LRCRYPT: Leakage-Resilient Cryptographic System (Design and Implementation)

Conference Paper *in* Lecture Notes in Computer Science · March 2017

DOI: 10.1007/978-3-319-56549-1_20

Ruoqing Zhang
The University of Hong Kong
10 PUBLICATIONS **31** CITATIONS

LRCRYPT: Leakage-Resilient Cryptographic System (Design and Implementation)

Xiaoqi Yu[1], Nairen Cao[2], Gongxian Zeng[1], Ruoqing Zhang[1], and Siu-Ming Yiu[1](✉)

[1] Department of Computer Science,
The University of Hong Kong, Hong Kong, China
rpyxqi@gmail.com, naksi@connect.hku.hk,
rockyzhanghku@gmail.com, smyiu@cs.hku.hk

[2] Department of Computer Science,
Georgetown University, Washington, D.C., USA
nc645@georgetown.edu

Abstract. Due to the advancement of side-channel attacks, leakage-resilient cryptography has attracted a lot of attention in recent years. Many fruitful results have been proposed by researchers. Most, if not all, of these results are theoretical in nature. Not much has been done to realize these schemes for practical use. In this work, we design and provide a leakage-resilient cryptographic system $\mathcal{LRCRYPT}$ with programming interfaces for users to build leakage-resilient cryptographic applications. $\mathcal{LRCRYPT}$ consists of a few fundamental building blocks that perform leakage-resilient public-key encryption, leakage-resilient signature, and leakage-resilient secret-key encryption, which can also be extended to many existing leakage resilience cryptographic primitives. We have conducted both a security analysis and a performance evaluation on $\mathcal{LRCRYPT}$. To our knowledge, $\mathcal{LRCRYPT}$ is the first to work in this domain.

1 Introduction

Leakage-resilient Cryptography. Traditionally, cryptographic primitives are considered as mathematical objects with a well-defined interface between the primitives and the user/adversary. The primitives are provably secure in common sense: if an efficient adversary can break a primitive π with significant probability, there exists a simulator that can solve a known hard problem P. By the assumption that P is hard, it implies that there is no efficient adversary for π.

However, the actual interactions between the primitive and the adversary may be influenced by its implementations and physical devices in real-world applications. In such cases, extra information about the primitive may be leaked to the adversary, which accumulatively may lead to the attack on the primitive that is proved secure. Main types of these "side-channel" attacks are elaborated below. The literature dealing with the circumstance that can capture adversaries who execute the attacks through the implementations of a primitive is called Leakage-resilient Cryptography.

D. Choi and S. Guilley (Eds.): WISA 2016, LNCS 10144, pp. 233–244, 2017.
DOI: 10.1007/978-3-319-56549-1_20

Types of Side Channel Attacks. In practice, there are numerous attacks by side-channels, *i.e.* time attacks, power dissipation, cold-boot attacks [15–17]. Followings are some main types of attacks that fall into the broader class of such attacks.

- Power Analysis Attacks: This type of attack is executed by measuring the power consumption of a cryptographic hardware, which is stated in Kocher *et al.* [16].
- Timing Attacks: When the adversary uses the running time of a protocol as extra information, it may achieve partial knowledge of the implementation to derive information about the secret key.
- Fault Injection Attacks: This type of attacks are carried out in the ways that the adversary enforces the device to run an erroneous operation, which might leak information for the secret key.
- Memory Attacks: Proposed by Halderman *et al.* [15], Memory Attacks stem from the property that DRAM will hold the states for some period without the power of refresh. Hence, the adversary can read the content of parts of the cells and retrieve some information about the secret key as the examples presented in Halderman *et al.* [15].

Hardware-Level Countermeasures. In terms of countermeasures, one may try to modify the implementation or secure hardware, which is easy to be restrictedly secure against a certain type of attack. In other words, the hardware-level solutions target a specific side-channel attack, which should be revised once a more efficient side-channel attack is captured.

Software-Level Solutions. On the line of software-level solutions, researchers try to capture many possible side-channel adversaries and extract the details of the hardware and implementation, which is commonly represented as a leakage function f. By the definition of a leakage function family $\mathcal{F}$, the theoretical constructions work under various models and security targets.

Since the detailed models are out of the scope of this paper, we only introduce the main existing models in the followings: computation-leak-information by Micali and Reyzin [15], bounded-memory model considered by [3], continual memory leakage ($\mathcal{CML}$) [18], auxiliary-input model [24].

Difficulties for Practical System. Hardware-level measures are impractical to be integrated into a platform since the difficulties stem from numberous attack types and devices, while software-level solutions seem better to fit the requirements to build a general platform that can be applied in real-world applications. Even so, it is non-trivial to extend an existing cryptographic system to a leakage-resilient one since the difficulties inherent from the complexities of the attack types and various models. Besides, only the schemes that are built under standard models and simple assumptions can be candidate building blocks for such a practical system. To the best of our knowledge, no existing leakage-resilient cryptographic systems are found until now. In this work, we aim to integrate the software-level results to build a practical system that can be easily applied in many cryptographic platforms, which can defend the potential attacks caused by side-channel information.

1.1 Our Results and Techniques

Overview of Results. In this work, we explore the solutions to build a general leakage-resilient cryptographic system that can be applied in various real-world applications. It is easy to find that the main types of cryptographic applications consist of public-key encryption, signature and secret-key encryption. Specifically, our system begins with leakage-resilient public-key encryption, which is implemented with an identity-based encryption. In addition, we extend it to a leakage-resilient signature construction. Furthermore, by applying leakage-resilient secret sharing layer, we design a general method to extend the existing secret-key encryption primitives to a leakage-resilient secret-key encryption. In this paper, we call our system $\mathcal{LRCRYPT}$, whose details are postponed to Sect. 3. In the followings we outline several key observations for the detailed designs.

Leakage-resilient Public-key Encryption ($\mathcal{LRPKE}$): To build a practical leakage-resilient public-key encryption based on the software-level solutions, we hope that the construction is secure based on simple assumption and standard model, along with efficiency compared to the original systems. With regards to these goals, we find that an identity-based hash proof system ($\mathcal{IBHPS}$) in Alwen *et al.* [4] can be used to construct public-key and identity-based encryption schemes in Bounded Retrieval Model (BRM). In addition, the techniques in Chow *et al.* [11] showed how to build a leakage-resilient identity-based encryption ($\mathcal{LRIBE}$) from $\mathcal{IBHPS}$. The advantages of these techniques include but not limited to simper security definitions and no need to deal with the leakage details. On a very high level, one can construct a desired $\mathcal{LRIBE}$ by combining an $\mathcal{IBHPS}$ with a randomness extractor (see [14]). Thus we will construct a $\mathcal{LRIBE}$ via $\mathcal{IBHPS}$ as the $\mathcal{LRPKE}$ building blocks since it can be easily applied to construct a general public-key encryption, and can also be extended to a secure signature scheme.

Leakage-resilient Signature ($\mathcal{LRSIG}$) **from** $\mathcal{LRIBE}$: Provided that IND-CCA secure $\mathcal{IBE}$ implies secure public-key signature [8], we also expand $\mathcal{LRCRYPT}$ to cover the domain of leakage-resilient signature ($\mathcal{LRSIG}$) almost freely.

Leakage-resilient Secret Key Encryption ($\mathcal{LRSKE}$): Compared to traditional cryptography that assumes perfect secrecy of the secret key, leakage-resilient schemes build on the condition that leakage on a fraction of secret key exists. We will apply the secret sharing method on the secret key which will be updated accordingly [13] to avoid a significant fraction of the secret key to be derived by the adversary. Through this abstraction, we can extend it to various secret key encryption constructions. We design a software-level layer that can protect the secret key from being derived from the leaked informations, and be updated efficiently, through which we can achieve a secure $\mathcal{LRSKE}$.

1.2 Our Contributions

In the followings, we outline the contributions of this paper.

- $\mathcal{LRCRYPT}$ is among the first to explore a practical leakage-resilient platform.
- We build a library with programming interfaces for cryptographic applications consisting of $\mathcal{LRPKE}$, $\mathcal{LRSIG}$, $\mathcal{LRSKE}$.
- We build an extended leakage-resilient layer $\mathcal{CLRS}$, which derives from leakage-resilient secret sharing schemes [13], to be applied in other cryptographic primitives.
- As independent interest, we develop a pairing based matrix calculation building block PIM, which can be applied to matrix operations on the group elements and big integers.

2 Preliminaries

Identity-Based Encryption ($\mathcal{IBE}$). $\mathcal{IBE}$ [21] is public-key encryptions based on users' identities, which can solve difficulties of the public-key deployment [5][9]. $\mathcal{IBE}$ scheme consists of four PPT algorithms (Setup, KeyGen, Enc, Dec).

1. $(MPK, MSK) \leftarrow \mathsf{Setup}(1^\lambda)$: The Setup algorithm takes the security parameter λ as input, and output the (MPK, MSK) key pair. MPK is included in public parameter, while MSK is held by the authority as a secret message.
2. $SK_{ID} \leftarrow \mathsf{KeyGen}(ID, MSK)$: The KeyGen algorithm is run by the key generator authority to generate the secret key SK_{ID} according to the user's id.
3. $\mathsf{C} \leftarrow \mathsf{Enc}(ID, M)$: The Enc algorithm outputs the ciphertext C with the input of user ID and plaintext M.
4. $M \leftarrow \mathsf{Dec}(CSK_{ID})$: Decrypt algorithm with ciphertext C and secret key SK_{ID} outputs plaintext M.

Identity-Based Hash Proof System ($\mathcal{IBHPS}$). In Chow *et al.* [11], they proposed a practical $\mathcal{LRIBE}$ system built from the $\mathcal{IBHPS}$ with a random extractor and designed invalid ciphertext in the encapsulation phase of $\mathcal{IBHPS}$ system. Particularly, they applied this technique to $\mathcal{BBIBE}$ [7], $\mathcal{WIBE}$ [22], $\mathcal{LWIBE}$ [19] respectively. The most encouraging results are that the overhead for $\mathcal{LRIBE}$ is linear of their counterpart in terms of the key calculations such as exponentiation on group elements and pairing operations. $\mathcal{IBHPS}$ consist of five PPT algorithms:

1. $(MPK, MSK) \leftarrow \mathsf{Setup}(1^\lambda)$: Setup generates the master public-key and master secret key of the system with the input of security parameter λ.
2. $SK_{ID} \leftarrow \mathsf{KeyGen}(ID, MSK)$: KeyGen outputs a secret key corresponding to the user ID.
3. $(C, k) \leftarrow \mathsf{EnCap}(ID)$: The valid encapsulation algorithm creates a valid ciphertext C paired with an encapsulation key.
4. $C \leftarrow \mathsf{EnCap}^*(ID)$: The invalid encapsulation algorithm outputs an invalid ciphertext to the given id.

5. $\mathsf{k} \leftarrow \mathsf{DecCap}(\mathsf{C}, \mathsf{SK_{ID}})$: The deterministic decapsulation algorithm recovers the encapsulation key with the input of ciphertext C and secret key SK_{ID}.

Continual-Leakage-Resilient Sharing ($\mathcal{CLRS}$). Following is the description of $\mathcal{CLRS}$.

1. $(\mathsf{sh}_1, \mathsf{sh}_2) \leftarrow \mathsf{ShareGen}(1^\lambda, M)$: The generation algorithm outputs two shares sh_1 and sh_2 with inputs security parameter and message M.
2. $\mathsf{sh}'_\mathsf{b} \leftarrow \mathsf{Update_b}(\mathsf{sh_b})$: The randomized update algorithm updates the current version of share sh_b to sh'_b.
3. $M \leftarrow \mathsf{Reconstruct}(sh_1, sh_2)$: The reconstruction algorithm will output the secret message M with inputs of secret shares.

3 System Design

$\mathcal{LRCRYPT}$ is built on the existing arithmetic library GMP [2], pairing-based cryptographic library PBC [20], and Openssl [1]. Based on GMP [2] and PBC [20], we build the $\mathcal{IBHPS}$ layer, which is the basic structures for $\mathcal{LRIBE}$ and hence $\mathcal{LRSIG}$. In addition, $\mathcal{CLRS}$ [13] works as the leakage-resilient secret sharing layer to blind the secret key and efficiently update the secret key of the existing cryptographic primitives. As independent interest, $\mathcal{PIM}$ is designed for the system that utilises matrix based on group elements or the big integer in GMP [2].

Figure 1 depicts the system architecture of $\mathcal{LRCRYPT}$ and shows the interactions between the layers. Based on the basic libraries is our $\mathsf{Develop}$ layer, which is the key components of $\mathcal{LRCRYPT}$. Precisely, $\mathsf{Develop}$ consists of $\mathcal{LRPKE}$ and $\mathcal{LRSIG}$ both of which are built on $\mathcal{IBHPS}$, $\mathcal{PIM}$, and $\mathcal{LRP}$. Assumed that P is the cryptographic primitives. The topmost one is the $\mathsf{Application}$ layer, which provides the programming interfaces of $\mathcal{LRCRYPT}$ for the users.

In $\mathcal{LRIBE}$, we implement the system based on $\mathcal{IBHPS}$, consisting of leakage-resilient $\mathcal{BBIBE}$ [7], leakage-resilient $\mathcal{WIBE}$ [22], and leakage-resilient $\mathcal{LWIBE}$ [19]. After initialisation, users will obtain the corresponding secret keys and call the encryption or decryption algorithms as the normal identity-based encryption system.

To begin with $\mathcal{LRP}$, users first initialise parameters of the $\mathcal{CLRS}$ layer. Following that, a cryptographic encryption primitive, *i.e.* the secret key encryption $\mathcal{DES}$, has to be initialised as well. Then $\mathcal{LRP}$ is called to build $\mathcal{LRDES}$ by adopting the $\mathcal{CLRS}$ as the secret key encapsulation layer that can blind the protected message and update it accordingly.

3.1 Leakage Resilient Public-Key Encryption (LRPKE)

$\mathcal{IBHPS}$ is regarded as a special $\mathcal{IBE}$ in the circumstance that there are many valid secret key SK_{ID} for given identity ID, and also valid and invalid ciphertexts. Even given SK_{ID}, a random valid ciphertext C is indistinguishable from

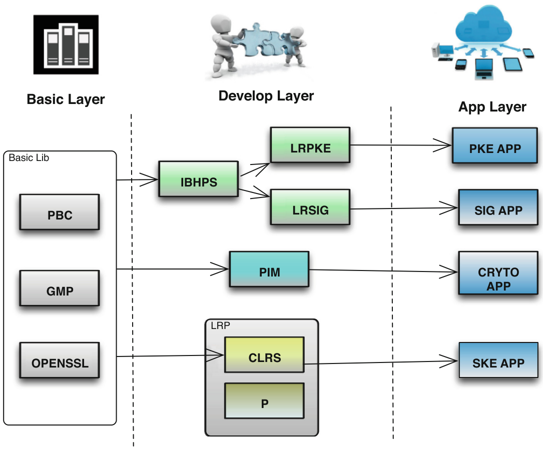

Fig. 1. System architecture

a random invalid one C'. Moreover, a valid ciphertext C decrypts in coincidence with SK_{ID} while an invalid ciphertext C' decrypts to a random value R'. To decrypt a valid ciphertext C, it will output a value R that is indistinguishable from a value that has $|R| - l$ bits of entropy. In this case, $\mathcal{LRIBE}$ via $\mathcal{IBHPS}$ with a $\mathsf{randomnessextractor}$. Luckily, the $\mathsf{extractor}$ can be implemented by applying the existing random functions designed in the lower libraries PBC and GMP.

3.2 Leakage-Resilient Signature (LRSIG)

IND-CCA $\mathcal{IBE}$ implies a public-key signature scheme that is existentially unforgeable against a chosen message attack [8]. Therefore, we build a leakage-resilient public-key signature scheme from $\mathcal{LRIBE}$. Intuitively, the master secret key for $\mathcal{IBE}$ scheme $\mathcal{I} = (\mathsf{Setup_I}, \mathsf{KeyGen_I}, \mathsf{Enc_I}, \mathsf{Dec_I})$ is set as secret key of the signature scheme $\mathcal{S} = (\mathsf{Setup_S}, \mathsf{Sign_S}, \mathsf{Ver_S})$, and the public-key for $\mathcal{S}$ is included in the public parameters for $\mathcal{I}$. Supposed that identity $\mathsf{ID} = \mathsf{M}$ (M is set as input message), the decryption key for ID works as the signature σ on a message M. In order to verify a signature σ, the verifier simply chooses a random message $\bar{M}$, then encrypt $\bar{M}$ with the public-key $ID = M$. Then in the Verify process, it tries to decrypt it using the signature on $\bar{M}$ as the decryption key. If decryption succeeds, Verify returns pass, else reject. Hence given $\mathcal{I}$ and $\mathcal{S}$, we can build the leakage-resilient signature scheme LRSIG $\pi = (\mathsf{Setup}, \mathsf{Sign}, \mathsf{Verify})$.

3.3 Leakage-Resilient Primitive ($\mathcal{LRP}$)

Without loss of generality, we assume that the encryption primitive $\mathcal{P} = (\mathsf{Setup}_P,$ KeyGen_P, Enc_P, $\mathsf{Dec}_P)$. The difference of leakage-resilient cryptography is built on the assumption that some side-channel information will be leaked through implementation and devices, thus we figure out some efficient approaches to limit the probability that the secret key of the encryption scheme to be derived from such dangerous information. Given $\mathcal{CLRS}$ scheme $\mathcal{C} = (\mathsf{Setup}_C, \mathsf{ShareGen}_C, \mathsf{Update}_C, \mathsf{Reconstruct}_C)$ and $\mathcal{P}$, we describe the leakage-resilient encryption scheme $\mathcal{E} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ in the followings.

1. $PP \leftarrow \mathsf{Setup}(1^\lambda)$: This algorithm takes security parameter λ as input, and calls two Setup algorithms in both $\mathcal{E}$ and $\mathcal{P}$, then includes the public parameters in both algorithms to the output PP.
2. $sh \leftarrow$ **keygen**(PP): This algorithm calls $\mathcal{P}$'s KeyGen to get usk, then it takes this usk as secret share and runs $\mathsf{ShareGen}(1^\lambda, \Phi(usk))$, where Φ is a map from the group usk to the group M and it's not difficult to find its inverse Φ^{-1} in polynomial time. Then output the two shares $sh = (sh_1, sh_2)$ as user's secret key.
3. $C \leftarrow \mathsf{Enc}(P, sh)$: Enc takes plaintext P and secret key sh, then calls Enc_P in the same way, and outputs ciphertext C.
4. $(M, \perp) \leftarrow \mathsf{Dec}(sh_1, sh_2, C)$: This algorithm firstly calls $\mathsf{Reconstruct}(sh_1, sh_2)$ of $\mathcal{CLRS}$ and gets $\Phi(usk)$, then computes usk and runs Dec and outputs the plaintext M or $\perp$ if decryption fails.

3.4 Pairing and Big Integer Based Matrix ($\mathcal{PIM}$)

As independent interest, we abstract the matrix operations over group elements and develop a component called Pairing/Integer Matrix ($\mathcal{PIM}$), which is designed for the matrix and list operations for abundant matrix operations. Precisely, $\mathcal{PIM}$ is built directly on a library of GMP [2] and PBC [20]. $\mathcal{PIM}$ is not only available as an integrated component in $\mathcal{LRCRYPT}$ but also aims to provide other systems with common matrix calculations based on cryptographic group elements and big integers. To our knowledge, we are among the first to demonstrate such a calculation library. $\mathcal{IML}$ [10], which is based on GMP [2] library, cannot be applied to the group elements.

3.5 Programming Interfaces

In $\mathcal{LRCRYPT}$, we provide abundant programming interfaces that allow users to build it in a leakage-resilient cryptographic application. In Table 1, we list the parts the programming interfaces of our system, and complete interface description will be found in our *Github* project. In $\mathcal{LRIBE}$ and $\mathcal{LRSIG}$, we can set the parameter $TYPE$ in range (1, 3) as in the subtypes of $\mathcal{IBE}$ types settings as $\mathcal{BBIBE}$, $\mathcal{LWIBE}$, and $\mathcal{WIBE}$.

3.6 Security Parameter and Leakage Parameter

Leakage parameter l is defined as the amount of bits leaked on secret key over the total size of the secret key, in particular, $l = |leakage\ info|/|secret\ key|$. In terms of choice for the curve in $\mathcal{LRCRYPT}$, we initialise the group by

Table 1. Programming interfaces

$\mathcal{LRIBE}$	$(PP, MSK) \leftarrow \mathsf{Setup}(1^\lambda, TYPE)$ $TYPE$ ranges from 1–3
	$SK \leftarrow \mathsf{KeyGen}(PP, MSK, ID)$
	$C \leftarrow \mathsf{Enc}(PP, MSK, SK, M)$
	$M \leftarrow \mathsf{Dec}(PP, MSK, SK, C)$
$\mathcal{LRSIG}$	$(PP, MSK) \leftarrow \mathsf{Setup}(1^\lambda, SK, \mathsf{TYPE})$ $TYPE$ ranges from 1–3, $MSK = SK$
	$\sigma \leftarrow \mathsf{Sign}(PP, MSK, M)$
	$(\mathsf{accept}, \mathsf{reject}) \leftarrow \mathsf{Verify}(\mathsf{PP}, \mathsf{msk}, \sigma)$
$\mathcal{LRP}$	$(PP) \leftarrow \mathsf{Setup}(1^\lambda, SK)$
	$sh \leftarrow \mathsf{KeyGen}(PP)$
	$C \leftarrow \mathsf{Enc}(P, sh)$
	$P \leftarrow \mathsf{Dec}(C, sh)$
$\mathcal{PIM}$	$\mathsf{mtr} \leftarrow \mathsf{Init}(\mathsf{SIZE\ R}, \mathsf{SIZE\ C}, \mathsf{INFO\ E})$ initialise matrix mtr with size $\mathsf{R} * \mathsf{C}$
	Assigning functions
	Arithmetic calculation $\mathsf{Add/Sub/Mul/Pow}(\mathsf{mtr}_{\mathsf{out}}, \mathsf{mtr}_1, \mathsf{mtr}_2)$
	Matrix property calculation $\mathsf{Kernel}, \mathsf{Xor}$

A-$TYPE$, which is 512-bit length of the group order. Without loss of generality, we set leakage parameter as $l = c \times \lambda$, where c is denoted as the leakage ratio and $c \in (0, 1)$. In the subtype of $\mathcal{LRIBE}$-$\mathcal{BB}$ and $\mathcal{LRIBE}$-$\mathcal{W}$ [12], the leakage rate is $\frac{1}{3}$, while it is $\frac{1}{9}$ for the $\mathcal{LRIBE}$-$\mathcal{LW}$ type [11]. Assumed that the size of key space is 2^μ, then the system will operate on the message space $M = (0, 1)^\nu$, where ν satisfies $v \leq \mu - 2log(1/\epsilon) - 1$. $\mathcal{LRIBE}$-$\mathcal{LW}$ structure is built from the composite order group. We denote length of the composite order $N = 1024$ bits, where $N = p_1 \times p_2 \times p_3 (p_1,\ p_2,\ p_3$ are all primes).

4 Performance Analysis

In terms of performance evaluation, we mainly focus on the followings:

Running Time: As general, we will record running time of the key algorithms of the system, which is the prominent part of the performance evaluations.

Blowup: In order to achieve leakage-resilience, we need to operate extra operations than their counterparts. Intuitively, the blowup is defined as the ratio of the time cost in the algorithms of leakage-resilient settings over the non-leakage-resilience baseline. Our goal is to build the system that can achieve comparative blowup and leakage-resilience.

4.1 System Analysis

Efficient Pairing Calculation Overhead. Since pairing calculation $\mathsf{e}(\mathsf{g}_1, \mathsf{g}_2)$ that maps from group $\mathbb{G}_1$ $\mathbb{G}_2$ to $\mathbb{G}_T$ is time-consuming, we will store the pairing result in private memory after the setup of the system in order to save running time when the number of pairings is relatively large by reading an element instead of calculating the pairing.

Efficient for Large Scale. Although $\mathcal{LRP}$ is not necessarily more efficient than some existing leakage resilient secret key constructions [6], $\mathcal{LRP}$ will outperform its counterpart in terms of large scale. Since we only have to initialise the $\mathcal{CLRS}$ layer once and apply it in the secret key encryptions for multiple times. In this domain, the blowup for leakage-resilient will be significantly lower in average. Therefore, our system achieves significant improvements for large-scale calculations, which is common nowadays.

4.2 Security Analysis

In this section, we demonstrate the security analysis of the main building blocks for $\mathcal{LRCRYPT}$.

Security of Leakage-Resilient Signature: Through the constructions in Sect. 3, we can easily conclude that if there exists an adversary $\mathcal{A}$ that can break the signature scheme π, we can construct a simulator $\mathcal{B}$ that breaks the security of $\mathcal{LRIBE}$ system with non-trivial probability in the defined security model.

$\mathcal{LRSKE}$ *from* $\mathcal{CLRS}$: Denoted that $\mathcal{P}$ is a cryptographic primitive and *usk* is the user's secret key in $\mathcal{P}$. At the first step, we recap the $\mathcal{CLRS}$ $\mathcal{C} = (\mathsf{ShareGen}, \mathsf{Update}, \mathsf{Reconstruct})$.

Now we discuss the definition of the security for $\mathcal{LRP}$. Particularly, the security model is a split-state model [23], which means the adversary can only learn the secret information in a split way. Under this model, we design the system of generic leakage resilient cryptographic scheme with $\mathcal{CLRS}$. More precisely, a user in LRP has secret key $\mathsf{sk} = \mathsf{sh} = (\mathsf{sh}_1, \mathsf{sh}_2)$, and the adversary can only get the leakage information of $f(\mathsf{sh}_1)$ and $f(\mathsf{sh}_2)$, where $f \in \mathcal{F}$ is the leakage function the adversary can access. However, the adversary cannot get any information of $f(\mathsf{sh}_1, \mathsf{sh}_2)$ simultaneously. From the results of [23], we can infer that our $\mathcal{LRSKE}$ scheme built in this way is secure in the split-state model. Then we apply this leakage-resilient layer in the secret key primitive. e.g. $\mathcal{DES}$ in our system.

4.3 Implementation Results

We do the implementations in the windows system of Intel(R) Xeon(R) E7-2830 CPU@3.40 GHz configured with GMP and PBC libraries. We build our system with $C/C++$ program, and output the library $\mathcal{LRCRYPT}$ for development in some cryptographic systems which are sensitive to side-channel attacks. In this section, we will present the implementation results for performance evaluation. In addition, we also maintain our project in *Github*, which will be public in the coming future.

$\mathcal{LRIBE}$ Results. With varied parameters, we collect the running time for various subtypes of $\mathcal{LRIBE}$ (Fig. 2), consisting of a four algorithms Setup, KeyGen, Enc, Dec. $\mathcal{LRIBE}$-$\mathcal{BB}$ is the simplest version with selectively secure [11] which only consists of constant number of pairing calculations. While $\mathcal{LRIBE}$-W is the compromising one since its complexity of the group multiplication is linear in the size of the identity input *id*. However, the composite version $\mathcal{LRIBE}$-$\mathcal{LW}$ subtype is most time-consuming. Composite order group operations suffer from the efficiency constraint compared to its counterparts. However, it can not be always transferred to the prime order groups, since the security of the scheme inherits directly from the composite order group.

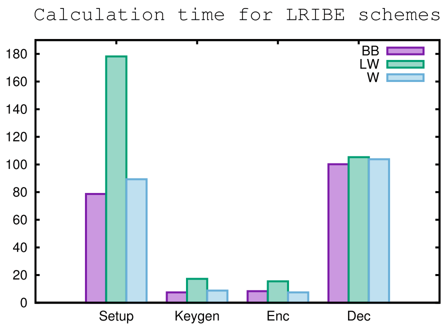

Fig. 2. LRIBE results

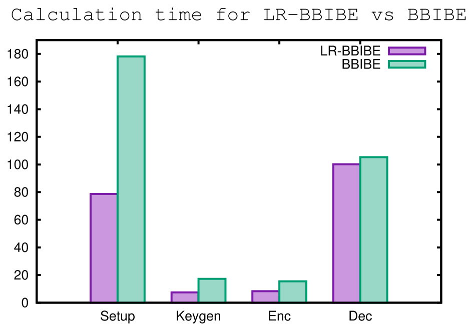

Fig. 3. LR-BBIBE vs BBIBE

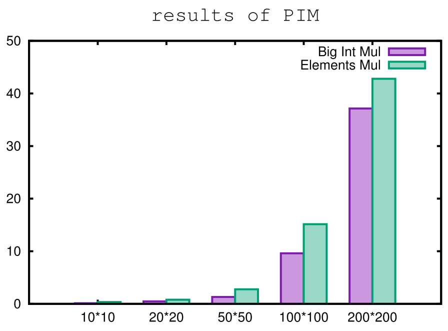

Fig. 4. PIM results

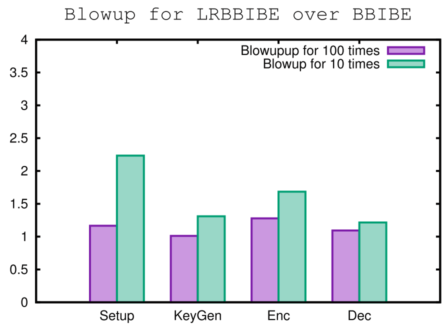

Fig. 5. Blowup for LRIBE

LRIBE and IBE Results. In terms of practical requirements, we compare our $\mathcal{LRIBE}$ subtype with the original identity-based encryption scheme in Fig. 3. Specifically, we implement one example by both calling the programming interface of $\mathcal{LRCRYPT}$ and original $\mathcal{BBIBE}$ [7] scheme. As shown in Fig. 5, the curves depict the cost for $\mathcal{LRIBE}$ and $\mathcal{IBE}$ share the similar trend, though running time for $\mathcal{LR}$-$\mathcal{BBIBE}$ is about 1.3 times of $\mathcal{BBIBE}$[7]. As the increase of running number, the average blowup for Setup decreases since the extra cost for $\mathcal{CLRS}$ will be divided to numerous process, resulting in lowering blowup for each round.

PIM Results. To evaluate the performance of PIM layer, we choose the operations of matrix addition and multiplication with varying dimensions $20 * 20, 50 * 50, 100 * 100$, and execute the calculations for 50, 100, 1000 times to lower the error bar. As discussed in previous sections, we set the bit size of operation element as 128 bits and present the results in Fig. 4.

In the experiments, we compare $\mathcal{PIM}$ results with the existing integer matrix library called IML with large elements, and notice that running time of our system is comparable with results of IML. Moreover, we also compare the results for matrix with big integers and group elements. Without loss of generality, we denoted Mz and Me as a matrix for bit integer and a matrix for group elements respectively.

Acknowledgments. This work was supported in part by NSFC/RGC Joint Research Scheme (N_HKU 72913) of Hong Kong, Seed Funding Programme for Basic Research of HKU (201511159034, 201411159142), and National High Technology Research and Development Program of China (2015AA016008).

References

1. https://www.openssl.org/
2. https://gmplib.org/
3. Akavia, A., Goldwasser, S., Vaikuntanathan, V.: Simultaneous hardcore bits and cryptography against memory attacks. In: Reingold, O. (ed.) TCC 2009. LNCS, vol. 5444, pp. 474–495. Springer, Heidelberg (2009). doi:10.1007/978-3-642-00457-5_28
4. Alwen, J., Dodis, Y., Wichs, D.: Leakage-resilient public-key cryptography in the bounded-retrieval model. In: Halevi, S. (ed.) CRYPTO 2009. LNCS, vol. 5677, pp. 36–54. Springer, Heidelberg (2009). doi:10.1007/978-3-642-03356-8_3
5. Barker, E., Barker, W., Burr, W., Polk, W., Smid, M.: Recommendation for Key Management-Part 1: General (revised). NIST special publication, Citeseer (2006)
6. Belaıd, S., Grosso, V., Xavier-Standaert, F.: Masking and leakage-resilient primitives: one, the other (s) or both? (2014)
7. Boneh, D., Boyen, X.: Efficient selective-ID secure identity-based encryption without random oracles. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 223–238. Springer, Heidelberg (2004). doi:10.1007/978-3-540-24676-3_14
8. Boneh, D., Franklin, M.: Identity-based encryption from the weil pairing. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 213–229. Springer, Heidelberg (2001). doi:10.1007/3-540-44647-8_13
9. Canetti, R., Halevi, S., Katz, J.: A forward-secure public-key encryption scheme. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 255–271. Springer, Heidelberg (2003). doi:10.1007/3-540-39200-9_16
10. Chen, Z.: https://cs.uwaterloo.ca/~astorjoh/iml.html

11. Chow, S.S., Dodis, Y., Rouselakis, Y., Waters, B.: Practical leakage-resilient identity-based encryption from simple assumptions. In: Proceedings of the 17th ACM Conference on Computer and Communications Security, pp. 152–161. ACM (2010)
12. Conti, M., Di Pietro, R., Mancini, L.V., Mei, A.: (old) Distributed data source verification in wireless sensor networks. Inf. Fusion **10**(4), 342–353 (2009)
13. Dodis, Y., Lewko, A., Waters, B., Wichs, D.: Storing secrets on continually leaky devices. In: IEEE 52nd Annual Symposium on Foundations of Computer Science (FOCS), pp. 688–697. IEEE (2011)
14. Dodis, Y., Reyzin, L., Smith, A.: Fuzzy extractors: how to generate strong keys from biometrics and other noisy data. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 523–540. Springer, Heidelberg (2004). doi:10.1007/978-3-540-24676-3_31
15. Halderman, J.A., Schoen, S.D., Heninger, N., Clarkson, W., Paul, W., Calandrino, J.A., Feldman, A.J., Appelbaum, J., Felten, E.W.: Lest we remember: cold-boot attacks on encryption keys. Commun. ACM **52**(5), 91–98 (2009)
16. Kocher, P., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 388–397. Springer, Heidelberg (1999). doi:10.1007/3-540-48405-1_25
17. Kocher, P.C.: Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 104–113. Springer, Heidelberg (1996). doi:10.1007/3-540-68697-5_9
18. Lewko, A., Rouselakis, Y., Waters, B.: Achieving leakage resilience through dual system encryption. In: Ishai, Y. (ed.) TCC 2011. LNCS, vol. 6597, pp. 70–88. Springer, Heidelberg (2011). doi:10.1007/978-3-642-19571-6_6
19. Lewko, A., Waters, B.: New techniques for dual system encryption and fully secure HIBE with short ciphertexts. In: Micciancio, D. (ed.) TCC 2010. LNCS, vol. 5978, pp. 455–479. Springer, Heidelberg (2010). doi:10.1007/978-3-642-11799-2_27
20. Lynn, B.: PBC: the pairing-based cryptography library. http://crypto.stanford.edu/pbc/
21. Shamir, A.: Identity-based cryptosystems and signature schemes. In: Blakley, G.R., Chaum, D. (eds.) CRYPTO 1984. LNCS, vol. 196, pp. 47–53. Springer, Heidelberg (1985). doi:10.1007/3-540-39568-7_5
22. Waters, B.: Efficient identity-based encryption without random oracles. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 114–127. Springer, Heidelberg (2005). doi:10.1007/11426639_7
23. Xiong, H., Zhang, C., Yuen, T.H., Zhang, E.P., Yiu, S.M., Qing, S.: Continual leakage-resilient dynamic secret sharing in the split-state model. In: Chim, T.W., Yuen, T.H. (eds.) ICICS 2012. LNCS, vol. 7618, pp. 119–130. Springer, Heidelberg (2012). doi:10.1007/978-3-642-34129-8_11
24. Yuen, T.H., Chow, S.S.M., Zhang, Y., Yiu, S.M.: Identity-based encryption resilient to continual auxiliary leakage. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 117–134. Springer, Heidelberg (2012). doi:10.1007/978-3-642-29011-4_9