How to Keep a Secret: Leakage Deterring Public-key Cryptosystems[⋆]

Aggelos Kiayias[1] and Qiang Tang[2⋆⋆]

[1] University of Edinburgh
akiayias@inf.ed.ac.uk
[2] New Jersey Institute of Technology
qiang@njit.edu

Abstract. How is it possible to prevent the sharing of cryptographic functions? This question appears to be fundamentally hard to address since in this setting the owner of the key *is* the adversary: she wishes to share a program or device that (potentially only partly) implements her main cryptographic functionality. Given that she possesses the cryptographic key, it is impossible for her to be *prevented* from writing code or building a device that uses that key. She may though be *deterred* from doing so. We introduce *leakage-deterring* public-key cryptographic primitives to address this problem. Such primitives have the feature of enabling the embedding of owner-specific private data into the owner's public-key so that given access to *any* (even partially functional) implementation of the primitive, the recovery of the data can be facilitated. We formalize the notion of leakage-deterring in the context of encryption, signature, and identification and we provide efficient generic constructions that facilitate the recoverability of the hidden data while retaining privacy as long as no sharing takes place.

1 Introduction

Consider any organization that maintains a PKI supporting various cryptographic functions including public-key encryption, signatures and identification. How is it possible to prevent individuals from sharing their cryptographic functions? Certified PKI members, out of convenience or even malice, can delegate their private keys to each other (or even to outsiders), thus violating accountability and organizational policy. Even worse, delegation can be partial: for instance, a public-key encryption user can share (or, in fact, even sell) an implementation that only decrypts messages of a certain form (e.g., only e-mails from a specific source). Seemingly, very little can be done to prevent this as the adversary in this case is the owner of the cryptographic key and hence she may freely choose to share it with others either directly or by incorporating its functionality within a larger system that is shared.

The above scenario puts forth the central problem our work aims to solve: how is it possible to prevent the sharing of cryptographic functions? The main challenge here is that the owner of the key is adversarial: she wishes to share a program or hardware device that (potentially only partly) implements her main cryptographic functionality. Given that she possesses the cryptographic key (either in software or hardware), it is impossible for her to be prevented from delegating it. However, as we highlight, there can be ways for her to be *deterred* from doing so. A straightforward deterrence mechanism would be to identify and penalize the sharing behavior. However, the enforcement of a penalty mechanism is contingent on detecting the act of sharing — something that limits the

[⋆] Preliminary version of this paper appeared at ACM CCS 2013. This research was supported by ERC project CODAMODA.

[⋆⋆] Work done while the second author was visiting University of Athens.

effectiveness of penalties: a cautious adversary can keep itself "below the radar" and thus remain penalty-free. To address this we put forth and explore a more proactive approach.

A cryptographic scheme will be called *leakage-deterring* if the release of any implementation of the cryptographic function (e.g, decryption, signing), leads to the recovery of some private information (that the owner prefers to keep hidden) by anyone that possesses the implementation. Leakage deterrence is thus achieved in the sense that sharing the cryptographic function in any form incurs the penalty of revealing the private information (while non-sharing maintains its privacy).

Note that a leakage-deterring primitive should retain its original functionality (e.g., encryption, signing, identification) but it also offers two additional operations: first, it is possible to embed private data into the public-key of the primitive in a way that they are (at least) semantically secure – we call this property *privacy*. The embedding operation is facilitated through an interaction with an authority that vouches for the integrity of the private data and is akin to a PKI certification of the owner's public-key. In this fashion, the primitive's public-key becomes "enhanced" and is a carrier of private information itself (i.e., a ciphertext) — otherwise the intended functionality of the primitive should remain unchanged. The second operation that is offered by a leakage-deterring primitive comes into play when the owner of the secret key produces an implementation of the main operation in the form of a "box" and shares it with other entities (in software or hardware). Given such a box, any entity that receives it can utilize a public recovering algorithm that will interact with the box and produce the private data that are embedded into the owner's enhanced public-key – we call this property *recoverability.*

In a nutshell, designing a leakage-deterring scheme requires the transformation of the public-key of the primitive into a (one-time) ciphertext that can be decrypted by *any* working implementation of the cryptographic functionality. The main challenge comes precisely from this latter requirement: any working implementation of the cryptographic functionality should be usable as a decryption key that unlocks the private data embedded into the public-key, even if the adversarial implementor takes into account the recoverability algorithm and the enhanced public-key that carries the private data when implementing the functionality.

To appreciate the complexity of the problem, consider a naive attempt to produce a leakage-deterring public-key encryption (PKE): the authority certifies as the enhanced public key the pair (pk, ψ) where $\psi = \mathbf{Enc}(pk, s)$ and s is the private data related to the owner. Recoverability can be attempted by feeding ψ into a decryption box. It is apparent that this construction can be defeated by an adversarial implementation of decryption that given input c, it decrypts it only in the case $c \neq \psi$ (or even $\mathbf{Dec}(c) \neq s$). The constructions we seek should facilitate recoverability even if the adversarial box implementor releases implementations that work for *arbitrary* input distributions of her choice.

The applications of leakage-deterring cryptographic primitives are in any context where the intentional leakage of a cryptographic functionality should be deterred or restricted in some fashion or in a context where the leakage of an implementation should enable the computation of a value that is otherwise hidden. In the simplest scenario, the enhanced public-key contains some piece of information that the owner prefers to keep secret (e.g., her credit-card number or similar piece of private information as suggested by Dwork, Lotspiech and Naor [13] that introduced the concept of *self-enforcement*; more effectively, as we argue here, is to use a Bitcoin account secret-key, see below). It follows that the system setup "self-enforces" the owner to keep the cryptographic functionality to herself. Depending on the deployment environment, different types of secret-information can be used. We describe more application scenarios of leakage deterring cryptographic primitives in section 7.

Our Contributions. We introduce, formalize and implement leakage-deterring cryptographic primitives for public-key encryption, digital signatures, and identification schemes. The main technical contributions we provide are three different techniques for constructing leakage-deterring cryptographic primitives. Our techniques enable the secure embedding of private information into the public key of the primitive in a way that is recoverable given any (even partially) working implementation. Our first method, applies to encryption that is partially homomorphic; given a box that works only for some adversarially chosen distributions we show how to exploit the homomorphic property to appropriately manipulate a target ciphertext and make it decryptable by the adversarial decryption box. Our second method, which can rely on any encryption scheme, hides the key that unlocks the private information into an exponentially large key space that is encoded in the public-keys. By using appropriate redundancy in the public key space we enable the tracing of the vector of keys that identify the private information, out of any (even partially working) implementation. Achieving recoverability while maintaining small ciphertext size in this setting requires an involved recoverability algorithm which is one of the highlights of our contributions. Finally, our third method applies to signature and identification schemes. It uses the fact that working implementations of suitably chosen such primitives can be used to build "knowledge extractors." These are algorithms that reveal information about the secret-key of the underlying primitives which we use to hide the private information. We note that the idea of using extractors for preventing the transfer of identification tokens has been used before [7, 34, 6, 25] in the sense that sharing a token implies sharing the key. Going beyond this we show here that secret key can be of sufficient entropy so that it simultaneously hides the embedded owner information while still maintaining the security of the underlying scheme. In fact we show that no additional intractability assumptions are necessary for achieving leakage-deterring signature and identification schemes.

Our first construction for public-key encryption requires a standard homomorphic property and achieves constant size ciphertexts while offering recoverability for any (non-trivial) adversarial distribution. The second construction is generic and the size of ciphertexts is a parameter that increases as the min-entropy of the allowed adversarial distributions becomes smaller. We analyze our constructions in the IND-CPA setting and then present a generic transformation to obtain IND-CCA2 security[3]. It is evident that there is a trade-off between privacy and recoverability.[4] For encryption schemes, we aim at maximizing the recoverability while privacy can only be achieved if no decryption query is allowed. For the case of signatures, we present a construction that maintains the privacy of the embedded information even if the adversary has arbitrary access to the signing functionality (which is most desirable since digital signatures are typically publicly available). We still manage to enable recoverability by exploiting the random oracle model and non-black-box access to the implementation. Security properties of our identification schemes are shown in the

[3] It may come as a surprise that recoverability and IND-CCA2 can actually coexist. Attaining IND-CCA2 intuitively means that a decryption oracle basically leaks no useful information about manipulated ciphertexts. Thus, the recovering algorithm can seemingly do nothing useful with access to a decryption implementation beyond decrypting valid ciphertexts, which if related to the enhanced public-key can be rejected. Still, the paradox can be resolved, if one observes that the decryption oracle should be useless only with respect to breaking the security of regular ciphertexts and not the security of the data that are somehow embedded into the enhanced public-key.

[4] In particular, black-box recoverability and privacy w.r.t secret key oracles, are antagonistic to each other. See section 3.2 for more details.

standard model. To attain privacy in the standard model we utilize strong extractors for random variables with high conditional unpredictability.

Using Bitcoin as collateral. In follow up work [27], we describe how a service provider can use bitcoin as a collateral to de-incentive unauthorized pirate decoder re-distribution in multi-recipient encryption setting. The concept can be similarly combined with our techniques here as well, in order to establish leakage-deterrence. Recall that the user secret can be an arbitrary string that is “vouched” and embedded by an the authority, e.g., the CA, hence a bitcoin account secret key can be chosen to be the user secret. In particular, the CA generates a new bitcoin account when the user requests a certificate for her public key. Suppose s is the secret key associated with the bitcoin account; the CA will embed s privately into the certificate and the user will transfer a small amount of bitcoin into the account. As long as the amount of bitcoin is held frozen in the account corresponding to s, no action will be required. In case sharing happens, with our recovering algorithm, any recipient of a partially working software that implements the cryptographic functionality will retrieve the secret key s, and thus may transfer the money out from the bitcoin account corresponding to the key owner. Note that bitcoin has a public ledger that records every transaction, hence the CA can observe in the transaction ledger that money are removed from the collateral account and when this happens it can issue a revocation for this certificate. In this way key owners can be self-enforced to keep their secret keys to themselves since there will be an immediate monetary penalty (or certificate revocation) associated with sharing the functionality. Observe that the user can retrieve her money back when the certificate expires.

Related work. The most relevant work to ours is [13] that introduced self-enforcement as a way of avoiding illegal content redistribution in a multi-user setting. Self-enforcement was argued in that paper by ensuring (under certain assumptions) that an owner has only two options when implementing a decoder: either using her private key (that includes private personal information), or encoding a derived key that is of size proportional to the data to be decrypted. In our terminology, this means that the schemes of [13] exhibit a leakage-deterrence/program-length tradeoff and hence are not leakage-deterring *per se*. Furthermore, recoverability in [13] is only “white-box” as opposed to the black-box type that our constructions achieve. On the other hand, our work focuses on the single user setting – it is an interesting direction to design leakage-deterring primitives in the multi-user setting. In another related line of work [7, 34, 6, 25] it was discussed how to deter a user from transferring her credentials (or the secret key directly) to others in the context of identification systems. The techniques from these works – by nature – were restricted to only identification schemes and digital signatures. In contrast, our work encompasses all major public key cryptographic primitives (including public-key encryption). The primitive of circular encryption introduced in [6] might look promising at first sight to achieve the leakage-deterrence in the public-key encryption setting as well, however, no recovery mechanism which works for all partial implementations is provided by this primitive. To resolve the main difficulty we explained in a previous paragraph, new techniques other than circular encryption alone are needed. Furthermore, for the case of identification and signature schemes, our method shows that leakage-deterrence can be achieved without any assumptions beyond the one employed by the underlying primitive. Other forms of share deterring techniques were considered in various settings, e.g., limited delegation [17], data collection [18], e-payments [39] or designated verifier signatures in [32, 40] in the form of *non-delegatability* (which is a weaker notion than our leakage-deterring concept).

Another related notion, introduced in [35], dealt with the problem *copyrighting* a public-key decryption function: a single public-key decryption functionality should be implemented in many distinct ways so that if an implementation is derived from some of them, then it is possible to discover the index of at least one of the implementations that was used. This notion was further investigated in [29] and was related to traitor tracing schemes [10]. In the context of public-key encryption, the objective of copyrighting a function or of a traitor tracing scheme is orthogonal to ours. While in both cases we deal with adversarial implementations of cryptographic functionalities (hence the similarities in terminology), the adversarial goal is different: in the case of traitor tracing, the adversary has many different implementations of the same functionality and tries to produce a new one that is hard to trace back to the ones she is given. In an attack against a leakage-deterring scheme on the other hand, the adversary possesses an implementation of a cryptographic functionality and tries to modify it in a way that it cannot be used to extract some information that is hidden in the primitive's public-key.

More importantly, there are two critical technical difference between the notion of traitor tracing and share deterring. (1.) Traitor tracing is normally run only by the service provider (an authority who sets up the system), and often requires the tracer to have a secret tracing key; while the main motivation of share deterring PKE is to proactively deter the illegal redistribution and thus the recover algorithm should be run by any recipient using strictly public information only. (2.)Traitor tracing only requires to extract an identity which lies in a polynomial size space, however, in share deterring PKE, the user collateral information could be in arbitrary form, thus it has to lie in an exponentially large space. Combining the two functionalities in one is an interesting question. (a step towards this general direction but in a much weaker model than ours was suggested in the work of [30] but the leakage-deterring aspect (in our terminology) was found to be insecure in [28]. A follow up work of us [27] gave a systematic study of this idea and finally resolve the problem).

Accountable authority identity based encryption (AIBE) [19, 20, 31, 38, 26] considers the problem of judging whether an implementation of decryption belongs to the owner or the PKG (in the context of IBE). In this setting, both the owner and the PKG may be the potential adversary who try to implicate the other. Hence, some property similar to our recoverability is needed. In any case, the single bit decisional output required by AIBE is much weaker than our recoverability requirement in leakage-deterring public-key encryption (even in the IBE setting) where by interacting with a decryption box, one should recover the whole private data embedded in the enhanced public-key. Notably, in [23], the authors considered how to defend key cloning attacks in attribute based encryption. This notion is similar to our concept of leakage-deterrence, in that a secret key corresponding to a user ID will enable the holder/recipient of the key to extract some personal information of ID. However, there are following major differences: (1.) An extra token server is required to assist decryption and the recovering of the personal secret; (2.) Key uncloneability focused on defending the malicious key owner from copying her key material directly, thus in our terminology, it is white-box leakage-deterrence.

2 Preliminaries

First, we recall some known primitives and results which we utilize in our constructions or security analysis.

Proof of Knowledge: [1] Proof of knowledge protocol is one that a prover convinces the verifier he knows a witness to a publicly known polynomial-time predicate. This is a protocol between two parties P, V where P proves a statement $x \in L$ for a language L's instance x with its witness w from a witness set denoted by $W(x)$. Suppose $O_V[P(x,w) \leftrightarrow V(x)]$ denotes the output of the verifier

V after interacting in the protocol with the prover P, a proof of knowledge protocol satisfies the following two properties:

- Completeness: Honest prover always convinces the verifier: if $w \in W(x)$, then $\Pr[O_V[(P(x,w) \leftrightarrow V(x)] = 1] = 1$.
- Soundness: There exist an expected polynomial-time "knowledge extractor" $\mathbf{E}$ which interacts with a malicious prover P^*, and outputs a witness with overwhelming probability as long as the success probability that P^* convinces V is non-negligible. Formally, for all x, w^*, whenever $\Pr[O_V[(P^*(x,w^*) \leftrightarrow V(x)] = 1]$ is non-negligible it holds that $\Pr[\mathbf{E}^{P^*}(x) \in W(x)]$ happens with overwhelming probability.

Σ-Protocol: [12] One frequently used type of proof of knowledge protocols is the class of Σ-protocols, which have a three move structure (a, e, z), starting with the prover sending a 'commit' message a, then the verifier sending a 'challenge' message e, and finally the prover answering with a 'response' message z. Using the Fiat-Shamir transformation [14], one can construct a signature scheme based on such protocol in the random oracle model [4]. Security of such signature schemes is comprehensively studied in [37], and it mainly relies on the existence of a knowledge extractor algorithm (which is implied by the soundness of the protocol).

General Forking Lemma: [2] The general forking lemma states that that if an adversary, on inputs drawn from some distribution, produces an output, then the adversary will produce another correlated output with different inputs from same distribution and same random tape. Rigorously, let $\mathcal{A}$ be a probabilistic algorithm, with inputs $(x, r_1, \ldots, r_q; \rho)$ that outputs a pair (J, σ), where ρ refers to the random tape of $\mathcal{A}$ (that is, the random coins $\mathcal{A}$ will make). Suppose further that x is sampled from some distribution X, and R is a super-polynomially large set and r_i is sampled uniformly from R. Let *acc* be the probability that $J \geq 1$. We can then define a "forking algorithm" as follows,

- on input x: pick a random tape ρ for $\mathcal{A}$.
- $r_1, \ldots, r_q \overset{\mathrm{r}}{\leftarrow} R$.
- $(J, \sigma) \leftarrow \mathcal{A}(x, r_1, \ldots, r_q; \rho)$
- If $J = 0$, return $(0, \epsilon, \epsilon)$.
- $r'_J, \ldots, r'_q \overset{\mathrm{r}}{\leftarrow} R$
- $(J', \sigma') \leftarrow \mathcal{A}(x, r_1, \ldots, r_{J-1}, r'_J, \ldots, r'_q; \rho)$
- If $J' = J$ and $r_J \neq r'_J$ then return $(1, \sigma, \sigma')$, otherwise, return $(0, \epsilon, \epsilon)$.

Let *frk* be the probability that $\mathcal{A}$ outputs (b, σ, σ'), and $b = 1$, then $\mathit{frk} \geq \mathit{acc}(\frac{\mathit{acc}}{q} - \frac{1}{|R|})$.

Strong one-time signature [33] A signature scheme **Sig**=(**KeyGen**, **Sign**, **Verify**) is a strong one-time signature scheme if the success probability of any PPT adversary $\mathcal{A}$ in the following game is negligible:

- The challenger sends $\mathcal{A}$ a verification key vk.
- $\mathcal{A}$ asks one signing query on some message m chosen by her and the challenger returns a valid signature σ.
- $\mathcal{A}$ outputs a new signature (m^*, σ^*).

We say the adversary succeeds if: $\mathbf{Verify}(vk, (m^*, \sigma^*)) = 1 \wedge (m^*, \sigma^*) \neq (m, \sigma)$.

3 Definitions and Security Modeling

3.1 Definitions of Leakage-Deterring Cryptographic Primitives

A leakage-deterring cryptographic primitive includes two additional algorithms on top of the regular algorithms the primitive should normally possess: **EnKey**$(\cdot)$, which embeds some (private) owner related information into the public key, and **Rec**$(\cdot)$ which recovers the private information from the public-key by interacting with any non-trivial implementation (or "box") and can be executed by anyone. While the concept of a leakage-deterring cryptographic primitive can be defined in abstract terms for a wide class of primitives we find it more instructive to present it for three main cryptographic primitives individually; we focus on public-key encryption first; definitions of leakage-deterring signatures and identification are presented in the following part. With these three examples at hand, it is relatively straightforward to derive leakage-deterring definitions for other cryptographic primitives following the same general structure (see also remarks below).

Leakage-deterring Public Key Encryption:

- **KeyGen**(1^λ): On input security parameter λ, this algorithm returns a key pair (pk, sk).
- **EnKey**(O, A): This is a protocol between two parties O (owner) and A (authority), with inputs (pk, sk, s) and (pk, s) respectively that has the objective to embed the private owner's data s into his public-key; the protocol terminates with the owner O obtaining an enhanced key pair (epk, esk) while A obtains simply the enhanced epk.
- **Enc**(epk, m): On input a message m, the user's enhanced public key epk, this algorithm returns a ciphertext c.
- **Dec**(esk, c): On input a ciphertext c and enhanced secret key esk, this algorithm returns m or fail $\perp$.
- **Rec**$^{B,\mathcal{D}}(epk, \delta)$:[5] Using access to a decryption box B and a plaintext distribution $\mathcal{D}$ (which is supposedly the one that B is suited for and is correct with probability δ), as well as input the enhanced public key epk for a certain user, this algorithm outputs s or fail $\perp$.

The definitions for other public-key primitives are similar and we present them below. They share the same basic structure in terms of the syntax of the recovering algorithm but there is some variability across primitives with respect to when this algorithm is supposed to operate. We tackle this question in the following section.

Leakage-deterring Signature Scheme: The definition of a leakage-deterring signature scheme is defined in a similar vein to the definition of leakage-deterring public-key encryption. Specifically, algorithms **KeyGen**, **EnKey** are identical. The rest are defined as follows.

- **Sign**(esk, m): On input a message m, the user's enhanced secret key esk this algorithm returns a signature σ.
- **Verify**(epk, m, σ): On input message-signature pair (m, σ), and enhanced public key epk, this algorithm returns 1 if valid, or 0 otherwise.
- **Rec**$^{\mathcal{D}}(epk, B, \delta)$: Given a signing box B and a message distribution $\mathcal{D}$, and on input an enhanced public key epk, for a certain user, this algorithm outputs the private string s belongs to this user or fail $\perp$.

[5] Having access to $\mathcal{D}$ is necessary; to see that, consider the following simple example: the box processes the input only if the message encrypted is of the form $sc||m$ for some secret string sc; otherwise it outputs $\perp$. It follows that without knowledge of $\mathcal{D}$, the box is useless. This counterexample applies to the other leakage-deterring primitives.

Leakage-deterring Identification Scheme: The case of leakage-deterring identification schemes is similar to digital signatures with the differentiation that the **Sign** and **Verify** algorithms are substituted by a protocol between two parties, the prover P and the verifier V. Normally, the owner is assumed to be the prover but the owner as it has access to the secret-key can issue implementations of the prover interactive algorithm in the identification protocol. Specifically we have the following. First **KeyGen** and **EnKey** are the same as in the previous case of leakage-deterring signature schemes, for the other two,

- **Identify**(P, V): this is a protocol between the prover P with inputs epk, esk and the verifier V on input epk that terminates with the verifier outputting 1 (accepting) or 0 (rejecting the identification). We denote the transcripts as $P(epk, esk) \leftrightarrow V(epk)$.
- **Rec**(epk, B, δ): Given an implementation B of the prover P algorithm in the identification protocol, and input the enhanced public key epk, this algorithm outputs s or fails $\perp$.

Remark 1. One can think of **EnKey** as an extension of a public-key certification operation by an authority. The owner may still utilize (pk, sk) for the primitive's operation (as in a PKI one may still use an uncertified key) but epk is the key designated for public use.

Furthermore, we note that in the **Rec** algorithm, one may distinguish several ways that the algorithm may have access to the main functionality box (which is assumed to be resettable, i.e., it does not maintain state from one query to the next). Specifically, beyond black-box access we will also consider a certain type of non-black-box access.

3.2 Correctness and Security Modelling

In this section we introduce the main security requirements for leakage-deterring cryptographic primitives. In general any leakage-deterring primitive should offer *privacy* for the owner (as long as no implementation of the primitive is leaked) and *recoverability*, i.e., that the recovering algorithm will be able to produce the private data of the owner as long as it has access to a *non-trivial* implementation of the cryptographic primitive. Finally, it is important that the introduction of the additional functionality does not disturb the standard cryptographic properties of the primitive. We examine these properties below.

Definition 1. Privacy *(of Owner's Data): For an honest owner who does not leak any non-trivial box, the privacy of its data bound in the enhanced public key should be protected. To define the property formally we introduce the following game between a challenger and an adversary $\mathcal{A}$.*

- *The challenger runs **KeyGen**$(\cdot)$ and sends to the adversary $\mathcal{A}$ the public key pk.*
- *The adversary $\mathcal{A}$ chooses two private strings s_0, s_1 and sends them to the challenger.*
- *The challenger chooses b and simulates **EnKey**$(\cdot)$ on s_b and pk, sk; it sends epk to the adversary.*
- *$\mathcal{A}$ returns his guess b' about b.*

If there is no efficient adversary $\mathcal{A}$ that can correctly guess b with non-negligible advantage, i.e., for all PPT $\mathcal{A}$, $|\Pr[b' = b] - \frac{1}{2}| \leq \epsilon$ where ϵ is a negligible function, we say the leakage-deterring cryptographic scheme achieves privacy (in the sense of indistinguishability).

Furthermore, in the above game, we may allow the adversary to observe the cryptographic functionality on a certain input distribution. If the above definition holds even in the case that the adversary has access to an oracle $\mathcal{O}(esk, \cdot)$ (that is dependent on the enhanced secret-key of the owner, e.g, decryption oracle or signing oracle w.r.t. some plaintext distribution $\mathcal{D}$) we will say that the scheme achieves privacy with respect to the secret-key oracle $\mathcal{O}(esk, \cdot)$. Note that with respect to privacy we consider both owner and authority honest. It is possible to extend the model to the case of a dishonest authority but this goes beyond the scope of our current exposition (and intended use cases).

Definition 2. Recoverability *(of Owner's Data): If a dishonest owner releases a functional box B, anyone having access to B should be able to recover the owner's private data from the enhanced public key epk. Formally, consider the following game between a challenger and an adversary $\mathcal{A}$:*

- *The adversary $\mathcal{A}$ on input 1^λ generates a key pair (sk, pk) and submits it together with the owner private data s to the challenger.*
- *The challenger acting as the authority runs **EnKey** with the adversary (playing the role of the owner) to produce the enhanced key pair (epk, esk).*
- *$\mathcal{A}$ outputs an implementation B and a distribution $\mathcal{D}$.*
- *The challenger outputs the value $s' = \textbf{\textit{Rec}}^{B,\mathcal{D}}(epk, \delta)$.*

For a given δ, we will say that the leakage-deterring cryptographic primitive satisfies black-box recoverability with respect to the class of input distributions $\mathscr{D}$, if for any efficient adversary $\mathcal{A}$ the following event in the game above happens with negligible probability.

$$(B \text{ is } \delta\text{-correct w.r.t. } \mathcal{D}) \wedge (\mathcal{D} \in \mathscr{D}) \wedge (s' \neq s)$$

Definition 3. δ-correctness: *The predicate "B is δ-correct w.r.t. $\mathcal{D}$" takes a different form depending on the cryptographic primitive and is intended to capture the fact that the box produced by the adversary should have some minimum utility which is parameterized by δ.*

Consider the case of a public-key encryption scheme $(\textbf{\textit{KeyGen}}, \textbf{\textit{Enc}}, \textbf{\textit{Dec}})$. The predicate for δ-correctness w.r.t. $\mathcal{D}$ in this case is as follows:

$$\Pr[B(\textbf{\textit{Enc}}(epk, m)) = m] \geq \delta, \text{ where } m \leftarrow \mathcal{D}$$

where the random variables $epk, \mathcal{D}, B$ are defined as in the game.

For a digital signature scheme $(\textbf{\textit{KeyGen}}, \textbf{\textit{Sign}}, \textbf{\textit{Verify}})$, the notion of δ-correctness of a box B for a message distribution $\mathcal{D}$ is defined as follows:

$$\Pr[\textbf{\textit{Verify}}(epk, m, B(m)) = 1] \geq \delta, \text{ where } m \leftarrow \mathcal{D}$$

For an identification scheme $(\textbf{\textit{KeyGen}}, \textbf{\textit{Identify}})$, we define δ-correctness of an implementation B as follows:

$$\Pr[O_V[B(epk, esk) \leftrightarrow V(epk)] = 1] \geq \delta,$$

where $O_V[B(epk, esk) \leftrightarrow V(epk)]$ denotes the output of verifier V after interacting with box B in an identification protocol.

It is worth noting that the largest class of distributions $\mathscr{D}$ we can hope recoverability to work for a leakage-deterring PKE is one that includes those distributions whose *predicting probability*[6]

[6] Denoted by $p(\mathcal{D})$ is equal to $2^{-\mathbf{H}_\infty(\mathcal{D})}$, where $\mathbf{H}_\infty(\mathcal{D}) = -\log \max_x \Pr[x \in \mathcal{D}]$ is the min-entropy of $\mathcal{D}$.

is by a non-negligible amount smaller than δ; otherwise, one can implement a decryption box by always returning the most probable sample from $\mathcal{D}$.

Also, note that for a signature box to be "non-trivial", δ should be required to be non-negligible w.r.t. distributions $\mathcal{D}$ with super-logarithmic min-entropy[7]. In the case of identification schemes there is no general input passed in the identification box and hence the choice of distribution $\mathcal{D}$ is immaterial.

We will also consider a form of the above definition where a non-black-box technique is used for recovering the owner's private data. In this case we can think of the **Rec** algorithm as a family of algorithms parameterized by the box algorithm B (as opposed to being a single algorithm with black-box access to B).

Blackbox recoverability and CCA type of privacy. Finally, we compare privacy and recoverability and observe a natural trade-off between the two properties. *Privacy* w.r.t. a secret-key oracle $\mathcal{O}(esk, \cdot)$ for a distribution $\mathcal{D}$ (i.e., when adversarial access to the cryptographic primitive is allowed for input distribution $\mathcal{D}$) can not be achieved if the leakage-deterring cryptographic primitive satisfies black-box *recoverability* w.r.t. $\mathscr{D}$, in case $\mathcal{D} \in \mathscr{D}$. This easily follows from the fact that the privacy adversary can simulate the **Rec** algorithm with the help of the secret key oracle. Thus for different primitives, we have to choose a combination of black-box recoverability with CPA type of privacy or non-black-box recoverability with CCA type of privacy.

Security Properties. We next consider how the individual security properties for leakage-deterring primitives should be amended. In general, the original security property (e.g., IND-CPA or unforgeability) should be retained with respect to the enhanced public and secret-keys even in the presence of a corrupted authority running the **EnKey** protocol.

• *IND-CPA Security* (for leakage-deterring PKE with a dishonest authority): Consider the following game between the adversary and the challenger:

- The challenger runs $\mathbf{KeyGen}(\cdot)$ to get (pk, sk) and returns pk to the adversary $\mathcal{A}$.
- The adversary $\mathcal{A}$ selects s and playing the role of the authority runs $\mathbf{EnKey}(\cdot)$ with the challenger on input pk, s.
- The adversary $\mathcal{A}$ chooses two messages m_0, m_1, and sends them to the challenger.
- The challenger randomly picks a bit $b \in \{0, 1\}$, and gives $\mathcal{A}$ the encryption of m_b under epk.
- Finally, $\mathcal{A}$ returns a guess b' about b.

Suppose there is no efficient adversary $\mathcal{A}$ that can output a correct guess about b with non-negligible advantage, i.e, $|Pr[b' = b] - \frac{1}{2}| \leq \epsilon$, where ϵ is a negligible function. In this case, we say that the leakage-deterring encryption is IND-CPA-secure (with a dishonest authority).

If we allow the adversary to ask decryption queries at anytime before outputting the guess (it can be both before and after receiving the challenge ciphertext, with the only restriction being that the challenge ciphertext cannot be queried), then we refer to this property as IND-CCA2 security.

[7] This entropy requirement for a non-trivial signing box is necessary. To see this consider the following example. The key owner prepares a list containing a polynomial number of message-signature pairs, and implements a signing box for a distribution $\mathcal{D}$ which has support of those messages only. The signing box works by simply checking whether a queried message belongs to the list and if yes, it outputs the corresponding signature. Given that such implementation can be produced from publicly collected signatures, the recoverability of the owner secret from such implementation would imply that the privacy property collapses. To exclude this trivial implementation we require that the min-entropy of the message distribution is super-polynomial (and hence this trivial implementation has a super-polynomial description).

We can also consider the security definition with an honest authority, in which case both the algorithms **KeyGen**, **EnKey** are executed by the challenger.

• *IND-CPA Security* (for leakage-deterring PKE with an honest authority) Consider the following game between an adversary and a challenger:

- The challenger runs **KeyGen**$(\cdot)$ to get (pk, sk) and returns pk to the adversary $\mathcal{A}$.
- The adversary $\mathcal{A}$ selects s and playing the role of the owner runs **EnKey**$(\cdot)$ with the challenger (as authority) to get epk.
- The adversary $\mathcal{A}$ chooses two messages m_0, m_1, and sends them to the challenger.
- The challenger randomly picks a bit $b \in \{0, 1\}$, and gives $\mathcal{A}$ the encryption of m_b under epk.
- Finally, $\mathcal{A}$ returns a guess b' about b.

Suppose there is no efficient adversary $\mathcal{A}$ that can output a correct guess about b with non-negligible advantage, i.e, $|Pr[b' = b] - \frac{1}{2}| \leq \epsilon$, where ϵ is a negligible function. In this case, we say that the leakage-deterring encryption is IND-CPA-secure (with honest authority). The only difference with standard IND-CPA security is that here we have an extra second step.

Below, we will only give unforgeability/impersonation resistance with a dishonest authority, it is straightforward to derive definitions in the setting of an honest authority by changing the roles of challenger and adversary play during **EnKey** protocol.

• *Unforgeability* (for leakage-deterring digital signatures): Consider the following game between the adversary and the challenger:

- The challenger runs **KeyGen**$(\cdot)$to get pk and returns pk to the adversary $\mathcal{A}$.
- The Adversary $\mathcal{A}$ selects s and playing the role of the authority runs **EnKey**$(\cdot)$ with the challenger as a user on input pk, s to get epk;
- $\mathcal{A}$ is allowed to ask queries to a **Sign**$(esk, \cdot)$ oracle.
- The adversary $\mathcal{A}$ outputs a message-signature pair (m^*, σ^*).

The adversary wins the game if m^* was never queried to the **Sign** oracle while it also holds that **Verify**$(epk, m^*, \sigma^*) = 1$. If for any efficient $\mathcal{A}$ the probability of winning the game is negligible, we say the leakage-deterring signature is unforgeable under adaptively chosen message attacks.

• *Impersonation Resistance* (for leakage-deterring identification schemes): Consider the following game between an adversary and a challenger:

- The challenger runs **KeyGen**$(\cdot)$ to get pk and sends pk to the adversary $\mathcal{A}$.
- The Adversary $\mathcal{A}$ selects s and playing the role of the authority, and runs the **EnKey**$(\cdot)$ protocol with the challenger on input pk, s to get epk;
- The adversary is allowed to query the challenger for **Identify**(P, V) protocol transcripts.
- The adversary $\mathcal{A}$ engages in an **Identify** protocol execution with the challenger playing the role of the prover.

The adversary wins the game if at the end the challenger playing the role of the verifier accepts the interaction with the adversary. If it holds that for any efficient adversary $\mathcal{A}$, the probability of winning the above game is negligible, we say that the leakage-deterring identification protocol is impersonation resistant against passive attacks. Impersonation resistance against active attacks can be also expressed in a standard way (following e.g., [3]).

4 Leakage-deterring Public Key Encryption

In this section, we present constructions of leakage-deterring public key encryption schemes. We start with a construction from any *additive* homomorphic encryption to demonstrate our first technique for implementing recoverability, then, we show a generic construction of IND-CPA secure leakage-deterring PKE from any IND-CPA secure encryption along with an improvement that achieves constant size ciphertext, this generic construction can be easily extended to the identity based setting as well. In section 5, we provide a general way to achieve IND-CCA2 security for all leakage-deterring encryption schemes.

4.1 IND-CPA-secure Leakage-deterring PKE from Homomorphic Encryption

Recall the trivial solution presented in the introduction (encrypting the owner's private data with its public-key). It does not work because an adversarial decryption box is able to test whether the queries fed by the recovering algorithm match the ciphertext stored in epk. A seeming fix is to query via rerandomizing the ciphertext contained in the enhanced public key. However, given that the private data are known to the attacker, the adversarial box can check for them and still reject. So in some sense to go around the problem one has to re-randomize the plaintext as well! (so that after re-randomization, the plaintexts should be distributed according to $\mathcal{D}$ but still somehow be useful for decrypting the private data). We provide a solution along these lines in this section.

Informally, an encryption algorithm $E(\cdot)$ has a homomorphic property if $E(m_1+m_2) = E(m_1)\cdot E(m_2)$ for some operations $(+,\cdot)$ over plaintexts and ciphertexts respectively. For instance, we can submit a ciphertext $c^*\cdot E(r)$ to the decryption box B, and retrieve the message in c^* from the answer by subtracting r. This method would be effective for our purpose only if B satisfies correctness w.r.t. to random distributions over the whole message space. However we would like a solution that works *even* for adversarially chosen distributions that are unknown at the time of the generation of epk. The recovering technique we introduce below achieves this goal.

First assume that we have an underlying encryption E : (**KeyGen**, **Enc**, **Dec**) that is an IND-CPA secure PKE with a homomorphic property. Specifically, we assume that for any message m and any a, b from the message space, $\mathbf{Enc}(m)^a \cdot \mathbf{Enc}(b)$ is identically distributed to $\mathbf{Enc}(am + b)$. We call the following construction Scheme-I.

- $\mathbf{KeyGen}(1^\lambda)$: Run the **KeyGen** algorithm of E, return (pk, sk).
- $\mathbf{EnKey}(O, A)$: This is a protocol between O and A with input (pk, sk, s) and (pk, s) respectively. A randomly chooses $n = |s|$ messages $\omega_i, i = 1, \ldots, n$ according to the uniform distribution over $\{0,1\}$. Then A calculates $s'_i = \omega_i \oplus s_i$, $\{c_i = E(pk, \omega_i)\}, s' = s'_1 \ldots s'_n$. The protocol terminates with O obtaining the enhanced key pair (epk, esk) where $epk = (pk, \{c_i\}, s')$, and $esk = sk$, while A gets only the enhanced public key epk.
- $\mathbf{Enc}(epk, m)$: This algorithm runs the encryption algorithm **Enc**, returning $c = \mathbf{Enc}(pk, m)$.
- $\mathbf{Dec}(esk, c)$: This algorithm runs the decryption algorithm **Dec**, returning $m = \mathbf{Dec}(sk, c)$.
- $\mathbf{Rec}^{B,\mathcal{D}}(epk, \delta)$: With access to a decryption box B and a distribution $\mathcal{D}$, that B supposedly works on with δ-correctness, the objective of this algorithm is to transform the ciphertexts $c_1, \ldots, c_n$ found in the epk to ciphertexts that look *inconspicuous* from the point of view of the box B. For each ciphertext c_i the algorithm will operate as follows. First it will calculate a sufficiently long sequence of pairs (x, y) (the exact length N of the sequence depends on the parameters of B and $\mathcal{D}$ and will be determined in our security analysis). For each pair, the

algorithm first independently samples two plaintexts m_0, m_1 according to $\mathcal{D}$. Then it calculates x, y as $x = m_1 - m_0, y = m_0$.

Let $(x_l, y_l)_{l=1,\ldots,N}$ be the pairs produced by running the above procedure N times and $m_{0,l}, m_{1,l}$ be the pair of plaintexts used as constant terms of the linear system for the l-th sample. Having calculated those, the algorithm computes $c'_{i,l} = c_i^{x_l} \cdot E(pk, y_l)$ for $l = 1, \ldots, N$, and feeds B with those ciphertexts (whose corresponding plaintexts follow $\mathcal{D}$). Let $a_1, \ldots, a_N$, be the answers of the box B where $a_l = \perp$ if the box does not provide an answer for the l-th ciphertext. Now consider the modified answer sequence to be a string over $\{0, 1, \perp\}$ defined as follows:

$$a_l^* = \begin{cases} (a_l - y_l)/x_l & a_l \in \{m_{0,l}, m_{1,l})\} \wedge x_l \neq 0 \\ \perp & \text{otherwise} \end{cases}$$

Note that $a_l^* \in \{0, 1, \perp\}$. If the majority symbol among the non-$\perp$ symbols of $\langle a_1^*, \ldots, a_N^* \rangle$ is defined the recovering algorithm calculates it as v_i and proposes it as the decryption of c_i (otherwise the algorithm fails). This procedure is repeated for all ciphertexts $c_1, \ldots, c_n$ thus forming $v = v_1 \ldots v_n$. Finally, the recovering algorithm proposes as the private data of the owner the string $s' \oplus v$ where s' is parsed from the epk.

Security Analysis: We will first sketch correctness and three security properties, i.e, *security, privacy, recoverability.* First observe that correctness is trivial, according to the correctness of the underlying encryption scheme E while IND-CPA security is also relatively obvious since the extra information exposed due to our extension are some independent values $(\omega_1 \ldots \omega_n, s)$. Now regarding the privacy property, we can see that the **EnKey** algorithm in Scheme-I is a KEM/DEM mechanism [11], using a KEM which encrypts each bit of the key with a secure encryption. Given $\{c_i\}$, the adversary is not able to predict the bit ω_i with a sufficient bias, thus every ω_i is random conditioned on the adversary's view. This proves privacy (assuming no secret-key oracle $\mathcal{O}(esk, \cdot)$ is given). Regarding recoverability we can prove it w.r.t. essentially any distribution $\mathcal{D}$. The **Rec** algorithm produces a sequence of ciphertexts with plaintexts following $\mathcal{D}$ whose correct decryption reveals the bits ω_i by a majority argument. As long as the correctness of the box B is non-negligibly larger than the collision probability of $\mathcal{D}$ (which is a minimal characteristic of "box usefulness") the recovering algorithm will produce the ω_i values with overwhelming certainty since it can do a perfect simulation of ciphertexts with $\mathcal{D}$ distributed plaintexts. The formal proofs are as follows:

Theorem 1. *Scheme-I achieves IND-CPA security (against dishonest authority) and privacy (without secret-key oracle) if the underlying PKE is IND-CPA secure. It also satisfies black-box recoverability w.r.t. any* $\delta > 0$ *and the class of distributions* $\mathscr{D} = \{\mathcal{D} \mid \exists \alpha : \alpha$ *is non-negligible and* $\mathbf{H}_\infty(\mathcal{D}) \geq \log \frac{1}{\delta - \alpha}\}$.

Proof. As explained above,the correctness and IND-CPA security is obvious. We will demonstrate the proof for *privacy* and *recoverability* as follows:

- First we examine the privacy. Let us define $E'(\omega)$ as $E(\omega_1) \ldots E(\omega_n)$ which is a bitwise encryption using E. (w.l.o.g, we assume messages are bitstrings with length n, if not, challenger can do a proper padding on them). It is straightforward to see that E' is IND-CPA secure if E is, otherwise, one can easily distinguish $E(0), E(1)$ by distinguishing $E'(m_0), E'(m_1)$, where m_0, m_1 are identical except at one bit.

Suppose $\mathcal{C}$ is the challenger of E', $\mathcal{A}$ is the adversary who can break the privacy of scheme-I. We will build a simulator algorithm $\mathcal{S}$ which uses $\mathcal{A}$ as an oracle to break the IND-CPA security of E'.

$\mathcal{S}$ first forwards the public key *pk* from $\mathcal{C}$ to $\mathcal{A}$, he then chooses two random messages m_0, m_1, sends them to $\mathcal{C}$ and gets the challenge c, also he receives s_0, s_1 from $\mathcal{A}$. $\mathcal{S}$ randomly select b_0, b_1, computes $s' = s_{b_0} \oplus m_{b_1}$, and sends (pk, c, s') to $\mathcal{A}$ as *epk*. If $\mathcal{A}$ returns b_0 correctly, $\mathcal{S}$ outputs b_1 as his answer, otherwise he outputs $1 - b_1$.

It is easy to see that the simulator's advantage of breaking the semantic security of E' is at least half of the advantage that $\mathcal{A}$ has to break the privacy (denoted by Δ). If $c = E'(m_{1-b_1})$, then s' perfectly hides s_{b_1} since $s_{b_0} \oplus m_{b_1}$ now is independent with *epk*, $\mathcal{A}$ can only guess b_0 correctly with probability $1/2$. If $c = E'(m_{b_1})$, the *epk* is in a valid form and this time $\mathcal{A}$ will have advantage Δ. Thus, the simulator's advantage is $\frac{1}{2}(\frac{1}{2} + \frac{1}{2} + \Delta) - \frac{1}{2} = \frac{\Delta}{2}$.

- Next, we examine the recoverability. It is obvious that the **KeyGen** and **EnKey** procedures can be easily simulated. Note that the way we sample (x, y) in the **Rec** algorithm, every query is an encryption of a message independently sampled from $\mathcal{D}$ (the only exception is that $\mathcal{D}$ almost always outputs only one message but in this case, any box becomes "trivial"). Thus, the recovering query is identically distributed as normal decryption queries and B would have δ-correctness for every recovering query!

Now we analyze the number of repetitions needed(in terms of an asymptotic function of the security parameter λ) to guarantee we are almost certain that s will be returned in the **Rec** algorithm.

First, we call the experiment a useful one if $m_0 \neq m_1$, otherwise, we will always record a $\perp$ for this query. The probability of having one useful experiment after sampling N_0 pairs of (m_0, m_1) will be $1 - \mathbf{Col}(\mathcal{D})^{N_0}$, where $\mathbf{Col}(\mathcal{D})$ is the collision probability of distribution $\mathcal{D}$ which denotes the probability of sampling a same element from two independent trials. Observe that $\mathbf{Col}(\mathcal{D}) < 1 - \gamma$, for some non-negligible γ, if $\mathcal{D}$ is not a trivial distribution which has probability almost 1 over one single element. If we repeat $N_0 = O(\log^2 \lambda)$ times sampling, we will get a useful experiment with probability almost 1.

Further, for one useful query, only two answers x, $x+y$ (m_0, m_1 respectively) would be considered possibly correct, all other answers are simply ignored (denoted by $\perp$). The probability of getting at least one correct answer after repeating N_1 times useful experiments is $1 - (1 - \delta)^{N_1}$, if $N_1 = O(\log^2 \lambda)$, this probability is almost 1(with negligibly small difference).

Recall that $\delta = p(\mathcal{D}) + \alpha$, where $p(\mathcal{D}) = 2^{-\mathbf{H}_\infty(\mathcal{D})}$ denotes the predicting probability of $\mathcal{D}$. In one useful experiment, the probability of returning an incorrect but non-$\perp$ answer is at most $p(\mathcal{D})$, since this happens only when the box returns m_b while the correct answer is m_{1-b}, for $b = 0, 1$ (recall these two messages are independently sampled from $\mathcal{D}$).

Now we focus on non-$\perp$ answers only, the probability of an incorrect answer appearing among the non-$\perp$ answers is at most $q = \frac{p(\mathcal{D})}{2p(\mathcal{D}) + \alpha}$ (This can be argued as follows: suppose $p(\mathcal{D}) = p + t$,where p is the probability of returning an incorrect but non-$\perp$ answer in one query and $t \geq 0$; then, the probability of obtaining an incorrect but non-$\perp$ answer among all non-$\perp$ answers is at most $\frac{p}{\delta + p} \leq \frac{p+t}{\delta + p + t} = q$). Suppose X is the random variable that denotes the number of appearances of incorrect but non-$\perp$ answers after collecting N_2 non-$\perp$ answers, μ denotes the expectation of X which is no bigger than $N_2 q$. The probability that correct answers do not constitute the majority is $\Pr[X \geq \frac{N_2}{2}]$. Using the upper tail of the Chernoff bound:$\Pr[X > (1+\beta)\mu)] \leq e^{\frac{-\beta^2 \mu}{3}}$, we can verify that this probability is bounded by $\exp(-N_2\alpha^2/(24p(\mathcal{D})^2 + 6p(\mathcal{D})\alpha))$.

So if we collect more than $N_2 = O(\alpha^{-2}\log^2 \lambda)$ non-$\perp$ symbols, the majority will be occupied by the correct answers with probability almost 1 (negligibly small difference), and hence we can recover the bit.

Combining these procedures, if we repeat the recovering procedure $N_0 \times N_1 \times N_2 = O(\alpha^{-2}\log^6 \lambda)$ times for each bit of s we will successfully recover s with probability almost 1. □

Remark 2. Note that the restriction for the class of distribution is optimal in the sense that otherwise, the box would be "trivial".

4.2 Generic IND-CPA-secure Leakage-deterring PKE with Honest Authority

In this section, we relax further the requirements of leakage-deterring PKE to minimal by constructing a scheme based on *any* secure PKE. We will only consider IND-CPA security with honest authority in this section and we will show how to go beyond this and achieve security against dishonest leakage-deterringties (and actually IND-CCA2) in the next section.

Linear-Size Construction. To make the exposition more accessible we present first a less efficient construction (with linear size ciphertexts in the length of hidden information); then we show our main generic construction which is constant size. Consider a semantically secure public key encryption E. The main idea of the construction is as follows. For each bit of private data there is a pair of public keys, and the owner has only one of the secret keys. The ambiguity of which secret key the owner has offers the opportunity for the recovering algorithm to work. We call this construction Scheme-II, details are as follows:

- **KeyGen**(1^λ): This algorithm generates $n = |s|$ key pairs $(pk_1, sk_1), \ldots, (pk_n, sk_n)$.
- **EnKey**(O, A): (O, A) have inputs $(pk_1, \ldots, pk_n, s, sk_1, \ldots, sk_n)$, and $(pk_1, \ldots, pk_n, s)$ respectively, where $s \in \{0,1\}^n$. A randomly generates $r \in \{0,1\}^n$ which we call indicating string, and n new random public keys $pk'_1, \ldots, pk'_n$. The enhanced public key epk is n pairs of public keys $(pk_1^0, pk_1^1), \ldots, (pk_n^0, pk_n^1)$, together with $s' = r \oplus s$, where for $i = 1, \ldots, n$, $pk_i^{r_i} = pk_i, pk_i^{1-r_i} = pk'_i$, and the enhanced secret key is $esk = (sk, r)$, where $sk = (sk_1, \ldots, sk_n)$.
- **Enc**(epk, m):This algorithm first randomly picks $m_1, \ldots, m_{n-1}$, and computes $m_n = m - \sum_{i=1}^{n-1} m_i$ (wlog we assume that additive secret-sharing works over the plaintext space). It outputs the ciphertext $c = [(c_1^0, c_1^1), \ldots, (c_n^0, c_n^1)]$, where $c_i^0 = \mathbf{Enc}(pk_i^0, m_i), c_i^1 = \mathbf{Enc}(pk_i^1, m_i)$.
- **Dec**(esk, c): To decrypt ciphertext c, this algorithm chooses from c the ciphertexts corresponding to the indicating string r, and returns $m = \sum_{i=1}^n \mathbf{Dec}(sk_i, c_i^{r_i})$.
- $\mathbf{Rec}^{B,\mathcal{D}}(epk, \delta)$: With access to a decryption box B and a plaintext distribution $\mathcal{D}$ for which the box supposedly works with δ-correctness, the algorithm recovers each bit s_i of s by repeating the following procedure N times (the exact value of N will be specified in the analysis):

  It first samples m, m' independently, according to $\mathcal{D}$, randomly chooses $m_1, \ldots, m_{i-1}, m_{i+1}, \ldots, m_n$, and computes $m_i^0 = m - \sum_{j \neq i} m_j$, and $m_i^1 = m' - \sum_{j \neq i} m_j$.

  Then it feeds B with $[(c_1^0, c_1^1), \ldots, (c_n^0, c_n^1)]$, where for all $j \neq i$, c_j^0, c_j^1 encrypts the same message m_j while $c_i^0 = \mathbf{Enc}(pk_i^0, m_i^0), c_i^1 = \mathbf{Enc}(pk_i^1, m_i^1)$.

  The algorithm records a 0, if the response from the box is m, 1 if the response is m', and $\perp$ in any other case including the case $m = m'$. For each i, the algorithm will propose r_i to be the majority of the recorded non-$\perp$ values (the algorithm fails if majority is not well-defined).

  The above procedure is repeated for all $i \in \{1, \ldots, n\}$ to form a string r, and finally, the algorithm outputs $s = s' \oplus r$, where s' is parsed from epk.

Security Analysis. We first sketch the security properties of Scheme-II. Let us call an encryption using a single pair of keys as a "unit building block". It is not hard to see that IND-CPA security of the unit building block implies the IND-CPA of scheme-II (assuming authority is honest and taking into the security of additive secret-sharing). Regarding privacy, observe that s is perfectly hidden within *epk* as a one-time pad ciphertext. Finally, regarding recoverability w.r.t any distribution $\mathcal{D}$, the recovering algorithm can attempt to query different encrypted messages in any single unit location. Due to the secret-sharing throughout all pairs, any box (even partly successful) has to include a key for each coordinate. Due to these facts, the recovering algorithm can detect which secret key does the owner possess at each location something that leads to the calculation of the indicating string and hence the recover of the private data. A detailed analysis is as follows:

Theorem 2. *Scheme-II achieves* privacy*(without secret-key oracle access). It also satisfies IND-CPA security (with honest authority) and black-box recoverability w.r.t. any* $\delta > 0$ *and the class of distributions* $\mathscr{D} = \{\mathcal{D} \mid \exists\alpha : \alpha$ *is non-negligible and* $\mathbf{H}_\infty(\mathcal{D}) \geq \log\frac{1}{\delta-\alpha}\}$*, if the underlying encryption is a regular IND-CPA secure PKE.*

Proof. The privacy property is trivially achieved because the owner private data is hidden with a one-time pad. Also it is easy to see correctness, owner has one key for each pair and can successfully decrypt one of the ciphertext in each pair. We prove the IND-CPA security with honest authority, and recoverability as follows.

- We first prove the IND-CPA security of a simplified version of Scheme-II, (recall that we named it "unit building block") in which there is only one pair of public keys (pk_1^0, pk_1^1), the secret information s will be one bit only, and during encryption, one directly encrypts message m under both public key (without doing secret sharing of the message).

  Claim. The unit building block E^* of Scheme-II is IND-CPA secure if the underlying encryption E is IND-CPA secure.

  Proof. of the claim: For any two message m_0, m_1, we define the pair $E(pk_1^0, m_i), E(pk_1^1, m_j)$ as E_{ij}. We would show that both E_{00}, E_{11} are indistinguishable from E_{01}, thus E_{00} is indistinguishable from E_{11}.
  Suppose $\mathcal{A}$ can distinguish E_{00} from E_{01} with advantage Δ, one can use $\mathcal{A}$ to break the semantic security of E as follows:
  Assume $\mathcal{C}$ is the challenger of E, when the simulator receives pk from $\mathcal{C}$ and a private bit s from $\mathcal{A}$, he randomly selects another public key pk', and sends $\mathcal{A}$ $(pk', pk, s \oplus 1)$ as epk. Then, the simulator forwards m_0, m_1 from $\mathcal{A}$ to $\mathcal{C}$. Whenever the simulator gets a challenge ciphertext c from $\mathcal{C}$, he computes $c' = E(pk', m_0)$, and sends (c', c) to $\mathcal{A}$ as his challenge. The simulator will outputs $\mathcal{A}$'s guess directly to $\mathcal{C}$ (suppose that $\mathcal{A}$ outputs 0 for E_{00}, and 1 for E_{01}).
  It is easy to see that, (c', c) is exactly E_{00} if $c = E(m_0)$, or E_{01} if $c = E(m_1)$, so simulator breaks semantic security of E with the same advantage Δ as $\mathcal{A}$ distinguishes E_{00}, E_{01}.
  Similarly, we can prove E_{01}, E_{11} are indistinguishable. Thus $E_{00} = E^*(m_0)$, and $E_{11} = E^*(m_1)$ are indistinguishable. □

  With the above claim, we will reduce the security of the unit building block E^* to the security of Scheme-II. Suppose $\mathcal{C}$ is the challenger of E^*, $\mathcal{A}$ is the adversary who successfully breaks the semantic security of Scheme-II with advantage Δ, we will build a simulator to break the security of E^* using $\mathcal{A}$.

After receiving a private string s from $\mathcal{A}$, the simulator forwards the first bit s_1 to $\mathcal{C}$. After receiving (pk_1^0, pk_1^1, b_1) from $\mathcal{C}$, the simulator generates another $n-1$ pairs of public keys and secret keys $\{(pk_i^b, sk_i^b)\}_{i=2,\dots,n-1;b=0,1}$, and sends $[(pk_1^0, pk_1^1), \dots, (pk_n^0, pk_n^1), s']$ to $\mathcal{A}$ as epk, where $s'_i = b_i$, and $b_2 \dots b_|s|$ are random bits.

After getting m_0, m_1 from $\mathcal{A}$, the simulator randomly selects $n-1$ messages $m_2, \dots, m_n$, computes $m = m_0 - \sum_{i=2}^n m_i$, $m' = m_1 - \sum_{i=2}^n m_i$, and forwards m, m' to $\mathcal{C}$. When the simulator receives a challenge $c = (c_1^0, c_1^1)$ from $\mathcal{C}$, then, for all $i \in \{2, \dots, n\}$, $b \in \{0,1\}$, he computes $\{c_i^b = \mathbf{Enc}(pk_i^b, m_b)\}$, and sends $\mathcal{A}$ his challenge $[(c_1^0, c_1^1), \dots, (c_n^0, c_n^1)]$.

Simulator directly outputs $\mathcal{A}$'s guess as his own guess to $\mathcal{C}$ assumes that $\mathcal{A}$ outputs 0 for m_0 and 1 for m_1. It is easy to see that the simulator's advantage of breaking the IND-CPA security of Scheme-II is exactly $\varDelta$. □

- Next we prove the recoverability holds. First we argue that adversarial box can not distinguish recovering queries (for some pair, two ciphertext contain different messages) from normal ciphertext. Essentially, we will argue that the box will have similar performance in both cases. Note that when a box gets a normal ciphertext and the message is sampled according to $\mathcal{D}$, it will return a correct answer with probability at least δ due to the δ-correctness, and in such cases, the box will return an incorrect but non-$\perp$ answer with probability at most $p(\mathcal{D})$ because of the fact that no information about the incorrect answer is contained in the decryption query.

Claim. Suppose the underlying encryption scheme E satisfies ϵ-semantic security (i.e, no efficient adversary can distinguish $E(m_0)$ and $E(m_1)$ with advantage ϵ for any pair of different messages (m_0, m_1)), then any box with δ-correctness created by an efficient adversary will return a correct answer with probability at least $\delta - 2\epsilon$ and will return a incorrect but non-$\perp$ answer with probability at most $p(\mathcal{D}) + 2\epsilon$, when fed with a recovering query.

Proof. of the claim: We prove for probability of returning a correct answer only, for the case of incorrect but non-$\perp$, it follows straightforwardly. First note that, in a recovering query, the messages are also sampled independently from $\mathcal{D}$. Suppose there is an adversary $\mathcal{A}$ producing a box $\mathcal{B}$ with δ-correctness when queried with normal ciphertext and with $(\delta - \varDelta)$-correctness for a non-negligible $\varDelta$ when queried with recovering ciphertext, one can use $\mathcal{A}$ to distinguish two ciphertexts $E(m_0), E(m_1)$ in the IND-CPA game of E as follows:

First, the simulator receives public key pk from the challenger $\mathcal{C}$ of E, also it receives n public keys $(pk_1, \dots, pk_n)$ and s from the adversary $\mathcal{A}$.

The simulator randomly chooses a position $i \in \{1, \dots, n\}$, a random bitstring $r \in \{0,1\}^n$, and chooses $n-1$ random public keys $pk'_1, \dots, pk'_{i-1}, pk'_{i+1}, \dots, pk'_n$. The simulator sends $\mathcal{A}$ the enhanced public key $epk = [(pk_1^0, pk_1^1), \dots, (pk_n^0, pk_n^1), r \oplus s]$, where for all $j \neq i$, $pk_j^{r_j} = pk_j, pk_j^{1-r_j} = pk'_j$, and $pk_i^{r_i} = pk_i, pk_i^{1-r_i} = pk$.

$\mathcal{A}$ then produces a decryption box B and a distribution $\mathcal{D}$. To create a challenge ciphertext for B, the simulator first randomly chooses two messages m, m' according to $\mathcal{D}$, then he randomly chooses $n-1$ messages $m_1, \dots, m_{i-1}, m_{i+1}, \dots, m_n$. Further, he computes $m_i^0 = m - \sum_{j \neq i} m_j$, and $m_i^1 = m' - \sum_{j \neq i} m_j$, then sends m_i^0, m_i^1 to $\mathcal{C}$, and receives a challenge c from $\mathcal{C}$. The simulator then randomly selects m_i^b from m_i^0, m_i^1, and feeds B with ciphertext $[(c_1^0, c_1^1), \dots, (c_n^0, c_n^1)]$, where for all $j \neq i$, both c_j^0, c_j^1 contain the same message m_j, and $c_i^{r_i} = E(pk_i, m_i^b), c_i^{1-r_i} = c$.

If B returns the right message (if $b = 0$, it is m, if $b = 1$, it is m'), then the simulator returns b as his guess, otherwise, he returns $1 - b$.

It is obvious that if the simulator guesses correctly(m_i^b is the message $\mathcal{C}$ encrypts in c), the challenge for $\mathcal{A}$ is a normal ciphertext, otherwise, it is a recovering query. Thus, simulator will break the semantic security of E(return a correct b) with probability $\frac{1}{2}\delta + \frac{1}{2}(1-\delta+\Delta) = \frac{1}{2} + \frac{\Delta}{2}$, thus $\Delta \leq 2\epsilon$.
Similarly, we can show the probability of returning an incorrect but non-$\perp$ answer will be 2ϵ close to $p(\mathcal{D})$, too. □

The rest parts are very similar to the analysis of theorem 1. For each bit of the owner data, in each query there are only two potentially correct answers (m, m' as we used in the **Rec** algorithm in scheme-II). Since for each query, we have the probability of returning a correct answer is almost δ, and the probability of returning an incorrect but non-$\perp$ answer is almost $p(\mathcal{D})$, we only need to proceed to estimate the number of repetitions needed. The analysis is also composed of three main steps, we first estimate the number of samples needed for getting a useful experiment, and then estimate the number of repetition needed for collecting at least one correct answer, then we estimate the number of correct answers needs to be collected to ensure that the majority will be the correct answer. With the above claim, the performance of the adversarial box is negligibly close to that in scheme-I, thus the number of repetitions needed is also $O(\alpha^{-2}\log^6\lambda)$. For details of the calculation, we refer to the proof of theorem 1. □

Main Generic Construction. In the previous construction, the sender splits the message into n pieces. This makes the ciphertext size (number of ciphertext units) linear in the length of the owner's private data. We now improve the generic construction to achieve a ciphertext size $O(\log\frac{1}{\delta})$ by using an error correcting code to create the indicating string, where δ is a specified minimum correct decryption probability that is assumed to be constant and is a parameter of the construction. We call this construction Scheme-III.

- **KeyGen**(1^λ): This algorithm generates n' key pairs $(pk_1, sk_1), \ldots, (pk_{n'}, sk_{n'})$, where n' will be specified below.
- **EnKey**(O, A): (O, A) have inputs $(pk_1, \ldots, pk_{n'}, s, sk_1, \ldots, sk_{n'})$, and $(pk_1, \ldots, pk_{n'}, s)$ respectively where $s \in \{0,1\}^n$; the parameter n' is determined based on n according to an error correcting code ECC:$\{0,1\}^n \to \{0,1\}^{n'}$ (e.g., from [21]) that corrects up to $\frac{n'}{5}$ errors. A randomly generates $\tilde{r} \in \{0,1\}^n$, computes the indicating string $r = ECC(\tilde{r})$. Also, A selects n' random public keys $pk'_1, \ldots, pk'_{n'}$. The protocol terminates with O obtaining (epk, esk) and A obtaining epk, where epk contains $(pk_1^0, pk_1^1), \ldots, (pk_{n'}^0, pk_{n'}^1)$, together with $s' = \tilde{r} \oplus s$, and esk contains $(sk_1, \ldots, sk_{n'}, r)$. For $i = 1, \ldots, n'$, $pk_i^{r_i} = pk_i, pk_i^{1-r_i} = pk'_i$, and $sk_i^{r_i} = sk_i$.
- **Enc**(epk, m): To encrypt a message m, the algorithm first chooses a random subset $S \subseteq \{1, \ldots, n'\}$ with size $t = 5\ln\frac{4}{\delta}$. Then it randomly picks $m_1, \ldots, m_{t-1}$, and computes $m_t = m - \sum_{i=1}^{t-1} m_i$. The ciphertext $c = [S, (c_1^0, c_1^1), \ldots, (c_t^0, c_t^1)]$, where $c_i^b = \mathbf{Enc}(pk_{S_i}^b, m_i)$ for $b \in \{0,1\}$.
- **Dec**(esk, c): To decrypt ciphertext c, this algorithm chooses from c the ciphertexts corresponding to the indicating string r projected on S, and returns $m = \sum_{i=1}^{t}\mathbf{Dec}(sk_{S_i}, c_i^{r_{S_i}})$.
- **Rec**$^{B,\mathcal{D}}(epk, \delta)$: With access to a decryption box B and a plaintext distribution $\mathcal{D}$, the algorithm recovers each bit s_i of s by repeating the following procedure N times (the exact number will be specified in the analysis):
  It first randomly selects a subset $S \subseteq \{1, \ldots, n'\}$ with size t.
  If $i \in S$, and i is the k-th element of S, the algorithm randomly chooses m, m' independently according to $\mathcal{D}$ as well as random values $m_1, \ldots, m_{k-1}, m_{k+1}, \ldots, m_t$. Then, it computes $m_k^0 =$

$m - \sum_{j \neq k} m_j$, and $m_k^1 = m' - \sum_{j \neq i} m_j$. It feeds B with $[(c_1^0, c_1^1), \ldots, (c_t^0, c_t^1)]$ where, for all $j \neq k$, the pair c_j^0, c_j^1 encrypts the same message m_j using pk_i^0, pk_i^1 respectively, while $c_k^b = \mathbf{Enc}(pk_i^b, m_k^b)$ for $b \in \{0, 1\}$.
If $i \notin S$, the algorithm proceeds by performing a regular encryption of a plaintext from $\mathcal{D}$.
If $i \in S$ and the response of the decryption box is m, the algorithm records 0; if $i \in S$ and the response is m', this algorithm records a 1; Otherwise (in any other case including $i \notin S$ or $m = m'$), it records $\perp$. For each i the majority of the non-$\perp$ recorded values is proposed as the value of r_i. If no majority is defined, a random bit is produced as r_i.
The above procedure is repeated for all $i \in \{1, \ldots, n'\}$, and a string r is formed. The decoding algorithm of ECC is now executed on r to obtain $\tilde{r}$. The algorithm terminates by returning $s = s' \oplus \tilde{r}$, where s' is parsed from epk.

Security Analysis: The IND-CPA and privacy properties are essentially the same as in scheme II. We only discuss recoverability which is significantly more complex. The intuition is that because of the error correcting code, the **Rec** algorithm would work as long as a linear fraction of bits of r can be recovered. As we will prove in detail that, suppose that q is the number of positions among the m for which our recoverability procedure fails. We will show that the probability of correct decryption will become roughly smaller than $e^{-tq/n'} = \delta^{5q/n'}$. From this we derive that any decryption box operating with probability at least δ (as postulated) can make our algorithm fail in at most $n'/5$ of the n' secret keys which is sufficient for correct decoding. The full analysis is presented as follows:

Theorem 3. *Scheme-III parameterized by any $\delta > 0$, achieves* privacy *(without secret-key oracle access). Further, if the underlying public key encryption scheme is IND-CPA secure, it satisfies IND-CPA security (with honest authority) and black-box recoverability w.r.t. the class of distributions $\mathscr{D}_\delta = \{\mathcal{D} \mid \mathbf{H}_\infty(\mathcal{D}) \geq \log|s| + \log\frac{1}{\delta} - c\}$, where c is a constant (depending on the ECC) and $|s|$ the length of the embedded private information.*

Proof. All properties except recoverability are the same as Scheme-II. We only analyze recoverability below. The major difference in this analysis is that we need to examine the maximum number of positions for which the recoverability algorithm can fail while still maintaining that the decryption box B has more than δ-correctness.

First we set the number of experiments executed for each index i, denoted by N to be calculated as in the proof of theorem 1 with the following modifications: First, we use a probability α_0 in place of α, which is defined as $(\delta - \kappa)/n'^2$ where $\kappa = p(\mathcal{D}) = 2^{-\mathbf{H}_\infty(\mathcal{D})}$ denoting the predicting probability. Second, we need to repeat more times due to the fact that randomly selecting S will not always contain i to be a "useful query" for recovering. But sampling S randomly instead of always containing some i aims at producing the recovering queries indistinguishable from normal ciphertext. Suppose now the target of the recovering algorithm is the i-th bit, in one selection, the probability $\Pr[i \in S]$ is $C_{n'-1}^{t-1}/C_{n'}^{t} = t/n'$. From the lower tail of Chernoff bound, the probability of selecting N_3 times without hitting i once is smaller than $e^{-(\frac{N_3 t}{2n'} - 1)}$. After randomly sample $4n'/t$ times, one will be sure that one of the subsets will contain i, and one useful query is created. In total, The recovering procedure repeats for $N_0 \times N_1 \times N_2 \times N_3$ times, where N_0, N_1, N_2 are as in the analysis of theorem 1 where α is substituted with α_0 (note that if we can reset the box across experiments the ciphertexts for which $i \notin S$ can be just omitted).

The main challenge in the proof of the theorem is the fact that the box B might behave differently depending on i and thus force us to err in a number of locations i. We will prove that we can bound this number and hence our error-correction layer will be sufficient for recovering the hidden information in the epk. Let $\delta_i = \Pr[B \text{ decrypts correctly} \mid i \in S]$. We divide the indices $i \in \{1, \ldots, n'\}$ in two sets, Bad and Good, according to the rule $i \in \mathsf{Good}$ if and only if $\delta_i \geq \kappa + \alpha_0$. Based on our choice of N, if $i \in \mathsf{Good}$ the recoverability will return the proper bit in the i-th coordinate with overwhelming probability. In order to upper bound the size of Bad consider the following. Let D be the event of correct decryption. We have that,

$$\Pr[\mathsf{D}] = \Pr[\mathsf{D} \mid S \cap \mathsf{Bad} = \emptyset] \cdot \Pr[S \cap \mathsf{Bad} = \emptyset] + \Pr[\mathsf{D} \mid S \cap \mathsf{Bad} \neq \emptyset] \cdot \Pr[S \cap \mathsf{Bad} \neq \emptyset]$$

Regarding $\Pr[S \cap \mathsf{Bad} = \emptyset]$ observe that if $k = |\mathsf{Bad}|$, the probability is bounded by $p(k,t) = C^t_{n'-k}/C^t_{n'} = \prod_{i=0}^{t-1}(1 - \frac{k}{n'-i}) \leq (1 - \frac{k}{n'})^t$. From inequality $e^x \geq 1 + x$, we can get $p(k,t) \leq e^{-kt/n'}$. Regarding $\Pr[\mathsf{D} \mid S \cap \mathsf{Bad} \neq \emptyset]$ note that it is bounded by $\sum_{i \in \mathsf{Bad}} \delta_i \leq n'(\kappa + \alpha_0)$ (This bound follows directly from the fact that $\Pr[F | \cup_{i=1}^n A_i] \leq \sum_{i=1}^n \Pr[F|A_i]$, for any event F, A_i). We now derive the following,

$$\delta \leq \Pr[\mathsf{D}] \leq e^{-tk/n'} + n'(\kappa + \alpha_0)$$

From which we obtain the upper bound $k \leq \frac{n'}{t} \cdot \ln(\delta - n'(\kappa + \alpha_0))^{-1}$. Now observe that due to the condition for the min-entropy, we derive a bound on $\kappa \leq 2^c \delta/|s| = c'\delta/|s|$, for some constant c'. From the choice of α_0 we can prove that $\delta - n'(\kappa + \alpha_0) \geq \delta/4$ as long as c is selected appropriately (taking into account the error-correcting rate which is constant). We plug this condition and the fact that $t = 5\ln(4\delta^{-1})$ we conclude that $k \leq n' \ln(4\delta^{-1})/5\ln(4\delta^{-1}) = n'/5$.

The rest is similar to the proof of theorem 2, whenever the i-th secret key is contained in the decryption box, as argued in the analysis of theorem 2, adversary will have similar performance when fed with a recovering query, and from the estimation of number of repetitions, if one repeat $O(\alpha_0^{-2} m \log^6 \lambda) = O(\alpha^{-2} \lambda^5 \log^6 \lambda)$ times(given that the length of the codeword $n' = O(\lambda)$, and $\alpha = \delta - p(\mathcal{D})$), one can recover r_i correctly as long as $i \in \mathsf{Good}$ with overwhelming probability. So the number of errors in recovering r is at most k, while the ECC is able to correct up to $\frac{n'}{5}$ errors, and $\frac{n'}{5} \geq k$, thus r_0 will be recovered correctly with overwhelming probability and hence also s. □

Remark 3. In this construction, the ciphertext size is parameterized by the correctness δ which is influenced by the min-entropy of the distribution the box works on. Essentially, if the desired min-entropy that the scheme should be leakage deterring against gets smaller, then the ciphertext size should increase.

Remark 4. The generic construction can also be easily adapted to the identity based setting. To accomplish this, observe that we can replace pk_i^b with $(ID|i, b)$, instantiate the IBE system with a hierarchical IBE scheme and apply the above generic construction to derive an identity based leakage-deterring encryption. We omit the detailed construction of identity based leakage-deterring encryption, but for the sake of completeness, we present the definitions needed in the appendix.

5 Generic CCA2 Secure Construction with Dishonest Authority

In this section, we introduce a general method to construct an IND-CCA2 secure leakage-deterring encryption with dishonest authority from any leakage-deterring PKE which satisfies IND-CPA security with honest authority, and any IND-CCA2 secure standard PKE. The main idea is to compose these two encryptions to form a nested encryption with the outer layer encryption to be the IND-CCA2 secure one. *Recoverability* could be maintained because the **Rec** algorithm can run the **Rec** algorithm of the inner leakage-deterring encryption to collect queries, and encrypt them using the outer layer public key to form its own recovering queries.

Construction Suppose E_1 is any IND-CPA secure leakage-deterring PKE (with an honest authority), and E_2 is any IND-CCA2 secure PKE. We call the following construction Scheme-IV.

- $\mathbf{KeyGen}(1^\lambda)$: This algorithm first executes the **KeyGen** algorithm of both E_1, E_2, and return $(pk_1, sk_1), (pk_2, sk_2)$.
- $\mathbf{EnKey}(O, A)$: This is a protocol between O, A with inputs (pk_1, sk_1, s) and (pk_1, s) respectively; it proceeds by executing the **EnKey** protocol of E_1 to get (epk_1, esk_1) first, and this protocol terminates with O obtaining enhanced key pair (epk, esk), and A obtaining epk only, where $(epk, esk) = ((epk_1, pk_2), (esk_1, sk_2))$.
- $\mathbf{Enc}(epk, m)$: To encrypt a message m, this algorithm runs the encryption algorithms of both of E_1, E_2, and returns the ciphertext as $c = \mathbf{Enc}(pk_2, \mathbf{Enc}(epk_1, m))$
- $\mathbf{Dec}(esk, c)$: To decrypt a ciphertext c, this algorithm runs the decryption algorithms of both E_1, E_2 and returns $m = \mathbf{Dec}(esk_1, \mathbf{Dec}(sk_2, c))$.
- $\mathbf{Rec}^{B,\mathcal{D}}(epk, \delta)$: With access to a decryption box B and a plaintext distribution $\mathcal{D}$, this algorithm calls the **Rec** algorithm R_1 of E_1. For each query c of R_1, this algorithm feeds B with $\mathbf{Enc}(pk_2, c)$. It then passes the responses of the box to R_1 and returns whatever R_1 returns.

Security Analysis *Correctness* is obvious. Privacy, recoverability and IND-CCA2 security follow easily from the security of the outer layer encryption and the properties of the inner-layer encryption. The details are presented below:

Theorem 4. *Scheme-IV is IND-CCA2 secure with dishonest authority if E_2 is a IND-CCA2 secure PKE, and achieves same privacy and recoverability as the underlying leakage-deterring PKE E_1.*

Proof. We will reduce the IND-CCA2 security of the outer layer encryption E_2 to the security of scheme-IV. Suppose the challenger of E_2 is $\mathcal{C}$, and the adversary of scheme-II is $\mathcal{A}$. We can build a simulator algorithm to attack the security of E_2 by using $\mathcal{A}$, the adversary on scheme-IV.

After receiving the public key pk from $\mathcal{C}$, the simulator randomly chooses a key pair (pk_1, sk_1) for E_1, the inner layer encryption. Also, he randomly chooses a secret string s. The simulator then sends $\mathcal{A}$ (pk_1, pk, s) and receives the enhanced key pairs $(epk, esk) = ((epk_1, pk_2), (esk_1, sk_2))$.

When $\mathcal{A}$ asks a decryption query c_i, the simulator forwards it to $\mathcal{C}$ and gets an answer of $c_1 = Dec(sk, c_i)$, and then he uses esk_1 to decrypt the inner layer ciphertext, and returns $\mathcal{A}$ the answer. It is simple to see that the decryption query is answered correctly.

After receiving m_0, m_1 from $\mathcal{A}$, the simulator encrypts them using the enhanced public key of the inner layer encryption E_1, sends $(m_0', m_1') = (E_1(epk_1, m_0), E_1(epk_1, m_1))$ to $\mathcal{C}$ and forwards the challenge c directly to $\mathcal{A}$.

The simulator continues to answer $\mathcal{A}$'s decryption queries as before. And sends $\mathcal{A}$'s guess b directly to $\mathcal{C}$ as his guess.

Since the challenge is exactly the encryption of m_0 or m_1, so the simulator would have the same advantage for $\mathcal{A}$ to distinguish encryption of m_0', m_1' using scheme-IV.

As the **EnKey** protocol is the same as in E_1, thus privacy of scheme-IV is the same as that of E_1; with respect to recoverability, from the **Rec** algorithm we can see that the ciphertext queries in scheme-IV and that in E_1 have a one-to-one correspondence, the algorithm will return a correct value as long as the **Rec** algorithm in E_1 can return one. □

Remark 5. An alternative way to achieve IND-CCA2 security is to instantiate each PKE scheme in the generic construction of section 4.2 with a lossy trapdoor function, and apply the Peikert-Waters paradigm [36] to convert this IND-CPA secure scheme into an IND-CCA2 secure scheme by utilizing a strong one time signature. We will use only a unit building block as an example to demonstrate the idea, and the full construction can be derived straightforwardly. Using the notation from [36], $F_{ltdf}(s, x)$ denotes a lossy trapdoor function F evaluated at input x using public key s; $G_{abo}(s', vk, x)$ denotes an all-but-one trapdoor function evaluated at input x using public key s' and branch vk. For details of these two primitives, we refer to [36].

The public keys will be $(s_0, s_1, s'_0, s'_1, h_0, h_1)$, and the secret key is t_b for a bit b, where t_b is the trapdoor corresponding to s_b, and h_0, h_1 are random universal hash functions [9]. For **Enc**, the algorithm first generates a random key pair (sk, vk) for a strong one-time signature scheme, randomly selects x_0, x_1, and output the ciphertext as $c = (vk, [F_{ltdf}(s_0, x_0), G_{abo}(s'_0, vk, x_0), h_0(x_0) \oplus m], [F_{ltdf}(s_1, x_1), G_{abo}(s'_1, vk, x_1), h_1(x_1) \oplus m], \sigma)$, where σ is a signature signed using sk on all the other components in the ciphertext. The **Dec** algorithm just selects the corresponding ciphertext, and inverts $F_{ltdf}(s_b, x_b)$ using t_b to get x_b, and checks whether $G_{abo}(s'_b, vk, x_b)$ is well-formed and retrieves m by $h_b(x_b) \oplus m \oplus h_b(x_b)$. We can argue the IND-CCA2 security in a similar way to the analysis in [36], with the minor difference in the indistinguishability between lossy keys and injective keys where we use a pair of lossy trapdoor functions instead of one; similarly, for the hidden lossy branch property we use a pair of all-but-one trapdoor functions instead of one. We omit the detailed proof here. This alternative construction will yield a more efficient IND-CCA2 scheme when applied over the generic construction compared to the one we presented above. Furthermore, we can use the one-time signature paradigm [8] in the setting of identity based leakage-deterring encryption as well.

To deal with the transformation from IND-CCA2 security with a honest authority to IND-CCA2 security with a dishonest authority without using the nested encryption mechanism, we can have the public keys in one of the pairs to be the same as the one generated by the user, for which the authority does not know the secret key. The encryption algorithm will always use this public key. It is easy to see that this "special" public key guarantees the security in a model with a dishonest authority. Note that the **EnKey** protocol and the **Rec** algorithm in the above alternative construction are the same as those in the generic constructions and privacy and recoverability will not be affected.

6 Leakage-Deterring Signature & Identification

In this section, we design leakage-deterring signatures and identification schemes. The main idea is that we treat any functional box as an unforgeability or impersonation adversary, and then take advantage of "witness extractability" used in the security arguments of the underlying primitive to extract the secret-key which will unlock the private data. To achieve this type of extractability we apply rewinding, and hence this means that a certain level of non-black-box access to the adversarial implementation is needed that was unnecessary before. Specifically, in the case of digital signatures we assume the recoverability algorithm can "hook" the hash function calls of the adversarial implementation while in the case of identification schemes the assumption is that the adversarial identification can be rewound to a previous state (something that is true for software adversarial implementations but not necessarily true in the case of a hardware adversarial implementation). We stress that our leakage-deterring constructions do not employ any additional intractability assumptions beyond the ones used in the underlying primitives.

6.1 Leakage-Deterring Signature In the Random Oracle Model

We construct a leakage-deterring signature scheme based on a class of Σ-protocol-based signature schemes as in [37]. The security proofs of these signatures rely on the fact that if the adversary can forge one signature, then he could also forge another correlated signature for the same message with the same random tape but a different random oracle. Using these two forgeries that are correlated, one can extract the secret key of the owner.

Our construction of leakage-deterring signature is based on two independent digital signatures instances $\mathbf{Sig}_0$ and $\mathbf{Sig}_1$ that are unforgeable under adaptively chosen message attacks. Further, $\mathbf{Sig}_1$ is required to be unforgeable in the random oracle(RO) model following [37]; specifically, the signature has the form of $(m, \sigma_1, h, \sigma_2)$ as in [37], and satisfies $h = H(m, \sigma_1)$, and σ_2 only depends on m, σ_1, h, where H is a RO. We call the following construction Scheme-V.

- $\mathbf{KeyGen}(1^\lambda)$: This algorithm executes the **KeyGen** algorithm of $\mathbf{Sig}_0$, and returns the key pair (pk_0, sk_0).
- $\mathbf{EnKey}(O, A)$: This protocol is executed between O, A with inputs (pk_0, sk_0, s), and (pk_0, s) respectively. A runs **KeyGen** algorithm of $\mathbf{Sig}_1$ to generate a key pair (pk_1, sk_1). The protocol terminates with O obtaining (epk, esk), and A obtaining epk, where $epk = (pk_0, pk_1, H(sk_1) \oplus s)$, and $esk = (sk_0, sk_1)$.
- $\mathbf{Sign}(esk, m)$: On input a message m, this algorithm returns the signature $\sigma = (\sigma_0, \sigma_1)$, where $\sigma_0 = \mathbf{Sign_0}(sk_0, m)$, and $\sigma_1 = \mathbf{Sign_1}(sk_1, m) = (\sigma_1^1, h_1, \sigma_1^2)$.
- $\mathbf{Verify}(epk, m, \sigma)$: On input a message-signature pair (m, σ) and enhanced public key $epk = (pk, pk')$, this algorithm returns 1 if both of the two signatures are valid, 0 otherwise.
- $\mathbf{Rec}^{\mathcal{D}}(epk, B, \delta)$: The recovering algorithm follows the security proof argument of [37]: Whenever the box B asks a random oracle query (suppose total number of such queries is bounded by q), the algorithm selects a uniform response from the range of the random oracle and feeds it to the box; it also maintains a table of all these queries. The recovering algorithm samples a message m according to $\mathcal{D}$ and simulates the box B on m. When the box outputs a valid signature σ_0, σ_1, where $\sigma_1 = (\sigma_1^1, h_1, \sigma_1^2)$, algorithm checks the table and identifies the index i of the first query from B on (m, σ_1^1). Then, it rewinds B to the state prior to the i-th query, and continues the simulation picking new random query responses.
  The above procedure is repeated until the box outputs another valid signature (σ_0', σ_1') on the same message m, where $\sigma_1' = (\sigma_1^1, h_1', \sigma_2')$, and also the index i that (m, σ_1^1) was queried is the same for both σ_1 and σ_1'.
  Now the algorithm can extract the second secret key sk_1 from $(m, \sigma_1^1, h_1, \sigma_1^2), (m, \sigma_1^1, h, \sigma_2')$ using the Σ protocol properties of the scheme that define $\mathbf{Sig}_1$. The recovery of s follows immediately.

Security Analysis: We first give some brief intuition about the three properties. It is easy to see that unforgeability against adaptively chosen message attacks can be derived from the property of $\mathbf{Sig}_0$ as any forgery will imply also a forgery of $\mathbf{Sig}_0$. Note that signing queries are easy to simulate because the simulator has the secret key for $\mathbf{Sig}_1$, and can ask signing queries to the challenger for $\mathbf{Sig}_0$. *Privacy* w.r.t. a secret-key oracle for any distribution can be achieved because any successful privacy attacker will have to eventually query sk_1 to the random oracle hence violating the unforgeability of $\mathbf{Sig}_1$. Note that *recoverability* cannot violate *privacy* w.r.t. an arbitrary secret key oracle, since it is achieved now via a non-black-box technique. It uses the fact that rewinding the signing box and controlling the random coins in an execution, one can always find a pair of signatures that reveal the secret key, something that yields the private data. Note that we consider only the "non-trivial" signing box works for super polynomially many messages, otherwise, it can be produced without containing the secret key. Details of the analysis are as follows:

Theorem 5. *Scheme-V is unforgeable under adaptively chosen message attacks if the underlying signature* $\boldsymbol{Sig}_0$ *is unforgeable under adaptively chosen message attacks. It achieves privacy w.r.t. any secret key(signing) oracle if* $\boldsymbol{Sign}_1$ *is unforgeable under adaptively chosen message attacks. Also, Scheme-V achieves non-black-box recoverability w.r.t any non-negligible* δ *and any message distribution* $\mathcal{D}$ *with super logarithmic min-entropy.*

Proof. Correctness is relatively obvious, the validity of a secret key for a signature scheme can be easily verified, owner would check the validity when receiving the additional key pairs from the authority. We will only demonstrate the security properties as follows:

- We first examine the unforgeability. Suppose $\mathcal{A}$ is an adversary who breaks the unforgeability of Scheme-V, and $\mathcal{C}$ is the challenger in the security game of $\mathbf{Sig}_0$.
  After receiving s, pk_0 from $\mathcal{A}, \mathcal{C}$ respectively, the simulator generates a random key pair (sk_1, pk_1), and runs the **EnKey** protocol with $\mathcal{A}$ and terminates with $\mathcal{A}$ obtaining epk, and the challenger obtaining epk, sk_1, where $epk = (pk_0, pk_1, s')$ and $s' = H(sk_1) \oplus s$.
  Whenever $\mathcal{A}$ asks a signing query on m, the simulator asks such signing query to $\mathcal{C}$ to get σ_0, signs with sk_1 to get σ_1, and returns $\mathcal{A}$ with (σ_0, σ_1).
  If $\mathcal{A}$ outputs a forgery (σ_0^*, σ_1^*) on m^* which is never asked for signing query, then the simulator outputs m^*, σ_0^* as his forgery to $\mathcal{C}$.
- Next, we will show privacy (with secret key oracle access). We will reduce the unforgeability of $\mathbf{Sign}_1$to the privacy property.
  First, the simulator $\mathcal{S}$ gets pk_1 from the unforgeability challenger $\mathcal{C}$ and creates another key pair (pk, sk), and sends pk to the adversary $\mathcal{A}$. After receiving s_0, s_1 from $\mathcal{A}$, $\mathcal{S}$ randomly chooses r and a bit b, and sends $(pk, pk', r \oplus s_b)$ as epk.
  $\mathcal{S}$ can always answer the signing queries since he can sign with sk to get the first half of a signature, and asks $\mathcal{C}$ for the second half.
  Note that only when one queries an input a to the random oracle, it is possible for him to predict even one bit of the output $H(a)$. If the adversary can predict at least one bit of s, that means he can predict at least one bit of r, thus he has to ask a random oracle query about sk (which is the input of the random oracle for r) at some point. The simulator simply collects all the random oracle queries of $\mathcal{A}$, and checks whether it is the secret key corresponding to pk_1, whenever sk_1 is found, the simulator stops and outputs a signature using sk_1 on a message which $\mathcal{C}$ is never asked for a signing query.
- Further, we examine the recoverability. According to the notations in the general forking lemma [2], we can see that $acc = \Pr[J \geq 1] = \Pr[F_1 \wedge F_2] = \Pr[F_2]\Pr[F_1|F_2]$, where F_1 denotes the event that B makes a call to the hash function for a query m, and F_2 denotes the event that B outputs a valid signature for m. First, all messages submitted to be signed by

Rec are sampled from $\mathcal{D}$, thus B will output a valid signature for each query with probability at least δ; hence, we have $\Pr[F_2] \geq \delta$. Second, an efficient B can only store polynomially many hash values or message-signature pairs. This is seen as follows: let L be the list of messages m for which it holds that $B(m)$ outputs a signature with non-negligible probability without querying the hash function on that message. We claim that the size of this list L is is bounded by Q=poly(λ). Indeed, if the list is super-polynomial one can execute B repeatedly and obtain all valid signatures that form list L. The signatures determine a list of values of size Q from the table of the hash function that B never queries to its hash oracle. It follows that this corresponds to a superpolynomial amount of information that is encoded in the description of B. By a standard information-theoretic argument one can show that this is impossible as it suggests that the adversary that constructs the box can be used to encode a super-polynomial amount of information in a polynomial size description (from which the encoded information can still be recovered).

Now recall that $H_\infty(\mathcal{D})$ is $\omega(\log \lambda)$, thus for a random sample m from $\mathcal{D}$ the probability that m is included in the list L is at most $Q/2^{\omega(\log \lambda)}$. It follows that $\Pr[F_1|F_2] \geq 1 - Q/2^{\omega(\log \lambda)} \geq \alpha$ for some non-negligible α.

Combining the above fact with $\Pr[F_2] \geq \delta$, we obtain that $acc \geq \alpha\delta$. Following the general forking lemma, we can see that the probability of finding a successful pair of signatures on a same message in one rewinding is no smaller than $(\alpha\delta)^2/q - \epsilon$, where ϵ is a negligible function so that $1/\epsilon$ is the size of the range of random oracle, and q is the number of random oracle queries. After repeating the rewinding procedure for some polynomial in λ number of times, where λ is the security parameter, one can find a successful pair with probability $1 - \epsilon'$ where ϵ' is a negligible function. With such a successful pair, one can extract the secret key sk_1 as the knowledge extractor does in the Σ-protocol (or as in the reduction in [37]), and outputs $s = s' \oplus H(sk_1)$. □

6.2 Leakage-deterring Identification

We will construct a leakage-deterring identification scheme by using a similar approach as in the signature case. However here we will show our construction secure in the *standard model*, and thus we need a novel method to embed the owner private data into the enhanced public key. In fact we will need no additional assumption beyond the one employed for the underlying scheme.

Our construction of a leakage-deterring identification scheme is based on the class of identification schemes which are derived from zero-knowledge proofs of knowledge protocols that can be parallelly composed. We utilize the fact that given access to the code of any box that implements the identification functionality, one can rewind the box and implement the knowledge extractor assured to exist due to the soundness property of the zero-knowledge proof (this idea was used before to obtain the related notion of non-transferability of credentials in [7, 34]). We call the following construction Scheme-VI and is based on a parameter t that we specify below.

- **KeyGen**(1^λ): This algorithm executes the **KeyGen** algorithm of the underlying identification scheme, and returns the key pair (pk, sk).
- **EnKey**(O, A): This is a protocol executed between O, A with inputs (pk, sk, s), and (pk, s) respectively. A runs the **KeyGen** algorithm to generate t new key pairs $(pk_1, sk_1), \ldots, (pk_t, sk_t)$, and further, A calculates $s' = r \oplus s$, where $r = Ext(sk_1|| \ldots ||sk_t, \rho)$ and Ext is a strong randomness extractor (see below for implementation remarks) while ρ is the random seed.

The protocol terminates with O obtaining (epk, esk), and A obtaining epk, where $epk = (pk, pk_1, \ldots, pk_t, s', \rho)$, and $esk = (sk, sk_1, \ldots, sk_t)$.

- **Identify**(P, V): This protocol is executed between P, V with inputs (epk, esk), and epk respectively. The protocol is the parallel composition of the $t + 1$ underlying identification schemes. The protocol terminates with V outputting 1 if he accepts the proof of knowledge of all secret keys, and 0 otherwise.
- **Rec**(epk, B): The algorithm given B, runs the knowledge extractor algorithm for the parallel composition of the t schemes until all the secret keys of $\{sk_1, \ldots, sk_t\}$ are recovered. Then it applies the extractor on ρ and returns $s = s' \oplus Ext(sk_1|| \ldots ||sk_t, \rho)$

Security Analysis: We first sketch the security properties. *Recoverability* is essentially the same as the *recoverability* of Scheme-V. *Impersonation resistance* is also similar to the *unforgeability* property of Scheme-V; this property mainly relies on the fact that nothing related to the original secret key of the owner sk is added to the epk, therefore the security of identification using the original (pk, sk) can be reduced to the impersonation resistance of Scheme-VI. Regarding *privacy*, according to impersonation resistance, after seeing a polynomial number of transcripts of interaction between P, V, there is still unpredicatability on the secret key, (otherwise, one can impersonate by eavesdropping) then applying the strong extractor one can get pure randomness out of the secret-keys, and thus the owner data is hidden computationally. Details are given as follows:

Theorem 6. *Scheme-VI is impersonation resistant if the underlying identification schemes are impersonation resistant under parallel composition. It achieves privacy w.r.t. the secret key oracle that plays the role of the prover and performs with the adversary the identification protocol. Also, Scheme-VI achieves non-black-box recoverability w.r.t any non-negligible δ.*

Proof. Correctness follows straightforwardly from the correctness of the underlying identification scheme. Also, as explained above, *impersonation resistance* and *recoverability* are very similar to the unforgeability and recoverability of Scheme-V, thus we only focus on the proof for *privacy* here.

We first show that every secret key sk_i still has sufficient conditional unpredicatability given that the adversary adaptively makes queries for transcripts during identification.

Claim. The conditional unpredicatability of each sk_i is at least $\omega(\log \lambda)$, where λ is the security parameter, if the underlying identification scheme is impersonation resistant against a passive adversary.

Proof. of the claim: After seeing adaptively queried transcripts, if there exists an adversary $\mathcal{A}$ who can predict one of the secret keys sk_i with non-negligible probability (conditional unpredicatability is asymptotically smaller than $\omega(\log \lambda)$), then one can build a simulator which breaks the impersonation resistance of the underlying identification protocol. In more details:

Suppose $\mathcal{C}$ is the challenger in the impersonation resistance game, when the simulator receivers pk from $\mathcal{C}$, he generates $t-1$ key pairs $(pk_1, sk_1), \ldots, (pk_{i-1}, sk_{i-1}), (pk_{i+1}, sk_{i+1}, \ldots, (pk_t, sk_t)$, and sends $\mathcal{A}$ $(pk_1, \ldots, pk_{i-1}, pk, pk_{i+1}, \ldots, pk_t)$.

The simulator asks the identification transcripts for pk from $\mathcal{C}$, and for other public keys, the transcripts can be perfectly simulated since the simulator knows the secret keys.

When $\mathcal{A}$ outputs a guess for sk_i, the simulator uses this sk_i to execute **Identify** as a prover with $\mathcal{C}$ as a verifier.

It is obvious that $\mathcal{C}$ will accept the identification attempt of the simulator on behalf of the prover with the same probability that $\mathcal{A}$ correctly outputs sk_i. □

Under the above claim, the unpredictability of all the sk_i's concatenated together is sufficiently large, therefore after applying the randomness extractor, the output r is uniform, thus the owner's private data is hidden due to the fact that the embedding works as a one-time pad. □

Remark 6. The strong randomness extractor Ext should work on any source with sufficient conditional unpredictability along the lines of [24]. For instance, we can use the extractor derived from the Goldreich-Levin hard-core predicate [16]. Intuitively, one can think of the view (protocol transcripts adaptively queried) of the adversary as the output of a one way function on input $\{sk_i\}$. Using this, [16] implies an extractor of $\log \lambda$ bits per instance and thus t should be as long as $|s|/\log \lambda$.

Remark 7. If one is willing to allow additional intractability assumptions, a more compact construction for leakage-deterring signature (in the RO model) and leakage-deterring identification is also possible[8]. The construction would utilize two key pairs $(pk_0, sk_0), (pk_1, sk_1)$ and the secret information will be embedded as $E(pk_1, s)$, thus only sk_1 will be used by the recoverability algorithm. Observe now that privacy will rely on the security of the encryption scheme (and thus may require assumptions going beyond the underlying identification scheme). Furthermore reusing the same key for signing and decrypting may not always be secure and some specialized systems would need to be employed, for instance cf. [22].

7 Leakage-Deterring Cryptography Applications

In this section we explore in more detail practical scenarios where leakage-deterring cryptosystems can be used to provide novel solutions to security problems related to sharing and transferring cryptographic functions.

Let us start with a more detailed motivating scenario: consider a user that maintains all her e-mail encrypted on a mailserver. The user is approached by someone wishing to buy all e-mails sent by the e-mail address x@y in the past, present and future. Using a regular encryption, the user may release to the attacker an implementation of her decryption function that works only if the plaintext is an e-mail sent by x@y (and rejects all other input). If the user does not care about the secrecy of the e-mails from x@y, she has no strong reason to be deterred from releasing the implementation (all her other messages can still be relatively safe assuming the implementation is sufficiently obfuscated or delivered in hardware). Using our encryption however, she is deterred: if she releases the above implementation (even in the form of a hardware token) an adverse action is guaranteed to take place (via the recoverability algorithm): her private information will be revealed. Obviously, a determined secret-key owner can always decrypt and release the plaintexts corresponding to those e-mails individually. But this has to be done one by one, at a potentially high cost. In this scenario, leakage-deterring public-key encryption ensures there is no way to optimize this operation: if one wants to provide access to his decryption he has to do it on a "per-ciphertext" basis. Within a PKI this enforces secret-key owners to practice more responsible secret-key management.

Recall privacy w.r.t. to secret key oracles (that would be the CCA flavor of our privacy property) and recoverability can not be achieved simultaneously in the general case: the two properties are mutually exclusive. Thus, one needs to choose a proper trade-off if he wants to implement leakage-deterring public key schemes. Regarding PKE, our objective in this work is to maximize the scope

[8] We thank an anonymous reviewer for pointing this out.

of recoverability: it should work for all (even partially functional) implementations; this makes our primitive most useful from a self-enforcement perspective and necessitates the restrictions we have made in terms of the privacy property. If the user wishes the private information to remain hidden, she should provide no access to her secret-key. In the case of signature/identification schemes the situation is more tricky since by nature of the functionality, the user is expected to release signatures/identification transcripts publicly (which in some cases they may even be adaptively selected). Thus, we must compromise and weaken our recoverability property in some way. We resolved this by adopting a non-black-box recoverability algorithm. As expected, if the implementation becomes "obfuscated" then recoverability would be infeasible. We believe the tradeoffs we utilized are natural for the primitives studied, but of course different tradeoffs can be possible between privacy and recoverability, and we leave them as future work.

Depending on different application scenarios, we can embed various types of private owner information to deter the leakage of a cryptographic functionality. We list three relevant scenarios below.

Self-enforcement. In the context of self-enforcement the owner of the cryptographic functionality has embedded into her enhanced public-key some private information that she normally prefers to keep secret. This can be e.g., her credit-card number or similar piece of private information as suggested by Dwork, Lotspiech and Naor [13] that introduced self-enforcement (in a different context - see related work in the introduction). In this way, when using our leakage-deterring primitives, if the owner releases any implementation of the cryptographic functionality, any recipient of the implementation will become privy to the hidden information. This property "self-enforces" the owner to keep the functionality to herself and solves the problem of how to deter the sharing of software or hardware devices that implement cryptographic functionalities.

All-or-nothing sharing of cryptographic functionalities. In this scenario, the owner is obliged to embed the secret key of the cryptographic primitive itself into the enhanced public-key (in practice this can be done e.g., by a trusted key generator algorithm which will be running the embedding algorithm that is executed by the authority in our model). Using our techniques this means that any working implementation of the cryptographic functionality would leak the whole secret-key. In this sense, the cryptographic functionality becomes "unobfuscatable", any program that partially implements it, say for some types of inputs, can be transformed to a program that implements it perfectly. leakage-deterring primitives used in this way suggest a type of *all-or-nothing* property for cryptographic keys: owners of a cryptographic functionality cannot partially share it, they either have to keep it to themselves or share it fully. In practice, one can expect that this is also a type of self-enforcing mechanism: either all information about the cryptographic key will be leaked or none.

Anonymity revocation from implementations. In this setting, the owner of the cryptographic functionality operates it under a pseudonym (i.e., the enhanced public-key is certified but without openly identifying the owner). However, the embedded information is ensured by the authority to be either the owner's real identity or an identity credential that the owner prefers to hide. In this setting, using our methodology, if any working implementation of the functionality is confiscated, it will be possible to use the recovering algorithm to reveal the hidden identity credential. This in turn, ensures some level of accountability: the owner remains pseudonymous as long as he does not share the cryptographic functionality but can be identified in case any (even partially working) implementation is leaked.

8 Conclusions and Open Problems

We introduced the notion of leakage-deterring cryptosystems. Our schemes have the property that whenever an owner releases an (even partially) "functional" box for others to use instead of herself, anyone who has access to the box can recover some private information that is embedded into the public-key of the owner. We defined the security properties of these primitives and we provided several constructions for public key encryption, signatures, identification.

Since this is the first step in the formal investigation of such primitives, several interesting open questions remain. A natural is how to combine the notion with traitor tracing and other multi-user oriented cryptosystems; we explored this relation in [27]. Another direction is with respect to CCA2 security: our construction can potentially be optimized for efficiency and avoid the nesting of two encryptions. A third direction is to see to what extent it is feasible to construct leakage-deterring signatures and identification with black-box recoverability in the standard model or more generally explore the tradeoff between recoverability and privacy in a comprehensive fashion. Last but not least, it would be desirable to see how the trust to the authority can be reduced (and e.g., obviate the need for the authority to know the secret information).

Acknowledgement: The authors thank the anonymous reviewers of this paper for their valuable comments.

References

1. M. Bellare and O. Goldreich. On defining proofs of knowledge. In *CRYPTO*, pages 390–420, 1992.
2. M. Bellare and G. Neven. Multi-signatures in the plain public-key model and a general forking lemma. In *ACM CCS*, pages 390–399, 2006.
3. M. Bellare and A. Palacio. Gq and schnorr identification schemes: Proofs of security against impersonation under active and concurrent attacks. In *CRYPTO*, pages 162–177, 2002.
4. M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM CCS*, pages 62–73, 1993.
5. D. Boneh and M. Franklin. Identity-based encryption from the weil pairing. *SIAM J. Comput.*, 32(3):586–615, 2003.
6. J. Camenisch and A. Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In *EUROCRYPT*, pages 93–118, 2001.
7. R. Canetti, M. Charikar, S. R. Sridhar Rajagopalan, A. Sahai, and A. Tomkins. Non-transferrable anonymous credentials. US Patent 7,222,362., 2008.
8. R. Canetti, S. Halevi, and J. Katz. Chosen-ciphertext security from identity-based encryption. In *EUROCRYPT*, pages 207–222, 2004.
9. L. Carter and M. Wegman. Universal classes of hash functions. *Journal of Computer and System Sciences*, 18(2):143–154, 1979.
10. B. Chor, A. Fiat, and M. Naor. Tracing traitors. In *CRYPTO*, pages 257–270, 1994.
11. R. Cramer and V. Shoup. Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. *SIAM J. Comput.*, 33(1):167–226, 2004.
12. I. Damgard. On σ- protocols. In *http://www.daimi.au.dk/ ivan/Sigma.pdf*, 2010.
13. C. Dwork, J. B. Lotspiech, and M. Naor. Digital signets: Self-enforcing protection of digital information (preliminary version). In *STOC*, pages 489–498, 1996.
14. A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO*, pages 186–194, 1986.
15. C. Gentry and A. Silverberg. Hierarchical id-based cryptography. In *ASIACRYPT*, pages 548–566, 2002.
16. O. Goldreich and L. A. Levin. A hard-core predicate for all one-way functions. In *STOC*, pages 25–32, 1989.
17. O. Goldreich, B. Pfitzmann, and R. L. Rivest. Self-delegation with controlled propagation-or-what if you lose your laptop. In *CRYPTO*, pages 153–168, 1998.

18. P. Golle, F. McSherry, and I. Mironov. Data collection with self-enforcing privacy. In *ACM CCS*, pages 69–78, 2006.
19. V. Goyal. Reducing trust in the pkg in identity based cryptosystems. In *CRYPTO*, pages 430–447, 2007.
20. V. Goyal, S. Lu, A. Sahai, and B. Waters. Black-box accountable authority identity-based encryption. In *ACM CCS*, pages 427–436, 2008.
21. V. Guruswami and P. Indyk. Expander-based constructions of efficiently decodable codes. In *FOCS*, pages 658–667, 2001.
22. S. Haber and B. Pinkas. Securely combining public-key cryptosystems. In *ACM CCS*, pages 215–224, 2001.
23. M. J. Hinek, S. Jiang, R. Safavi-Naini, and S. F. Shahandashti. Attribute-based encryption without key cloning. *Int. J. Appl. Cryptol.*, 2(3):250–270, Feb. 2012.
24. C.-Y. Hsiao, C.-J. Lu, and L. Reyzin. Conditional computational entropy, or toward separating pseudoentropy from compressibility. In *EUROCRYPT*, pages 169–186, 2007.
25. M. Jakobsson, A. Juels, and P. Q. Nguyen. Proprietary certificates. In *CT-RSA*, pages 164–181, 2002.
26. A. Kiayias and Q. Tang. Making any identity based encryption accountable, efficiently. In *Proceedings of the 20th European Symposium on Research in Computer Security*, ESORICS '15, pages 326–346, 2015.
27. A. Kiayias and Q. Tang. Traitor deterring schemes: Using bitcoin as collateral for digital content. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, pages 231–242, 2015.
28. A. Kiayias and M. Yung. Breaking and repairing asymmetric public-key traitor tracing. In *Digital Rights Management Workshop*, pages 32–50, 2002.
29. A. Kiayias and M. Yung. Traitor tracing with constant transmission rate. In *EUROCRYPT*, pages 450–465, 2002.
30. H. Komaki, Y. Watanabe, G. Hanaoka, and H. Imai. Efficient asymmetric self-enforcement scheme with public traceability. In *Public Key Cryptography*, pages 225–239, 2001.
31. B. Libert and D. Vergnaud. Towards black-box accountable authority ibe with short ciphertexts and private keys. In *Public Key Cryptography*, pages 235–255, 2009.
32. H. Lipmaa, G. Wang, and F. Bao. Designated verifier signature schemes: Attacks, new security notions and a new construction. In *ICALP*, pages 459–471, 2005.
33. L.Lamport. Constructing digital signatures from a one-way function. In *Technical Reprot SRI-CSL-98*, 1979.
34. A. Lysyanskaya, R. L. Rivest, A. Sahai, and S. Wolf. Pseudonym systems. In *International Workshop on Selected Areas in Cryptography*, SAC '99, pages 184–199, 2000.
35. D. Naccache, A. Shamir, and J. P. Stern. How to copyright a function? In *Public Key Cryptography*, pages 188–196, 1999.
36. C. Peikert and B. Waters. Lossy trapdoor functions and their applications. In *STOC*, pages 187–196, 2008.
37. D. Pointcheval and J. Stern. Security arguments for digital signatures and blind signatures. *J. Cryptology*, 13(3):361–396, 2000.
38. A. Sahai and H. Seyalioglu. Fully secure accountable-authority identity-based encryption. In *Public Key Cryptography*, pages 296–316, 2011.
39. T. Sander and A. Ta-Shma. Auditable, anonymous electronic cash extended abstract. In *CRYPTO*, pages 555–572, 1999.
40. S. F. Shahandashti and R. Safavi-Naini. Construction of universal designated-verifier signatures and identity-based signatures from standard signatures. In *Public Key Cryptography*, pages 121–140, 2008.
41. A. Shamir. Identity-based cryptosystems and signature schemes. In *CRYPTO*, pages 47–53, 1984.

A Identity Based Leakage-deterring Encryption

Definition of IBE: [41, 5] IBE is a public key encryption mechanism that any string can be used as a public key, while the corresponding private key of a string can be extracted by an authority. It is composed of four algorithms:

- **Setup**(1^λ): On input a security parameter, this algorithm generates a master key pair (mpk, msk).
- **Extract**(ID, msk): On input an identity ID, and the master secret key msk, this algorithm outputs a key pair (pk_{ID}, sk_{ID}) for this identity ;

- **Enc**(ID, m, mpk): On input a message m and an identity ID, this algorithm returns a ciphertext c;
- **Dec**(sk_{ID}, c): On input a ciphertext c and a secret key sk_{ID}, this algorithm returns the message m.

Regarding to 2-layer hierarchical IBE (HIBE) [15], there is an extra **Derive** algorithm, that given the secret key of any identity $ID \in \{0,1\}^t$, it derives the secret key for any identity $ID||v \in \{0,1\}^{2t}$ which has ID as a prefix. For more details about the definition of HIBE schemes, we refer to [15].

ID-based leakage-deterring Encryption: In identity based setting, the **KeyGen** algorithm and **EnKey** algorithm can be merged into the **EnExtract** algorithm. Details are as follows:

- **Setup**(1^λ) This algorithm generates a master key pair (mpk, msk).
- **EnExtract**(O, A) This is a protocol between a user O and a key generator A with inputs (ID, s), and $[ID, (mpk, msk), s]$ respectively. The protocol terminates with both parties obtaining the enhanced key pair $(epk, esk) = ((pk_{ID}, s'), sk_{ID})$ for the input identity.
- **Enc**(ID, m, mpk) On inputs message m and identity ID's enhanced public key, this algorithm returns a ciphertext c;
- **Dec**(sk_{ID}, c) On input ciphertext c and secret key sk_{ID}, this algorithm returns the message m.
- **Rec**$^{B,\mathcal{D}}(epk, \delta)$ Given oracle access to a decryption box B, and a message distribution $\mathcal{D}$, which B has δ-correctness on, and with input the enhanced public key for a certain identity, this algorithm returns s or $\perp$.

ID-IND-CPA Security (for ID-based leakage-deterring encryption): The IND-CPA security for identity based leakage-deterring encryption is slightly different with that of leakage-deterring PKE which is defined in 3.2, as the authority is always assumed to be honest in this setting. Details are described in the following game between the adversary and the challenger:

- The challenger runs the **Setup** algorithm and returns mpk to the adversary.
- The adversary adaptively chooses a sequence of identities $ID_1, \ldots, ID_q$, and $s_1, \ldots, s_q$ as the private data for each identity, and then interacts with the challenger in the **EnExtract** protocol to get enhanced public and secret keys for these identities.
- The adversary chooses two messages m_0, m_1, a target identity ID which was not queried for secret key before, and a string s^*, and sends them to the challenger.
- The challenger randomly choose a bit b, and sends to the adversary $c = \textbf{Enc}(ID, m_b, mpk)$ and epk where epk is the enhanced public-key of the user that corresponds to ID with private information s^*.
- The adversary outputs his guess b'.

We say an ID-based leakage-deterring encryption is ID-IND-CPA secure if $\Pr[b' = b] \leq 1/2 + \epsilon$ in the above game, where ϵ is a negligible function. If the adversary is required to claim the target identity and the string in the beginning (before the secret key queries), we call it selective-ID-IND-CPA secure. Furthermore, if decryption queries are allowed for the target identity (with the exclusion of the challenge ciphertext) before the adversary outputs his guess, we call it ID-IND-CCA2 secure.

We note that the enhanced public-key epk will not be actually used for encryption; this is in order to be compliant with the id-based nature of the primitive. However, the epk still carries the private information of the user and is assumed to be a public-value that is available to anyone. The only time that this value is relevant is in the operation of the recoverability algorithm that may happen only after a leakage incident takes place.

As in remark 4 at the end of section 4.2, following the generic construction of leakage-deterring PKE, we can similarly construct an identity based leakage-deterring encryption scheme by generating exponentially many secret keys for one identity, and using the "indicating string" to select corresponding keys as in the generic construction. We omit the detailed description of the construction here.