that are accessed individually) by specifying leakage functions f_1, f_2 such that $f_1(a) + f_2(b) = f(a, b)$ (the adversary can ask to learn $f_1(S_1)$ and $f_2(S_2)$ as S_1 and S_2 are accessed respectively, and then compute $f(S)$ locally.) This simple observation already shows that claims made in the literature arguing that the bounded range & domain restrictions do not cover attacks like "cold-boot attacks" [29] or static leakage (as claimed in [51]) are not well-founded.[5] As argued by Dziembowski,[6] this restriction not only covers all linear function $f(a, b) = f_1(a) + f_2(b)$, but in fact any function $f(a, b)$ which has a communication complexity of at most λ. A good candidate for an actual leakage function that does invalidate this assumption[7] is the inner product $f(a, b) = \sum_i a_i \cdot b_i \bmod 2$ which has linear communication complexity.

1.1 ElGamal Encryption

The ElGamal encryption scheme [24] over a cyclic group $\mathbb{G}$ of prime order p works as follows. The public key consists of a generator g of $\mathbb{G}$ and $X = g^x$, where $x \in \mathbb{Z}_p$ is the secret key. Encryption defines the ciphertext as $C = g^r$ and uses the symmetric key $K = X^r$ to blind the message. Decryption reconstructs the key by computing $K = C^x$. In its hybrid version, ElGamal encryption is contained in many standard bodies (e.g., [48, 32, 50]) and it is (using the name Elliptic Curve Integrated Encryption System, "ECIES") commonly considered to be the *standard method* to encrypt over elliptic curves. At this point it may be instructive to see why the ElGamal encryption scheme is not leakage resilient. An adversary, in the ith decryption query, can specify a leakage function that outputs the i-th bit of the secret key x. Therefore, after $q = |x|$ queries to the leakage oracle the entire secret key can be reconstructed. As we already pointed out, the inherent reason why the above attack works is that decryption is stateless.

Let's first look at a straight forward (but unsuccessful) attempt to make the ElGamal scheme leakage resilient. To this end we make decryption stateful and

[5] In the above argument we implicitly assumed that ultimately the entire secret state will be touched, although this seems obvious (after all, why would one save a secret state if it's not supposed to be ever read), the tokens used in the construction of one-time programs [27] are an example where exactly this happens. For such primitives obeying the "only computation leaks information" axiom in its original physical sense is necessary.

[6] at the workshop "Provable security against physical attacks", February 2010, Leiden.

[7] and thus might be used to construct an actual real world counterexample where the security of an implementation gets broken because the bounded domain restriction is invalidated.

split it into two parts $\mathsf{Dec1}^*$ and $\mathsf{Dec2}^*$. The secret key is *additively* shared into $x = \sigma_0 + \sigma'_0$ by setting $\sigma_0 = x - r_0$ and $\sigma'_0 = x + r_0$. Decryption works as follows. The first part $\mathsf{Dec1}^*$ computes $\sigma_i = \sigma_{i-1} + r_i \bmod p$, $K' = C^{\sigma_i}$ and passes K' as input to the second part. $\mathsf{Dec2}^*$ computes $\sigma'_i = \sigma'_{i-1} - r_i \bmod p$ and then $K = K' \cdot C^{\sigma'_i}$. Note that the state information is randomly re-shared subject to $\sigma_i + \sigma'_i = x$. However, this scheme is not leakage resilient since an attacker can adaptively learn certain bits of $\sigma_i = x + R_i$ and $\sigma'_i = x - R_i$ (where $R_i = \sum_{j=0}^{i} r_j$) that enable him to fully reconstruct the secret key x.[8]

1.2 Our results

Conjectured leakage resilient ElGamal encryption. We consider a practical randomization method to make the ElGamal PKE scheme (or one if its standardized hybrid variants) leakage resilient under chosen-ciphertext attacks in the above sense. In the context of leakage resilience this method (or variants thereof) were already proposed in [12, 37, 52]. The central idea is to use *multiplicative secret sharing* to share the secret key x, i.e., x is shared as $\sigma_i = xR_i^{-1} \bmod p$ and $\sigma'_i = R_i \bmod p$, for some random $R_i \in \mathbb{Z}_p^*$. More precisely, the first part of decryption computes $\sigma_i = \sigma_{i-1} r_i^{-1} \bmod p$ and $K' = C^{\sigma_i}$. The second part computes $\sigma'_i = \sigma'_{i-1} r_i \bmod p$ and then $K = K'^{\sigma'_i}$. Again note that the state information is randomly reshared subject to $\sigma_i \cdot \sigma'_i = x$. We remark that our method does not modify ElGamal's encryption algorithm, it only modifies the way ciphertexts are decrypted. In particular, public-keys and ciphertexts are the same as in ElGamal encryption and therefore our method offers an attractive way to update existing ElGamal-based systems with algorithmic security against side-channel attacks. Unfortunately, we are not able to prove that the above method is provable leakage resilient and therefore we can only state the scheme's security as a conjecture.

Provable leakage resilient ElGamal encryption. We also propose to apply multiplicative secret sharing to the ElGamal encryption scheme instantiated over *bilinear groups*. Our main theorem (Theorem 1) states that this scheme is leakage resilient against CCA1 attack in the generic group model. The key observation is that the secret key is a group element X and decryption performs a pairing operation with X as one fixed base. This allows us to multiplicatively share the secret key as a *group element*, i.e., $X = \sigma_i \cdot \sigma'_i \in \mathbb{G}$. Intuitively, we use the fact that in the generic group model some bits of the representation of σ_i and σ'_i essentially look random and therefore are useless to the leakage adversary. To formally prove this intuition, however, turns out to be surprisingly difficult.

We also mention that a proof in the generic group model has its obvious weaknesses. (See, e.g., [35].) In particular in connection with side channel attacks

[8] Since $x = \sigma_i + \sigma'_i \bmod p$, the first $t \approx \lambda$ least significant bits of x can be computed as $(\sigma_i \bmod 2^t) + (\sigma'_i \bmod 2^t) \bmod 2^t$, minus an additive factor $p \bmod 2^t$ in case there is an overflow mod$p$. (The latter can be checked from the high order bits of σ_i and σ'_i.) This process can be iterated to learn the entire secret key.

the generic group model may "abstract away" too much important information an adversary may obtain in a real implementation of the scheme. This should be taken into account when interpreting our formal security statement. However, our result seems to be the first PKE scheme that is provably leakage resilient. Furthermore, the scheme is very practical. Another possible interpretation of our result is that when protecting the exponentiation function against (a large class of) side-channel attacks, multiplicative secret sharing techniques seem more suitable than additive ones.

Leakage Resilient Exponentiation and Pairing Operation. Speaking more generally, our above mentioned methods how to secure ElGamal against side-channel attacks show that one can possibly make *discrete exponentiation* and *a pairing operation* leakage resilient. Let $\mathbb{G}$ be a group of prime order p and g be a generator of $\mathbb{G}$. In discrete exponentiation one wants to take public group elements Y_i to some fixed secret power x (which is only leaked through g^x). We propose to share x as $x = x' \cdot x'' \bmod p$ and compute the values $K_i = Y_i^x$ in two iterative steps as $K_i' = Y_i^{x'}$ followed by $K_i = (K_i')^{x''}$. After each such computation x' and x'' get randomly reshared subject to $x = x' \cdot x'' \bmod p$. In a pairing operation one is given public group elements Y_i and want to compute $e(Y_i, X)$, for some fixed secret group element X (which is only leaked though $e(g, X)$). Here $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ is a bilinear pairing. Again we propose to share X as $X = X' \cdot X'' \in \mathbb{G}$ and compute the values $K_i = e(Y_i, X)$ in three iterative steps as $K_i' = e(Y_i, X')$, $K_i'' = e(Y_i, X'')$, and $K_i = K_i' \cdot K_i'' \in \mathbb{G}_T$, followed by a resharing of $X = X' \cdot X'' \in \mathbb{G}$. Our main result (Theorem 1) shows that our method to perform a pairing operation is provable leakage resilient in the generic group model.

Difficulty to prove leakage resilience against CCA2 attacks. It is well known that the ElGamal encryption scheme, where the key K is hashed and the one-time pad is replaced with a chosen-ciphertext secure symmetric cipher, is secure against CCA2 attacks [1]. We remark that this scheme is not leakage resilient against CCA2 attack since an adversary can adaptively obtain some bits about the unhashed symmetric key of the challenge ciphertext. Indeed, building a PKE scheme that is (provably) leakage resilient against CCA2 attacks remains a challenging open problem.

1.3 Related Work

In the hardware community the usefulness of secret-sharing in the context of side-channel countermeasures is well known. In particular, secret-sharing has been proposed as a countermeasure against "differential power analysis attacks" for exponentiation algorithms in [11, 12, 12, 37, 52], but without any formal analysis.

Most works on side-channel countermeasures, including the ones just mentioned, consider countermeasures against particular side-channel attacks. Micali and Reyzin [40] in their work on "physically observable cryptography" proposed an influential theoretical framework to capture side-channel attacks on a more general level.

Besides leakage resilience, there are several other models that consider cryptosystems which remain secure even if a function $f(sk)$ (chosen by the adversary from a very broad class of functions) of the secret key sk is leaked. We shortly mention these models below. The main difference to leakage resilience is that those models consider stateless cryptosystems, and thus cannot tolerate any kind of "continuous" leakage (an exception is the very recent work on "continuous memory attacks.") On the other hand, the leakage function in those works gets the entire state as input, and not just the part of the state that was accessed.

Memory Attacks. Akavia et al. [2] introduce the model of "security against memory attacks," where one requires that the scheme remains secure even if a function $f(sk)$ of the secret sk is leaked *once*, where the only restriction on $f(\cdot)$ one makes is its bounded output length. (Clearly the bound must satisfy $|f(sk)| \ll |sk|$. This model is a restricted version of the BRM model discussed below.) [2, 41] construct public-key encryption schemes in this model, Katz and Vaikuntanathan [34] constructs digital signatures.

Bounded Retrieval Model. The bounded retrieval model (BRM) [13, 19, 20, 10, 22, 4] is a generalization of the previous model, where one requires that the secret key can be made huge, while the scheme still remains efficient. Such schemes can provide security against malware like viruses or Trojans, which temporarily take control over a computer, but do not have enough "bandwidth" to leak the entire artificially huge key. Most works on intrusion resilient crypto consider symmetric primitives, but after the first success in constructing public-key cryptosystems secure against memory attacks (mentioned above), Alwen et al. achieved public-key crypto also in the BRM model. In particular authentication and signature schemes [4] and public-key encryption [3].

Auxiliary Input. Dodis et al. construct symmetric [17] and public-key [14] encryption schemes in a model where the range of $f(\cdot)$ may be unbounded, but one only requires that it is hard to recover sk from $f(sk)$. (i.e. any polynomial time adversary should output sk with exponentially small probability.)

Continuous Memory Attacks. Very recently, Dodis, Haralambiev, Lopez-Alt, and Wichs [15] and Brakerski, Kalai, Katz and Vaikuntanathan [9] introduce the model of "continuous memory attacks." This model generalizes the notion of memory attacks. Also here the adversary can learn a bounded amount, λ bits say, of leakage about the (entire) secret key. But now there's an additional "refresh" procedure which takes the secret key sk and outputs a new secret key sk'. The adversary can learn λ bits (where λ is $c|sk|$ for some constant $c > 0$) in-between any two refresh phases, but the refreshing itself has to be completely leak-free [15] or leak at most a logarithmic number of bits [9]. Remarkably, in this model [15] construct authentication and signature schemes, [9] obtain get public-key encryption. Both papers work in the standard model, the underlying assumption in both papers is the linear assumption over bilinear groups. The models of leakage resilience and continuous memory attacks are incomparable: leakage resilience assumes "only computation leaks" whereas continuous memory attacks need an (almost) leak-free refresh phase. As mentioned, the constructions

[15, 9] are proven secure in the standard model, whereas we use a strong idealized model. On the positive side, our scheme is very efficient (only about two times slower than standard ElGamal) whereas, e.g., [9] needs a constant number of pairings to encrypt a single bit.

2 Definitions

If A is a deterministic algorithm we write $y \leftarrow \mathsf{A}(x)$ to denote that A outputs y on input x. If A is randomized we write $y \xleftarrow{*} \mathsf{A}(x)$ or, $y \xleftarrow{r} \mathsf{A}(x)$ if we want to make the randomness r used by the algorithm explicit (for future reference).

Key Encapsulation Mechanisms. A key encapsulation mechanism (KEM) is defined similarly to a public-key encryption scheme, except that the encryption algorithm (called encapsulation) does not take any input, but rather outputs the encryption of a random key K, which then can be used with as a key in any symmetric encryption scheme to encrypt the actual message.

Formally, a key-encapsulation mechanism KEM consists of three algorithms $\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec}$. $\mathsf{KG} : \{0,1\}^* \to \mathcal{PK} \times \mathcal{SK}$ is the probabilistic key-generation algorithm, which on input a security parameter κ outputs a public/secret-key pair. The probabilistic encapsulation algorithm $\mathsf{Enc} : \mathcal{PK} \to \mathcal{K} \times \mathcal{C}$ and decapsulation algorithm $\mathsf{Dec} : \mathcal{SK} \times \mathcal{C} \to \mathcal{K} \cup \perp$ satisfy the following correctness property for all κ

$$\Pr[K = K' \mid (pk, sk) \xleftarrow{*} \mathsf{KG}(\kappa);\ (C, K) \xleftarrow{*} \mathsf{Enc}(pk);\ K' \leftarrow \mathsf{Dec}(sk, C)] = 1$$

The CCA1 security (aka. security against lunchtime attacks) of a key-encapsulation mechanism KEM is defined by the following experiment.

Experiment $\mathbf{Exp}^{\mathrm{cca1}}_{\mathsf{KEM}}(\mathcal{F}, \kappa)$
$(pk, sk) \xleftarrow{*} \mathsf{KG}(\kappa)$
$w \xleftarrow{*} \mathcal{F}^{\mathcal{O}_{sk}(\cdot)}(pk)$
$b \xleftarrow{*} \{0,1\}$
$(C^*, K_0) \xleftarrow{*} \mathsf{Enc}(pk)$
$K_1 \xleftarrow{*} \mathcal{K}$
$b' \xleftarrow{*} \mathcal{F}(w, C^*, K_b)$

Oracle $\mathcal{O}^{\mathrm{cca1}}_{sk}(C)$
$K \leftarrow \mathsf{Dec}(sk, C)$
Return K

Let μ denote the probability that $b = b'$ in the above experiment, then we define the advantage of $\mathcal{F}$ as $\mathbf{Adv}^{\mathrm{cca1}}_{\mathsf{KEM}}(\mathcal{F}, \kappa) = 2|1/2 - \mu|$. In CCA2 security, the adversary is additionally allowed to query the decryption oracle in its second (guess) stage.

Stateful key encapsulation and leakage resilience. To formally define our notion of leakage resilience we consider stateful key encapsulation mechanisms $\mathsf{KEM}^* = (\mathsf{KG}^*, \mathsf{Enc}^*, \mathsf{Dec1}^*, \mathsf{Dec2}^*)$ in which decapsulation is stateful and can formally split into two sequential stages $\mathsf{Dec} = (\mathsf{Dec1}^*, \mathsf{Dec2}^*)$. The input/output behavior will stay exactly the same as in a standard KEM.

More formally, the key generation algorithm $\mathsf{KG}^*(\kappa)$ generates a public key and and two initial states, σ_0 and σ'_0. Intuitively, the states shares the secret key of the scheme and will be used by the stateful decapsulation algorithms $\mathsf{Dec1}^*, \mathsf{Dec2}^*$.

On the ith invocation of decapsulation, the decapsulated key K_i is computed as follows

$$(\sigma_i, w_i) \xleftarrow{r_i} \mathsf{Dec1}^*(\sigma_{i-1}, C_i) \ ; \ (\sigma'_i, K_i) \xleftarrow{r'_i} \mathsf{Dec2}^*(\sigma'_{i-1}, w_i) \tag{1}$$

Here r_i and r'_i is the explicit randomness of the two randomized algorithms, σ_i and σ'_i are the updated states and w_i is some state information that is passed from $\mathsf{Dec1}^*$ to $\mathsf{Dec2}^*$.

We now define leakage resilience. Let $\lambda \in \mathbb{N}$ be some leakage parameter. We will consider attacks, where the adversary can not only query its oracle for the decapsulated values $K_i = \mathsf{Dec}(sk, C_i)$, but additionally gets leakage from the computation of those values. That is, in the security experiment the adversary can, additionally to the input C_i, specify two efficiently computable leakage functions f_i, g_i with bounded range $\{0,1\}^\lambda$, and additionally to the regular output K_i also gets Λ_i, Λ'_i computed as

$$\Lambda_i = f_i(\sigma_{i-1}, r_i) \ ; \ \Lambda'_i = g_i(\sigma'_{i-1}, w_i, r'_i) \ ,$$

where the notation is as in (1). So the functions f_i, g_i get as input exactly the same data as $\mathsf{Dec1}^*/\mathsf{Dec2}^*$.[9] We define the CCLA1 (chosen ciphertext with leakage attack) security of KEM by the experiment below. (Note that now we not only have to specify the security parameter k, but also a leakage bound λ.)

Experiment $\mathbf{Exp}^{\mathrm{ccla}}_{\mathsf{KEM}}(\mathcal{F}, \kappa, \lambda)$	**Oracle** $\mathcal{O}^{\mathrm{ccla1}}(C, f, g)$
$(pk, \sigma_0, \sigma'_0) \xleftarrow{*} \mathsf{KG}(\kappa)$	If range of f or g is $\neq \{0,1\}^\lambda$ return $\perp$
$w \xleftarrow{*} \mathcal{F}^{\mathcal{O}^{\mathrm{ccla1}}(\cdot)}(pk)$	$i \leftarrow i + 1$
$b \xleftarrow{*} \{0,1\}$	$(\sigma_i, w_i) \xleftarrow{r_i} \mathsf{Dec1}^*(\sigma_{i-1}, C)$
$(C^*, K_0) \xleftarrow{*} \mathsf{Enc}(pk)$	$(\sigma'_i, K_i) \xleftarrow{r'_i} \mathsf{Dec2}^*(\sigma'_{i-1}, w_i)$
$K_1 \xleftarrow{*} \mathcal{K}$	$\Lambda_i \leftarrow f_i(\sigma_{i-1}, r_i)$
$i \leftarrow 0$	$\Lambda'_i \leftarrow g_i(\sigma'_{i-1}, w_i, r'_i)$
$b' \xleftarrow{*} \mathcal{F}(w, C^*, K_b)$	Return $(K_i, \Lambda_i, \Lambda'_i)$

Let μ denote the probability that $b = b'$ in the above experiment, then we define the advantage of $\mathcal{F}$ as $\mathbf{Adv}^{\mathrm{ccla}}_{\mathsf{KEM}}(\mathcal{F}, \kappa, \lambda) = 2|1/2 - \mu|$.

It is well-known that a CCA1 secure KEM plus a one-time secure symmetric cipher (such as a one-time pad) yields a CCA1-secure PKE scheme. For trivial reasons the same statement is also true for CCLA1 secure KEMs so for our purpose it is sufficient to build a CCLA1 secure KEM. On the other hand we remark that the respective composition theorem is wrong in general for CCLA2

[9] Note that C_i need not be explicitly given to f_i as the adversary chooses f_i and C_i together, and thus can "hard-code" C_i into f_i.

secure KEMs. That is, a CCLA2 secure KEM and a CCA secure DEM will in general not yield a CCLA2 secure PKE scheme.[10]

Bilinear Groups We assume the existence of a bilinear group generator BGen which is a randomized algorithm that outputs a bilinear group $\mathbb{PG} = (\mathbb{G}, \mathbb{G}_T, g, e, p)$ such that the following properties hold.

1. $\mathbb{G}$ and $\mathbb{G}_T$ are (multiplicative) cyclic groups of prime order p.
2. g is a generator of $\mathbb{G}$.
3. e is a bilinear map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ which is
   (a) bilinear: for all $u, v \in \mathbb{G}$ and $a, b \in \mathbb{Z}, e(u^a, v^b) = e(u, v)^{ab}$.
   (b) non-degenerate: $e(g, g) \neq 1$.

We say that $\mathbb{G}$ is a bilinear group if there exists a group $\mathbb{G}_T$ and a bilinear map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ as above, where e and the group action in $\mathbb{G}$ and $\mathbb{G}_T$ can be computed efficiently. We will use $\circ$ and $\star$ for the group operation in $\mathbb{G}$ and $\mathbb{G}_T$ respectively.

Generic Bilinear Groups. In the generic group model [42, 49] one encodes the group elements by unique, randomly chosen strings. This enforces that the only property which can be tested by an adversary is equality.

In the generic bilinear model (GBG) [7] the encoding is given by randomly chosen injective functions $\xi : \mathbb{Z}_p \to \Xi$ and $\xi_T : \mathbb{Z}_p \to \Xi_T$ which give the representations of the elements in the base and target group respectively (w.l.o.g. we will assume that $\Xi \cap \Xi_T = \emptyset$). The group operation and the bilinear map are performed by three public oracles $\mathcal{O}, \mathcal{O}_T, \mathcal{O}_e$, where for any $a, b \in \mathbb{Z}_p$

- $\mathcal{O}(\xi(a), \xi(b)) \to \xi(a + b \bmod p)$ (group operation on base group).
- $\mathcal{O}_T(\xi_T(a), \xi_T(b)) \to \xi_T(a + b \bmod p)$ (group operation on target group).
- $\mathcal{O}_e(\xi(a), \xi(b)) \to \xi_T(a \cdot b \bmod p)$ (bilinear map).

All oracles output $\perp$ when queried on an input outside of their domain. For a fixed generator g of $\mathbb{G}$ and $g_T \stackrel{\mathsf{def}}{=} e(g, g)$, one can think of $\xi(a)$ as an encoding of g^a, $\xi_T(a)$ as an encoding of g_T^a and $\xi_e(a, b)$ as an encoding of $g_T^{a \cdot b} = e(g^a, g^b)$. Of course one also must provide some means of computing the group representation $\xi(a)$ or $\xi_T(a)$ for any $a \in \mathbb{Z}_p$, say by providing oracles to do so. We can get away without additional oracles, by providing $\xi(1)$ and observing that then $\xi(a)$ can be computed making $\leq 2 \log p$ queries to $\mathcal{O}$ (by square and multiply). $\xi_T(1)$ (and thus any $\xi_T(a)$) can be computed by $\xi_T(1) \leftarrow \mathcal{O}_e(\xi(1), \xi(1))$.

3 Leakage Resilient ElGamal Encryption

In this section we present a general method to secure ElGamal encryption against leakage attacks. First, we present a modification of the standard ElGamal cryptosystem over any cyclic group of prime order. Unfortunately, we are not able

[10] An attacker may make a number of decryption queries only modifying the symmetric part of the challenge ciphertext. The decryption algorithm (internally) uses the challenge symmetric key that can be learned (bit-by-bit) through the leakage function.

to formally prove the leakage resilience of this scheme so we state its security as a conjecture. Next, we move to the ElGamal scheme over Bilinear Groups. Here we are able to prove that our method leads to a leakage resilient public-key encryption scheme (in the sense of CCLA1) in the generic group model.

3.1 ElGamal Key Encapsulation

Let Gen be a randomized algorithm that outputs a cyclic group $\mathbb{G}$ of order p where p is a strong prime. The ElGamal key-encapsulation mechanism $\mathsf{EG} = (\mathsf{KG}_{\mathsf{EG}}, \mathsf{Enc}_{\mathsf{EG}}, \mathsf{Dec}_{\mathsf{EG}})$ is defined as follows.

- $\mathsf{KG}_{\mathsf{EG}}(\kappa)$: Compute $(\mathbb{G}, p) \xleftarrow{*} \mathsf{Gen}(\kappa)$ and choose random $g \xleftarrow{*} \mathbb{G}$ and random $x \xleftarrow{*} \mathbb{Z}_p$. Set $X = g^x$. The public key is $pk = (\mathbb{G}, p, X)$ and the secret key is $sk = x$.
- $\mathsf{Enc}_{\mathsf{EG}}(pk)$: choose random $r \xleftarrow{*} \mathbb{Z}_p$. Set $C \leftarrow g^r \in \mathbb{G}$ and $K \leftarrow X^r \in \mathbb{G}$. The ciphertext is C and the key is K.
- $\mathsf{Dec}_{\mathsf{EG}}(sk, C)$: Compute the key as $K = C^x \in \mathbb{G}$.

As mentioned in the introduction, EG (or any other stateless scheme) cannot be leakage resilient since in the CCLA1 experiment an adversary can simply obtain the successive bits of the secret key x.

We will now describe a leakage resilient stateful key encapsulation mechanism $\mathsf{EG}^* = (\mathsf{KG}^*_{\mathsf{EG}}, \mathsf{Enc}^*_{\mathsf{EG}}, \mathsf{Dec1}^*_{\mathsf{EG}}, \mathsf{Dec2}^*_{\mathsf{EG}})$, which is derived from EG. As described in Section 2, the decapsulation algorithm is stateful and split in two parts.

- $\mathsf{KG}^*_{\mathsf{EG}}(\kappa)$: Run $(sk, pk) \xleftarrow{*} \mathsf{KG}_{\mathsf{EG}}(\kappa)$. (Recall that $sk = x$ and $pk = (\mathbb{G}, p, X = g^x)$.) Choose random $\sigma_0 \xleftarrow{*} \mathbb{Z}^*_p$ and set $\sigma'_0 = x\sigma_0^{-1} \bmod p$. The public key is pk and the two secret states are σ_0 and σ'_0.
- $\mathsf{Enc}^*_{\mathsf{EG}}(pk)$: the same as $\mathsf{Enc}_{\mathsf{EG}}(pk)$.
- $\mathsf{Dec1}^*_{\mathsf{EG}}(\sigma_{i-1}, C)$: choose random $r_i \xleftarrow{*} \mathbb{Z}^*_p$, set $\sigma_i = \sigma_{i-1} r_i^{-1} \bmod p$, $K' = C^{\sigma_i}$ and return (r_i, K').
- $\mathsf{Dec2}^*_{\mathsf{EG}}(\sigma'_{i-1}, (r_i, K'))$: set $\sigma'_i = \sigma'_{i-1} r_i^{-1} \bmod p$, and $K = K'^{\sigma'_i}$. The symmetric key is K and the updated state information is σ_i and σ'_i.

We cannot formally prove CCLA1 security of the scheme so we have to resort to the following conjecture.

Conjecture 1. EG^* is CCLA1 secure if $p-1$ has a large prime factor (say, $p-1 = 2p'$ for a prime p').[11]

One can furthermore make ElGamal key-encapsulation CCA2 secure (without leakage) under the strong Diffie-Hellman assumption in the random oracle model

[11] The reason we require p to be not smooth is to prevent the leakage functions to possibly compute discrete logarithms in $\mathbb{Z}_{p-1}$, as otherwise the multiplicative sharing σ, σ' (where $\sigma \cdot \sigma' = x$) can be efficiently turned into an additive sharing (of the discrete log of the secret key) $\sigma = h^{\Sigma}$,$\sigma' = h^{\Sigma'}$ where $x = h^X$ and $X = \Sigma + \Sigma'$. As described in Section 1.1, an additive sharing cannot give a leakage resilient scheme. The above also hints the inherent difficulty of proving this conjecture. Let us mention that already in [37] it is suggested to use a prime p where (p-1)/2 is prime in a very similar context. Our result can be seen as a formal justification for this choice.

by hashing the symmetric key symmetric key K [1]. This Hashed ElGamal scheme is contained in many standard bodies, e.g. [48, 32, 50]. Hashing the symmetric key clearly does not affect its CCLA1 security and therefore Hashed ElGamal is CCLA1 and CCA2 secure.

However, as we will explain now, in our leakage resilience setting hashing K will not make the scheme CCLA2 secure. The (unhashed) EG scheme is not CCA2 secure since it is malleable. (An adversary, given the challenge ciphertext C (enciphering a key K), can ask for a decryption of $C^2 \neq C$ to obtain K^2 from which it can reconstruct K.) Without considering leakage, hashing the key prevents this attack as now the adversary only sees a hashed key $\mathcal{H}(K^2)$. Unfortunately, in the leakage setting hashing will not help at all because the adversary can specify a leakage function which outputs λ bits of the unhashed key K^2. By asking for the decryption of the same ciphertext C^2 several times, leaking λ different bits of K^2 on each invocation, will ultimately reveal the entire K^2.

3.2 Bilinear ElGamal Key Encapsulation

The Bilinear ElGamal key-encapsulation mechanism
$\mathsf{BEG} = (\mathsf{KG}_{\mathsf{BEG}}, \mathsf{Enc}_{\mathsf{BEG}}, \mathsf{Dec}_{\mathsf{BEG}})$ is defined as follows.

- $\mathsf{KG}_{\mathsf{BEG}}(\kappa)$: Compute $\mathbb{PG} = (\mathbb{G}, \mathbb{G}_T, p, e) \xleftarrow{*} \mathsf{BGen}(\kappa)$ and choose random $g \xleftarrow{*} \mathbb{G}$ and random $x \xleftarrow{*} \mathbb{Z}_p$. Set $X = g^x$ and $X_T = e(g,g)^x$. The public key is $pk = (\mathbb{PG}, g, X_T)$ and the secret key is $sk = X$.
- $\mathsf{Enc}_{\mathsf{BEG}}(pk)$: choose random $r \xleftarrow{*} \mathbb{Z}_p$. Set $C \leftarrow g^r \in \mathbb{G}$ and $K \leftarrow X_T^r \in \mathbb{G}_T$. The ciphertext is C and the key is K.
- $\mathsf{Dec}_{\mathsf{BEG}}(sk, C)$: Compute the key as $K = e(C, X) \in \mathbb{G}_T$.

Note that correctness follows from the bilinear property $X_T^r = e(g,g)^{xr} = e(g^r, g^x) = e(g^r, X)$.

We will now describe a leakage resilient key encapsulation
$\mathsf{BEG}^* = (\mathsf{KG}^*_{\mathsf{BEG}}, \mathsf{Enc}^*_{\mathsf{BEG}}, \mathsf{Dec1}^*_{\mathsf{BEG}}, \mathsf{Dec2}^*_{\mathsf{BEG}})$, which is derived from BEG.

- $\mathsf{KG}^*_{\mathsf{BEG}}(\kappa)$: Run $(sk, pk) \xleftarrow{*} \mathsf{KG}_{\mathsf{BEG}}(\kappa)$. (Recall that $sk = X = g^x$ and $pk = (\mathbb{PG}, g, X_T = e(g,g)^x$.) Choose random $r_0 \xleftarrow{*} \mathbb{Z}_p^*$ and set $\sigma_0 \leftarrow g^{r_0}, \sigma_0' \leftarrow g^{x - r_0}$. The public key is pk and the secret states are σ_0, σ_0'.
- $\mathsf{Enc}^*_{\mathsf{BEG}}(pk)$: the same as $\mathsf{Enc}_{\mathsf{BEG}}(pk)$.
- $\mathsf{Dec1}^*_{\mathsf{BEG}}(\sigma_{i-1}, C)$: choose random $r_i \xleftarrow{*} \mathbb{Z}_p$, set $\sigma_i \leftarrow \sigma_{i-1} \circ g^{r_i}$, $K' \leftarrow e(\sigma_i, C)$ and return (r_i, K').
- $\mathsf{Dec2}^*_{\mathsf{BEG}}(\sigma'_{i-1}, (r_i, K'))$: set $\sigma'_i \leftarrow \sigma'_{i-1} \circ g^{-r_i}$ and $K'' \leftarrow e(\sigma'_i, C)$. The symmetric key is $K \leftarrow K' \star K'' \in \mathbb{G}_T$.

Note that for every $i, R_i \stackrel{\mathsf{def}}{=} \sum_{j=0}^{i} r_j$, we have $\sigma_i \circ \sigma'_i = g^{R_i} \circ g^{x - R_i} = g^x$, so the σ_i, σ'_i are a secret sharing of the secret key.

Theorem 1. *In the bilinear generic group model the scheme* BEG^* *is CCLA1 secure: the advantage of a q-query adversary who gets* λ *bits of leakage per invocation of* $\mathsf{Dec1}^*_{\mathsf{BEG}}$ *and* $\mathsf{Dec2}^*_{\mathsf{BEG}}$*, respectively, is at most* $\frac{2^{2\lambda+1} \cdot q^3}{p}$.

Thus, for a statistical security parameter n, we can tolerate $\lambda = \log(p)/2 - 3\log(q) - n/2$ bits of leakage. For space reasons, here we only can give a proof outline. Lemma 2 which we will state and prove later in Appendix A gives a quantitative statement of this theorem. This lemma roughly states that the scheme is secure as long as the leakage is sufficiently smaller than $\log(p)/2$. We now give an outline of the proof. The full proof is postponed to Appendix A.

Proof Outline. For technical reasons, we will consider a setting where the generic bilinear group is extended with an additional oracle $\mathcal{O}_{DL} : \Xi \cup \Xi_T \to \mathbb{Z}_p \cup \perp$, we will call this the *extended* generic bilinear group model. Intuitively, $\mathcal{O}_{DL}$ is an oracle for the discrete log problem, but only works on inputs that have not yet appeared since the oracles $\mathcal{O}, \mathcal{O}_e, \mathcal{O}_T$ have been initialized.

The proof outline is as follows. We will first show that the discrete logarithm problem (DL) is hard in the (base group of the) *extended* GBG model. We then give a reduction which shows how any adversary that can break the CCA1 security (without leakage) of BEG^* in the (normal) GBG model, can solve the discrete log problem in the extended GBG model. Next, we extend this proof to get our main result, namely a reduction of the CCLA1 security of BEG^* to the discrete log problem.

CCA1 security of BEG^*. Let $\mathcal{F}$ be an adversary that can break the CCA1 security of BEG^*. We construct an adversary $\mathcal{G}$ for DL (using $\mathcal{F}$ as a black-box) by letting $\mathcal{G}$ simulate the $\mathbf{Exp}^{\mathrm{cca1}}_{\mathsf{BEG}^*}(\mathcal{F}, p)$ experiment, where in this experiment $\mathcal{G}$ uses its DL challenge $\xi(y)$ as either the secret key $\xi(x)$ or the challenge encapsulated key $\xi(s)$ with probability $1/2$ respectively.

During the CCA1 experiment, $\mathcal{F}$ (which initially gets $\xi_T(x)$, and after the last decapsulation query gets $\xi(s)$) will learn the representation of elements $\xi_T(e_1), \xi_T(e_2), \ldots$ from the target group. One can show that just from observing $\mathcal{F}$'s oracle queries, $\mathcal{G}$ can assign to each e_i an equation $e_i = a_i + b_i \cdot x + c_i \cdot s + d_i \cdot s^2$ where it knows the coefficients $a_i, b_i, c_i, d_i \in \mathbb{Z}_p$. Similarly, for representations $\xi(e_1), \xi(e_2), \ldots$ of elements in the base group that $\mathcal{F}$ learns, $\mathcal{G}$ can extract a_i, b_i such that $e_i = a_i + b_i \cdot s$. To get an idea why this is the case, consider, e.g., the case where $\mathcal{F}$ makes a query $\xi_T(v \cdot w) \leftarrow \mathcal{O}_e(\xi(v), \xi(w))$. If $\xi(v)$ (same for $\xi(w)$) was never seen before, $\mathcal{G}$ first calls $\mathcal{O}_{DL}(\xi(v))$ to learn v. (Recall that $\mathcal{G}$ is in the extended GBG model.) Now $\mathcal{G}$ knows a, b, a', b' s.t. $v = a + b \cdot s$ and $w = a' + b' \cdot s$, which implies $v \cdot w = a'' + b'' \cdot x + c'' \cdot s + d'' \cdot s^2$ with $a'' = a + a', b'' = 0, c'' = a \cdot b' + a' \cdot b, d'' = b \cdot b'$.

Recall that $\mathcal{F}$'s goal is to distinguish the decapsulated key $\xi_T(x \cdot s)$ from a random element. If $\mathcal{F}$ has advantage ϵ in doing so, it actually must compute the element $\xi_T(x \cdot s)$ with probability ϵ. Which means we learn a, b, c, d such that

$$x \cdot s = a + b \cdot x + c \cdot s + d \cdot s^2, \tag{2}$$

this can be solved for s or x (or both). Thus $\mathcal{G}$ will learn the discrete log of $\xi(y)$ with probability at least $\varepsilon/2$ (as we initially randomly set $\xi(y) = s$ or $\xi(y) = x$).

CCLA1 security of BEG^*. The proof in the case of leakage attacks is more delicate. In a CCLA1 attack, with the ith decapsulation query, the adversary $\mathcal{F}$

also learns the output of the leakage functions f_i, g_i. If we had no upper bound on the output length of those functions, then f_i and g_i could just leak $\xi(R_i)$ and $\xi(x - R_i)$ respectively, from which $\mathcal{F}$ then could first compute the secret key $\xi(x)$ and then $\xi_T(x \cdot s)$. In this case, the reduction $\mathcal{G}$ does not learn an equation of the form eq.(2), but only the trivial equality $x \cdot s = x \cdot s$. We will prove that if the leakage bound $\lambda \ll \log p/2$, then the leakage functions will not leak any representation of an element to $\mathcal{F}$ that $\mathcal{F}$ could not efficiently compute itself.

To see this, let us first make the simplifying assumption that the leakage functions f_i, g_i are not given access to the group oracles $\mathcal{O}, \mathcal{O}_e, \mathcal{O}_T$. Then all the leakage functions can try to do, is to leak some element they get as input. Consider any such element, say $\xi(R_i)$. As $\xi(R_i)$ is only given as input to f_{i-1} and f_i, at most 2λ bits about this element can leak. If $2\lambda \ll \log p$, then $\mathcal{F}$ will have high min-entropy about $\xi(R_i)$ even given this 2λ bits of leakage. Thus it is very unlikely that it can guess $\xi(R_i)$.

Now consider the general case, where the leakage functions can use the group oracles. Now the leakage functions can trivially leak the representation of some group element, say $f_1, f_2, \ldots$ all use $\mathcal{O}$ to compute $\xi(z)$ for some fixed z and each leaks λ bit of $\xi(z)$ until $\mathcal{F}$ learns the entire $\xi(a)$. Now $\mathcal{F}$ does get the representation of an element $\xi(a)$ without receiving it from the group oracles, but that is no problem, as $\mathcal{G}$ will know an a, b such that $a + b \cdot s = z$ (namely $a = z$ and $b = 0$), and that's all we care about.

Now the f_i leakage function (similarly for g_i) can use their input $\xi(R_{i-1})$ to compute elements $\xi(z)$ where $\mathcal{G}$ only knows a, b (where $b \neq 0$) such that $z = a + b \cdot r_0$. We call such a representation "bound" (as opposed to "free" representations $\xi(z)$ where $\mathcal{G}$ trivially learns z by just observing f_i's oracle queries). It would be a problem if a bound representation could leak to $\mathcal{F}$. As said before, the f_i's can trivially leak 2λ bits about a bound element, as, e.g., f_{i-1} and f_i have access to $\xi(R_i)$ (recall that $R_i = \sum_{j=0}^{i} r_j$ where each r_j is uniformly random). But it is not clear how any other leakage function f_j ($j \notin \{i-1, i\}$) would compute the element $\xi(R_i)$ or any other element derived from it; since the sharings are randomized during each invocation, the values $\xi(R_{j-1}), r_j$ that f_j has are completely independent of R_i (and thus $\xi(R_i)$). In fact, we show that if $\mathcal{F}$ manages to choose leakage functions such that the same bound element is computed by f_i and f_j (where $j > i+1$) with probability ϵ, then $\mathcal{F}$ can be used to solve the discrete logarithm problem with probability $\epsilon/2^{2\lambda}q$. The idea is to use the discrete logarithm challenge $\xi(y)$ as $\xi(r_j)$ for a random j. Note that to simulate the experiment, $\mathcal{G}$ only needs $\xi(r_j)$ not r_j, except to compute the 2λ bits of leakage from the jth decapsulation query. (As here the leakage functions f_j, g_j expect r_j as input.) We let $\mathcal{G}$ randomly guess this leakage, which will be correct with probability $2^{-2\lambda}$. Now assume we have two identical bound elements $\xi(z)$ computed by $f_{i'}$ and $f_{i''}$ where $i'' > i' + 1$. As this query was made by $f_{i'}$, and up to this point $\mathcal{G}$ only used $r_0, \ldots, r_{i'}$ that it sampled himself, he will know z. As this query was also made by i'', $\mathcal{G}$ learns $a, b \neq 0$ such that $z = a + b \cdot r_j$, and thus can solve this equality to get r_j.

References

1. Michel Abdalla, Mihir Bellare, and Phillip Rogaway. The oracle Diffie-Hellman assumptions and an analysis of DHIES. In *CT-RSA 2001*.
2. Adi Akavia, Shafi Goldwasser, and Vinod Vaikuntanathan. Simultaneous hardcore bits and cryptography against memory attacks. In *TCC 2009*.
3. Joel Alwen, Yevgeniy Dodis, Moni Naor, Gil Segev, Shabsi Walfish, and Daniel Wichs. Public-key encryption in the bounded-retrieval model. In *EUROCRYPT 2010*, 2010.
4. Joël Alwen, Yevgeniy Dodis, and Daniel Wichs. Leakage-resilient public-key cryptography in the bounded-retrieval model. In *CRYPTO2009*, 2009.
5. Boaz Barak, Ronen Shaltiel, and Avi Wigderson. Computational analogues of entropy. In *RANDOM-APPROX*, pages 200–215, 2003.
6. Eli Biham and Adi Shamir. Differential fault analysis of secret key cryptosystems. In Burton S. Kaliski Jr., editor, *CRYPTO'97*, volume 1294 of *LNCS*, pages 513–525. Springer-Verlag, Berlin, Germany, August 1997.
7. Dan Boneh, Xavier Boyen, and Eu-Jin Goh. Hierarchical identity based encryption with constant size ciphertext. In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 440–456. Springer-Verlag, Berlin, Germany, May 2005.
8. Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. On the importance of checking cryptographic protocols for faults (extended abstract). In Walter Fumy, editor, *EUROCRYPT'97*, volume 1233 of *LNCS*, pages 37–51. Springer-Verlag, Berlin, Germany, May 1997.
9. Zvika Brakerski, Yael Tauman Kalai, Jonathan Katz, and Vinod Vaikuntanathan. Overcoming the hole in the bucket: Public-key cryptography resilient to continual memory leakage. In *51st FOCS*. IEEE Computer Society Press, 2010.
10. David Cash, Yan Zong Ding, Yevgeniy Dodis, Wenke Lee, Richard J. Lipton, and Shabsi Walfish. Intrusion-resilient key exchange in the bounded retrieval model. In Salil P. Vadhan, editor, *TCC 2007*, volume 4392 of *LNCS*, pages 479–498. Springer-Verlag, Berlin, Germany, February 2007.
11. Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards sound approaches to counteract power-analysis attacks. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 398–412. Springer-Verlag, Berlin, Germany, August 1999.
12. Christophe Clavier and Marc Joye. Universal exponentiation algorithm. In Çetin Kaya Koç, David Naccache, and Christof Paar, editors, *CHES 2001*, volume 2162 of *LNCS*, pages 300–308. Springer-Verlag, Berlin, Germany, May 2001.
13. Giovanni Di Crescenzo, Richard J. Lipton, and Shabsi Walfish. Perfectly secure password protocols in the bounded retrieval model. In Shai Halevi and Tal Rabin, editors, *TCC 2006*, volume 3876 of *LNCS*, pages 225–244. Springer-Verlag, Berlin, Germany, March 2006.
14. Yevgeniy Dodis, Shafi Goldwasser, Yael Tauman Kalai, Chris Peikert, and Vinod Vaikuntanathan. Public-key encryption schemes with auxiliary inputs. In *TCC 2010*, LNCS, pages 361–381. Springer-Verlag, Berlin, Germany, 2010.
15. Yevgeniy Dodis, Kristiyan Haralambiev, Adriana Lopez-Alt, and Daniel Wichs. Cryptography against continuous memory attacks. In *51st FOCS*. IEEE Computer Society Press, 2010.
16. Yevgeniy Dodis, Kristiyan Haralambiev, Adriana Lopez-Alt, and Daniel Wichs. Efficient public-key cryptography in the presence of key leakage. In *ASIACRYPT*, 2010.

17. Yevgeniy Dodis, Yael Tauman Kalai, and Shachar Lovett. On cryptography with auxiliary input. In *41st ACM STOC*. ACM Press, 2009.
18. Yevgeniy Dodis and Krzysztof Pietrzak. Leakage-resilient pseudorandom functions and side-channel attacks on feistel networks. In *CRYPTO*, pages 21–40, 2010.
19. Stefan Dziembowski. Intrusion-resilience via the bounded-storage model. In Shai Halevi and Tal Rabin, editors, *TCC 2006*, volume 3876 of *LNCS*, pages 207–224. Springer-Verlag, Berlin, Germany, March 2006.
20. Stefan Dziembowski. On forward-secure storage (extended abstract). In Cynthia Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 251–270. Springer-Verlag, Berlin, Germany, August 2006.
21. Stefan Dziembowski and Ueli M. Maurer. Optimal randomizer efficiency in the bounded-storage model. *Journal of Cryptology*, 17(1):5–26, January 2004.
22. Stefan Dziembowski and Krzysztof Pietrzak. Intrusion-resilient secret sharing. In *48th FOCS*, pages 227–237. IEEE Computer Society Press, October 2007.
23. Stefan Dziembowski and Krzysztof Pietrzak. Leakage-resilient cryptography. In *49th FOCS*, pages 293–302. IEEE Computer Society Press, 2008.
24. Taher ElGamal. On computing logarithms over finite fields. In Hugh C. Williams, editor, *CRYPTO'85*, volume 218 of *LNCS*, pages 396–402. Springer-Verlag, Berlin, Germany, August 1986.
25. Sebastian Faust, Eike Kiltz, Krzysztof Pietrzak, and Guy N. Rothblum. Leakage-resilient signatures. In *TCC 2010*, LNCS, pages 343–360. Springer-Verlag, Berlin, Germany, 2010.
26. Karine Gandolfi, Christophe Mourtel, and Francis Olivier. Electromagnetic analysis: Concrete results. In *CHES*, pages 251–261, 2001.
27. Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. One-time programs. In David Wagner, editor, *CRYPTO 2008*, LNCS, pages 39–56. Springer-Verlag, Berlin, Germany, August 2008.
28. Shafi Goldwasser and Guy N. Rothblum. Securing computation against continuous leakage. In *CRYPTO*, pages 59–79, 2010.
29. J. Alex Halderman, Seth D. Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A. Calandrino, Ariel J. Feldman, Jacob Appelbaum, and Edward W. Felten. Lest we remember: cold-boot attacks on encryption keys. *Commun. ACM*, 52(5):91–98, 2009.
30. Danny Harnik and Moni Naor. On everlasting security in the hybrid bounded storage model. In Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener, editors, *ICALP 2006, Part II*, volume 4052 of *LNCS*, pages 192–203. Springer-Verlag, Berlin, Germany, July 2006.
31. Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28(4):1364–1396, 1999.
32. IEEE P1363a Committee. IEEE P1363a / D9 — standard specifications for public key cryptography: Additional techniques. `http://grouper.ieee.org/groups/1363/index.html/`, June 2001. Draft Version 9.
33. Ali Juma and Yevgeniy Vahlis. Protecting cryptographic keys against continual leakage. In *CRYPTO*, pages 41–58, 2010.
34. Jonathan Katz and Vinod Vaikuntanathan. Signature schemes with bounded leakage resilience. In *ASIACRYPT*, 2009.
35. Neal Koblitz and Alfred. J. Menezes. Another look at generic groups. In Advances in Mathematics of Communications, Vol. 1, 2007, 13–28.

36. Paul C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In Neal Koblitz, editor, *CRYPTO'96*, volume 1109 of *LNCS*, pages 104–113. Springer-Verlag, Berlin, Germany, August 1996.
37. Paul C. Kocher and Joshua Jaffe. Leak-Resistant Cryptographic Method and Apparatus. United States Patent 6304658 B1. Oct. 16, 2001.
38. Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 388–397. Springer-Verlag, Berlin, Germany, August 1999.
39. Ueli M. Maurer. A provably-secure strongly-randomized cipher. In Ivan Damgård, editor, *EUROCRYPT'90*, volume 473 of *LNCS*, pages 361–373. Springer-Verlag, Berlin, Germany, May 1990.
40. Silvio Micali and Leonid Reyzin. Physically observable cryptography (extended abstract). In *TCC*, pages 278–296, 2004.
41. Moni Naor and Gil Segev. Public-key cryptosystems resilient to key leakage. In Shai Halevi, editor, *CRYPTO 2009*, LNCS, pages 18–35. Springer-Verlag, Berlin, Germany, August 2009.
42. V. I. Nechaev. Complexity of a determinate algorithm for the discrete logarithm. *Mathematical Notes*, 55(2):165–172, 1994.
43. European Network of Excellence (ECRYPT). The side channel cryptanalysis lounge. http://www.crypto.ruhr-uni-bochum.de/en_sclounge.html. retrieved on 29.03.2008.
44. Krzysztof Pietrzak. A leakage-resilient mode of operation. In Antoine Joux, editor, *EUROCRYPT 2009*, LNCS, pages 462–482. Springer-Verlag, Berlin, Germany, April 2009.
45. Jean-Jacques Quisquater and David Samyde. Electromagnetic analysis (ema): Measures and counter-measures for smart cards. In *E-smart*, pages 200–210, 2001.
46. Jean-Jaques Quisquater and Franois Koene. Side channel attacks: State of the art, October 2002. [43].
47. Charles Rackoff and Daniel R. Simon. Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack. In Joan Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 433–444. Springer-Verlag, Berlin, Germany, August 1992.
48. Certicom research, standards for efficient cryptography group (SECG) — sec 1: Elliptic curve cryptography. `http://www.secg.org/secg_docs.htm`, September 20, 2000. Version 1.0.
49. Victor Shoup. Lower bounds for discrete logarithms and related problems. In Walter Fumy, editor, *EUROCRYPT'97*, volume 1233 of *LNCS*, pages 256–266. Springer-Verlag, Berlin, Germany, May 1997.
50. Victor Shoup. ISO 18033-2: An emerging standard for public-key encryption. `http://shoup.net/iso/std6.pdf`, December 2004. Final Committee Draft.
51. Francois-Xavier Standaert, Olivier Pereira, Yu Yu, Jean-Jacques Quisquater, Moti Yung, and Elisabeth Oswald. Leakage resilient cryptography in practice. Cryptology ePrint Archive, Report 2009/341, 2009. `http://eprint.iacr.org/`.
52. Elena Trichina and Antonio Bellezza. Implementation of elliptic curve cryptography with built-in counter measures against side channel attacks. In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *CHES 2002*, volume 2523 of *LNCS*, pages 98–113. Springer-Verlag, Berlin, Germany, August 2002.
53. Salil P. Vadhan. On constructing locally computable extractors and cryptosystems in the bounded storage model. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 61–77. Springer-Verlag, Berlin, Germany, August 2003.

A Proof of Theorem 1

For technical reasons we will consider a setting where the generic bilinear group is extended with an additional oracle $\mathcal{O}_{DL} : \Xi \cup \Xi_T \to \mathbb{Z}_p \cup \perp$, we will call this the *extended* generic bilinear group model. This oracle is somewhat special, as it is stateful and its behaviour depends on all the queries that were ever made to the other group oracles $\mathcal{O}, \mathcal{O}_e, \mathcal{O}_T$. The oracle $\mathcal{O}_{DL}$ on input $\alpha \in \Xi$ outputs $\xi^{-1}(\alpha)$ if α was never received as output form $\mathcal{O}$ (by any party), and $\perp$ otherwise. Similarly, on input $\beta \in \Xi_T$ the output is $\xi_T^{-1}(\beta)$ if β was never received as output from $\mathcal{O}_e$ or $\mathcal{O}_T$ and $\perp$ otherwise. Thus $\mathcal{O}_{DL}$ is an oracle for the discrete log problem, but only works on inputs that have not yet appeared since the oracles $\mathcal{O}, \mathcal{O}_e, \mathcal{O}_T$ have been initialized.

We will show that the discrete logarithm problem is hard in the extended GBG model. This follows by a rather straight forward adaption of Shoup's proof that DL is hard in "normal" generic groups [49]. We then will reduce the CCLA1 security of BEG^* in GBG groups to the DL problem in *extended* BEG^*. The fact that we reduce to DL in extended GBG (and not just GBG) gives us an extra handle, the $\mathcal{O}_{DL}$ oracle, which will make the proof much cleaner and easier as we don't have to worry about adversaries who simply "guess" representations of group elements. Usually, when proving some lower bounds of some problem in a generic group, one can simply assume that an adversary never computes with representations which he did not explicitly receive as input or as output form its generic group oracle. But when considering leakage, this is no longer true, as now the adversary can learn information about the representation of group elements that are part of the secret state. The hard part of our security proof is to show that the adversary cannot accumulate enough information about any "interesting" representation as to guess it correctly with non-negligible probability. Simply not allowing the adversary to use representations it has not explicitly received form its generic group oracles would trivialize the problem.

We first prove the hardness of the discrete log problem in the extended GBG model.

Lemma 1. *Let*

$$\mathbf{Adv}^{\mathsf{DL}}(q,p) \stackrel{\mathsf{def}}{=} \max_{q-query} \Pr_{\mathcal{F}\, x,\xi,\xi_T} [\mathcal{F}^{\mathcal{O},\mathcal{O}_T,\mathcal{O}_e,\mathcal{O}_{DL}}(\xi(1),\xi(x)) \to x]$$

denote the advantage of the best q query adversary solving the discrete logarithm problem in the extended *bilinear generic group of order p. Then,* $\mathbf{Adv}^{\mathsf{DL}}(q,p) \leq \frac{q^2}{p}$.

Proof. Recall that in the extended GBG we first sample random representations ξ, ξ_T. Then we sample a random $y \stackrel{*}{\leftarrow} \mathbb{Z}_p$, ask $\mathcal{O}$ for $\xi(y)$ and then run $\mathcal{F}(\xi(1),\xi(y))$ who has accessed to the group oracles $\mathcal{O}, \mathcal{O}_e, \mathcal{O}_T$ and the discrete logarithm oracle $\mathcal{O}_{DL}$, which on input $\xi(z)$ or $\xi_T(z)$ will output z, except for $\xi(y)$ or any other element that $\mathcal{F}$ has received from its group oracles.

Let $\mathcal{F}'$ be and adversary where $\mathcal{F}'(\xi(1),\xi(y))$ simply runs $\mathcal{F}(\xi(1),\xi(y))$, but whenever $\mathcal{F}$ makes a query to either of $\mathcal{O}, \mathcal{O}_e, \mathcal{O}_T$ using as input an element α it

has not received as output from any group oracle (nor $\xi(y)$), it first makes the query $\mathcal{O}_{DL}(\alpha)$ to learn $\xi^{-}(\alpha)$ or $\xi_T^{-1}(\alpha)$. Finally $\mathcal{F}'$ outputs whatever $\mathcal{F}$ outputs, if the final output of $\mathcal{F}$ is z, then $\mathcal{F}'$ uses $\mathcal{O}$ to compute $\xi(z)$. By definition the advantage of $\mathcal{F}'$ in finding the discrete log is equal to the advantage of $\mathcal{F}$.

Let $\xi(u_1), \xi(u_2), \ldots$ and $\xi_T(v_1), \xi(v_2), \ldots$ denote all the representations that $\mathcal{F}'$ learns during execution (as they are output by $\mathcal{O}$ and $\mathcal{O}_e, \mathcal{O}_T$ respectively, except $u_1 = 1$ and $u_2 = y$ which are the input). By observing the oracle queries of $\mathcal{F}$, for every u_i we know $a_i, b_i \in \mathbb{Z}_p$ such that $u_i = a_i + b_i \cdot y$ and a'_i, b'_i, c'_i such that $v_i = a'_i + b'_i \cdot y + c'_i \cdot y^2$. This can be seen by induction on the queries that $\mathcal{F}$ makes. E.g. if $\xi_T(v_i) \leftarrow \mathcal{O}_e(u_j, u_k)$, then $a'_i = a_j + a_k$, $b'_i = a_j \cdot b_k + a_k \cdot b_j$ and $c'_i = b_j \cdot b_k$.

We say that $\mathcal{F}'$ finds a non-trivial collision, if for any i, j we have $u_i = u_j$ (but $(a_i, b_i) \neq (a_j, b_j)$) or $v_i = v_j$ (but $(a'_i, b'_i, c'_i) \neq (a'_j, b'_j, c'_j)$. If $\mathcal{F}'$ finds the discrete log, it also found a non-trivial collision (as we assume that $\mathcal{F}'$ knows $\xi(u_t)$ if it outputs u_t, thus if $u_t = y$ we have a non trivial collision with $u_2 = u_t$ (with $(a_2, b_2) = (0, 1)$ and $(a_t, b_t) = (y, 0)$). □

We now bound $\mathbf{Adv}^{\mathrm{cca}}_{\mathsf{BEG}}(\mathcal{F}, p)$ (or, equivalently, $\mathbf{Adv}^{\mathrm{ccla}}_{\mathsf{BEG}^*}(\mathcal{F}, p, 0)$) to prove standard CCA1 security of BEG.

Theorem 2 (CCA1 Security). *Consider the bilinear Diffie-Hellman key encapsulation* BEG *over the generic bilinear group of order $p \in \mathbb{N}$. Let $\mathcal{F}$ be any CCA1 adversary making q oracle queries to its generic group oracles, then*

$$\mathbf{Adv}^{\mathrm{cca}}_{\mathsf{BEG}}(\mathcal{F}, p) \leq 2 \cdot \mathbf{Adv}^{\mathsf{DL}}(q, p) \; . \tag{3}$$

Proof. We must upper bound the distinguishing advantage of any adversary $\mathcal{F}$ in the random experiment $\mathbf{Exp}^{\mathrm{cca1}}_{\mathsf{KEM}}(\mathcal{F}, p)$ for K_0 and K_1 which are defined as

$$K_0 = \xi_T(s \cdot x) \qquad K_1 = \xi_T(r)$$

where $r \xleftarrow{*} \mathbb{Z}_p$, $\xi(x)$ and $\xi_T(x)$ are the secret and public-key, and $\xi(s)$ is the encapsulated challenge key.

To this end, we construct an adversary $\mathcal{G}$ for the discrete log problem which can use $\mathcal{F}$ in a black-box fashion, and lower bound the advantage of $\mathcal{G}(\xi(1), \xi(y))$ in finding the discrete log y in terms of $\mathcal{F}$'s advantage in distinguishing K_0 from K_1 in the CCA1 experiment.

$\mathcal{G}(\xi(1), \xi(y))$ first flips a random coin $\beta \xleftarrow{*} \{0, 1\}$ and samples a random element $z \xleftarrow{*} \mathbb{Z}_p$. If $\beta = 0$ then $\mathcal{G}(\xi(1), \xi(y))$ simulates the experiment $\mathbf{Exp}^{\mathrm{cca1}}_{\mathsf{KEM}}(\mathcal{F}, p)$ using $\xi(x) := \xi(y)$ as the secret key and $\xi(s) := \xi(z)$ as the challenge encapsulated key. If $\beta = 1$ it is the other way round, i.e., the discrete log challenge is used as the challenge encapsulated key $\xi(s) := \xi(x)$ and $\xi(x) := \xi(z)$.

During the simulated experiment, $\mathcal{F}$ will receive outputs $\xi(u_1), \xi(u_2), \ldots$ from $\mathcal{O}$. For each of u_i, $\mathcal{G}$ knows a_i, b_i such that

$$a_i + b_i \cdot s \tag{4}$$

To see this, assume this is true before $\mathcal{F}$ makes the ith query $\xi(u_i) \leftarrow \xi(\alpha_1, \alpha_2)$. If α_1 has been received as input (i.e., is $\xi(1)$ or $\xi(s)$) or as output from $\mathcal{O}$ then

by assumption we know a, b s.t. $\alpha_1 = \xi(a + b \cdot s)$. Otherwise (i.e., if $\mathcal{F}$ just guessed some representation α_1 which it has not encountered before) $\mathcal{G}$ queries $\mathcal{O}_{DL}(\alpha_1)$ and will receive $a = \xi^{-1}(\alpha_1)$. By the same argument $\mathcal{G}$ knows a', b' s.t. $\xi(\alpha_2) = a' + b' \cdot s$. Now $u_i = a'' + b'' \cdot s$ with $a'' = a + a'$ and $b'' = b + b'$.

Similarly, $\mathcal{F}$ (using its inputs $\xi(1)$, the public key $\xi_T(x)$, encapsulated key $\xi(s)$ and the challenge $\xi_T(k)$) will receive outputs $\xi_T(v_1), \xi_T(v_2), \ldots$ from $\mathcal{O}_e$ or $\mathcal{O}_T$, and for each v_i the $\mathcal{G}$ will know a_i, b_i, c_i, d_i, e_i such that

$$v_i = a_i + b_i \cdot x + c_i \cdot s + d_i \cdot s^2 + e_i \cdot k \tag{5}$$

We say that $\mathcal{F}$ found a non-trivial collision if either $u_i = u_j$ for some i, j where $(a_i, b_j) \neq (a_j, b_j)$ or $v_i = v_j$ for some i, j where $(a_i, \cdots, e_i) \neq (a_j, \cdots, e_j)$. The probability of $\mathcal{F}$ finding a collision can be upper bounded by q^2/p (see e.g. [49]) in both cases (i.e., when $\xi_T(k) = \xi_T(x \cdot s)$ is the correct key or $\xi_T(k)$ is random). If a collision $v_i = v_j$ happens, $\mathcal{G}$ can always extract either s or x (or both), if $u_i = u_j$ $\mathcal{G}$ can always extract s. Thus if a collision happens, $\mathcal{G}$ will learn the discrete log of $\xi(y)$ (recall that $y = s$ or $y = x$) with probability at least $1/2$, so

$$\Pr[\mathcal{F} \text{ finds a collision}] \leq 2 \cdot \mathbf{Adv}^{\mathsf{DL}}(q, p) \tag{6}$$

Further, conditioned on no collision happening, the view of $\mathcal{F}$ is exactly the same no matter if $k = x \cdot s$ or $k \xleftarrow{*} \mathbb{Z}_p$, thus

$$\mathbf{Adv}^{\text{cca1}}_{\mathsf{BEG}^*}(\mathcal{F}, p) \leq \Pr[\mathcal{F} \text{ finds a collision}] \tag{7}$$

The theorem follows from eq.(6) and eq.(7). □

We now prove that BEG^* is a leakage resilient CCA1 secure PKE as claimed in Theorem 1. Lemma 2 below is a quantitative statement of Theorem 1.

Lemma 2 (CCLA1 Security). *Consider the bilinear Diffie-Hellman key encapsulation BEG^* over the generic bilinear group of order $p \in \mathbb{N}$. Let $\mathcal{F}$ be any CCLA1 adversary making q oracle queries to its generic group oracles, then*

$$\mathbf{Adv}^{\text{ccla1}}_{\mathsf{BEG}^*}(\mathcal{F}, p, \lambda) \quad \leq \quad 3 \cdot \mathbf{Adv}^{\mathsf{DL}}(q, p) + \mathbf{Adv}^{\mathsf{DL}}(q, p) \cdot 2^{2\lambda} \cdot q + \frac{q^2 \cdot 2^{2\lambda}}{p - q} \quad \leq \quad \frac{2^{2\lambda+1} \cdot q^3}{p}$$

Proof. The difference to the proof of Theorem 2 is that now we consider a CCLA1 (and not just CCA1) adversary $\mathcal{F}$. Thus $\mathcal{F}$, when making the ith decapsulation query $\xi(\gamma_i)$, will not only get the decapsulated key $\xi_T(\gamma_i \cdot x)$, but also the leakage

$$\Lambda_i = f_i(\xi(R_{i-1}), r_i) \qquad \Lambda'_i = g_i(\xi(x - R_{i-1}), \xi_T(\gamma_i \cdot R_{i-1}), r_i)$$

Recall that $r_0, r_1, \ldots$ are all sampled uniformly at random, and $R_i = \sum_{j=0}^{i} r_i$. As in the proof of Theorem 2, we consider an adversary $\mathcal{G}(\xi(1), \xi(y))$ for the discrete logarithm problem in the extended GBG which will use the CCLA1 adversary $\mathcal{F}$ in a black-box way.

$\mathcal{G}$ initially samples a random $\beta \leftarrow \{0, 1, 2\}$ and then simulates the random experiment $\mathbf{Exp}^{\text{ccla1}}_{\mathsf{KEM}}(\mathcal{F}, p, \lambda)$, where if $\beta = 0, 1$ or 2 $\mathcal{G}$ uses its discrete logarithm

challenge as $\xi(x)$, $\xi(s)$ or $\xi(r_0)$ respectively (the other two values $\mathcal{G}$ samples at random, note that to compute the initial sharing $\xi(r_0), \xi(x - r_0)$ it is sufficient to know either $x, \xi(r_0)$ or $r_0, \xi(x)$).

Assume that $\mathcal{F}$ and all the leakage functions f_i, g_i would behave very "nice" and only make queries to the group oracles $\mathcal{O}, \mathcal{O}_e, \mathcal{O}_T$ with inputs that they received as inputs or outputs from the group oracles. Then (as in the proof of Theorem 2) for all the outputs $\xi(u_1), \xi(u_2), \ldots$ and $\xi_T(v_1), \xi_T(v_2), \ldots$ that $\mathcal{F}$ will learn, $\mathcal{G}$ knows a_i, b_i such that (4) holds, and $a_i, \ldots, e_i$ such that (5) holds.

Moreover for all outputs $\xi(u_1^f), \xi(u_2^f), \ldots$ and $\xi_T(v_1^f), \xi_T(v_2^f), \ldots$ that the leakage functions $f_1, f_2, \ldots$ get as outputs from their oracles, $\mathcal{G}$ will know a_i, b_i, c_i such that (recall that f_i will as additional input get $\xi(R_{i-1}) = \xi(r_0 + \sum_{j=1}^{i} r_{i-1})$ where $\mathcal{G}$ knows all r_i except possibly r_0, moreover no leakage functions will be evaluated after $\mathcal{F}$ gets the challenge $\xi(s)$, that is why s does not show up in the equation below)

$$u_i^f = a_i + b_i \cdot r_0 \tag{8}$$

and it will know $a_i, \ldots, e_i$ such that

$$v_i^f = a_i + b_i \cdot x + c_i \cdot r_0 + d_i \cdot r_0^2. \tag{9}$$

Similarly, all outputs $\xi(u_1^g), \xi(u_2^g), \ldots$ and $\xi_T(v_1^g), \xi_T(v_2^g), \ldots$ that the leakage functions $g_1, g_2, \ldots$ get, $\mathcal{G}$ can write as

$$u_i^g = a_i + b_i \cdot (x - r_0) \tag{10}$$

and it will know $a_i, \ldots, e_i$

$$v_i^g = a_i + b_i \cdot x + c_i \cdot (x - r_0) + d_i \cdot (x - r_0)^2 \tag{11}$$

As in Theorem 2, we say that we have a non-trivial collision, if at any point in this experiment $\mathcal{G}$ learns a non-trivial equality. This can be any collision between two u_i, u_i^f, u_i^g values or v_i, v_i^g, v_i^g values which are not trivially equivalent.

As in Theorem 2, the advantage of $\mathcal{F}$ in distinguishing $\xi(x \cdot s)$ from a random element can be upper bounded by the probability that $\mathcal{F}$ provokes a non-trivial collision, and from a non-trivial collision, $\mathcal{G}$ can always extract either r_0, x or s, and thus solve its discrete log challenge if it embedded its DL challenge in the value that can be extracted. Thus, for an $\mathcal{F}$ which behaves "nicely" (as we defined before)

$$\mathbf{Adv}^{\mathrm{ccla1}}_{\mathsf{BEG}^*}(\mathcal{F}, p, \lambda) \leq 3 \cdot \mathbf{Adv}^{\mathsf{DL}}(q, p)$$

For the above argument to go through we don't require $\mathcal{F}$ and the leakage functions to behave "nice", it is fine if, e.g., $\mathcal{F}$ uses an input $\alpha \in \Xi$ to $\mathcal{O}$ which it did not explicitly receive, as long as either

- $\mathcal{G}$ knows a, b s.t. $\xi^{-1}(\alpha) = a + b \cdot s$ (say, because some leakage function f_i received this input, then $\mathcal{G}$ knows a', b' such that $\alpha = a' + b' \cdot r_0$, and whenever $b' = 0$ we simply have $a = a'$ and $b = 0$).
- α has never appeared during the entire experiment, then $\mathcal{G}$ can learn $\xi^{-1}(\alpha)$ by querying $\mathcal{O}_{DL}$.

We define an event $\mathcal{E}_{\mathrm{g}}$ (where g stands for "guess") which holds if during the entire experiment $\mathbf{Exp}^{\mathrm{ccla1}}_{\mathsf{KEM}}(\mathcal{F}, p, \lambda)$ (simulated by $\mathcal{G}$) if neither $\mathcal{F}$ nor any f_i or g_i makes a "bad" query (to be defined) to a group oracle. A query made by $\mathcal{F}$ is "bad" if it contains an element which $\mathcal{G}$ cannot write as (4) or (5), a query made by f_i is "bad" if $\mathcal{G}$ cannot write as (8) or (9) and a query made by g_i is "bad" if $\mathcal{G}$ cannot write it as (10) or (11).

So unless $\neg\mathcal{E}_{\mathrm{g}}$ occurs, $\mathcal{G}$ will learn the discrete log of its challenge $\xi(y)$ whenever $\mathcal{F}$ finds a non-trivial collision with probability 1/3 (if it initially sampled the right β). That is, for any $\mathcal{F}$ we have

$$\mathbf{Adv}^{\mathrm{ccla1}}_{\mathsf{BEG}^*}(\mathcal{F}, p, \lambda) \leq 3 \cdot \mathbf{Adv}^{\mathsf{DL}}(q, p) + \Pr[\neg\mathcal{E}_{\mathrm{g}}] \tag{12}$$

Thus it remains to bound the probability $\Pr[\neg\mathcal{E}_{\mathrm{g}}]$ that a "bad" query occurs. Note that if the leakage bound λ is so big that the leakage function can output an entire group element, then provoking $\neg\mathcal{E}_{\mathrm{g}}$ is trivial, just let f_0 output $\xi(r_0)$, and then $\mathcal{F}$ uses $\xi(r_0)$ as input to $\mathcal{O}$. Thus the leakage bound λ will necessarily come up when bounding the probability of this event.

We will first define and bound the probability of another event $\mathcal{E}_{\mathrm{r}}$. Recall that as long as $\mathcal{E}_{\mathrm{g}}$ holds for every representation $\xi(u_i^f)$ learned by f_j we know a_i, b_i s.t. $u_i^f = a_i + b_i \cdot r_0$. We say that such an element is *bound* if $b_i \neq 0$. Similarly, v_i^f, u_i^g, v_i^g are bound if c_i, b_i, c_i respectively are not 0 (using notation from eq.(9)-(11).)

Now the event $\mathcal{E}_{\mathrm{r}}$ fails to hold, if in any two leakage functions f_i and f_j (or g_i and g_j) where $i < j+1$ get to see the same bound element (either as input, or output from the generic group oracles). Note that we need to require $i < j+1$ (and not just $i < j$) as f_i and f_{i+1} by definition can get the same bound element, namely $\xi(R_{i+1})$. We will upper bound the probability of $\mathcal{E}_{\mathrm{r}}$ as

Claim.

$$\Pr[\neg\mathcal{E}_{\mathrm{r}}] \leq \mathbf{Adv}^{\mathsf{DL}}(q, p) \cdot 2^{2\lambda} \cdot q \tag{13}$$

Proof (of Claim). We will explain how to construct an adversary $\mathcal{F}'$ which solves the discrete logarithm problem with probability $\epsilon/2^{2\lambda}q$ if given black-box access to an adversary $\mathcal{F}$ which can provoke the event $\neg\mathcal{E}_{\mathrm{r}}$ with probability ϵ.

$\mathcal{F}'(\xi(1), \xi(y))$ simulates the $\mathbf{Exp}^{\mathrm{ccla1}}_{\mathsf{KEM}}(\mathcal{F}, p, \lambda)$ experiment sampling the secret key x, the challenge s and all the r_i, except for one random $j \geq 1$, where $\mathcal{F}'$ will try to use y for r_j in the simulation of the experiment. Note that even knowing only $\xi(r_j) = \xi(y)$ (and not r_j), $\mathcal{F}'$ can simulate the entire experiment, except for the leakage Λ_j, Λ'_j (which is a function of r_j). We let $\mathcal{F}'$ simply guess this leakage uniformly at random. As $|\Lambda_j| + |\Lambda'_j| = 2\lambda$, $\mathcal{F}'$ will guess correctly with probability $2^{-2\lambda}$. Thus $\mathcal{F}$ will provoke $\neg\mathcal{E}_{\mathrm{r}}$ with probability at least $\epsilon/2^{2\lambda}$ in the simulated experiment. Assume $\mathcal{F}'$ guessed correctly and two leakage functions f_i, f'_i where $i < i' + 1$ (and $j \neq i, i'$) computed the same bound element $\xi(u)$, as said, this happens with probability at least $q/2^{2\lambda}$. (The same argument works with the g's instead the f's.) Then $\mathcal{F}'$ (observing $\mathcal{F}$'s queries) can write $u = a + b \cdot R_i$ and $u = a' + b' \cdot R_{i'}$ for known a, b, a', b'. Now assume that $i < j < i'$ (which happens

with probability at least $1/q$). Then, as $\mathcal{F}'$ chooses all the r's except r_j, $\mathcal{F}$ will know R_i and thus u, and will also be able to write $u = a'' + b'' \cdot r_j$ for a $b'' \neq 0$ (namely $b'' = b'$ and $a'' = a' + b' \cdot (R_{i'} - r_j)$, note that $\mathcal{F}'$ knows $R_{i'} - r_j$ as it chooses all the r's except r_j) and then can solve this equation for r_j (which is the DL challenge y). $\triangle$

We can bound $\Pr[\neg\mathcal{E}_{\mathrm{g}}]$ as[12]

$$\Pr[\neg\mathcal{E}_{\mathrm{g}}] = \Pr[\neg\mathcal{E}_{\mathrm{g}} \wedge \mathcal{E}_{\mathrm{r}}] + \Pr[\neg\mathcal{E}_{\mathrm{r}} \wedge \neg\mathcal{E}_{\mathrm{r}}] \leq \Pr[\neg\mathcal{E}_{\mathrm{g}} \wedge \mathcal{E}_{\mathrm{r}}] + \Pr[\neg\mathcal{E}_{\mathrm{r}}] \tag{14}$$

As we already bounded $\Pr[\neg\mathcal{E}_{\mathrm{r}}]$ in the previous Claim, it remains to bound $\Pr[\neg\mathcal{E}_{\mathrm{g}} \wedge \mathcal{E}_{\mathrm{r}}]$. The event $\neg\mathcal{E}_{\mathrm{g}} \wedge \mathcal{E}_{\mathrm{r}}$ means that at some point, $\mathcal{F}$ or some leakage function guessed the representation of some element on with at most 2λ bits were leaked (this is enforced by $\mathcal{E}_{\mathrm{r}}$). The guessed element is uniformly random over at least $p - q$ possible representations (it is $p - q$ not p as $\mathcal{F}$ can exclude some representations it has already observed). As the expected probability of guessing an element X with $H_\infty(X) = m$ given $f(X)$ for any function $f(\cdot)$ with range n is at most 2^{n-m}, we get (the q^2 term in the equation below comes from taking the union bound over any guessed query and any bound element, both are at most q).

$$\Pr[\neg\mathcal{E}_{\mathrm{g}} \wedge \mathcal{E}_{\mathrm{r}}] \leq \frac{q^2 \cdot 2^{2\lambda}}{p - q} \tag{15}$$

The bound claimed in the Theorem follows from eq. (12) to (15). $\square$

[12] We use that for any events $\mathcal{E}, \mathcal{E}'$ we have $\Pr[\mathcal{E}] = \Pr[\mathcal{E} \wedge \mathcal{E}'] + \Pr[\mathcal{E} \wedge \neg\mathcal{E}']$ and $\Pr[\mathcal{E} \wedge \mathcal{E}'] \leq \Pr[\mathcal{E}]$.